# MUSES: Efficient Multi-User Searchable Encrypted Database*

Tung Le
*Virginia Tech*

Rouzbeh Behnia
*University of South Florida*

Jorge Guajardo
*Robert Bosch LLC – RTC*

Thang Hoang
*Virginia Tech*

## Abstract

Searchable encrypted systems enable privacy-preserving keyword search on encrypted data. Symmetric systems achieve high efficiency (e.g., sublinear search), but they mostly support single-user search. Although systems based on public-key or hybrid models support multi-user search, they incur inherent security weaknesses (e.g., keyword-guessing vulnerabilities) and scalability limitations due to costly public-key operations (e.g., pairing). More importantly, most encrypted search designs leak statistical information (e.g., search, result, and volume patterns) and thus are vulnerable to devastating leakage-abuse attacks. Some pattern-hiding schemes were proposed. However, they incur significant user bandwidth/computation costs, and thus are not desirable for large-scale outsourced databases with resource-constrained users.

In this paper, we propose MUSES, a new multi-writer encrypted search platform that addresses the functionality, security, and performance limitations in the existing encrypted search designs. Specifically, MUSES permits single-reader, multi-writer functionalities with permission revocation and hides *all* statistical information (including search, result, and volume patterns) while featuring minimal user overhead. In MUSES, we demonstrate a unique incorporation of various emerging distributed cryptographic protocols including Distributed Point Function, Distributed PRF, and Oblivious Linear Group Action. We also introduce novel distributed protocols for oblivious counting and shuffling on arithmetic shares for the general multi-party setting with a dishonest majority, which can be found useful in other applications. Our experimental results showed that the keyword search by MUSES is two orders of magnitude faster with up to 97× lower user bandwidth cost than the state-of-the-art.

## 1 Introduction

Commodity cloud service (e.g., AWS IAM, Google Cloud IAM) provides data storage and sharing facilities for a large number of users. However, data outsourcing might lead to privacy concerns, especially for sensitive data (e.g., medical/financial). An adversarial cloud can access and exploit data illegitimately. Although end-to-end encryption permits confidentiality, it prevents data utility (e.g., querying, analytics), thereby invalidating the benefits of outsourcing services.

To address the data utilization and privacy dilemma, Searchable Encryption (SE) was proposed to enable keyword search over encrypted data while respecting the confidentiality of the data and the search query. There are two main SE models including Symmetric SE (SSE) [12,28,30,42,47,52,78] and Public-Key SE (PKSE) [4,8,10,91]. While SSE offers high efficiency, forward/backward privacy [12,42,59,59,79,83], and diverse queries (e.g., range [29,56,83]), it only supports a single user, where the data can only be searched by its owner. This strictly limits its practicality to apply for real-world settings, where the data can be contributed by multiple users. Moreover, SSE leaks statistical information including search/result/volume patterns, thus are vulnerable to leakage-abuse attacks [16,49,54,58,60,64,71,72,74,89,93]. To prevent these leakages, some oblivious SSE schemes (e.g., [28,36,47] were proposed using Oblivious RAM [77] or Private Information Retrieval (PIR) [43]; however, they incur significant overhead (bandwidth, computation) to the user [70].

On the other hand, PKSE enables multi-user encrypted search, in which one user (reader) can search on encrypted documents shared by the other users (writers) [62,66,90,91]. However, PKSE has some security issues including lack of forward privacy and dictionary attacks. Recently, Wang et al. proposed Hybrid SE (HSE) [84], which elegantly combines SSE and PKSE to achieve the benefits of both models: forward privacy and search efficiency by SSE, and multi-writer capability by PKSE. Despite its merits, HSE inherits other security weaknesses of both models, including keyword-guessing vulnerabilities and pattern leakages. Given that all existing SE schemes pose certain fundamental security, functionality, and efficiency limitations, we raise the following question:

*Can we design a new SE scheme that not only supports multi-writer search but also achieves concrete efficiency with*

---

*This is the full version of our USENIX Security'24 paper [61].

**Table 1:** Comparison of MUSES with prior encrypted search systems.

| Scheme | #Servers $L$ | Search Leakage $\mathcal{L}_{\mathsf{Search}}$ | Forward Privacy | Backward Privacy | Multi-Writer | Search Complexity (per writer) Server | Reader | Communication |
|---|---|---|---|---|---|---|---|---|
| PEKS [10] | 1 | $\{\mathsf{kw}, \mathsf{sp}(\mathsf{kw}), \mathsf{rp}(\mathsf{kw}), \mathsf{sv}(\mathsf{kw})\}$ | ✗ | ✗ | ✗ | $O(d_w N^{\dagger})$ | $O(\lambda)$ | $O(\lambda + n_s)$ |
| SSE [17] | 1 | $\{\mathsf{sp}(\mathsf{kw}), \mathsf{rp}(\mathsf{kw}), \mathsf{sv}(\mathsf{kw})\}$ | ✗ | ✗ | ✗ | $O(n_s)$ | $O(\lambda)$ | $O(\lambda + n_s)$ |
| NTRU-PEKS [8] | 1 | $\{\mathsf{kw}, \mathsf{sp}(\mathsf{kw}), \mathsf{rp}(\mathsf{kw}), \mathsf{sv}(\mathsf{kw})\}$ | ✗ | ✗ | ✗ | $O(d_w N^{\dagger})$ | $O(\lambda)$ | $O(\lambda + n_s)$ |
| DORY [28] | 2 | $\{\emptyset\}$ | ✓ | ✓ | ✗ | $O(mN)$ | $O(\log m + N)$ | $O(\lambda \log m + N)$ |
| FP-HSE [84] | 1 | $\{\mathsf{kw}, \mathsf{sp}(\mathsf{kw}), \mathsf{rp}(\mathsf{kw}), \mathsf{sv}(\mathsf{kw})\}$ | ✓ | ✗ | ✓ | $O(\lvert W\rvert^{\dagger})$ | $O(\lambda)$ | $O(\lambda + n_s)$ |
| AESM$^2$ [85] | 1 | $\{\mathsf{sp}(\mathsf{kw}), \mathsf{rp}(\mathsf{kw}), \mathsf{sv}(\mathsf{kw})\}$ | ✓ | ✗ | ✓ | $O(\lvert W\rvert^{\dagger})$ | $O(\lambda)$ | $O(\lambda + n_s)$ |
| Q-$\mu$SE [20] | 1 | $\{\mathsf{sp}(\mathsf{kw}), \mathsf{rp}(\mathsf{kw}), \mathsf{sv}(\mathsf{kw})\}$ | ✓ | ✓ | ✗ | $O(n_s)$ | $O(n_s + n_u)$ | $O(n_s + n_u)$ |
| Our MUSES | $\geq 2$ | $\{\emptyset\}$ | ✓ | ✓ | ✓ | $O(mN)$ | $O(\tau + N)$ | $O(\lambda\tau + n_s)$ |

- $\mathcal{L}_{\mathsf{Search}}$: Search leakage function with kw as the input keyword; sp: Search pattern; rp: Result pattern; sv: Search volume (see §3 for more details).
- $\lambda$: Security parameter; $N$: Total number of documents; $d_w$: Number of keywords per document. $n_u$: Number of updates after previous search. $W$: Collection of all unique keywords in the database (keyword universe); $m$: Size of keyword representation *per* document; $n_s$: Number of matched results *per* keyword search.
- In practice, $d_w < m \ll W$, $n_s \leq N$.
- Number of servers $L$ is considered as a constant number in complexity analysis. We assume document identifiers can be represented using a constant number of bits to skip this quantity in search complexity.

‡ $\tau = O(\log m)$ when $L = 2$, or $O(\sqrt{m})$ when $L \geq 3$.

† Public-key pairing operations.

*high security and privacy guarantees simultaneously?*

## 1.1 Our Contributions

We answer the above question affirmatively by proposing MUSES, a new distributed multi-writer encrypted search system that achieves a high level of security with concrete efficiency. MUSES achieves the following desirable properties.

- **Multi-writer functionalities**: MUSES allows single-reader, multi-writer functionalities similar to other multi-writer SE schemes (e.g., [7, 62, 66, 84, 85, 90, 92]. MUSES enables the writers to update their data that can be searched by the reader. MUSES also permits the writers to revoke the reader's search permission with writer-efficiency.
- **Security against statistical attacks:** MUSES is not vulnerable to dictionary attacks, achieves forward/backward privacy, and hides all pattern leakages simultaneously. Thus, it has more security guarantees than most prior SE schemes [10,55,62,66,84,85,87,91]. MUSES offers semi-honest security with dishonest majority. With $L$ servers, it can achieve $L-1$ privacy threshold, meaning that the confidentiality of data and queries is protected as long as one server is honest.
- **User-driven efficiency**: MUSES is designed with user efficiency in mind, and therefore, it is highly favorable to thin users with resource constraints (e.g., mobile devices). In MUSES, the reader only performs lightweight computations (e.g., modular additions) and its bandwidth is constant, compared with linear (w.r.t. collection size) in prior oblivious SE schemes (e.g., [28, 36, 47]). Evaluation results indicate that MUSES achieves up to 97× lower user bandwidth than the state-of-the-art oblivious SE. MUSES permits the writer to revoke the reader's search permission efficiently by offloading all processing tasks to the servers. This is more efficient than prior systems that do not naturally support revocation [28, 84] and thus, the writer needs to rebuild the index itself, which incurs high bandwidth and computation.
- **Low server cost**: In MUSES, the servers only perform low-cost operations (e.g., modular addition, rounding over small modulus). It is more efficient than PKSE/HSE designs that incur costly pairing operations [4,7,10,33,62,84,85,91,92].
- **Fully-fledged implementation and evaluation**: We fully implemented MUSES and evaluated its performance on commodity servers. Under real environments, experimental results demonstrated that MUSES performs search 126.8×–631.8× and 1.6×–3.9× faster than state-of-the-art SE techniques [28, 84] in the multi-writer and single-writer settings, respectively. Our implementation is available at `https://github.com/vt-asaplab/MUSES`.

**Techniques: Multi-party Oblivious Count and Shuffle.** To build MUSES, we construct several multi-party oblivious protocols that can be of independent interest. We design an $L$-party protocol to privately count how many elements in an additive secret-shared vector equal to a specific (small) value. Our protocol is novel as it does not require binary-arithmetic share conversion and/or costly comparison circuits. It operates in an online/offline model with a highly efficient online counting, where the parties only communicate in one round and perform lightweight computations (e.g., addition, circular shift) locally. We also design a new oblivious shuffle protocol for $L$ parties to randomly permute an additive secret-shared vector such that one party learns the shuffled vector, while each other party learns a part of a permutation composition[1]. Although there exists an $L$-party secret-shared shuffle [34], it is designed for secure communication, thus its online phase outputs the shares instead

---

[1] A permutation composition is formed by applying $L-1$ separate permutations $\pi_1, \ldots, \pi_{L-1}$ to shuffle a vector $\mathbf{d}$ as: $\pi_{L-1}(\pi_{L-2}(\ldots(\pi_1(\mathbf{d}))\ldots))$.

of the shuffled vector. We design a new preprocessing in a way that the online protocol outputs the shuffled vector directly. For applications that need shuffled data as output, our protocol is more direct and round-efficient than [34], which incurs an additional round for opening.

Table 1 compares MUSES with prior SE designs. To our knowledge, MUSES is the first multi-writer SE that can prevent all vulnerabilities and achieve high user efficiency (optimal bandwidth, low processing) simultaneously. Note that MUSES makes use of distributed servers to also mitigate inherent vulnerabilities in the multi-user setting (e.g., rollback attacks in Appendix F), albeit at the cost of additional processing overhead as a trade-off.

## 1.2 Technical Highlights

We present the technical highlights of our construction. MUSES is inspired by DORY [28] – an SSE scheme, and HSE [84] – a framework that shows how to adapt SSE to the multi-writer setting. We begin by giving DORY's overview and the challenges when adopting it to the multi-writer setting. We then present high-level ideas to address these challenges.

**Brief Overview of DORY.** DORY [28] is a (single-client) oblivious search scheme with dynamic update capability. Its search index is instantiated with a table structure, where columns represent keywords for performing search and rows represent documents with corresponding identifiers. To reduce the index size, Bloom Filter (BF) is employed to compress the keyword representation per document. The search index is row-wise encrypted with a symmetric key. To search for a keyword, the user computes its BF representation, and executes a PIR protocol based on the Distributed Point Function (DPF) [43] to privately retrieve $K$ encrypted columns of the index, where $K$ is a BF membership checking parameter. The user then decrypts these columns and aggregates them together to identify what document identifiers match the search query. To update a document, the user replaces its row in the index with a new encrypted BF representation.

In principle, DORY can be extended to support multi-writer (separate reader/writer) by incorporating public-key cryptography to distribute the symmetric key of the writer to the reader. However, due to its cryptographic protocol back-end, it may incur high processing complexity. Specifically, the reader's search complexity is linear to the number of documents $N$ in the database in terms of both network bandwidth and computation overhead (due to PIR reconstruction, decryption, and aggregation sequence) to obtain the search result. This overhead is significant for thin readers to search on large data collections. Meanwhile, an efficient search should only return a small number of matched identifiers. Furthermore, as the key is shared with the reader directly, it is challenging to enable access control. For example, a writer may want to restrict the reader's search permission on her index temporarily. A potential solution is to re-encrypt the search index with a fresh key unknown to the reader. However, this requires the writer to download the entire encrypted index, re-encrypt it with a new key, and then transmit it back to the server. This incurs significant bandwidth/processing costs to the writer. *Can we address all these challenges while maintaining efficiency?*

**Idea 1: Minimize reader overhead by delegating aggregation and decryption tasks to the server.** Instead of decrypting and aggregating $K$ PIR-retrieved encrypted columns on the reader side, we delegate all these processing tasks to the distributed servers securely. Specifically, we develop a protocol that incorporates Key-Homomorphic Pseudorandom Function (KH-PRF) [11] and DPF-based PIR [14, 15, 43] together, which permits the servers to partially decrypt the encrypted columns to obtain additive shares, and perform secure aggregation on the shares. Our protocol outputs the shares of the search result to the servers, which can be opened afterwards to return only matched document identifiers to the reader. This process ensures the servers do not learn anything during the partial decryption and aggregation (e.g., what columns are being aggregated/decrypted and decryption keys), given that at least one server does not collude with the others. However, although this strategy reduces the reader's processing and bandwidth costs, if we open the search result to the servers at this point, they can learn result patterns (i.e., what document identifiers match the query), volume patterns (i.e., the number of matched documents), thereby inferring search patterns (i.e., whether the same/different keywords are being searched). *Can we seal all these pattern leakages before opening while maintaining reader efficiency?*

**Idea 2: Conceal all pattern leakages via oblivious padding and random shuffling.** To hide volume information, we perform oblivious padding so that the search result always contains a fixed small number ($n_s$) of document identifiers[2]. We design an oblivious counting protocol based on linear group action [6], which permits the servers to privately count the current number of matched documents on the shares of the search result vector and open the count $s$ to the reader. Next, the reader creates a padding vector of size $n_s$, which contains $n_s - s$ padded matched values and secret shares it with $L$ servers. The servers then concatenate (the shares of) the search result vector and the padding vector together, forming a shared vector of size $(N+n_s)$. Our followed step is to design an $L$-party oblivious shuffle protocol based on the two-party scheme in [21], which enables $L$ servers to randomly permute the concatenated vector on its shares, and open the permuted vector to one server while other $L-1$ servers obtain the permutation composition. Since the permuted vector contains exactly $n_s$ document identifiers, all of which have been obfuscated due to oblivious shuffle, our strategy can conceal all volume, result, and search pattern leakages. To this end, the servers send obfuscated identifiers along with the permutation

---

[2] This assumption is similar to that of volume-hiding SE schemes [5], where an effective search should only return a small number of results

composition to the reader. The reader obtains the search result by applying the permutation inverses on the obfuscated list. Note that our padding and shuffle strategies require one more communication round in the search procedure; however, the optimal bandwidth complexity of $O(n_s)$ is maintained.

**Idea 3: Minimize writer overhead in revoking reader' permission via "key rotation" on the servers.** To revoke the reader's search capability, we re-encrypt the writer's search index on the servers with fresh keys unknown to the reader. At a high level, we incorporate the homomorphic property of KH-PRF with random masking techniques, which enables the servers to "rotate" the index that is currently encrypted by the old KH-PRF keys to the new ones on behalf of the writer in a privacy-preserving manner. The writer only needs to share the old and the new fresh KH-PRF keys with the servers, and does not need to stay involved in the later process.

## 2 Preliminaries

**Notation.** $||$ denotes the concatenation operator. $\lambda$ is the security parameter and $\mathbb{Z}_p$ is a ring of integers. We denote by $[n]$ the set $\{1,\ldots,n\}$. $x \xleftarrow{\$} [n]$ means $x$ is selected uniformly at random from $[n]$. For integers $q$ and $p$ where $q \geq p \geq 2$, we define $\lfloor \cdot \rceil_p : \mathbb{Z}_q \to \mathbb{Z}_p$ as a rounding function as $\lfloor x \rceil_p = i$ where $i \cdot \lfloor q/p \rceil$ is the largest multiple of $\lfloor q/p \rceil$ that does not exceed $x$. Given an integer $x$, $x^+$ denotes $x+1$ while $x^-$ denotes $x-1$. 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 two vectors $\mathbf{a}$ and $\mathbf{b}$. Given $\mathbf{M}$, $\mathbf{M}[u,*]$ and $\mathbf{M}[*,v]$ denote accessing the row $u$ and column $v$ of $\mathbf{M}$, respectively. $\mathbf{M}[u,v]$ denotes accessing the cell at row $u$ and column $v$. We denote the execution of protocol $A$ by $L$ parties $(o_1;o_2;\ldots;o_L) \leftarrow A(i_1;i_2;\ldots;i_L)$, where the input/output of each party is separated by a semicolon (;).

Let $\Sigma = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ be a public-key encryption: $(\mathsf{pk}, \mathsf{sk}) \leftarrow \Sigma.\mathsf{Gen}(1^\lambda)$ generating a public and private-key pair with security parameter $\lambda$; $c \leftarrow \Sigma.\mathsf{Enc}(\mathsf{pk}, m)$ encrypting plaintext $m$ with public key $\mathsf{pk}$; $m \leftarrow \Sigma.\mathsf{Dec}(\mathsf{sk}, c)$ decrypting ciphertext $c$ with private key $\mathsf{sk}$. Let $\mathsf{BF} = (\mathsf{Init}, \mathsf{Gen}, \mathsf{Vrfy})$ be a Bloom Filter (BF) [9]: $(H_1,\ldots,H_K) \leftarrow \mathsf{BF.Init}(m,K)$: generating $K$ mappings $H_k : \mathcal{S} \to [m]$ $\forall k \in [K]$ 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 $\mathbf{u}$.

**Secret Sharing.** Secret sharing enables a secret to be shared among $L$ parties. We denote $x^{(i)}$ as the additive share of a secret $x \in \mathbb{Z}_p$ to party $i$ such that $x = \sum_{i=1}^{L} x^{(i)} (\mathrm{mod}\ p)$.

**Bit Operations.** We denote $\oplus$ and $\otimes$ as the bit-wise XOR and AND operations, respectively. $x \lll t$ and $x \ggg t$ denote left-shift and right-shit operations by $t$ bits of value $x$.

Table 2 summarizes the symbols and notation in our scheme.

| | |
|---|---|
| $L$ | Number of parties (servers) |
| $m$ | Number of columns of index (BF parameter) |
| $K$ | Number of retrieved columns (BF parameter) |
| $N$ | Number of rows of index (Number of documents) |
| $n_s$ | Search output volume bound |
| $w$ | Writer identifier |
| $\mathsf{st}_w$ | State of writer $w$ |
| $\mathsf{EIDX}_w$ | Encrypted index of writer $w$ |
| $\mathsf{PTkn}_w$ | Private token of writer $w$ |
| $\mathsf{STkn}_w$ | Shared token of writer $w$ |
| $\mathcal{P}_i$ | Party (server) $i$ |
| $x^{(i)}$ | Secret share of $x$ owned by $\mathcal{P}_i$ |
| $\Sigma$ | Public-key encryption |
| $e$ | Rounding error of KH-PRF |
| $z$ | Number of reserved bits for accumulated error |
| $\pi/\pi^{-1}$ | Permutation/Permutation inverse |
| $\lll / \ggg$ | Bitwise left/right shift operator |
| $\circlearrowright$ | Circular shift right operator |
| $x^+/x^-$ | $x^+ = x+1, x^- = x-1$ |
| $[x]$ | $\{1,\ldots,n\}$ |
| $\lceil x \rceil, \lfloor x \rfloor$ | Round up/down of $x$ |

**Table 2:** Summary of notation

### 2.1 Distributed Point Function

Distributed Point Function (DPF) [14,15,43] permits $L$ parties to jointly evaluate a point function. For $a,b \in \{0,1\}^*$, let $P_{a,b} : \{0,1\}^{|a|} \to \{0,1\}^{|b|}$ s.t. $P_{a,b}(a) = b$ and $P_{a,b}(a') = 0^{|b|}$ $\forall$ $a' \neq a$. A DPF scheme contains the following algorithms.

- $(k^{(1)},\cdots,k^{(L)}) \leftarrow \mathsf{DPF.Gen}(1^\lambda, a, b)$: Given security parameter $\lambda$, and values $a,b \in \{0,1\}^*$, it outputs $L$ keys $k^{(1)},\cdots,k^{(L)} \in \mathcal{K}$.
- $y^{(\ell)} \leftarrow \mathsf{DPF.Eval}(k^{(\ell)}, x)$: Given a key $k^{(\ell)} \in \mathcal{K}$ and $x \in \{0,1\}^{|a|}$, it outputs $y^{(\ell)}$ as the share of $P_{a,b}(x)$.

DPF can be used to realize Private Information Retrieval (PIR) on a database $\mathbf{B} = (\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_m)$. The client creates $L$ keys $(k^{(1)},\ldots,k^{(L)}) \leftarrow \mathsf{DPF.Gen}(1^\lambda, j, 1)$, $j \in [m]$, for $L$ parties, $k^{(1)},\ldots,k^{(L)} \in \{0,1\}^n$ ($n = O(\lambda \log m)$ for $L = 2$, or $n = O(\lambda\sqrt{m})$ for $L \geq 3$) and $k^{(\ell)}$ is sent to party $\mathcal{P}_\ell \in \{\mathcal{P}_1,\ldots,\mathcal{P}_L\}$. Each party $\mathcal{P}_\ell$ returns $\mathbf{r}^{(\ell)} \leftarrow \sum_{i=1}^{m} \mathsf{DPF.Eval}(k^{(\ell)}, i) \times \mathbf{b}_i$ and the client reconstructs the retrieved item $\mathbf{b}_j \leftarrow \sum_{\ell=1}^{L} \mathbf{r}^{(\ell)}$.

### 2.2 Key-Homomorphic PRF (KH-PRF)

KH-PRF [11] enables distributed evaluation of a PRF function $F^* : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$ such that $(\mathcal{K}, *)$ and $(\mathcal{Y}, \bullet)$ are groups and for every $k_1, k_2 \in \mathcal{K}$, $F^*(k_1 * k_2, x) = F^*(k_1, x) \bullet F^*(k_2, x)$. An $L$-party KH-PRF scheme contains the following algorithms.

- $\mathbf{k} \leftarrow \mathsf{KH\text{-}PRF.Gen}(1^\lambda)$: Given a security parameter $\lambda$, it outputs a secret key $\mathbf{k} \in \mathcal{K}$.
- $(\mathbf{k}^{(1)},\ldots,\mathbf{k}^{(L)}) \leftarrow \mathsf{KH\text{-}PRF.Share}(\mathbf{k})$: Given a key $\mathbf{k} \in \mathcal{K}$, it outputs $L$ keys $\mathbf{k}^{(1)},\ldots,\mathbf{k}^{(L)} \in \mathcal{K}$ s.t. $\mathbf{k}^{(1)} * \cdots * \mathbf{k}^{(L)} = \mathbf{k}$.
- $y \leftarrow \mathsf{KH\text{-}PRF.Eval}(\mathbf{k}, s)$: Given a key $\mathbf{k} \in \mathcal{K}$ and a seed $s \in \{0,1\}^*$, it outputs the evaluation $y = F^*(\mathbf{k}, s) \in \mathcal{Y}$.

We extend the 2-party (almost) KH-PRF scheme based on Learning with Rounding (LWR) under Random Oracle

Model (ROM) by Boneh et al. [11] into $L$-party setting. Let $H_2 : \{0,1\}^* \rightarrow \mathbb{Z}_q^n$ be a hash function modeled as a random oracle. The KH-PRF function $F^* : \mathbb{Z}_q^n \times \{0,1\}^* \rightarrow \mathbb{Z}_p$ is defined as $F^*(\mathbf{k}, s) = \lfloor \langle H_2(s), \mathbf{k} \rangle \rceil_p$, where $\mathbf{k}^{(1)} + \cdots + \mathbf{k}^{(L)} = \mathbf{k}$, $F^*$ is an almost key homomorphic such that $F^*(\mathbf{k}, s) = e + \sum_{\ell=1}^{L} F^*(\mathbf{k}^{(\ell)}, s) \pmod{p}$ where $e \in \{0, \ldots, L\}$.

## 2.3 Linear Group Action

Let $(\mathbb{G}, \circ, 1)$ be a group. A linear group action [6] of $\mathbb{G}$ on $\mathbb{Z}_p^n$ is a function $\psi : \mathbb{Z}_p^n \times G \rightarrow \mathbb{Z}_p^n$ satisfying:

$$\begin{aligned} \psi(\mathbf{x}, 1) &= \mathbf{x} && \forall \mathbf{x} \in \mathbb{Z}_p^n \\ \psi(\mathbf{x}, g_1 \circ g_2) &= \psi(\psi(\mathbf{x}, g_1), g_2) && \forall \mathbf{x} \in \mathbb{Z}_p^n, g_1, g_2 \in \mathbb{G} \\ \psi(\mathbf{x} + \mathbf{y}, g) &= \psi(\mathbf{x}, g) + \psi(\mathbf{y}, g) && \forall \mathbf{x}, \mathbf{y} \in \mathbb{Z}_p^n, g \in \mathbb{G} \\ r\psi(\mathbf{x}, g) &= \psi(r\mathbf{x}, g) && \forall r \in \mathbb{Z}_p, \mathbf{x} \in \mathbb{Z}_p^n, g \in \mathbb{G} \end{aligned}$$

**Shuffle.** We define $\psi : \mathbb{Z}_p^n \times \mathbb{G} \rightarrow \mathbb{Z}_p^n$ as follows:

$$\psi(\mathbf{x}, \pi) \leftarrow \pi(\mathbf{x})$$

where $\pi$ is a permutation. For example, with $n = 4$, $\pi = (2,1,4,3)$ and $\mathbf{x} = (5,6,7,8)$, $\psi(\mathbf{x}, \pi) = \pi(\mathbf{x}) = (6,5,8,7)$.

Chase et al. [21] proposed a two-party Secret-shared Shuffle (TSS) protocol, which permits each party to obtain the share of $\pi(\mathbf{x})$. TSS consists of the following PPT algorithms.

- $(\boldsymbol{\Delta}; \mathbf{a}, \mathbf{b}) \leftarrow \mathsf{TSS.ShrTrns}(\pi; 1^\lambda)$: Given a permutation $\pi$ for $n$ elements from $\mathcal{P}_1$, and a security parameter $\lambda$ from $\mathcal{P}_2$, it outputs $\boldsymbol{\Delta} = \mathbf{b} - \pi(\mathbf{a}) \in \mathbb{Z}_p^n$ to $\mathcal{P}_1$ and $\mathbf{a}, \mathbf{b} \in \mathbb{Z}_p^n$ to $\mathcal{P}_2$.
- $(\mathbf{x}'; \mathbf{b}) \leftarrow \mathsf{TSS.Shffl}(\pi, \boldsymbol{\Delta}; \mathbf{x}, \mathbf{a}, \mathbf{b})$: Given a permutation $\pi$ and its corresponding $\boldsymbol{\Delta}$ from $\mathcal{P}_1$, a set $\mathbf{x}$ and masks $\mathbf{a}, \mathbf{b}$ from $\mathcal{P}_2$, it outputs $\mathbf{x}' = \pi(\mathbf{x}) + \mathbf{b} \in \mathbb{Z}_p^n$ as a masked permutation of $\mathbf{x}$ to $\mathcal{P}_1$, and the mask value $\mathbf{b}$ to $\mathcal{P}_2$.

**Circular Shift.** Let $\mathbb{G}$ be a group modulo $n$. We define $\psi : \mathbb{Z}_p^n \times \mathbb{G} \rightarrow \mathbb{Z}_p^n$ as follows:

$$\psi(\mathbf{x}, g) \leftarrow (\mathbf{x} \circlearrowright g)$$

where $\circlearrowright$ denotes circular shift operator, which rotates vector $\mathbf{x}$ to the right by $g$ positions. For example, with $\mathbf{x} = (1,2,3,4)$ and $g = 1$, we have $\psi(\mathbf{x}, g) = \mathbf{x} \circlearrowright g = (4,1,2,3)$.

# 3 Models

**System Model.** Our system consists of an honest reader, $n_w$ independent writers, and $L$ servers. WLOG, we identify each writer as a member of $[n_w]$, i.e., $\mathcal{W} = [n_w]$. Each writer $w \in \mathcal{W}$ owns a collection of $N$ documents (each identified as a member of $[N]$) and desires to share the collection with the reader. The reader would like to perform an encrypted keyword search over the collections of a writer subset $\mathcal{W}' \subseteq \mathcal{W}$. Also, the writer can revoke the permission of the reader if needed. The reader and writers are independent and they do not communicate directly with each other in any operations (i.e., search, update, revoke) after system setup.

**Definition 1 (MUSES).** *A MUSES scheme is a tuple of PPT algorithms defined as follows:*

- $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{RSetup}(1^\lambda)$*: Given a security parameter* $\lambda$*, it outputs a public and private key pair* $(\mathsf{pk}, \mathsf{sk})$.
- $(\mathsf{EIDX}_w, \mathsf{st}_w, \mathsf{PTkn}_w, \mathsf{STkn}_w) \leftarrow \mathsf{WSetup}(1^\lambda, w, \mathsf{pk})$*:Given a security parameter* $\lambda$*, a writer identifier* $w$*, the reader's public key* $\mathsf{pk}$*, it outputs an encrypted index* $\mathsf{EIDX}_w$*, a state* $\mathsf{st}_w$*, a private token* $\mathsf{PTkn}_w$*, and a shared token* $\mathsf{STkn}_w$ *encrypted under* $\mathsf{pk}$.
- $\mathfrak{s} \leftarrow \mathsf{SearchToken}(\mathsf{kw}, \mathcal{W}')$*: Given a keyword* $\mathsf{kw}$*, and a subset of writers* $\mathcal{W}'$*, it outputs a search token* $\mathfrak{s}$.
- $O \leftarrow \mathsf{Search}(\mathfrak{s}, \mathsf{sk}, \{(\mathsf{EIDX}_w, \mathsf{STkn}_w, \mathsf{st}_w)\}_{w \in [n_w]})$*: Given a search token* $\mathfrak{s}$*, the reader's private key* $\mathsf{sk}$*, encrypted search indices* $\mathsf{EIDX}_w$*, shared tokens* $\mathsf{STkn}_w$*, and states* $\mathsf{st}_w$ *of writers* $w \in [n_w]$*, it outputs the search result* $O$.
- $\mathfrak{u}_w \leftarrow \mathsf{UpdateToken}(\mathcal{V}_u, u, w, \mathsf{PTkn}_w, \mathsf{st}_w)$*: Given an updated document identifier* $u$*, a set of updated keywords* $\mathcal{V}_u$*, a writer identifier* $w$*, the writer's private token* $\mathsf{PTkn}_w$ *and state* $\mathsf{st}_w$*, it outputs an update token* $\mathfrak{u}_w$.
- $(\mathsf{EIDX}'_w, \mathsf{st}'_w) \leftarrow \mathsf{Update}(\mathfrak{u}_w, \mathsf{EIDX}_w, \mathsf{st}_w)$*: Given an update token* $\mathfrak{u}_w$*, an encrypted index* $\mathsf{EIDX}_w$ *and a state* $\mathsf{st}_w$ *of writer* $w$*, it outputs updated index* $\mathsf{EIDX}'_w$ *and state* $\mathsf{st}'_w$.
- $(\mathsf{EIDX}'_w, \mathsf{PTkn}'_w) \leftarrow \mathsf{RvkPrm}(w, \mathsf{PTkn}_w, \mathsf{EIDX}_w, \mathsf{st}_w)$*: Given a writer identifier* $w$*, the writer's private token* $\mathsf{PTkn}_w$*, the writer's index* $\mathsf{EIDX}_w$ *and state* $\mathsf{st}_w$*, it outputs an updated index* $\mathsf{EIDX}'_w$ *and updated token* $\mathsf{PTkn}'_w$.

The correctness of MUSES is presented in Appendix D.

**Threat and Security Models.** We assume the adversary can corrupt up to $L-1$ servers and any number of writers. We assume the adversary is semi-honest, meaning that it is curious about the query of other honest writers/reader but follows the protocols faithfully.

We concentrate on the security of the search index and its related operations. Let $\mathsf{EIDX}_w = (I_{w,1}, I_{w,2}, \ldots, I_{w,N})$ be an encrypted search index of writer $w$, where $I_{w,u}$ contains information about keywords of the $u$-th document. Let $\mathbb{O}_w$ be a sequence of operations on $\mathsf{EIDX}_w$. We denote $t$ as the timestamp when an operation happens, $\mathbb{O}_w$ records $(t, \mathsf{kw})$ for a search on keyword $\mathsf{kw}$, and $(t, u, \mathcal{V}_u)$ for an update of document $u$ with new keywords $\mathcal{V}_u$ on $\mathsf{EIDX}_w$.

**Definition 2 (Keyword Pattern Leakages).** *The pattern leakages in keyword search are defined as follows.*

- Search pattern $\mathsf{sp}$ *indicates the frequency of search operations on a keyword, i.e.,* $\mathsf{sp}(\mathsf{kw}) = \{t : (t, \mathsf{kw}) \in \mathbb{O}_w)\}$.
- Result pattern $\mathsf{rp}$ *reveals what documents match the queried keyword* $\mathsf{kw}$*, i.e.,* $\mathsf{rp}(\mathsf{kw}) = \{u_1, \ldots, u_{N'} : \mathsf{kw} \in I_{w,u_l} \forall\, l \in [N'] \subseteq [N]\}$.
- Search volume $\mathsf{sv}$ *indicates the number of documents that matched the queried keyword, i.e.,* $\mathsf{sv}(\mathsf{kw}) = N'$ *s.t.* $\mathsf{kw} \in I_{w,u}\ \forall\ u \in [N'] \subseteq [N]$.

*We follow [5, 73] to derive a security definition for an encrypted search scheme that can hide the above search volume. Let* $p_b^{\mathcal{A},\mathcal{L}_{\mathcal{H}}}$ *be the probability that* $\mathcal{A}$ *outputs* $b$ *when playing* $\mathsf{VHGame}_b^{\mathcal{A},\mathcal{L}_{\mathcal{H}}}(1^\lambda)$ *(Figure 15). A leakage function* $\mathcal{L}_{\mathcal{H}} = (\mathcal{L}_{\mathcal{H}}^{\mathsf{Setup}}, \mathcal{L}_{\mathcal{H}}^{\mathsf{Search}}, \mathcal{L}_{\mathcal{H}}^{\mathsf{Update}})$ *is volume-hiding if for all adversaries* $\mathcal{A}$*:*

$$p_0^{\mathcal{A},\mathcal{L}_{\mathcal{H}}}(1^\lambda) = p_1^{\mathcal{A},\mathcal{L}_{\mathcal{H}}}(1^\lambda)$$

**Definition 3 (Adaptive Security of MUSES).** *For all PPT adversary* $\mathcal{A}$ *and the game* $\mathsf{IND}^b_{\mathit{MUSES},\mathcal{A},\mathcal{L}_{\mathcal{H}}}(1^\lambda)$ *in Figure 14, MUSES is* $\mathcal{L}_{\mathcal{H}}$*-adaptively-secure if:* $|\Pr[\mathsf{IND}^0_{\mathit{MUSES},\mathcal{A},\mathcal{L}_{\mathcal{H}}}(1^\lambda) = 1] - \Pr[\mathsf{IND}^1_{\mathit{MUSES},\mathcal{A},\mathcal{L}_{\mathcal{H}}}(1^\lambda) = 1]| \leq \mathsf{negl}(\lambda)$.

**Definition 4 (Forward and Backward Privacy).** *MUSES is* forward-private *if* $\mathcal{L}_{\mathcal{H}}^{\mathsf{Update}}(w, u, \mathcal{V}_u)$ *of any update* $(w, u, \mathcal{V}_u)$ *by writer* $w$ *s.t.* $(\mathsf{CorruptWriter}, w) \notin \mathcal{H}$ *can be written as* $\mathcal{L}'(w, u)$*, and* backward private *if* $\mathcal{L}_{\mathcal{H}}^{\mathsf{Search}}$ *can be written as* $\mathcal{L}_{\mathcal{H}}^{\mathsf{Search}}(\mathsf{kw}, \mathcal{W}') = \mathcal{L}''(\mathcal{W}')$*, where* $\mathcal{L}'$*,* $\mathcal{L}''$ *are stateless.*

**Definition 5 (Update Pattern Leakage).** *The update pattern* $\mathsf{up}$ *records the update frequency on documents, i.e.,* $\mathsf{up}(u) = \{t : (t, u, \perp) \in \mathbb{O}_w\}$

The adversary can issue a sequence of queries to the MUSES oracle for any of the following: (*i*) writer corruption query, which returns the secret key of a specific writer; (*ii*) search query, which returns the search token of the queried keyword under a writer subset; (*iii*) update query, which returns the update token for a document of a specific writer; and (*iv*) revoke query, which returns the updated search index and private tokens of a specific writer. The adversary can issue queries based on prior outcomes. To define security, we define the notion of *history* that captures a sequence of queries issued by the adversary into MUSES as follows.

**Definition 6 (History).** *A history of MUSES is a sequence of queries* $\mathcal{H} = \{\mathsf{Hist}_t\}$*, where sequence number* $t$ *denotes the timestamp when the query happens and* $\mathsf{Hist}_t \in \{(\mathsf{CorruptWriter}, w), (\mathsf{Search}, \mathsf{kw}, \mathcal{W}'), (\mathsf{Update}, w, u, \mathcal{V}_u), (\mathsf{Revoke}, w)\}$.

We introduce a leakage function family $\mathcal{L}_{\mathcal{H}} = \{\mathcal{L}_{\mathcal{H}}^{\mathsf{Setup}}, \mathcal{L}_{\mathcal{H}}^{\mathsf{Search}}, \mathcal{L}_{\mathcal{H}}^{\mathsf{Update}}, \mathcal{L}_{\mathcal{H}}^{\mathsf{Revoke}}, \mathcal{L}_{\mathcal{H}}^{\mathsf{CorruptWriter}}\}$ to cover the information of history $\mathcal{H}$ leaked during setup, search, update, permission revocation, and writer corruption, respectively. When an oracle is queried for the $t$-th operation, any function in $\mathcal{L}_{\mathcal{H}}$ is initialized with $\mathcal{H}$, which is the history consisting of the previous $(t-1)$ operations and the $t$-th operation as the function input. It captures the leakage incurred by the current operation and all historical operations. Before any query (i.e., $\mathcal{H} = \{\emptyset\}$), $\mathcal{L}_{\mathcal{H}} = \mathcal{L}_{\mathcal{H}}^{\mathsf{Setup}}$. We also implicitly assume that MUSES histories are *non-singular* as defined in [26, 84].

**Definition 7 (Non-Singular History).** *A history* $\mathcal{H}$ *is non-singular if there exists at least one history* $\mathcal{H}' \neq \mathcal{H}$ *can be found in polynomial-time given* $\mathcal{L}_{\mathcal{H}}$ *such that* $\mathcal{L}_{\mathcal{H}} = \mathcal{L}_{\mathcal{H}'}$.

**Corruption leakage.** To capture corruption leakage, we define $\mathsf{UpdateBy}(w) = \{\mathsf{Hist}_t : \mathsf{Hist}_t = (\mathsf{Update}, w, u, \mathcal{V}_u) \in \mathcal{H}\}$, which lists all updates by the writer $w$ in the history.

**Document access leakage.** In this paper, we focus only on the security of the search index. Accessing document content from the collection after the search is out-of-scope. Sealing document access leakage is an independent study. Some oblivious file systems (e.g., [22, 23, 65, 67]) can be used orthogonally with our scheme for system-wide end-to-end security.

# 4 Our Proposed Scheme

## 4.1 Data Structures

**Search Index.** In MUSES, each writer $w \in \mathcal{W}$ has an independent index for keyword-document representation. We use a probabilistic data structure (i.e., BF) to create a compressed index for each writer. Suppose there are $N$ documents, the writer extracts a set of unique keywords $\mathcal{V}_u$ for each document $u \in [N]$ and computes its BF representation as $\mathbf{v}_u \leftarrow \mathsf{BF.Gen}(\mathcal{V}_u) \in \{0,1\}^m$. The search index contains $N$ BF vectors, which is interpreted as a $N \times m$ binary matrix $\mathsf{IDX}_w = [\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_N] \in \{0,1\}^{N \times m}$. Searching a keyword incurs checking BF membership by reading $K$ columns in $\mathsf{IDX}_w$, where $K$ is the BF parameter. Updating a document $u$ recreates a new BF representation of the updated keywords and overwrites the corresponding row $\mathsf{IDX}_w[u, *]$.

$\mathsf{IDX}_w$ is encrypted for confidentiality. For reader efficiency, the writer encrypts her index in a way that the decryption can be delegated to the servers during search. MUSES employs KH-PRF function $F^*$ for such delegation as follows.

First, the writer $w$ interprets the index as a matrix $\mathsf{IDX}_w = [\mathbf{c}_1 \quad \mathbf{c}_2 \quad \ldots \quad \mathbf{c}_m] \in \{0,1\}^{N \times m}$, where $\mathbf{c}_v \in \{0,1\}^N \ \forall v \in [m]$. For each column $v \in [m]$, the writer generates a KH-PRF key as $\mathbf{r}_v \leftarrow \mathsf{KH\text{-}PRF.Gen}(1^\lambda) \in \mathbb{Z}_q^n$ for column-wise encryption. Since $F^*$ is an almost KH-PRF, there exists a small error $e$ (§2.2) during the KH-PRF evaluation. Thus, it is necessary to "reserve" several bits for the error in the column data before being encrypted with KH-PRF so that such error can be "ruled out" after KH-PRF decryption. Moreover, since the servers also perform secure addition of $K$ columns after KH-PRF evaluations for BF membership verification, we need to reserve enough space for the accumulated error. Let $z = \lceil \log_2(e \cdot K) \rceil$ be the number of bits for the accumulated error when adding $K$ columns encrypted by KH-PRF together. For each $u \in [N]$, the writer encrypts the element $\mathbf{c}_v[u]$ as

$$\hat{\mathbf{d}}_v[u] \leftarrow (\mathbf{c}_v[u] \lll z) + F^*(\mathbf{r}_v, u \,||\, \mathsf{st}_{w,u}) \pmod{p} \quad (1)$$

where $\mathsf{st}_{w,u}$ is the update state of document $u$ (initialized with 0). The final encrypted index is $\mathsf{EIDX}_w = [\hat{\mathbf{d}}_1 \quad \hat{\mathbf{d}}_2 \quad \ldots \quad \hat{\mathbf{d}}_m] \in \mathbb{Z}_p^{N \times m}$, where $\hat{\mathbf{d}}_v \in \mathbb{Z}_p^N \ \forall v \in [m]$.

**Auxiliary Information.** In MUSES, each writer $w$'s index $\mathsf{EIDX}_w$ is associated with three auxiliary components:

- Private token $\mathsf{PTkn}_w$: It contains KH-PRF keys $\mathbf{r}_v$ that are

```
$\mathsf{RSetup}(1^\lambda)$:
1. $(\mathsf{pk}, \mathsf{sk}) \leftarrow \Sigma.\mathsf{Gen}(1^\lambda)$
2. return $(\mathsf{pk}, \mathsf{sk})$

$\mathsf{WSetup}(1^\lambda, w, \mathsf{pk})$:
1. For $v = 1$ to $m$
    (a) $\mathbf{r}_{w,v} \leftarrow \mathsf{KH\text{-}PRF.Gen}(1^\lambda)$
    (b) $\mathsf{STkn}_{w,v} \leftarrow \Sigma.\mathsf{Enc}(\mathsf{pk}, \mathbf{r}_{w,v})$
2. $\mathsf{st}_w \leftarrow (\mathsf{st}_{w,1}, \mathsf{st}_{w,2}, \ldots, \mathsf{st}_{w,N})$, where $\mathsf{st}_{w,u} \leftarrow 0$, for $u \in [N]$
3. $\mathsf{EIDX}_w[u,v] \leftarrow F^*(\mathbf{r}_{w,v}, u || \mathsf{st}_{w,u})$, for $u \in [N]$ and $v \in [m]$
4. $\mathsf{PTkn}_w \leftarrow (\mathbf{r}_{w,1}, \mathbf{r}_{w,2}, \ldots, \mathbf{r}_{w,m})$
5. $\mathsf{STkn}_w \leftarrow (\mathsf{STkn}_{w,1}, \mathsf{STkn}_{w,2}, \ldots, \mathsf{STkn}_{w,m})$
6. return $(\mathsf{EIDX}_w, \mathsf{st}_w, \mathsf{PTkn}_w, \mathsf{STkn}_w)$
```

**Figure 1:** MUSES setup.

used to encrypt the search index.

- Shared token $\mathsf{STkn}_w$: It contains information for the reader to search on $\mathsf{EIDX}_w$, which are the KH-PRF keys $\mathbf{r}_v$, encrypted under the reader's public key. Specifically, $\mathsf{STkn}_w = (\mathsf{STkn}_{w,1}, \ldots, \mathsf{STkn}_{w,m})$, where $\mathsf{STkn}_{w,v} \leftarrow \Sigma.\mathsf{Enc}(\mathsf{pk}, \mathbf{r}_v)$ for $v \in [m]$ and $(\mathsf{pk}, \mathsf{sk}) \leftarrow \Sigma.\mathsf{Gen}(1^\lambda)$ is the reader's public/private keys.
- Update state $\mathsf{st}_w$: It contains public information concatenated with document identifier as seed value for KH-PRF evaluation. Specifically, $\mathsf{st}_w = (\mathsf{st}_{w,1}, \ldots, \mathsf{st}_{w,N})$, where $\mathsf{st}_{w,u}$ is the counter value (initialized with 0) for document $u$ incremented after each update on that document happens.

**Setup.** Figure 1 presents the setup procedures as follows.

- Reader: It executes RSetup algorithm to generate a public/private key pair $(\mathsf{pk}, \mathsf{sk})$ and broadcasts pk to all writers.
- Writer: Each writer $w$ executes WSetup algorithm on reader's public key pk to generate four components: the encrypted index $\mathsf{EIDX}_w$, a private token $\mathsf{PTkn}_w$, a shared token $\mathsf{STkn}_w$, and an update state $\mathsf{st}_w$. The writer first generates $m$ KH-PRF keys $(\mathbf{r}_{w,1}, \ldots, \mathbf{r}_{w,m})$ to encrypt $m$ columns of the search index (step 1(a)). These KH-PRF keys are stored as the private token $\mathsf{PTkn}_w$ by the writer and also encrypted under the reader's public key pk as shared token $\mathsf{STkn}_w$ (steps 1(a)–1(b)) to grant search permission to the reader. The writer initializes a public update state $\mathsf{st}_w$ as zero values (step 2). Finally, the writer encrypts an empty search index cell-by-cell by evaluating KH-PRF function $F^*$ with KH-PRF keys $\mathbf{r}_{w,v}$ and the seeds formed by row indices and update counters (step 3). Finally, the writer sends $\mathsf{EIDX}_w$ and its auxiliary components ($\mathsf{st}_w$, $\mathsf{STkn}_w$) to $L$ servers while keeping $\mathsf{PTkn}_w$ private.

## 4.2 Search Procedure

We present the search protocol of MUSES in Figure 2. To search for a keyword kw on a writer subset $\mathcal{W}'$, the reader first computes its BF representation as column indices $(v_1, \ldots, v_K)$, and creates corresponding DPF keys $\{q_1^{(i)}, \ldots, q_K^{(i)}\}_{i \in [L]}$ (SearchToken, steps 1–2), and sends them to $L$ servers, where each server $\mathcal{P}_i$ receives a search token $\mathfrak{s}_i = (\mathcal{W}', \{q_k^{(i)}\}_{k \in [K]})$. The servers then execute Search algorithm, which performs three main operations: (*i*) partial decryption, (*ii*) oblivious counting/padding, and (*iii*) oblivious shuffle.

```
$\mathsf{SearchToken}(\mathsf{kw}, \mathcal{W}')$:
1. For each $k \in [K]$:
    (a) $v_k \leftarrow H_k(\mathsf{kw})$
    (b) $(q_k^{(1)}, \ldots, q_k^{(L)}) \leftarrow \mathsf{DPF.Gen}(1^\lambda, v_k, 1)$
2. $\mathfrak{s}_i \leftarrow (\mathcal{W}', \{q_k^{(i)}\}_{k \in [K]})$ for each $i \in [L]$
3. return $\mathfrak{s} \leftarrow (\mathfrak{s}_1, \ldots, \mathfrak{s}_L)$

$\mathsf{Search}(\mathfrak{s}, \mathsf{sk}, \{(\mathsf{EIDX}_w, \mathsf{STkn}_w, \mathsf{st}_w)\}_{w \in [n_w]})$:
Let $\mathfrak{s} = (\mathcal{W}', \{q_k^{(i)}\}_{k \in [K]})$. For each writer $w \in \mathcal{W}'$, repeat the following process on its components $\mathsf{EIDX} = \mathsf{EIDX}_w$, $\mathsf{STkn} = \mathsf{STkn}_w$, $\mathsf{st} = \mathsf{st}_w$:
4. For each $k \in [K]$, each server $\mathcal{P}_i$ locally computes:
    (a) $\hat{\mathbf{d}}_{v_k}^{(i)} \leftarrow \sum_{v=1}^{m} \mathsf{DPF.Eval}(q_k^{(i)}, v) \times \mathsf{EIDX}[*, v]$
    (b) $\mathsf{STkn}_{v_k}^{(i)} \leftarrow \sum_{v=1}^{m} \mathsf{DPF.Eval}(q_k^{(i)}, v) \times \mathsf{STkn}_v$
    (c) Send $\mathsf{STkn}_{v_k}^{(i)}$ to the reader.
5. For each $k \in [K]$, the reader computes:
    (a) $\mathsf{STkn}_{v_k} \leftarrow \sum_{i=1}^{L} \mathsf{STkn}_{v_k}^{(i)} \pmod{p}$
    (b) $\mathbf{r}_{v_k} \leftarrow \Sigma.\mathsf{Dec}(\mathsf{sk}, \mathsf{STkn}_{v_k})$
    (c) $(\mathbf{r}_{v_k}^{(1)}, \ldots, \mathbf{r}_{v_k}^{(L)}) \leftarrow \mathsf{KH\text{-}PRF.Share}(\mathbf{r}_{v_k})$
6. The reader sends $\{\mathbf{r}_{v_k}^{(i)}\}_{k \in [K]}$ to server $\mathcal{P}_i$ for each $i \in [L]$
7. For each $u \in [N]$, each server $\mathcal{P}_i$ locally computes:
    (a) $\tilde{\mathbf{d}}_{v_k}^{(i)}[u] \leftarrow \hat{\mathbf{d}}_{v_k}^{(i)}[u] - F^*(\mathbf{r}_{v_k}^{(i)}, u \,||\, \mathsf{st}_u) \pmod{p}$ for $k \in K$
    (b) $\mathbf{d}^{(i)}[u] \leftarrow \sum_{k=1}^{K} \tilde{\mathbf{d}}_{v_k}^{(i)}[u] \pmod{p}$
8. All servers jointly execute $\Pi_{\mathsf{LOC}}$ online protocol on shares $\mathbf{d}^{(i)}$ and obtain $(s^{(i)}, \mathbf{d}'^{(i)})$. Each $\mathcal{P}_i$ sends $s^{(i)}$ to the reader
9. The reader computes:
    (a) $s \leftarrow \sum_{i=1}^{L} s^{(i)} \pmod{p}$
    (b) Generates padding vector $\mathbf{p} = (\mathbf{1}^{n_s - s} \,||\, \mathbf{0}^s)$
    (c) Secret shares $\mathbf{p}$ as $\mathbf{p}^{(i)} \xleftarrow{\$} \mathbb{Z}_p^{n_s}$ such that $\sum_{i=1}^{L} \mathbf{p}^{(i)} = \mathbf{p}$
    (d) Sends $\mathbf{p}^{(i)}$ to each server $\mathcal{P}_i \in \{\mathcal{P}_1, \ldots, \mathcal{P}_L\}$.
10. Every $\mathcal{P}_i$ concatenates $\mathbf{o}^{(i)} = (\mathbf{d}'^{(i)} || \mathbf{p}^{(i)})$
11. All servers jointly execute $\Pi_{\mathsf{LOS}}$ online protocol on shares $\mathbf{o}^{(i)}$, which outputs permutation $\pi_i$ to each $\mathcal{P}_i \in \{\mathcal{P}_1, \ldots, \mathcal{P}_{L-1}\}$ and shuffled vector $\hat{\mathbf{o}}$ to $\mathcal{P}_L$. $\mathcal{P}_L$ sends $O = \{u : \hat{\mathbf{o}}[u] = 1\}$ to the reader, and each $\mathcal{P}_i \in \{\mathcal{P}_1, \ldots, \mathcal{P}_{L-1}\}$ sends $\pi_i$ to the reader
12. return $O' = \{u' : u' \leq N \wedge u' = \pi_1^{-1}(\ldots(\pi_{L-1}^{-1}(u))\ldots) \forall u \in O\}$
```

**Figure 2:** MUSES search.

### 4.2.1 Partial decryption

First, each server $\mathcal{P}_i$ evaluates DPF on two components of the writer: the search index EIDX and shared token STkn. The former is to privately retrieve $(\hat{\mathbf{d}}_{v_1}^{(i)}, \ldots, \hat{\mathbf{d}}_{v_K}^{(i)})$ as the additive shares of $K$ requested columns in EIDX, while the latter is to privately retrieve $(\mathsf{STkn}_{v_1}^{(i)}, \ldots, \mathsf{STkn}_{v_K}^{(i)})$ as the additive shares of the $K$ corresponding shared tokens (step 4).

The servers send the shares of the shared tokens to the reader to open the secret keys $\mathbf{r}_{v_k}$ (for $k \in [K]$) (steps 5(a)–5(b)). The reader then delegates the (partial) decryption to the servers by creating $(\mathbf{r}_{v_k}^{(1)}, \ldots, \mathbf{r}_{v_k}^{(L)})$ as the KH-PRF (additive) shares of each $\mathbf{r}_{v_k}$ (step 5(c)), and distributes each $\mathbf{r}_{v_k}^{(i)}$ to each

**Protocol 1.** $\Pi_{\mathsf{LOC}}$ – Multi-party Oblivious Count Protocol

```
Preprocessing:
Input: Security parameter $\lambda$ and value to be counted $K$
Output: Each $\mathcal{P}_i$ obtains shares $r^{(i)} \in \mathbb{Z}_{K^+}, \mathbf{b}^{(i)} \in \mathbb{Z}_p^{K^+}$ s.t. $r = \sum_{i=1}^{L} r^{(i)}$, $\mathbf{b} = \sum_{i=1}^{L} \mathbf{b}^{(i)}$ is a unit vector with $\mathbf{b}[r] = 1$
1. Every $\mathcal{P}_i$ sets $r^{(i)} \xleftarrow{\$} \mathbb{Z}_{K^+}, \pi^{(i)} \leftarrow [K^+] \circlearrowright r^{(i)}$
2. For each $\mathcal{P}_i \in \{\mathcal{P}_1, \dots, \mathcal{P}_{L-1}\}, \mathcal{P}_j \in \{\mathcal{P}_{i+1}, \dots, \mathcal{P}_L\}$:
    (a) $\mathcal{P}_j \leftrightarrow \mathcal{P}_i$: $(\mathbf{\Delta}_i^{(j)}; \mathbf{a}_j^{(i)}, \mathbf{b}_j^{(i)}) \leftarrow \mathsf{TSS.ShrTrns}(\pi^{(j)}; 1^\lambda)$
3. Every $\mathcal{P}_i$ sends $\mathbf{\Delta}_i'^{(j)} \leftarrow \mathbf{a}_{j+1}^{(i)} - \mathbf{b}_j^{(i)}$ to each $\mathcal{P}_j \in \{\mathcal{P}_{i+1}, \dots, \mathcal{P}_{L-1}\}$
4. $\mathcal{P}_1$ sets $\mathbf{\Delta}^{(1)} \leftarrow \mathbf{a}_2^{(1)}$. For $i = 2$ to $L-1$, $\mathcal{P}_i$ sets $\mathbf{\Delta}^{(i)} \leftarrow \sum_{j=1}^{i-1} \mathbf{\Delta}_j^{(i)} + \sum_{i=1}^{\ell-1} \mathbf{\Delta}_j'^{(i)} + \mathbf{a}_{i+1}^{(i)}$. $\mathcal{P}_L$ sets $\mathbf{\Delta}^{(L)} \leftarrow \sum_{j=1}^{L-1} \mathbf{\Delta}_j^{(L)}$
5. $\mathcal{P}_1$ initializes $\mathbf{e} \leftarrow \{0\}^{K^+}$, sets $\mathbf{e}[0] = 1$ and $\mathbf{o}^{(1)} \leftarrow \mathbf{e}$
6. Each $\mathcal{P}_i \in \{\mathcal{P}_1, \dots, \mathcal{P}_{L-1}\}$ sets $\mathbf{b}^{(i)} \leftarrow -\mathbf{b}_L^{(i)}$ and sends $\mathbf{o}^{(i+1)} \leftarrow \pi^{(i)}(\mathbf{o}^{(i)}) + \mathbf{\Delta}^{(i)}$ to $\mathcal{P}_{i+1}$
7. $\mathcal{P}_L$ sets $\mathbf{b}^{(L)} \leftarrow \pi^{(L)}(\mathbf{o}^{(L)}) + \mathbf{\Delta}^{(L)}$

Online phase:
Input: Each $\mathcal{P}_i$ inputs $\mathbf{d}^{(i)} \in \mathbb{Z}_p^N$ as share of aggregated column with $z$-lower bit noise, $N$ preprocessed shares $\{(r_u^{(i)}, \mathbf{b}_u^{(i)})\}_{u \in [N]}$
Output: Each $\mathcal{P}_i$ obtains shares $s^{(i)} \in \mathbb{Z}_p$, $\mathbf{d}'^{(i)} \in \mathbb{Z}_p^N$ s.t. $\sum_{i=1}^{L} \mathbf{d}'^{(i)} = \mathbf{d}'$, $\sum_{i=1}^{L} s^{(i)} = s$, $\mathbf{d}'[u] = ((\sum_{i=1}^{L} \mathbf{d}^{(i)}[u] \ggg z) \overset{?}{=} K)$, $s = \sum_{u=1}^{N} \mathbf{d}'[u]$
1. For each $u \in [N]$ in parallel:
    (a) Every $\mathcal{P}_i$ computes $\hat{r}_u^{(i)} \leftarrow -r_u^{(i)} \pmod{K^+}$ and sends $q_u^{(i)} \leftarrow (\hat{r}_u^{(i)} \lll z) + \mathbf{d}^{(i)}[u] \pmod{p}$ to all other parties
    (b) Every $\mathcal{P}_i$ computes $q_u \leftarrow \left(\sum_{j=1}^{L} q_u^{(j)}\right) \ggg z \pmod{K^+}$, $\mathbf{b}_u'^{(i)} \leftarrow \mathbf{b}_u^{(i)} \circlearrowright q_u$, and $\mathbf{d}'^{(i)}[u] \leftarrow \mathbf{b}_u'^{(i)}[K]$
2. Every $\mathcal{P}_i$ computes $s^{(i)} \leftarrow \sum_{u=1}^{N} \mathbf{b}_u'^{(i)}[K]$
```

server $\mathcal{P}_i$ (step 6). Each server performs KH-PRF evaluation to obtain the shares of the *decrypted* columns (step 7(a)) as:

$$\tilde{\mathbf{d}}_{v_k}^{(i)}[u] = \hat{\mathbf{d}}_{v_k}^{(i)}[u] - F^*(\mathbf{r}_{v_k}^{(i)}, u \,||\, \mathsf{st}_u) \pmod{p} \tag{2}$$

for each $u \in [N]$ and $k \in [K]$, where $\mathsf{st}_{i,u}$ is the (update) counter of document $u$. Finally, each server $\mathcal{P}_i$ aggregates the shares of $K$ decrypted columns (step 7(b)) as $\mathbf{d}^{(i)} \leftarrow \sum_{k=1}^{K} \tilde{\mathbf{d}}_{v_k}^{(i)} \in \mathbb{Z}_p^N$ to obtain the share of the search result according to BF membership verification. Specifically, keyword kw appears in document $u$ of the writer iff $\left(\sum_i \mathbf{d}^{(i)}[u]\right) \ggg z = K$, where $z = \lceil \log_2(e \cdot K) \rceil$ is the reserved space for accumulated error when aggregating $K$ KH-PRF-evaluated columns together by Eq. (1). Since broadcasting $\mathbf{d}^{(i)}$ permits the servers to learn the plain search result leading to result/volume pattern leakages, the next steps are to perform oblivious padding and oblivious shuffle on the shares $\mathbf{d}^{(i)}$ before opening the search result.

### 4.2.2 Oblivious padding

To seal the volume leakage, we perform oblivious padding so that the search result always returns $n_s$ document identifiers. We first count $s$ as the number of occurrences of $K$ in the aggregated column $\mathbf{d}$ (Figure 2, step 8), and then create a padding vector containing $(n_s - s)$ elements of value $K$.

**Oblivious count protocol.** Given a vector $\mathbf{v} \in \mathbb{Z}_m^N$, counting how many elements in $\mathbf{v}$ equal a specific value $x \in \mathbb{Z}_m$ can be done by transforming each $\mathbf{v}[u]$ to a one-hot vector $\mathbf{b}_u \in \{0,1\}^m$, where $\mathbf{b}_u[\mathbf{v}[u]] = 1$, and computing $\sum_{u=1}^{N} \mathbf{b}_u[x]$. With this unit-vectorization trick, we design an online/offline $L$-party oblivious counting algorithm (Protocol 1) to compute $s$ (i.e., number of elements in the noise-free aggregated column $\mathbf{v} \in \mathbb{Z}_m^N$ equal $x$) in arithmetic secret-sharing. We present a toy example of our protocol in Figure 3.

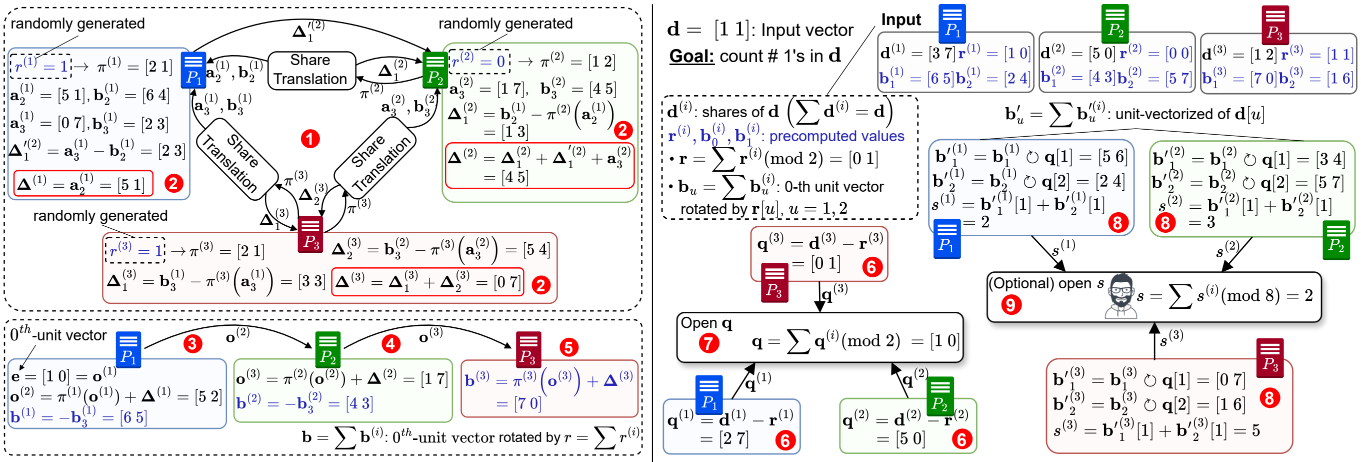
Suppose $p = 2^\alpha$, $K^+ = K + 1 = 2^{\alpha'}$ for some integers $\alpha > \alpha'$ and $s < p$ (e.g., $p = 8, K = 1$ in Figure 3). In the preprocessing phase (Figure 3a), we precompute the share of $N$ random roulette pairs $(\mathbf{b}_u, r_u)$, where $\mathbf{b}_u \in \{0,1\}^{K^+}$ is a random unit vector that $\mathbf{b}_u[r_u] = 1$, and $r_u \in \mathbb{Z}_{K^+}$ is a random circular shift. Let $\mathbf{b}_u^{(i)} \in \mathbb{Z}_p^{K^+}$ and $r_u^{(i)} \in \mathbb{Z}_{K^+}$ be the share of $\mathbf{b}_u$ and $r_u$, respectively. In the online phase (Figure 3b), the share of $s$ is computed by masking each element $\mathbf{d}[u]$ with $\hat{r}_u^{(i)}$, where $\hat{r}_u^{(i)} \leftarrow -r_u^{(i)} \pmod{K^+}$ is the complement of circular shift value $r_u^{(i)}$ (step ❻), followed by opening the masked data (step ❼), and unmasking with the unit vector $\mathbf{b}_u$ (step ❽).

As $z$ least significant bits (LSBs) of $\mathbf{d}[u]$ contain KH-PRF error (Eq. (1)), we left-shift the circular shift value $\hat{r}_u^{(i)}$ by $z$ bits to only mask the actual data in $\mathbf{d}[u]$ as $q_u^{(i)} = \mathbf{d}^{(i)}[u] + \hat{r}_u^{(i)} \cdot 2^z \in \mathbb{Z}_p$. We then open and remove $z$ LSBs of the masked data as $q_u = \left(\sum_{i=1}^{L} q_u^{(i)}\right) \ggg z \in \mathbb{Z}_{K^+}$. Finally, the share of $s$ is computed by unmasking the opened data with the share of $\mathbf{b}_u$ as $s^{(i)} = \sum_{u=1}^{N} \mathbf{b}'^{(i)}_u[K] \in \mathbb{Z}_p$, where $\mathbf{b}'^{(i)}_u = \mathbf{b}_u^{(i)} \circlearrowright q_u$ (step ❽). Moreover, as $\mathbf{d}[u] \ggg z$ contains aggregated values from 1 to $K$, it may reveal information about the search pattern when opened in the next search operation (i.e., oblivious shuffle). Meanwhile, the search only needs the information whether $\mathbf{d}[u] \ggg z \overset{?}{=} K$, which is available in $\mathbf{b}'_u[K]$. Thus, our online protocol also outputs the shares of $\mathbf{b}'_u[K]$ in the form of shared vectors $\mathbf{d}'^{(i)}[u] = \mathbf{b}'_u[K]$ for the next processing.

We now show how to compute a roulette pair $(\mathbf{b}, r)$ in the offline phase. Our idea is to employ the two-party shuffle in [21] and extend it to $L$-party setting. As $\mathbf{b}[r] = 1$, it can be written as linear group operations as $\mathbf{b} = \pi^{(L)}(\dots(\pi^{(1)}(\mathbf{e}))$, where $\mathbf{e} \in \mathbb{Z}_2^{K^+}$ is the unit-vector that $\mathbf{e}[0] = 1$, $\pi^{(i)} = [K+1] \circlearrowright r^{(i)}$ is a random circular shift permutation with $r^{(i)} \xleftarrow{\$} \mathbb{Z}_{K^+}$ and $r = \sum_{i=1}^{L} r^{(i)} \in \mathbb{Z}_{K^+}$. Thus, $L$ shares of $\mathbf{b}$ can be set as $\mathbf{b}^{(i)} \xleftarrow{\$} \mathbb{Z}_p^{K^+}$ for $i \in [L^-]$ and $\mathbf{b}^{(L)} = \pi^{(L)}(\dots(\pi^{(1)}(\mathbf{e})) - \sum_{i=1}^{L-1} \mathbf{b}^{(i)}$. That means $L-1$ parties $\mathcal{P}_i \in \{\mathcal{P}_1, \dots, \mathcal{P}_{L-1}\}$ can independently generate random circular shift and vector as the shares $r^{(i)}$ and $\mathbf{b}^{(i)}$, respectively, and interact with each other to help party $\mathcal{P}_L$ obtain $\mathbf{b}^{(L)} = \pi^{(L)}(\dots(\pi^{(1)}(\mathbf{e})) - \sum_{i=1}^{L-1} \mathbf{b}^{(i)}$ with its chosen random circular shift permutation $\pi^{(L)}$. Specifically, each party $\mathcal{P}_i$ computes a translation function $\mathbf{\Delta}^{(i)}$ such that

$$\pi^{(L)}(\dots(\pi^{(2)}(\mathbf{\Delta}^{(1)}) + \mathbf{\Delta}^{(2)})\dots) + \mathbf{\Delta}^{(L)} = \pi^{(L)}(\dots(\pi^{(2)}(\pi^{(1)}(\mathbf{e})))\dots) - \sum_{i=1}^{L-1} \mathbf{b}^{(i)}$$

The idea is to execute the two-party share translation protocol in [21] between $(\mathcal{P}_i, \mathcal{P}_j)$ for $i, j \in [L]$ and $i < j$ (with circular



**(a)** Preprocessing ($\Pi_{\mathsf{LOC}}.\mathsf{Prep}$)

**(b)** Online phase ($\Pi_{\mathsf{LOC}}.\mathsf{Cnt}$)

**Figure 3:** A numerical example of $\Pi_{\mathsf{LOC}}$ with $p=8$ and $K=1$.

permutation $\pi^{(j)}$ chosen by $\mathcal{P}_j$, and random vectors $\mathbf{a}_j^{(i)}, \mathbf{b}_j^{(i)}$ chosen by $\mathcal{P}_i$) to generate $\mathbf{\Delta}_i^{(j)} = \mathbf{b}_j^{(i)} - \pi^{(j)}(\mathbf{a}_j^{(i)})$ for $\mathcal{P}_j$ (step ① in Figure 3a). Then, each $\mathcal{P}_i$, for $\mathcal{P}_i \in \{\mathcal{P}_1,\dots,\mathcal{P}_{j-1}\}$, sends $\mathbf{\Delta}_i'^{(j)} \leftarrow \mathbf{a}_{j+1}^{(i)} - \mathbf{b}_j^{(i)}$ to $\mathcal{P}_j$, for $\mathcal{P}_j \in \{\mathcal{P}_{i+1},\dots,\mathcal{P}_{L-1}\}$. From $\mathbf{\Delta}_j^{(i)}$ and $\mathbf{\Delta}_j'^{(i)}$, each $\mathcal{P}_i \in \{\mathcal{P}_1,\dots,\mathcal{P}_L\}$ can compute $\mathbf{\Delta}^{(i)} = \mathbf{x}_{i+1} - \pi^{(i)}(\mathbf{x}_i)$ (step ②), where

$$\mathbf{x}_i = \begin{cases} \sum_{j=1}^{L-1} \mathbf{b}_L^{(j)} & \text{if } i = L+1 \\ \sum_{j=1}^{i-1} \mathbf{a}_i^{(j)} & \text{otherwise} \end{cases}$$

Next, $\mathcal{P}_1$ initializes the unit-vector $\mathbf{e}$ and sends $\mathbf{o}^{(2)} \leftarrow \pi^{(1)}(\mathbf{e}) + \mathbf{\Delta}^{(1)} = \pi^{(1)}(\mathbf{e}) + \mathbf{a}_2^{(1)}$ to $\mathcal{P}_2$ (step ③), who then computes $\mathbf{o}^{(3)} \leftarrow \pi^{(2)}(\mathbf{o}^{(2)}) + \mathbf{\Delta}^{(2)} = \pi^{(2)}(\pi^{(1)}(\mathbf{e})) + \sum_{i=1}^{2} \mathbf{a}_3^{(i)}$ (step ④) and forwards it to $\mathcal{P}_3$, and so on until the final party $\mathcal{P}_L$ who receives $\mathbf{o}^{(L)} = \pi^{(L-1)}(\dots(\pi^{(1)}(\mathbf{e}))\dots) + \sum_{i=1}^{L-1} \mathbf{a}_L^{(i)}$ from $\mathcal{P}_{L-1}$. Then, $\mathcal{P}_L$ computes the share as $\mathbf{b}^{(L)} \leftarrow \pi^{(L)}(\mathbf{o}^{(L)}) + \mathbf{\Delta}^{(L)} = \pi^{(L)}(\dots(\pi^{(1)}(\mathbf{e}))\dots) + \sum_{i=1}^{L-1} \mathbf{b}_L^{(i)}$ (step ⑤).

We argue correctness/security of $\Pi_{\mathsf{LOC}}$ in Appendix B.

**Oblivious padding.** Once the count $s$ in the aggregated column is computed, the oblivious padding can be easily achieved. All servers send $s^{(i)}$ to the reader to open $s = \sum_{i=1}^{L} s^{(i)}$ (Figure 2, step 9(a)). The reader generates a padding vector $\mathbf{p} \in \{0,1\}^{n_s}$ containing $n_s - s$ elements of 1 and secret-shares it with $L$ servers (steps 9(b)-9(d)). The servers form a concatenated vector from $\mathbf{p}$ and the vector $\mathbf{d}'$ output from the $\Pi_{\mathsf{LOC}}$ online protocol as $\mathbf{o} = (\mathbf{d}'||\mathbf{p})$ (step 10). To completely seal pattern leakages, $\mathbf{o}$ must be shuffled before opening to obtain the search result. In the next section, we present an oblivious shuffle protocol to shuffle $\mathbf{o}$ in $L$-party arithmetic secret-sharing.

### 4.2.3 Oblivious shuffle

We construct a new $L$-party random shuffle protocol to shuffle vector elements in arithmetic secret sharing.

***L*-party secret-shared shuffle.** Protocol 2 presents our $L$-party shuffle $\Pi_{\mathsf{LOS}}$ scheme. Similar to $\Pi_{\mathsf{LOC}}$, $\Pi_{\mathsf{LOS}}$ is extended from the two-party shuffle in [21] to work with $L$-party and arithmetic shares. However, unlike $\Pi_{\mathsf{LOC}}$ or prior works [34], the goal of $\Pi_{\mathsf{LOS}}$ is to output the shuffled vector directly, instead of its shares which may cost an additional communication round for opening. To achieve this, we design a new preprocessing protocol that permits all parties to compute precomputed materials for direct online shuffle. Specifically, we precompute translation functions for every $\mathcal{P}_i$ (except $\mathcal{P}_L$) with its chosen random permutation $\pi^{(i)}$ as $\mathbf{\Delta}^{(i)} = \mathbf{x}_{i+1} - \pi^{(i)}(\mathbf{x}_i)$ such that

$$\mathbf{x}_i = \begin{cases} \sum_{j=i+1}^{L} \mathbf{a}_i^{(j)}, & \text{if } i = 1 \\ \sum_{j=i}^{L} \mathbf{b}_{i-1}^{(j)}, & \text{otherwise} \end{cases}$$

where $\mathbf{a}_i^{(j)}, \mathbf{b}_i^{(j)}$ are random vectors chosen by $\mathcal{P}_j$.

With these precomputed materials, our online protocol can output the shuffled vector directly as follows. Let $\mathbf{d}^{(i)}$ be the share of the vector to be shuffled. Each $\mathcal{P}_i \in \{\mathcal{P}_2,\dots,\mathcal{P}_L\}$ sends its masked share $\mathbf{z}^{(i)} \leftarrow \mathbf{d}^{(i)} + \mathbf{a}_1^{(i)}$ to $\mathcal{P}_1$. $\mathcal{P}_1$ computes $\mathbf{o}^{(1)} \leftarrow \mathbf{d}^{(1)} + \sum_{i=2}^{L} \mathbf{z}^{(i)}$ sends $\mathbf{o}^{(2)} \leftarrow \pi^{(1)}(\mathbf{o}^{(1)}) + \mathbf{\Delta}^{(1)} = \pi^{(1)}(\mathbf{d}) + \sum_{i=2}^{L} \mathbf{b}_1^{(i)}$ to $\mathcal{P}_2$, which in turn computes $\mathbf{o}^{(3)} \leftarrow \pi^{(2)}(\mathbf{o}^{(2)}) + \mathbf{\Delta}^{(2)} = \pi^{(2)}(\pi^{(1)}(\mathbf{d})) + \sum_{i=3}^{L} \mathbf{b}_2^{(i)}$ and forwards it to $\mathcal{P}_3$ and so on until $\mathcal{P}_L$ receives $\mathbf{o}^{(L)} = \pi^{(L-1)}(\dots(\pi^{(1)}(\mathbf{d}))\dots) + \mathbf{b}_{L-1}^{(L)}$ from $\mathcal{P}_{L-1}$. As $\mathcal{P}_L$ holds $\mathbf{b}_{L-1}^{(L)}$, it can compute $\mathbf{r} \leftarrow \mathbf{o}^{(L)} - \mathbf{b}_{L-1}^{(L)} = \pi^{(L-1)}(\dots(\pi^{(1)}(\mathbf{d})))$.

To generate precomputed materials, we execute two-party share translation scheme in [21] between $(\mathcal{P}_i, \mathcal{P}_j)$ for $i, j \in [L]$ and $i < j$ (with random permutation $\pi^{(i)}$ chosen by $\mathcal{P}_i$, and random vectors $\mathbf{a}_i^{(j)}$, $\mathbf{b}_i^{(j)}$ chosen by $\mathcal{P}_j$) to compute

```
Protocol 2. $\Pi_{\mathsf{LOS}}$ – Multi-party Oblivious Shuffle Protocol

Preprocessing:
Input: Security parameter $\lambda$ and size $n$
Output: Preprocessing values, including $(\pi^{(1)}, \mathbf{\Delta}_1)$ for $\mathcal{P}_1$, $(\pi^{(i)}, \mathbf{\Delta}_i, \mathbf{a}_i^{(1)})$ for $\mathcal{P}_i$, with $i = 2, \ldots, L-1$, and $\mathbf{a}_L^{(1)}, \mathbf{b}_L^{(L-1)}$ for $\mathcal{P}_L$
1. For each $\mathcal{P}_i \in \{\mathcal{P}_1, \ldots, \mathcal{P}_{L-1}\}$, $\mathcal{P}_j \in \{\mathcal{P}_{i+1}, \ldots, \mathcal{P}_L\}$:
   (a) $\mathcal{P}_i$ generates a random $\pi^{(i)}$ for $n$ elements
   (b) $\mathcal{P}_i \leftrightarrow \mathcal{P}_j$: $(\mathbf{\Delta}_j^{(i)}; \mathbf{a}_i^{(j)}, \mathbf{b}_i^{(j)}) \leftarrow \mathsf{TSS.ShrTrns}(\pi^{(i)}; 1^\lambda)$
2. Each $\mathcal{P}_i \in \{\mathcal{P}_3, \ldots, \mathcal{P}_L\}$ sends $\mathbf{\Delta}_i'^{(j)} \leftarrow \mathbf{b}_{j-1}^{(i)} - \mathbf{a}_j^{(i)}$ to each $\mathcal{P}_j \in \{\mathcal{P}_2, \ldots, \mathcal{P}_{i-1}\}$. Each $\mathcal{P}_i \in \{\mathcal{P}_1, \ldots, \mathcal{P}_{L-1}\}$ computes $\mathbf{\Delta}^{(i)} \leftarrow \sum_{j=i+1}^{L} \mathbf{\Delta}_j^{(i)}$.
   Each $\mathcal{P}_i \in \{\mathcal{P}_2, \ldots, \mathcal{P}_L\}$ computes $\mathbf{\Delta}^{(i)} \leftarrow \mathbf{\Delta}^{(i)} - \pi^{(i)}(\sum_{j=i+1}^{L} \mathbf{\Delta}_j'^{(i)} + \mathbf{b}_{i-1}^{(i)})$

Online phase:
Input: Secret shares $\mathbf{d}^{(i)}$ and preprocessing values including $(\pi^{(1)}, \mathbf{\Delta}^{(1)})$ for $\mathcal{P}_1$, $(\pi^{(i)}, \mathbf{\Delta}^{(i)}, \mathbf{a}_1^{(i)})$ for $\mathcal{P}_i$, with $i = 2, \ldots, L-1$, and $\mathbf{a}_1^{(L)}, \mathbf{b}_{L-1}^{(L)}$ for $\mathcal{P}_L$
Output: $\pi^{(i)}$ for each $\mathcal{P}_i \in \{\mathcal{P}_1, \ldots, \mathcal{P}_{L-1}\}$, shuffled vector $\mathbf{r}$ for $\mathcal{P}_L$
1. Each $\mathcal{P}_i \in \{\mathcal{P}_2, \ldots, \mathcal{P}_L\}$ sends $\mathbf{z}^{(i)} \leftarrow \mathbf{d}^{(i)} + \mathbf{a}_1^{(i)}$ to $\mathcal{P}_1$
2. $\mathcal{P}_1$ computes $\mathbf{o}^{(1)} \leftarrow \mathbf{d}^{(1)} + \sum_{i=2}^{L} \mathbf{z}^{(i)}$
3. Each $\mathcal{P}_i \in \{\mathcal{P}_1, \ldots, \mathcal{P}_{L-1}\}$ sends $\mathbf{o}^{(i+1)} \leftarrow \pi^{(i)}(\mathbf{o}^{(i)}) + \mathbf{\Delta}^{(i)}$ to $\mathcal{P}_{i+1}$
4. $\mathcal{P}_L$ computes $\mathbf{r} \leftarrow \mathbf{o}^{(L)} - \mathbf{b}_{L-1}^{(L)}$
```

$\mathbf{\Delta}_j^{(i)} = \mathbf{b}_i^{(j)} - \pi^{(i)}(\mathbf{a}_i^{(j)})$ for $\mathcal{P}_i$. Then, each $\mathcal{P}_i \in \{\mathcal{P}_2, \ldots, \mathcal{P}_{L-1}\}$ receives $\mathbf{\Delta}_j'^{(i)} = \mathbf{b}_{i-1}^{(j)} - \mathbf{a}_i^{(j)}$ from each $\mathcal{P}_j \in \{\mathcal{P}_{i+1}, \ldots, \mathcal{P}_L\}$. From $\mathbf{\Delta}_j^{(i)}$ and $\mathbf{\Delta}_j'^{(i)}$, $\mathcal{P}_i \in \{\mathcal{P}_1, \ldots, \mathcal{P}_{L-1}\}$ can compute $\mathbf{\Delta}^{(i)}$.

We argue correctness/security of $\Pi_{\mathsf{LOS}}$ in Appendix C.

**Obtaining final search result.** Once the concatenated search vector $\mathbf{o}$ is randomly shuffled (Figure 2, step 11) and opened to $\mathcal{P}_L$ as $\hat{\mathbf{o}}$, it can extract $n_s$ indices $u$ such that $\hat{\mathbf{o}}[u] = 1$ and send them to the reader. All other servers send their permutations to the reader. The reader obtains the final search result by applying the permutation inverses on indices from $\mathcal{P}_L$ and removing padded indices (step 12).

## 4.3 Search Permission Revocation

MUSES permits a writer to revoke access permission of the reader on her search index. The idea is to re-encrypt the writer's index with refreshed (column) KH-PRF keys unknown to the reader. Figure 4 presents our revocation protocol, where the re-encryption operation is delegated securely to the servers for writer efficiency. Its high-level idea is as follows.

To re-encrypt $\mathsf{EIDX}_w$, the writer $w$ first parses the current secret column keys $\mathbf{r}_{w,v}$, for $v \in [m]$ (step 1(a)) and generates new secret column keys $\mathbf{r}'_{w,v}$ (step 1(b).i). Next, the writer creates secret-shares of these keys as $(\mathbf{r}_{w,v}^{(1)}, \ldots, \mathbf{r}_{w,v}^{(L)})$, $(\mathbf{r}_{w,v}'^{(1)}, \ldots, \mathbf{r}_{w,v}'^{(L)})$ (steps 1(b).ii–iii), then updates private token (step 1(c)) and sends $\{\mathbf{r}_{w,v}^{(i)}, \mathbf{r}_{w,v}'^{(i)}\}_{v \in [m]}$ to servers $\mathcal{P}_i \in \{\mathcal{P}_1, \ldots, \mathcal{P}_L\}$ (step 2).

Each server $\mathcal{P}_i$ computes $\mathbf{T}_1^{(i)}$ and $\mathbf{T}_2^{(i)}$, where $\mathbf{T}_1^{(i)}$ is the masked component to remove the shared encryption computed by the secret-shared key $\mathbf{r}_{w,v}^{(i)}$, and $\mathbf{T}_2^{(i)}$ is the component to unmask the value $\mathbf{M}^{(i)}$ added by $\mathbf{T}_1^{(i)}$ and add the shared encryption computed by the new secret-shared key $\mathbf{r}_{w,v}'^{(i)}$ (step 3(a)). The server $\mathcal{P}_i$ then distributes $\mathbf{T}_1^{(i)}$ and $\mathbf{T}_2^{(i)}$ to other servers (step 4). Next, each server $\mathcal{P}_i$ obtains the masked value with error denoted as $\mathbf{T}'$ (step 5(a).i), where $\mathbf{T}'[u,v] = (\mathsf{IDX}[u,v] \lll z) + \sum_{j=1}^{L} \mathbf{M}^{(j)}[u,v] + e_{u,v}$. The random mask $\mathbf{M}^{(j)}$ generated by each server $\mathcal{P}_j$ is to hide the plaintext data $(\mathsf{IDX}[u,v] \lll z)$ when the servers remove the current encryption by adding EIDX with $\sum_{j=1}^{L} \mathbf{T}_1^{(j)}$ to obtain $\mathbf{T}'$. By clearing $z$ lower bits of each $\mathbf{T}'[u,v]$, the error part $e_{u,v}$ can be removed, and the servers now hold the masked plaintext value $\hat{\mathbf{T}}[u,v]$ (step 5(a).ii). To retrieve the final $\mathsf{EIDX}'_w$ encrypted by the new secret keys, each server $\mathcal{P}_i$ computes $\mathsf{EIDX}'_w[u,v]$ based on $\hat{\mathbf{T}}[u,v]$ and $\sum_{j=1}^{L} \mathbf{T}_2^{(j)}[u,v]$ (step 5(a).iii). The value $\mathsf{max}_e = L$ is the max error when using LWR-based KH-PRF. It is necessary for decryption in keyword search later. Finally, all servers hold the same updated $\mathsf{EIDX}'_w$, which is encrypted with the new column keys.

```
$\mathsf{RvkPrm}(w, \mathsf{PTkn}_w, \mathsf{EIDX}_w, \mathsf{st}_w)$:
1. Writer $w$:
   (a) parse $\mathsf{PTkn}_w = (\mathbf{r}_{w,1}, \mathbf{r}_{w,2}, \ldots, \mathbf{r}_{w,m})$
   (b) For $v = 1$ to $m$
       i. $\mathbf{r}'_{w,v} \leftarrow \mathsf{KH\text{-}PRF.Gen}(1^\lambda)$
       ii. $(\mathbf{r}_{w,v}^{(1)}, \ldots, \mathbf{r}_{w,v}^{(L)}) \leftarrow \mathsf{KH\text{-}PRF.Share}(\mathbf{r}_{w,v})$
       iii. $(\mathbf{r}_{w,v}'^{(1)}, \ldots, \mathbf{r}_{w,v}'^{(L)}) \leftarrow \mathsf{KH\text{-}PRF.Share}(\mathbf{r}'_{w,v})$
   (c) $\mathsf{PTkn}'_w \leftarrow (\mathbf{r}'_{w,1}, \mathbf{r}'_{w,2}, \ldots, \mathbf{r}'_{w,m})$
2. Writer $w \rightarrow$ Server $\mathcal{P}_i \in \{\mathcal{P}_1, \ldots, \mathcal{P}_L\}$: $\{\mathbf{r}_{w,v}^{(i)}, \mathbf{r}_{w,v}'^{(i)}\}_{v \in [m]}$
3. Server $\mathcal{P}_i \in \{\mathcal{P}_1, \ldots, \mathcal{P}_L\}$:
   (a) For $u = 1$ to $N$, $v = 1$ to $m$ do
       i. $x_{u,v} \xleftarrow{\$} \mathbb{Z}_p$, $\mathbf{M}^{(i)}[u,v] \leftarrow x_{u,v} \lll z$
       ii. $\mathbf{T}_1^{(i)}[u,v] \leftarrow \mathbf{M}^{(i)}[u,v] - F^*(\mathbf{r}_{w,v}^{(i)}, u || \mathsf{st}_{w,u}) + \mathsf{max}_e \pmod{p}$
       iii. $\mathbf{T}_2^{(i)}[u,v] \leftarrow -\mathbf{M}^{(i)}[u,v] + F^*(\mathbf{r}_{w,v}'^{(i)}, u || \mathsf{st}_{w,u}) \pmod{p}$
4. Server $\mathcal{P}_i \rightarrow$ Server $\mathcal{P}_j$: $\mathbf{T}_1^{(i)}, \mathbf{T}_2^{(i)}$, for $\mathcal{P}_i, \mathcal{P}_j \in \{\mathcal{P}_1, \ldots, \mathcal{P}_L\}$ and $\mathcal{P}_i \neq \mathcal{P}_j$
5. Server $\mathcal{P}_i \in \{\mathcal{P}_1, \ldots, \mathcal{P}_L\}$:
   (a) For $u = 1$ to $N$, $v = 1$ to $m$ do
       i. $\mathbf{T}'[u,v] \leftarrow \mathsf{EIDX}_w[u,v] + \sum_{j=1}^{L} \mathbf{T}_1^{(j)}[u,v]$
       ii. $\hat{\mathbf{T}}[u,v] \leftarrow ((\mathbf{T}'[u,v] \ggg z) \lll z)$
       iii. $\mathsf{EIDX}'_w[u,v] \leftarrow \hat{\mathbf{T}}[u,v] + \sum_{j=1}^{L} \mathbf{T}_2^{(j)}[u,v] + \mathsf{max}_e \pmod{p}$
6. return $(\mathsf{EIDX}'_w, \mathsf{PTkn}'_w)$
```

**Figure 4:** MUSES permission revocation.

## 4.4 Document Update

MUSES supports document update as other bitmap-based dynamic encrypted search schemes (e.g., [28, 47]). We present the update procedure of MUSES in Figure 5. Given an updated document with identifier $u$ and a set of its keywords $\mathcal{V}_u$, the writer parses KH-PRF keys $\mathbf{r}_{w,v}$, for $v \in [m]$, from its private token $\mathsf{PTkn}_w$ (step 1). The writer computes the new BF representation $\mathbf{u} \in \{0,1\}^m$ of the updated document with input keywords $\mathcal{V}_u$ (steps 2–3). The writer encrypts $\mathbf{u}$ with KH-PRF keys and the incremented counter value as $\mathbf{u}'[v] \leftarrow \mathbf{u}[v] + F^*(\mathbf{r}_v, u \,||\, (\mathsf{st}_{w,u} + 1)) \pmod{p}$, for each $v \in [m]$ (step 4). Finally, the writer sends $\mathfrak{u} = (w, u, \mathbf{u}')$ as the update token to the servers to update its search index accordingly as

```
$\mathsf{UpdateToken}(\mathcal{V}_u, u, w, \mathsf{PTkn}_w, \mathsf{st}_w)$:        $\triangleright$ Executed by writer $w$
1. parse $\mathsf{PTkn}_w = (\mathbf{r}_{w,1}, \mathbf{r}_{w,2}, \ldots, \mathbf{r}_{w,m})$
2. $\mathbf{u} \leftarrow \{0\}^m$, $\mathbf{u}' \leftarrow \{0\}^m$
3. For $\mathsf{kw}_j \in \mathcal{V}_u$, $k = 1$ to $K$: $\mathsf{cid}_{j,k} \leftarrow H_k(\mathsf{kw}_j)$, $\mathbf{u}[\mathsf{cid}_{j,k}] \leftarrow 1$
4. For $v \in [m]$: $\mathbf{u}'[v] \leftarrow (\mathbf{u}[v] \lll z) + F^*(\mathbf{r}_{w,v}, u || (\mathsf{st}_{w,u} + 1)) \pmod p$
5. return $\mathfrak{u}_w \leftarrow (w, u, \mathbf{u}')$

$\mathsf{Update}(\mathfrak{u}_w, \mathsf{EIDX}_w, \mathsf{st}_w)$:        $\triangleright$ Executed by each server $\mathcal{P}_i \in \{\mathcal{P}_1, \ldots, \mathcal{P}_L\}$:
6. parse $\mathfrak{u}_w = (w, u, \mathbf{u}')$
7. $\mathsf{EIDX}'_w[u, *] \leftarrow \mathbf{u}'$, $\mathsf{st}'_{w,u} \leftarrow \mathsf{st}_{w,u} + 1$
8. return $(\mathsf{EIDX}'_w, \mathsf{st}'_w)$
```

**Figure 5:** MUSES document update.

$\mathsf{EIDX}'_w[u, *] \leftarrow \mathbf{u}'$ and $\mathsf{st}'_{w,u} \leftarrow \mathsf{st}_{w,u} + 1$ (steps 6–7).

## 5 Analysis

**Complexity.** We analyze the online asymptotic cost of MUSES. We consider the number of servers ($L$) and BF parameters ($m, K$) as small constants. Let $N$ be the number of documents. To search for a keyword kw in a writer's database, the reader creates a query of size $O(K \cdot \lambda \cdot \tau) = O(\lambda\tau)$, where $\tau = O(\log m)$ for $L = 2$, and $\tau = O(\sqrt{m})$ for $L \geq 3$. For each writer in $\mathcal{W}'$, the reader sends shares of KH-PRF keys to $L$ servers, which costs $O(L \cdot K \cdot n \cdot \log q) = O(\lambda)$ in total (as $n, q = O(\lambda)$ are the LWR parameters of KH-PRF). Let $n_s$ be the bound on the size of the search output. For oblivious padding, the reader receives secret shares of the count from servers, then creates secret shares of padding values with communication and computation costs are $O(L \cdot n_s) = O(n_s)$. To obtain the search result, the reader receives PRP seeds from $L-1$ servers, and shuffled output from $\mathcal{P}_L$, then re-generates the permutations $\pi^{(1)}, \ldots, \pi^{(L-1)}$, and reverses permutations to obtain the final search output, which incurs $O((L-1)\lambda + n_s) = O(\lambda + n_s)$ communication and $O((L-1) \cdot N) = O(N)$ computation cost. The overall reader's bandwidth cost is $O(\lambda\tau + |\mathcal{W}'|(\lambda + n_s))$.

For keyword search, each server incurs $(K \cdot N \cdot m) = O(N \cdot m)$ modulo additions and multiplications for retrieval. Each server performs $O(K \cdot N) = O(N)$ additions for KH-PRF evaluation, $O(N)$ additions and circular shift operations for oblivious counting, and $O(N + n_s)$ additions for oblivious shuffle. The overall server computation cost per search on the writer set $\mathcal{W}'$ is $O(|\mathcal{W}'| \cdot (N \cdot m + N + n_s)) = O(|\mathcal{W}'| \cdot N \cdot m)$.

To update a document, the writer creates a new BF representation of size $O(m)$ and re-encrypts it. Thus, the total writer's bandwidth cost and computation cost per document update are both $O(m)$. The server does not incur any computation other than replacing the writer's components (e.g., a row in the search index and a state value).

For permission revocation, the writer's bandwidth and computation cost are similar to the corresponding overhead in document update, which is $O(m)$ for both. Since the servers are responsible for updating the encrypted search index with the new secret keys on behalf of the writer, the computation and inter-server communication costs are $O(N \cdot m)$.

For storage, the reader and each writer store a private/secret key of size $O(\lambda)$. Although the index is encrypted by $O(m)$ KH-PRF keys, the writer does not need to store these keys separately for update/revocation as they can be regenerated from a (master) secret key using a key derivation function. The update state st is public and, therefore, can be maintained at the server and retrieved when needed. For each writer, the servers store an index of size $O(\log p \cdot N \cdot m) = O(N \cdot m)$, a state st of size $O(\lambda \cdot N)$, shared tokens $\mathsf{STkn}$ of size $O(m \cdot n \cdot \log q) = O(m \cdot \lambda)$. The total server cost is $O(n_w \cdot N \cdot m + n_w \cdot \lambda \cdot N + n_w \cdot m \cdot \lambda)$, where $n_w$ is the number of writers.

**Security.** We state the security of MUSES as follows.

**Theorem 1.** *Assuming that the adversary corrupts at most $L-1$ out of the $L$ servers and some writers, MUSES hides all pattern leakages during search by Definition 2, achieves $\mathcal{L}_{\mathcal{H}}$-adaptive security by Definition 3 and forward and backward privacy by Definition 4, where $\mathcal{L}_{\mathcal{H}}^{\mathsf{Setup}}(1^\lambda) = \{w, N, m\}_{w \in [n_w]}$, $\mathcal{L}_{\mathcal{H}}^{\mathsf{CorruptWriter}}(w) = \{\mathsf{UpdateBy}(w)\}$, $\mathcal{L}_{\mathcal{H}}^{\mathsf{Search}}(\mathsf{kw}, \mathcal{W}') = \{\emptyset\}$, $\mathcal{L}_{\mathcal{H}}^{\mathsf{Update}}(w, u, \mathcal{V}_u) = \{w, \mathsf{up}(u)\}$, and $\mathcal{L}_{\mathcal{H}}^{\mathsf{Revoke}}(w) = \{w\}$, where $\mathcal{W}'$ is a writer subset.*

We present the proof in Appendix D.

## 6 Experimental Evaluation

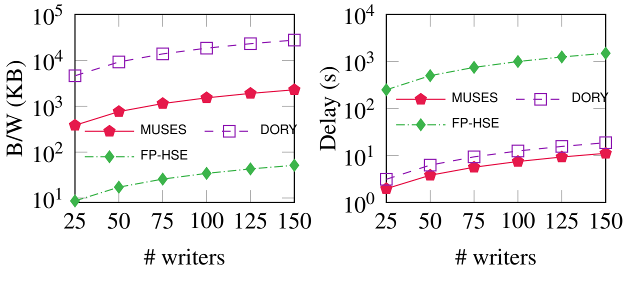
**Implementation.** We fully implemented all our proposed techniques including MUSES, $\Pi_{\mathsf{LOC}}$ and $\Pi_{\mathsf{LOS}}$ in `C++` consisting of approximately 2,500 lines of code. We used standard cryptographic libraries, including `OpenSSL` [1] for IND-CPA encryption and hash functions, `libsecp256k1` [88] for public-key encryption in our scheme, and `EMP-Toolkit` [86] for IKNP OT protocol. We implemented KH-PRF from scratch. We used `libzeromq` [2] for network communication between servers and client. Our implementation is available and ready for public release. Our source code is available at: `https://github.com/vt-asaplab/MUSES`.

**Hardware and network.** We used EC2 `r5n.4xlarge` instances with 8-core Intel Xeon Platinum 8375C CPU @ 2.90 GHz and 128 GB memory as servers. For the user, we used a laptop with an Intel i7-6820HQ CPU @ 2.7 GHz and 16 GB RAM. The bandwidth between servers is 3 Gbps and the client bandwidth is 20 Mbps with 10ms RTT.

**Dataset.** We used the Enron email dataset [3] which includes about 500K emails of 150 employees. We extracted unique keywords using the standard tokenization method as described in [28]. Each email has an average of 73.18 keywords. The average number of keywords in each writer database is 11,017.

**Counterparts and parameters selection.** We compared MUSES with the state-of-the-art schemes including FP-HSE [84] and DORY [28]. We selected their parameters as follows.

- **MUSES**: For KH-PRF, we selected $q = 2^{13}$, $p = 2^{10}$, $n = 256, l = 2$ as suggested in [32] for secure LWR with 126-



**(a)** Reader's bandwidth **(b)** E2E delay

**Figure 6:** Keyword search performance (log scale on y-axis).

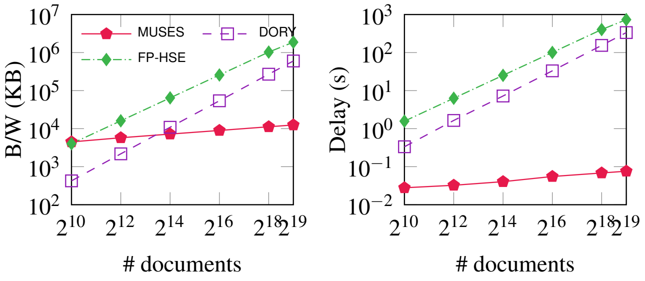
bit security, where each KH-PRF key is of 1 KB. We used SHA-256 for the hash function. We used 256-bit keys for IND-CPA encryption and PRF/PRG seeds. Each folder in the dataset is considered as a writer. We created a reader that can perform a search over all databases for experiments. To cover the largest folder in the dataset which contains $28,229$ documents, we let $N = 2^{15}$ be the bound on the number of documents. We chose BF parameters such that $N \times \text{FP rate} < 1$. For $K = 7$, we chose $m = 2000$ to achieve FP rate $\approx 3e^{-5}$. To evaluate the permission revocation, we used a varied number of documents $N$ from $2^{10}$ to $2^{19} (\approx 500K)$ with the corresponding $m$ from 1120 to 3120 (with $K = 7$) to achieve a low FP rate.

- **FP-HSE** [84]: We selected the original parameters with a 96-bit security level, where PRFs and keyed hash functions are instantiated with `HMAC-SHA-256`, and `MNT224` curve for pairings. Each folder in the dataset is a separate writer.
- **DORY** [28]. We run experiments with DORY in the semi-honest setting similar to our MUSES and FP-HSE. We configured BF parameters of DORY similar to our scheme as DORY uses DPF-based PIR scheme for oblivious search, and 256-bit keys for IND-CPA encryption, and PRG seeds.

## 6.1 Overall Results

### 6.1.1 Keyword search

**Reader's bandwidth.** Figure 6a shows the search bandwidth between the reader and the servers of our MUSES, DORY and FP-HSE. In this experiment, we consider $L = 2$ servers and the search result size $n_s = 255$. The network overhead in MUSES increases from 0.4 MB to 2.2 MB, corresponding to the cases of 25 to 150 writers mostly due to transmitting KH-PRF key shares. For DORY, it incurs 4.6 MB–27.6 MB network overhead per search operation depending on the writer subset size, which is 11.9×–12.1× larger than the communication cost of MUSES. FP-HSE incurs the lowest bandwidth as the reader only sends a search token of 65 B to the server and receives the results. Although FP-HSE achieves the minimum bandwidth overhead among all schemes, it suffers from many vulnerabilities and leaks more information than the others.



**(a)** Writer's bandwidth **(b)** Writer's latency

**Figure 7:** Permission revocation performance (log scale on y-axis).

**Keyword search.** Figure 6b illustrates the corresponding end-to-end delay in keyword search of our scheme under 2 servers and the search result size of 255, in comparison with DORY and FP-HSE. The latency of all schemes grows almost linearly to the number of writers. MUSES is about 126.8×–134.7× faster than FP-HSE, and 1.6×–1.7× faster than DORY. With 25 writers, MUSES takes approximately 2.0s to accomplish a search, and increases to about 11.1s for 150 writers. The overhead of FP-HSE mainly comes from pairing operations, in which decrypting each encrypted search token needs two pairing operations, while the overhead of our scheme mainly stems from KH-PRF evaluation. By contrast, the overhead of DORY is mostly due to network overhead.

Three factors contributing to the search delay include reader processing, communication latency, and server processing. Specifically, the reader processing is low, which only takes 13.6ms–69.4ms, while server processing and communication take 1.8s–10.1s and 0.2s–0.9s, respectively. Their corresponding portion in the total delay is 0.6%–0.7%, 90.9%–91.3%, and 8.0%–8.4% respectively.

### 6.1.2 Permission revocation

We evaluate the performance of MUSES when a writer wants to update secret column keys to revoke search permission of the reader on her database, and compare it with other schemes. For DORY and FP-HSE, as these schemes do not offer permission revocation function for a user/writer's database by offloading re-encrypting work to the servers as ours, we measure their latency to re-encrypt a user/writer's search index on the user/writer side. For FP-HSE, the writer only re-encrypts her underlying SSE with another secret key and ignores updating encrypted search tokens to stop sharing her database.

**Writer's bandwidth.** Figure 7a demonstrates the bandwidth cost of all schemes in permission revocation. The bandwidth overhead of MUSES grows slightly when increasing BF size (from 1120 to 3120 corresponding to the cases from 1K to 500K documents) as the writer just needs to transmit secret-shares of KH-PRF keys, together with updated shared tokens while DORY and FP-HSE requires downloading and uploading the whole search index. MUSES incurs 4.4 MB–12.1 MB communication overhead, while DORY produces 0.4 MB–

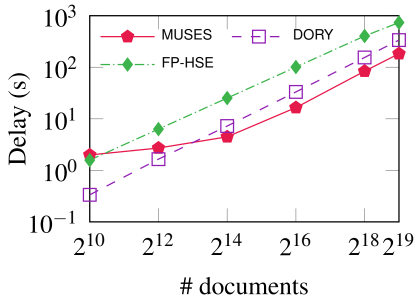
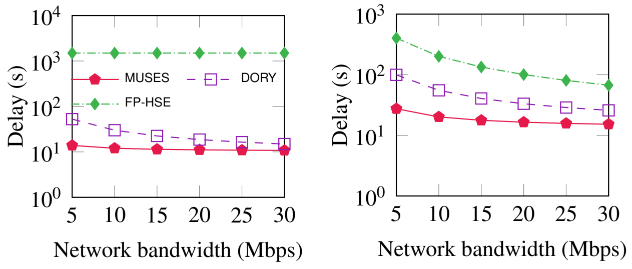
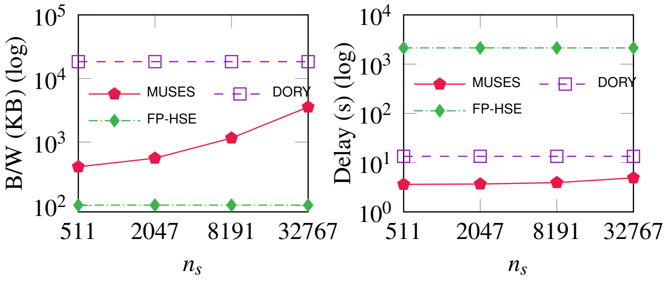


**Figure 8:** E2E permission revocation delay (log scale on y-axis).



**(a)** Keyword search **(b)** Permission revocation

**Figure 9:** E2E delay w/ varying bandwidths (log scale on y-axis).



**(a)** Reader's bandwidth **(b)** E2E delay

**Figure 10:** Keyword search performance with varying $n_s$.

587.0 MB, which is 1.5×–48.2× larger than MUSES starting from $2^{14}$ documents. The network overhead of FP-HSE is 3.9 MB–1833.4 MB, which is 2.8×–150.4× larger than our MUSES. Its cost is high due to the transmission of the search index back and forth similar to DORY.

**Writer's latency.** Figure 7b presents the computing time on the writer side to re-encrypt her search index. MUSES requires the minimum amount of time on the writer side, which is 27.9ms–76.1ms for the database sizes increasing from 1K to 500K documents, where most overhead is for deriving the current and new secret column keys. By contrast, a user in DORY takes 0.3s–335.5s to download, re-encrypt the search index with a new secret key and upload it again, which is 11.9×–4407.6× larger than the computing time of the writer in MUSES. FP-HSE incurs longer latency on the writer side due to its larger search index size, where it takes 1.6s–734.3s, corresponding to 56.4×–9647.4× longer than our MUSES.

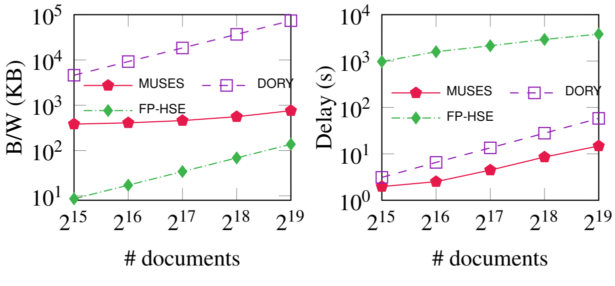
**Permission revocation.** Figure 8 illustrates the end-to-end delay to finish re-encryption of the search index of FP-HSE, DORY, and MUSES. It is noticeable that when increasing the number of documents, the end-to-end delay of all schemes grows linearly but by varying degrees. MUSES takes about 2.0s (resp. 183.1s) to update a search index including keyword representations of 1K (resp. 500K) documents. For DORY, its latency is around 0.3s and 336.6s, respectively, which is 1.6×–2.0× slower than MUSES starting from $2^{14}$ documents. FP-HSE incurs the largest overhead to re-encrypt the search index due to its larger index size. As a result, it takes 1.6s–736.1s, which is 2.3×–6.1× larger than MUSES.

### 6.1.3 Performance under varied parameters

**Varying network bandwidths.** Figure 9 demonstrates the end-to-end latency of keyword search and permission revocation of different schemes w.r.t various connection bandwidths. The search delay (Figure 9a) is measured in the case of 150 writers and the permission revocation delay (Figure 9b) is measured in the typical case of $2^{16} \approx 64$ K documents. As search operations of FP-HSE incur the lowest communication overhead, its performance is barely affected by varying network bandwidths, where it takes around $1.5 \times 10^3$s to finish a search operation. The end-to-end delay of MUSES slightly decreases when increasing the network bandwidth, where it takes 13.9s with 5 Mbps, and decreases to 10.8s with 30 Mbps, while DORY takes 52.6s and 14.9s, respectively. In permission revocation, since both DORY and FP-HSE need to download and upload the search index again, their latency decreases significantly when the network bandwidth is higher, in which DORY takes 99.6s–25.8s, and FP-HSE takes 402.6s–67.2s, corresponding to the bandwidths 5 Mbps–30 Mbps. By contrast, MUSES incurs a delay of 27.5s–15.3s to finish a permission revocation, where most communication overhead is for transmitting secret shares of KH-PRF keys to the servers.

**Varying search result sizes.** We present the performance of MUSES and other schemes with varying search result sizes $n_s$ (from 511 to 32767) in Figure 10. We consider 25 writers each having $N = 2^{17}$ documents and the keyword universe of 94,549 (with the corresponding BF parameter $m = 2520$). The bandwidth of MUSES increases from 408.9 KB to 3558.9 KB, respectively, which is 5.2×–45.0× smaller than DORY (Figure 10a). The corresponding delay increases from 3.6s to 4.9s, which is 2.7×–3.7× and 435.2×–590.3× faster than DORY and FP-HSE, respectively (Figure 10b).

**Varying database sizes.** We present the search performance of MUSES and other schemes with varying dataset sizes $N$ (from $2^{15}$ to $2^{19}$ documents per writer) in Figure 11. In this experiment, we fix the number of writers to be 25. When increasing the number of documents in the dataset, it also increases the size of the keyword universe (from $43,209$ to $168,770$) and the search result size (from 255 to 4095). As shown in Figure 11a, the bandwidth cost of MUSES slightly



**(a)** Reader's bandwidth **(b)** E2E delay

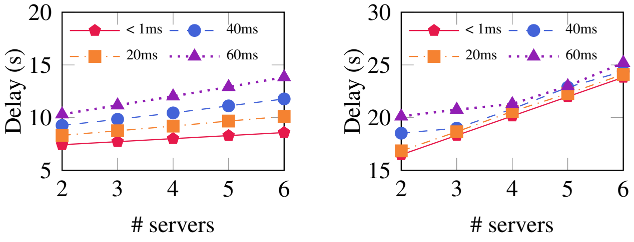
**Figure 11:** Keyword search performance w/ varying database sizes.

increases from 0.4 MB to 0.7 MB while DORY and FP-HSE increase from 4.5 MB to 71.9 MB and 8.6 KB–137.6 KB, respectively. That means MUSES incurs 12.0×–97.0× smaller bandwidth overhead than DORY and 5.5× larger than FP-HSE for $2^{19}$ documents. Figure 11b presents the corresponding search delay of MUSES. Specifically, MUSES takes about 2.0s–14.8s while DORY takes 3.1s–58.1s and FP-HSE takes 976.1s–3812.6s. Thus, MUSES is 1.6×–3.9× and 257.9×–631.8× faster than DORY and FP-HSE, respectively.

**Varying numbers of servers.** To achieve a higher privacy threshold, more servers can be added to the system. Figure 12 illustrates the end-to-end delay of keyword search and permission revocation on $2^{16}$ documents of 100 writers with varying numbers of servers (from 2 to 6) and inter-server network latencies (from 1ms to 60ms). In MUSES, adding more servers does not significantly increase the online computation work of each server. Instead, it requires more communication rounds between the servers to forward and open the shuffled search output. In addition, a small amount of extra overhead in search operations is put on the reader to create and send more DPF keys, secret shares of KH-PRF keys, as well as padding values when there are more servers in the system. The server network latency does not significantly impact the search delay. As shown in Figure 12a, searching takes 7.4s-8.6s under 1ms network latency while it takes 10.3s-13.8s under 60ms latency. In permission revocation, having more servers increases network traffic of the system because the servers need to broadcast their components for updating encrypted index to each other. Also, the writer has to send the secret-shares of the previous and fresh KH-PRF keys to all servers. As shown in Figure 12b, revoking permission takes 16.5s-23.8s under 1ms latency (with 2 to 6 servers), while it takes 20.1s-25.2s under 60ms latency.

#### 6.1.4 Setup Time and Document Update

Figure 13a shows the setup time of MUSES including writer processing time and communication delay. Specifically, the writer takes 9.0s–217.3s to create an encrypted index and its auxiliary components for a database of $2^{15}$–$2^{19}$ documents. Transmitting all these components to the servers takes 54.3s–1312.9s under network bandwidth 20 Mbps.



**(a)** Keyword search **(b)** Permission revocation

**Figure 12:** MUSES latency with varying numbers of servers.

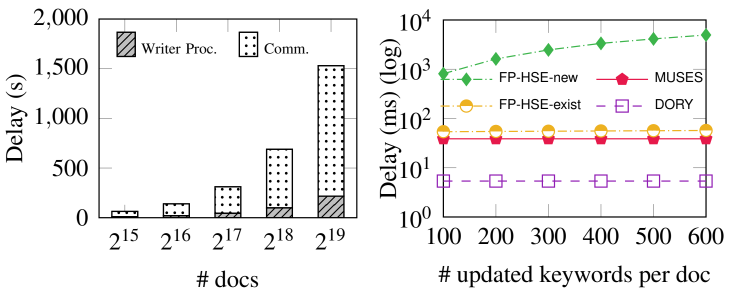
Figure 13b presents the update delay of MUSES, DORY, and FP-HSE. We measure the latency of FP-HSE in document update in two cases. The worst case is when all keywords of the updated document are new (FP-HSE-new), and the best case is when all keywords have appeared (FP-HSE-exist). For each new keyword, FP-HSE-new needs two public-key pairings to generate a new search token, thus its cost is linear to the number of new keywords, while DORY, FP-HSE-exist and our scheme remain nearly unchanged. MUSES takes about 38.7ms to update the index per document update, while FP-HSE-new takes 0.8s–4.9s for the cases increasing from 100 to 600 keywords, and DORY takes about 5.4ms. The update latency of FP-HSE-exist slightly grows from 53.9ms to 57.1ms as it does not incur pairing operations.

#### 6.1.5 Storage overhead

In MUSES, each writer stores a 256-bit secret key and the reader stores a 256-bit private key. As MUSES utilizes LWR-based KH-PRF in [11], each bit of the search index is encrypted to a 10-bit ciphertext in $\mathbb{Z}_p$ where $p = 2^{10}$. Each column of the index is encrypted by a separate key of size 1 KB. Thus, for a database with $2^{15}$ documents, the size of EIDX, STkn, and st is approximately 78.1 MB, 2.05 MB, and 0.25 MB, respectively. In total, the server storage cost for each writer is approximately 80.4 MB, which goes up to 12.1 GB for 150 writers. For the largest database in our experiment (i.e., $2^{19}$ documents), the server storage cost per writer is approximately 2.0 GB, which goes up to 48.9 GB for 25 writers. Note that in MUSES, the storage cost depends on the number of documents $N$ and the number of keywords per document, but not search result size $n_s$.

## 7 Related Work

**SSE.** Song et al. [76] were the first to propose and formalize SE. Secure SSE schemes permit encrypted search on encrypted data via an encrypted index with improved security (e.g., forward privacy [12, 59], backward privacy [13, 42, 79, 80]), efficiency (e.g., [19, 30]), query functionality (e.g., [56]) and/or updatability (e.g., [18, 44, 51, 52, 78]). Most SSE schemes are vulnerable to statistical inference at-



**(a)** Setup time **(b)** Document update

**Figure 13:** E2E setup time and document update delay

tacks [16, 54, 72, 74, 89], some of which exploit fundamental leakage in SSE (e.g., search pattern [58, 64, 71], access pattern [49], volume pattern [60], file-injection [93, 94]). Recently, Xu et al. [89] showed that the desirable security properties of SSE (e.g., forward/backward privacy) may not be sufficient to prevent devastating leakage-abuse attacks.

While SSE mainly supports single-user queries, several attempts (e.g., [20, 55] have been proposed to enable multi-user functionalities. Chamani et al. [20] proposed a multi-user SSE scheme that supports data and query policy control along with verifiability using blockchain. The scheme in [55] permits a reader to query encrypted data without interacting with the data owner using some helper users. Some other schemes permit multi-user functionalities by requiring all users to be trusted [28] or using costly cryptographic protocols such as multi-party computation [36, 50]. The scheme in [50] uses two non-colluding custodians executing garbled circuit to generate search tokens for readers. Its search follows standard SSE so it leaks pattern information (e.g., search, volume, result). Leakage-suppression technique [41] can prevent some pattern leakage; however, it fits more in the single-user setting due to the costly rebuild process. When using it in multi-user settings (separate reader and writers), the writer needs to keep track of the reader's activities to rebuild its index accordingly for security. Multi-key SSE [55, 87] provides decentralization between users but does not prevent pattern leakages.

**PKSE.** PKSE schemes such as [7, 10, 33, 91] can support multi-writer setting, but they do not adapt to forward privacy, which might lead to injection attacks [94]. Although hybrid-based model [84] can ensure forward privacy, it requires each writer to be stateful and present to rebuild encrypted tokens periodically. In addition, most PKSE systems are vulnerable to KGA. Some PKSE schemes can prevent KGA, but they require a dedicated trusted third party [57, 62, 85].

**Oblivious platforms.** Some oblivious storage platforms employ ORAM and/or PIR primitives to hide search patterns during data operations (e.g., data sharing/access [22–24, 28, 65, 67], search [31, 36, 40, 47, 70]. However, these schemes incur a large communication overhead, which costs $O(N)$ with $N$ is the number of documents in the collection. Differential Privacy-based technique [75] can obfuscate search access patterns, but it incurs high computation and latency.

**Hardware-assisted SE.** Trusted hardware (e.g., Intel SGX [25]) was used to build practical oblivious platforms with diverse functionalities (e.g., keyword search [39, 46, 69], SQL queries [35, 38], data storage [27, 45], oblivious memory [81]). These platforms require a security assumption on the hardware (e.g., isolation, tamper-free, side-channel resistance).

## Acknowledgment

We would like to thank the shepherd and the anonymous reviewers for their valuable comments. This research was supported by an unrestricted gift from Robert Bosch, 4-VA, 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`. Rouzbeh Behnia was supported by the USF Sarasota-Manatee campus Office of Research through the Interdisciplinary Research Grant program.

## References

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

[2] Zeromq: An open-source universal messaging library. `https://github.com/zeromq/libzmq`.

[3] Enron dataset. `https://www.cs.cmu.edu/~enron`, 2015.

[4] Michel Abdalla, Mihir Bellare, Dario Catalano, Eike Kiltz, Tadayoshi Kohno, Tanja Lange, John Malone-Lee, Gregory Neven, Pascal Paillier, and Haixia Shi. Searchable encryption revisited: Consistency properties, relation to anonymous ibe, and extensions. *J. Cryptology*, 21:350–391, 01 2008.

[5] Ghous Amjad, Sarvar Patel, Giuseppe Persiano, Kevin Yeo, and Moti Yung. Dynamic volume-hiding encrypted multi-maps with applications to searchable encryption. *Proc. Priv. Enhancing Technol.*, 2023(1):417–436, 2023.

[6] Nuttapong Attrapadung, Goichiro Hanaoaka, Takahiro Matsuda, Hiraku Morita, Kazuma Ohara, Jacob C. N. Schuldt, Tadanori Teruya, and Kazunari Tozawa. Oblivious linear group actions and applications. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, CCS '21, page 630–650, New York, NY, USA, 2021. Association for Computing Machinery.

[7] 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.

[8] Rouzbeh Behnia, Muslum Ozgur Ozmen, and Attila Altay Yavuz. Lattice-based public key searchable encryption from experimental perspectives. *IEEE Transactions on Dependable and Secure Computing*, 17(6):1269–1282, 2020.

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

[10] 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`.

[11] Dan Boneh, Kevin Lewi, Hart William Montgomery, and Ananth Raghunathan. Key homomorphic prfs and their applications. In *Annual International Cryptology Conference*, 2013.

[12] 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.

[13] 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.

[14] Elette Boyle, Niv Gilboa, and Yuval Ishai. Function secret sharing. In *EUROCRYPT (2)*, pages 337–367. Springer, 2015.

[15] Elette Boyle, Niv Gilboa, and Yuval Ishai. Function secret sharing: Improvements and extensions. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, page 1292–1303, New York, NY, USA, 2016. Association for Computing Machinery.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] Melissa Chase, Esha Ghosh, and Oxana Poburinnaya. Secret-shared shuffle. In *Advances in Cryptology – ASIACRYPT 2020: 26th International Conference on the Theory and Application of Cryptology and Information Security, Proceedings, Part III*, page 342–372. Springer-Verlag, 2020.

[22] 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`.

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

[24] Sherman SM Chow, Katharina Fech, Russell WF Lai, and Giulio Malavolta. Multi-client oblivious ram with polylogarithmic 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] Emma Dauterman, Mayank Rathee, Raluca Ada Popa, and Ion Stoica. Waldo: A private time-series database from function secret sharing. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 2450–2468, 2022.

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

[31] 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.

[32] Jan-Pieter D'Anvers, Angshuman Karmakar, Sujoy Sinha Roy, and Frederik Vercauteren. Saber: Module-lwr based key exchange, cpa-secure encryption and cca-secure kem. Cryptology ePrint Archive, Paper 2018/230, 2018. `https://eprint.iacr.org/2018/230`.

[33] 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.

[34] Saba Eskandarian and Dan Boneh. Clarion: Anonymous communication from multiparty shuffling protocols. In *29th Annual Network and Distributed System Security Symposium, NDSS*

*2022, San Diego, California, USA, April 24-28, 2022*. The Internet Society, 2022.

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

[36] 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.

[37] Ariel J. Feldman, William P. Zeller, Michael J. Freedman, and Edward W. Felten. SPORC: Group collaboration using untrusted cloud resources. In *9th USENIX Symposium on Operating Systems Design and Implementation (OSDI 10)*, Vancouver, BC, October 2010. USENIX Association.

[38] 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.

[39] 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.

[40] 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.

[41] 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.

[42] 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.

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

[44] 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.

[45] 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.

[46] 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.

[47] 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.

[48] Yuncong Hu, Sam Kumar, and Raluca Ada Popa. Ghostor: Toward a secure data-sharing system from decentralized trust. In *Proceedings of the 17th Usenix Conference on Networked Systems Design and Implementation*, NSDI'20, page 851–878, USA, 2020. USENIX Association.

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

[50] 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.

[51] 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.

[52] 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.

[53] Nikolaos Karapanos, Alexandros Filios, Raluca Ada Popa, and Srdjan Capkun. Verena: End-to-end integrity protection for web applications. In *2016 IEEE Symposium on Security and Privacy (SP)*, pages 895–913, 2016.

[54] 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.

[55] 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.

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

[57] 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.

[58] 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.

[59] 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.

[60] 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.

[61] 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.

[62] 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.

[63] Jinyuan Li, Maxwell Krohn, David Mazières, and Dennis Shasha. Secure untrusted data repository (SUNDR). In *6th Symposium on Operating Systems Design & Implementation (OSDI 04)*, San Francisco, CA, December 2004. USENIX Association.

[64] 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.

[65] 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.

[66] Fucai Luo, Haiyan Wang, Changlu Lin, and Xingfu Yan. Abaeks: Attribute-based authenticated encryption with keyword search over outsourced encrypted data. *IEEE Transactions on Information Forensics and Security*, 18:4970–4983, 2023.

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

[68] David Mazières and Dennis Shasha. Building secure file systems out of byzantine storage. In *Proceedings of the Twenty-First Annual Symposium on Principles of Distributed Computing*, PODC '02, page 108–117, New York, NY, USA, 2002. Association for Computing Machinery.

[69] 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.

[70] 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`.

[71] 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.

[72] 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.

[73] Sarvar Patel, Giuseppe Persiano, Kevin Yeo, and Moti Yung. Mitigating leakage in secure cloud-hosted data structures: Volume-hiding for multi-maps via hashing. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November 11-15, 2019*, pages 79–93. ACM, 2019.

[74] 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.

[75] 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.

[76] 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.

[77] 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.

[78] 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`.

[79] 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.

[80] 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.

[81] Afonso Tinoco, Sixiang Gao, and Elaine Shi. Enigmap: Signal should use oblivious algorithms for private contact discovery. *Cryptology ePrint Archive*, 2022.

[82] Alin Tomescu and Srinivas Devadas. Catena: Efficient non-equivocation via bitcoin. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 393–409, 2017.

[83] Jiafan Wang and Sherman Chow. Forward and backward-secure range-searchable symmetric encryption. *Proceedings on Privacy Enhancing Technologies*, 2022:28–48, 01 2022.

[84] 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.

[85] 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.

[86] Xiao Wang, Alex J. Malozemoff, and Jonathan Katz. EMP-toolkit: Efficient MultiParty computation toolkit. `https://github.com/emp-toolkit`, 2016.

[87] 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.

[88] Pieter Wuille. libsecp256k1. `https://github.com/bitcoin-core/secp256k1`.

[89] 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.

[90] Lingling Xu, Wanhua Li, Fangguo Zhang, Rong Cheng, and Shaohua Tang. Authorized keyword searches on public key encrypted data with time controlled keyword privacy. *IEEE Transactions on Information Forensics and Security*, 15:2096–2109, 2020.

[91] 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.

[92] Ming Zeng, Haifeng Qian, Jie Chen, and Kai Zhang. Forward secure public key encryption with keyword search for outsourced cloud storage. *IEEE Transactions on Cloud Computing*, 10(1):426–438, 2019.

[93] 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.

[94] 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 Security Game for MUSES

The security game for MUSES is presented in Figure 14 and the volume-hiding game is presented in Figure 15.

## B Proof of $\Pi_{\mathsf{LOC}}$

Let $\Pi_{\mathsf{LOC}}.\mathsf{Prep}$ and $\Pi_{\mathsf{LOC}}.\mathsf{Cnt}$ be the preprocessing and the online phases of $\Pi_{\mathsf{LOC}}$ protocol, respectively. We state the correctness and security of $\Pi_{\mathsf{LOC}}$ as follows.

```
$\mathsf{IND}^b_{\mathsf{MUSES},\mathcal{A},\mathcal{L}}(1^\lambda)$:
1. $n_w \leftarrow \mathcal{A}(1^\lambda)$
2. $(\mathsf{pk},\mathsf{sk}) \leftarrow \mathsf{RSetup}(1^\lambda)$
3. $(\mathsf{EIDX}_w,\mathsf{st}_w,\mathsf{PTkn}_w,\mathsf{STkn}_w) \leftarrow \mathsf{WSetup}(1^\lambda,w,\mathsf{pk}),\ \forall w \in [n_w]$
4. $\mathcal{H}_0 \leftarrow \{\emptyset\}$, $\mathcal{H}_1 \leftarrow \{\emptyset\}$
5. $\mathcal{O} \leftarrow \{\mathsf{CorruptWriterO}_b,\mathsf{SearchO}_b,\mathsf{UpdateO}_b,\mathsf{RevokeO}_b\}$
6. $b' \leftarrow \mathcal{A}^{\mathcal{O}}(\mathsf{pk},\{(\mathsf{EIDX}_w,\mathsf{st}_w,\mathsf{STkn}_w)\}_{w\in[n_w]})$
7. return $b'$

$\mathsf{SearchO}_b(\{(\mathsf{kw}_k,\mathcal{W}''_k)\}_{k\in\{0,1\}})$:
1. if $\mathcal{L}^{\mathsf{Search}}_{\mathcal{H}_0}(\mathsf{kw}_0,\mathcal{W}''_0) = \mathcal{L}^{\mathsf{Search}}_{\mathcal{H}_1}(\mathsf{kw}_1,\mathcal{W}''_1)$
    (a) $\forall k \in \{0,1\}, \mathcal{H}_k \leftarrow \mathcal{H}_k \cup (\mathsf{Search},\mathsf{kw}_k,\mathcal{W}''_k)$
    (b) return $\mathsf{SearchToken}(\mathsf{kw}_b,\mathcal{W}''_b)$
2. else return $\perp$

$\mathsf{UpdateO}_b(\{(w_k,u_k,\mathcal{V}_{u_k})\}_{k\in\{0,1\}})$:
1. if $\mathcal{L}^{\mathsf{Update}}_{\mathcal{H}_0}(w_0,u_0,\mathcal{V}_{u_0}) = \mathcal{L}^{\mathsf{Update}}_{\mathcal{H}_1}(w_1,u_1,\mathcal{V}_{u_1})$
    (a) $\forall k \in \{0,1\}, \mathcal{H}_k \leftarrow \mathcal{H}_k \cup (\mathsf{Update},w_k,u_k,\mathcal{V}_{u_k})$
    (b) return $\mathsf{UpdateToken}(\mathcal{V}_{u_b},u_b,w_b,\mathsf{PTkn}_{w_b},\mathsf{st}_{w_b})$
2. else return $\perp$

$\mathsf{RevokeO}_b(w_0,w_1)$:
1. if $\mathcal{L}^{\mathsf{Revoke}}_{\mathcal{H}_0}(w_0) = \mathcal{L}^{\mathsf{Revoke}}_{\mathcal{H}_1}(w_1)$
    (a) $\forall k \in \{0,1\}, \mathcal{H}_k \leftarrow \mathcal{H}_k \cup (\mathsf{Revoke},w_k)$
    (b) $(\mathsf{EIDX}'_{w_b},\mathsf{PTkn}'_{w_b}) \leftarrow \mathsf{RvkPrm}(w_b,\mathsf{PTkn}_{w_b},\mathsf{EIDX}_{w_b},\mathsf{st}_{w_b})$
    (c) return $\mathsf{EIDX}'_{w_b}$
2. else return $\perp$

$\mathsf{CorruptWriterO}_b(w_0,w_1)$:
1. if $\mathcal{L}^{\mathsf{CorruptWriter}}_{\mathcal{H}_0}(w_0) = \mathcal{L}^{\mathsf{CorruptWriter}}_{\mathcal{H}_1}(w_1)$
    (a) $\forall k \in \{0,1\}, \mathcal{H}_k \leftarrow \mathcal{H}_k \cup (\mathsf{CorruptWriter},w_k)$
    (b) return $\mathsf{PTkn}_{w_b}$
2. else return $\perp$
```

**Figure 14:** Security Game for MUSES

**Correctness.** For any $\lambda, N, K$, and any $\mathbf{d} = \sum_{i=1}^{L} \mathbf{d}^{(i)} \in \mathbb{Z}_p^N$, $\Pr[\sum_{i=1}^{L} s^{(i)} = \sum_{u=1}^{N}(\mathbf{d}[u] \overset{?}{=} K) \wedge \sum_{i=1}^{L} \mathbf{d}'^{(i)}[u] = (\mathbf{d}[u] \overset{?}{=} K)\forall u \in [N] \mid \{(r_u^{(1)},\mathbf{b}_u^{(1)})\}_{u\in[N]};\dots;\{(r_u^{(L)},\mathbf{b}_u^{(L)})\}_{u\in[N]}) \leftarrow \Pi_{\mathsf{LOC}}.\mathsf{Prep}(1^\lambda,K), (s^{(1)},\mathbf{d}'^{(1)};\dots;s^{(L)},\mathbf{d}'^{(L)}) \leftarrow \Pi_{\mathsf{LOC}}.\mathsf{Cnt}(\mathbf{d}^{(1)},\{(r_u^{(1)},\mathbf{b}_u^{(1)})\}_{u\in[N]};\dots;\mathbf{d}^{(L)},\{(r_u^{(L)},\mathbf{b}_u^{(L)})\}_{u\in[N]})] = 1$.

The security of $\Pi_{\mathsf{LOC}}$ is guaranteed as follows.

**Lemma 1.** $\Pi_{\mathsf{LOC}}$ *protocol is perfectly semi-honest secure in the* $\mathsf{TSS}$*-hybrid model.*

*Proof.* We prove Lemma 1 by showing that there exists a simulator that can produce an ideal world $\mathbf{Ideal}_{\mathcal{A},\mathcal{S},\mathcal{L}}(1^\lambda)$ that is secure in the presence of at most $L-1$ statically corrupt parties (out of $L$), and indistinguishable from the real game $\mathbf{Real}_{\mathcal{A}}(1^\lambda)$. Our simulator $\mathcal{S}$ behaves as follows. For each party $\mathcal{P}_i \in \{\mathcal{P}_1,\dots,\mathcal{P}_L\}$ and $u \in [N]$, it outputs a random circular shift permutation $\pi_u^{(i)}$ computed as $[1,2,\dots,K+1] \circlearrowleft x_u^{(i)}$, where $x_u^{(i)} \overset{\$}{\leftarrow} [K+1]$. For each party $\mathcal{P}_i \in \{\mathcal{P}_1,\dots,\mathcal{P}_{L-1}\}$, it outputs random vectors $\mathbf{r}_u^{(i)}, \mathbf{r}_u'^{(i)}$, and outputs $\boldsymbol{\Delta}_u^{(i)}$ for each

$\mathsf{VHGame}_b^{\mathcal{A},\mathcal{L}_{\mathcal{H}}}(1^\lambda)$:

1. $\mathcal{A}$ picks size $N$ and volume upper bound $n_s$ with $N \geq n_s \geq 1$, sends two search indices $\mathsf{EIDX}_0^0$ and $\mathsf{EIDX}_1^0$ to challenger $\mathcal{C}$ satisfying $N$ and $n_s$.
2. $\mathcal{C}$ computes $\mathcal{L}_{\mathcal{H}}^{\mathsf{Setup}}(\mathsf{EIDX}_b^0, N, n_s)$ which is sent to $\mathcal{A}$.
3. For $t = 1, \ldots,$
   (a) $\mathcal{A}$ adaptively picks $o_0^t = (w_0^t, \mathbf{d}_0^t, v^t, N^t, n_s^t)$ and $o_1^t = (w_1^t, \mathbf{d}_1^t, v^t, N^t, n_s^t)$ such that:
      i. Two operations are performed on the same label: $w_0^t = w_1^t$.
      ii. Let $\mathsf{EIDX}_0^t$ ($\mathsf{EIDX}_1^t$) be the $\mathsf{EIDX}$ obtained by executing $o_0^t$ ($o_1^t$) on $\mathsf{EIDX}_0^{t-1}$ ($\mathsf{EIDX}_1^{t-1}$), where $N^t$ and $n_s^t$ must be valid size and volume upper bounds for $\mathsf{EIDX}_0^t$ and $\mathsf{EIDX}_1^t$.
   (b) $\mathcal{C}$ returns $\mathcal{L}_{\mathcal{H}}^{\mathsf{Search}}(\mathsf{EIDX}_b^t, o_b^1, \ldots, o_b^t)$ for search queries and $\mathcal{L}_{\mathcal{H}}^{\mathsf{Update}}(\mathsf{EIDX}_b^t, o_b^1, \ldots, o_b^t)$ for updates.
4. Finally, $\mathcal{A}$ outputs a bit $b \in \{0,1\}$.

**Figure 15:** Volume-Hiding Game for MUSES

party $\mathcal{P}_i \in \{\mathcal{P}_1, \ldots, \mathcal{P}_L\}$ computed as follows:

$$\mathbf{\Delta}_u^{(i)} = \begin{cases} \mathbf{r}_u^{(1)} & \text{,if } \ell = 1 \\ \mathbf{r}_u^{(i)} - \pi_u^{(i)}(\mathbf{r}_u^{(i-1)}) & \text{,if } i \in 2, \ldots, L-1 \\ \sum_{j=1}^{L-1} \mathbf{r}_u'^{(j)} - \pi_u^{(i)}(\mathbf{r}_u^{(i-1)}) & \text{,otherwise.} \end{cases}$$

Simulator $\mathcal{S}$ executes the rest of the protocol between $L$ parties to output the view for each party $\mathcal{P}_i$ as $(s^{(i)}, \mathbf{d}'^{(i)})$, where $s^{(i)} \in \mathbb{Z}_p$, and $\mathbf{d}'^{(i)} \in \mathbb{Z}_p^N$.

By showing that each game (except the first) is indistinguishable from its previous one, we conclude that the adversary cannot distinguish the real game from the ideal game with non-negligible probability.

**Hybrid**$_0$: As **Real**, Pr[**Hybrid**$_0 = 1$] = Pr[**Real**$_{\mathcal{A}}(1^\lambda) = 1$].

**Hybrid**$_1$: Instead of invoking $\mathsf{TSS.ShrTrns}$ between $\mathcal{P}_i \in \{\mathcal{P}_1, \ldots, \mathcal{P}_{L-1}\}$ and $\mathcal{P}_j \in \{\mathcal{P}_{i+1}, \ldots, \mathcal{P}_L\}$, **Hybrid**$_1$ outputs random vectors $\mathbf{a}_{j,u}^{(i)}, \mathbf{b}_{j,u}^{(i)}$ for $\mathcal{P}_i$, and outputs $\mathbf{\Delta}_{i,u}^{(j)} \leftarrow \mathbf{b}_{j,u}^{(i)} - \pi_u^{(j)}(\mathbf{a}_{j,u}^{(i)})$ for $\mathcal{P}_j$, $u \in [N]$. If $\mathcal{A}$ can distinguish **Hybrid**$_0$ from **Hybrid**$_1$, we can build a reduction to distinguish between a pseudorandom generator and a truly random generator.

**Hybrid**$_2$: Similar to **Hybrid**$_1$, but now **Hybrid**$_2$ outputs random vectors $\mathbf{r}_u'^{(i)}$ for $\mathcal{P}_i \in \{\mathcal{P}_1, \ldots, \mathcal{P}_{L-1}\}$, and generates $\mathbf{r}_u^{(i)} \leftarrow \mathbb{Z}_p^{K+1}$, with $i = 1, \ldots, L$, then outputs $\mathbf{\Delta}_u^{(1)} \leftarrow \mathbf{r}_u^{(1)}$ for $\mathcal{P}_1$, $\mathbf{\Delta}_u^{(i)} \leftarrow \mathbf{r}_u^{(i)} - \pi_u^{(i)}(\mathbf{r}_u^{(i-1)})$ for $\mathcal{P}_i \in \{\mathcal{P}_2, \ldots, \mathcal{P}_{L-1}\}$, and $\mathbf{\Delta}^{(L)} \leftarrow \sum_{i=1}^{L-1} \mathbf{r}_u'^{(i)} - \pi_u^{(L)}(\mathbf{r}_u^{(L-1)})$ for $\mathcal{P}_L$, with $u \in [N]$. In the real game, $\mathbf{r}_u'^{(i)}$ is $\mathbf{b}_{L,u}^{(i)}$, for $i = 1, \ldots, L-1$, and $\mathbf{r}_u^{(i)} \leftarrow \sum_{j=1}^{i} \mathbf{a}_{i+1,u}^{(j)}$, for $i = 2, \ldots, L-1$. As $\mathsf{TSS.ShrTrns}$ is secure, $\mathbf{a}_{i+1,u}^{(j)}$ is only known by $\mathcal{P}_j$, similar to $\mathbf{b}_{i,u}^{(j)}$. The quantities $\mathbf{a}_{i+1,u}^{(j)}$ and $\mathbf{b}_{i,u}^{(j)}$ are related in the following equations (in execution of $\Pi_{\mathsf{LOC}}.\mathsf{Prep}$):

$$\begin{cases} \mathbf{a}_{i+1,u}^{(j)} - \mathbf{x}_1 = \mathbf{c}_1, \\ \mathbf{b}_{i,u}^{(j)} - \mathbf{x}_2 = \mathbf{c}_2, \\ \mathbf{a}_{i+1,u}^{(j)} - \mathbf{b}_{i,u}^{(j)} = \mathbf{c}_3. \quad (\text{if } L > 2) \end{cases}$$

With 4 unknown variables, $\mathbf{a}_{i+1,u}^{(j)}, \mathbf{b}_{i,u}^{(j)}, \mathbf{x}_1, \mathbf{x}_2$ and 3 equations, the adversary who corrupts $\mathcal{P}_i$ and $\mathcal{P}_{i+1}$ cannot learn $\mathbf{a}_{i+1,u}^{(j)}$ and $\mathbf{b}_{i,u}^{(j)}$. Therefore, no party can learn $\mathbf{a}_{i+1,u}^{(j)}$, $\mathbf{b}_{i,u}^{(j)}$ except $\mathcal{P}_j$. If $\mathcal{A}$ can distinguish **Hybrid**$_1$ from **Hybrid**$_2$, we can build a reduction to distinguish between a pseudorandom generator and a truly random generator.

**Hybrid**$_3$: Similar to **Hybrid**$_2$, but now **Hybrid**$_3$ outputs the view for $\mathcal{P}_1$ as $\mathbf{o}_u^{(1)} \leftarrow \mathbf{e} \in \mathbb{Z}_p^{K+1}$, in which $\mathbf{e}[1] = 1$ and $\mathbf{e}[k] = 0 \forall k \neq 1$. Next, it computes and outputs the view for $\mathcal{P}_i \in \{\mathcal{P}_1, \ldots, \mathcal{P}_{L-1}\}$ and $\mathcal{P}_{i+1}$ as $\mathbf{o}_u^{(i+1)} \leftarrow \pi_u^{(i)}(\mathbf{o}_u^{(i)}) + \mathbf{\Delta}_u^{(i)}$, and outputs the view for $\mathcal{P}_L$ as $\mathbf{o}_u^{(L+1)} \leftarrow (\mathbf{e} \circlearrowright x_u^{(1)} \circlearrowright x_u^{(2)} \circlearrowright \cdots \circlearrowright x_u^{(L)}) + \sum_{i=1}^{L} \mathbf{r}_u'^{(i)}$, with $u \in [N]$. At the final step, each party $\mathcal{P}_i \in \{\mathcal{P}_1, \ldots, \mathcal{P}_{L-1}\}$ holds $\{(x_u^{(i)}, \mathbf{b}_u^{(i)} \leftarrow -\mathbf{r}_u'^{(i)})\}_{u \in [N]}$, and $\mathcal{P}_L$ holds $\{(x_u^{(L)}, \mathbf{b}_u^{(L)} \leftarrow \mathbf{o}_u^{(L+1)})\}_{u \in [N]}$. As the adversary can statically corrupt at most $L-1$ out of $L$ parties, it cannot differentiate between **Hybrid**$_2$ and **Hybrid**$_3$ because the position of the "one-hot" value and the permuted "one-hot" vector are not known by the adversary.

**Hybrid**$_4$: Instead of invoking the online phase $\Pi_{\mathsf{LOC}}.\mathsf{Cnt}$, **Hybrid**$_4$ outputs $x_u'^{(i)} \leftarrow -x_u^{(i)} \in \mathbb{Z}_p$, $q_u^{(i)} \leftarrow (x_u'^{(i)} \lll z) + \mathbf{d}^{(i)}[u] \pmod p$, and $q_u \leftarrow (\sum_{i=1}^{L} q_u^{(i)} \ggg z) \in \mathbb{Z}_{K+1}$ for all parties $\mathcal{P}_i \in \{\mathcal{P}_1, \ldots, \mathcal{P}_L\}$, where $\mathbf{d}^{(i)}[u] \xleftarrow{\$} \mathbb{Z}_p$ and $u \in [N]$. Next, it outputs the view for each party $\mathcal{P}_i$ as $\mathbf{b}_u'^{(i)} \leftarrow \mathbf{b}_u^{(i)} \circlearrowright x_u^{(i)}$, $\mathbf{d}'^{(i)}[u] \leftarrow \mathbf{b}_u'^{(i)}[K]$, for $u \in [N]$, and $s^{(i)} \leftarrow \sum_{u=1}^{N} \mathbf{b}_u'^{(i)}[K] \in \mathbb{Z}_p$. As the adversary can statically corrupt at most $L-1$ out of $L$ parties, it cannot differentiate between **Hybrid**$_3$ and **Hybrid**$_4$ because the counting result $s$ for the values equal to $K$ in the input vector $\mathbf{d} = \sum_{i=1}^{L} \mathbf{d}^{(i)}$ is secret shared between $L$ parties, and similar to the output vector $\mathbf{d}' = \sum_{\ell=1}^{L} \mathbf{d}'^{(\ell)}$, where $\mathbf{d}'[u] = \left( \left( \sum_{i=1}^{L} \mathbf{d}^{(i)}[u] \right) \overset{?}{=} K \right)$, with $u \in [N]$.

It is easy to see that **Hybrid**$_4$ is the ideal game **Ideal**$_{\mathcal{A},\mathcal{S},\mathcal{L}}(1^\lambda)$ and this completes the proof. □

## C Proof of $\Pi_{\mathsf{LOS}}$

Let $\Pi_{\mathsf{LOS}}.\mathsf{Prep}$ and $\Pi_{\mathsf{LOS}}.\mathsf{Shffl}$ be the preprocessing and the online phases of $\Pi_{\mathsf{LOS}}$ protocol, respectively. We state the correctness and security of $\Pi_{\mathsf{LOS}}$ as follows.

**Correctness.** For any integers $\lambda, n$, and any $\mathbf{d}$ with $L$ secret shares $\mathbf{d}^{(i)} \in \mathbb{Z}_p^n$, $i \in [L]$, such that $\mathbf{d} = \sum_{i=1}^{L} \mathbf{d}^{(i)} \in \mathbb{Z}_p^n$, $\Pr[\pi^{(1)^{-1}}(\ldots(\pi^{(L-1)^{-1}}(\mathbf{r}))\ldots) = \mathbf{d} \mid (\pi^{(1)}, \mathbf{\Delta}^{(1)}; \pi^{(2)}, \mathbf{\Delta}^{(2)}, \mathbf{a}^{(2)}; \ldots; \mathbf{a}^{(L)}, \mathbf{b}_{L-1}^{(L)}) \leftarrow \Pi_{\mathsf{LOS}}.\mathsf{Prep}(1^\lambda, n), (\pi^{(1)}; \pi^{(2)}; \ldots; \pi^{(L-1)}; \mathbf{r}) \leftarrow \Pi_{\mathsf{LOS}}.\mathsf{Shffl}(\mathbf{d}^{(1)}, \pi^{(1)}, \mathbf{\Delta}^{(1)}; \mathbf{d}^{(2)}, \pi^{(2)}, \mathbf{\Delta}^{(2)}, \mathbf{a}^{(2)}; \ldots; \mathbf{d}^{(L)}, \mathbf{a}^{(L)}, \mathbf{b}_{L-1}^{(L)})] = 1$.

**Lemma 2.** *Assuming that the adversary can statically corrupt at most $L-1$ out of the $L$ parties, $\Pi_{\mathsf{LOS}}$ is $\mathcal{L}$-secure in the TSS-hybrid model, where $\mathcal{L}_{\Pi_{\mathsf{LOS}}}.\mathsf{Prep} = \{\emptyset\}$, and $\mathcal{L}_{\Pi_{\mathsf{LOS}}}.\mathsf{Shffl} = \{\pi(\mathbf{d}) \wedge \mathcal{P}_L \in I_c\}$, where $\pi(\mathbf{d})$ is a pseudorandom permutation of input vector $\mathbf{d}$, and $I_c$ is the set of $L-1$ corrupted parties.*

*Proof.* Similar to the proof of Lemma 1 above, we prove Lemma 2 by showing a simulator that can produce an ideal game $\mathbf{Ideal}_{\mathcal{A},\mathcal{S},\mathcal{L}}(1^\lambda)$ which is secure in the presence of at most $L-1$ statically corrupt parties (out of $L$), and indistinguishable from the real game $\mathbf{Real}_{\mathcal{A}}(1^\lambda)$.

The simulator $\mathcal{S}$ behaves as follows. In $\Pi_{\mathsf{LOS}}.\mathsf{Prep}$, for each party $\mathcal{P}_i$, where $i \in [L-1]$, it outputs a random permutation $\pi^{(i)}$. For each party $\mathcal{P}_i \in \{\mathcal{P}_2, \ldots, \mathcal{P}_L\}$, it outputs a random mask $\mathbf{r}^{(i)}$. Also, it generates random vectors $\mathbf{r}'^{(i)}$, for $i = 2, \ldots, L$, and outputs $\mathbf{\Delta}^{(i)}$ for each party $\mathcal{P}_i$ computed as follows:

$$\mathbf{\Delta}^{(i)} = \begin{cases} \mathbf{r}'^{(i+1)} - \pi^{(i)}(\sum_{j=2}^{L} \mathbf{r}^{(j)}) & ,\text{if } i = 1 \\ \mathbf{r}'^{(i+1)} - \pi^{(i)}(\mathbf{r}'^{(i)}) & ,\text{if } i \in 2, \ldots, L-1 \\ \mathbf{r}'^{(i)} & ,\text{otherwise.} \end{cases}$$

Simulator $\mathcal{S}$ executes $\Pi_{\mathsf{LOS}}.\mathsf{Shffl}$ between $L$ parties as $(\pi^{(1)}; \pi^{(2)}; \ldots; \pi^{(L-1)}; \mathbf{o}) \leftarrow \Pi_{\mathsf{LOS}}.\mathsf{Shffl}(\mathbf{d}^{(1)}, \pi^{(1)}, \mathbf{\Delta}^{(1)}; \mathbf{d}^{(2)}, \pi^{(2)}, \mathbf{\Delta}^{(2)}, \mathbf{r}^{(2)}; \ldots; \mathbf{d}^{(L)}, \mathbf{r}^{(L)}, \mathbf{\Delta}^{(L)})$ to output the view for $L$ parties in the online phase, where $\mathbf{d}^{(i)} \xleftarrow{\$} \mathbb{Z}_p^N$, for $i \in [L]$.

**Hybrid**$_0$: As **Real**, $\Pr[\mathbf{Hybrid}_0 = 1] = \Pr[\mathbf{Real}_{\mathcal{A}}(1^\lambda) = 1]$.

**Hybrid**$_1$: Similar to **Hybrid**$_0$, but now **Hybrid**$_1$ outputs a random permutation $\pi^{(i)}$ for each $\mathcal{P}_i \in \{\mathcal{P}_1, \ldots, \mathcal{P}_{L-1}\}$. If $\mathcal{A}$ can distinguish **Hybrid**$_0$ from **Hybrid**$_1$, we can build a reduction to distinguish between a pseudorandom permutation and a truly random permutation.

**Hybrid**$_2$: Instead of invoking TSS.ShrTrns between $\mathcal{P}_i \in \{\mathcal{P}_1, \ldots, \mathcal{P}_{L-1}\}$ and $\mathcal{P}_j \in \{\mathcal{P}_{i+1}, \ldots, \mathcal{P}_L\}$, **Hybrid**$_2$ outputs random vectors $\mathbf{a}_i^{(j)}, \mathbf{b}_i^{(j)}$ for $\mathcal{P}_j$, and outputs $\mathbf{\Delta}_j^{(i)} \leftarrow \mathbf{b}_i^{(j)} - \pi^{(i)}(\mathbf{a}_i^{(j)})$ for $\mathcal{P}_i$. If $\mathcal{A}$ can distinguish **Hybrid**$_1$ from **Hybrid**$_2$, we can build a reduction to distinguish between a pseudorandom generator and a truly random generator.

**Hybrid**$_3$: Similar to **Hybrid**$_2$, but now **Hybrid**$_3$ outputs random vectors $\mathbf{r}^{(i)}$ for $\mathcal{P}_i \in \{\mathcal{P}_2, \ldots, \mathcal{P}_L\}$, and $\mathbf{r}'^{(i)}$ for $\mathcal{P}_i \in \{\mathcal{P}_1, \ldots, \mathcal{P}_{L-1}\}$, then outputs $\mathbf{\Delta}^{(1)} \leftarrow \mathbf{r}'^{(2)} - \pi^{(1)}(\sum_{i=2}^{L} \mathbf{r}^{(i)})$ for $\mathcal{P}_1$, $\mathbf{\Delta}^{(i)} \leftarrow \mathbf{r}'^{(i+1)} - \pi^{(i)}(\mathbf{r}'^{(i)})$ for $\mathcal{P}_i \in \{\mathcal{P}_2, \ldots, \mathcal{P}_{L-1}\}$, and $\mathbf{\Delta}^{(L)} \leftarrow -\mathbf{r}'^{(L)}$ for $\mathcal{P}_L$. If $\mathcal{A}$ can distinguish **Hybrid**$_2$ from **Hybrid**$_3$, we can build a reduction to distinguish between a pseudorandom generator and a truly random generator.

**Hybrid**$_4$: Instead of invoking in the online phase $\Pi_{\mathsf{LOS}}.\mathsf{Shffl}$, it computes $\mathbf{z}^{(i)} \leftarrow \mathbf{d}^{(i)} + \mathbf{r}^{(i)}$ as the view of each party $\mathcal{P}_i \in \{\mathcal{P}_2, \ldots, \mathcal{P}_L\}$, where $\mathbf{d}^{(i)}$ is the secret share of the input data vector $\mathbf{d} = \sum_{i=1}^{L} \mathbf{d}^{(i)}$, and $\mathbf{o}^{(1)} \leftarrow \mathbf{d}^{(1)} + \sum_{i=2}^{L} \mathbf{z}^{(i)}$ as the view of $\mathcal{P}_1$. Next, it computes and outputs the view for each $\mathcal{P}_i \in \{\mathcal{P}_1, \ldots, \mathcal{P}_{L-1}\}$ and $\mathcal{P}_{i+1}$ as $\mathbf{o}^{(i+1)} \leftarrow \pi^{(i)}(\mathbf{o}^{(i)}) + \mathbf{\Delta}^{(i)}$. Because each $\mathcal{P}_i \in \{\mathcal{P}_1, \ldots, \mathcal{P}_L\}$, only knows $\mathbf{\Delta}^{(i)} \leftarrow \mathbf{r}'^{(i+1)} - \pi^{(i)}(\mathbf{r}'^{(i)})$, where $\mathbf{r}'^{(1)} = \sum_{i=2}^{L} \mathbf{r}^{(i)}$, $\mathbf{r}'^{(L+1)} = \{0\}^n$, and $\pi^{(L)} = \varnothing$, without knowing $\mathbf{r}'^{(i)}$ and $\mathbf{r}'^{(i+1)}$, except $\mathcal{P}_L$ who knows $\mathbf{r}'^{(L)}$, any collusion of $L-1$ parties cannot recover data before being permuted. In particular, in the real game, $\mathbf{r}^{(i)}$ is $\mathbf{a}_1^{(i)}$, for $i = 2, \ldots, L$, and $\mathbf{r}'^{(i)} \leftarrow \sum_{j=i+1}^{L} \mathbf{b}_i^{(j)}$, for $i = 2, \ldots, L-1$. As TSS.ShrTrns is secure, $\mathbf{a}_1^{(i)}$ is only known by $\mathcal{P}_i$ . For the quantity $\mathbf{b}_i^{(j)}$, with $j > 1$, on the one hand, it is related in the following equations (in execution of $\Pi_{\mathsf{LOS}}.\mathsf{Prep}$):

$$\begin{cases} \mathbf{b}_i^{(j)} - \mathbf{x} = \mathbf{c}_1, \\ \mathbf{b}_i^{(j)} - \mathbf{a}_{i+1}^{(j)} = \mathbf{c}_2. \quad (\text{if } L > 2) \end{cases}$$

With 3 unknown variables $\mathbf{b}_i^{(j)}, \mathbf{a}_{i+1}^{(j)}, \mathbf{x}$ and 2 equations, the adversary who corrupts $\mathcal{P}_i$ and $\mathcal{P}_{i+1}$ cannot learn $\mathbf{b}_i^{(j)}$. On the other hand, $\mathbf{b}_i^{(j)}$ can also be derived from $\mathbf{a}_i^{(j)}$ as they are the sum of rows and columns of a punctured matrix, respectively, which is related in the following equations:

$$\begin{cases} \mathbf{a}_i^{(j)} - \mathbf{x}_1 = \mathbf{c}_1, \\ \mathbf{b}_{i-1}^{(j)} - \mathbf{x}_2 = \mathbf{c}_2. \\ \mathbf{b}_{i-1}^{(j)} - \mathbf{a}_i^{(j)} = \mathbf{c}_3. \quad (\text{if } L > 2) \end{cases}$$

With 4 unknown variables, $\mathbf{a}_i^{(j)}, \mathbf{b}_{i-1}^{(j)}, \mathbf{x}_1, \mathbf{x}_2$ and 3 equations, the adversary who corrupts $\mathcal{P}_i$ and $\mathcal{P}_{i-1}$ cannot learn $\mathbf{a}_i^{(j)}$ and $\mathbf{b}_{i-1}^{(j)}$. Therefore, no party can learn $\mathbf{b}_i^{(j)}$ except $\mathcal{P}_j$. While each of servers $\mathcal{P}_1, \ldots, \mathcal{P}_{L-1}$ holds an independent permutation, party $\mathcal{P}_L$ holds a final mask to open the shuffled output. Thus the adversary cannot distinguish **Hybrid**$_3$ from **Hybrid**$_4$.

It is easy to see that **Hybrid**$_4$ is the ideal game $\mathbf{Ideal}_{\mathcal{A},\mathcal{S},\mathcal{L}}(1^\lambda)$ and this completes the proof.

□

## D Proof of MUSES

**Correctness of MUSES.** For all $\lambda$, $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{RSetup}(1^\lambda)$, $(\mathsf{EIDX}_w,\ \mathsf{st}_w,\ \mathsf{PTkn}_w,\ \mathsf{STkn}_w) \leftarrow \mathsf{WSetup}(1^\lambda, w, \mathsf{pk})$, $w \in \mathcal{W}$, and all sequences of Search, Update operations over $\{\mathsf{EIDX}_w\}_{w \in [n_w]}$ using tokens generated respectively from $\mathsf{SearchToken}(\mathsf{kw}, \mathcal{W}')$, and $\mathsf{UpdateToken}(\mathcal{V}_u,\ u,\ w,\ \mathsf{PTkn}_w,\ \mathsf{st}_w)$, as well as RvkPrm operations over $w \in \widetilde{\mathcal{W}} \subseteq \mathcal{W}$, Search returns the correct results w.r.t the inputs $(\mathcal{V}_u, u, w)$ of UpdateToken when $w \in \mathcal{W}' \setminus \widetilde{\mathcal{W}}$, except with negligible probability in $\lambda$.

Following is the security proof of Theorem 1.

*Proof.* We derive a $(t+1)$-hybrid sequence starting from $\mathbf{Hybrid}_0 = \mathsf{IND}^0_{\mathsf{MUSES}}$, and the last hybrid $\mathbf{Hybrid}_t$ is exactly $\mathsf{IND}^1_{\mathsf{MUSES}}$. For $l \in \{0, \ldots, t-1\}$, the only difference between $\mathbf{Hybrid}_l$ and $\mathbf{Hybrid}_{l+1}$ is that the oracle responds to the $(l+1)$-th query in $\mathbf{Hybrid}_l$ with input $b = 0$, while responding to the $(l+1)$-th query in $\mathbf{Hybrid}_{l+1}$ with input $b = 1$.

We prove that $\mathcal{A}$ cannot distinguish $\mathsf{IND}^0_{\mathsf{MUSES}}$ from $\mathsf{IND}^1_{\mathsf{MUSES}}$ with non-negligible probability by showing that each hybrid (except the first) is indistinguishable from its previous.

For $l \in \{0, \ldots, t-1\}$, $\mathsf{Hist}_{l+1}$ can fall into four cases:

*(1)* $\mathsf{CorruptWriterO}_b$ on $(w_0, w_1)$: It will only be answered when identifier $w = w_0 = w_1$. As the update token provided by the corrupted writer is revealed, it requires that the update tuples provided by the writer $w$ (i.e., $\mathsf{UpdateBy}(w)$) are the

same for either $b = 0$ or $b = 1$. We have $\mathbf{Hybrid}_l = \mathbf{Hybrid}_{l+1}$ as the views of $\mathcal{A}$ are identical.

(2) $\mathsf{SearchO}_b$ on $(\{\mathsf{kw}_k, \mathcal{W}''_k\}_{k\in\{0,1\}})$: As the target writers of any search leaks, it will only be answered when $\mathcal{W}' = \mathcal{W}''_0 = \mathcal{W}''_1$. Because at most $L-1$ out of $L$ servers are corrupted, $\mathcal{A}$ cannot learn the search output unless there are corrupted writers. If servers $\mathcal{P}_1, \ldots, \mathcal{P}_{L-1}$ are corrupted, permutations $\pi_1, \ldots, \pi_{L-1}$ are the only leaked information. If server $\mathcal{P}_L$ is included in the corrupted servers together with $L-2$ remaining servers who hold permutations, as the search volume of $\mathsf{kw}_k$ (i.e., $\mathsf{sv}(\mathsf{kw}_k)$) is a constant ($\mathsf{sv}(\mathsf{kw}_0) = \mathsf{sv}(\mathsf{kw}_1) = n_s$), the adversary cannot differentiate $\mathsf{kw}_0$ with $\mathsf{kw}_1$ based on their search volumes, and still lacks a permutation of the honest (uncorrupted) server to recover the search output. Therefore, with the permuted search output, the adversary cannot distinguish whether it is due to $\mathsf{kw}_0$ or $\mathsf{kw}_1$. The indistinguishability between $\mathbf{Hybrid}_l$ and $\mathbf{Hybrid}_{l+1}$ is guaranteed by the computationally indistinguishable security of DPF-based PIR, KH-PRF, $\Pi_{\mathsf{LOC}}$, and $\Pi_{\mathsf{LOS}}$ schemes.

(3) $\mathsf{UpdateO}_b$ on $(\{w_k, u_k, \mathcal{V}_{u_k}\}_{k\in\{0,1\}})$: The oracle answers the queries when $w = w_0 = w_1$ and $u = u_0 = u_1$ (i.e., $\mathsf{up}(u)$), as the writer identifier and the position of the updated row (i.e., $\mathsf{EIDX}[u,*]$, as well as the position of the updated state (i.e., $\mathsf{st}_u$) will be leaked during update.

Obviously, if $(\mathsf{CorruptWriter}, w) \in \mathcal{H}$, $\mathcal{A}$ will have the knowledge of the update token. In particular, if writer $w$ has been corrupted, the BF representation of keywords in $\mathcal{V}_{u_k}$, as well as the position $u_k$ is divulged. Thus, the oracle only answers when two update tuples are similar in this case.

The indistinguishability between $\mathbf{Hybrid}_l$ and $\mathbf{Hybrid}_{l+1}$ is guaranteed by the computationally indistinguishable security of KH-PRF.

(4) $\mathsf{RevokeO}_b$ on $(w_0, w_1)$: The oracle answers the queries when $w = w_0 = w_1$, since the writer identifier will be leaked in permission revocation. In addition, if $(\mathsf{CorruptWriter}, w) \in \mathcal{H}$, $\mathcal{A}$ will still be able to obtain the updated private tokens (i.e., KH-PRF keys) of writer $w$ after they are updated.

The indistinguishability between $\mathbf{Hybrid}_l$ and $\mathbf{Hybrid}_{l+1}$ is guaranteed by the computationally indistinguishable security of KH-PRF.

By repeating the above procedure for $l \in [t-1]$, we conclude that $\mathcal{A}$ cannot distinguish $\mathbf{Hybrid}_0 = \mathsf{IND}^0_{\mathsf{MUSES}}$ from $\mathbf{Hybrid}_t = \mathsf{IND}^1_{\mathsf{MUSES}}$. Thus, MUSES is $\mathcal{L}_{\mathcal{H}}$-adaptively-secure.

□

## E Query-Independent Preprocessing

We analyze preprocessing cost of the search protocol in MUSES. The overhead mostly comes from executing Share Translation protocol (ST) between two servers: $(\mathbf{\Delta}; \mathbf{a}, \mathbf{b}) \leftarrow \mathsf{TSS.ShrTrns}(\pi; 1^\lambda)$, which lets one server obtain $\mathbf{a}, \mathbf{b}$, and the other server learn the corresponding translation function $\mathbf{\Delta} \leftarrow \mathbf{b} - \pi(\mathbf{a})$ without revealing the permutation $\pi$. As the input/output of the preprocessing is independent of search queries, the precomputed materials can be stored in a temporary memory and used when a search happens. The communication and computation complexity of the primitive Share Translation protocol in [21] is $O(n \log n)$ and $O(n^2)$, respectively, where $n$ is the number of elements. We analyze the communication and computation cost of preprocessing for a search query happening on a writer subset $\mathcal{W}'$ as follows, in which the number of servers $L$ is a small constant and omitted. As the preprocessing of $\Pi_{\mathsf{LOS}}$ executes ST $|\mathcal{W}'|$ times w.r.t. the number of elements $N + n_s$ (the total bound on the number of documents and padded values), the communication and computation cost for preprocessing of $\Pi_{\mathsf{LOS}}$ is $O(|\mathcal{W}'|(N+n_s)\log(N+n_s))$ and $O(|\mathcal{W}'|(N+n_s)^2)$, respectively. Similarly, the preprocessing of $\Pi_{\mathsf{LOC}}$ executes ST $|\mathcal{W}'|N$ times w.r.t. the number of elements $K+1$, where $K$ is a BF parameter, the communication and computation cost for preprocessing of $\Pi_{\mathsf{LOC}}$ is $O(|\mathcal{W}'|N(K+1)\log(K+1))$ and $O(|\mathcal{W}'|N(K+1)^2)$, respectively.

## F Preventing rollback attacks

In the "rollback" attacks (e.g., [37, 48, 53, 63, 68, 82]), the malicious server can omit some writer's update, and present the outdated data to the reader. We present a (simple) extension to our semi-honest MUSES construction to prevent such attacks. The high-level idea is to employ additional servers and perform "pair-wise" checking of the responses from different subsets of servers in processing the reader's request. For simplicity, assume our original MUSES scheme uses two servers. We add one more server to detect rollback attacks as follows. Let $\mathcal{W}'$ be the writer subset and kw is the keyword that the reader would like to search. For each pair of servers $\mathcal{P}_i, \mathcal{P}_j \in \{\mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3\}$, the reader executes the following:

- $\mathfrak{s}_{ij} = (\mathfrak{s}_i, \mathfrak{s}_j) \leftarrow \mathsf{SearchToken}(\mathsf{kw}, \mathcal{W}')$
- $O_{ij} \leftarrow \mathsf{Search}(\mathfrak{s}_{ij}, \mathsf{sk}, \{(\mathsf{EIDX}_w, \mathsf{STkn}_w, \mathsf{st}_w)\}_{w\in[n_w]})$

WLOG, assume $(\mathcal{P}_2, \mathcal{P}_3)$ are honest and $\mathcal{P}_1$ is corrupt, in which it uses a mutated search index $\mathsf{EIDX}'_w = \mathsf{EIDX}_w + \varepsilon$ (compared with $\mathsf{EIDX}_w$ in $\mathcal{P}_2$ and $\mathcal{P}_3$). As DPF and KH-PRF are computed on $\mathsf{EIDX}'_w$, there will be an error in $\mathcal{P}_1$'s computation, making the responses $O_{12}$, $O_{13} \neq O_{23}$.

By checking the consistency of $O_{12}$, $O_{13}$, and $O_{23}$, the reader can tell whether there is a rollback attack happens, and abort the protocol accordingly. Note that this strategy can only detect the rollback attack, a special case of malicious behavior on integrity, but is unable to tell which server is corrupted. Meanwhile, a malicious adversary can deviate from the protocol to not only compromise the integrity but also the privacy of the reader's query. That requires a more comprehensive investigation to completely achieve malicious security, which we leave as our future work.