Efficient Single-Round Obfuscation of Search and Result Patterns in Searchable Encryption

Tung Le
Virginia Tech
Blacksburg, VA, USA
tungle@vt.edu

Thang Hoang
Virginia Tech
Blacksburg, VA, USA
thanghoang@vt.edu

Abstract

Searchable Symmetric Encryption (SSE) enables data owners to securely store encrypted data on untrusted cloud servers while retaining the ability to perform secure searches and retrieve relevant documents. However, standard SSE schemes expose search patterns (whether two queries are identical), and result patterns (which documents are returned), making them susceptible to leakage-abuse attacks that can infer sensitive information such as the queried keywords and/or document contents. While Oblivious RAM (ORAM) and Private Information Retrieval (PIR) can hide these patterns, their high computation and communication overhead often render them impractical for real-world search workloads. A more efficient alternative is to obfuscate search and result patterns using Differential Privacy (DP). Unfortunately, existing DP-based SSE schemes either provide insufficient query privacy protection, or still incur substantial performance overhead.

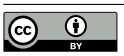
In this paper, we propose FROST, a novel differentially private SSE scheme that efficiently obfuscates both search and result patterns, while providing strong resilience against all known statistical leakage-abuse attacks. The core component of FROST is our new rerandomized PIR (RePIR) scheme designed for private databases, which allows server-side rerandomization of encrypted PIR query responses. In FROST, we also introduce a novel method for applying DP noises to SSE for search result obfuscation using only simple arithmetic operations. An important property of FROST is that it requires only a single round of communication, with small user-side storage as an additional benefit. We fully implemented FROST and conducted extensive experiments over real-world datasets to rigorously assess its practical performance and resilience. Our experiments showed that FROST not only effectively mitigates pattern-leakage attacks while maintaining reasonable utility, but also achieves up to *four orders of magnitude faster* keyword search and *three orders of magnitude lower bandwidth* overhead than prior DP-based SSE schemes.

CCS Concepts

• **Security and privacy → Database and storage security**; **Privacy-preserving protocols**; **Cryptography**.

Keywords

Searchable Encryption, Access Pattern, Differential Privacy



CCS '26, The Hague, Netherlands

ACM ISBN 979-8-4007-2871-6/2026/11
https://doi.org/10.1145/3830454.3832727

ACM Reference Format:
Tung Le and Thang Hoang. 2026. Efficient Single-Round Obfuscation of Search and Result Patterns in Searchable Encryption. In *Proceedings of the 2026 ACM SIGSAC Conference on Computer and Communications Security (CCS '26), November 15–19, 2026, The Hague, Netherlands.* ACM, New York, NY, USA, 20 pages. https://doi.org/10.1145/3830454.3832727

1 Introduction

Searchable Symmetric Encryption (SSE) [83] enables a user to conduct secure searches over encrypted data stored remotely on a server. Typically, the user encrypts both the database and an associated search index on their local machine before outsourcing them to the server. When a search is requested, the user generates a query that allows the server to perform a secure lookup on the encrypted index and return the relevant encrypted documents, which the user can then decrypt on their end.

Although both queries and documents are encrypted, the server can still infer sensitive information during the user's interaction. In particular, it can observe which documents are retrieved (revealing the result patterns) and identify repeated queries (revealing the search patterns). Many existing SSE schemes [13, 17, 26, 53, 54, 65, 83] permit this type of leakage to maintain efficiency for keyword search. However, a lot of research [15, 44, 51, 74, 76, 81, 100] has shown that if the server has some background or partial similar knowledge about a portion of the stored data or queries, it can infer the actual queried keywords and/or document content with high accuracy, posing a severe threat to user and data privacy.

Many improvements have been developed to strengthen the privacy guarantees of SSE systems, but these often come with trade-offs in performance. These trade-offs typically include increased communication costs, higher computational demands, and additional storage burdens on the user side. Solutions using Oblivious RAM (ORAM) [42, 84, 90] or Private Information Retrieval (PIR) [23, 35, 41, 46, 62] can completely conceal which documents are accessed in a database. However, ORAM introduces significant communication overhead, while PIR incurs substantial computation cost, and both are generally not tailored for secure keyword search in encrypted databases (an exception for TWORAM [38]). Another line of research uses distributed/multiparty computation (MPC) with multiple servers [28, 34, 63, 64] to hide search and result patterns, relying on the assumption that at least one server is honest (i.e., the servers do not all collude). To actually retrieve query-matching documents, these protocols often necessitate multiple communication rounds, which is an important criterion of searchable encryption [26] and should be kept minimal. Connectivity is relatively scarce in isolated areas and mobile IoT networks,

Table 1: Comparison of FROST with prior encrypted search systems.

Scheme	#Servers L	Search Leakage $\mathcal{L}_{\text{Search}}$	Forward Privacy	#Rounds	Search Complexity		
					Server	User	Communication
SSE [16]	1	$\{\text{sp}(\text{kw}), \text{rp}(\text{kw}), \text{sv}(\text{kw})\}$	✗	1	$O(a_w)$	$O(\lambda)$	$O(\lambda + n_w)$
$\Sigma o\phi o\varsigma$ [13]	1	$\{\text{sp}(\text{kw}), \text{rp}(\text{kw}), \text{sv}(\text{kw})\}$	✓	1	$O(a_w)$	$O(\lambda)$	$O(\lambda + n_w)$
CLRZ [20]	1	$\{\text{sp}(\text{kw}), \text{sv}(\text{kw})\}$	✗	1	$O(a_w)$	$O(\lambda)$	$O(\lambda + n_w)$
OSSE [81]	1	$\{\varnothing\}$	✗	1	$O(d_w N^{\dagger})$	$O(d_w N)$	$O(d_w N)$
DORY [28]	2	$\{\varnothing\}$	✓	2*	$O(mN)$	$O(\log m + N)$	$O(\lambda \log m + N)$
Branch [94]	1	$\{\text{sp}(\text{kw}), \text{rp}(\text{kw}), \text{sv}(\text{kw})\}$	✓	1	$O(a_w)$	$O(\lambda)$	$O(\lambda + n_w)$
L-SD$_d$[1C][72]	1	$\{\text{sp}(\text{kw}), \text{rp}(\text{kw}), \text{sv}(\text{kw})\}$	✓	1	$O(\log N_{\text{kw},f})$	$O(\lambda)$	$O(\lambda + n_w)$
Our FROST	1	$\{\varnothing\}$	✓	1	$O(mN)$	$O(m + N)$	$O(m + N)$

• kw: the input search keyword; sp: Search pattern; rp: Result pattern; sv: Search volume (see §2.2 for more details).
• λ: Security parameter; N: Total number of documents; d_w: Number of keywords per document; $N_{\text{kw},f}$: Number of (keyword, file) pairs; m: Size of keyword representation *per* document; a_w: Number of times the queried keyword w is historically added (updated) to the database; n_w: Number of matched results for keyword w.
• In practice, $d_w < m \ll W$, where W is the keyword universe set, and $n_w \leq N$.
We assume document identifiers can be represented using a constant number of bits to skip this quantity in search complexity.
† Public-key pairing operations.
* DORY [28] requires two communication rounds to search and retrieve documents. In DORY, the user needs to decrypt search results computed by the server first before being able to retrieve documents in the next round.

and round-trip latency can be large. Therefore, minimizing user-server interactions (preferably one round) is often a major goal (e.g., [81]).

To address result pattern leakage in SSE with a single round-trip, a differential privacy (DP)-based approach by Chen et al. [20] involves obfuscating the database index prior to outsourcing it. This ensures the server only observes obfuscated result patterns, thereby limiting the potential for attacks by such leakage. Nevertheless, while this method improves security, it does not conceal search patterns, since the result pattern for each keyword is determined and remains unchanged once the database is outsourced. As a result, search pattern leakage still enables several real-world attacks [51, 68, 74–76, 78] to be effective, even when result patterns are obfuscated. The follow-up work OSSE by Shang et al. [81] obfuscates both search and result patterns but incurs substantial computational and bandwidth overhead, which takes about 27 minutes to process a search query over only 30K documents on a 160-core powerful server [81, Section IX].

Regarding the security of OSSE, applying DP at the document-level granularity, as OSSE does, is still an effective approach to mitigate existing attacks. The most recent attack on querying schemes that use DP to protect query and data privacy focuses on "bucketization" and "partitions with encodings" techniques, which produce noise at the bucket or partition level during user interactions when executing search [79]. However, these schemes always return the requested bucket or partition in the result. The attack leverages the fact that buckets or partitions returned as result in the same set of queries would correspond to the same label. Consequently, by observing the relation in results of sufficient queries, an adversary can estimate the bounds on label values of buckets or partitions, then recover the histogram of labels in data. Given that OSSE remains secure against state-of-the-art attacks under proper differential-privacy parameters, our goal is to improve its performance while preserving its security guarantees.

In this work, we propose FROST, which stands for Searchable Encryption with Efficient Single-Round Obfuscation of Search and Result Patterns, a new SSE scheme that protects both search and result patterns with high performance and only requires a *single-round* communication. The main idea behind FROST is that it produces a fresh obfuscation per query by adding false positives and false negatives to the output results through lightweight arithmetic operations. More specifically, the high-level idea of FROST search is to add a random mask onto PIR output, which is known by the user, followed by using dot product operator with transformation vectors to set the output as true positives, false positives, or random non-matches. Our approach is fundamentally different from OSSE, which requires a large number of additional tokens to obfuscate real queries, as well as matches and unmatched results, incurring high computation and bandwidth costs. Instead, we design RePIR, a new scheme for private information retrieval on a private database, which outputs the queried data masked with random values that can be precomputed, facilitating obfuscation for the search results later. Next, we propose a novel approach for applying DP in SSE. Intuitively, existing works that hide pattern leakage [28, 49, 63] structure the search index as a table, where the value '1' at row f and column w indicates that keyword w appears in document f, and '0' otherwise. Executing a keyword search is privately retrieving the column corresponding to the queried keyword and filter out matched documents with value '1'. Several works apply Bloom filter [28, 34, 63, 64] to compress the search index, at the cost of checking more columns in the online phase. We instead represent these states '0' and '1' as integer vectors, which allows us to leverage dot products to transform these state vectors into arbitrary scalar values. For keyword search, the user sends transformation vectors specifically constructed to compute dot products with the state vectors, thereby collapsing them into precomputable scalar values. Therefore, the user can configure the transformation vectors to classify search results to true positives, false positives, or random non-matches. Our approach with RePIR and state/transformation vectors achieves

significantly better performance by only using lightweight operations (i.e., additions/multiplications between 32 and 64-bit integers), as opposed to OSSE, which relies on computation-heavy cryptographic techniques (i.e., public-key pairing). Also, FROST allows to perform queries on the encrypted database and receive the matched documents in the same communication round. (MUSES [63] and TWORAM [38] require at least 3 and 4 rounds, respectively). Moreover, it requires small user-side storage. Our experimental evaluation showed that applying DP is highly effective to mitigate state-of-the-art query recovery attacks for SSE while maintaining efficiency for keyword search.

Briefly, inspired by the prior work OSSE [81], which establishes a strong formal foundation for DP-based SE, our FROST targets the same DP objective while significantly improving performance. In summary, our contributions are the following:

- We propose FROST, an SSE scheme whose main features include: (*i*) obfuscation of both search and result patterns; (*ii*) single-round communication with small user-side storage; and (*iii*) high resilience against known practical query-recovery attacks, which assume that adversaries have access to a substantial amount of similar data as auxiliary background information, as much as half of the target user's dataset (see details in §6).
- We introduce RePIR, a novel primitive for efficient private information retrieval on private databases that supports rerandomization on the PIR encrypted output, setting the stage for subsequent secure computations on the retrieved masked data.
- We introduce a new approach to applying DP in SSE effectively. Our method builds the search index using state vectors, which are rerandomized upon retrieval through RePIR. We then apply transformation vectors to output obfuscated search results.
- We fully implemented FROST and evaluated its performance on commodity servers. Under real environments, experimental results demonstrated that FROST performs search 460×–10012× faster with 1152×–3748× smaller bandwidth overhead than OSSE [81] with the same security guarantees. FROST renders leakage-abuse attacks leveraging search-pattern and query-frequency leakage [68, 74] largely ineffective, reducing attack accuracy from 39.9–81.4% to near-random guessing (≈ 0.1%) (see §6.1.1). Our implementation is available at https://github.com/vt-asaplab/FROST.

Table 1 compares FROST with prior SSE designs. To our knowledge, FROST is the first SSE that can obfuscate search result patterns and achieve efficiency simultaneously without using costly oblivious access primitives or distributed servers. Furthermore, FROST achieves single-round search, which is important in practice for high-latency or intermittently connected settings, as it allows the user to issue a query and later obtain the result without sustained interaction. We stress that achieving single-round search with pattern obfuscation like FROST is technically challenging, since the server must directly output the final search result in a single response, with no room for extra user-side post-processing (e.g., noise addition). By contrast, multi-round protocols can leverage existing techniques such as ORAM or PIR to decouple query execution from result obfuscation, at the cost of additional interaction.

2 Preliminaries

Notation. $||$ denotes the concatenation operator. We denote by λ the security parameter and by $\mathbb{Z}_p$ the ring of integer scalars modulo p. We denote $\hat{\mathbb{Z}}_p$ the ring of integer vectors modulo p. We denote by $[n]$ the set $\{1, \ldots, n\}$. $x \xleftarrow{\$} [n]$ means x is selected uniformly at random from the set $\{1, \ldots, n\}$. Bold small letters ($\mathbf{a}$) denote vectors, while capitalized bold letters ($\mathbf{M}$) denote matrices. We denote by $\langle \mathbf{a}, \mathbf{b} \rangle$ the dot product of $\mathbf{a}$ and $\mathbf{b}$. Given a vector $\mathbf{a}$, we denote $|\mathbf{a}|$ as the cardinality of vector $\mathbf{a}$, and $\mathbf{a}[i]$ as the i-th element of vector $\mathbf{a}$. Given a matrix $\mathbf{M}$, $\mathbf{M}[i, *]$ and $\mathbf{M}[*, j]$ denote accessing the row i and column j of $\mathbf{M}$, respectively. $\mathbf{M}[i, j]$ denotes accessing the cell indexed at row i and column j. We denote $A \overset{c}{\approx} B$ as the computational indistinguishability between two distributions A and B.

Let $W = \{w_1, w_2, ..., w_{|W|}\}$ be the keyword universe, and let $|W|$ denote its size. Let $\mathcal{D}$ be a dataset of N documents, sorted by their ids. $\mathcal{D}[i]$ is the i-th document in the dataset; without loss of generality, we assume $\mathsf{id}(\mathcal{D}[i]) = i$. Let $\mathcal{D}(w)$ be the list of documents that contain w ordered by id.

Let $\mathsf{BF} = (\mathsf{Init}, \mathsf{Gen}, \mathsf{Vrfy})$ be a Bloom filter (BF) [11]: $(H_1, \ldots, H_K) \leftarrow \mathsf{BF.Init}(m, K)$: generating K mappings $H_k : \mathcal{S} \rightarrow [m]$ $\forall k \in [K]$ with two parameters m (BF size) and K; $\mathbf{u} \leftarrow \mathsf{BF.Gen}(\mathcal{S})$: computing the BF representation $\mathbf{u} \in \{0,1\}^m$ of a given set $\mathcal{S}$; $\{0,1\} \leftarrow \mathsf{BF.Vrfy}(\mathbf{u}, s)$: checking if an element s belongs to the set represented by $\mathbf{u}$. We denote by $\hat{d} \leftarrow \lceil d \rfloor_\Delta / \Delta$ rounding d to the nearest multiple of Δ then divided by Δ, with $\Delta \in \mathbb{Z}$, and by $\hat{d} \leftarrow \lfloor d \rfloor_\Delta / \Delta$ rounding down d to the nearest multiple of Δ then divided by Δ.

Bit Operations. We denote $x \lll t$ and $x \ggg t$ as the (binary) left-shift and right-shift operations by t bits of value x.

2.1 Learning With Errors (LWE)

The security of our RePIR scheme (see §4.2) relies on the decision version of the learning-with-errors assumption [80].

Definition 1 (LWE). *Given a set of n basis vectors $\mathbf{a}_i \in \mathbb{Z}_q^m$ represented by matrix $\mathbf{A}$ and a point $\mathbf{t} \in \mathbb{Z}_q^m$. The LWE is the problem of determining a set of secret coefficients $\mathbf{s} = (s_1, s_2, \ldots, s_n)$ with $s_i \in \mathbb{Z}_q$, such that $\mathbf{A} \cdot \mathbf{s} + \mathbf{e} = \mathbf{t}$ mod q, where $\mathbf{e}$ is an unknown error vector consisting of small integers modulo q.*

The assumption is parameterized by the dimension of the LWE secret $n \in \mathbb{N}$, the number of samples $m \in \mathbb{N}$, the integer modulus $q \geq 2$, and an error distribution χ over $\mathbb{Z}$. The LWE assumption then asserts that for a matrix $\mathbf{A} \xleftarrow{\$} \mathbb{Z}_q^{m \times n}$, a secret $\mathbf{s} \xleftarrow{\$} \mathbb{Z}_q^n$, an error vector $\mathbf{e} \xleftarrow{\$} \chi^m$, and a random vector $\mathbf{r} \xleftarrow{\$} \mathbb{Z}_q^m$, the following distributions are computationally indistinguishable:

$$\{(\mathbf{A}, \mathbf{A} \cdot \mathbf{s} + \mathbf{e})\} \overset{c}{\approx} \{(\mathbf{A}, \mathbf{r})\}.$$

More specifically, the (n, q, χ)-LWE problem with m samples is $(T, \tilde{\epsilon})$-hard if all adversaries running in time T have advantage at most $\tilde{\epsilon}$ in distinguishing between the two distributions.

Secret-key Regev Encryption. Regev [80] gives a secret-key encryption scheme that is secure under the LWE assumption. With LWE parameters (n, q, χ) and a plaintext modulo p, the Regev secret

key is a vector $\mathbf{s} \xleftarrow{\$} \mathbb{Z}_q^n$. The Regev encryption of $\mu \in \mathbb{Z}_p$ is:

$$(\mathbf{a}, c) = (\mathbf{a}, \langle \mathbf{a}, \mathbf{s} \rangle + e + \lfloor q/p \rfloor \cdot \mu) \in \mathbb{Z}_q^n \times \mathbb{Z}_q,$$

for $e \xleftarrow{\$} \chi$. To decrypt the ciphertext, anyone who knows the secret $\mathbf{s}$ can compute $(c - \langle \mathbf{a}, \mathbf{s} \rangle) \bmod q$ and round the result to the nearest multiple of $\lfloor q/p \rfloor$. Decryption succeeds as long as the absolute value of the error sampled from the error distribution χ is smaller than $\frac{1}{2} \cdot \lfloor q/p \rfloor$. We say that a setting of the Regev parameters supports *correctness error* δ if the probability of a decryption error is at most δ (over the encryption algorithm's randomness).

2.2 Searchable Encryption

Searchable Encryption permits a user to perform privacy-preserving keyword searches over encrypted documents without leaking the plaintext of the keyword and documents to the storage server. We recall Dynamic Searchable Symmetric Encryption (DSSE) that allows both secure keyword search and document update.

Definition 2 (DSSE). *A* DSSE *scheme is a tuple of PPT algorithms defined as follows:*

- $(\kappa, \mathsf{st}, \mathsf{EDB}) \leftarrow \mathsf{Setup}(1^\lambda)$*: Given a security parameter* λ*, it outputs a secret key* κ *and a state* st *to be stored locally, and an initialized encrypted database* EDB.
- $\mathfrak{s} \leftarrow \mathsf{SrchTkn}(\mathsf{st}, \kappa, w)$*: Given a state* st*, a secret key* κ*, and a keyword* w*, it outputs a search token* $\mathfrak{s}$.
- $\mathcal{R} \leftarrow \mathsf{Srch}(\mathfrak{s}, \mathsf{EDB})$*: Given a search token* $\mathfrak{s}$ *and an encrypted database* EDB*, it outputs the search result* $\mathcal{R}$.
- $(\mathfrak{u}, \mathsf{st}') \leftarrow \mathsf{UpdtTkn}(\mathsf{st}, \kappa, f, \mathcal{W}_f)$*: Given a state* st*, a secret key* κ*, a document* f *and its keyword set* $\mathcal{W}_f$*, it outputs an update token* $\mathfrak{u}$ *and a new state* st'.
- $\mathsf{EDB}' \leftarrow \mathsf{Updt}(\mathfrak{u}, \mathsf{EDB})$*: Given an update token* $\mathfrak{u}$ *and an encrypted database* EDB*, it outputs an updated encrypted database* EDB'.

Definition 3 (Correctness of DSSE). *For all security parameter* λ*, all* $(\kappa, \mathsf{st}, \mathsf{EDB}) \leftarrow \mathsf{Setup}(1^\lambda)$*, and all sequences of* Srch, Updt *operations over encrypted database* EDB *using tokens generated respectively from* $\mathsf{SrchTkn}(\mathsf{st}, \kappa, w)$*, and* $\mathsf{UpdtTkn}(\mathsf{st}, \kappa, f, \mathcal{W}_f)$*,* Srch *returns the correct results w.r.t the inputs* $(f, \mathcal{W}_f)$ *of* UpdtTkn*, except with negligible probability in* λ.

To define the security of a DSSE scheme, we borrow the concepts of *history*, *view* and *trace* [81]. The history contains the sensitive information that the user wants to keep private, namely the document plaintexts $\mathcal{D}$ and the sequence of queried keywords $\mathbf{w}$:

Definition 4 (History). *A history over* $\mathcal{D}$ *is a tuple:*

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

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

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

Next, the view specifies what the server can see when running the DSSE protocol, namely the document identifiers $\mathsf{id}(\mathcal{D})$, the encryption of each document $\mathcal{E}(\mathcal{D}[i])$, an encrypted database EDB, and the query tokens (trapdoors) $\mathfrak{s}_i$:

Definition 5 (View). *The view of* H_t *under key* κ *is the vector:*

$$V_\kappa(H_t) = (\mathsf{id}(\mathcal{D}), \mathcal{E}(\mathcal{D}[1]), \ldots, \mathcal{E}(\mathcal{D}[N]), \mathsf{EDB}, \mathfrak{s}_1, \ldots, \mathfrak{s}_t, \mathfrak{u}_1, \ldots, \mathfrak{u}_t)$$

where $\mathfrak{s}_i$ *and* $\mathfrak{u}_i$ *are the search query token and the update token for the* i*-th query, respectively. The partial view* $V_\kappa^s(H_t)$ *only contains the tokens up to* $\mathfrak{s}_s$ *and* $\mathfrak{u}_s$*, with* $s \le t$.

Finally, the *trace* of a history models the actual leakage of the DSSE scheme. Before defining trace, we first formulate three types of leakage of DSSE schemes, including the search pattern, result pattern, and search volume as follows.

Definition 6 (Search Pattern). *The search pattern (*sp*), denoted as* $\Phi_\mathbf{w}$*, given a dataset* $\mathcal{D}$ *and a query vector* $\mathbf{w}$ *is a symmetric binary matrix of size* $t \times t$ *such that:*

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

Definition 7 (Result Pattern and Search Volume). *The result pattern (*rp*), denoted as* $\Pi_\mathbf{w}$*, given a dataset* $\mathcal{D}$ *and a query vector* $\mathbf{w}$ *is a binary matrix of size* $t \times N$ *such that:*

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

*The search volume (*sv*) of a keyword* $\mathbf{w}[i]$ *can be derived from the result pattern as:* $\mathsf{sv}(\mathbf{w}[i]) = \sum_{j=1}^{N} \Pi_\mathbf{w}[i, j]$.

Intuitively, the search pattern reveals which queries have the same underlying keyword, the result pattern reveals which documents contain the queried keyword, and the search volume indicates the number of documents that match the queried keyword.

Definition 8 (Trace). *The trace of* H_t *is the vector:*

$$T(H_t) = (\mathsf{id}(\mathcal{D}), N, m, \Phi_\mathbf{w}, \Pi_\mathbf{w}).$$

3 Models

System Model. Our system includes a user and a server. The user outsources its database, including encrypted documents, and an encrypted search index to support private keyword search and document update.

Threat and Security Models. We consider the standard threat model of DSSE, where the server can be corrupt while the user is honest [16]. We assume a semi-honest (honest but curious) adversary that adheres to the protocol but attempts to infer information about the user's queries and data (i.e., keyword search and document updates).

We are ready to define the adaptive semantic security for DSSE. Informally, the definition states that a DSSE 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 captures all the information that is known by the adversary.

Definition 9 (Adaptive Security of DSSE). *A DSSE scheme is adaptively semantically secure if for all* $t \in \mathbb{N}$ *and for all (non-uniform) probabilistic polynomial time algorithm (the simulator)* $\mathcal{S}$ *such that for all traces* T_t *of length* t*, all polynomial 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 $g : \{0,1\}^{|H_t|} \to \{0,1\}^{\mathsf{poly}(|H_t|)}$, all $0 \le s \le t$, and sufficiently large λ:

$$\left|\Pr[\mathcal{A}(V_\kappa^s(H_t)) = g(H_t^s)] - \Pr[\mathcal{S}(T(H_t^s)) = g(H_t^s)]\right| < \delta(\lambda),$$

where $H_t \leftarrow \mathcal{H}_t$, $(\kappa, \mathsf{st}, \mathsf{EDB}) \leftarrow \mathsf{Setup}(1^\lambda)$, and probabilities are taken over $\mathcal{H}_t$ and the internal coins of $\mathcal{A}, \mathcal{S}$ and the underlying Setup, SrchTkn, Srch, UpdtTkn, Updt *algorithms.*

We introduce a leakage function family $\mathcal{L}_H = \{\mathcal{L}_H^{\mathsf{Setup}}, \mathcal{L}_H^{\mathsf{Srch}}, \mathcal{L}_H^{\mathsf{Updt}}\}$ to control exactly the information of history H leaked during setup, search, and update, respectively. When an oracle is queried for the t-th operation, any function in $\mathcal{L}_H$ is instantiated with H being the history consisting of the first $(t-1)$ operations and with the t-th operation as a function input. It records the leakage incurred due to the last operation while taking all historical operations into consideration. We omit H if there is no ambiguity. Before any query (i.e., $H = \{\varnothing\}$), $\mathcal{L} = \mathcal{L}^{\mathsf{Setup}}$.

Definition 10 (Forward Privacy of DSSE). *An adaptively-secure* DSSE *is forward-private if the update leakage $\mathcal{L}_H^{\mathsf{Update}}(f, \mathcal{W}_f)$ of any update $(f, \mathcal{W}_f)$ can be written as $\mathcal{L}'(f)$, where $\mathcal{L}'$ is stateless.*

Differential privacy (DP) [31] is a widely recognized formal framework for protecting privacy, and it has been adopted in various domains, such as database security [20, 81, 89, 91]. The notion is defined by a privacy parameter ϵ, which guarantees that when two neighboring inputs are given to a DP algorithm, the probability of obtaining any particular output remains nearly the same. As a result, an observer cannot reliably infer sensitive information about the underlying input from the output. Here, we represent a DSSE scheme as a function SE that takes the user input (a dataset $\mathcal{D}$ and a query vector $\mathbf{w}$) and outputs the leakage (a trace T). We borrow the definitions of differential privacy for documents and keywords of DSSE from [81], which imply search result pattern and search volume protection as follows.

Definition 11 (Differential Privacy for Documents [81]). *A DSSE characterized by function* SE *provides ϵ-DP for documents iff for any keyword list of length t, $\mathbf{w} \in W^t$, for any pair of neighboring databases $\mathcal{D}, \mathcal{D}' \in 2^{2^W}$ (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:*

$$\Pr[\mathsf{SE}(\mathcal{D}, \mathbf{w}) = T] \le e^{t\cdot\epsilon}\Pr[\mathsf{SE}(\mathcal{D}', \mathbf{w}) = T]. \tag{1}$$

We refer to this notion "differential privacy *for documents*" instead of "for result and volume 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 indicates that a DSSE that provides ϵ-DP for documents also provides ϵ-DP for result pattern and search volume since the similarity in result pattern implies the similarity in search volume (formally stated in Corollary 1). Note that document size leakage in search result is considered out-of-scope in our work.

Definition 12 (Differential Privacy for Keywords [81]). *A DSSE characterized by function* SE *provides ϵ-DP for keywords iff for any database $\mathcal{D} \in 2^{2^W}$, for any pair of neighboring keyword lists $\mathbf{w}, \mathbf{w}' \in W^{|\mathbf{w}|}$ ($\mathbf{w}$ and $\mathbf{w}'$ differ in only one element, $\mathbf{w}[i] \neq \mathbf{w}'[i]$), and any trace T, the following holds:*

$$\Pr[\mathsf{SE}(\mathcal{D}, \mathbf{w}) = T] \le e^{d\cdot\epsilon}\Pr[\mathsf{SE}(\mathcal{D}, \mathbf{w}') = T]. \tag{2}$$

Here, d is the number of different documents between $\mathcal{D}(\mathbf{w}[i])$ and $\mathcal{D}(\mathbf{w}'[i])$. A scheme that ensures differential privacy makes it difficult for an observer, even when considering the trace (permitted leakage), to distinguish whether the user's query corresponds to one keyword list or the other. Consequently, this property ensures search pattern privacy, preventing the server from inferring whether two queries are associated with the same keyword.

4 Rerandomized PIR (RePIR)

PIR [23, 41, 46] enables private retrieval on a *public* database (owned by the server) without revealing the queried data. We propose a new PIR-based scheme for *private* database (encrypted and owned by the user) that supports output rerandomization, allowing a fresh random mask to be applied to the retrieved result. In other words, our scheme permits the server to obtain the content of the queried item that is decrypted and then re-encrypted with a new mask (without leaking the corresponding item index). We introduce RePIR, which stands for Rerandomized PIR, formally defined as follows.

4.1 Definitions

Definition 13 (RePIR). *A RePIR scheme is a tuple of PPT algorithms defined as follows:*

- $(\kappa, \mathsf{st}, \mathsf{EDB}) \leftarrow \mathsf{Setup}(1^\lambda, \mathsf{IDX}, N, m)$*: Given a security parameter λ, and an index* IDX *of size $N \times m$, it outputs a secret key κ, a state* st *and an encrypted database* EDB.
- $(\mathsf{qu}, \mathbf{r}) \leftarrow \mathsf{Query}(v, \kappa, \mathsf{st})$*: Given an index $v \in [m]$, a secret key κ and a state* st*, it outputs a query* qu *and a random mask* $\mathbf{r}$.
- $\mathbf{d} \leftarrow \mathsf{Answer}(\mathsf{qu}, \mathsf{EDB})$*: Given a query* qu *and an encrypted database* EDB*, it outputs a masked result* $\mathbf{d} = \mathsf{IDX}[*, v] \boxplus \mathbf{r}$.

Correctness. When the user and the server execute the RePIR protocol faithfully, the user should recover its desired data column with all but negligible probability in the implicit correctness parameter. We say that a RePIR scheme has *correctness error* δ if, on index size $N \times m$, with $N, m \in \mathbb{N}$, for all indices $\mathsf{IDX} = \begin{bmatrix}\mathbf{d}_1 & \mathbf{d}_2 & \dots & \mathbf{d}_m\end{bmatrix}$, and for all indices $v \in [m]$, the following probability is at least $1-\delta$:

$$\Pr\left[\mathbf{d}_v \boxplus \mathbf{r} = \hat{\mathbf{d}}_v : \begin{array}{l}(\kappa, \mathsf{st}, \mathsf{EDB}) \leftarrow \mathsf{Setup}(1^\lambda, \mathsf{IDX}, N, m)\\ (\mathsf{qu}, \mathbf{r}) \leftarrow \mathsf{Query}(v, \kappa, \mathsf{st})\\ \hat{\mathbf{d}}_v \leftarrow \mathsf{Answer}(\mathsf{qu}, \mathsf{EDB})\end{array}\right].$$

Security. The user's query should reveal no information about its desired data column. That is, we say that a RePIR scheme is $(T, \tilde{\epsilon})$-secure if, for all adversaries $\mathcal{A}$ running in time at most T, on index IDX of size $N \times m$, and for all $v_1, v_2 \in \mathbb{N}$:

$$\Big|\Pr[\mathcal{A}(1^N, \mathsf{qu}) = 1 : (\mathsf{qu}, \_) \leftarrow \mathsf{Query}(v_1, \kappa, \mathsf{st})] - \Pr[\mathcal{A}(1^N, \mathsf{qu}) = 1 : (\mathsf{qu}, \_) \leftarrow \mathsf{Query}(v_2, \kappa, \mathsf{st})]\Big| \le \tilde{\epsilon}.$$

4.2 Our Concrete RePIR Scheme

We introduce our RePIR construction for private information retrieval, which supports fresh random masking for each query. Figure 1 presents our RePIR scheme. Given an index IDX of size $N \times m$,

The parameters of the construction are an index representing as a matrix $\mathsf{IDX} \in \hat{\mathbb{Z}}_{p'}^{N\times m}$, LWE parameters (n, q, χ) with q is a prime number, a ciphertext modulus $p \ll q$, a plaintext modulus $p' \cdot 2^{z+1} < p$, a LWE matrix $\mathbf{A} \in \hat{\mathbb{Z}}_q^{m\times n}$, secret random matrices $\mathbf{B} \in \hat{\mathbb{Z}}_q^{n\times n}$ with linearly independent rows, and $\mathbf{C} \in \hat{\mathbb{Z}}_q^{N\times n}$. We use PRF $F : \{0,1\}^\lambda \times \{0,1\}^* \to \hat{\mathbb{Z}}_p$ for encryption. Matrices $\mathbf{B}$ and $\mathbf{C}$ can be derived in practice using PRF $\tilde{F} : \{0,1\}^\lambda \times \{0,1\}^* \to \hat{\mathbb{Z}}_q^n$. Define the scalar $\Delta \leftarrow \lfloor q/p \rfloor \in \mathbb{Z}$.

$\mathsf{Setup}(1^\lambda, \mathsf{IDX}, N, m)$: ▷ Executed by the user

1: $\kappa \xleftarrow{\$} \{0,1\}^\lambda$
2: Initialize $\mathsf{st} \leftarrow (\mathsf{st}_1, \mathsf{st}_2, \ldots, \mathsf{st}_N)$, where $\mathsf{st}_i \leftarrow 0$ for $i \in [N]$
3: **for** $i \in [N], j \in [m]$ **do**
4: $\quad \mathsf{EIDX}[i,j] \leftarrow (\mathsf{IDX}[i,j] \lll z) + F(\kappa, j\|i\|\mathsf{st}_i) \in \hat{\mathbb{Z}}_p$
5: $\mathbf{B}[i,*] \leftarrow \tilde{F}(\kappa, i)$ for $i \in [n]$ s.t. $\mathbf{B}^{-1}$ exists
6: $\mathbf{C}[i,*] \leftarrow \tilde{F}(\kappa, i\|\mathsf{st}_i)$ for $i \in [N]$
7: Compute $\mathsf{hint} \leftarrow \mathsf{EIDX} \cdot \mathbf{A} \cdot \mathbf{B}^{-1} - \mathbf{C} \in \hat{\mathbb{Z}}_q^{N\times n}$
8: Return $(\kappa, \mathsf{st}, \mathsf{EDB} \leftarrow (\mathsf{EIDX}, \mathsf{hint}))$

$\mathsf{Query}(v, \kappa, \mathsf{st})$: ▷ Executed by the user

1: $\mathbf{B}[i,*] \leftarrow \tilde{F}(\kappa, i)$ for $i \in [n]$; $\mathbf{C}[i,*] \leftarrow \tilde{F}(\kappa, i\|\mathsf{st}_i)$ for $i \in [N]$
2: Sample $\mathbf{s} \xleftarrow{\$} \hat{\mathbb{Z}}_q^n$ and $\mathbf{e} \xleftarrow{\$} \hat{\chi}^m$; $\mathbf{u} \leftarrow 0^m$; $\mathbf{u}[v] = 1$
3: Compute $\mathbf{s}' \leftarrow \mathbf{B} \cdot \mathbf{s} \in \hat{\mathbb{Z}}_q^n$ and $\mathbf{q} \leftarrow \mathbf{A} \cdot \mathbf{s} + \mathbf{e} + \Delta \cdot \mathbf{u} \in \hat{\mathbb{Z}}_q^m$
4: Sample $\mathbf{r} \xleftarrow{\$} \hat{\mathbb{Z}}_{p'}^N$ and compute $\mathbf{t} \leftarrow \lfloor \mathbf{C} \cdot \mathbf{s}' \rfloor_\Delta / \Delta \in \hat{\mathbb{Z}}_p^N$
5: **for** $i \in [N]$ **do**
6: $\quad \mathbf{m}[i] \leftarrow (\mathbf{r}[i] \lll z) - \mathbf{t}[i] - F(\kappa, v\|i\|\mathsf{st}_i) \in \hat{\mathbb{Z}}_p$
7: Return $(\mathsf{qu} \leftarrow (\mathbf{q}, \mathbf{s}', \mathbf{m}), \mathbf{r})$

$\mathsf{Answer}(\mathsf{qu}, \mathsf{EDB})$: ▷ Executed by the server

8: Parse $\mathsf{qu} = (\mathbf{q}, \mathbf{s}', \mathbf{m})$, and $\mathsf{EDB} = (\mathsf{EIDX}, \mathsf{hint})$
9: Compute $\hat{\mathbf{d}} \leftarrow \lceil \mathsf{EIDX} \cdot \mathbf{q} - \mathsf{hint} \cdot \mathbf{s}' \rfloor_\Delta / \Delta \in \hat{\mathbb{Z}}_p^N$
10: Return $\mathbf{d} \leftarrow (\hat{\mathbf{d}} + \mathbf{m}) \ggg z \in \hat{\mathbb{Z}}_{p'}^N$

Figure 1: Our proposed RePIR scheme.

the user wishes to query for a column at index $v \in [m]$. The user builds a unit vector $\mathbf{u} \in \hat{\mathbb{Z}}_2^m$ (i.e., the vector of all zeros with a single '1' at index v), element-wise encrypts it using Regev's LWE-based encryption scheme (§2.1), and sends this encrypted vector to the server. The server computes the matrix-vector product between the index and the query vector, followed by unmasking the hint while also adding a random value to the final output. The added random value is fresh per query and can be precomputed by the user, setting the stage for later computation/processing steps on the new masked data if necessary.

We assume that $\mathsf{IDX} \in \mathbb{Z}_{p'}^{N\times m}$ is encrypted to protect confidentiality. RePIR employs a pseudorandom function (PRF) function F whose output is in $\hat{\mathbb{Z}}_p$ for encryption. Furthermore, RePIR needs to reserve a bit in the original data before encryption to avoid rounding error when evaluating the output answer (RePIR.Answer, lines 9–10). Note that $\lfloor a + b \rceil_\Delta / \Delta = a + \lfloor b \rceil_\Delta / \Delta + e$, where $e \in \{0,1\}$ is a small error. Thus, "reserving" a bit for the error in the data before being encrypted with PRF can "rule out" such error after unpacking the hint, rounding then adding the random mask in the later step. Assume that $p' \cdot 2^{z+1} < p$, with $z = 1$. For each $\mathsf{IDX}[i,j]$ with $i \in [N]$ and $j \in [m]$, the user encrypts it using a pseudorandom function (PRF) $F : \{0,1\}^\lambda \times \{0,1\}^* \to \hat{\mathbb{Z}}_p$ as:

$$\mathsf{EIDX}[i,j] \leftarrow (\mathsf{IDX}[i,j] \lll z) + F(\kappa, j\|i\|\mathsf{st}_i) \pmod p,$$

where st_i is the update state of the data in row i (a counter initialized with 0, and incremented by one for each update). The final encrypted index is $\mathsf{EIDX} \in \hat{\mathbb{Z}}_p^{N\times m}$.

SimplePIR [46] uses Regev encryption to encrypt PIR queries, but the hint is stored on the user side, which is large in our case if applied for SSE because of the LWE parameter n (e.g., $n = 2048$), and also the server cannot unfold the hint since that reveals the queried data. As our output is masked with a random value, we propose a new approach to store the hint on the server. Let (n, q, χ) be LWE parameters, where q is a prime number, and $\mathbf{A}$ is a LWE matrix. Let $\mathbf{B} \in \hat{\mathbb{Z}}_q^{n\times n}$ and $\mathbf{C} \in \hat{\mathbb{Z}}_q^{N\times n}$ be two secret random matrices independent of the database that can be derived in practice using a PRF $\tilde{F} : \{0,1\}^\lambda \times \{0,1\}^* \to \hat{\mathbb{Z}}_q^n$ as: $\mathbf{B}[i,*] \leftarrow \tilde{F}(\kappa, i)$ for $i \in [n]$, and $\mathbf{C}[i,*] \leftarrow \tilde{F}(\kappa, i\|\mathsf{st}_i)$ for $i \in [N]$. Conditioned on $\mathbf{B}$ having linearly independent rows over the prime field, its inverse $\mathbf{B}^{-1}$ exists. Therefore, the user computes and outsources a hint to the server as

$$\mathsf{hint} \leftarrow \mathsf{EIDX} \cdot \mathbf{A} \cdot \mathbf{B}^{-1} - \mathbf{C} \in \hat{\mathbb{Z}}_q^{N\times n},$$

Let $\mathbf{s} \xleftarrow{\$} \hat{\mathbb{Z}}_q^n$ and $\mathbf{e} \xleftarrow{\$} \hat{\chi}^m$. The Regev encryption of a unit vector $\mathbf{u} \in \hat{\mathbb{Z}}_p^m$ ($\mathbf{u}[v] = 1$ and $\mathbf{u}[j] = 0 \,\forall\, j \neq v$) is a vector:

$$\mathbf{q} = \mathsf{Enc}(\mathbf{u}) \leftarrow (\mathbf{A}, \mathbf{A} \cdot \mathbf{s} + \mathbf{e} + \lfloor q/p \rfloor \cdot \mathbf{u}) \pmod q,$$

Matrix $\mathbf{A}$ is a pseudorandom matrix and can be used to hide polynomially many secrets $\mathbf{s}$, provided that each query uses an independent vector $\mathbf{s}$ and error vector $\mathbf{e}$. Also, the user computes the mask $\mathbf{m}$ as follows:

$$\mathbf{t} \leftarrow \lfloor \mathbf{C} \cdot \mathbf{s}' \rfloor_\Delta / \Delta \pmod p, \quad \mathbf{r} \xleftarrow{\$} \hat{\mathbb{Z}}_{p'}^N,$$
$$\mathbf{m}[i] \leftarrow (\mathbf{r}[i] \lll z) - \mathbf{t}[i] - F(\kappa, v\|i\|\mathsf{st}_i) \pmod p,$$

for $i \in [N]$. Let $\mathbf{s}' \leftarrow \mathbf{B} \cdot \mathbf{s} \in \hat{\mathbb{Z}}_q^n$. Under the LWE assumption (§2.1), given $(\mathbf{A}, \mathbf{A} \cdot \mathbf{s} + \mathbf{e})$, recovering the secret $\mathbf{s}$ is computationally hard. Also, $\mathbf{B}$ is a secret matrix whose linearly independent rows form a basis that spans $\hat{\mathbb{Z}}_q^n$, therefore $\mathbf{B} \cdot \mathbf{s} \overset{c}{\approx} \tilde{\mathbf{r}}$, where $\tilde{\mathbf{r}} \xleftarrow{\$} \hat{\mathbb{Z}}_q^n$. Together with the encrypted query $\mathbf{q}$, the user encloses $\mathbf{s}'$ and $\mathbf{m}$ for decryption, removing the complementary hint incurred by $\mathbf{C} \cdot \mathbf{s}'$, and adding a random mask value $\mathbf{r}$ simultaneously. The server uses the encrypted query to retrieve the queried data $\mathsf{EIDX}[*, v]$ but hidden by the complementary hint value $\mathbf{C} \cdot \mathbf{s}'$:

$$\begin{aligned}
\hat{\mathbf{d}} &= \lceil \mathsf{EIDX} \cdot \mathbf{q} - \mathsf{hint} \cdot \mathbf{s}' \rfloor_\Delta / \Delta \\
&= \lceil \mathsf{EIDX} \cdot (\mathbf{A} \cdot \mathbf{s} + \mathbf{e} + \Delta \cdot \mathbf{u}) \\
&\quad - (\mathsf{EIDX} \cdot \mathbf{A} \cdot \mathbf{B}^{-1}) \cdot \mathbf{B} \cdot \mathbf{s} + \mathbf{C} \cdot \mathbf{s}' \rfloor_\Delta / \Delta \\
&= \lceil \mathsf{EIDX} \cdot \Delta \cdot \mathbf{u} + \mathbf{C} \cdot \mathbf{s}' + \mathsf{EIDX} \cdot \mathbf{e} \rfloor_\Delta / \Delta \\
&= \mathsf{EIDX}[*, v] + \lfloor \mathbf{C} \cdot \mathbf{s}' \rfloor_\Delta / \Delta + \mathbf{e}' \pmod p,
\end{aligned}$$

with $\mathbf{e}' \in \{0,1\}^N$ and $\mathbf{e}'[i] \in \hat{\mathbb{Z}}_p$. After unpacking the hint, the server continues to remove the complementary component $\lfloor \mathbf{C} \cdot \mathbf{s}' \rfloor_\Delta / \Delta$, then obtains the retrieved data $\mathsf{IDX}[*, v]$ masked with a

Setup(1^λ): ▷ Executed by the user

1: IDX $\leftarrow 0^{N\times m}$, where IDX$[i,j] \in \mathbb{Z}_{p'}^{\ell}$ for $i \in [N]$ and $j \in [m]$
2: $(\kappa, \mathsf{st}, \mathsf{EDB}) \leftarrow \mathsf{RePIR.Setup}(1^\lambda, \mathsf{IDX}, m, N)$
3: **return** $(\kappa, \mathsf{st}, \mathsf{EDB})$

Figure 2: FROST setup algorithm.

random value $\mathbf{r}$ by adding $\mathbf{m}$:

$$\begin{aligned}\mathbf{d}[i] &= (\hat{\mathbf{d}}[i] + \mathbf{m}[i]) \ggg z \\ &= \big(\mathsf{EIDX}[i,v] + \lfloor \mathbf{C}\cdot\mathbf{s}'\rceil_\Delta/\Delta[i] + \mathbf{e}'[i] + (\mathbf{r}[i] \lll z) \\ &\quad - \lfloor \mathbf{C}\cdot\mathbf{s}'\rceil_\Delta/\Delta[i] - F(\kappa, v\|i\|\mathsf{st}_i)\big) \ggg z \\ &= \big((\mathsf{IDX}[i,v] \lll z) + (\mathbf{r}[i] \lll z) + \mathbf{e}'[i]\big) \ggg z \\ &= \mathsf{IDX}[i,v] + \mathbf{r}[i] \pmod{p'},\end{aligned}$$

for $i \in [N]$. RePIR outputs correctly as long as for all $i \in [N]$, the absolute value of the error $\langle \mathsf{EIDX}[i,*]\cdot\mathbf{e}\rangle$ is less than or equal to $\Delta = \lfloor q/p \rfloor$.

Theorem 1 (RePIR). *On database size $N \times m$, correctness failure probability δ, adversary time T, and security failure probability ϵ, let*

- *χ be the discrete Gaussian distribution with variance σ^2,*
- *(n, q, χ) be LWE parameters achieving (T, ϵ)-security for m LWE samples, and*
- *$p \in \mathbb{N}$ and $p' \in \mathbb{N}$ be a ciphertext and a plaintext modulus, respectively, chosen to satisfy $p' \cdot 2^{z+1} < p$ and*

$$\lfloor q/p \rfloor \geq \sqrt{2}/2 \cdot \sigma \cdot p \cdot \sqrt{m \cdot \ln(2/\delta)}. \tag{3}$$

Then, for a random LWE matrix $\mathbf{A} \xleftarrow{\$} \hat{\mathbb{Z}}_q^{m\times n}$, RePIR is a $(T - O(m), \epsilon)$-secure PIR scheme on database size $N \times m$, over plaintext space $\hat{\mathbb{Z}}_{p'}$, with correctness error δ.

Proof. See Appendix A. □

We prove RePIR security via query indistinguishability (Lemma 1 gives single query and Corollary 1 extends to Q queries via hybrid arguments). Therefore, this implies RePIR is simulation-based secure (SIM-secure), since RePIR query transcripts can be simulated from allowed leakage (e.g., public parameters N and m).

5 Our Proposed FROST Scheme

Building on RePIR, we present our proposed FROST scheme.

5.1 Search Index Initialization

In FROST, the user builds an index for keyword-document representation. For high-level idea, we use a probabilistic data structure (i.e., BF) to create a compressed index. Suppose there are N documents, the user extracts a set of unique keywords $\mathcal{W}_f$ for each document $f \in [N]$ and computes its BF representation as $\mathbf{u}_f \leftarrow \mathsf{BF.Gen}(\mathcal{W}_f) \in \{0,1\}^m$. The search index contains N BF vectors, which is interpreted as an $N \times m$ binary matrix $\mathsf{IDX} = [\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_N] \in \{0,1\}^{N\times m}$. Searching a keyword incurs checking BF membership by reading K columns $\mathbf{d}_{H_1(w)}, \mathbf{d}_{H_2(w)}, \ldots, \mathbf{d}_{H_K(w)}$ in IDX, where $\mathbf{d}_{H_k(w)} = \mathsf{IDX}[*, H_k(w)]$ for $k \in [K]$, with H_k is a mapping of BF and K is the BF parameter (defined in §2). The keyword w appears in document $i \in [N]$ iff $\sum_{k=1}^K \mathbf{d}_{H_k(w)}[i] = K$.

SrchTkn(st, κ, w): ▷ Executed by the user

1: TPSet $\leftarrow \{\varnothing\}$, FPSet $\leftarrow \{\varnothing\}$
2: **for** $i = 1$ to N **do**
▷ Sample TPSet and FPSet independently with probabilities p and q, resp.
3: With probability p: TPSet $\leftarrow$ TPSet $\cup \{i\}$
4: With probability q: FPSet $\leftarrow$ FPSet $\cup \{i\}$
5: Initialize mv $\leftarrow 0^N \in \mathbb{Z}_{p'}^N$
6: **for** $k = 1$ to K **do**
▷ Create a RePIR query for retrieving column u_k with $k \in [K]$
7: $u_k \leftarrow H_k(w)$, $(\mathsf{qu}_k, \mathbf{r}_k) \leftarrow \mathsf{RePIR.Query}(u_k, \kappa, \mathsf{st})$
▷ Create a transformation vector and compute the corresponding
▷ match value for each document, depending on whether the document
▷ is selected as a false positive, a true positive, or a random non-match
8: **for** $i = 1$ to N **do**
9: $\mathbf{v}_{\mathsf{nm}} \leftarrow F^*(\kappa, u_k\|i\|\mathsf{st}_i\|0) + \mathbf{r}_k[i] \in \mathbb{Z}_{p'}^{\ell}$
10: $\mathbf{v}_{\mathsf{m}} \leftarrow F^*(\kappa, u_k\|i\|\mathsf{st}_i\|1) + \mathbf{r}_k[i] \in \mathbb{Z}_{p'}^{\ell}$
11: **if** $i \in$ FPSet **then** ▷ If document i is selected as a false positive
12: Sample $y_{i,k} \xleftarrow{\$} \mathbb{Z}_{p'}$
13: Compute $\mathbf{x}_{i,k} \in \mathbb{Z}_{p'}^{\ell}$, s.t. $\langle \mathbf{v}_{\mathsf{nm}}, \mathbf{x}_{i,k}\rangle = \langle \mathbf{v}_{\mathsf{m}}, \mathbf{x}_{i,k}\rangle = y_{i,k} \pmod{p'}$
14: **else if** $i \in$ TPSet **then** ▷ Else if document i is selected as a true positive
15: Sample $\mathbf{x}_{i,k} \xleftarrow{\$} \mathbb{Z}_{p'}^{\ell}$; and compute $y_{i,k} \leftarrow \langle \mathbf{v}_{\mathsf{m}}, \mathbf{x}_{i,k}\rangle \pmod{p'}$
16: **else** ▷ Otherwise, document i is selected as a random non-match
17: Sample $\mathbf{x}_{i,k} \xleftarrow{\$} \mathbb{Z}_{p'}^{\ell}$, and $y_{i,k} \xleftarrow{\$} \mathbb{Z}_{p'}$
▷ Store transformation vectors and accumulate match values
18: $\mathsf{tv}_k[i] \leftarrow \mathbf{x}_{i,k}$; $\mathsf{mv}[i] \leftarrow \mathsf{mv}[i] + y_{i,k} \pmod{p'}$
19: **return** $\mathfrak{s} \leftarrow (\{(\mathsf{qu}_k, \mathsf{tv}_k)\}_{k\in[K]}, \mathsf{mv})$

Srch($\mathfrak{s}$, EDB): ▷ Executed by the server

20: Parse $\mathfrak{s} = (\{(\mathsf{qu}_k, \mathsf{tv}_k)\}_{k\in[K]}, \mathsf{mv})$
21: **for** $k = 1$ to K **do**
▷ Retrieve the column w.r.t query qu_k and obtain the rerandomized output
22: $\hat{\mathbf{d}}_k \leftarrow \mathsf{RePIR.Answer}(\mathsf{qu}_k, \mathsf{EDB})$
23: **for** $i = 1$ to N **do**
▷ Compute dot product with the given transformation vector
24: $\mathbf{d}_k[i] \leftarrow \langle \hat{\mathbf{d}}_k[i], \mathsf{tv}_k[i]\rangle \pmod{p'}$
25: **for** $k = 1$ to K **do**
26: $\mathbf{d} \leftarrow \sum_{k=1}^K \mathbf{d}_k \pmod{p'}$ ▷ Accumulate results over K columns
27: $\mathcal{R} \leftarrow \{\varnothing\}$
28: **for** $i = 1$ to N **do**
29: **if** $\mathbf{d}[i] = \mathsf{mv}[i]$ **then** $\mathcal{R} \leftarrow \mathcal{R} \cup \{i\}$ ▷ Check with the given match value
30: **return** $\mathcal{R}$

Figure 3: FROST search algorithm.

Updating a document f recreates a new BF representation of the updated keywords and overwrites the corresponding row $\mathsf{IDX}[f, *]$.

The user can build a secure search system using RePIR with IDX as input index, similar to the existing work using PIR [28]. In particular, RePIR allows the user to identify query-matching documents without revealing them directly as the results are masked with random values. The user can then unmask these values to know which documents are matched, followed by using other techniques (e.g., ORAM) to retrieve the documents without exposing the result pattern. In this work, we propose FROST, a new scheme that achieves single-round-trip keyword search while protecting search result pattern and search volume with differential privacy.

We first introduce a new approach in building the search index to facilitate obfuscating the results of keyword search via dot products as follows. We assume that $\mathsf{IDX}[i,j] \in \mathbb{Z}_{p'}^{\ell}$ for $i \in [N]$ and $j \in [m]$ and F^* is a PRF with secret key κ as input and output a value in $\mathbb{Z}_{p'}^{\ell}$. A value at row i and column j in the search index is represented by an integer vector generated as $\mathbf{v}_{\mathsf{m}} \leftarrow F^*(\kappa, j\|i\|\mathsf{st}_i\|1) \in \mathbb{Z}_{p'}^{\ell}$ if keyword j appears in document i (i.e., corresponding to match value 1),

$\underline{\mathsf{UpdtTkn}(\mathsf{st}, \kappa, f, \mathcal{W}_f)}$: ▷ Executed by the user

1: $\mathbf{u} \leftarrow \{0\}^m$; $\mathsf{st}'_f \leftarrow \mathsf{st}_f + 1$
2: **for** $j = 1$ to m **do**
3: $\quad \mathbf{u}[j] \leftarrow F^*(\kappa, j \| f \| \mathsf{st}'_f \| 0) \in \mathbb{Z}_{p'}^{\ell}$ ▷ Initialize an empty BF
4: **for each** $w \in \mathcal{W}_f$, $k = 1$ to K **do** ▷ Insert keywords into BF
5: $\quad v_k \leftarrow H_k(w)$, $\mathbf{u}[v_k] \leftarrow F^*(\kappa, v_k \| f \| \mathsf{st}'_f \| 1) \in \mathbb{Z}_{p'}^{\ell}$
6: **for** $j = 1$ to m **do** ▷ Encrypt the updated row with PRF
7: $\quad \mathbf{u}'[j] \leftarrow (\mathbf{u}[j] \lll z) + F(\kappa, j \| f \| \mathsf{st}'_f) \in \mathbb{Z}_p^{\ell}$
8: $\mathbf{B}[f, *] \leftarrow \tilde{F}(\kappa, f)$ for $f \in [N]$; $\mathbf{C}'[f, *] \leftarrow \tilde{F}(\kappa, f \| \mathsf{st}'_f)$
▷ Update the corresponding row of the hint
9: $\mathsf{hint}'_f \leftarrow \mathbf{u}' \cdot \mathbf{A} \cdot \mathbf{B}^{-1} - \mathbf{C}'[f, *]$, where $\mathsf{hint}'_f[j] \in \mathbb{Z}_q^{\ell}$ for $j \in [n]$
10: **return** $(\mathsf{st}', \mathfrak{u} \leftarrow (f, \mathbf{u}', \mathsf{hint}'_f))$

$\underline{\mathsf{Updt}(\mathfrak{u}, \mathsf{EDB})}$: ▷ Executed by the server

11: Parse $\mathfrak{u} = (f, \mathbf{u}', \mathsf{hint}'_f)$ and $\mathsf{EDB} = (\mathsf{EIDX}, \mathsf{hint})$
12: $\mathsf{EIDX}[f, *] \leftarrow \mathbf{u}'$, $\mathsf{hint}[f, *] \leftarrow \mathsf{hint}'_f$
13: **return** $\mathsf{EDB}' \leftarrow (\mathsf{EIDX}, \mathsf{hint})$

Figure 4: FROST update algorithm.

and $\mathbf{v}_{\mathsf{nm}} \leftarrow F^*(\kappa, j \| i \| \mathsf{st}_i \| 0) \in \mathbb{Z}_{p'}^{\ell}$ otherwise (i.e., corresponding to non-match value 0). In keyword search, the user sends transformation vectors tv to compute dot products with these state vectors (i.e., $\langle \mathbf{v}_{\mathsf{m}}, \mathsf{tv} \rangle$ or $\langle \mathbf{v}_{\mathsf{nm}}, \mathsf{tv} \rangle$), collapsing them into scalar values that may belong to true positives, false positives, or random non-matches. Also, the user encloses a match value mv for the server to identify matched/unmatched documents, which corresponds to the true value if the document is a match, i.e., $\mathsf{mv} = \langle \mathbf{v}_{\mathsf{m}}, \mathsf{tv} \rangle$.

Setup. Figure 2 presents the setup procedure of FROST. The user initializes an empty search index IDX of size $N \times m$, where N is the number of documents and m is Bloom filter size, then executes RePIR.Setup algorithm with security parameter λ, IDX, N, and m as input to generate three components: a secret key κ, a local state st, and an encrypted database EDB. Note that Figure 2 follows the standard DSSE interface, where IDX is initialized as empty. If a full database is already available, IDX can instead be populated by document update, which is described in §5.3, for setting corresponding IDX entries during setup.

5.2 Keyword Search

Search result is the set of documents returned by the server in response to a query. In differentially private systems, the output may include documents that lack the target keyword or omit documents that do contain it. The impact of this inaccuracy can be described using two measurement metrics:

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

In most cases, the user aims for large TPR values (e.g., 0.95–0.99) and low FPR values (e.g., 0.025–0.2). We present the search protocol of FROST in Figure 3. To search for a keyword w in a database of N documents, the user computes its BF representation as column indices $(u_1, u_2, \ldots, u_k)$, and creates corresponding RePIR queries for each u_k with $k \in [K]$ (SrchTkn algorithm, lines 6–7). To execute a single-round-trip search while obfuscating search result patterns, the user first specifies which results should be true positives (TP, with high probability TPR $\approx$ 0.95–0.99), false positives (FP, with low probability FPR $\approx$ 0.025–0.2), or random non-matches (the remaining ones) (lines 1–4). Followed by creating RePIR queries for retrieving K columns, the user adds false positives and false negatives to the final output as follows. For each state vector ($\mathbf{v}_{\mathsf{m}}$ or $\mathbf{v}_{\mathsf{nm}}$) at the retrieved column k and row i, for all $k \in [K]$ and $i \in [N]$, the user sends a transformation vector (tv) $\mathbf{x}_{i,k} \in \mathbb{Z}_{p'}^{\ell}$, depending on whether document i belongs to true positives (TP), false positives (FP), or random non-matches set. For each document $i \in \mathsf{FP}$, the user computes $\mathbf{x}_{i,k}$ such that the dot products between $\mathbf{x}_{i,k}$ with match value $\mathbf{v}_m$ (i.e., value '1'), and non-match value $\mathbf{v}_{\mathsf{nm}}$ (i.e., value '0') are the same, i.e., $y_{i,k} \leftarrow \langle \mathbf{v}_{\mathsf{m}}, \mathbf{x}_{i,k} \rangle = \langle \mathbf{v}_{\mathsf{nm}}, \mathbf{x}_{i,k} \rangle \pmod{p'}$ (lines 11–13) as follows:

$$\mathbf{x}_{i,k}[j] \xleftarrow{\$} \mathbb{Z}_{p'}, \quad \text{for } j \in [\ell] \setminus \{l\},$$

$$\mathbf{x}_{i,k}[l] \leftarrow \frac{\sum_{j=1, j \neq l}^{\ell} (\mathbf{v}_{\mathsf{m}}[j] - \mathbf{v}_{\mathsf{nm}}[j]) \mathbf{x}_{i,k}[j]}{\mathbf{v}_{\mathsf{nm}}[l] - \mathbf{v}_{\mathsf{m}}[l]} \in \mathbb{Z}_{p'},$$

where $l \in [\ell]$ such that $\mathbf{v}_{\mathsf{nm}}[l] \neq \mathbf{v}_{\mathsf{m}}[l]$. The purpose is to make $y_{i,k}$ the same regardless of the actual value in the search index, which can be either $\mathbf{v}_{\mathsf{m}}$ or $\mathbf{v}_{\mathsf{nm}}$, thereby enforcing the document to appear in the search results. By contrast, for each document in TP, the user takes a random $\mathbf{x}_{i,k}$ and computes $y_{i,k} \leftarrow \langle \mathbf{v}_{\mathsf{m}}, \mathbf{x}_{i,k} \rangle \pmod{p'}$, which is the correct dot product value between match value $\mathbf{v}_{\mathsf{m}}$ and $\mathbf{x}_{i,k}$ (lines 14–15). A document in the TP set is returned in the search result if it truly contains the queried keyword, which corresponds to the value $\mathbf{v}_{\mathsf{m}}$ in the search index. For the remaining documents that do not belong to either TP or FP, the user takes as random values for $\mathbf{x}_{i,k}$ and $y_{i,k}$, such that these belong to random non-matches with a high probability (lines 16–17). To let the server know whether document i is a match or not, the user computes match value (mv) as $\mathsf{mv}[i] = \sum_{k=1}^{K} y_{i,k} \pmod{p'}$, which is the summation value of $y_{i,k}$ for K columns with $i \in [N]$ (line 18). It is used on the server to check and return the matches as the search output (Srch algorithm, lines 25–29). Note that we can get a true positive if the document is correctly selected as a true positive (with probability p), or when it is not but randomly selected as a false positive (with probability q), i.e., $\mathsf{TPR} = \mathsf{q} + (1 - \mathsf{q})\mathsf{p} = \mathsf{p} + (1 - \mathsf{p})\mathsf{q}$. The probability of false positive is $\mathsf{FPR} = \mathsf{q}$. In our scheme, the value ℓ is a security parameter, which is chosen to ensure that the probability of two duplicated transformation vectors is negligible. For example, with p' being a 24-bit prime number and $\ell = 5$, the probability of duplicates is $1/2^{24(\ell-1)} = 2^{-96}$ (since $\mathbf{x}_{i,k}[l]$ is not randomly selected for FP cases), which is negligible.

5.3 Document Update

FROST supports document update as other bitmap-based dynamic encrypted search schemes (e.g., [28, 49, 63]). We present the update procedure of FROST in Figure 4. Given an updated document with identifier f and a set of its keywords $\mathcal{W}_f$, the user increments the counter state value corresponding to document f as $\mathsf{st}'_f \leftarrow \mathsf{st}_f + 1$ (UpdtTkn algorithm, line 1). The user computes the new BF representation $\mathbf{u} \in \mathbb{Z}_{p'}^{\ell}$ of the updated document with input keywords $\mathcal{W}_f$ (lines 2–5). The user encrypts $\mathbf{u}$ with PRF key κ

and the incremented counter state value st'_f as $\mathbf{u}'[j] \leftarrow (\mathbf{u}[j] \ll z) + F(\kappa, j \| f \| \mathsf{st}'_f)$ (lines 6–7). Also, as the row corresponding to the updated document changes, the hint at row f also needs to be updated as $\mathsf{hint}'_f \leftarrow \mathbf{u}' \cdot \mathbf{A} \cdot \mathbf{B}^{-1} - \mathbf{C}'[f, *]$, where $\mathbf{C}'[f, *] \leftarrow \tilde{F}(\kappa, f \| \mathsf{st}'_f)$ (lines 8–9). Finally, the user sends the update token $\mathfrak{u}$ including the document index f, the updated encrypted row $\mathbf{u}'$, and the updated hint hint'_f to the server to update its encrypted search index and hint accordingly as $\mathsf{EIDX}[f, *] \leftarrow \mathbf{u}'$, and $\mathsf{hint}[f, *] \leftarrow \mathsf{hint}'_f$ (Updt algorithm, lines 11–12).

5.4 Optimization

We present several optimization strategies that can be applied to our FROST scheme to further improve its practicality.

Reducing computation overhead by batching user queries. Similar to PIR techniques [46, 50], we can apply "batch RePIR" to allow the user to fetch K columns at server-side cost $\ll K \cdot m \cdot N$, without increasing the hint size. The idea is to randomly partition the index of m columns into K chunks, each represented as a matrix of dimension N-by-m/K. If the K columns that the user wants to fetch fall into distinct chunks, the server can obtain these columns by running RePIR once on each index chunk. This evenly-partitioned characteristic of BF is desirable in PIR settings, where the entire index must be traversed to prevent pattern leakage. BF can be tweaked to achieve this while maintaining the same FPR at the cost of a larger BF size m (see Appendix B). However, with standard BF, more than one of the user's desired columns may fall into the same chunk. In this case, the user can make λ RePIR queries to each of the K chunks to achieve the failure probability $2^{-\Omega(\lambda)}$ [6, 50]. This optimization saves on server work as long as $\lambda < K$. However, it requires the user to send additional transformation vectors to eliminate the influence of redundant columns retrieved due to batching queries.

Optimizing TPR and FPR. 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 L shards and requiring $l < L$ shards for document recovery [20]. This incurs an extra communication, computation, and server storage cost. Likewise, false positives can be trivially filtered out by the user when checking whether or not a returned document contains the queried keyword. Thus, the effective cost of false positives is bandwidth consumption. As discussed earlier, existing SSE schemes balance privacy, performance, and utility in different ways. Schemes that leak search and result patterns often achieve high performance and utility, whereas stronger privacy solutions, such as ORAM or PIR, tend to introduce higher computation, communication, or user storage costs. We aim to design a middle-ground approach that obfuscates search result patterns with a trade-off between privacy and utility while remaining efficient in terms of computation and bandwidth costs.

5.5 Analysis

5.5.1 Complexity. We analyze the online asymptotic cost of FROST. We consider the number of bits of modulo p, p', q, the LWE parameter n, parameter ℓ, and the value of the BF parameter K as small constants. Let N be the number of documents. We implicitly include the cost for deriving the $n \times n$ matrix $\mathbf{B}$, which is $O(1)$ (as n is constant), and the cost for deriving matrix $\mathbf{C}$, which is $O(N \cdot n) = O(N)$, to the search and update cost of the user in the following analysis. To search for a keyword w, the user creates a query qu of size $O(K \cdot (m + N + n) \cdot \ell) = O(m + N)$. Also with each document, the user sends a transformation vector tv of size $O(\ell \cdot K) = O(1)$, and a match value mv of size $O(1)$ for comparison. Therefore, the overall user's computation and bandwidth cost in search is $O(m + N)$.

For keyword search, the server incurs $O(K \cdot N \cdot m \cdot \ell) = O(N \cdot m)$ modulo additions and multiplications for retrieval. The server performs $O(K \cdot N \cdot \ell) = O(N)$ multiplications for dot products, followed by $O(N)$ additions and comparisons to obtain the final matches. The overall server computation cost per search query is $O(N \cdot m + N) = O(N \cdot m)$.

To update a document, the user creates a new BF representation of size $O(m \cdot \ell) = O(m)$, followed by encrypting it, together with the hint of size $O(n \cdot \ell) = O(1)$. Thus, the total user's bandwidth cost and computation cost per document update are both $O(m)$. The server does not incur any computation other than replacing the user's database components (i.e., a row in the index and hint).

For storage, the user stores a secret key κ of size $O(\lambda)$, and matrices independent of the database that can be derived from a PRF with the secret key κ including: matrix $\mathbf{B}$ of size $O(n \cdot n \cdot \ell) = O(1)$, and matrix $\mathbf{C}$ of size $O(N \cdot n \cdot \ell) = O(N)$, and a LWE matrix $\mathbf{A}$ of size $O(m \cdot n \cdot \ell) = O(m)$. The user stores a counter state st of size $O(N)$. The server stores an index of size $O(K \cdot N \cdot m \cdot \ell) = O(N \cdot m)$, and a hint of size $O(N \cdot n \cdot \ell) = O(N)$. In summary, the total server storage cost is $O(N \cdot m)$.

5.5.2 Security Analysis. We state the differential privacy for documents and keywords of FROST, implying search result pattern and search volume privacy, also the adaptive security of FROST when the underlying RePIR is SIM-secure as follows.

Theorem 2. FROST *provides ϵ-DP for documents and keywords, where $\epsilon = \ln\left(\frac{\mathsf{TPR}}{\mathsf{FPR}} \frac{1-\mathsf{FPR}}{1-\mathsf{TPR}}\right)$, with* $\mathsf{TPR} = \mathsf{p} + (1 - \mathsf{p})\mathsf{q}$ *and* $\mathsf{FPR} = \mathsf{q}$.

Note that the resulting differential privacy parameter ϵ of FROST is identical to that of OSSE. We adapt the differential privacy proof of OSSE [81, Appendix A] to prove our Theorem 2, with details given in Appendix C.

Corollary 1. FROST *provides ϵ-DP for documents and keywords, resulting in ϵ-DP for search result pattern and search volume.*

Theorem 3. FROST *is adaptively semantically secure by Definition 9, and achieves forward privacy by Definition 10.*

We adapt the semantic security proof of OSSE [81, Section VI] to prove our Theorem 3, with details provided in Appendix D.

6 Experimental Evaluation

Implementation. We fully implemented all our proposed techniques in C++ consisting of about 1,200 lines of code. We used standard cryptographic libraries, including `OpenSSL` [1] for PRF and hash functions, `libNTL` [82] for modulo arithmetic. Our source code is available at: https://github.com/vt-asaplab/FROST.

Hardware. We used a server with 48-core Intel® Xeon®Platinum 8360Y CPU (2.40 GHz) and 512 GB RAM, running Rocky Linux 8.10.

Parameters. For LWE, we set the secret dimension $n = 2048$, use a 64-bit prime number modulus $q = 18409157466922956641$, set the error distribution χ to be the discrete Gaussian distribution with standard deviation $\sigma = 6.4$, and allow correctness error $\delta = 2^{-40}$ even with the largest BF size $m = 11088$ (see §6.1.2). Our parameter selection ensures that the best-known attack against LWE requires at least about 2^{106} ring operations to break security, according to the modern lattice-attack-cost estimator [5]. We selected the BF size m to ensure less than 1 expected false positive, which is several orders of magnitude smaller than the number of false positives by DP, where each keyword hashes to $K = 7$ BF indices (see Appendix B). Therefore, the false positives observed in our evaluation come from the DP mechanism rather than BF. We selected plaintext modulus $p' = 18483887$, ciphertext modulus $p = 80172943$, and the dimension of state and transformation vectors $\ell = 5$ as explained in §5.2.

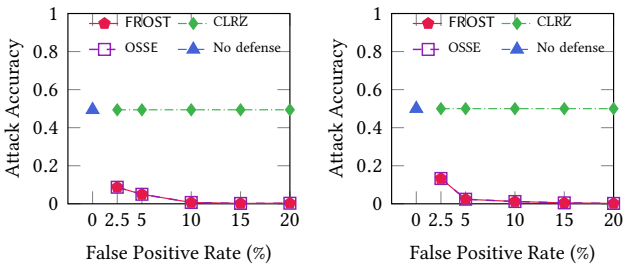
Datasets. We used publicly available datasets to evaluate the performance of FROST, as well as its security against leakage abuse attacks in comparison with other SSE schemes using DP:

- *Enron* [3]: Enron email corpus, which contains about 500K emails of 150 employees from Enron corporation. We leveraged the same standard tokenization method described in Oblix [71]. We stemmed the words and removed stopwords and words that were >20 or <4 characters long or contained non-alphabetic characters.
- *Lucene* [2]: *Java-user* mailing list from the Lucene project, used by Cash et al. [15]. We took the emails of this mailing list from September 2001 until May 2020 (around 66,400 emails). Each email is one document in the dataset, and its keyword list is the set of words in the main body of the email that are part of an English dictionary, excluding English stopwords.
- *Wiki Dataset* [4]: `enwiki-latest-pages-article` dataset. We used `WikiExtractor` [7] to extract 18,607,516 distinct text-only articles from the dataset and removed empty articles that contain no keyword. We extracted unique keywords using the standard tokenization method.

Attack-Resilience Evaluation. We evaluated the resilience of FROST with state-of-the-art statistical-based leakage abuse attacks.

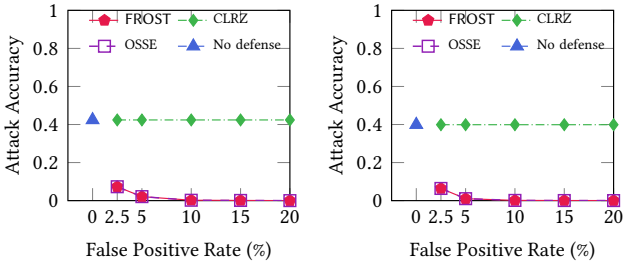
- `Freq` [68]: the frequency attack by Liu et al., which uses exclusively frequency information, and simply maps each query token to the keyword whose frequency is closest in Euclidean distance.
- `Jigsaw` [74]: `Jigsaw` is an attack by Nie et al., which works by first recovering the most distinctive queries that contain keywords with high volumes/frequencies, followed by leveraging co-occurrence information to further refine and recover the remaining queries. We used the original implementation and default parameters: $\alpha = 0.1$, $\beta = 0.9$, BaseRec = 15, ConfRec = 10, and Refspeed = 5.
- `IHOP` [76]: `IHOP` is an attack by Oya and Kerschbaum that leverages volume and frequency leakage and a heuristic approach to solve quadratic assignment problem (QAP) via many iterations. `IHOP` takes advantage of frequency and volume co-occurrence information, which refers to how many documents two queries have in common. At each iteration, it fixes some assignments at random, and solves the linear assignment problem (LAP) with the remaining free assignments. We used the default parameters $p_{\text{free}} = 0.25$ and $n_{\text{iters}} = 1000$ as recommended by the authors.

We also evaluated against other attacks, including `SAP` [75] and `GraphM` attack [78] in Appendix E.



(a) Enron **(b) Lucene**

Figure 5: Accuracy of Freq attack ($|W| = 1000$).



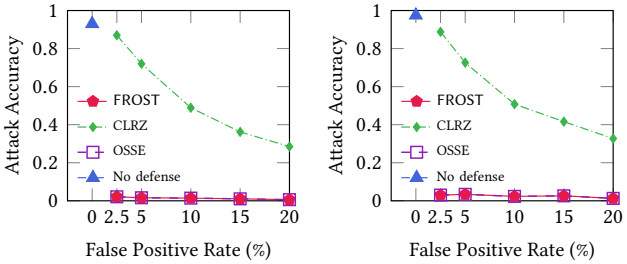
(a) $|W| = 3000$ **(b)** $|W| = 5000$

Figure 6: Accuracy of Freq attack (Wiki dataset).

Counterparts. We compared FROST with state-of-the-art DP-based SSE schemes, including CLRZ [20] and OSSE [81] with their default parameters (see our Open Science section). To evaluate the effectiveness of these defenses to leakage abuse attacks, we used a subset of 30K documents from different datasets with varied keyword set sizes $|W| \in \{1000, 3000, 5000\}$, and varied FPRs in the range 0.025–0.2, while TPR = 0.95 is kept constant for all schemes. We split the dataset into two subsets: 20K documents for building the user database, and 10K documents for the adversary to use as auxiliary information. In addition, the attacks make use of query frequency information obtained from Google Trends as auxiliary data. For each attack, we report its accuracy, defined as the proportion of queries for which the underlying keywords are correctly recovered. To obtain stable and reliable results, we averaged the accuracy of each attack over 10 executions.

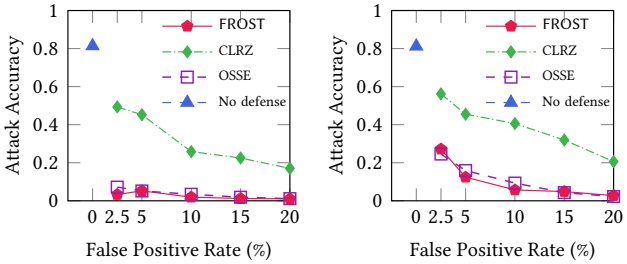
6.1 Results

6.1.1 Accuracy of Attacks. Figure 5 shows the accuracy of `Freq` attack on FROST and its counterparts over 100K queries with the same TPR 0.95 and varied FPRs increasing from 0.025 to 0.2. Figure 5a and Figure 5b are the evaluation on Enron and Lucene datasets, respectively, both with 1K keywords. Without defense (i.e., TPR = 1.0 and FPR = 0.0), the `Freq` attack achieves 49.4% accuracy on Enron, and 50.0% on Lucene. CLRZ cannot prevent this attack as it produces identical obfuscated outputs for the same keyword, thereby revealing search patterns. Consequently, the attack achieves the same accuracy when CLRZ is applied. FROST and OSSE effectively



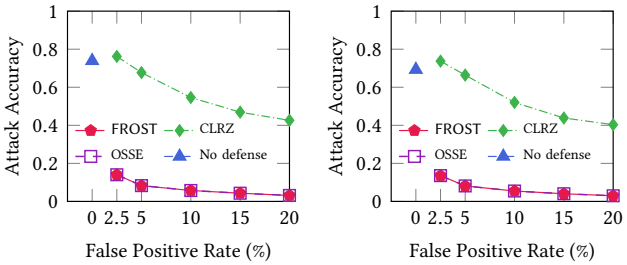
(a) Enron **(b) Lucene**

Figure 7: Accuracy of IHOP attack ($|W| = 1000$).



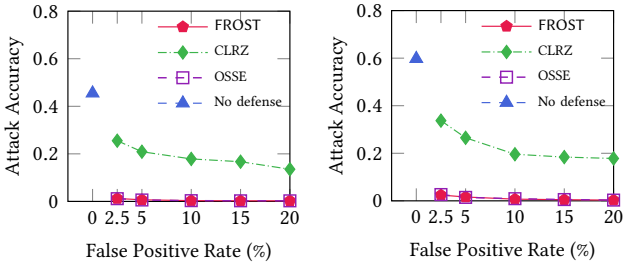
(a) Enron **(b) Lucene**

Figure 9: Accuracy of Jigsaw attack ($|W| = 1000$).



(a) $|W| = 3000$ **(b)** $|W| = 5000$

Figure 8: Accuracy of IHOP attack (Wiki dataset).



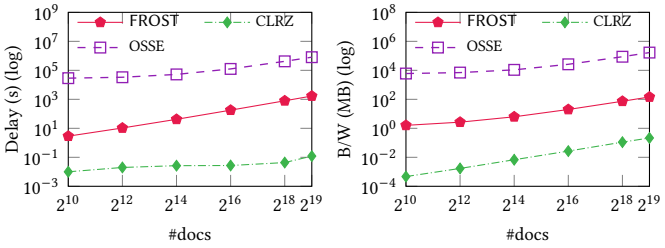
(a) $|W| = 3000$ **(b)** $|W| = 5000$

Figure 10: Accuracy of Jigsaw attack (Wiki dataset).

mitigate this attack since they output fresh obfuscation for repeated queries, where the attack accuracy is only 8.7%–0.2% on Enron, and 13.3%–0.2% on Lucene (lower is better). This demonstrates that applying clustering algorithms (e.g., k-means) to group similar result patterns does not weaken the differential privacy guarantees provided by FROST and OSSE in protecting search patterns. Figure 6 presents the attack accuracy on the Wiki dataset over 100K queries for larger database sizes, 3K and 5K keywords. Without any defense or when CLRZ is applied, the attack accuracies are 42.4% (3K keywords, Figure 6a) and 39.9% (5K keywords, Figure 6b). With FROST and OSSE, only 7.3%–0.1%, and 6.4%–0.1% of queries can be recovered correctly for 3K and 5K keywords, respectively.

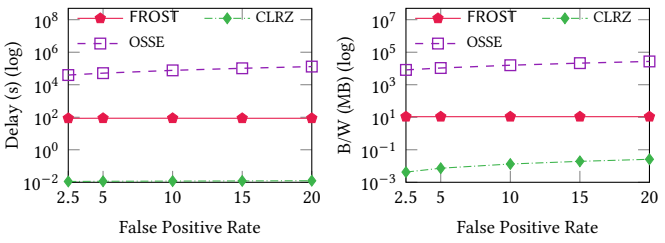
Figure 9 presents the accuracy of `Jigsaw` attack over 25K queries. Without obfuscation, the attack can recover queried keywords with accuracy 81.4% and 81.2% for Enron and Lucene, respectively. For Enron dataset (Figure 9a), the attack accuracy drops from 3.3% to 1.0% for FROST, 7.2%–1.1% for OSSE, and 49.3%–17.0% for CLRZ. For Lucene dataset (Figure 9b), the corresponding figures are 27.2%–2.7%, 24.6%–2.2%, and 56.2%–20.6% for FROST, OSSE, and CLRZ, respectively. FROST and OSSE perform very closely overall, in which differences are insignificant and not monotonic in FPR since the differential privacy parameter ϵ is the same in both works. As mentioned in [74], `Jigsaw` is good at recovering high-volume queries, which is about 10% of data per Zipfian law. It is explained in [74] that the accuracy of the attack can be negatively affected for data that does not follow Zipfian law, e.g., by padding random noise. Furthermore, the attack accuracy mostly relies on the precise recovery of distinctive queries, i.e., queries with distinguished frequency/volume. Figure 10 shows the accuracy of the `Jigsaw` attack on Wiki dataset with 3K keywords over 90K queries, and 5K keywords over 150K queries. Per Figure 10, without any defense, `Jigsaw` attack achieves accuracy 45.5% and 59.7% for 3K and 5K keywords, respectively. FROST and OSSE can significantly reduce these numbers, which decline down to 1.2%–0.3% for FROST, 1.2%–0.2% for OSSE (Figure 10a), and 2.4%–0.3% for FROST, 2.6%–0.3% for OSSE (Figure 10b), while the corresponding figures of CLRZ are 25.5%–13.5% and 33.7%–17.8%. This emphasizes the importance of protecting result patterns with fresh obfuscation to also obfuscate search patterns, which can only be achieved with FROST and OSSE, but cannot with CLRZ.

Figure 7 shows the accuracy of `IHOP` attack. Due to the high computational complexity of this attack, we tested with a smaller number of queries, which is 10K queries with 1K keywords. Without defense, the attack achieves accuracy 93.0% for Enron (Figure 7a) and 97.5% for Lucene (Figure 7b). For Enron dataset, the accuracy of the attack decreases from 2.1% down to 0.6% with FROST and OSSE, and 87.0%–28.5% with CLRZ. For Lucene dataset, the corresponding figures of the attack are 3.0%–1.3% for FROST and OSSE, and 88.8%–32.7% for CLRZ. DP-based approaches can mitigate the effectiveness of `IHOP` attack since it can obfuscate result patterns, falsifying the volume co-occurrence leakage information. Moreover, FROST and OSSE also obfuscate search patterns, reducing the impact of query frequency leakage. Figure 8 shows the accuracy of `IHOP` attack on Wiki dataset with significantly large numbers of queries and keywords, 30K queries with 3K keywords (Figure 8a), and 50K queries with 5K keywords (Figure 8b). Without defense, `IHOP` attack can precisely recover 73.9% of queries with database including 3K keywords, and 69.3% with 5K keywords. However, FROST and OSSE can reduce the attack accuracy down to 14.0%–3.1% with 3K keywords, and to 13.6%–3.0% with 5K keywords, while the corresponding figures of CLRZ are 76.2%–42.6%, and 73.7%–40.4%, respectively. We notice that under FROST and OSSE defenses, increasing the number



(a) Search delay (b) User bandwidth

Figure 11: Total search delay and bandwidth (Enron, FPR = 0.1).



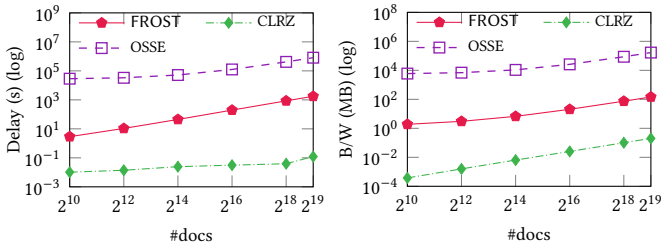
(a) Search delay (b) User bandwidth

Figure 12: Total search delay and bandwidth (Enron, 32K documents).

of queries leads to lower accuracy and requires much longer attack execution time.

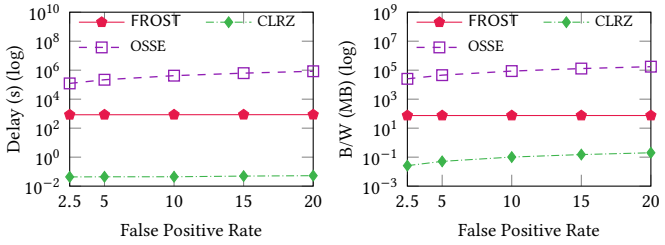
6.1.2 Keyword Search. For keyword search, we only used 8 CPU cores for evaluation. Figure 11 shows the total search delay and user bandwidth with the Enron dataset, where each email has an average of 73.2 keywords. We set TPR = 0.95 and FPR = 0.1. For the number of documents increasing from 2^{10} to 2^{19}, we selected the corresponding BF sizes $m \in \{2912, 3745, 4676, 5810, 7273, 8099\}$. Figure 11a presents the total delay to execute a keyword search, in which FROST takes 2.9s–1683.6s, OSSE takes 28798.8s–806757.0s, and CLRZ takes 10.0ms–120.8ms. FROST is 479.2×–10012.6× faster than OSSE while providing the same security guarantee. Although CLRZ is more efficient, it does not produce fresh obfuscations for repeated queries on the same keyword, thereby leaking search patterns. Figure 11b demonstrates the user bandwidth, including the query and response size for a keyword search. The communication cost of FROST increases from 1.6 MB to 144.9 MB, while that of OSSE is 5.9 GB–163.9 GB, which is 1157.9×–3747.9× larger.

Figure 12 shows the total search delay and user bandwidth for a subset of 32K documents with TPR = 0.95 and varied FPRs, increasing from 0.025 to 0.2. While FROST and CLRZ remain nearly unchanged, OSSE incurs significantly more delay and communication cost when increasing FPR. In particular, for Figure 12a, while the delay of OSSE rises sharply from 39466.2s to 130980.0s, that of FROST just slightly increases 86.1s–86.7s, and 11.2ms–12.5ms for CLRZ. Regarding communication cost (Figure 12b), OSSE incurs 8.0 GB–26.5 GB, whereas the bandwidth overhead of FROST is around 10.8 MB, and that of CLRZ is 4.3 KB–26.6 KB. The reason for the overhead of OSSE is when increasing FPR, the user must send additional search tokens, requiring extra computation for search, thereby incurring higher latency and bandwidth cost.



(a) Search delay (b) User bandwidth

Figure 13: Total search delay and bandwidth (Wiki, FPR = 0.1).



(a) Search delay (b) User bandwidth

Figure 14: Total search delay and bandwidth (Wiki, 256K documents).

We also executed the same experiments with the Wiki dataset. In particular, we excluded documents containing no keywords, and randomly sampled 500K documents. For our extracted subset, each document has an average of 100.3 keywords. For the number of documents increasing from 2^{10} to 2^{19}, we selected the corresponding BF sizes $m \in \{3983, 5124, 6398, 7966, 9954, 11088\}$. Figure 13 illustrates the total search delay and bandwidth overhead with varied numbers of documents. For Figure 13a, FROST, OSSE, and CLRZ correspondingly take 2.9s–1762.2s, 28819.8s–810216.6s, 10.2ms–121.9ms to execute a keyword search. FROST is 459.8×–9988.5× faster than OSSE and 281.9×–14456.7× slower than CLRZ. For Figure 13b, the communication costs of FROST, OSSE, and CLRZ are 1.9 MB–145.7 MB, 5.8 GB–163.9 GB, and 0.4 KB–206.4 KB, respectively. FROST incurs 1151.9×–3161.9× smaller bandwidth overhead than OSSE, while requiring an additional 1.9 MB–145.5 MB compared to CLRZ as a trade-off for protecting the search patterns. Figure 14 presents the total search delay and user bandwidth for a subset of 256K documents with varied FPRs. While FROST and CLRZ remain virtually unchanged, in which FROST takes 839.0s–854.6s, and CLRZ takes 43.2ms–52.7ms, the total search delay of OSSE increases from 122188.8s–843029.4s (Figure 14a). Also, the bandwidth overhead of OSSE rises from 24.7 GB–170.5 GB, while that of FROST and CLRZ is 74.2 MB–74.4 MB, and 26.5 KB–205.2 KB, respectively (Figure 14b).

Privacy–utility trade-offs and parameter insights. For repeated queries, FROST generates fresh obfuscation each time because TPSet and FPSet are sampled anew per query, so repeated executions of the same keyword search do not return the same noisy result patterns, which is the same as OSSE, and stronger than CLRZ. For closely related or correlated queries, the privacy-utility trade-off is more application-dependent and depends on user behavior and database characteristics in the concrete deployment. For

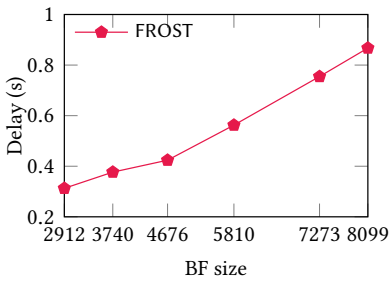


Figure 15: Total document update delay (100 keywords).

the evaluation in this section, we selected TPR and FPR to make existing attacks close to nearly random-guessing. Intuitively, with DP-based SE, larger false negative rate (i.e., lower TPR) affects utility and bandwidth, while larger FPR only increases extra returned documents and bandwidth. Unlike OSSE, FROST's query size and server computation are independent of TPR and FPR.

6.1.3 Document Update. As OSSE and CLRZ do not explicitly provide update functionality, we only evaluate the performance of document update for FROST in Figure 15. When increasing the BF size from 2912 to 8099 and keeping the same number of updated keywords as 100, it takes 0.3s–0.9s to update a document.

7 Related Work

SSE. SSE was first introduced and formalized by Song et al. [83]. SSE enables encrypted queries to be executed over encrypted data using an encrypted index, while supporting enhanced security properties like forward privacy [13, 60], backward privacy [14, 40, 86, 87], efficiency improvements [18, 29], and extended query capabilities [57], as well as support for updates [17, 45, 53, 85]. Despite these benefits, many SSE schemes are susceptible to statistical inference attacks [15, 43, 55, 76, 78, 96], which often exploit inherent leakage patterns such as search [59, 68, 75], result [51], volume [61], and file-injection [99, 100]. Xu et al. [96] demonstrated that even schemes with strong security guarantees, such as forward and backward privacy, still remain vulnerable to leakage-abuse attacks.

Multi-User SSE. While SSE primarily supports single-user access, several approaches have been developed to extend functionality to multi-user scenarios [19, 56]. For instance, Chamani et al. [19] introduced a multi-user SSE model that incorporates data and query policy enforcement, along with verifiability through blockchain technology. The scheme in [56] allows users to query encrypted data without direct interaction with the data owner, relying instead on helper users. Other solutions either assume all users are trusted [28] or involve resource-intensive cryptographic mechanisms like MPC [34, 52, 63]. The scheme in [52] employs two independent custodians running a garbled circuit protocol to create search tokens, though it still reveals typical SSE leakage such as search, volume, and result patterns. Leakage mitigation techniques [39] can reduce some exposure but are mainly suited for single-user settings due to the high overhead of index rebuilding. Applying such techniques in multi-user contexts, where readers and writers are distinct, requires writers to monitor readers' query activities to maintain security. Multi-key SSE schemes [56, 95] offer decentralized access across users, yet they do not eliminate pattern leakage.

PKSE. PKSE schemes [8, 12, 32, 97] enable support for multiple writers; however, they do not ensure forward privacy, making them susceptible to file-injection attacks [100]. While hybrid model [92] can provide forward privacy, they require each writer to be actively involved in periodically regenerating encrypted tokens. Furthermore, many PKSE approaches are prone to be vulnerable by KGA. Some PKSE schemes can prevent KGA, but they depend on the presence of a trusted third party [58, 66, 93], or trusted setup [65].

Oblivious storage systems. Oblivious storage platforms make use of ORAM and/or PIR techniques to conceal search and result patterns during data operations such as retrieval or sharing [10, 21, 22, 24, 28, 69, 70], and keyword search [30, 34, 38, 49, 64, 73]. DP-based approach [20] can obfuscate result patterns, but still leaks search patterns or comes at the cost of significant computation and communication overhead to also hide search patterns [81].

Hardware-assisted SE. Trusted hardware technologies, such as Intel SGX [25], have been employed to develop efficient oblivious platforms supporting a range of tasks, including keyword search [37, 48, 71], SQL queries [33, 36], data storage [27, 47], and oblivious memory [88]. These systems rely on specific hardware security guarantees, such as isolation, resistance to tampering, and protection against side-channel attacks.

8 Conclusion

In this paper, we proposed and designed FROST, a new SSE scheme that conceals search patterns and obfuscates result patterns to mitigate leakage-abuse query-recovery attacks while maintaining high efficiency and requiring only a single communication round for keyword search. Compared with OSSE, the state-of-the-art SSE design with DP, FROST offers a significant advance in terms of concrete performance. In addition, we introduced RePIR, a new primitive for private information retrieval on a private database that supports output rerandomization, and also a novel approach for applying DP in SSE by using state and transformation vectors.

Acknowledgment

We would like to thank the shepherd and anonymous reviewers for their insightful comments and constructive feedback on improving the quality of this work. This project was supported in part by NSF grant CNS-2350215 and the Commonwealth Cyber Initiative (CCI), an investment in the advancement of cyber R&D, innovation, and workforce development. For more information about CCI, visit www.cyberinitiative.org.

Ethics Considerations

We have carefully considered the ethics following the conference guideline. All datasets utilized in our experiments are publicly available and do not contain any harmful content. Our experiments were conducted exclusively within rigorously controlled environments, ensuring no disruption to other public systems. We believe our research was done ethically and complies with the ethics guidelines.

Open Science

Our FROST implementation is available at: https://github.com/vt-asaplab/FROST. The repository includes a file `README.md` providing detailed instructions for setup, building the project, dataset extraction, and program execution.

We evaluated the effectiveness of DP-based approaches including our FROST, CLRZ [20], and OSSE [81] against the `Freq` [68], `IHOP` [76], `SAP` [75], and `GraphM` [78] attacks using the implementation available at https://github.com/simon-oya/USENIX22-ihop-code/, and against the `Jigsaw` [74] attack using the implementation at https://github.com/JigsawAttack/JigsawAttack/. For completeness, we also included the implementation of these attacks in our repository. For evaluating keyword search performance, we reported results based on the publicly available artifacts of OSSE: https://github.com/z6shang/OSSE/, and CLRZ: https://github.com/donnod/APOSSE/. The CLRZ implementation instantiates its underlying SSE using the RR2Lev scheme from the open-source SSE library Clusion https://github.com/encryptedsystems/Clusion/.

References

[1] Openssl: Cryptography and ssl/tls toolkit. https://www.openssl.org.

[2] Apache Foundation. Mail archives of lucene. https://mail-archives.apache.org/mod_mbox/#lucene, 1999.

[3] CMU William W. Cohen, MLD. Enron email datasets. https://www.cs.cmu.edu/~enron, 2015.

[4] The wikipedia corpus. https://dumps.wikimedia.org/, 2022.

[5] Martin Albrecht, Rachel Player, and Sam Scott. On the concrete hardness of learning with errors. *Journal of Mathematical Cryptology*, 9, 10 2015.

[6] Sebastian Angel, Hao Chen, Kim Laine, and Srinath Setty. Pir with compressed queries and amortized query processing. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 962–979, 2018.

[7] Giuseppe Attardi. Wikiextractor. https://github.com/attardi/wikiextractor, 2015.

[8] Adam J. Aviv, Michael E. Locasto, Shaya Potter, and Angelos D. Keromytis. Ssares: Secure searchable automated remote email storage. In *Twenty-Third Annual Computer Security Applications Conference (ACSAC 2007)*, pages 129–139, 2007.

[9] W. Banaszczyk. Inequalities for convex bodies and polar reciprocal lattices in rn. *Discrete & computational geometry*, 13(2):217–232, 1995.

[10] Erik-Oliver Blass, Travis Mayberry, Guevara Noubir, and Kaan Onarlioglu. Toward robust hidden volumes using write-only oblivious ram. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, CCS '14, page 203–214, New York, NY, USA, 2014. Association for Computing Machinery.

[11] Burton H Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.

[12] Dan Boneh, Giovanni Di Crescenzo, Rafail Ostrovsky, and Giuseppe Persiano. Public key encryption with keyword search. Cryptology ePrint Archive, Paper 2003/195, 2003. https://eprint.iacr.org/2003/195.

[13] Raphael Bost. $\sum o\phi o\varsigma$: Forward secure searchable encryption. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, page 1143–1154, New York, NY, USA, 2016. Association for Computing Machinery.

[14] Raphael Bost, Brice Minaud, and Olga Ohrimenko. Forward and backward private searchable encryption from constrained cryptographic primitives. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 1465–1482, 2017.

[15] David Cash, Paul Grubbs, Jason Perry, and Thomas Ristenpart. Leakage-abuse attacks against searchable encryption. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, CCS '15, page 668–679, New York, NY, USA, 2015. Association for Computing Machinery.

[16] David Cash, Joseph Jaeger, Stanislaw Jarecki, Charanjit S. Jutla, Hugo Krawczyk, Marcel-Catalin Rosu, and Michael Steiner. Dynamic searchable encryption in very-large databases: Data structures and implementation. *IACR Cryptol. ePrint Arch.*, 2014:853, 2014.

[17] David Cash, Stanislaw Jarecki, Charanjit Jutla, Hugo Krawczyk, Marcel-Cătălin Roşu, and Michael Steiner. Highly-scalable searchable symmetric encryption with support for boolean queries. In *Annual cryptology conference*, pages 353–373. Springer, 2013.

[18] Javad Ghareh Chamani, Dimitrios Papadopoulos, Mohammadamin Karbasforushan, and Ioannis Demertzis. Dynamic searchable encryption with optimal search in the presence of deletions. In Kevin R. B. Butler and Kurt Thomas, editors, *31st USENIX Security Symposium, USENIX Security 2022*, pages 2425–2442. USENIX Association, 2022.

[19] Javad Ghareh Chamani, Yun Wang, Dimitrios Papadopoulos, Mingyang Zhang, and Rasool Jalili. Multi-user dynamic searchable symmetric encryption with corrupted participants. *IEEE Transactions on Dependable and Secure Computing*, 20(1):114–130, 2023.

[20] Guoxing Chen, Ten-Hwang Lai, Michael K. Reiter, and Yinqian Zhang. Differentially private access patterns for searchable symmetric encryption. In *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*, pages 810–818, 2018.

[21] Weikeng Chen, Thang Hoang, Jorge Guajardo, and Attila A. Yavuz. Titanium: A metadata-hiding file-sharing system with malicious security. Cryptology ePrint Archive, Paper 2022/051, 2022. https://eprint.iacr.org/2022/051.

[22] Weikeng Chen and Raluca Ada Popa. Metal: a metadata-hiding file-sharing system. In *NDSS Symposium 2020*, 2020.

[23] Benny Chor, Eyal Kushilevitz, Oded Goldreich, and Madhu Sudan. Private information retrieval. *J. ACM*, 45(6):965–981, November 1998.

[24] Sherman SM Chow, Katharina Fech, Russell WF Lai, and Giulio Malavolta. Multi-client oblivious ram with poly-logarithmic communication. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 160–190. Springer, 2020.

[25] Victor Costan and Srinivas Devadas. Intel sgx explained. *Cryptology ePrint Archive*, 2016.

[26] Reza Curtmola, Juan Garay, Seny Kamara, and Rafail Ostrovsky. Searchable symmetric encryption: Improved definitions and efficient constructions. In *Proceedings of the 13th ACM Conference on Computer and Communications Security*, CCS '06, page 79–88, New York, NY, USA, 2006. Association for Computing Machinery.

[27] Emma Dauterman, Vivian Fang, Ioannis Demertzis, Natacha Crooks, and Raluca Ada Popa. Snoopy: Surpassing the scalability bottleneck of oblivious storage. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*, pages 655–671, 2021.

[28] Emma Dauterman, Eric Feng, Ellen Luo, Raluca Ada Popa, and Ion Stoica. Dory: An encrypted search system with distributed trust. In *Proceedings of the 14th USENIX Conference on Operating Systems Design and Implementation*, OSDI'20, USA, 2020. USENIX Association.

[29] Ioannis Demertzis, Javad Ghareh Chamani, Dimitrios Papadopoulos, and Charalampos Papamanthou. Dynamic searchable encryption with small client storage. 01 2020.

[30] Ioannis Demertzis, Dimitrios Papadopoulos, and Charalampos Papamanthou. Searchable encryption with optimal locality: Achieving sublogarithmic read efficiency. In *Advances in Cryptology – CRYPTO 2018: 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19–23, 2018, Proceedings, Part I*, page 371–406, Berlin, Heidelberg, 2018. Springer-Verlag.

[31] Cynthia Dwork. Differential privacy: A survey of results. In *Theory and Applications of Models of Computation—TAMC*, volume 4978 of *Lecture Notes in Computer Science*, pages 1–19. Springer Verlag, April 2008.

[32] Nabeil Eltayieb, Rashad Elhabob, Alzubair Hassan, and Fagen Li. An efficient attribute-based online/offline searchable encryption and its application in cloud-based reliable smart grid. *Journal of Systems Architecture*, 98:165–172, 2019.

[33] Saba Eskandarian and Matei Zaharia. Oblidb: Oblivious query processing for secure databases. *Proceedings of the VLDB Endowment*, 13(2), 2019.

[34] Brett Hemenway Falk, Steve Lu, and Rafail Ostrovsky. Durasift: A robust, decentralized, encrypted database supporting private searches with complex policy controls. In *Proceedings of the 18th ACM Workshop on Privacy in the Electronic Society*, WPES'19, page 26–36, New York, NY, USA, 2019. Association for Computing Machinery.

[35] Ben Fisch, Arthur Lazzaretti, Zeyu Liu, and Charalampos Papamanthou. Thorpir: Single server pir via homomorphic thorp shuffles. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, CCS '24, page 1448–1462, New York, NY, USA, 2024. Association for Computing Machinery.

[36] Benny Fuhry, Jayanth Jain H. A, and Florian Kerschbaum. Encdbdb: Searchable encrypted, fast, compressed, in-memory database using enclaves. In *51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2021, Taipei, Taiwan, June 21-24, 2021*, pages 438–450. IEEE, 2021.

[37] Benny Fuhry, Raad Bahmani, Ferdinand Brasser, Florian Hahn, Florian Kerschbaum, and Ahmad-Reza Sadeghi. Hardidx: Practical and secure index with sgx. In *IFIP Annual Conference on Data and Applications Security and Privacy*, pages 386–408. Springer, 2017.

[38] Sanjam Garg, Payman Mohassel, and Charalampos Papamanthou. Tworam: Efficient oblivious ram in two rounds with applications to searchable encryption. In *Proceedings, Part III, of the 36th Annual International Cryptology Conference on Advances in Cryptology — CRYPTO 2016 - Volume 9816*, page 563–592, Berlin, Heidelberg, 2016. Springer-Verlag.

[39] Marilyn George, Seny Kamara, and Tarik Moataz. Structured encryption and dynamic leakage suppression. In Anne Canteaut and François-Xavier Standaert, editors, *Advances in Cryptology – EUROCRYPT 2021*, pages 370–396, Cham, 2021. Springer International Publishing.

[40] Javad Ghareh Chamani, Dimitrios Papadopoulos, Charalampos Papamanthou, and Rasool Jalili. New constructions for forward and backward private symmetric searchable encryption. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, page 1038–1055, New York, NY, USA, 2018. Association for Computing Machinery.

[41] Niv Gilboa and Yuval Ishai. Distributed point functions and their applications. In *EUROCRYPT*, 2014.

[42] Oded Goldreich and Rafail Ostrovsky. Software protection and simulation on oblivious rams. *J. ACM*, 43(3):431–473, May 1996.

[43] Zichen Gui, Kenneth G. Paterson, and Sikhar Patranabis. Rethinking searchable symmetric encryption. Cryptology ePrint Archive, Paper 2021/879, 2021. https://eprint.iacr.org/2021/879.

[44] Zichen Gui, Kenneth G. Paterson, and Tianxin Tang. Security analysis of MongoDB queryable encryption. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 7445–7462, Anaheim, CA, August 2023. USENIX Association.

[45] Florian Hahn and Florian Kerschbaum. Searchable encryption with secure and efficient updates. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, CCS '14, page 310–320, New York, NY, USA, 2014. Association for Computing Machinery.

[46] Alexandra Henzinger, Matthew M. Hong, Henry Corrigan-Gibbs, Sarah Meiklejohn, and Vinod Vaikuntanathan. One server for the price of two: Simple and fast Single-Server private information retrieval. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 3889–3905, Anaheim, CA, August 2023. USENIX Association.

[47] Thang Hoang, Rouzbeh Behnia, Yeongjin Jang, and Attila A Yavuz. Mose: Practical multi-user oblivious storage via secure enclaves. In *Proceedings of the Tenth ACM Conference on Data and Application Security and Privacy*, pages 17–28, 2020.

[48] Thang Hoang, Muslum Ozgur Ozmen, Yeongjin Jang, and Attila A Yavuz. Hardware-supported oram in effect: Practical oblivious search and update on very large dataset. *Proceedings on Privacy Enhancing Technologies*, 2019(1), 2019.

[49] Thang Hoang, Attila Yavuz, F. Durak, and Jorge Guajardo. A multi-server oblivious dynamic searchable encryption framework. *Journal of Computer Security*, 27:1–28, 09 2019.

[50] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Batch codes and their applications. In *Proceedings of the Thirty-Sixth Annual ACM Symposium on Theory of Computing*, STOC '04, page 262–271, New York, NY, USA, 2004. Association for Computing Machinery.

[51] Mohammad Saiful Islam, Mehmet Kuzu, and Murat Kantarcioglu. Access pattern disclosure on searchable encryption: Ramification, attack and mitigation. In *NDSS*, 2012.

[52] Seny Kamara, Tarik Moataz, Andrew Park, and Lucy Qin. A decentralized and encrypted national gun registry. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 1520–1537, 2021.

[53] Seny Kamara and Charalampos Papamanthou. Parallel and dynamic searchable symmetric encryption. In *International conference on financial cryptography and data security*, pages 258–274. Springer, 2013.

[54] Seny Kamara, Charalampos Papamanthou, and Tom Roeder. Dynamic searchable symmetric encryption. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 965–976, 2012.

[55] Georgios Kellaris, George Kollios, Kobbi Nissim, and Adam O'Neill. Generic attacks on secure outsourced databases. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, page 1329–1340, New York, NY, USA, 2016. Association for Computing Machinery.

[56] Shabnam Kasra Kermanshahi, Joseph K. Liu, Ron Steinfeld, Surya Nepal, Shangqi Lai, Randolph Loh, and Cong Zuo. Multi-client cloud-based symmetric searchable encryption. *IEEE Transactions on Dependable and Secure Computing*, 18(5):2419–2437, 2021.

[57] Florian Kerschbaum and Anselme Tueno. An efficiently searchable encrypted data structure for range queries. In *European Symposium on Research in Computer Security*, 2017.

[58] Aggelos Kiayias, Ozgur Oksuz, Alexander Russell, Qiang Tang, and Bing Wang. Efficient encrypted keyword search for multi-user data sharing. In *European symposium on research in computer security*, pages 173–195. Springer, 2016.

[59] Evgenios M. Kornaropoulos, Charalampos Papamanthou, and Roberto Tamassia. The state of the uniform: Attacks on encrypted databases beyond the uniform query distribution. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 1223–1240, 2020.

[60] Russell W. F. Lai and Sherman S. M. Chow. Forward-secure searchable encryption on labeled bipartite graphs. In *International Conference on Applied Cryptography and Network Security*, 2017.

[61] Steven Lambregts, Huanhuan Chen, Jianting Ning, and Kaitai Liang. Val: Volume and access pattern leakage-abuse attack with leaked documents. In *European Symposium on Research in Computer Security*, pages 653–676. Springer, 2022.

[62] Arthur Lazzaretti and Charalampos Papamanthou. Single pass Client-Preprocessing private information retrieval. In *33rd USENIX Security Symposium (USENIX Security 24)*, pages 5967–5984, Philadelphia, PA, August 2024. USENIX Association.

[63] Tung Le, Rouzbeh Behnia, Jorge Guajardo, and Thang Hoang. MUSES: Efficient Multi-user Searchable Encrypted Database. In *33rd USENIX Security Symposium (USENIX Security 24)*, Philadelphia, PA, August 2024. USENIX Association.

[64] Tung Le and Thang Hoang. MAPLE: A Metadata-Hiding Policy-Controllable Encrypted Search Platform with Minimal Trust. In *Proceedings on Privacy Enhancing Technologies Symposium (PETS)*, pages 184–203, 2023.

[65] Tung Le and Thang Hoang. Hermes: Efficient and Secure Multi-Writer Encrypted Database . In *2025 IEEE Symposium on Security and Privacy (SP)*, pages 2865–2884, Los Alamitos, CA, USA, May 2025. IEEE Computer Society.

[66] Hongbo Li, Qiong Huang, Jianye Huang, and Willy Susilo. Public-key authenticated encryption with keyword search supporting constant trapdoor generation and fast search. *IEEE Transactions on Information Forensics and Security*, 18:396–410, 2023.

[67] Richard Lindner and Chris Peikert. Better key sizes (and attacks) for lwe-based encryption. In Aggelos Kiayias, editor, *Topics in Cryptology – CT-RSA 2011*, pages 319–339, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[68] Chang Liu, Liehuang Zhu, Mingzhong Wang, and Yu-An Tan. Search pattern leakage in searchable encryption: Attacks and new construction. *Inf. Sci.*, 265:176–188, may 2014.

[69] Jacob R Lorch, Bryan Parno, James Mickens, Mariana Raykova, and Joshua Schiffman. Shroud: Ensuring private access to {Large-Scale} data in the data center. In *11th USENIX Conference on File and Storage Technologies (FAST 13)*, pages 199–213, 2013.

[70] Travis Mayberry, Erik-Oliver Blass, and Agnes Hui Chan. Efficient private file retrieval by combining oram and pir. *NDSS 2013*, 2013.

[71] Pratyush Mishra, Rishabh Poddar, Jerry Chen, Alessandro Chiesa, and Raluca Ada Popa. Oblix: An efficient oblivious search index. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 279–296. IEEE, 2018.

[72] Priyanka Mondal, Javad Ghareh Chamani, Ioannis Demertzis, and Dimitrios Papadopoulos. I/O-Efficient dynamic searchable encryption meets forward & backward privacy. In *33rd USENIX Security Symposium (USENIX Security 24)*, pages 2527–2544, Philadelphia, PA, August 2024. USENIX Association.

[73] Muhammad Naveed. The fallacy of composition of oblivious ram and searchable encryption. Cryptology ePrint Archive, Paper 2015/668, 2015. https://eprint.iacr.org/2015/668.

[74] Hao Nie, Wei Wang, Peng Xu, Xianglong Zhang, Laurence T. Yang, and Kaitai Liang. Query recovery from easy to hard: Jigsaw attack against SSE. In *33rd USENIX Security Symposium (USENIX Security 24)*, pages 2599–2616, Philadelphia, PA, August 2024. USENIX Association.

[75] Simon Oya and Florian Kerschbaum. Hiding the access pattern is not enough: Exploiting search pattern leakage in searchable encryption. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 127–142. USENIX Association, August 2021.

[76] Simon Oya and Florian Kerschbaum. IHOP: Improved statistical query recovery against searchable symmetric encryption through quadratic optimization. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 2407–2424, Boston, MA, August 2022. USENIX Association.

[77] Chris Peikert, Vinod Vaikuntanathan, and Brent Waters. A framework for efficient and composable oblivious transfer. In David Wagner, editor, *Advances in Cryptology – CRYPTO 2008*, pages 554–571, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[78] David Pouliot and Charles 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*, CCS '16, page 1341–1352, New York, NY, USA, 2016. Association for Computing Machinery.

[79] Daniel Pöllman and Tianxin Tang. Differentially private access in encrypted search: Achieving privacy at a small cost? Cryptology ePrint Archive, Paper 2025/1636, 2025.

[80] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. *J. ACM*, 56(6), September 2009.

[81] Zhiwei Shang, Simon Oya, Andreas Peter, and Florian Kerschbaum. Obfuscated access and search patterns in searchable encryption. In *28th Annual Network and Distributed System Security Symposium, NDSS 2021*, 2021.

[82] Victor Shoup et al. Ntl: A library for doing number theory, 2001.

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

[84] Emil Stefanov, Marten Van Dijk, Elaine Shi, T.-H. Hubert Chan, Christopher Fletcher, Ling Ren, Xiangyao Yu, and Srinivas Devadas. Path oram: An extremely simple oblivious ram protocol. *J. ACM*, 65(4), apr 2018.

[85] Emil Stefanov, Charalampos Papamanthou, and Elaine Shi. Practical dynamic searchable encryption with small leakage. Cryptology ePrint Archive, Paper 2013/832, 2013. https://eprint.iacr.org/2013/832.

[86] Shi-Feng Sun, Ron Steinfeld, Shangqi Lai, Xingliang Yuan, Amin Sakzad, Joseph K Liu, Surya Nepal, and Dawu Gu. Practical non-interactive searchable encryption with forward and backward privacy. In *NDSS*, 2021.
[87] Shi-Feng Sun, Xingliang Yuan, Joseph K Liu, Ron Steinfeld, Amin Sakzad, Viet Vo, and Surya Nepal. Practical backward-secure searchable encryption from symmetric puncturable encryption. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 763–780, 2018.
[88] Afonso Tinoco, Sixiang Gao, and Elaine Shi. Enigmap: Signal should use oblivious algorithms for private contact discovery. *Cryptology ePrint Archive*, 2022.
[89] Raphael Toledo, George Danezis, and Ian Goldberg. Lower-cost epsilon-private information retrieval. *Proceedings on Privacy Enhancing Technologies*, 2016, 04 2016.
[90] Adithya Vadapalli, Ryan Henry, and Ian Goldberg. Duoram: A Bandwidth-Efficient distributed ORAM for 2- and 3-party computation. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 3907–3924, Anaheim, CA, August 2023. USENIX Association.
[91] Sameer Wagh, Paul Cuff, and Prateek Mittal. Root oram: A tunable differentially private oblivious ram. 01 2016.
[92] Jiafan Wang and Sherman S. M. Chow. Omnes pro uno: Practical Multi-Writer encrypted database. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 2371–2388, Boston, MA, August 2022. USENIX Association.
[93] Mingyue Wang, Yinbin Miao, Yu Guo, Hejiao Huang, Cong Wang, and Xiaohua Jia. Aesm2 attribute-based encrypted search for multi-owner and multi-user distributed systems. *IEEE Transactions on Parallel and Distributed Systems*, 34(1):92–107, 2023.
[94] Wei Wang, Dongli Liu, Peng Xu, Laurence Tianruo Yang, and Kaitai Liang. Keyword search shareable encryption for fast and secure data replication. *IEEE Transactions on Information Forensics and Security*, 18:5537–5552, 2023.
[95] Yun Wang and Dimitrios Papadopoulos. Multi-user collusion-resistant searchable encryption with optimal search time. In *Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security*, ASIA CCS '21, page 252–264, New York, NY, USA, 2021. Association for Computing Machinery.
[96] Lei Xu, Leqian Zheng, Chengzhi Xu, Xingliang Yuan, and Cong Wang. Leakage-abuse attacks against forward and backward private searchable symmetric encryption. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, CCS '23, page 3003–3017, New York, NY, USA, 2023. Association for Computing Machinery.
[97] Peng Xu, Qianhong Wu, Wei Wang, Willy Susilo, Josep Domingo-Ferrer, and Hai Jin. Generating searchable public-key ciphertexts with hidden structures for fast keyword search. *IEEE Transactions on Information Forensics and Security*, 10(9):1993–2006, 2015.
[98] Mikhail Zaslavskiy, Francis Bach, and Jean-Philippe Vert. A path following algorithm for the graph matching problem. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(12):2227–2242, 2009.
[99] Xianglong Zhang, Wei Wang, Peng Xu, Laurence T. Yang, and Kaitai Liang. High recovery with fewer injections: Practical binary volumetric injection attacks against dynamic searchable encryption. In *32nd USENIX Security Symposium (USENIX Security 23)*, 2023.
[100] Yupeng Zhang, Jonathan Katz, and Charalampos Papamanthou. All your queries are belong to us: The power of File-Injection attacks on searchable encryption. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 707–720, Austin, TX, August 2016. USENIX Association.

A Proof of RePIR (Theorem 1)

We prove correctness and security of our RePIR as follows.

Correctness. Consider the user that interacts with the server to query for the data at index $v \in [m]$. Let $\mathbf{u}$ denote the unit vector v in $\hat{\mathbb{Z}}_q^m$ (i.e., the vector of all zeros, with a single '1' at index v). The Answer routine in RePIR computes:

$$\begin{aligned}\hat{\mathbf{d}} &= \lceil \mathsf{EIDX} \cdot \mathbf{q} - \mathsf{hint} \cdot \mathbf{s}' \rfloor_\Delta / \Delta \\ &= \lceil \mathsf{EIDX} \cdot (\mathbf{A} \cdot \mathbf{s} + \mathbf{e} + \Delta \cdot \mathbf{u}) \\ &\quad - (\mathsf{EIDX} \cdot \mathbf{A} \cdot \mathbf{B}^{-1}) \cdot \mathbf{B} \cdot \mathbf{s} + \mathbf{C} \cdot \mathbf{s}' \rfloor_\Delta / \Delta \\ &= \lceil \mathsf{EIDX} \cdot \Delta \cdot \mathbf{u} + \mathbf{C} \cdot \mathbf{s}' + \mathsf{EIDX} \cdot \mathbf{e} \rfloor_\Delta / \Delta \\ &= \mathsf{EIDX}[*, v] + \lfloor \mathbf{C} \cdot \mathbf{s}' \rfloor_\Delta / \Delta + \mathbf{e}' \pmod p,\end{aligned}$$

with $\mathbf{e}' \in \{0,1\}^N$, and for $i \in [N]$:

$$\begin{aligned}\mathbf{d}[i] &= \left(\hat{\mathbf{d}}[i] + (\mathbf{r}[i] \ll z) - \lfloor \mathbf{C} \cdot \mathbf{s}' \rfloor_\Delta / \Delta[i] - F(\kappa, v \| i \| \mathsf{cnt}_i)\right) \gg z \\ &= \Big((\mathsf{IDX}[i, v] \ll z) + \lfloor \mathbf{C} \cdot \mathbf{s}' \rfloor_\Delta / \Delta[i] + \mathbf{e}' \\ &\quad + (\mathbf{r}[i] \ll z) - \lfloor \mathbf{C} \cdot \mathbf{s}' \rfloor_\Delta / \Delta[i]\Big) \gg z \\ &= \mathsf{IDX}[i, v] + \mathbf{r}[i] \pmod{p'},\end{aligned}$$

where $z = 1$, and $\mathsf{EIDX}[*, v]$ denotes column v of the encrypted database EIDX, and $\mathsf{IDX}[i, v]$ denotes the element at (i, v) in database IDX. Recovery succeeds if and only if $\langle \mathsf{EIDX}[i, *] \cdot \mathbf{e} \rangle \le \Delta$ and $p' \cdot 2^{z+1} < p$.

Proof of Correctness. We can guarantee successful retrieval by ensuring that $\lfloor p/2 \rfloor \cdot m \cdot |\mathbf{e}|_\infty \le \Delta$ (because we can store the encrypted database entries, which are elements in $\mathbb{Z}_p$, as values in the range $\{-\lfloor p/2 \rfloor, -\lfloor p/2 \rfloor + 1, \ldots, \lceil p/2 \rceil - 1\}$, so that they have maximal norm $\lfloor p/2 \rfloor$). However, we instead use a tighter bound on $\langle \mathsf{EIDX}[i, *] \cdot \mathbf{e} \rangle$ by relying on properties of the error distribution, χ. As a result, we will obtain a scheme which retrieval succeeds with probability $1 - \delta$ (rather than 1) with a larger plaintext modulus, p.

Indeed, as χ is the discrete Gaussian distribution with variance $\sigma^2 = \frac{s^2}{2\pi}$ for some $s > 0$, by [67, Lemma 2.2] [9, Lemma 2.4], for any $\mathrm{T} > 0$, it holds that,

$$\Pr\left[\langle \mathsf{EIDX}[i, *] \cdot \mathbf{e} \rangle \ge T \cdot s \| \mathsf{EIDX}[i, *] \|\right] < 2 \exp(-\pi \cdot T^2),$$

where $\|.\|$ denotes the Euclidean norm.

Taking $T = \Delta / (s \cdot \|\mathsf{EIDX}[i, *]\|)$, we see that

$$\Pr\left[\langle \mathsf{EIDX}[i, *] \cdot \mathbf{e} \rangle \ge \Delta\right] < \delta,$$

as long as

$$2 \exp\left(-\pi \cdot \left(\frac{\Delta}{s \cdot \|\mathsf{EIDX}[i, *]\|}\right)^2\right) \le \delta.$$

Equivalently, recovery fails with probability at most δ, if

$$\Delta \ge s \cdot \|\mathsf{EIDX}[i, *]\| \cdot \sqrt{\frac{\ln(2/\delta)}{\pi}}.$$

Again, as we can store the elements in EIDX in the range $[-\lfloor p/2 \rfloor, \lfloor p/2 \rfloor]$, we have that

$$\|\mathsf{EIDX}[i, *]\| \le \sqrt{m \cdot \lfloor p/2 \rfloor^2} = \sqrt{m} \cdot \lfloor p/2 \rfloor.$$

In addition, we know that $s = \sigma \cdot \sqrt{2\pi}$. Thus, recovery fails with probability at most δ, as long as:

$$\lfloor q/p \rfloor \ge \sigma \cdot \sqrt{2\pi} \cdot \sqrt{m} \cdot p/2 \cdot \sqrt{\frac{\ln(2/\delta)}{\pi}},$$

or, equivalently,

$$\lfloor q/p \rfloor \ge \sqrt{2}/2 \cdot \sigma \cdot p \cdot \sqrt{m \cdot \ln(2/\delta)},$$

By (3), this condition holds. Thus, RePIR is correct, except with error probability at most δ. □

Security. Security of RePIR follows from the fact that the LWE problem is hard, even when the LWE matrix $\mathbf{A}$ is reused across many independent trials, each using an independently generated LWE secret $\mathbf{s} \xleftarrow{\$} \hat{\mathbb{Z}}_q^n$, as shown in prior work [77, Lemma 7.3]. We connect the security of RePIR to the hardness of LWE with the following lemma:

Lemma 1. *Let $N \times m$ be the database size, (n, q, χ) be the LWE parameters, $\mathbf{A} \in \hat{\mathbb{Z}}_q^{m \times n}$ be the random LWE matrix, and $\mathbf{B} \in \hat{\mathbb{Z}}_q^{n \times n}, \mathbf{C} \in \hat{\mathbb{Z}}_q^{N \times n}$ be the secret random matrices used in* RePIR*, in which $\mathbf{B}$ has linearly independent rows. For any $i \in [m]$, we define the distribution*

$$\mathcal{Q}_i = \{(\mathbf{A}, \mathsf{qu}_i) : (\mathsf{qu}_i, \_) \leftarrow \mathsf{Query}(i, \kappa, \mathsf{st})\}.$$

If the (n, q, χ)-LWE problem with m samples is $(T, \tilde{\epsilon})$-hard, then any algorithm running in time $T - O(m)$ can have success probability at most $\tilde{\epsilon}$ in distinguishing $\mathcal{Q}_i$ from the distribution $\{(\mathbf{A}, \mathbf{r}_1, \mathbf{r}_2, \mathbf{r}_3) : \mathbf{r}_1 \xleftarrow{\$} \hat{\mathbb{Z}}_q^m, \mathbf{r}_2 \xleftarrow{\$} \hat{\mathbb{Z}}_q^n, \mathbf{r}_3 \xleftarrow{\$} \hat{\mathbb{Z}}_q^N\}$.

Proof. Consider any index $i \in [m]$ and let $\mathbf{u}_i$ denote unit vector i in $\hat{\mathbb{Z}}_q^m$. Additionally, we define the following distributions:

- $\mathcal{D}_1 = \{(\mathbf{A}, \mathbf{A} \cdot \mathbf{s} + \mathbf{e}, \mathbf{B} \cdot \mathbf{s}, \mathbf{m}) : \mathbf{s} \xleftarrow{\$} \hat{\mathbb{Z}}_q^n, \mathbf{e} \xleftarrow{\$} \hat{\chi}^m, \mathbf{m} \xleftarrow{\$} \hat{\mathbb{Z}}_q^N\}$, and
- $\mathcal{D}_2 = \{(\mathbf{A}, \mathbf{r}_1, \mathbf{r}_2, \mathbf{r}_3) : \mathbf{r}_1 \xleftarrow{\$} \hat{\mathbb{Z}}_q^m, \mathbf{r}_2 \xleftarrow{\$} \hat{\mathbb{Z}}_q^n, \mathbf{r}_3 \xleftarrow{\$} \hat{\mathbb{Z}}_q^N\}$.

Now, consider the simulator $\mathcal{S}$ that, given as input $(\mathbf{A}, \mathbf{v}, \mathbf{s}', \mathbf{m}) \in \hat{\mathbb{Z}}_q^m \times \hat{\mathbb{Z}}_q^n \times \hat{\mathbb{Z}}_q^N$, computes and outputs $(\mathbf{A}, \mathbf{v} + \lfloor q/p \rfloor \cdot \mathbf{u}_i, \mathbf{s}', \mathbf{m})$. We observe that:

- When $(\mathbf{A}, \mathbf{v}, \mathbf{s}', \mathbf{m})$ is sampled from $\mathcal{D}_1$, the simulator's output is distributed identically to $\mathcal{Q}_i$.
- When $(\mathbf{A}, \mathbf{v}, \mathbf{s}', \mathbf{m})$ is sampled from $\mathcal{D}_2$, the simulator's output is distributed identically to $\mathcal{D}_2$.

As the (n, q, χ)-LWE problem with m samples is $(T, \tilde{\epsilon})$-hard, we know that any algorithm running in time T has advantage at most $\tilde{\epsilon}$ in distinguishing between $\mathcal{D}_1$ and $\mathcal{D}_2$. Our simulator $\mathcal{S}$ runs in time $O(m)$. Thus, it must hold that any algorithm distinguishing between $\mathcal{Q}_i$ and $\mathcal{D}_2$ in time at most $T - O(m)$ can have success probability at most $\tilde{\epsilon}$. □

Lemma 1 shows that any algorithm running in time $T - O(m)$ can distinguish the queries made by the user in RePIR from random vectors in $\hat{\mathbb{Z}}_q^m$ with success probability at most $\tilde{\epsilon}$. Therefore, any algorithm running in time $T - O(m)$ can distinguish the queries made by the user in RePIR to any index $i \in [m]$ with success probability at most $\tilde{\epsilon}$. In other words, RePIR is $(T - O(m), \tilde{\epsilon})$-secure.

We can extend the security definition in §4.1 to handle Q queries by requiring that any two sequences of Q queries produce indistinguishable query distributions. A hybrid argument proves the following corollary:

Corollary 1 (Q-query security of RePIR). *For any sequence $I \in [m]^Q$ of Q indices, we define $\mathcal{D}_I$ to be the query distribution that it induces, i.e.,*

$$\mathcal{D}_I = \{(\mathbf{A}, \mathsf{qu}_k) : (\mathsf{qu}_k, \_) \leftarrow \mathsf{Query}(I_k, \kappa, \mathsf{st})\}_{k \in [Q]}$$

Also, we define the random query distribution, $\mathcal{R}$:

$$\mathcal{R} = \left\{(\mathbf{A}, \mathbf{r}_1, \mathbf{r}_2, \mathbf{r}_3) : \mathbf{r}_1 \xleftarrow{\$} \hat{\mathbb{Z}}_q^m, \mathbf{r}_2 \xleftarrow{\$} \hat{\mathbb{Z}}_q^n, \mathbf{r}_3 \xleftarrow{\$} \hat{\mathbb{Z}}_q^N\right\}_{k \in [Q]}$$

If the (n, q, χ)-LWE problem with m samples is $(T, \tilde{\epsilon})$-hard, then $\mathcal{D}_I$ is $(T - O(Qnm), Q\tilde{\epsilon})$-indistinguishable from $\mathcal{R}$.

Proof. The proof follows by a standard hybrid argument. Let $I \in [m]^Q$ be a sequence of Q indices. For any $j \in [Q]$, we define the hybrid distribution, H_j:

$$H_j = \left\{(\mathbf{A}, \mathbf{r}_1, \mathbf{r}_2, \mathbf{r}_3) : \mathbf{r}_1 \xleftarrow{\$} \hat{\mathbb{Z}}_q^m, \mathbf{r}_2 \xleftarrow{\$} \hat{\mathbb{Z}}_q^n, \mathbf{r}_3 \xleftarrow{\$} \hat{\mathbb{Z}}_q^N\right\}_{k \in \{1, \ldots, j\}}$$
$$\cup \{(\mathbf{A}, \mathsf{qu}_k) : (\mathsf{qu}_k, \_) \leftarrow \mathsf{Query}(I_k, \kappa, \mathsf{st})\}_{k \in \{j+1, \ldots, Q\}}.$$

In other words, H_j is the distribution where the first j queries are truly random vectors, and the remaining $(Q - j)$ queries are sampled from $\mathcal{D}_I$. As such, $H_0 = \mathcal{D}_I$, while $H_Q = \mathcal{R}$.

Now, consider any algorithm $\mathcal{A}$ that runs in time t and distinguishes between distributions H_0 and H_Q with advantage at least p. For any $j \in [Q]$, we define p_j to be the probability that $\mathcal{A}$ outputs 1 when given samples from H_j. By definition,

$$p = |p_Q - p_0| = \left| \sum_{j=1}^{Q} (p_j - p_{j-1}) \right|.$$

Thus, there exists some $j^* \in [Q]$ such that

$$|p_{j^*} - p_{j^*-1}| \geq p/Q.$$

Using $\mathcal{A}$ as a subroutine, we then build an algorithm $\mathcal{B}$ that distinguishes between the distributions $\mathcal{Q}_{I_{j^*}}$ (defined in Lemma 1) and $\left\{(\mathbf{A}, \mathbf{r}_1, \mathbf{r}_2, \mathbf{r}_3) : \mathbf{r}_1 \xleftarrow{\$} \hat{\mathbb{Z}}_q^m, \mathbf{r}_2 \xleftarrow{\$} \hat{\mathbb{Z}}_q^n, \mathbf{r}_3 \xleftarrow{\$} \hat{\mathbb{Z}}_q^N\right\}$.

$\mathcal{B}(\mathbf{A}, \mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_3)$:

1: $\kappa \xleftarrow{\$} \{0,1\}^\lambda$; $\mathsf{st} \leftarrow (\mathsf{st}_1, \mathsf{st}_2, \ldots, \mathsf{st}_N)$, where $\mathsf{st}_i \leftarrow 0$ for $i \in [N]$
2: **for** $j = 1$ to $j^* - 1$ **do**
3: Compute $\mathbf{r}_{j,1} \xleftarrow{\$} \hat{\mathbb{Z}}_q^m, \mathbf{r}_{j,2} \xleftarrow{\$} \hat{\mathbb{Z}}_q^n, \mathbf{r}_{j,3} \xleftarrow{\$} \hat{\mathbb{Z}}_q^N$
4: **for** $j = j^* + 1, \ldots, Q$ **do**
5: Compute $(\mathsf{qu}_j, \_) \leftarrow \mathsf{Query}(I_j, \kappa, \mathsf{st})$
6: Output
$\mathcal{A}\big((\mathbf{A}, \mathbf{r}_{1,1}, \mathbf{r}_{1,2}, \mathbf{r}_{1,3}), \ldots, (\mathbf{A}, \mathbf{r}_{j^*-1,1}, \mathbf{r}_{j^*-1,2}, \mathbf{r}_{j^*-1,3}), (\mathbf{A}, \mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_3),$
$(\mathbf{A}, \mathsf{qu}_{j^*+1}), \ldots, (\mathbf{A}, \mathsf{qu}_Q)\big)$

Then, we observe that:

- When $\mathcal{B}$ is given an input sampled from $\mathcal{Q}_{I_{j^*}}$, the input that $\mathcal{B}$ feeds to $\mathcal{A}$ is distributed following H_{j^*}. As such, $\mathcal{B}$ outputs 1 with probability p_{j^*}.
- When $\mathcal{B}$ is given an input sampled from $\{(\mathbf{A}, \mathbf{r}_1, \mathbf{r}_2, \mathbf{r}_3) : \mathbf{r}_1 \xleftarrow{\$} \hat{\mathbb{Z}}_q^m, \mathbf{r}_2 \xleftarrow{\$} \hat{\mathbb{Z}}_q^n, \mathbf{r}_3 \xleftarrow{\$} \hat{\mathbb{Z}}_q^N\}$, the input that $\mathcal{B}$ feeds to $\mathcal{A}$ is distributed following H_{j^*+1}. As such, $\mathcal{B}$ outputs 1 with probability p_{j^*+1}.

So, $\mathcal{B}$ distinguishes between the distributions $\mathcal{Q}_{I_{j^*}}$ and $\{(\mathbf{A}, \mathbf{r}_1, \mathbf{r}_2, \mathbf{r}_3) : \mathbf{r}_1 \xleftarrow{\$} \hat{\mathbb{Z}}_q^m, \mathbf{r}_2 \xleftarrow{\$} \hat{\mathbb{Z}}_q^n, \mathbf{r}_3 \xleftarrow{\$} \hat{\mathbb{Z}}_q^N\}$ with advantage at least $|p_{j^*} - p_{j^*-1}| \geq p/Q$. Further, $\mathcal{B}$ runs in time $t + O(Qn(m+N))$, as Query takes time $O(n(m+N))$.

By Lemma 1, we conclude that, if the (n, q, χ)-LWE problem with m samples is $(T, \tilde{\epsilon})$-hard and $t + O(Qn(m+N)) \leq T - O(m)$, it must hold that $p/Q < \tilde{\epsilon}$. In other words, if the (n, q, χ)-LWE problem with m samples is $(T, \tilde{\epsilon})$-hard, then $\mathcal{D}_I$ is $(O(T - Qn(m+N)), Q\tilde{\epsilon})$-indistinguishable from $\mathcal{R}$. □

B Evenly-Partitioned Bloom Filter

In this section, we present a variant of the Bloom filter (BF) in which each of the hash functions $H_1, H_2, \ldots, H_k$ maps to a unique partition among k predefined partitions, rather than mapping randomly as in standard BF. For H_i, the valid positions to hash into are those not belonging to partitions selected by any previous H_j with $j < i$. We compute the FPR of the evenly-partitioned BF as follows. Let k be the number of hash functions and each has no significant correlation between each other, then the probability that the bit is not set to 1 by any of the hash functions is:

$$\prod_{i=1}^{k}\left(1-\frac{1}{(k+1-i)\frac{m}{k}}\right)$$

If we have inserted n elements, the probability that a certain bit is still 0 is:

$$\prod_{i=1}^{k}\left(1-\frac{1}{(k+1-i)\frac{m}{k}}\right)^{n}$$

Now test membership of an element that is not in the set. Each of the k array positions computed by the hash functions is 1 with a probability as above. The probability of all of them being 1, which would cause the algorithm to erroneously claim that the element is in the set, is given as:

$$\epsilon=\left(1-\prod_{i=1}^{k}\left(1-\frac{1}{(k+1-i)\frac{m}{k}}\right)^{n}\right)^{k}$$

Compared with the FPR formula of standard BF, $\epsilon = (1-(1-\frac{1}{m})^{kn})^k$, we need to increase BF size m to maintain the equivalent FPR. For example, with $k = 7$ and $m = 8080$, our evenly-partitioned BF can achieve the FPR equivalent to standard BF with the same k and $m = 3120$, which is approximate $1.8e^{-6}$.

C Differential Privacy Analysis (Theorem 2)

We prove the differential privacy guarantees of FROST stated in Theorem 2 by adapting the differential privacy analysis of OSSE [81, Appendix A] as follows. We begin by introducing the following results on ratios of probability distributions.

Let $A \sim \mathbf{Bern}(\mathrm{p})$ and $B \sim \mathbf{Bern}(\mathrm{q})$ be two Bernoulli random variables. We have the following result:

Lemma 2. *For $\alpha \in \{0, 1\}$:*

$$\frac{\Pr(A+B=\alpha)}{\Pr(B=\alpha)} \le \frac{\mathrm{q}+\mathrm{p}(1-\mathrm{q})}{\mathrm{q}}, \tag{4}$$

$$\frac{\Pr(B=\alpha)}{\Pr(A+B=\alpha)} \le \frac{1}{1-\mathrm{p}}. \tag{5}$$

Proof. The ratio takes the following values depending on α:

$$\frac{\Pr(A+B=\alpha)}{\Pr(B=\alpha)}=\begin{cases}\dfrac{(1-\mathrm{p})(1-\mathrm{q})}{(1-\mathrm{q})}=1-\mathrm{p}, & \text{if } \alpha=0;\\ \dfrac{\mathrm{p}+(1-\mathrm{p})\mathrm{q}}{\mathrm{q}}=\dfrac{\mathrm{q}+\mathrm{p}(1-\mathrm{q})}{\mathrm{q}}, & \text{otherwise.}\end{cases}$$

The bounds in (4) and (5) follow from the fact that $1-\mathrm{p} \le \frac{\mathrm{q}+\mathrm{p}(1-\mathrm{q})}{\mathrm{q}}$. □

C.1 Differential Privacy for Documents

We derive the ϵ guarantee that DSSE provides according to Definition 11. 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 result pattern for that w, i.e., $\tilde{\Pi}_w$. Regarding definition, the *obfuscated result pattern* $\tilde{\Pi}_w$ is a N-sized random vector characterized by:

$$\tilde{\Pi}_w[i] \sim \begin{cases}\mathbf{Bern}(\mathrm{p})+\mathbf{Bern}(\mathrm{q}) & \text{if } w \in \mathcal{D}[i],\\ \mathbf{Bern}(\mathrm{q}) & \text{if } w \notin \mathcal{D}[i],\end{cases} \quad \text{for } i \in [N]. \tag{6}$$

Let $\mathcal{D}$ and $\mathcal{D}'$ be two adjacent datasets that are identical except for their k-th 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 result pattern. Then, we want to find an upper bound for the ratio:

$$\frac{\Pr(\mathcal{M}(\mathcal{D},w)=\pi)}{\Pr(\mathcal{M}(\mathcal{D}',w)=\pi)}=\frac{\Pr(\tilde{\Pi}_w=\pi)}{\Pr(\tilde{\Pi}'_w=\pi)}=\prod_{i=1}^{N}\frac{\Pr(\tilde{\Pi}_w[i]=\pi[i])}{\Pr(\tilde{\Pi}'_w[i]=\pi[i])}.$$

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 $\mathcal{D}'[k]$). Therefore, we have:

$$\frac{\Pr(\mathcal{M}(\mathcal{D},w)=\pi)}{\Pr(\mathcal{M}(\mathcal{D}',w)=\pi)}=\frac{\Pr(\tilde{\Pi}_{w_*}[k]=\pi[k])}{\Pr(\tilde{\Pi}'_{w_*}[k]=\pi[k])}.$$

Since $\tilde{\Pi}_{w_*}[k] \sim \mathbf{Bern}(\mathrm{p})+\mathbf{Bern}(\mathrm{q})$ and $\tilde{\Pi}'_{w_*}[k] \sim \mathbf{Bern}(\mathrm{q})$, we can bound the left coefficient using (4):

$$\frac{\Pr(\mathcal{M}(\mathcal{D},w)=\pi)}{\Pr(\mathcal{M}(\mathcal{D}',w)=\pi)} \le \frac{\mathrm{q}+\mathrm{p}(1-\mathrm{q})}{\mathrm{q}}=1+\frac{\mathrm{p}(1-\mathrm{q})}{\mathrm{q}}.$$

We can follow the same procedure to prove the bound when the user queries for w'_*, which is:

$$\frac{\Pr(\mathcal{M}(\mathcal{D},w)=\pi)}{\Pr(\mathcal{M}(\mathcal{D}',w)=\pi)}=\frac{\Pr(\tilde{\Pi}_{w'_*}[k]=\pi[k])}{\Pr(\tilde{\Pi}'_{w'_*}[k]=\pi[k])} \le \frac{1}{1-\mathrm{p}}.$$

For both cases, we have:

$$\frac{\Pr(\mathcal{M}(\mathcal{D},w)=\pi)}{\Pr(\mathcal{M}(\mathcal{D}',w)=\pi)} \le \left(1+\frac{\mathrm{p}(1-\mathrm{q})}{\mathrm{q}}\right)\frac{1}{1-\mathrm{p}}=1+\frac{\mathrm{p}}{\mathrm{q}(1-\mathrm{p})}.$$

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 11 is:

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

which is identical to that of OSSE [81, Eq. (32)]. Using $\mathrm{TPR} = \mathrm{p} + \mathrm{q}(1-\mathrm{p})$ and $\mathrm{FPR} = \mathrm{q}$, we obtain the value in Theorem 2.

C.2 Differential Privacy for Keywords

Consider a database $\mathcal{D}$, and a pair of neighboring keyword lists $\mathbf{w}, \mathbf{w}' \in W^{|\mathbf{w}|}$ that are identical except for their k-th element, that is w in $\mathbf{w}$ and w' in $\mathbf{w}'$. 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 the ratio:

$$\frac{\Pr(\mathcal{M}(\mathcal{D}, w) = \pi)}{\Pr(\mathcal{M}(\mathcal{D}, w') = \pi)} = \frac{\Pr(\tilde{\Pi}_w = \pi)}{\Pr(\tilde{\Pi}_{w'} = \pi)} = \prod_{i=1}^{N} \frac{\Pr(\tilde{\Pi}_w[i] = \pi[i])}{\Pr(\tilde{\Pi}_{w'}[i] = \pi[i])}.$$

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 $i \in D_{0,1}$, $\tilde{\Pi}_w[i] \sim \mathbf{Bern}(q)$ and $\tilde{\Pi}_{w'}[i] \sim \mathbf{Bern}(p) + \mathbf{Bern}(q)$. The ratio between the probabilities of these distributions can be bounded using (5). 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 (4) and (5), we can get:

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

Therefore:

$$\begin{aligned} \prod_{i=1}^{N} \frac{\Pr(\tilde{\Pi}_w[i] = \pi[i])}{\Pr(\tilde{\Pi}_{w'}[i] = \pi[i])} &\leq \left(\frac{1}{1-p}\right)^{|D_{0,1}|} \left(\frac{q + p(1-q)}{q}\right)^{|D_{1,0}|} \\ &\leq \left(\frac{1}{1-p}\right)^{|D_{0,1}|+|D_{1,0}|} \left(\frac{q + p(1-q)}{q}\right)^{|D_{1,0}|+|D_{0,1}|} \\ &= \left(1 + \frac{p}{q(1-p)}\right)^{|D_{0,1}|+|D_{1,0}|}, \end{aligned}$$

and therefore according to Definition 12:

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

which is identical to that of OSSE [81, Eq. (44)] and implies Theorem 2.

D Security Proof of FROST (Theorem 3)

We prove that FROST is adaptively semantically secure when the underlying RePIR is simulation-based secure (SIM-secure) by adapting the semantic security analysis of OSSE [81, Section VI]. For this, we show that a simulator receiving the *obfuscated trace* can simulate the view of an adversary who is allowed to adaptively issue queries, which is defined as follows:

Definition 14 (Obfuscated Trace). *The trace of a history H_t when searching for t keywords $\mathbf{w}$ in* FROST*, called obfuscated trace and denoted by $\tilde{T}$, is:*

$$\tilde{T}(H_t) = (\mathsf{id}(\mathcal{D}), N, m, \tilde{\Pi}_{\mathbf{w}}),$$

where $\tilde{\Pi}_{\mathbf{w}}$ is the obfuscated result pattern for the query vector $\mathbf{w}$, with the i^{th}-row $\tilde{\Pi}_{\mathbf{w}[i]}$ is characterized by (6), N and m are the number of documents and keyword representation size, respectively.

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 $V_\kappa^s(H_t)$ for all $0 \leq s \leq t$, all polynomial-bounded functions g_p, all probabilistic polynomial-time adversaries $\mathcal{A}$, and all distributions $\mathcal{H}_t$; except with a negligible probability, where $t \in \mathbb{N}, H_t \xleftarrow{\$} \mathcal{H}_t$ and $(\kappa, \mathsf{st}, \mathsf{EDB}) \leftarrow \mathsf{Setup}(1^\lambda)$.

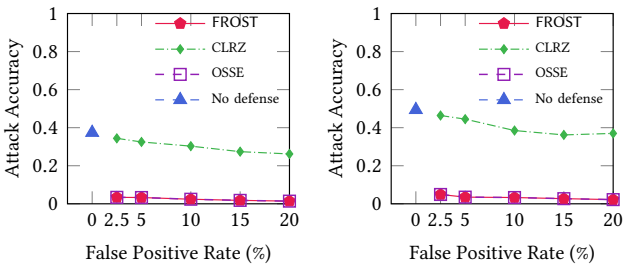
We recall the notions of (partial) view and (partial) obfuscated trace in FROST:

$$\begin{aligned} V_\kappa(H_t) &= (\mathsf{id}(\mathcal{D}), \mathcal{E}(\mathcal{D}[1]), \ldots, \mathcal{E}(\mathcal{D}[N]), \mathsf{EDB}, \\ &\qquad \mathfrak{s}_1, \ldots, \mathfrak{s}_t, \mathfrak{u}_1, \ldots, \mathfrak{u}_t), \\ V_\kappa^s(H_t) &= (\mathsf{id}(\mathcal{D}), \mathcal{E}(\mathcal{D}[1]), \ldots, \mathcal{E}(\mathcal{D}[N]), \mathsf{EDB}, \\ &\qquad \mathfrak{s}_1, \ldots, \mathfrak{s}_s, \mathfrak{u}_1, \ldots, \mathfrak{u}_s), \\ \tilde{T}(H_t) &= (\mathsf{id}(\mathcal{D}), N, m, \tilde{\Pi}_{\mathbf{w}[1]}, \ldots, \tilde{\Pi}_{\mathbf{w}[t]}), \\ \tilde{T}(H_t^s) &= (\mathsf{id}(\mathcal{D}), N, m, \tilde{\Pi}_{\mathbf{w}[1]}, \ldots, \tilde{\Pi}_{\mathbf{w}[s]}). \end{aligned}$$

First, note that simulating $\mathsf{id}(\mathcal{D})$ is trivial since we assumed that $\mathsf{id}(\mathcal{D}) = (1, \ldots, N)$. Also, encrypted documents $\mathcal{E}(\mathcal{D}[i])$ are indistinguishable from a random string $e_i^* \xleftarrow{\$} \{0,1\}^{|\mathcal{D}[i]|}$ since $\mathcal{E}$ is semantically secure. Similarly, update tokens $\mathfrak{u}_1, \ldots, \mathfrak{u}_t$ are indistinguishable from random by the security guarantee of pseudorandom functions. However, simulating the encrypted database $\mathsf{EDB} = (\mathsf{EIDX}, \mathsf{hint})$ and the search token $\mathfrak{s}$ is non-trivial.

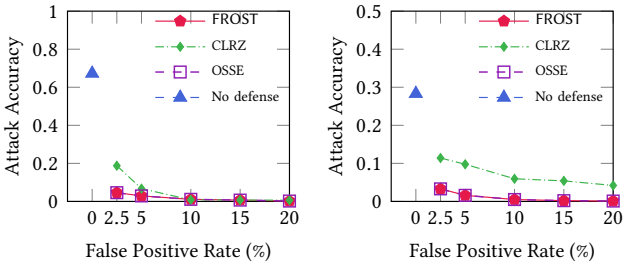
Since the underlying RePIR is SIM-secure, there exists a simulator $\mathcal{S}_{\mathsf{RePIR}}$, which can simulate RePIR, namely $\mathcal{S}_{\mathsf{RePIR}}$ has access to a query generation oracle $\mathcal{S}_{\mathsf{RePIR}}.\mathsf{Query}$ and an answer oracle $\mathcal{S}_{\mathsf{RePIR}}.\mathsf{Answer}$ such that the outputs of the two oracles are indistinguishable from those of RePIR.Query and RePIR.Answer.

- **Simulate** EDB. At time $s = 0$, $\mathcal{S}$ only holds $\tilde{T}(H_t^0) = (\mathsf{id}(\mathcal{D}), N, m)$ by which it generates EDB as follows: for a document i with an empty set of keywords, $\mathcal{S}$ calls $\mathcal{S}_{\mathsf{FROST}}.\mathsf{UpdtTkn}(\mathsf{st}^*, \kappa^*, i, \{\emptyset\})$ to compute $\mathsf{EIDX}^*[i, *]$ and $\mathsf{hint}^*[i, *]$, where $\kappa^* \xleftarrow{\$} \{0,1\}^\lambda$, and $\mathsf{st}_i^* \leftarrow 0$. EIDX^* and hint^* can be obtained by combining all $\mathsf{EIDX}^*[i, *]$ and $\mathsf{hint}^*[i, *]$ together, respectively. Also, $\mathcal{S}$ sets $\mathsf{EDB}^* \leftarrow \{\mathsf{EIDX}^*, \mathsf{hint}^*\}$. We claim that EDB^* is indistinguishable from EDB which will be proved below.
- **Simulate** $\mathfrak{s}_k$. At time $s \leq t$, $\mathcal{S}$ knows $\tilde{T}(H_t^s)$, by which it simulates $\mathfrak{s}_s$ as follows (simulations of $\mathfrak{s}_k$, for $k < s$, can be constructed similarly). Initialize an empty set FPSet. For $i \in [N]$, add i to FPSet if $\tilde{\Pi}_{\mathbf{w}[s]}[i] = 1$. Run lines 5 onward of Figure 3 to get the search token set $\mathfrak{s}_s^*$. We prove that $\mathfrak{s}_s^*$ and $\mathfrak{s}_s$ are indistinguishable.
- **Indistinguishability.** We prove that EDB^* and $\mathfrak{s}^*$ are indistinguishable from EDB and $\mathfrak{s}$ by contradiction: if there exists an adversary $\mathcal{A}$ which can distinguish EDB^* and $\mathfrak{s}^*$ from EDB and $\mathfrak{s}$, we show that $\mathcal{A}$ breaks SIM-security. Assume $\mathcal{A}$ can distinguish EDB^* and $\mathfrak{s}^*$ from EDB and $\mathfrak{s}$. Let $\{x_{j,0}\}$ (resp. $\{x_{j,1}\}$) be the underlying secrets of $\mathsf{EIDX}[*, j]$ (resp. $\mathsf{EIDX}^*[*, j]$), and $\{y_{j,0}\}$ (resp. $\{y_{j,1}\}$) be the underlying secrets of $\mathfrak{s}_i$ (resp. $\mathfrak{s}_i^*$). Consider the following two security games where one is between $\mathcal{A}$ and RePIR, denoted by $G_{\mathcal{A},\mathsf{RePIR}}$, and the other one is between $\mathcal{A}$ and $\mathcal{S}_{\mathsf{RePIR}}$, denoted by $G_{\mathcal{A},\mathcal{S}_{\mathsf{RePIR}}}$. In both games, the adversary $\mathcal{A}$ issues the same queries as follows:
  (1) Ciphertext query. $\mathcal{A}$ queries $(x_{j,0}, x_{j,1})$, $\forall\, j \in [m]$.
  (2) Token query. $\mathcal{A}$ queries $(y_{j,0}, y_{j,1})$, $\forall\, j \in [|\mathfrak{s}_i|]$, where $|\mathfrak{s}_i|$ means the number of tokens in $\mathfrak{s}_i$.



(a) Enron **(b) Lucene**

Figure 16: Accuracy of SAP attack ($|W| = 1000$).



(a) Enron **(b) Lucene**

Figure 17: Accuracy of GraphM attack ($|W| = 1000$).

In $G_{\mathcal{A},\text{RePIR}}$, with probability $1/2$, $b = 0$ and RePIR outputs EDB′ and $\mathfrak{s}_i'$ (since RePIR is probabilistic) which are indistinguishable from EDB and $\mathfrak{s}_i$. Likewise, in $G_{\mathcal{A},\mathcal{S}_{\text{RePIR}}}$, with probability $1/2$, $b = 1$ and $\mathcal{S}_{\text{RePIR}}$ outputs EDB$^{*'}$ and $\mathfrak{s}_i^{*'}$ (since $\mathcal{S}_{\text{RePIR}}$ is probabilistic) which are indistinguishable from EDB* and $\mathfrak{s}_i^*$.

Since $\mathcal{A}$ can distinguish EDB* and $\mathfrak{s}_i^*$ from EDB and $\mathfrak{s}_i$, it must be able to distinguish EDB′ and $\mathfrak{s}_i'$ from EDB$^{*'}$ and $\mathfrak{s}_i^{*'}$, namely $\mathcal{A}$ can distinguish the game with RePIR, denoted by $G_{\mathcal{A},\text{RePIR}}$ (the real world) from the game with $\mathcal{S}_{\text{RePIR}}$, denoted by $G_{\mathcal{A},\mathcal{S}_{\text{RePIR}}}$ (the ideal world). This contradicts the fact RePIR is SIM-secure. Therefore, $\mathcal{A}$ cannot distinguish EDB* and $\mathfrak{s}_i^*$ from EDB and $\mathfrak{s}_i$, and thus FROST is adaptively semantically secure. □

E Additional Evaluation

We evaluated the effectiveness of differential privacy approaches against other modern pattern leakage attacks including:

• SAP [75]: SAP is an attack by Oya and Kerschbaum that uses search result pattern leakage and a maximum likelihood approach to solve a LAP. Different from IHOP, which takes frequency and volume co-occurrence into account, SAP only uses individual keyword frequency and volume information. We used the original implementation by the authors. We set the hyperparameter $\alpha = 0.5$ as recommended by the authors.

• GraphM [78]: GraphM is a graph matching attack by Pouliot and Wright, which aims at solving a labeled graph matching problem (a QAP) by a heuristic convex-concave relaxation method using PATH algorithm [98]. It is quite similar to the IHOP attack [76] (see §6.1.1) as both aim to solve the same underlying optimization problem (a QAP), but IHOP uses a refined cost function with a heuristic approach for better convergence. We set the default weight parameter $\alpha = 0.5$, since we empirically found it performs the best.

Figure 16 shows the accuracy of SAP attack over 10K queries, which reaches 37.4% and 49.4% for Enron and Lucene datasets, respectively, without any defense. On Enron dataset (Figure 16a), when maintaining the TPR = 0.95 and increasing the FPR from 0.025 to 0.2, the attack accuracy is 3.4%–1.4% for FROST and OSSE, and 34.4%–26.2% for CLRZ. On Lucene dataset (Figure 16b), the corresponding figures are 3.3%–2.2% for FROST and OSSE, and 46.4%–37.0% for CLRZ. Since the SAP attack depends on individual query/keyword frequency and volume information, DP-based techniques reduce its effectiveness by adding false positives, thereby skewing the volume leakage distribution. Also, FROST and OSSE further obfuscate search patterns, making it difficult to infer true query frequency information.

Figure 17 presents the accuracy of GraphM attack on Enron and Lucene datasets with 1K keywords and over 10K queries. Without any defense, this attack can precisely recover 67.3% of queries on Enron dataset (Figure 17a), and 28.3% of queries on Lucene dataset (Figure 17b). For Enron dataset, the attack accuracy of GraphM drops to 4.6%–0.2% for FROST and OSSE, and 18.8%–0.6% for CLRZ. For Lucene dataset, the corresponding figures are 3.3%–0.1% for FROST and OSSE, and 11.4%–4.2% for CLRZ.