MUSES: Efficient Multi-User Searchable Encrypted Database

Tung Le
Virginia Tech

Rouzbeh Behnia
University of South Florida, Sarasota

Jorge Guajardo
Robert Bosch LLC — RTC

Thang Hoang
Virginia Tech

ABSTRACT

Searchable encrypted systems enable privacy-preserving keyword search on encrypted data. Symmetric Searchable Encryption (SSE) achieves high security (e.g., forward privacy) and efficiency (i.e., sublinear search), but it only supports single-user. Public Key Searchable Encryption (PEKS) supports multi-user settings, however, it suffers from inherent security limitations such as being vulnerable to keyword-guessing attacks and the lack of forward privacy. Recent work has combined SSE and PEKS to achieve the best of both worlds: support multi-user settings, provide forward privacy while having sublinear complexity. However, despite their elegant design, the existing hybrid scheme inherits some of the security limitations of the underlying paradigms (e.g., patterns leakage, keyword-guessing) and might not be suitable for certain applications due to costly public-key operations (e.g., bilinear pairing).

In this paper, we propose MUSES, a new multi-user encrypted search scheme that addresses the limitations in the existing hybrid design, while offering user efficiency. Specifically, MUSES permits multi-user functionalities (reader/writer separation, permission revocation), prevents keyword-guessing attacks, protects search/result patterns, achieves forward/backward privacy, and features minimal user overhead. In MUSES, we demonstrate a unique incorporation of various state-of-the-art distributed cryptographic protocols including Distributed Point Function, Distributed PRF, and Secret-Shared Shuffle. We also introduce a new oblivious shuffle protocol for the general L-party setting with dishonest majority, which can be of independent interest. Our experimental results indicated that the keyword search in our scheme is two orders of magnitude faster with 13× lower user bandwidth overhead than the state-of-the-art.

KEYWORDS

Privacy-enhancing technologies; encrypted search; data privacy.

1 INTRODUCTION

Data outsourcing services (*e.g.*, Dropbox, Google Drive, MS OneDrive) have been increasingly prevalent because of their accessibility and convenience. Commodity cloud storage (e.g., AWS IAM, Google Cloud IAM) not only can provide users with data storage facilities, but also support a fine-grained access control for data sharing across a large number of users. Nevertheless, outsourcing data to external clouds might lead to privacy concerns, especially for sensitive data (*e.g.*, health and financial records). This is because a compromised cloud provider can access and exploit data illegitimately. Although end-to-end encryption can enable data confidentiality, it also prevents data utility (e.g., querying, analytics), thereby invalidating the benefits of data outsourcing services.

To address the data utilization and privacy dilemma, Searchable Encryption (SE) was proposed to enable keyword search on encrypted data while respecting the confidentiality of the data and the search query. There are two main lines of SE research including Symmetric SE (SSE) [10, 25, 27, 37, 42, 46, 67] and Public-Key SE (PEKS) [5, 8, 77]. While SSE offers high security guarantees (e.g., forward privacy [10, 37, 53, 68, 72], backward privacy [37, 53, 68, 72]), efficiency (e.g., sublinear search), and diverse query functionalities (e.g., range query [26, 50, 72]), it mainly supports the single-user setting, in which the data can only be searched by its owner. This strictly limits its practicality when adopted in real-world settings, where the data can be contributed by multiple users (*e.g.*, emails). On the other hand, PEKS enables encrypted search in the multi-user setting, in which one user (the reader) can search on encrypted documents sent/shared by the other users (the writers). Unfortunately, PEKS is known to suffer from various inherent security flaws including the lack of forward privacy and vulnerability to dictionary attacks. Meanwhile, forward privacy has been shown (via practical attack demonstrations [79]) to become a *de facto* requirement in SE for long-term security.

Recently, Wang et al. [73] proposed the notion of Hybrid SE (HSE), which elegantly combines SSE and PEKS to achieve the benefits of both SE paradigms: forward privacy and sublinear complexity in SSE, and multi-user functionalities in PEKS. Despite its merits and elegant design, the proposed HSE scheme accidentally inherits the security weaknesses of both worlds, including the *keyword-guessing attack* (KGA) vulnerabilities and *search/result pattern* leakage. In addition, the proposed HSE instantiation does not achieve backward privacy [11], which is necessary to prevent extra information leakage during search. Many devastating attacks (e.g., [4, 14, 44, 48, 52, 54, 56, 62, 63]) have shown that search/access pattern leakage reveals significant information about the query and the data even though they are both encrypted. While some techniques such as Oblivious RAM (ORAM) [66] can hide the search/result pattern in SE, they incur high bandwidth cost to the user(s) [61]. Given that HSE is still in the early stages and there is a lack of privacy-focused designs, we raise the following challenging and practically relevant question:

Can we design a new SSE scheme that not only supports multi-writer similar to HSE, but also addresses the inherent security limitations of the existing paradigms (i.e., in both SSE and PEKS), while achieving concrete efficiency?

Conference'17, July 2017, Washington, DC, USA
2023. ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

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

Scheme	**#Servers**	**Search Leakage**	**Update Leakage**	**Multi-**	**Search Complexity**			**Update Complexity**	
	L	$\mathcal{L}_{\text{Search}}$	$\mathcal{L}_{\text{Update}}$	**Writer**	**Server**	**Reader**	**Comm.**	**Writer**	**Comm.**
SSE [15]	1	$\{\text{sp}(w), \text{rp}(w), \text{sv}(w)\}$	$\{\text{up}(u)\}$	✗	$O(n_s)$	$O(\lambda)$	$O(\lambda + n_s)$	$O(W_u)$	$O(W_u + \lvert W \rvert)$
PEKS [8]	1	$\{w, \text{sp}(w), \text{rp}(w), \text{sv}(w)\}$	$\{\text{up}(u), (t, w_i) \in \mathbb{O}\}$	✓	$O(d_w N^{\dagger})$	$O(\lambda)$	$O(\lambda + n_s)$	$O(W_u)$	$O(W_u)$
DORY [25]	2	$\{\emptyset\}$	$\{\text{up}(u)\}$	✗	$O(mN)$	$O(\log m + N)$	$O(\lambda \log m + N)$	$O(m)$	$O(m)$
FP-HSE [73]	1	$\{w, \mathcal{W}', \text{sp}(w), \text{rp}(w), \text{sv}(w)\}$	$\{i, \text{up}(u)\}$	✓	$O(\lvert W \rvert^{\dagger})$	$O(\lambda)$	$O(\lambda + n_s)$	$O(W_u)$	$O(W_u)$
Our MUSES	≥ 2	$\{\mathcal{W}', \text{sv}(w)\}$	$\{i, \text{up}(u)\}$	✓	$O(mN)$	$O(\tau + N)$	$O(\lambda\tau + n_s)$	$O(m)$	$O(\lambda m)$

• $\mathcal{L}_{\text{Search}}$: Search leakage function with w as the input keyword and $\mathcal{W}'$ as the writer subset when a search happens; $\mathcal{L}_{\text{Update}}$: Update leakage function with i as the identifier of the writer, op as the operation type (add/delete), u as the updated document, and w (*resp.*, $\mathbf{w}$) is the keyword to be added/deleted (*resp.*, a vector of keywords in the updated document). sp: Search pattern; rp: Result pattern; sv: Search volume; up: Document update pattern; $\mathbb{O}$ is a sequence of operations and (t, w) as a search on keyword w at the timestamp t (see §3 for more details). λ: Security parameter; N: Total number of documents; d_w: Number of keywords per document. W: Total number of unique keywords over *all* documents (keyword universe); m: Size of keyword representation *per* document; n_s: Number of documents matched per keyword search; In practice, $d_w < m \ll W$, $n_s \ll N$. W_u: Number of updated keywords (added/deleted). Number of servers L is considered as a constant number in complexity analysis.
‡ $\tau = O(\log m)$ when $L = 2$, or $O(\sqrt{m})$ when $L \geq 3$.
We assume document identifiers can be represented using a constant number of bits to skip this quantity in search and update complexity.
In update, server(s) in the above schemes mostly perform(s) I/O operations, therefore update complexity analysis on the server side is bypassed.
† Public-key pairing operations.

1.1 Our Contributions

We answer the above question affirmatively by proposing MUSES, a new distributed multi-user SE database scheme that achieves a high level of security with concrete efficiency simultaneously. MUSES achieves the following desirable properties.

- **Multi-user functionalities**: MUSES allows multiple users with reader and writer(s) separation similar to multi-user PEKS/HSE schemes (e.g., [6, 8, 30, 73, 78]). 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 if necessary with writer-efficiency.
- **High security:** MUSES is secure against KGA, offers forward and backward privacy, and hides search/result patterns simultaneously. Thus, it offers a much higher security guarantee than most prior multi-user SE schemes ([8, 49, 73, 75, 77]). Our MUSES offers semi-honest security with dishonest majority. It can also support any number L of servers with $L - 1$ privacy degree guarantee, meaning the confidentiality of users' data and queries is protected as long as one (out of L) server is honest. To our knowledge, there is no prior PEKS/HSE scheme that achieves all the aforementioned security properties.
- **User-driven efficiency**: MUSES is designed with user efficiency in mind, and therefore, it is highly favorable to thin users with limited computing and network resources (e.g., mobile). In MUSES, the reader only performs lightweight operations (*e.g.*, modular additions), and the search bandwidth in MUSES is proportional to the number of matched documents, compared with linear w.r.t entire database in prior oblivious SE schemes (e.g., [25, 32, 42]). Evaluation results indicate that MUSES achieves up to 13× lower reader-bandwidth than state-of-the-art oblivious SE designs. On the other hand, the writer in MUSES can revoke the reader's access efficiently by offloading entire re-encryption task securely to the servers. This is more efficient than prior systems that do not naturally support revocation and require writers to re-encrypt the index themselves, which incurs high bandwidth for index transmission and computation cost.
- **Low server processing overhead**: In MUSES, the servers only perform low-cost operations (e.g., modular addition/multiplication, rounding over small modulus). Therefore, it is more efficient than prior designs that incur costly public-key operations (e.g., pairing [5, 6, 8, 30, 73, 77, 78]).
- **Fully-fledged implementation and evaluation**: We fully implemented our MUSES scheme and evaluated its performance on commodity servers. Experimental results demonstrate that our technique performs search 129.1×−137.2× faster than the state-of-the-art multi-user SE technique (i.e., [73]). MUSES is also faster than single-user SE counterparts (around 1.6×−1.7× faster than [25]) under limitted bandwidth settings. *Our implementation is available and ready to be publicly released for reproducibility, comparison, and adaptation (see the attached artifact).*

Technique: Our Multi-party secret-shared shuffle. To construct a new multi-user SE with user-driven efficiency, we come up with a new oblivious shuffle technique that can be of independent interest and can lead to other interesting applications. Specifically, we construct a generic multi-party secret-shared shuffle extended from the specific two-party one in [18], which permits multiple parties to obtain randomly shuffled data from their additive shares with dishonest majority.

Table 1 compares MUSES with the state-of-the-art SE designs in terms of security, functionality and complexity. To our knowledge, we are the first to propose a multi-user SE scheme that can achieve small leakage and high efficiency (optimal user bandwidth, low reader and writer overhead) simultaneously.

Why distributed servers? One might wonder why MUSES makes use of multiple servers rather than a single server as commonly used in SE platforms (e.g., [8, 15, 73]). This is due to the fact that in the multi-user setting with separate reader and writer roles, there are inherent security vulnerabilities that cannot be prevented with a single server. For instance, in the "rollback" attacks [33, 43, 47, 55, 59, 71], the malicious server can omit some update of the writer, and present the old version of the writer's data to the reader. Preventing such realistic attacks requires coordination and communication between the reader and the writer, or a separate system for integrity such as blockchain [43, 71]. Given that it is costly to deploy such a dedicated system, we use multiple servers to not only prevent these attacks but also gain performance benefits of underlying cryptographic building blocks (e.g., Distributed Point Function, distributed PRF) used in our scheme. Also note that although we only present the semi-honest MUSES scheme in the main body, it is straightforward to make it secure against rollback attacks (by using more servers). We present such an extension in Appendix C.

1.2 Technical Highlights

We present the technical highlights of our construction. MUSES is inspired by DORY [25], a symmetric SE scheme, and HSE [73], a framework that provides a generic transformation guideline to adopt symmetric SE in the multi-user setting. We begin by giving DORY's overview, outline the challenges when transforming DORY to multi-user setting, and then present our high-level idea to address these challenges.

Brief Overview of DORY. DORY is a (single-client) encrypted search scheme that supports oblivious search and update capabilities. Its high-level idea is to instantiate the index for searchable keyword representation with a table data structure, in which the keyword search operation occurs in one dimension (e.g., column), while the document update occurs in the other dimension (i.e., row). To reduce the index size, Bloom Filter (BF) is employed to compress the keyword representation in the document, resulting in the total index size of $O(N \cdot m)$, where N and m is the number of documents and the BF representation size, respectively. For confidentiality, the search index is row-wise encrypted for efficient update. To search for a keyword w, the user computes the BF representation of w, and uses Private Information Retrieval (PIR) based on Distributed Point Function (DPF) to retrieve K encrypted columns in the search index that are indicated by the BF representation (i.e., K is the number of elements "1" in the BF representation). To update a document, the user replaces the corresponding row in the search index with a new (encrypted) BF representation for all the keywords in the updated document.

In principle, DORY can be extended to support multi-user setting (separate reader/writer) by incorporating public-key cryptography (as suggested in [73]) to distribute the key that is used to encrypt the search index to the reader. However, it incurs inherit performance limitations due to its underlying cryptographic building blocks. Specifically, the reader incurs a linear complexity with respect to the document collection size (i.e., $O(N \cdot K)$) in terms of both network bandwidth and computation overhead (due to the PIR, aggregation, and decryption sequence) to obtain the search result. This overhead is significant especially in the context of thin reader (e.g., mobile) and large-scale database (e.g., email), while an efficient search should only return a small subset of the matched documents.

Second, as the symmetric key is shared with the reader directly, it may not be easy to enable access control functionalities. 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. Our first idea is that instead of performing decryption and aggregation after retrieving K encrypted columns via PIR, the reader can delegate all these processing tasks to the server in a privacy-preserving manner. Specifically, we utilize multiple servers and develop a new protocol that is well-coordinated with Key-Homomorphic Pseudorandom Function (KH-PRF) [9] and DPF-based PIR [12, 13, 38] together, which permits each server to "partially" decrypt the encrypted columns and perform secure aggregation, respectively. At the end of our protocol, each server obtains a share of the final search result and therefore, they can exchange their shares together to reconstruct the result and return it to the reader. Our protocol ensures the servers do not learn anything (e.g., what columns are being aggregated/decrypted, decryption keys) apart from the final search result, given that all of them do not collude with each other.

While this strategy reduces the reader's processing and bandwidth overhead, it permits the server to learn the particular document identifiers that match the search query. These so-called *result patterns* permit an adversarial server to infer the search pattern (e.g., whether the same/different keywords are being searched). *Can we hide such result patterns while still maintaining the reader efficiency?*

Idea 2: Conceal result/search patterns via random shuffling. To hide result patterns, our idea is to perform a random shuffling on the shares of the (aggregated decrypted) search result across multiple servers before they come together to open the final output. We construct a generic L-party secret-shared shuffling technique with dishonest majority based on [18], which enables L parties to randomly shuffle secret-shared data in a way that one party learns the final shuffled result, while each other party learns a permutation in a composition[1] of $L-1$ permutations. We apply our L-party shuffle protocol to obfuscate the order of the search result, where one server learns the obfuscated document identifiers and each of the other servers learns a permutation in a permutation composition. Finally, the servers can individually send the obfuscated list and the permutation information to the reader so that he can obtain the final search result by computing the permutation inverse on the obfuscated list. This strategy slightly increases the computation overhead for the reader due to permutation inversion; however, the communication complexity $O(n_s)$ (where $n_s \ll N$ is the number of matched documents) is still maintained.

Idea 3: Minimize writer overhead in revoking reader' permission via "key rotation" on the servers. To revoke the reader's search ability, 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. We denote by λ the security parameter and by $\mathbb{Z}_p$ the ring of integers modulo p. We denote by $[n]$ the set $\{1, \ldots, n\}$. $x \overset{\$}{\leftarrow} [n]$ means x is selected uniformly at random from the set $\{1, \ldots, n\}$. 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 $i \cdot \lfloor q/p \rfloor$ is the largest multiple of $\lfloor q/p \rfloor$ that

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

does not exceed x. Bold small letters denote vectors, *i.e.*, $\mathbf{a} \in \mathbb{Z}_p^n$. We denote by $\langle \mathbf{a}, \mathbf{b} \rangle$ the dot product of two vectors $\mathbf{a}$ and $\mathbf{b}$. Capitalized bold letters denote matrices, *i.e.*, $\mathbf{M} \in \mathbb{Z}_p^{n \times m}$. Given a matrix $\mathbf{M}$, $\mathbf{M}[i, *]$ and $\mathbf{M}[*, j]$ denote accessing the row i and column j of $\mathbf{M}$, respectively. $\mathbf{M}[i, j]$ denotes accessing the cell indexed at row i and column j. We denote π as a permutation and π^{-1} as its inverse such that $\pi^{-1}(\pi(x)) = x$. 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 $\mathcal{E} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ be an IND-CPA symmetric encryption scheme: $\kappa \leftarrow \mathcal{E}.\mathsf{Gen}(1^\lambda)$ generating a key with security parameter λ; $c \leftarrow \mathcal{E}.\mathsf{Enc}(\kappa, \mathsf{ct}, m)$ encrypting plaintext m with key κ and counter ct; $m \leftarrow \mathcal{E}.\mathsf{Dec}(\kappa, \mathsf{ct}, c)$ decrypting ciphertext c with key κ and counter ct. Let $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ be a public-key encryption scheme: $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathcal{E}.\mathsf{Gen}(1^\lambda)$ generating a public and private-key pair with security parameter λ; $c \leftarrow \Pi.\mathsf{Enc}(\mathsf{pk}, m)$ encrypting plaintext m with public key pk; $m \leftarrow \Pi.\mathsf{Dec}(\mathsf{sk}, c)$ decrypting ciphertext c with private key sk. Let $\mathsf{BF} = (\mathsf{Init}, \mathsf{Gen}, \mathsf{Vrfy})$ be a Bloom Filter (BF) [7]: $(H_1, \ldots, H_K) \leftarrow \mathsf{BF.Init}(m, K)$: 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 whether an element s belongs to the set represented by BF vector $\mathbf{u}$.

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 $x = \sum_{i=1}^{L} x^{(i)} (\bmod\ p)$.

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.

2.1 Distributed Point Function

Distributed Point Function (DPF) [12, 13, 38] permits L parties to jointly evaluate a point function. For $a, b \in \{0,1\}^*$, let $P_{a,b} : \{0,1\}^{|a|} \to \{0,1\}^{|b|}$ such that $P_{a,b}(a) = b$ and $P_{a,b}(a') = 0^{|b|}\ \forall\ a' \neq a$. A DPF scheme contains the following PPT algorithms.

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

An application of DPF is to implement efficient private information retrieval (PIR). We present an L-party DPF-based PIR scheme to retrieve an item $\mathbf{b}_j$ in $\mathbf{B} = (\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_m)$ as follows. The client creates L keys $(k^{(1)}, \ldots, k^{(L)}) \leftarrow \mathsf{DPF.Gen}(1^\lambda, j, 1)$ for L parties, where $k^{(1)}, \ldots, k^{(L)} \in \{0,1\}^n$ ($n = O(\lambda \log m)$ for $L = 2$, or $n = O(\lambda\sqrt{m})$ for $L \geq 3$) and $k^{(\ell)}$ is sent to party $\mathcal{P}_\ell$ ($\ell \in [L]$). 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 [9] enables distributed evaluation of a secure PRF function $F^* : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$ such that $(\mathcal{K}, *)$ and $(\mathcal{Y}, \bullet)$ are both groups and for every $k_1, k_2 \in \mathcal{K}$, $F^*(k_1 * k_2, x) = F^*(k_1, x) \bullet F^*(k_2, x)$. We define an L-party KH-PRF scheme as a tuple of PPT algorithms $\mathsf{KH\text{-}PRF} = (\mathsf{Gen}, \mathsf{Share}, \mathsf{Eval})$ as follows.

- $\underline{\mathbf{k} \leftarrow \mathsf{KH\text{-}PRF.Gen}(1^\lambda)}$: Given a security parameter λ, it outputs a secret key $\mathbf{k} \in \mathcal{K}$.
- $\underline{(\mathbf{k}^{(1)}, \ldots, \mathbf{k}^{(L)}) \leftarrow \mathsf{KH\text{-}PRF.Share}(\mathbf{k})}$: Given a key $\mathbf{k} \in \mathcal{K}$, it outputs L keys $\mathbf{k}^{(1)}, \ldots, \mathbf{k}^{(L)} \in \mathcal{K}$ such that $\mathbf{k}^{(1)} * \cdots * \mathbf{k}^{(L)} = \mathbf{k}$.
- $\underline{y \leftarrow \mathsf{KH\text{-}PRF.Eval}(\mathbf{k}, s)}$: 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 present the extension of the 2-party (almost) KH-PRF scheme based on Learning with Rounding (LWR) under Random Oracle Model (ROM) by Boneh et al. [9] into L-party setting. Let $H_2 : \{0,1\}^* \to \mathbb{Z}_q^n$ be a hash function modeled as a random oracle. The KH-PRF function $F^* : \mathbb{Z}_q^n \times \{0,1\}^* \to \mathbb{Z}_p$ is defined as $F^*(\mathbf{k}, s) = \lfloor \langle H_2(s), \mathbf{k} \rangle \rfloor_p$, where $\mathbf{k}^{(1)} + \cdots + \mathbf{k}^{(L)} = \mathbf{k}$, F^* is an almost key homomorphic in the sense that $F^*(\mathbf{k}, s) = e + \sum_{\ell=1}^{L} F^*(\mathbf{k}^{(\ell)}, s) \pmod p$ where e is a small error, i.e., $e \in \{0, \ldots, L\}$.

2.3 Secret-Shared Shuffle

Secret-shared shuffle permits multiple parties to obliviously shuffle a set and obtain additive secret shares of the result. We recall a Two-party Secret-shared Shuffle (TSS) [18], which permits two parties $\mathcal{P}_1, \mathcal{P}_2$ to jointly shuffle a set and obtain two additive shares. TSS scheme in [18] is a tuple of PPT algorithms TSS = (Gen, ShrTrns, Shffl) defined as follows.

- $\underline{\pi \leftarrow \mathsf{TSS.Gen}(1^\lambda, n)}$: Given a security parameter λ and a set size n, it outputs a pseudorandom permutation π for n elements.
- $\underline{(\mathbf{\Delta}; \mathbf{a}, \mathbf{b}) \leftarrow \mathsf{TSS.ShrTrns}(\pi; 1^\lambda)}$: Given a permutation π for n elements to $\mathcal{P}_1$, and a security parameter λ to $\mathcal{P}_2$, it outputs $\mathbf{\Delta} = \mathbf{b} - \pi(\mathbf{a}) \in \mathbb{Z}_p^n$ to $\mathcal{P}_1$ and $\mathbf{a}, \mathbf{b} \in \mathbb{Z}_p^n$ to $\mathcal{P}_2$.
- $\underline{(\mathbf{x}'; \mathbf{b}) \leftarrow \mathsf{TSS.Shffl}(\pi, \mathbf{\Delta}; \mathbf{x}, \mathbf{a}, \mathbf{b})}$: Given a permutation π and its corresponding $\mathbf{\Delta}$ 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 $\mathcal{P}_2$.

3 MODELS

System Model. Our system consists of a reader, n_w independent writers, and L servers. WLOG, we identify each writer as a member of $[n_w]$ so that $\mathcal{W} = [n_w]$. Each writer $i \in \mathcal{W}$ owns a separate collection of N documents and would like to share it with the reader. We identify each document in the database as a member of $[N]$. We consider the reader would like to perform encrypted keyword search over all document collections of a writer subset $\mathcal{W}' \subseteq \mathcal{W}$. On the other hand, the writer can revoke the permission of the reader if needed. The reader and writers are independent parties and they do not have to communicate directly with each other. Our scheme is a Multi-User SE scheme (MUSES) defined as follows.

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

- $\underline{(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{RSetup}(1^\lambda)}$*: Given a security parameter λ, it outputs a public and private key pair* $(\mathsf{pk}, \mathsf{sk})$.
- $\underline{(\kappa_{w_i}, \mathsf{EIDX}_i, \mathsf{st}_i, \mathsf{STkn}_i, \mathsf{PTkn}_i) \leftarrow \mathsf{WSetup}(1^\lambda, i, \mathsf{pk})}$*: Given a security parameter λ, a writer identifier i, and the public key* pk*, it outputs a writer key κ_{w_i}, an encrypted search index* EIDX_i*, a state* st_i*, a secret token* STkn_i *encrypted under* κ_{w_i}*, and a private token* PTkn_i *encrypted under* pk.

- $\mathfrak{s} \leftarrow \mathsf{SearchToken}(w, \mathcal{W}')$: *Given a keyword w and a subset of writers $\mathcal{W}'$, it outputs a search token $\mathfrak{s}$.*
- $O \leftarrow \mathsf{Search}(\mathfrak{s}, \mathsf{sk}, \{(i, \mathsf{st}_i, \mathsf{EIDX}_i, \mathsf{PTkn}_i)\}_{i\in[n_w]})$: *Given a search token $\mathfrak{s}$, the reader's private key sk, a set of tuples $(i, \mathsf{st}_i, \mathsf{EIDX}_i, \mathsf{PTkn}_i)$ as the identifier, the state, the encrypted search index and the private token, respectively, of writers $i \in [n_w]$, it outputs the search result O.*
- $\mathfrak{u} \leftarrow \mathsf{UpdateToken}(\mathbf{w}, u, i, \kappa_{w_i}, \{\mathsf{st}_i, \mathsf{STkn}_i\}_{i\in[n_w]})$: *Given a set of keywords $\mathbf{w}$ for a document u, a writer identifier i, a writer key κ_{w_i}, states st_i, and secret tokens STkn_i of writers $i \in [n_w]$, it outputs an update token $\mathfrak{u}$.*
- $(\mathsf{EIDX}'_i, \mathsf{st}'_i) \leftarrow \mathsf{Update}(\mathfrak{u}, \{\mathsf{EIDX}_i, \mathsf{st}_i\}_{i\in[n_w]})$: *Given an update token $\mathfrak{u}$, encrypted indices EIDX_i and states st_i of writers $i \in [n_w]$, it outputs updated search index EIDX'_i and updated state st'_i.*
- $(\mathsf{EIDX}'_i, \mathsf{STkn}'_i) \leftarrow \mathsf{RevokePerm}(i, \kappa_{w_i}, \{\mathsf{EIDX}_i, \mathsf{st}_i, \mathsf{STkn}_i\}_{i\in[n_w]})$: *Given a writer identifier i and the corresponding writer's secret key κ_{w_i}, encrypted search indices EIDX_i, states st_i, and secret tokens STkn_i of writers $i \in [n_w]$, it outputs updated search index EIDX'_i, and updated secret token STkn'_i.*

Definition 2 (Correctness of MUSES). *For all λ, $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{RSetup}(1^\lambda)$, $(\kappa_{w_i}, \mathsf{st}_i, \mathsf{EIDX}_i, \mathsf{STkn}_i, \mathsf{PTkn}_i) \leftarrow \mathsf{WSetup}(1^\lambda, i, \mathsf{pk})$, $i \in \mathcal{W}$, and all sequences of* Search, Update *operations over $\{\mathsf{EIDX}_i\}_{i\in[n_w]}$ using tokens generated respectively from* $\mathsf{SearchToken}(w, \mathcal{W}')$*, and* $\mathsf{UpdateToken}(\mathbf{w}, u, i, \kappa_{w_i}, \{\mathsf{st}_i, \mathsf{STkn}_i\}_{i\in[n_w]})$*, as well as* RevokePerm *operations over $i \in \widetilde{\mathcal{W}} \subseteq \mathcal{W}$,* Search *returns the correct results w.r.t the inputs $(\mathbf{w}, u, i)$ of* UpdateToken *when $i \in \mathcal{W}' \setminus \widetilde{\mathcal{W}}$, except with negligible probability in λ.*

Threat and Security Models. We assume the adversary can corrupt up to $L-1$ out of the L servers, and an arbitrary 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 $\mathsf{EIDX}_i = (\mathcal{I}_{i,1}, \mathcal{I}_{i,2}, \ldots, \mathcal{I}_{i,N})$ be an encrypted search index of writer i, where $\mathcal{I}_{i,u}$ contains information about keywords of the u-th document. Let $\mathbb{O}_i$ be a sequence of operations on EIDX_i. We denote t as the timestamp when an operation happens, $\mathbb{O}_i$ records (t, w) for a search on keyword w, and $(t, u, \mathbf{w})$ for an update of document u with its new keywords $\mathbf{w}$ on EIDX_i.

Definition 3 (Search Pattern [23, 73]). *The search pattern* sp *indicates the frequency of search operations on some keywords, i.e.,* $\mathsf{sp}(w) = \{t : (t, w) \in \mathbb{O}_i)\}$.

Definition 4 (Result Pattern). *The result pattern* rp *reveals what documents match the queried keyword w, i.e.,* $\mathsf{rp}(w) = \{u_1, \ldots, u_{N'} : w \in I_{i,u_l} \, \forall\, l \in [N'] \subseteq [N]\}$.

Definition 5 (Search Volume). *The search volume* sv *indicates the number of documents matching the queried keyword, i.e.,* $\mathsf{sv}(w) = N'$ *s.t.* $w \in I_{i,u} \;\forall\; u \in [N'] \subseteq [N]$.

Definition 6 (Document Update Pattern). *The update pattern* up *records the update frequency on documents, i.e.,* $\mathsf{up}(u) = \{t : (t, u, \perp) \in \mathbb{O}_i\}$

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 secret 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.

```
IND^b_{MUSES,A,L}(1^λ):
1: $(n_r, n_w) \leftarrow \mathcal{A}(1^\lambda)$
2: $(\mathsf{pk}, \mathsf{msk}) \leftarrow \mathsf{SysInit}(1^\lambda, n_w)$
3: $(\kappa_{w_i}, \mathsf{EIDX}_i, \mathsf{st}_i, \mathsf{STkn}_i, \mathsf{PTkn}_i) \leftarrow \mathsf{WSetup}(1^\lambda, i, \mathsf{pk}),\ \forall\, i \in [n_w]$
4: $\kappa_{r_j} \leftarrow \mathsf{RSetup}(\mathcal{W}'_j, \mathsf{msk}),\ \forall\, j \in [n_r]$
5: $\mathcal{H}_0 \leftarrow \{\emptyset\}, \mathcal{H}_1 \leftarrow \{\emptyset\}$
6: $O \leftarrow \{\mathsf{CorruptWriterO}_b, \mathsf{SearchO}_b, \mathsf{UpdateO}_b, \mathsf{RevokeO}_b\}$
7: $b' \leftarrow \mathcal{A}^O(\mathsf{pk}, \{\mathsf{EIDX}_i, \mathsf{st}_i, \mathsf{STkn}_i, \mathsf{PTkn}_i\}_{i\in[n_w]})$
8: return $b'$

$\mathsf{SearchO}_b(\{w_k, \mathcal{W}'_k\}_{k\in\{0,1\}})$:
1: if $\mathcal{L}^{\mathsf{Search}}_{\mathcal{H}_0}(w_0, \mathcal{W}'_0) = \mathcal{L}^{\mathsf{Search}}_{\mathcal{H}_1}(w_1, \mathcal{W}'_1)$
2:     $\forall\, k \in \{0,1\}, \mathcal{H}_k \leftarrow \mathcal{H}_k \cup (\mathsf{Search}, w_k, \mathcal{W}'_k)$
3:     return $\mathsf{SearchToken}(w_b, \mathcal{W}'_b)$
4: else return $\perp$

$\mathsf{UpdateO}_b(\{i_k, u_k, \mathbf{w}_k\}_{k\in\{0,1\}})$:
1: if $\mathcal{L}^{\mathsf{Update}}_{\mathcal{H}_0}(i_0, u_0, \mathbf{w}_0) = \mathcal{L}^{\mathsf{Update}}_{\mathcal{H}_1}(i_1, u_1, \mathbf{w}_1)$
2:     $\forall\, k \in \{0,1\}, \mathcal{H}_k \leftarrow \mathcal{H}_k \cup (\mathsf{Update}, i_k, u_k, \mathbf{w}_k)$
3:     return $\mathsf{UpdateToken}(\mathbf{w}_b, u_b, i_b, \kappa_{w_{i_b}}, \mathsf{st}_{i_b}, \mathsf{STkn}_{i_b})$
4: else return $\perp$

$\mathsf{RevokeO}_b(\{i_k\}_{k\in\{0,1\}})$:
1: if $\mathcal{L}^{\mathsf{Revoke}}_{\mathcal{H}_0}(i_0) = \mathcal{L}^{\mathsf{Revoke}}_{\mathcal{H}_1}(i_1)$
2:     $\forall\, k \in \{0,1\}, \mathcal{H}_k \leftarrow \mathcal{H}_k \cup (\mathsf{Revoke}, i_k)$
3:     return $(\mathsf{EIDX}'_{i_b}, \mathsf{STkn}'_{i_b})$
4: else return $\perp$

$\mathsf{CorruptWriterO}_b(i_0, i_1)$:
1: if $\mathcal{L}^{\mathsf{CorruptWriter}}_{\mathcal{H}_0}(i_0) = \mathcal{L}^{\mathsf{CorruptWriter}}_{\mathcal{H}_1}(i_1)$
2:     $\forall\, k \in \{0,1\}, \mathcal{H}_k \leftarrow \mathcal{H}_k \cup (\mathsf{CorruptWriter}, i_k)$
3:     return $\kappa_{w_{i_b}}$
4: else return $\perp$
```

Figure 1: Security Game for MUSES

Definition 7 (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* $\mathsf{Hist}_t \in \{(\mathsf{CorruptWriter}, i), (\mathsf{Search}, w, \mathcal{W}'), (\mathsf{Update}, i, u, \mathbf{w}), (\mathsf{Revoke}, i)\}$.

We introduce a leakage function family $\mathcal{L}_{\mathcal{H}} = \{\mathcal{L}^{\mathsf{Setup}}_{\mathcal{H}}, \mathcal{L}^{\mathsf{Search}}_{\mathcal{H}}, \mathcal{L}^{\mathsf{Update}}_{\mathcal{H}}, \mathcal{L}^{\mathsf{Revoke}}_{\mathcal{H}}, \mathcal{L}^{\mathsf{CorruptWriter}}_{\mathcal{H}}\}$ 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.*, $\mathcal{H} = \{\emptyset\}$), $\mathcal{L}_{\mathcal{H}} = \mathcal{L}^{\mathsf{Setup}}_{\mathcal{H}}$. We also implicitly assume that MUSES histories are *non-singular* as defined in [46, 73].

Definition 8 (Adaptive Security of MUSES). *For all PPT adversary $\mathcal{A}$ and the game* $\mathsf{IND}^b_{MUSES,\mathcal{A},\mathcal{L}}(1^\lambda)$ *in Figure 1, MUSES is*

$\mathcal{L}_{\mathcal{H}}$-adaptively-secure if:
$|\Pr[\mathsf{IND}^0_{\mathsf{MUSES},\mathcal{A},\mathcal{L}}(1^\lambda) = 1] - \Pr[\mathsf{IND}^1_{\mathsf{MUSES},\mathcal{A},\mathcal{L}}(1^\lambda) = 1]| \leq \mathsf{negl}(1^\lambda)$.

Corruption leakage. To capture corruption leakage, we introduce the following function:

- $\mathsf{UpdateBy}(i)$: This function lists all updates by the writer i in the history. Formally, $\mathsf{UpdateBy}(i) = \{\mathsf{Hist}_t : \mathsf{Hist}_t = (\mathsf{Update}, i, u, \mathbf{w}) \in \mathcal{H}\}$.

Definition 9 (Forward Privacy of MUSES). *An $\mathcal{L}_{\mathcal{H}}$-adaptively-secure MUSES is forward-private if the update leakage $\mathcal{L}^{\mathsf{Update}}_{\mathcal{H}}(i, u, \mathbf{w})$ of any update $(i, u, \mathbf{w})$ by writer i s.t. $(\mathsf{CorruptWriter}, i) \notin \mathcal{H}$ can be written as $\mathcal{L}'(i, u)$ or $\mathcal{L}''(i)$, where $\mathcal{L}'$, $\mathcal{L}''$ are stateless functions.*

Definition 10 (Backward Privacy of MUSES). *An $\mathcal{L}_{\mathcal{H}}$-adaptively-secure MUSES is backward-private if the search and update leakage functions $\mathcal{L}^{\mathsf{Search}}_{\mathcal{H}}, \mathcal{L}^{\mathsf{Update}}_{\mathcal{H}}$ can be written as $\mathcal{L}^{\mathsf{Update}}_{\mathcal{H}}(i, u, \mathbf{w}) = \mathcal{L}''(i, u) = \mathcal{L}'''(i)$, $\mathcal{L}^{\mathsf{Search}}_{\mathcal{H}}(w, \mathcal{W}') = \mathcal{L}'(\mathcal{W}')$, where $\mathcal{L}'$, $\mathcal{L}''$, and $\mathcal{L}'''$ are stateless functions.*

Document retrieval/update leakage. In this paper, we focus only on the security of the search index component in encrypted search. Retrieving/updating actual documents from the writer's database is out-of-scope. Sealing leakage from actual document retrieval/update is an independent study and some oblivious techniques (e.g., PIR [57, 58], ORAM [19–21]) can be applied orthogonally to our scheme to achieve system-wide end-to-end security.

4 OUR PROPOSED SCHEME

We introduce our new building block for L-party oblivious shuffle. We then present our proposed MUSES scheme.

4.1 New Building Block: L-party Shuffle

We present a generic L-party shuffle protocol extended from the two-party shuffle in [18] that permits L parties to randomly shuffle their L-additive shares of a data vector together in a way that one party learns the output of the permuted data while each of other parties learns a permutation of a permutation composition. Note that this input/output of our scheme is opposite to that of the two-party protocol in [18], wherein one party has a plaintext data vector $(\mathbf{x})$ as input, and the output is the shares of permuted data, i.e., $\mathcal{P}_1$ has $\pi(\mathbf{x}) + \mathbf{b}$ and $\mathcal{P}_2$ has $\mathbf{b}$.

Recall that in the two-party shuffle in [18], there is a preprocessing phase, where one party $\mathcal{P}_2$ generates two random masking vectors $\mathbf{a}$ and $\mathbf{b}$ and interacts with the other party $\mathcal{P}_1$ owning a random permutation π in a way that $\mathcal{P}_1$ learns a translation function Δ demonstrating the relation of $\mathbf{b}$ and the permutation of $\mathbf{a}$ with respect to π, i.e., $\Delta = \mathbf{b} - \pi(\mathbf{a})$.

To extend into L-party shuffle setting, our high-level idea for precomputation is to have each party $\mathcal{P}_i$ with $1 \leq i < L$ obtain the translation function Δ_i w.r.t its chosen random permutation π_i, each party $\mathcal{P}_i$ with $2 \leq i \leq L$ keeps the share of the mask vector $\mathbf{a}_i$, and the last party $\mathcal{P}_L$ keeps a mask vector $\mathbf{b}_L$ such that:

$$\pi_{L-1}\Big(\dots\big(\pi_2(\Delta_1) + \Delta_2\big)\dots\Big) + \Delta_{L-1} = \pi_{L-1}\Big(\dots\Big(\pi_2\big(\pi_1(\textstyle\sum_{i=2}^{L}\mathbf{a}_i)\big)\Big)\dots\Big) + \mathbf{b}_L.$$

Specifically, each party $\mathcal{P}_i$ $(1 \leq i < L)$ first generates independent permutation π_i then interacts with each other (as well as with party $\mathcal{P}_L$) to compute the translation function Δ_i such that:

```
LSS.ShrTrns($1^\lambda, n$):
 1: Parties $\mathcal{P}_i : (\pi_i; \perp) \leftarrow$ TSS.Gen($1^\lambda, n$), for $i \in [L-1]$
 2: for $i = 1$ to $L-1$ do
 3:     for $j = i+1$ to $L$ do
 4:         $\mathcal{P}_i \leftrightarrow \mathcal{P}_j$: $(\Delta_i^{(j)}; \mathbf{a}_j^{(i)}, \mathbf{b}_j^{(i)}) \leftarrow$ TSS.ShrTrns($\pi_i; 1^\lambda$)
 5: for $i = 3$ to $L$ do
 6:     for $j = 2$ to $i-1$ do
 7:         $\mathcal{P}_i \rightarrow \mathcal{P}_j$: $\Delta_j'^{(i)} \leftarrow \mathbf{b}_i^{(j-1)} - \mathbf{a}_i^{(j)}$
 8: for $i = 1$ to $L-1$ do
 9:     $\mathcal{P}_i$: $\Delta_i \leftarrow \sum_{j=i+1}^{L} \Delta_i^{(j)} \pmod p$
10:     if $i > 1$ then
11:         $\mathcal{P}_i$: $\Delta_i \leftarrow \Delta_i - \pi_i(\sum_{j=i+1}^{L} \Delta_i'^{(j)}) - \pi_i(\mathbf{b}_i^{(i-1)}) \pmod p$
12: return $(\pi_1, \Delta_1; \pi_2, \Delta_2, \mathbf{a}_2^{(1)}; \dots; \mathbf{a}_L^{(1)}, \mathbf{b}_L^{(L-1)})$

LSS.Shffl($\mathbf{x}^{(1)}, \pi_1, \Delta_1; \mathbf{x}^{(2)}, \pi_2, \Delta_2, \mathbf{a}_2^{(1)}; \dots; \mathbf{x}^{(L)}, \mathbf{a}_L^{(1)}, \mathbf{b}_L^{(L-1)}$):
 1: $\mathcal{P}_i \rightarrow \mathcal{P}_1$: $\mathbf{z}_i \leftarrow \mathbf{x}^{(i)} + \mathbf{a}_i^{(1)} \pmod p$, for $i = 2, \dots, L$
 2: $\mathcal{P}_1$: $\mathbf{o}_1 \leftarrow \mathbf{x}^{(1)} + \sum_{i=2}^{L} \mathbf{z}_i \pmod p$
 3: for $i = 1, \dots, L-1$ do
 4:     $\mathcal{P}_i \rightarrow \mathcal{P}_{i+1}$: $\mathbf{o}_{i+1} \leftarrow \pi_i(\mathbf{o}_i) + \Delta_i \pmod p$
 5: $\mathcal{P}_L$: $\mathbf{r} \leftarrow \mathbf{o}_L - \mathbf{b}_L^{(L-1)} \pmod p$
 6: return $(\pi_1; \pi_2; \dots; \pi_{L-1}; \mathbf{r})$
```

Figure 2: Our L-party Oblivious Secret-Shares Shuffle (LSS).

$$\Delta_i = \mathbf{x}_{i+1} - \pi_i(\mathbf{x}_i) \tag{1}$$

with:

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

where $\mathbf{a}_j^{(i)}$ and $\mathbf{b}_j^{(i)}$ are the shares of random masks known by party $\mathcal{P}_j$ while its corresponding translation function $\Delta_i^{(j)} = \mathbf{b}_j^{(i)} - \pi_i(\mathbf{a}_j^{(i)})$ is known by party $\mathcal{P}_i$. To achieve this, we execute the two-party share translation protocol between $\mathcal{P}_i$ and $\mathcal{P}_j$, for $j = i+1, \dots, L$, to generate $\Delta_i^{(j)} = \mathbf{b}_j^{(i)} - \pi_i(\mathbf{a}_j^{(i)})$ for $\mathcal{P}_i$, and $\mathbf{a}_j^{(i)}, \mathbf{b}_j^{(i)}$ for $\mathcal{P}_j$ (Figure 2, LSS.ShrTrns algorithm, lines 1–4). Then, for each party $\mathcal{P}_i$ $(2 \leq i < L)$, each party $\mathcal{P}_j$, for $j = i+1, \dots, L$, sends $\Delta_i'^{(j)} = \mathbf{b}_j^{(i-1)} - \mathbf{a}_j^{(i)}$ to $\mathcal{P}_i$ (lines 5–7). From $\Delta_i^{(j)}$ and $\Delta_i'^{(j)}$, each party $\mathcal{P}_i$ $(1 \leq i < L)$ can obtain Δ_i as shown in (1) (lines 8–11).

Upon completing the above precomputation, the parties can shuffle their secret-shared data as follows. Let $\mathbf{d}^{(\ell)}$ be the share of $\mathbf{d}$ held by $\mathcal{P}_\ell$. First, each party $\mathcal{P}_\ell$ $(\ell \geq 2)$ sends its mask value $\mathbf{z}_\ell \leftarrow \mathbf{d}^{(\ell)} + \mathbf{a}_\ell^{(1)}$ to $\mathcal{P}_1$ (Figure 2, LSS.Shffl algorithm, line 1). Then $\mathcal{P}_1$ computes $\mathbf{o}_1 \leftarrow \mathbf{d}^{(1)} + \sum_{\ell=2}^{L} \mathbf{z}_\ell$, and forwards $\mathbf{o}_2 \leftarrow \pi_1(\mathbf{o}_1) + \Delta_1 = \pi_1(\mathbf{d}) + \sum_{\ell=2}^{L} \mathbf{b}_\ell^{(1)}$ to $\mathcal{P}_2$ (line 2), who in turn computes $\mathbf{o}_3 \leftarrow \pi_2(\mathbf{o}_2) + \Delta_2 = \pi_2(\pi_1(\mathbf{d})) + \sum_{\ell=3}^{L} \mathbf{b}_\ell^{(2)}$ and forwards it to $\mathcal{P}_3$, and so on (lines 3–4). The above process continues until the final party $\mathcal{P}_L$ receives $\mathbf{o}_L = \pi_{L-1}(\dots(\pi_1(\mathbf{d})\dots)) + \mathbf{b}_L^{(L-1)}$ from $\mathcal{P}_{L-1}$. Finally, as $\mathcal{P}_L$ holds the mask $\mathbf{b}_L^{(L-1)}$, it can compute $\mathbf{r} \leftarrow \mathbf{o}_L - \mathbf{b}_L^{(L-1)} = \pi_{L-1}(\dots(\pi_1(\mathbf{d})))$, which is the permutation of $\mathbf{d}$ (line 5).

As each party $\mathcal{P}_i$ holds an independent permutation, and the final vector (when sent to $\mathcal{P}_L$) is shuffled with $L-1$ independent permutations and masked with a random vector, it is easy to see our L-party shuffle achieves dishonest majority security in the sense that the collusion of $L-1$ parties does not learn the information of the whole permutation sequence applied on the data vector.

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

WSetup(1^λ, i, pk):
1: $\kappa_{w_i} \leftarrow \mathcal{E}.\mathsf{Gen}(1^\lambda)$
2: for $v = 1$ to $m$ do
3:     $\mathbf{r}_{i,v} \leftarrow \mathsf{KH\text{-}PRF.Gen}(1^\lambda)$
4:     $\mathsf{STkn}_{i,v} \leftarrow \mathcal{E}.\mathsf{Enc}(\kappa_{w_i}, v, \mathbf{r}_{i,v})$
5:     $\mathsf{PTkn}_{i,v} \leftarrow \Pi.\mathsf{Enc}(\mathsf{pk}, \mathbf{r}_{i,v})$
6: $\mathsf{st}_i \leftarrow (\mathsf{st}_{i,1}, \mathsf{st}_{i,2}, \dots, \mathsf{st}_{i,N})$, where $\mathsf{st}_{i,u} \leftarrow 0$, for $u \in [N]$
7: $\mathsf{EIDX}_i[u, v] \leftarrow F^*(\mathbf{r}_{i,v}, u||\mathsf{st}_{i,u})$, for $u \in [N]$ and $v \in [m]$
8: $\mathsf{STkn}_i \leftarrow (\mathsf{STkn}_{i,1}, \mathsf{STkn}_{i,2}, \dots, \mathsf{STkn}_{i,m})$
9: $\mathsf{PTkn}_i \leftarrow (\mathsf{PTkn}_{i,1}, \mathsf{PTkn}_{i,2}, \dots, \mathsf{PTkn}_{i,m})$
10: return $(\kappa_{w_i}, \mathsf{EIDX}_i, \mathsf{st}_i, \mathsf{STkn}_i, \mathsf{PTkn}_i)$
```

Figure 3: Our MUSES setup.

4.2 Detailed MUSES Construction

We describe MUSES scheme. We first present necessary data structures followed by detailed operations.

4.2.1 Data Structures. In MUSES, there are two main data structures including the search index and auxiliary components.

Search Index. Similar to [73], each writer $i \in \mathcal{W}$ in our scheme has an independent search index representing keyword-document relationship in her document collection. We make use of BF to create an efficient search index for each writer $i \in [n_w]$. Suppose that 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 $\mathbf{w}_u \leftarrow \mathsf{BF.Gen}(\mathcal{V}_u) \in \{0,1\}^m$. The search index contains N BF vectors, which can be interpreted as a binary matrix of size $N \times m$ as $\mathsf{IDX}_i = [\mathbf{w}_1, \mathbf{w}_2, \dots, \mathbf{w}_N] \in \{0,1\}^{N\times m}$. By this representation, searching a keyword incurs checking its membership with BF by reading K columns in IDX_i, where K is the BF parameter. On the other hand, updating a document u incurs recreating a new BF representation of the updated keywords in u and writing it to the corresponding row $\mathsf{IDX}_i[u, *]$.

For confidentiality, IDX_i needs to be encrypted. For reader efficiency, the writer needs to encrypt her index in a way that the reader can delegate the decryption task securely to the servers during search. MUSES makes use of KH-PRF for such a decryption delegation, where the writers' indices are encrypted with almost KH-PRF function denoted as F^*.

First, the writer i interprets the search index as a matrix $\mathsf{IDX}_i = \begin{bmatrix}\mathbf{d}_1 & \mathbf{d}_2 & \dots & \mathbf{d}_m\end{bmatrix} \in \{0,1\}^{N\times m}$, where $\mathbf{d}_v \in \{0,1\}^N \ \forall v \in [m]$. 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. Our work emphasizes on proper handling "almost" attribute of KH-PRF to achieve user efficiency. Notice that since F^* is an almost KH-PRF, there exists a small error e as shown in §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 to obtain the original data. Moreover, since the server will also perform secure addition of K columns after KH-PRF evaluations for BF membership check, we need to reserve enough space for the aggregated error. Let $z = \lceil \log_2(e \cdot K) \rceil$ be the number of bits for the aggregated error when adding K columns encrypted by KH-PRF together. For each $u \in [N]$, the writer encrypts the element $\mathbf{d}_v[u]$ as

$$\hat{\mathbf{d}}_v[u] \leftarrow (\mathbf{d}_v[u] \lll z) + F^*(\mathbf{r}_v, u \mid\mid \mathsf{st}_{i,u}) \pmod p \qquad (2)$$

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

Auxiliary Information. In MUSES, each writer i's encrypted index EIDX_i is associated with three auxiliary components as follows.

- Private token PTkn_i: It contains information for the reader to search on EIDX_i, which are the KH-PRF keys $\mathbf{r}_v$ that the writer i uses to encrypt IDX_i as discussed above. Let $(\mathsf{pk}, \mathsf{sk}) \leftarrow \Pi.\mathsf{Gen}(1^\lambda)$ be the public-key and private-key pair of the reader. The private token is $\mathsf{PTkn}_i = (\mathsf{PTkn}_{i,1}, \dots, \mathsf{PTkn}_{i,m})$, where $\mathsf{PTkn}_{i,v} \leftarrow \Pi.\mathsf{Enc}(\mathsf{pk}, \mathbf{r}_v)$ for $v \in [m]$.
- Secret token STkn_i: It contains information for the writer i to update her index, which is also the KH-PRF keys $\mathbf{r}_v$, for $v \in [m]$, but encrypted with the writer's secret key. Let $\kappa_{w_i} \leftarrow \mathcal{E}.\mathsf{Gen}(1^\lambda)$ be the secret key of the writer i. The secret token is $\mathsf{STkn}_i = (\mathsf{STkn}_{i,1}, \dots, \mathsf{STkn}_{i,m})$, where $\mathsf{STkn}_{i,v} \leftarrow \mathcal{E}.\mathsf{Enc}(\kappa_{w_i}, v, \mathbf{r}_v)$ for $v \in [m]$. The secret token is stored at the servers to achieve stateless writers.
- Update state st_i: It contains information concatenated with document identifier as seed value for KH-PRF evaluation. Specifically, the update state is $\mathsf{st}_i = (\mathsf{st}_{i,1}, \dots, \mathsf{st}_{i,N})$, where $\mathsf{st}_{i,u}$ is the counter value (initialized with 0) for document u that is incremented after each update operation on that document happens.

4.2.2 Setup protocol. Figure 3 presents the setup algorithms for the reader and the writers in MUSES with the following highlights.

- Reader: The reader executes RSetup algorithm to generate a public and private key pair $(\mathsf{pk}, \mathsf{sk})$. The reader keeps sk private, and distributes pk to all writers to setup necessary components.
- Writer: Each writer i executes WSetup algorithm on reader's public key pk to generate four components including the encrypted search index EIDX_i, a secret token STkn_i, a private token PTkn_i, and the update state st_i as discussed above. Specifically, the writer first generates a secret key κ_{w_i} (line 1), and m KH-PRF keys $(\mathbf{r}_{i,1}, \dots, \mathbf{r}_{i,m})$ to encrypt m columns of the search index (line 3). These KH-PRF column keys are then encrypted under the writer's secret key κ_{w_i} as the secret token STkn_i, and also encrypted under the reader's public key pk as private token PTkn_i (line 5). The writer initializes counter values as 0 stored in st_i (line 6). Finally, the writer encrypts an empty search index cell-by-cell by evaluating the almost KH-PRF function F^* with KH-PRF column keys $\mathbf{r}_{i,v}$ and the seeds formed by the update counters and row indices (line 7). To this end, the writer sends the encrypted index (EIDX_i) and auxiliary components (st_i, STkn_i, PTkn_i) to L servers, while keeping κ_{w_i} private.

4.2.3 Keyword search protocol. We present the search protocol of MUSES in Figure 4. Specifically, to search for a keyword w on the index of a writer subset $\mathcal{W}'$, the reader first executes SearchToken algorithm to compute its BF representation as column indices $(v_1, \dots, v_K)$, then creates corresponding DPF keys $\{q_1^{(\ell)}, \dots, q_K^{(\ell)}\}_{\ell\in[L]}$ (lines 1–4), and sends them to servers $\mathcal{P}_1, \dots, \mathcal{P}_L$ as search token $\mathfrak{s}$, where server $\mathcal{P}_\ell$ receives the corresponding token $\mathfrak{s}_\ell = (\mathcal{W}', \{q_k^{(\ell)}\}_{k\in[K]})$. Next, upon receiving the search token, the

```
SearchToken($w, \mathcal{W}'$):
1: for $k = 1$ to $K$ do
2:     $v_k \leftarrow H_k(w)$, $(q_k^{(1)}, \dots, q_k^{(L)}) \leftarrow \mathsf{DPF.Gen}(1^\lambda, v_k, 1)$
3: $\mathfrak{s}_\ell \leftarrow (\mathcal{W}', \{q_k^{(\ell)}\}_{k \in [K]})$, for $\ell \in [L]$
4: return $\mathfrak{s} \leftarrow (\mathfrak{s}_1, \dots, \mathfrak{s}_L)$

Search($\mathfrak{s}, \mathsf{sk}, \{(i, \mathsf{st}_i, \mathsf{EIDX}_i, \mathsf{PTkn}_i)\}_{i \in [n_w]}$):
$\mathcal{P}_1, \dots, \mathcal{P}_L$: for each $i \in \mathcal{W}'$
$(\pi_{i,1}, \Delta_{i,1}; \pi_{i,2}, \Delta_{i,2}, \mathbf{a}_{i,2}; \dots; \mathbf{a}_{i,L}, \mathbf{b}_{i,L}) \leftarrow \mathsf{LSS.ShrTrns}(1^\lambda, N)$
Each server $\mathcal{P}_\ell$:
5: for each $i \in \mathcal{W}'$, $k = 1$ to $K$ do
6:     $\hat{\mathbf{d}}_{i,v_k}^{(\ell)} \leftarrow \sum_{v=1}^{m} \mathsf{DPF.Eval}(q_k^{(\ell)}, v) \times \mathsf{EIDX}_i[*, v]$
7:     $\mathsf{PTkn}_{i,v_k}^{(\ell)} \leftarrow \sum_{v=1}^{m} \mathsf{DPF.Eval}(q_k^{(\ell)}, v)) \times \mathsf{PTkn}_{i,v}$
$\mathcal{P}_\ell \rightarrow$ Reader : $\mathsf{PTkn}_{i,v_k}^{(\ell)}$, for $i \in \mathcal{W}'$ and $k \in [K]$
Reader:
8: for each $i \in \mathcal{W}'$, $k = 1$ to $K$ do
9:     $\mathsf{PTkn}_{i,v_k} \leftarrow \sum_{\ell=1}^{L} \mathsf{PTkn}_{i,v_k}^{(\ell)}$
10:    $\mathbf{r}_{i,v_k} \leftarrow \Pi.\mathsf{Dec}(\mathsf{sk}, \mathsf{PTkn}_{i,v_k})$
11:    $(\mathbf{r}_{i,v_k}^{(1)}, \dots, \mathbf{r}_{i,v_k}^{(L)}) \leftarrow \mathsf{KH\text{-}PRF.Share}(\mathbf{r}_{i,v_k})$
12: Reader $\rightarrow \mathcal{P}_\ell$ : $\mathbf{r}_{i,v_k}^{(\ell)}$, for $\ell \in [L]$, $i \in \mathcal{W}'$, and $k \in [K]$
Each server $\mathcal{P}_\ell$:
13: for each $i \in \mathcal{W}'$ do
14:     $\tilde{\mathbf{d}}_i^{(\ell)} \leftarrow \{0\}^N$
15:     for $k = 1$ to $K$ do
16:         for $u = 1$ to $N$ do
17:             $\tilde{\mathbf{d}}_{i,v_k}^{(\ell)}[u] \leftarrow \hat{\mathbf{d}}_{i,v_k}^{(\ell)}[u] - F^*(\mathbf{r}_{i,v_k}^{(\ell)}, u \,||\, \mathsf{st}_{i,u}) \pmod p$
18:         $\tilde{\mathbf{d}}_i^{(\ell)} \leftarrow \tilde{\mathbf{d}}_i^{(\ell)} + \tilde{\mathbf{d}}_{i,v_k}^{(\ell)} \pmod p$
19:     $(\pi_{i,1}; \pi_{i,2}; \dots; \pi_{L-1}; \tilde{\mathbf{d}}'_i) \leftarrow \mathsf{LSS.Shffl}(\tilde{\mathbf{d}}_i^{(1)}, \pi_{i,1}, \Delta_{i,1}; \tilde{\mathbf{d}}_i^{(2)}, \pi_{i,2}, \Delta_{i,2}, \mathbf{a}_{i,2};$
    $\dots; \tilde{\mathbf{d}}_i^{(L)}, \mathbf{a}_{i,L}, \mathbf{b}_{i,L})$
Server $\mathcal{P}_L$: $O_i \leftarrow \{\}$, for $i \in \mathcal{W}'$
20: for each $i \in \mathcal{W}'$, $u = 1$ to $N$ do
21:     if $\tilde{\mathbf{d}}'_i[u] \ggg z = K$ then
22:         $O_i \leftarrow O_i \cup \{u\}$
$\mathcal{P}_\ell \rightarrow$ Reader: $\pi_{i,\ell}$, for $\ell \in [L-1]$ and $i \in \mathcal{W}'$
$\mathcal{P}_L \rightarrow$ Reader: $O_i$, for $i \in \mathcal{W}'$
Reader: $O \leftarrow \{\}$
23: for each $i \in \mathcal{W}'$, $u \in O_i$ do
24:     $u' = \pi_{i,1}^{-1}(\pi_{i,2}^{-1}(\dots \pi_{i,L-1}^{-1}(u) \dots))$, $O \leftarrow O \cup \{(i, u')\}$
25: return $O$
```

Figure 4: Our MUSES search.

servers execute Search algorithm as follows. Each server $\mathcal{P}_\ell$ performs DPF evaluation on its received DPF keys with search indices EIDX_i to privately retrieve the additive shares of requested columns $\mathsf{EIDX}_i[*, v_k]$ as $(\hat{\mathbf{d}}_{i,v_1}^{(\ell)}, \dots, \hat{\mathbf{d}}_{i,v_K}^{(\ell)})$, and with private tokens PTkn_i to retrieve the additive shares of requested private tokens $(\mathsf{PTkn}_{i,v_1}, \dots, \mathsf{PTkn}_{i,v_K})$ as $(\mathsf{PTkn}_{i,v_1}^{(\ell)}, \dots, \mathsf{PTkn}_{i,v_K}^{(\ell)})$, for $k \in [K]$, and $i \in \mathcal{W}'$ (lines 5–7).

The servers return the shares of private tokens corresponding to the queried columns to the reader, who reconstructs and decrypts them to retrieve the secret keys $\mathbf{r}_{i,v_k}$, for $i \in \mathcal{W}'$ and $k \in [K]$ (lines 9–10), By using KH-PRF, the servers can be allowed to evaluate to obtain the final search result directly without leaking the search index's encryption keys and search patterns. In particular, the reader creates additive secret-shares of each $\mathbf{r}_{i,v_k}$ as $(\mathbf{r}_{i,v_k}^{(1)}, \dots, \mathbf{r}_{i,v_k}^{(L)}) \leftarrow$ $\mathsf{KH\text{-}PRF.Share}(\mathbf{r}_{i,v_k})$ and sends $\mathbf{r}_{i,v_k}^{(\ell)}$ to $\mathcal{P}_\ell$, for each $\ell \in [L]$, $k \in [K]$ and $i \in \mathcal{W}'$ (lines 11–12). The servers use these secret-shared keys along with values contained in state st_i for KH-PRF evaluation to obtain secret shares of the queried columns (lines 13–18), where each server $\mathcal{P}_\ell$ computes the following for each $u \in [N]$, $k \in [K]$ and $i \in \mathcal{W}'$:

$$\tilde{\mathbf{d}}_{i,v_k}^{(\ell)}[u] \leftarrow \hat{\mathbf{d}}_{i,v_k}^{(\ell)}[u] - F^*(\mathbf{r}_{i,v_k}^{(\ell)}, u \,||\, \mathsf{st}_{i,u}) \pmod p \tag{3}$$

When using BF, the servers have to merge retrieved columns to obtain the final search output. Otherwise, unwrapping each column (even when being shuffled) during search process may reveal the patterns of retrieved columns, which leads to search pattern leakage. Therefore, each server $\mathcal{P}_\ell$ computes its share of the aggregated column as $\tilde{\mathbf{d}}_i^{(\ell)} \leftarrow \sum_{k=1}^{K} \mathbf{d}_{i,v_k}^{(\ell)} \in \mathbb{Z}_p^N$. After this step, the servers can broadcast $\tilde{\mathbf{d}}_i^{(\ell)}$ to reconstruct $\tilde{\mathbf{d}}_i \leftarrow \sum_{\ell=1}^{L} \tilde{\mathbf{d}}_i^{(\ell)} \in \mathbb{Z}_p^N$. However, it also permits the servers to obtain document identifiers matching the search query, which is known as result pattern leakage. We show how to further hide result pattern leakage on the search index in our scheme as follows. We utilize L-party secret-shared shuffle (LSS) (Figure 2) to securely permute the document identifiers in the search output. Our high-level idea is that after shuffling, each server $\ell \in 1, \dots, L-1$ holds an independent permutation, and the other server L holds shuffled opened (unmasked) output which is permuted by a permutation composition of servers $1, \dots, L-1$. As a result, only the reader who receives all permutations and the shuffled opened output can recover the actual data. So after obtaining the share of the aggregated column $\tilde{\mathbf{d}}_i^{(\ell)}$ above, L servers $\mathcal{P}_1, \dots, \mathcal{P}_L$ interact to perform oblivious shuffle (line 19) such that at the end of this process, each server $\mathcal{P}_\ell$, for $\ell \in [L-1]$, holds an independent permutation $\pi_{i,\ell}$, and server $\mathcal{P}_L$ holds $\tilde{\mathbf{d}}'_i$, which is shuffled opened (unmasked) $\tilde{\mathbf{d}}_i$. From (2) and (3), we can see that $\tilde{\mathbf{d}}'_i[u]$ contains an aggregated error as

$$\tilde{\mathbf{d}}'_i[u] \leftarrow \sum_{i=1}^{K} \hat{\mathbf{d}}_{i,v_k}[\pi_i(u)] + \sum_{i=1}^{K} e_{i,v_k,\pi_i(u)} \pmod p \tag{4}$$

where $\pi_i = \pi_{i,L-1}(\pi_{i,L-2}(\dots(\pi_{i,1})\dots))$ is the permutation composition constituted by $L-1$ permutations $\pi_{i,1}, \dots, \pi_{i,L-1}$, and $e_{i,v_k,\pi_i(u)}$ is the error during the KH-PRF evaluation of each $\hat{\mathbf{d}}_{i,v_k}[u]$ in (2). Therefore, the server L can perform $\tilde{\mathbf{d}}'_i[u] \leftarrow \tilde{\mathbf{d}}'_i[u] \ggg z$ for each $u \in [N]$ to remove z bits of the small aggregated errors in (4), thereby obtaining the final search result as $O_i = \{u \in [N] : \tilde{\mathbf{d}}'_i[u] = K\}$ (lines 20–22).

Finally, the server $\mathcal{P}_\ell$, for $\ell = 1, \dots, L-1$ sends $\pi_{i,\ell}$[2], while $\mathcal{P}_L$ sends O_i to the reader, for $i \in \mathcal{W}'$, and the reader can recover the actual search output by reversing each permutation sequentially (lines 23–25).

4.2.4 Access revocation protocol. 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 5 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 EIDX_i, the writer i first retrieves and decrypts STkn_i to obtain current column keys $\mathbf{r}_{i,v}$ (line 2). Also, the writer generates new secret column keys $\mathbf{r}'_{i,v}$ and encrypts them as the updated secret token $\mathsf{STkn}'_{i,v}$ (line 3). The writer creates secret-shares of these keys as $(\mathbf{r}_{i,v}^{(1)}, \dots, \mathbf{r}_{i,v}^{(L)})$, $(\mathbf{r}'^{(1)}_{i,v}, \dots, \mathbf{r}'^{(L)}_{i,v})$ (lines 4–5), and sends $\{\mathsf{STkn}'_i, \mathbf{r}_{i,v}^{(\ell)}, \mathbf{r}'^{(\ell)}_{i,v}\}$ to server $\mathcal{P}_\ell$, for $v \in [m]$ and $\ell \in [L]$.

[2]This can be done efficiently by sending a PRP seed.

```
$\mathsf{RevokePerm}(i, \kappa_{w_i}, \{\mathsf{EIDX}_i, \mathsf{st}_i, \mathsf{STkn}_i)\}_{n\in[n_w]}$:
Writer $i \longrightarrow$ Server $\mathcal{P}_0$: $i$
Server $\mathcal{P}_0 \longrightarrow$ Writer $i$: secret tokens $\mathsf{STkn}_i$
Writer $i$:
 1: for $v = 1$ to $m$ do
 2:     $\mathbf{r}_{i,v} \leftarrow \mathcal{E}.\mathsf{Dec}(\kappa_{w_i}, j, \mathsf{STkn}_{i,j})$, $\mathbf{r}'_{i,v} \leftarrow \mathsf{KH\text{-}PRF.Gen}(1^\lambda)$
 3:     $\mathsf{STkn}'_{i,v} \leftarrow \mathcal{E}.\mathsf{Enc}(\kappa_{w_i}, v, \mathbf{r}'_{i,v})$
 4:     $(\mathbf{r}^{(1)}_{i,v}, \ldots, \mathbf{r}^{(L)}_{i,v}) \leftarrow \mathsf{KH\text{-}PRF.Share}(\mathbf{r}_{i,v})$
 5:     $(\mathbf{r}'^{(1)}_{i,v}, \ldots, \mathbf{r}'^{(L)}_{i,v}) \leftarrow \mathsf{KH\text{-}PRF.Share}(\mathbf{r}'_{i,v})$
 6: $\mathsf{STkn}'_i \leftarrow (\mathsf{STkn}'_{i,1}, \ldots, \mathsf{STkn}'_{i,m})$
Writer $i \longrightarrow$ Server $\mathcal{P}_\ell$: $\{\mathsf{STkn}'_i, \mathbf{r}^{(\ell)}_{i,v}, \mathbf{r}'^{(\ell)}_{i,v}\}$, for $v \in [m]$ and $\ell \in [L]$
Server $\mathcal{P}_\ell$:
 7: $\mathsf{STkn}_i \leftarrow \mathsf{STkn}'_i$
 8: for $u = 1$ to $N$ do
 9:     for $v = 1$ to $m$ do
10:         $x_{u,v} \xleftarrow{\$} \mathbb{Z}_p$, $\mathbf{M}^{(\ell)}[u,v] \leftarrow x_{u,v} \lll z$
11:         $\mathbf{T}^{(\ell)}_1[u,v] \leftarrow \mathbf{M}^{(\ell)}[u,v] - F^*(\mathbf{r}^{(\ell)}_{i,v}, u||\mathsf{st}_{i,u}) + \mathsf{max}_e \pmod p$
12:         $\mathbf{T}^{(\ell)}_2[u,v] \leftarrow -\mathbf{M}^{(\ell)}[u,v] + F^*(\mathbf{r}'^{(\ell)}_{i,v}, u||\mathsf{st}_{i,u}) \pmod p$
$\forall\, j \in [L]$ and $j \neq i$: Server $\mathcal{P}_i \longrightarrow$ Server $\mathcal{P}_j$: $\mathbf{T}^{(i)}_1, \mathbf{T}^{(i)}_2$, for $i \in [L]$
Server $\mathcal{P}_\ell$:
13: for $u = 1$ to $N$ do
14:     for $v = 1$ to $m$ do
15:         $\mathbf{T}'[u,v] \leftarrow \mathsf{EIDX}_i[u,v] + \sum_{\ell=1}^{L} \mathbf{T}^{(\ell)}_1[u,v]$
16:         $\hat{\mathbf{T}}[u,v] \leftarrow ((\mathbf{T}'[u,v] \ggg z) \lll z)$
17:         $\mathsf{EIDX}'_i[u,v] \leftarrow \hat{\mathbf{T}}[u,v] + \sum_{\ell=1}^{L} \mathbf{T}^{(\ell)}_2[u,v] + \mathsf{max}_e \pmod p$
18: return $(\mathsf{EIDX}'_i, \mathsf{STkn}'_i)$
```

Figure 5: Our MUSES permission revocation.

Each server $\mathcal{P}_\ell$ replaces STkn_i by STkn'_i (line 7), computes $\mathbf{T}^{(\ell)}_1$ and $\mathbf{T}^{(\ell)}_2$, where $\mathbf{T}^{(\ell)}_1$ is the masked component to remove the shared encryption computed by the secret-shared key $\mathbf{r}^{(\ell)}_{i,v}$, and $\mathbf{T}^{(\ell)}_2$ is the component to unmask the value $\mathbf{M}^{(\ell)}$ added by $\mathbf{T}^{(\ell)}_1$ and add the shared encryption computed by the new secret-shared key $\mathbf{r}'^{(\ell)}_{i,v}$ (lines 8–12). The server $\mathcal{P}_\ell$ then distributes $\mathbf{T}^{(\ell)}_1$ and $\mathbf{T}^{(\ell)}_2$ to the other servers. Next, each server $\mathcal{P}_\ell$ obtains the masked value with error denoted as $\mathbf{T}'$ (lines 13–15), where $\mathbf{T}'[u,v] = (\mathsf{IDX}[u,v] \lll z) + \sum_{\ell=1}^{L} \mathbf{M}^{(\ell)}[u,v] + e_{u,v}$. The random mask $\mathbf{M}^{(\ell)}$ generated by each server $\mathcal{P}_\ell$ is to hide the plaintext data $(\mathsf{IDX}[u,v] \lll z)$ when the servers remove the current encryption by adding EIDX with $\sum_{\ell=1}^{L} \mathbf{T}^{(\ell)}_1$ to obtain $\mathbf{T}'$. By clearing z LSBs of each value $\mathbf{T}'[u,v]$ to 0, the error value part $e_{u,v}$ can be removed, and the servers now hold the masked plaintext value $\hat{\mathbf{T}}[u,v]$ (line 16). Finally, to retrieve the final EIDX'_i encrypted by the new secret-key, each server $\mathcal{P}_\ell$ computes $\mathsf{EIDX}'_i[u,v]$ based on $\hat{\mathbf{T}}[u,v]$ and $\sum_{\ell=1}^{L} \mathbf{T}^{(\ell)}_2[u]$. The value $\mathsf{max}_e = L$ is the max error when using LWR-based KH-PRF (line 17). It is necessary for decryption in keyword search later. At the end of this protocol, all servers hold the same updated EIDX'_i, which is the search index encrypted with the new column keys, as well as updated secret tokens STkn'_i.

4.2.5 Document update protocol. Given an updated document with identifier u and a list of its keywords $\mathbf{w}$, the writer i retrieves the state value $\mathsf{st}_{i,u}$ corresponding to the document u, and secret tokens $\mathsf{STkn}_i = (\mathsf{STkn}_{i,1}, \ldots, \mathsf{STkn}_{i,m})$ (Figure 6). Then, the writer uses her κ_{w_i} to decrypt STkn_i (lines 1–2) and obtain the secret keys of data columns as $\mathbf{r}_{i,v} \leftarrow \mathcal{E}.\mathsf{Dec}(\kappa_{w_i}, v, \mathsf{STkn}_{i,v})$, for $v \in [m]$. Next, the writer computes the new BF representation $\mathbf{u} \in \{0,1\}^m$ of the updated document with input keywords $\mathbf{w}$ (lines 3–5). Finally,

```
$\mathsf{UpdateToken}(\mathbf{w}, u, i, k_{w_i}, \{\mathsf{st}_i, \mathsf{STkn}_i\}_{i\in[n_w]})$:
Writer $i \longrightarrow$ Server $\mathcal{P}_0$: $(i, u)$
Server $\mathcal{P}_0 \longrightarrow$ Writer $i$: state $\mathsf{st}_{i,u}$, secret tokens $\mathsf{STkn}_i$
Writer $i$:
 1: for $v = 1$ to $m$ do
 2:     $\mathbf{r}_v \leftarrow \mathcal{E}.\mathsf{Dec}(\kappa_{w_i}, v, \mathsf{STkn}_{i,v})$
 3: $\mathbf{u} \leftarrow \{0\}^m$, $\mathbf{u}' \leftarrow \{0\}^m$
 4: for each $w_j \in \mathbf{w}$, $k = 1$ to $K$ do
 5:     $\mathsf{cid}_{j,k} \leftarrow H_k(w_j)$, $\mathbf{u}[\mathsf{cid}_{j,k}] \leftarrow 1$
 6: for $v = 1$ to $m$ do
 7:     $\mathbf{u}'[v] \leftarrow (\mathbf{u}[v] \lll z) + F^*(\mathbf{r}_v, u||(\mathsf{st}_{i,u}+1)) \pmod p$
 8: return $\mathfrak{u} \leftarrow (i, u, \mathbf{u}')$

$\mathsf{Update}(\mathfrak{u}, \{\mathsf{EIDX}_i, \mathsf{st}_i\}_{i\in[n_w]})$:
Each server $\mathcal{P}_\ell$: parse $\mathfrak{u} = (i, u, \mathbf{u}')$
 9: $\mathsf{EIDX}_i[u,*] \leftarrow \mathbf{u}'$, $\mathsf{st}_{i,u} \leftarrow \mathsf{st}_{i,u} + 1$
10: return $(\mathsf{EIDX}_i, \mathsf{st}_i)$
```

Figure 6: Our MUSES document update.

the writer encrypts the updated row $\mathbf{u}$ with column keys and the incremented counter value for KH-PRF evaluation as $\mathbf{u}'[v] \leftarrow \mathbf{u}[v] + F^*(\mathbf{r}_v, u \,||\, (\mathsf{st}_{i,u}+1))$ for each $v \in [m]$ (lines 6–7). Finally, the writer sends the update token $\mathfrak{u} = (i, u, \mathbf{u}')$ to the servers to update the search index of the writer accordingly as $\mathsf{EIDX}_i[u,*] \leftarrow \mathbf{u}'$ and $\mathsf{st}_{i,u} \leftarrow \mathsf{st}_{i,u} + 1$ (lines 9–10).

4.3 Analysis

4.3.1 Complexity. We analyze the online[3] asymptotic cost of MUSES. We consider the number of servers L as a small constant and omit it in the following analysis. Let m, K be the BF parameters and N be the number of documents. To search for a keyword w in a writer's database, the reader creates a query of size $O(K \cdot \lambda \cdot \tau) \sim O(\lambda\tau)$, where $\tau = O(\log m)$ for $L = 2$, and $\tau = O(\sqrt{m})$ for $L \geq 3$ (as K is a constant BF parameter), and with the corresponding computation complexity $O(K \cdot \tau) \sim O(\tau)$. Also, for each writer in $\mathcal{W}'$, the reader sends shares of KH-PRF keys to L servers, which costs $O(L \cdot K \cdot n \cdot \log q) \sim O(\lambda)$ in total (as $n, q \sim O(\lambda)$ are the LWR parameters of KH-PRF). Let n_s be the bound on the size of the search output. To obtain the search result, the reader receives seed values from $\mathcal{P}_1, \ldots \mathcal{P}_{L-1}$, 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 $O((L-1)\lambda + n_s) \sim O(\lambda + n_s)$ communication and $O((L-1).N) \sim O(N)$ computation cost. Overall, to search for a keyword on a writer subset $\mathcal{W}'$, the computation complexity at the reader is $O(\tau + |\mathcal{W}'|N)$, and the bandwidth cost between the reader and the servers is $O(\lambda\tau + |\mathcal{W}'|(\lambda + n_s))$.

To update a document, the writer retrieves and decrypts secret tokens to obtain the secret keys, which is $O(m \cdot n \cdot \log q) \sim O(m \cdot \lambda)$ in bandwidth and $O(m)$ in computation cost. The writer retrieves the counter value of size $O(\lambda)$ of the updated document, creates a new BF representation of size $O(m)$, and re-encrypts it with the secret keys and the incremented counter value. Therefore, the total writer's bandwidth cost and computation cost per document update is $O(m \cdot \lambda)$ and $O(m)$, respectively.

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

[3]We do not consider the precomputation cost in this analysis.

$O(|\mathcal{W}'| \cdot N \cdot m)$. For document update, the servers replace one row in EIDX and increment a counter value, and thus do not incur computation.

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

For the storage, the reader and each writer stores a private/secret key of size $O(\lambda)$. For each writer, the servers store an index of size $O(\log p \cdot N \cdot m) \sim O(N \cdot m)$, a state st of size $O(\lambda \cdot N)$, private tokens PTkn and secret tokens STkn, both are of the same size $O(m \cdot n \cdot \log q) \sim O(m \cdot \lambda)$. 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.

4.3.2 *Security Analysis.*

Theorem 1. *Assuming that the adversary can statically corrupt at most $L-1$ out of the L servers and some writers, MUSES is $\mathcal{L}_{\mathcal{H}}$-adaptively-secure by Definition 8 with forward privacy by Definition 9, and backward privacy by Definition 10, where* $\mathcal{L}_{\mathcal{H}}^{\text{Setup}}(1^{\lambda}) = \{i, N, m\}_{i \in [n_w]}$, $\mathcal{L}_{\mathcal{H}}^{\text{CorruptWriter}}(i) = \{\text{UpdateBy}(i)\}$, $\mathcal{L}_{\mathcal{H}}^{\text{Search}}(w, \mathcal{W}') = \{\mathcal{W}', \text{sv}(w)\}$, $\mathcal{L}_{\mathcal{H}}^{\text{Update}}(i, u, \mathbf{w}) = \{i, \text{up}(u)\}$, *and* $\mathcal{L}_{\mathcal{H}}^{\text{Revoke}}(i) = \{i\}$, *where $\mathcal{W}'$ is a writer subset.*

We present the proof in Appendix A.

5 EXPERIMENTAL EVALUATION

Implementation. We fully implemented all our proposed techniques 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` [76] for public-key encryption in our scheme, and `EMP-Toolkit` [74] for IKNP OT protocol. We implemented KH-PRF and OS3 from scratch. We used `libzeromq` [2] to implement network communication between servers and client. Our implementation is available and ready for public release (see the attached artifact).

Hardware and network. We used two EC2 `r5n.4xlarge` instances each equipped with 8-core Intel Xeon Platinum 8375C CPU @ 2.90 GHz and 128 GB memory as servers. For the reader, 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 bandwidth between the servers and the client is 20 Mbps with 10ms RTT.

Dataset. We used the Enron email dataset [3] which includes about 500K emails of 150 employees. We extract unique keywords using the standard tokenization method as described in [25]. 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 compare MUSES with the state-of-the-art schemes including FP-HSE [73] and DORY [25]. We select their parameters as follows.

- **MUSES**: For KH-PRF, we select $q = 2^{13}$, $p = 2^{10}$, and $n = 256$, $l = 2$ as suggested in [29] for secure LWR with 118-bit security, where each KH-PRF key is of 1 KB. We use `SHA-256` for the hash function. We used 128-bit keys for IND-CPA encryption and PRF/PRG seeds. Each folder in the dataset is considered as a writer. We create a reader that can perform search over all databases to run 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 choose BF parameters such that $N \times$FP rate < 1. For $K = 7$, we choose $m = 2000$ to achieve FP rate $\approx 3e^{-5}$. To evaluate the performance of permission revocation with various database sizes, we run experiments with the number of documents N from 2^{10} to $2^{19} (\sim 500K)$. For the number of documents from 2^{10} to 2^{19}, the corresponding BF parameter m is from 1120 to 3120 (with $K = 7$) to satisfy the condition of low FP rate above.
- **FP-HSE** [73]: We selected the originally suggested parameters with 96-bit security level, where PRFs and keyed hash functions are instantiated with `HMAC-SHA-256`, and `MNT224` curve for pairings. 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). Each folder in the dataset is considered as a separate writer.
- **DORY** [25]. We run experiments with DORY in the semi-honest setting similar to our MUSES and FP-HSE. We configure BF parameters of DORY similar to our scheme because DORY uses DPF-based PIR scheme for oblivious search, and 128-bit keys for IND-CPA encryption, and PRG seeds.

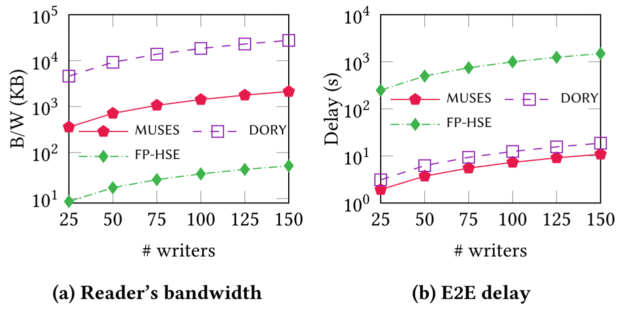


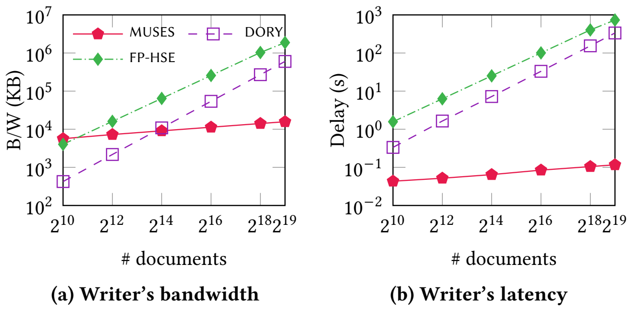
Figure 7: Keyword search performance (log scale on y-axis).

5.1 Overall Results

5.1.1 *Keyword search.*

Reader's Bandwidth. Figure 7a shows the search bandwidth between the reader and the servers of our MUSES, DORY and FP-HSE. The network overhead in MUSES increases from 0.3 MB to 2.1 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 12.8×–13.0× 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 security vulnerabilities and leaks more information than the others.

Keyword Search. Figure 7b illustrates the end-to-end delay in keyword search of our scheme with DORY and FP-HSE for different numbers of writers. The latency of all schemes grows almost linearly to the number of writers. MUSES is about 129.1×–137.2×



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

Figure 8: Permission revocation performance (log scale on y-axis).

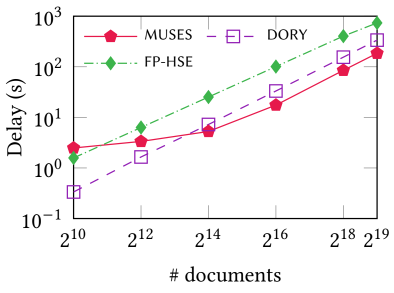


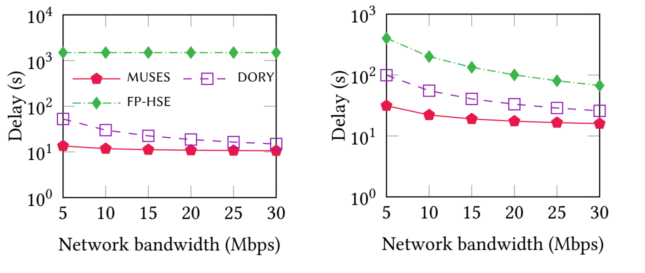
Figure 9: E2E permission revocation delay (log scale on y-axis).

faster than FP-HSE, and 1.6×-1.7× faster than DORY. With 25 writers, MUSES takes approximately 1.8s to accomplish a search, and increases to about 10.9s 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 delay include reader processing, communication latency, and server processing. Our scheme incurs a low reader processing cost in search, in which it only takes 12.7*ms*–64.3*ms*, attributing 0.5%–0.7% to the total delay. By contrast, the server cost in MUSES to search a keyword takes about 1.8*s*-9.9*s*, corresponding to 91.3%–91.7% of the total delay. The communication overhead during search in MUSES is about 0.1*s*–0.9*s*, which attributes about 7.6%–8.1% to the total delay.

5.1.2 Permission revocation. We evaluate the performance of MUSES when a writer wants to update secret column keys to revoke access permission of the reader on her database, and compare it with other schemes. For DORY and FP-HSE, as these schemes do not offer access 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 with the reader.

Writer's Bandwidth. Figure 8a 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 current and updated secret tokens while DORY and FP-HSE requires downloading and uploading the whole search index. MUSES incurs



(a) Keyword search (b) Permission revocation

Figure 10: E2E delay with varying bandwidths (log scale on y-axis).

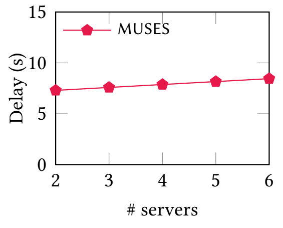
5.5 MB-15.4 MB communication overhead, while DORY produces 0.4 MB-587.0 MB, which is 1.2 × -38.2× larger than MUSES starting from 2^{14} documents. The network overhead of FP-HSE is about 3.9 MB−1833.4 MB, which is 2.2×−119.3× larger than our MUSES. The high overhead in FP-HSE is due to transmitting the search index back and forth similar to DORY.

Writer's Latency. Figure 8b 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 43.3ms–116.6ms for the database size increasing from 1K to 500K documents, where most overhead is for transferring secret tokens and the secret shares of 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 7.7×–2876.8× 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 36.3×−6296.7× longer than our MUSES.

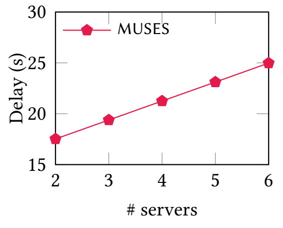
Permission revocation. Figure 9 illustrates the end-to-end delay to finish re-encryption of the search index of FP-HSE, DORY, and MUSES. It is noticeable that when increasing the number of documents, the end-to-end delay of all schemes grows linearly but by varying degrees. MUSES takes about 2.5*s* (resp. 184.8*s*) 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 1.4×−1.9× 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 1.9×−5.8× larger than MUSES.

5.1.3 Scalability.

Varying network bandwidths. Figure 10 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 10a) is measured in the case of 150 writers and the permission revocation delay (Figure 10b) is measure in the typical case of $2^{16} \approx 64$ K documents. As presented above, search operations of FP-HSE incur the lowest communication overhead, thus its performance is barely affected by varying network bandwidths, where it takes around 1.5×10^3s to finish a search operation. MUSES slightly increases when rising the network bandwidth, where it takes 13.5s with 5 Mbps, and decreases to 10.6s 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



(a) Keyword search



(b) Permission revocation

Figure 11: E2E delay with varying numbers of servers.

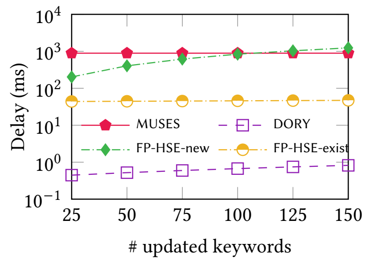


Figure 12: E2E document update delay (log scale on y-axis)

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 31.4s–15.9s to finish a permission revocation, where most communication overhead is for transmitting encrypted KH-PRF keys and their secret-shares to the servers.

Varying numbers of servers. To achieve higher security level, more servers can be added to the system. Figure 11 illustrates the end-to-end delay of keyword search and permission revocation with different numbers of servers increasing from 2 to 6. In MUSES, adding more servers does not significantly increase the online computation work of each server. Instead, it just 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, as well as secret-shares of KH-PRF keys when there are more servers joining the system. In particular, with 2 to 6 servers, the total delay to search for a keyword over the databases of 100 writers are 7.3s, 7.6s, 7.9s, 8.2s and 8.4s, respectively. Similarly, in permission revocation, the additional computation overhead on each server, when adding more counterparts, is insignificant as the servers have to broadcast their masked results to each other. Also, the writer has to create more secret-shares of KH-PRF keys, and upload secret tokens to more servers, which slightly increases communication overhead on the writer. Specifically, for the typical case of 2^{16} documents, the end-to-end latencies to finish a permission revocation corresponding to the number of servers growing from 2 to 6 are 17.5s, 19.4s, 21.2s, 23.1s and 24.9s, respectively.

5.1.4 Document update. Figure 12 presents the update delay of MUSES and FP-HSE. For each new keyword, FP-HSE-new needs two pairings to generate a new 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 0.9s to finish a document update, while FP-HSE-new takes 202.6ms-1.2s for the cases increasing from 25 to 150 keywords, and DORY takes about 0.5ms–0.8ms. Most of the overhead in MUSES is due to downloading secret tokens and decrypting them to obtain secret column keys of the search index. The update latency of FP-HSE-exist slightly grows from 43.9ms to 47.1ms as it does not incur pairing operations.

5.1.5 Storage overhead. In MUSES, each writer stores a 128-bit secret key and the reader stores a 256-bit private key. As MUSES utilizes LWR-based KH-PRF in [9], each bit in $\mathbb{Z}_2$ is encrypted to ciphertext in $\mathbb{Z}_p$ with $p = 2^{10}$ by 10 bits. As a result, for each writer with an EIDX including 32,768 documents and BF size of 2000, the size of EIDX is ≈ 78.1 MB. Each column of EIDX is encrypted by a separate key, in which each key is of 1 KB. In total, the database size of each writer including EIDX, PTkn, STkn, and st is ≈ 82.5M, which goes up to ≈ 12.4 GB for 150 writers.

6 RELATED WORK

SSE. Song et al. [65] were the first to propose and formalize searchable encryption, followed by a line of SSE schemes proposed that offer secure search over encrypted data plus dynamic update via an encrypted index [10, 11, 16, 17, 27, 37, 39, 45, 46, 50, 53, 67–69]. However, these schemes might be susceptible to many types of leakage-abuse attacks [4, 14, 44, 48, 52, 54, 56, 62, 63] and mainly support single-user setting. The multi-client SE exploited in [49] uses KH-PRF as a mechanism to share a secret key between the trusted and "helping" users to protect the confidentiality of search tokens, as well as to remove the interaction with the data owner in search operations. However, the proposed system might be impractical as each search operation requires the presence of certain "helping" users. In addition, the "almost" property of current KH-PRF schemes has been ignored in that design, which might require proper and subtle treatment in many circumstances.

PEKS. PEKS schemes such as [6, 8, 30, 77] can support multi-writer setting, but they do not adapt to forward privacy, which might lead to injection attacks [79]. Although hybrid-based architecture (*e.g.*, FP-HSE [73]) can ensure forward privacy, it requires each writer to be stateful and present to rebuild encrypted tokens periodically. In addition, most PEKS systems are vulnerable to KGA. PEKS in 2-server setting [51] can protect the privacy of the given trapdoor to prevent KGA, but it still incurs a large number of pairing operations in search (which runs linear to the number of keyword-document pairs). Single-key SSEs [25, 32] require that all users are trusted or assume a secure deploy environment with authenticated mechanisms. Multi-key SSE [49, 75] can provide decentralization between users but does not prevent pattern leakages.

Oblivious platforms. Some oblivious storage platforms employ ORAM and/or PIR primitives to hide search access patterns during data operations (e.g., data sharing/access [19–21, 25, 57, 58], search [28, 32, 36, 42, 61]. However, these schemes incur a large communication overhead, which costs $O(N)$ with N is the number of documents in the database. Differential Privacy-based technique [64] can obfuscate search access patterns, but it incurs high computation and latency.

Hardware-assisted SE. Trusted hardware (*e.g.*, Intel SGX [22]) has been used to build practical oblivious platforms with diverse functionalities [24, 31, 34, 35, 40, 41, 60, 70]. These platforms require a strong security assumption on the hardware (*e.g.*, isolation, tamper-free, side-channel resistance).

ACKNOWLEDGMENT

Thang Hoang is supported by an unrestricted gift from Robert Bosch, 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 is 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] IHOP: Improved statistical query recovery against searchable symmetric encryption through quadratic optimization. In *31st USENIX Security Symposium (USENIX Security 22)*, Boston, MA, August 2022. USENIX Association.

[5] 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.

[6] 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.

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

[8] 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.

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

[10] Raphael Bost. $\sum o\varphi o\varsigma$: 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.

[11] 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.

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

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

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

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

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

[23] 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.

[24] 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.

[25] 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.

[26] 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.

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

[28] 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.

[29] 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.

[30] 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.

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

[32] 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.

[33] 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.

[34] 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.

[35] 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.

[36] 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.

[37] 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.

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

[39] 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.

[40] 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.

[41] 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.

[42] 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.
[43] 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.
[44] Mohammad Saiful Islam, Mehmet Kuzu, and Murat Kantarcioglu. Access pattern disclosure on searchable encryption: Ramification, attack and mitigation. In *NDSS*, 2012.
[45] 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.
[46] 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.
[47] 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.
[48] 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.
[49] 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.
[50] Florian Kerschbaum and Anselme Tueno. An efficiently searchable encrypted data structure for range queries. In *European Symposium on Research in Computer Security*, 2017.
[51] 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.
[52] 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.
[53] 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.
[54] 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.
[55] 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.
[56] 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.
[57] 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.
[58] Travis Mayberry, Erik-Oliver Blass, and Agnes Hui Chan. Efficient private file retrieval by combining oram and pir. *NDSS 2013*, 2013.
[59] 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.
[60] 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.
[61] 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.
[62] 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.
[63] 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.
[64] 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.
[65] 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.
[66] 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.
[67] 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.
[68] 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.
[69] 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.
[70] Afonso Tinoco, Sixiang Gao, and Elaine Shi. Enigmap: Signal should use oblivious algorithms for private contact discovery. *Cryptology ePrint Archive*, 2022.
[71] Alin Tomescu and Srinivas Devadas. Catena: Efficient non-equivocation via bitcoin. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 393–409, 2017.
[72] Jiafan Wang and Sherman Chow. Forward and backward-secure range-searchable symmetric encryption. *Proceedings on Privacy Enhancing Technologies*, 2022:28–48, 01 2022.
[73] 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.
[74] Xiao Wang, Alex J. Malozemoff, and Jonathan Katz. EMP-toolkit: Efficient Multi-Party computation toolkit. https://github.com/emp-toolkit, 2016.
[75] 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.
[76] Pieter Wuille. libsecp256k1. https://github.com/bitcoin-core/secp256k1.
[77] 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.
[78] 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.
[79] 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 PROOF OF MUSES (THEOREM 1)

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 $l \in \{0, \ldots, t-1\}$, Hist_{l+1} can fall into four cases:

(*1*) $\mathsf{CorruptWriterO}_b$ on (i_0, i_1): It will only be answered when identifier $i = i_0 = i_1$. As the update token provided by the corrupted writer is revealed, it requires that the update tuples provided by the writer i (*i.e.*, $\mathsf{UpdateBy}(i)$) 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 $(\{\mathcal{W}'_k, w_k\}_{k\in\{0,1\}})$: 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. Otherwise, if server $\mathcal{P}_L$ is included in the corrupted servers together with $L-2$ remaining servers who hold permutations, the search volume of w_k (*i.e.*, $\mathsf{sv}(w_k)$) is revealed,

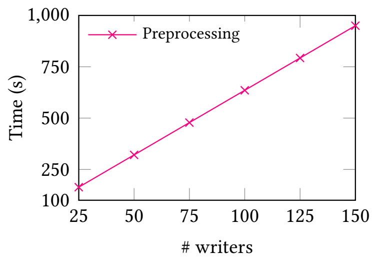


Figure 13: Preprocessing phase of MUSES in 2-server setting.

and the oracle only answers when $\mathsf{sv}(w_0) = \mathsf{sv}(w_1)$ in this case. The adversary still lacks a permutation of the honest (uncorrupted) server to completely recover the search output.

Depending on whether any writer in $\mathcal{W}'$ has been corrupted, there are two following cases:

- If $\exists$ Writer $i : i \in \mathcal{W}' \wedge (\mathsf{CorruptWriter}, i) \in \mathcal{H}$, $\mathcal{A}$ will have knowledge of the correct search output of the corresponding corrupted writer, and the oracle only answers when $w = w_0 = w_1$ in this case.
- Otherwise, because the adversary only gets access to at most $L - 1$ out of L servers, the adversary cannot tell the difference between the indices corresponding to the search queries, as well as whether the search output is correct or not.

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

(*3*) $\mathsf{UpdateO}_b$ on $(\{i_k, u_k, \mathbf{w}_k\}_{k\in\{0,1\}})$: The oracle answers the queries when $i = i_0 = i_1$ 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.*, st_u) will be leaked during update.

Obviously, if $(\mathsf{CorruptWriter}, i) \in \mathcal{H}$, $\mathcal{A}$ will have the knowledge of the update token. In particular, if writer i has been corrupted, the BF representation of keywords in $\mathbf{w}_k$, 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 IND-CPA security of the symmetric encryption scheme $\mathcal{E}$ and computationally indistinguishable security of KH-PRF.

(*4*) $\mathsf{RevokeO}_b$ on $(\{i_k\}_{k\in\{0,1\}})$: The oracle answers the queries when $i = i_0 = i_1$, since the writer identifier will be leaked in permission revocation. In addition, if $(\mathsf{CorruptWriter}, i) \in \mathcal{H}$, $\mathcal{A}$ will still have the knowledge of the updated secret tokens. Specifically, if writer i has been corrupted, $\mathcal{A}$ who holds κ_{w_i} can still decrypt to obtain secret keys contained in secret tokens of the writer i even after they are updated.

The indistinguishability between $\mathbf{Hybrid}_l$ and $\mathbf{Hybrid}_{l+1}$ is guaranteed by IND-CPA security of the symmetric encryption scheme $\mathcal{E}$ and 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.

□

B QUERY-INDEPENDENT PREPROCESSING

Figure 13 presents the time for preprocessing of our MUSES to run LSS.ShrTrns in the two-sever setting, which shows that it is linear to the number of writers. The overhead of the preprocessing phase mostly comes from Share Translation protocol between two servers: $(\boldsymbol{\Delta}; \mathbf{a}, \mathbf{b}) \leftarrow \mathsf{TSS.ShrTrns}(\pi; 1^\lambda)$, which lets one server obtain $\mathbf{a}, \mathbf{b}$, and the other server learn the corresponding translation function $\boldsymbol{\Delta} \leftarrow \mathbf{b} - \pi(\mathbf{a})$ without revealing the permutation π. However, as the preprocessing phase is independent with the search queries, it can be performed in the offline phase, then the results are stored in a temporary memory and used when a search happens. We implement the primitive version of Share Translation protocol in [18] for our evaluation. The advanced version of it uses special structure of Benes permutation network to accelerate and achieve better performance [18].

C PREVENTING ROLLBACK ATTACKS

We present a (simple) extension to our semi-honest MUSES construction to prevent rollback attacks, where a corrupt server may omit some updates from the writers or use a tampered writer' index to process the reader's request. 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 w is the keyword that the reader wants 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:

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

WLOG, assume $(\mathcal{P}_2, \mathcal{P}_3)$ are honest and $\mathcal{P}_1$ is corrupt, in which it uses a mutated search index $\mathsf{EIDX}'_i = \mathsf{EIDX}_i + \epsilon$ (compared with EIDX_i in $\mathcal{P}_2$ and $\mathcal{P}_3$). As DPF and KH-PRF are computed on EIDX'_i, 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.