Multi-User Dynamic Searchable Encryption for Prefix-Fixing Predicates from Symmetric-Key Primitives

Takato Hirano[1], Yutaka Kawai[1], Yoshihiro Koseki[1], Satoshi Yasuda[1], Yohei Watanabe[2], Takumi Amada[2], Mitsugu Iwamoto[2], and Kazuo Ohta[2]

[1] Mitsubishi Electric Corporation, Kamakura, Japan
Hirano.Takato@ay.MitsubishiElectric.co.jp,
Kawai.Yutaka@da.MitsubishiElectric.co.jp,
Koseki.Yoshihiro@ak.MitsubishiElectric.co.jp,
Yasuda.Satoshi@ea.MitsubishiElectric.co.jp
[2] The University of Electro-Communications, Chofu, Japan
{watanabe, mitsugu, kazuo.ohta}@uec.ac.jp

Abstract. *Dynamic searchable symmetric encryption* (SSE) enables clients to update and search encrypted data stored on a server and provides efficient search operations instead of leakages of inconsequential information. Towards dynamic SSE for more practical situations, researchers have tackled research on multi-user dynamic SSE (dynamic MUSE for short), sometimes with flexible access control. However, existing schemes assumed a trusted authority that played a crucial role during the search protocol, used pairings/lattices for their constructions, or provided no security proofs.

In this paper, we show the first dynamic MUSE scheme with reasonable access control without the above limitations. Namely, our dynamic MUSE model requires no trusted authority, and our concrete dynamic MUSE scheme with *prefix-fixing predicates*, which yields a reasonable, hierarchical access control, can be constructed from only symmetric-key primitives and with rigorous security proof. Our experimental results show that our dynamic MUSE scheme is reasonably efficient.

Keywords: Multi-user dynamic searchable symmetric encryption · Prefix-fixing predicates · Access control.

1 Introduction

Searchable symmetric encryption (SSE), which enables us to provide a way to search a large database efficiently (e.g., cloud storage) for *encrypted* data, has attracted attention over the past two decades. SSE was originally proposed by Song et al. [36], and later formalized by Curtmola

et al. [14]. In particular, dynamic SSE, which supports file update operations, has recently become an active area of SSE research [8, 11, 12, 17, 19, 22, 23, 31, 32, 40, 46] due to its practicality.

SSE can be used to utilize private data in a privacy-preserving way. There are, of course, various such advanced cryptographic protocols; for instance, homomorphic encryption (HE) is another option. However, in those cryptographic protocols, users with secret keys can access all data. For example, SSE users can search all encrypted data, and HE users are allowed to decrypt all ciphertexts. In a large-scale system where many users are involved, it is often desirable to support flexible access control; an administrator should access all data, whereas system users should have limited access to the data depending on user's attributes.

Multi-User (Dynamic) SSE with Flexible Access Control. Basically, update and search operations of dynamic SSE schemes are performed with a single secret key, which implies that there is only one user in dynamic SSE. Although dynamic SSE schemes can be naively extended to a multi-user setting by sharing the single secret key among all users, the security is easily compromised once a single user is corrupted. Taking into account practical encrypted search systems, dynamic SSE should support collusion resistance, i.e., dynamic SSE should guarantee security even if (polynomially many) users are corrupted. To this end, Curtmola et al. [14, 15] introduced *multi-user SSE* (MUSE), which allows multiple users to participate in the protocol with different keys, and it achieves collusion resistance. Following the seminal work, there are various lines of research on (dynamic) MUSE such as the multi-writer/multi-reader (MW/MR) setting [4, 29, 30, 33, 43, 48], the multi-writer/single-reader (MW/SR) setting [20, 27, 28, 42] including public-key encryption with keyword search (PEKS) [5], and the single-writer/multi-reader (SW/MR) setting [1, 2, 13, 16, 21, 24, 39, 44, 47]. In this paper, we focus on the SW/MR setting, which has been most researched. Meanwhile, most MUSE schemes support the attribute-based search, which aims to provide flexible access control to the contents of the database. However, the existing schemes still have either of the following limitations.

(i) *Access control* [13, 16, 24, 39, 44]. Their MUSE schemes only support restricted access control. Specifically, user secret keys are associated with keywords, not attributes. Therefore, keywords users can search for must be determined when generating their secret keys.

(ii) *Model* [1, 2, 4, 16, 29, 30]. A trusted third-party or trusted execution environment (TEE) is required. That is, users need to ask the trusted authority or use the TEE to compute something for operations.

(iii) *Construction* [1–4, 20, 26–30, 33, 38, 42, 43, 47, 48]. The proposed schemes are constructed from identity-based encryption (IBE) [6] and attribute-based encryption (ABE) [18, 35], which implies rich algebraic structures such as pairings or lattices, or directly from pairings. There are a few RSA-based and DDH-based schemes [3, 24, 39, 44], though they still employ public-key primitives and techniques.

(iv) *Security* [1, 2, 16, 45]. No security proofs are provided.

In this paper, we aim to overcome all the above issues. In other words, our aim in this paper is to realize a dynamic MUSE scheme that: (1) supports *reasonable and flexible* access control; (2) does not rely on any trusted authority and TEEs during the search operations; (3) can be constructed from only symmetric-key primitives; and (4) provides provable security.

Motivating Scenario. As can be seen in the ABE research [18, 35], there is a trade-off between the richness of access controls and efficiency. The more we try to make access control flexible, the more difficult it is to find an efficient solution that does not rely on public-key techniques and structures such as lattices and pairings. Therefore, to achieve our aims (1)–(4), we consider a reasonable access control that works well enough for the following scenario. Suppose a search system in a certain organization with a hierarchical structure, such as companies. For instance, an employee has a title T and is with a specific section S in a specific division D, and it can be expressed as a vector, say, $(\mathsf{D}, \mathsf{S}, \mathsf{T})$. We consider a search system where a manager uploads documents to a server accessible to all employees. We can, of course, apply standard DSSE schemes to make the search system secure. However, it is insufficient to support the situation where, e.g., some of them are confidential to only a particular section $\mathsf{S}^\star$. The manager wants to make the documents accessible to only all employees in $\mathsf{S}^\star$. That is, the manager want to specify a hierarchical access structure, i.e., $(\mathsf{D}, \mathsf{S}^\star, \star)$, where '$\star$' indicates a 'wildcard,' and upload the documents in an encrypted form so that only the employees in Section $\mathsf{S}^\star$ access them. However, the standard DSSE schemes allow *all employees in the company* to access the data.

Based on the above scenario, in this paper, we consider a dynamic MUSE scheme with a hierarchical access structure.

1.1 Our Contribution

In this paper, we propose a new dynamic MUSE scheme with the above features (1)–(4). Specifically, we introduce a new model of (attribute-based) dynamic MUSE. There are a data owner, users, and a server; hence, our model requires no trusted authority or TEEs during the search protocol. We define the syntax of dynamic MUSE and formalize its security notions.[3] We then construct a dynamic MUSE scheme with prefix-fixing predicates [7, 9, 25], which supports the hierarchical access structure, from only symmetric-key primitives. Therefore, our scheme provides both reasonable access control and efficient operations. We show the proposed construction is secure in the random oracle model and also implement our scheme to show experimental results.

1.2 Notations

For any integer $a, b \in \mathbb{Z}$ and $a \leq b$, let $[a, b]$ be an integer set $\{a, a + 1, \ldots, b\}$. In particular, we simply denote $[b]$ if $a = 1$. For a finite set $\mathcal{X}$, we use $x \xleftarrow{\$} \mathcal{X}$ to represent processes of choosing an element x from X uniformly at random. For a finite set $\mathcal{X}$, we denote by $\mathcal{X} \leftarrow x$ and $|\mathcal{X}|$ the addition x to $\mathcal{X}$ and cardinality of $\mathcal{X}$, respectively. Concatenation of a and b is denoted by $a\|b$. In the description of the algorithm, all arrays, strings, and sets are initialized to empty ones. For any non-interactive algorithm A, $\mathsf{out} \leftarrow \mathsf{A}(\mathsf{in})$ means that A takes in as input and outputs out. We denote by A^{O} A allowed access to an oracle O. In this paper, we consider two-party interactive algorithms between a client and a server, and it is denoted by $(\mathsf{out}_{\mathsf{U}}; \mathsf{out}_{\mathsf{S}}) \leftarrow \mathsf{A}(\mathsf{in}_{\mathsf{U}}; \mathsf{in}_{\mathsf{S}})$, where in_{U} and in_{S} are input of client and server, respectively and $\mathsf{out}_{\mathsf{U}}$ and $\mathsf{out}_{\mathsf{S}}$ are output of client and server, respectively. Throughout the paper, we denote by κ a security parameter and consider probabilistic polynomial-time algorithms (PPTAs). We say a function $\mathsf{negl}(\cdot)$ is negligible if for any polynomial $\mathsf{poly}(\cdot)$, there exists some constant $\kappa_0 \in \mathbb{N}$ such that $\mathsf{negl}(\kappa) < 1/\mathsf{poly}(\kappa)$ for all $\kappa \geq \kappa_0$.

2 Multi-User (Attribute-Based) Dynamic Searchable Symmetric Encryption

We introduce a new model of multi-user dynamic searchable symmetric encryption (dynamic MUSE for short), which enables a data owner

[3] Note that in this paper, we only consider addition procedures as dynamic update operations. One might consider dynamic MUSE schemes that support more flexible update operations, such as deletion.

to handle attribute-based access control. Let $\mathcal{U}$ be the universe of attributes. Roughly speaking, the data owner who has a master secret key msk can encrypt a document file f and its corresponding keywords under an attribute set $\mathcal{A} \subset \mathcal{U}$, and stores them to a server. The data owner can create a secret key sk_P for a *predicate* P. A user who has a secret key sk_P searches for a keyword q and obtains a search result from the server in a privacy-preserving way. The user only obtains search results if an attribute set $\mathcal{A}$ associated with each entry (id, q) satisfies $\mathsf{P}(\mathcal{A}) = 1$, where id is an identifier of a document file f_id. In other words, the file identifier id is not included in the search result for q if the corresponding attribute set $\mathcal{A}$ satisfies $\mathsf{P}(\mathcal{A}) = 0$.

Notations for Database. Let λ and ℓ be polynomials in κ. Let $\Lambda \coloneqq \{0,1\}^\lambda$ be a set of possible keywords (sometimes called a *dictionary*), and $\mathcal{F}$ be a set of possible document files. We assume that each file $f_\mathsf{id} \in \mathcal{F}$ has the corresponding identifier $\mathsf{id} \in \{0,1\}^\ell$, which is irrelevant to the content of f_id (e.g., document numbers). The data owner adds a pair of a file identifier and keyword with an attribute set $\mathcal{A}$ to the database (in an encrypted form). Therefore, a database DB is represented as a set of $((\mathsf{id}, w), \mathcal{A})$, and we denote its encrypted version by EDB. For any $w \in \Lambda$, and $\mathsf{P} \in \mathcal{P}$, let $\mathsf{ID}_{w,\mathsf{P}}$ be a set of identifiers that contains w and satisfies the predicate P (i.e., $\mathsf{ID}_{w,\mathsf{P}} \coloneqq \{\mathsf{id} \mid ((\mathsf{id}, w), \mathcal{A}) \in \mathsf{DB}\} \wedge \mathsf{P}(\mathcal{A}) = 1$), which corresponds to a search result for w under P.

2.1 Syntax

We give a formal definition of dynamic MUSE, which is introduced in this paper. Our definition is based on ABE (e.g., [18, 35]) and dynamic SSE (e.g., [11, 22, 23]), and requires no trusted third party or TEEs as in previous works [1, 2, 4, 16, 29, 30].

Definition 1 (Dynamic MUSE). *A multi-user dynamic searchable symmetric encryption (dynamic MUSE) scheme Σ over Λ consists of five-tuple algorithms $\Sigma \coloneqq (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Add}, \mathsf{Search}, \mathsf{Dec})$, which are defined as follows:*

- $(\mathsf{msk}, \mathsf{EDB}) \leftarrow \mathsf{Setup}(1^\kappa, \mathcal{P}, \mathcal{U})$*: It is a non-interactive probabilistic algorithm that takes a security parameter κ, a predicate set $\mathcal{P}$, and an attribute universe $\mathcal{U}$ as input and outputs a master secret key* msk *and initial encrypted database* EDB.
- $\mathsf{sk}_\mathsf{P} \leftarrow \mathsf{KeyGen}(\mathsf{msk}, \mathsf{P})$*: It is a non-interactive algorithm that takes* msk *and a predicate $\mathsf{P} \in \mathcal{P}$ as input, and outputs a secret key* sk_P *for* P.

- $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{msk}, f_{\mathsf{id}}, \mathcal{A})$*: It is a non-interactive algorithm that takes* msk*, a file* $f_{\mathsf{id}} \in \mathcal{F}$*, and an attribute set* $\mathcal{A}$ *as input, and outputs a ciphertext* ct*.*
- $(\mathsf{ack}; \mathsf{EDB}') \leftarrow \mathsf{Add}(\mathsf{msk}, (\mathsf{id}, w), \mathcal{A}; \mathsf{EDB})$*: It is an interactive algorithm that consists of* $\mathsf{Add}_{\mathrm{O}}$ *and* $\mathsf{Add}_{\mathrm{S}}$*.* $\mathsf{Add}_{\mathrm{O}}$ *takes* msk*,* (id, w)*, and an attribute set* $\mathcal{A}$ *as input, and outputs acknowledgement* ack*. Similarly,* $\mathsf{Add}_{\mathrm{S}}$ *takes* EDB *as input and outputs* EDB'*.*
- $(\mathcal{X}_{q,\mathsf{P}}; \mathsf{EDB}') \leftarrow \mathsf{Search}(\mathsf{sk}_{\mathsf{P}}, q; \mathsf{EDB})$*: It is an interactive algorithm that consists of* $\mathsf{Search}_{\mathrm{U}}$ *and* $\mathsf{Search}_{\mathrm{S}}$*.* $\mathsf{Search}_{\mathrm{U}}$ *takes* sk_{P} *and a keyword* q *to be searched as input, and outputs a search result* $\mathcal{X}_{q,\mathsf{P}}$*.* $\mathsf{Search}_{\mathrm{S}}$ *updates* EDB *to* EDB'*.*
- $f_{\mathsf{id}}/\perp \leftarrow \mathsf{Dec}(\mathsf{sk}_{\mathsf{P}}, \mathsf{ct})$*: It is a non-interactive algorithm that takes* sk_{P} *and* ct *as input, and outputs a file* f_{id} *or* $\perp$*, which indicates decryption failure.*

A dynamic MUSE scheme Σ satisfies the correctness if the following two conditions hold with overwhelming probability: (1) for any file $f \in \mathcal{F}$, any attribute set $\mathcal{A} \subset \mathcal{U}$, and any predicate $\mathsf{P} \in \mathcal{P}$, a ciphertext of f encrypted with $\mathcal{A}$ is correctly decrypted by sk_{P}; and (2) for any keyword $q \in \Lambda$ and any predicate $\mathsf{P} \in \mathcal{P}$, Search with sk_{P} outputs $\mathcal{X}_{q,\mathsf{P}}$ satisfying $\mathcal{X}_{q,\mathsf{P}} = \mathsf{ID}_{q,\mathsf{P}}$. The formal definition is given in Appendix A.

2.2 Security Definition

Following most SSE works, we provide the simulation-based security definition for dynamic MUSE. As in SSE, we consider an honest-but-curious server as an adversary, and allow the server to corrupt users to get their secret keys. We allow some leakage during operations to make the protocol efficient. Such information leakage is characterized as a *leakage function* $\mathcal{L} := (\mathcal{L}_{\mathsf{Setup}}, \mathcal{L}_{\mathsf{KG}}, \mathcal{L}_{\mathsf{Enc}}, \mathcal{L}_{\mathsf{Add}}, \mathcal{L}_{\mathsf{Srch}})$. To put it briefly, $\mathcal{L}_{\mathsf{Setup}}$, $\mathcal{L}_{\mathsf{KG}}$, $\mathcal{L}_{\mathsf{Enc}}$, $\mathcal{L}_{\mathsf{Add}}$, and $\mathcal{L}_{\mathsf{Srch}}$ are information leaked during the setup, key generation, encryption, addition, and search operations, respectively.

$(\mathcal{L}, \mathcal{P})$-Adaptive Security. We consider a security notion for dynamic MUSE based on adaptive security of dynamic SSE [37]. The notion, called $(\mathcal{L}, \mathcal{P})$-adaptive security, is parameterized by a leakage function $\mathcal{L}$ and a predicate class $\mathcal{P}$. Intuitively, $\mathcal{L}$-adaptive security guarantees that no information is leaked other than $\mathcal{L}$ even if an adversary, i.e., the honest-but-curious server, adaptively corrupts users and performs update and search operations. Formally, we consider the following two experiments: a real experiment $\mathsf{Exp}^{\mathsf{Real}}_{\mathsf{D},\mathcal{P}}$ between a PPT algorithm D and the client, who

plays the role of both the data owner and users; and an ideal experiment $\mathsf{Exp}^{\mathsf{Ideal}}_{\mathsf{D},\mathsf{S},\mathcal{L},\mathcal{P}}$ between D and a simulator S.

Definition 2 ($(\mathcal{L},\mathcal{P})$-adaptive security). *Let Σ be a dynamic MUSE scheme. Σ is said to be $(\mathcal{L},\mathcal{P})$-adaptively secure if for any PPT algorithm D, there exists a PPT algorithm S and it holds $|\Pr[\mathsf{Exp}^{\mathsf{Real}}_{\mathsf{D},\mathcal{P}}(\kappa)=1]-\Pr[\mathsf{Exp}^{\mathsf{Ideal}}_{\mathsf{D},\mathsf{S},\mathcal{L},\mathcal{P}}(\kappa)=1]|<\mathsf{negl}(\kappa)$, where $\mathsf{Exp}^{\mathsf{Real}}_{\mathsf{D},\mathcal{P}}$ and $\mathsf{Exp}^{\mathsf{Ideal}}_{\mathsf{D},\mathsf{S},\mathcal{L},\mathcal{P}}$ are defined as follows.*

$\mathsf{Exp}^{\mathsf{Real}}_{\mathsf{D},\mathcal{P}}(\kappa)$*:* D *sends the client a* `setup` *message together with the attribute universe $\mathcal{U}$. The client runs* $\mathsf{Setup}(1^\kappa,\mathcal{P},\mathcal{U})$ *to obtain* msk *and* EDB*, and returns* EDB *to* D*.* D *then adaptively issues* `kg`*,* `enc`*,* `add`*, and* `srch` *queries as follows.*

- *For* `kg`*,* D *gives the client a predicate $\mathsf{P}\in\mathcal{P}$, and the client returns $\mathsf{sk}_\mathsf{P}\leftarrow\mathsf{KeyGen}(\mathsf{msk},\mathsf{P})$ to* D *and adds sk_P to $\mathsf{SKList}[\mathsf{P}]$.*
- *For* `enc`*,* D *gives the client a file $f_\mathsf{id}\in\mathcal{F}$ and an attribute set $\mathcal{A}\subset\mathcal{U}$, and the client returns $\mathsf{Enc}(\mathsf{msk},f_\mathsf{id},\mathcal{A})$ to* D*.*
- *For* `add`*,* D *gives the client an entry $(\mathsf{id},w)\in\{0,1\}^\ell\times\Lambda$ and an attribute set $\mathcal{A}\subset\mathcal{U}$, and the client runs $\mathsf{Add}(\mathsf{msk},(\mathsf{id},w),\mathcal{A};\mathsf{EDB})$ with* D*.* D *obtains the output, i.e., EDB', and the transcript* trans *during the protocol.*
- *For* `srch`*,* D *gives the client a predicate $\mathsf{P}\in\mathcal{P}$ and a keyword $q\in\Lambda$, and the client runs $\mathsf{Search}(\mathsf{SKList}[\mathsf{P}],q;\mathsf{EDB})$ with* D*. The client returns the search result $\mathcal{X}_{q,\mathsf{P}}$ to* D*, and* D *obtains the output, i.e., EDB', and the transcript* trans *during the protocol.*

Finally, D *outputs a bit b as the output of the experiment.*

$\mathsf{Exp}^{\mathsf{Ideal}}_{\mathsf{D},\mathsf{S},\mathcal{L},\mathcal{P}}(\kappa)$*:* D *sends the client a* `setup` *message together with the attribute universe $\mathcal{U}$. The client forwards the message to* S*.* S *creates* EDB *from $\mathcal{L}_{\mathsf{Setup}}(1^\kappa,\mathcal{U},\mathcal{P})$ and returns it to* D *(via the client).* D *then adaptively issues* `kg`*,* `enc`*,* `add`*, and* `srch` *queries as follows.*

- *For* `kg`*,* D *gives the client a predicate $\mathsf{P}\in\mathcal{P}$, and the client forwards it to* S*.* S *creates and returns sk_P from $\mathcal{L}_{\mathsf{KG}}(\mathsf{P})$ to* D*, and adds sk_P to $\mathsf{SKList}[\mathsf{P}]$.*
- *For* `enc`*,* D *gives the client a file $f_\mathsf{id}\in\mathcal{F}$ and an attribute set $\mathcal{A}\subset\mathcal{U}$, and the client forwards them to* S*.* S *creates* ct *from $\mathcal{L}_{\mathsf{Enc}}(f_\mathsf{id},\mathcal{A})$ and returns it to* D*.*
- *For* `add`*,* D *gives the client an entry $(\mathsf{id},w)\in\{0,1\}^\ell\times\Lambda$ and an attribute set $\mathcal{A}\subset\mathcal{U}$, and the client forwards them to* S*.* S *communicates with* D *and simulates the* Add *algorithm using $\mathcal{L}_{\mathsf{Add}}((\mathsf{id},w),\mathcal{A})$.*

D *obtains the output, i.e.,* EDB'*, and the transcript* trans *during the protocol.*

- *For* `srch`*,* D *gives the client a predicate* $\mathsf{P} \in \mathcal{P}$ *and a keyword* $q \in \Lambda$*, and the client forwards them to* S*.* S *communicates with* D *and simulates the* Search *algorithm using* $\mathcal{L}_{\mathsf{Srch}}(\mathsf{P}, q)$*.* S *returns the search result* $\mathcal{X}_{q,\mathsf{P}}$ *to* D*, and* D *obtains the output, i.e.,* EDB'*, and the transcript* trans *during the protocol.*

Finally, D *outputs a bit* b *as the output of the experiment.*

3 Our Construction

3.1 Prefix-Fixing Predicates

We employ attribute vectors $(a_1, a_2, \ldots, a_\ell)$ that realize a hierarchical access structure described in the introduction and consider a set of them as an attribute set $\mathcal{A}$ for our construction. Let $\mathcal{U} \coloneqq \{(a_1, a_2, \ldots, a_\ell) \in \{0,1\}^\ell\}$ for some $\ell = \mathsf{poly}(\kappa)$. We consider prefix-fixing predicates [7, 9, 25]. We say a vector $\mathbf{p} \in \{0, 1, \star\}^\ell$ is a prefix-fixing vector if there exists $j \in [\ell]$ such that $\mathbf{p} \in \{0,1\}^j \times \{\star\}^{\ell-j}$, where '$\star$' indicates 'wild-card'. For a prefix-fixing vector $\mathbf{p} = (p_1, p_2, \ldots, p_\ell) \in \{0, 1, \star\}^\ell$ and $\mathcal{A} = \{(a_{i,1}, a_{i,2}, \ldots, a_{i,\ell})\}_{i=1}^n \subset \mathcal{U}$ $(n = \mathsf{poly}(\kappa))$, the prefix-fixing predicate $\mathsf{P}_{\mathbf{p}}^{(\mathrm{PF})} : 2^{\mathcal{U}} \to \{0,1\}$ is defined as

$$\mathsf{P}_{\mathbf{p}}^{(\mathrm{PF})}(\mathcal{A}) = 1 \iff \exists i \in [n] \text{ s.t. } (p_j = a_{i,j} \text{ or } p_j = \star) \text{ for } \forall j \in [\ell].$$

Let $\mathcal{P}^{(\mathrm{PF})} \coloneqq \bigcup_{j=1}^{\ell} \{\mathsf{P}_{\mathbf{p}}^{(\mathrm{PF})} \mid \mathbf{p} \in \{0,1\}^j \times \{\star\}^{\ell-j}\}$.

3.2 Our Dynamic MUSE Scheme with Prefix-Fixing Predicates

Construction Idea. Our aim is to realize a dynamic MUSE scheme that supports the above access control from only symmetric-key primitives. To this end, we employ a hash chain to achieve access control with prefix-fixing predicates. First, a secret key $\mathsf{sk}_{\mathsf{P}_{\mathbf{p}}^{(\mathrm{PF})}}$ for a predicate $\mathbf{p} = (p_1, \ldots, p_j, \star, \ldots, \star)$ generated from msk can be computed as $\mathsf{sk}_{\mathsf{P}_{\mathbf{p}}^{(\mathrm{PF})}} \coloneqq \mathsf{H}_\kappa(\mathsf{msk} \| p_1 \| \cdots \| p_j \| \star \| \cdots \| \star)$. For simplicity, we now consider a simple attribute $\mathcal{A} = (a_1, a_2, \ldots, a_\ell)$ such that its first j elements are the same as the first j elements of the predicate $\mathbf{p}$, i.e., $(a_1, \ldots, a_j) = (p_1, \ldots, p_j)$. Since it holds $\mathsf{P}_{\mathbf{p}}^{(\mathrm{PF})}(\mathcal{A}) = 1$, a ciphertext ct of a file f_{id} under the attribute $\mathcal{A}$ should be decrypted by $\mathsf{sk}_{\mathsf{P}_{\mathbf{p}}^{(\mathrm{PF})}}$. Therefore, we prepare the following ℓ

session keys for $(a_1, \ldots, a_i, \star, \ldots, \star)$ for all $i \in [\ell]$ to handle any level of wildcards: $k_i \coloneqq \mathsf{H}_\kappa(\mathsf{H}_\kappa(\mathsf{msk} \| a_1 \| \cdots \| a_i \| \star \| \cdots \| \star) \| s)$, where s is nonce to randomize the session keys per ciphertext. All the session keys are used to mask a single randomness r as $c_i \coloneqq k_i \oplus r$ for all $i \in [\ell]$. The randomness r is also used to produce a file ciphertext $c_f \coloneqq f_{\mathsf{id}} \oplus \mathsf{H}_{|f_{\mathsf{id}}|}(r \| 0)$ and a verification ciphertext $c_v \coloneqq \mathsf{H}_\kappa(r \| 1)$, and the entire ciphertext is $\mathsf{ct} \coloneqq (c_f d, s, \{c_i\}_{i=1}^{\ell}, c_v)$. The user with $\mathsf{sk}_{\mathsf{P}_{\mathbf{p}}^{(\mathrm{PF})}}$ can retrieve r_j from c_j (with the validity check $c_v = \mathsf{H}_\kappa(r_j \| 1)$), and hence decrypt the file ciphertext $f_{\mathsf{id}} = c_f \oplus \mathsf{H}_{|f_{\mathsf{id}}|}(r_j \| 0)$. Trapdoors to search a keyword q and tags for a newly-added entry (id, q) can be constructed in the same manner.

Construction. For any $k \in \mathbb{N}$, let $\mathsf{H}_k : \{0,1\}^* \to \{0,1\}^k$ be a hash function that maps a variable-length string to a k-bit one.

- $\mathsf{Setup}(1^\kappa, \mathcal{P}, \mathcal{U}\ (= \{0,1\}^\ell))$: It outputs $\mathsf{msk} \xleftarrow{\$} \{0,1\}^\kappa$ and an empty array EDB.
- $\mathsf{KeyGen}(\mathsf{msk}, \mathsf{P}_{\mathbf{p}}^{(\mathrm{PF})} \in \mathcal{P}^{(\mathrm{PF})})$: Parse $\mathbf{p}$ as $(p_1, p_2, \ldots, p_\ell)$. It outputs $\mathsf{sk}_{\mathbf{p}} \coloneqq \mathsf{H}_\kappa(\mathsf{msk} \| p_1 \| \cdots \| p_\ell)$, where we write $\mathsf{sk}_{\mathbf{p}}$ as $\mathsf{sk}_{\mathsf{P}_{\mathbf{p}}^{(\mathrm{PF})}}$ for simple and compact notations.
- $\mathsf{Enc}(\mathsf{msk}, f_{\mathsf{id}}, \mathcal{A})$: Parse $\mathcal{A}$ as $\{(a_{i,1}, a_{i,2}, \ldots, a_{i,\ell})\}_{i=1}^n$. It randomly chooses $s, r \xleftarrow{\$} \{0,1\}^\kappa$. For every $j \in [\ell]$, it computes

$$c_{i,j} \coloneqq \mathsf{H}_\kappa(\mathsf{H}_\kappa(\mathsf{msk} \| a_{i,1} \| a_{i,2} \| \cdots \| a_{i,j} \| \underbrace{\star \| \cdots \| \star}_{\ell - j \text{ symbols}}) \| s) \oplus r,$$

  for every $i \in [n]$, and sets $\mathcal{C}_{\mathcal{A},j} \coloneqq \{c_{1,j}, \ldots, c_{n,j}\}$ with deduplication.[4] It then computes $c_f \coloneqq f_{\mathsf{id}} \oplus \mathsf{H}_{|f_{\mathsf{id}}|}(r \| 0)$ and $c_v \coloneqq \mathsf{H}_\kappa(r \| 1)$. It outputs $\mathsf{ct} \coloneqq (c_f, s, \{\mathcal{C}_{\mathcal{A},j}\}_{j=1}^{\ell}, c_v)$.
- $\mathsf{Add}(\mathsf{msk}, (\mathsf{id}, w), \mathcal{A}; \mathsf{EDB})$: Parse $\mathcal{A}$ as $\{(a_{i,1}, a_{i,2}, \ldots, a_{i,\ell})\}_{i=1}^n$. The owner-side algorithm $\mathsf{Add}_{\mathrm{O}}$ randomly chooses $t, \gamma \xleftarrow{\$} \{0,1\}^\kappa$. For every $j \in [\ell]$, it computes

$$\mathsf{tag}_{i,j} \coloneqq \mathsf{H}_\kappa(\mathsf{H}_\kappa(\mathsf{H}_\kappa(\mathsf{msk} \| a_{i,1} \| a_{i,2} \| \cdots \| a_{i,j} \| \underbrace{\star \| \cdots \| \star}_{\ell - j \text{ symbols}}) \| w) \| t) \oplus \gamma,$$

[4] There might exist two attributes $(a_{i_1,1}, a_{i_1,2}, \ldots, a_{i_1,\ell})$ and $(a_{i_2,1}, a_{i_2,2}, \ldots, a_{i_2,\ell})$ whose prefixes are common, i.e., $\exists j \in [\ell]$ such that $(a_{i_1,1}, a_{i_1,2}, \ldots, a_{i_1,j}) = (a_{i_2,1}, a_{i_2,2}, \ldots, a_{i_2,j})$. Then, $c_{i_1,j} = c_{i_2,j}$ clearly holds. 'Deduplication' means that such duplicate ciphertexts are deduplicated, and hence $|\mathcal{C}_{\mathcal{A},j}| \leq n$ holds.

for every $i \in [n]$, and sets $\mathcal{T}_{\mathcal{A},j} \coloneqq \{\mathsf{tag}_{1,j}, \ldots, \mathsf{tag}_{n,j}\}$ with deduplication. It computes $\mathsf{tag}_v \coloneqq \mathsf{H}_\kappa(\gamma)$, sends $\mathsf{trans}_1 \coloneqq (\mathsf{id}, t, \mathcal{T}_{\mathcal{A}}, \mathsf{tag}_v)$ to the server, and outputs ack. The server-side algorithm Add_S adds (id, $t, \{\mathcal{T}_{\mathcal{A},j}\}_{j=1}^{\ell}, \mathsf{tag}_v$) to EDB, and outputs $\mathsf{EDB}' \coloneqq \mathsf{EDB}$.

– $\mathsf{Search}(\mathsf{sk}_{\mathbf{p}}, q; \mathsf{EDB})$: Suppose $\mathbf{p} = (p_1, \ldots, p_j, \star, \ldots, \star)$. The user-side algorithm $\mathsf{Search}_\mathrm{U}$ computes $\mathsf{trpdr} \coloneqq \mathsf{H}_\kappa(\mathsf{sk}_{\mathbf{p}} \| q)$, and sends $\mathsf{trans}_1 \coloneqq (j, \mathsf{trpdr})$ to the server. The server-side algorithm $\mathsf{Search}_\mathrm{S}$ does the following for every $(\mathsf{id}, t, \{\mathcal{T}_{\mathcal{A},j}\}_{j=1}^{\ell}, \mathsf{tag}_v) \in \mathsf{EDB}$.

  1. For every $\mathsf{tag} \in \mathcal{T}_{\mathcal{A},j}$, it computes $\gamma = \mathsf{H}_\kappa(\mathsf{trpdr} \| t) \oplus \mathsf{tag}$.
  2. If $\mathsf{H}_\kappa(\gamma) = \mathsf{tag}_v$ holds, it adds id to $\mathcal{X}_{q,\mathbf{p}}$.

  It then sends the user $\mathsf{trans}_2 \coloneqq \mathcal{X}_{q,\mathbf{p}}$ and outputs $\mathsf{EDB}' \coloneqq \mathsf{EDB}$. $\mathsf{Search}_\mathrm{U}$ outputs $\mathcal{X}_{q,\mathbf{p}}$.

– $\mathsf{Dec}(\mathsf{sk}_\mathsf{P}, \mathsf{ct})$: Suppose $\mathbf{p} = (p_1, \ldots, p_j, \star, \ldots, \star)$. For every $c \in \mathcal{C}_{\mathcal{A},j}$, it computes $r = \mathsf{H}_\kappa(\mathsf{sk}_{\mathbf{p}} \| s) \oplus c$. If $\mathsf{H}_\kappa(r \| 1) = c_v$ holds, it outputs $f \coloneqq c_f \oplus \mathsf{H}_{|c_f|}(r \| 0)$. If there is no such $c \in \mathcal{C}_{\mathcal{A},j}$, it outputs $\perp$.

Theorem 1. *Let $\{\mathsf{H}_k\}_{k \in [\mathsf{poly}(\kappa)]}$ be random oracles. Then, the proposed dynamic MUSE scheme Σ is correct in the random oracle model.*

Due to the page limitation, we give a proof in Appendix B.

3.3 Security

Adversary Model. We consider the following adversary model formally defined by Def. 2: the server corrupts a lot of users, who are expressed as `kg` queries in Def. 2. Using various information expressed as `enc`, `add`, and `srch` queries, they attempt to obtain some information on searched keywords or files such that their attributes do not satisfy the corrupted users' predicates, i.e., $\mathsf{P}_{\mathbf{p}}^{(\mathrm{PF})}(\mathcal{A}) = 0$.

Specific Leakage Functions. We consider the following leakage functions as the specific leakages in our scheme. Let `ctr` be a *global counter*, which is initially set to zero and incremented as each query, to describe a timeline of operations, and let $\mathcal{Q}_{\mathtt{kg}}$, $\mathcal{Q}_{\mathtt{enc}}$, $\mathcal{Q}_{\mathtt{add}}$, and $\mathcal{Q}_{\mathtt{srch}}$ are sets of the `kg`, `enc`, `add`, and `srch` queries (with the counters when issued).

– $\mathcal{L}_{\mathsf{Setup}}(\mathtt{ctr} \coloneqq 0, \kappa, \mathcal{P}^{(\mathrm{PF})}, \mathcal{U}) = (\Lambda, \mathcal{P}^{(\mathrm{PF})}, \mathcal{U})$. Namely, the Setup algorithm leaks nothing secret since a dictionary Λ, a predicate set $\mathcal{P}^{(\mathrm{PF})}$, and an attribute universe $\mathcal{U}$ are publicly available.

– $\mathcal{L}_{\mathsf{KG}}(\mathtt{ctr}, \mathbf{p}) = (\mathbf{p}, \mathsf{TimeF}_{\mathbf{p}}, \mathsf{TimeW}_{\mathbf{p}})$. This leakage captures what information a user who has a secret key $\mathsf{sk}_{\mathbf{p}}$ obtains. We suppose $\mathsf{sk}_{\mathbf{p}}$ contains the information about the corresponding predicate $\mathbf{p}$; this follows (ordinary) ABE research [18, 35]. Specifically, the user can decrypt all ciphertexts for $\mathcal{A}$ such that $\mathsf{P}_{\mathbf{p}}^{(\mathrm{PF})}(\mathcal{A}) = 1$ by using $\mathsf{sk}_{\mathbf{p}}$ and execute the Search algorithm with $\mathsf{sk}_{\mathbf{p}}$ to get search results $\mathcal{X}_{w,\mathbf{p}}$ for any keyword w. Note that these leakages are unavoidable ones. Formally, the decryption results $\mathsf{TimeF}_{\mathbf{p}}$ and the search results $\mathsf{TimeW}_{\mathbf{p}}$ are defined as follows. $\mathsf{TimeF}_{\mathbf{p}} \coloneqq \{(\mathtt{ctr}', f_{\mathsf{id}}) \mid (\mathtt{ctr}', f_{\mathsf{id}}, \mathcal{A}) \in \mathcal{Q}_{\mathtt{enc}} \wedge \mathsf{P}_{\mathbf{p}}^{(\mathrm{PF})}(\mathcal{A}) = 1\}$ is a set of document files f_{id} that have been issued as the $\mathtt{enc}$ queries $(f_{\mathsf{id}}, \mathcal{A})$ at $\mathtt{ctr}'$ and satisfies $\mathsf{P}_{\mathbf{p}}^{(\mathrm{PF})}(\mathcal{A}) = 1$. Similarly, $\mathsf{TimeW}_{\mathbf{p}} \coloneqq \{(\mathtt{ctr}', w) \mid (\mathtt{ctr}', (\mathsf{id}, w), \mathcal{A}) \in \mathcal{Q}_{\mathtt{add}} \wedge \mathsf{P}_{\mathbf{p}}^{(\mathrm{PF})}(\mathcal{A}) = 1\}$ is a set of keywords w that have been issued as the $\mathtt{add}$ queries $((\mathsf{id}, w), \mathcal{A})$ at $\mathtt{ctr}'$ and satisfies $\mathsf{P}_{\mathbf{p}}^{(\mathrm{PF})}(\mathcal{A}) = 1$.

– $\mathcal{L}_{\mathsf{Enc}}(\mathtt{ctr}, f_{\mathsf{id}}, \mathcal{A}) = \begin{cases} (f_{\mathsf{id}}, \mathcal{A}) & \text{if } \exists(\cdot, \mathbf{p}) \in \mathcal{Q}_{\mathtt{kg}} \text{ s.t. } \mathsf{P}_{\mathbf{p}}^{(\mathrm{PF})}(\mathcal{A}) = 1, \\ (|f_{\mathsf{id}}|, \mathcal{A}) & \text{otherwise.} \end{cases}$
Namely, although the server and corrupted users who have secret keys $\mathsf{sk}_{\mathbf{p}}$ can decrypt ciphertexts for $\mathcal{A}$ if $\mathsf{P}_{\mathbf{p}}^{(\mathrm{PF})}(\mathcal{A}) = 1$, they cannot obtain any information about other files (other than their lengths). The leakage of f_{id} is necessary since D can decrypt the ciphertext if D already obtained $\mathsf{sk}_{\mathbf{p}}$ such that $\mathsf{P}_{\mathbf{p}}^{(\mathrm{PF})}(\mathcal{A}) = 1$ (via the $\mathtt{kg}$ query). Note that, in this work, we do not hide attributes $\mathcal{A}$ from adversaries as in ordinary ABE [18, 35].

– $\mathcal{L}_{\mathsf{Add}}(\mathtt{ctr}, (\mathsf{id}, w), \mathcal{A}) = \begin{cases} ((\mathsf{id}, w), \mathcal{A}) & \text{if } \exists(\cdot, \mathbf{p}) \in \mathcal{Q}_{\mathtt{kg}} \text{ s.t. } \mathsf{P}_{\mathbf{p}}^{(\mathrm{PF})}(\mathcal{A}) = 1 \\ & \quad \vee\ \exists(\cdot, \cdot, w) \in \mathcal{Q}_{\mathtt{srch}}, \\ (\mathsf{id}, \mathcal{A}) & \text{otherwise.} \end{cases}$
The Add algorithm leaks no information on a newly added keyword w unless the corresponding attribute $\mathcal{A}$ satisfies $\mathsf{P}_{\mathbf{p}}^{(\mathrm{PF})}(\mathcal{A}) = 1$ for some corrupted user's key $\mathsf{sk}_{\mathbf{p}}$ or the keyword has ever been searched for.

– $\mathcal{L}_{\mathsf{Srch}}(\mathtt{ctr}, \mathbf{p}, q) = \begin{cases} (\mathbf{p}, q, \mathsf{SP}_{q,\mathbf{p}}, \mathsf{AP}_{q,\mathbf{p}}) & \text{if } \exists(\mathtt{ctr}', \mathsf{P}_{\mathbf{p}}^{(\mathrm{PF})}) \in \mathcal{Q}_{\mathtt{kg}}, \\ (\mathbf{p}, \mathsf{SP}_{q,\mathbf{p}}, \mathsf{AP}_{q,\mathbf{p}}) & \text{otherwise,} \end{cases}$
where $\mathsf{SP}_{q,\mathbf{p}}$ and $\mathsf{AP}_{q,\mathbf{p}}$ are defined as follows.

  - A *search pattern* $\mathsf{SP}_{q,\mathbf{p}}$ indicates a search history for a keyword q under a predicate $\mathbf{p}$. Specifically, $\mathsf{SP}_{q,\mathbf{p}} \coloneqq \{\mathtt{ctr}' \mid (\mathtt{ctr}', \mathbf{p}, q) \in \mathcal{Q}_{\mathtt{srch}}\}$ is a set of counters when D has searched for the same keyword under the same predicate.

- An *access pattern* $\mathsf{AP}_{q,\mathbf{p}}$ indicates a search result for a keyword q under a predicate $\mathbf{p}$. Specifically, $\mathsf{AP}_{q,\mathbf{p}} \coloneqq \{(\mathtt{ctr}', \mathsf{id}) \mid (\mathtt{ctr}', (\mathsf{id}, q), \mathcal{A}) \in \mathcal{Q}_{\mathsf{add}} \wedge \mathsf{P}_{\mathbf{p}}^{(\mathrm{PF})}(\mathcal{A}) = 1\}$ is a set of identifiers such that the corresponding keyword is q and the corresponding attribute $\mathcal{A}$ satisfies $\mathsf{P}_{\mathbf{p}}^{(\mathrm{PF})}(\mathcal{A}) = 1$. Namely, the identifiers in $\mathsf{AP}_{q,\mathbf{p}}$ form a correct search result for q under $\mathsf{sk}_{\mathbf{p}}$, i.e., $\mathsf{ID}_{q,\mathsf{P}_{\mathbf{p}}^{(\mathrm{PF})}}$.

A pair of search and access patterns is called the *L1 leakage* [10], which is considered as the standard search leakage profile in (ordinary) SSE schemes. Note that the leakage of q is necessary since D has $\mathsf{sk}_{\mathbf{p}}$ and hence can run the Search algorithm with it and any keyword by itself.

We are now ready to prove the security of our scheme.

Theorem 2. *Let $\{\mathsf{H}_k\}_{k\in[\mathsf{poly}(\kappa)]}$ be random oracles. Then, the proposed dynamic MUSE scheme Σ is $(\mathcal{L}, \mathcal{P}^{(\mathrm{PF})})$-adaptively secure in the random oracle model, where $\mathcal{L} = (\mathcal{L}_{\mathsf{Setup}}, \mathcal{L}_{\mathsf{KG}}, \mathcal{L}_{\mathsf{Enc}}, \mathcal{L}_{\mathsf{Add}}, \mathcal{L}_{\mathsf{Srch}})$ are defined above.*

Proof. Let $\mathsf{HList}_{\mathsf{sk}}$, $\mathsf{HList}_{\mathsf{ct}}$, $\mathsf{HList}_{\mathsf{tag}}$, and $\mathsf{HList}_{\mathsf{trpdr}}$ be multi-dimensional arrays for random oracles, and we sometimes regard them as sets for convenience; we may write $a \in \mathsf{HList}_{\mathsf{x}}$ for any $\mathsf{x} \in \{\mathsf{sk}, \mathsf{ct}, \mathsf{tag}, \mathsf{trpdr}\}$ if a is stored in somewhere in the list $\mathsf{HList}_{\mathsf{x}}$. We show that S can simulate all transcripts during the execution of all algorithms (except for Dec; it does not need to be simulated since we do not consider chosen ciphertext attacks) and all responses to random oracle queries. First, S can clearly simulate an initial database EDB from $\mathcal{L}_{\mathsf{Setup}}(\mathtt{ctr} \coloneqq 0, \kappa, \mathcal{P}^{(\mathrm{PF})}, \mathcal{U})$ since EDB is an empty list.

For `kg` query $\mathbf{p}$ at `ctr`: In $\mathsf{Exp}^{\mathsf{Real}}_{\mathsf{D},\mathcal{P}^{(\mathrm{PF})}}(\kappa)$, the client computes and returns $\mathsf{sk}_{\mathbf{p}} \coloneqq \mathsf{H}_\kappa(\mathsf{msk} \| p_1 \| \cdots \| p_\ell)$ to D, where $\mathbf{p} = (p_1, p_2, \ldots, p_\ell)$. In $\mathsf{Exp}^{\mathsf{Ideal}}_{\mathsf{D},\mathsf{S},\mathcal{L},\mathcal{P}^{(\mathrm{PF})}}(\kappa)$, S knows $\mathbf{p}$, $\mathsf{TimeF}_{\mathbf{p}}$, and $\mathsf{TimeW}_{\mathbf{p}}$ from $\mathcal{L}_{\mathsf{KG}}(\mathtt{ctr}, \mathbf{p})$. The latter two will be used to respond to random oracle queries and sets $\widehat{\mathcal{F}} \coloneqq \widehat{\mathcal{F}} \cup \mathsf{TimeF}_{\mathbf{p}}$ and $\widehat{\mathcal{W}} \coloneqq \widehat{\mathcal{W}} \cup \mathsf{TimeW}_{\mathbf{p}}$, where $\widehat{\mathcal{F}}$ and $\widehat{\mathcal{W}}$ are sets of leaked files and keywords, respectively. The secret key $\mathsf{sk}_{\mathbf{p}}$ might be already generated (but not returned) via the `enc` or `add` queries. If so, S retrieves $\mathsf{sk}_{\mathbf{p}}$ from $\mathsf{HList}_{\mathsf{sk}}[\mathbf{p}]$ and returns it; otherwise, S randomly chooses and returns $\mathsf{sk}_{\mathbf{p}} \leftarrow \{0,1\}^\kappa$ to D, and stores it in $\mathsf{HList}_{\mathsf{sk}}[\mathbf{p}]$. S adds $(\mathtt{ctr}, \mathbf{p})$ to $\widehat{\mathcal{Q}}_{\mathtt{kg}}$, which is a set of the `kg` queries that S maintains. Since $\mathsf{sk}_{\mathbf{p}}$ is uniformly distributed in the real and ideal environments, it is obvious that both environments are indistinguishable.

For `enc` query $(f_{\mathsf{id}}, \mathcal{A})$ at `ctr`: Let $\mathcal{A} = \{(a_{i,1}, a_{i,2}, \ldots, a_{i,\ell})\}_{i=1}^n$. In $\mathsf{Exp}^{\mathsf{Real}}_{\mathsf{D},\mathcal{P}^{(\mathrm{PF})}}(\kappa)$, the client computes and returns $\mathsf{ct} \coloneqq (c_f, s, \{\mathcal{C}_{\mathcal{A},j}\}_{j=1}^\ell, c_v)$

to D, where $s, r \xleftarrow{\$} \{0,1\}^\kappa$, $c_{i,j} \coloneqq \mathsf{H}_\kappa(\mathsf{H}_\kappa(\mathsf{msk}\|a_{i,1}\|\cdots\|a_{i,j}\|\star\|\cdots\|\star)\|s) \oplus r \in \mathcal{C}_{\mathcal{A},j}$ for $i \in [n]$ and $j \in [\ell]$, $c_f \coloneqq f_{\mathsf{id}} \oplus \mathsf{H}_{|f_{\mathsf{id}}|}(r\|0)$, and $c_v \coloneqq \mathsf{H}_\kappa(r\|1)$. We list important observations to simulate it in $\mathsf{Exp}^{\mathsf{Ideal}}_{\mathsf{D},\mathsf{S},\mathcal{L},\mathcal{P}^{(\mathrm{PF})}}(\kappa)$ as follows.

- A seed $k_{i,j} \coloneqq \mathsf{H}_\kappa(\mathsf{msk}\|a_{i,1}\|\cdots\|a_{i,j}\|\star\|\cdots\|\star)$ of $c_{i,j} = \mathsf{H}_\kappa(k_{i,j}\|s) \oplus r$ is indeed equivalent to $\mathsf{sk}_{\mathbf{p}}$ for $\mathbf{p} = (a_{i,1}\|\cdots\|a_{i,j}\|\star\|\cdots\|\star)$. Hence, S has to consistently maintain $k_{i,j}$ (with the list $\mathsf{HList}_{\mathsf{sk}}$).
- Each of s and r is unique to each ciphertext ct except for negligible probability. Therefore, all hash values $\mathsf{H}_\kappa(k_{i,j}\|s)$ for all $i \in [n]$ and $j \in [\ell]$ are uniquely determined to each ciphertext ct and uniformly distributed. Moreover, c_f $(= f_{\mathsf{id}} \oplus \mathsf{H}_{|f_{\mathsf{id}}|}(r\|0))$ and c_v $(= \mathsf{H}_\kappa(r\|1))$ are also unique to each ct and they are uniformly distributed. They are maintained with $\mathsf{HList}_{\mathsf{ct}}$.
- S has to consistently relate $c_{i,j} \in \mathcal{C}_{\mathcal{A},j}$ to $\mathsf{sk}_{\mathbf{p}}$ so that D can correctly decrypt ct with the secret key (if $\mathsf{P}^{(\mathrm{PF})}_{\mathbf{p}}(\mathcal{A}) = 1$), since $\mathbf{p}$ of $\mathsf{sk}_{\mathbf{p}}$ and $\mathcal{A}$ of ct are public information (as in ordinary ABE [18, 35]), and hence D knows there is a single sub-ciphertext $c_{i,j}$ decryptable by $\mathsf{sk}_{\mathbf{p}}$ in $\mathcal{C}_{\mathcal{A},j}$. (This property is properly handled with random oracle queries.)

Based on the above observations, S simulates ct in $\mathsf{Exp}^{\mathsf{Ideal}}_{\mathsf{D},\mathsf{S},\mathcal{L},\mathcal{P}^{(\mathrm{PF})}}(\kappa)$ as follows. Specifically, we consider two cases depending on whether D has made at least one $\mathtt{kg}$ query $\mathbf{p}$ such that $\mathsf{P}^{(\mathrm{PF})}_{\mathbf{p}}(\mathcal{A}) = 1$.

First, we consider the case where D has never made such a query, i.e., $\mathsf{P}^{(\mathrm{PF})}_{\mathbf{p}}(\mathcal{A}) = 0$ holds for all $(\mathtt{ctr}', \mathbf{p}) \in \widehat{\mathcal{Q}}_{\mathsf{kg}}$. S then knows $|f_{\mathsf{id}}|$ and $\mathcal{A}$ from $\mathcal{L}_{\mathsf{Enc}}(\mathtt{ctr}, f_{\mathsf{id}}, \mathcal{A})$. S simulates ct so that it is decryptable by secret keys $\mathsf{sk}_{\mathbf{p}}$ such that $\mathsf{P}^{(\mathrm{PF})}_{\mathbf{p}}(\mathcal{A}) = 1$ holds. To this end, S randomly chooses $r, c_v \xleftarrow{\$} \{0,1\}^\kappa$ and $c_f \xleftarrow{\$} \{0,1\}^{|f_{\mathsf{id}}|}$, and sets $\mathsf{HList}_{\mathsf{ct}}[r][0] \coloneqq c_f$ and $\mathsf{HList}_{\mathsf{ct}}[r][1] \coloneqq c_v$. S then samples $s \xleftarrow{\$} \{0,1\}^\kappa$, and simulates $c_{i,j} \in \mathcal{C}_{\mathcal{A},j}$ for every $i \in [n]$ and $j \in [\ell]$ as follows. A seed $k_{i,j}$ is stored in $\mathsf{HList}_{\mathsf{sk}}[(a_{i,1}, \ldots, a_{i,j}, \star, \ldots, \star)]$ if S has already generated $\mathsf{sk}_{\mathbf{p}}$ such that $\mathbf{p} = (a_{i,1}, \ldots, a_{i,j}, \star, \ldots, \star)$ (via the $\mathtt{kg}$ query), or previously received the $\mathtt{enc}$ query $(f'_{\mathsf{id}}, \mathcal{A}')$ such that $\mathcal{A}'$ contains $(a_{i,1}, \ldots, a_{i,j}, a'_{i,j+1}, \ldots, a'_{i,\ell})$. In such a case, S retrieves $k_{i,j}$ from $\mathsf{HList}_{\mathsf{sk}}[(a_{i,1}, \ldots, a_{i,j}, \star, \ldots, \star)]$. Otherwise, S randomly chooses $k_{i,j} \xleftarrow{\$} \{0,1\}^\kappa$ and stores it in $\mathsf{HList}_{\mathsf{sk}}[(a_{i,1}, \ldots, a_{i,j}, \star, \ldots, \star)]$. Then, S retrieves $\eta_{i,j}$ from $\mathsf{HList}_{\mathsf{ct}}[s][k_{i,j}]$ if it exists;[5] otherwise, S randomly chooses $\eta_{i,j} \xleftarrow{\$} \{0,1\}^\kappa$ and stores it in $\mathsf{HList}_{\mathsf{ct}}[s][k_{i,j}]$. S

[5] This occurs if $\mathcal{A}$ contains another attribute $(a_{\phi,1}, a_{\phi,2}, \ldots, a_{\phi,\ell})$ such that it holds $(a_{i,1}, a_{i,2}, \ldots, a_{i,j}) = (a_{\phi,1}, a_{\phi,2}, \ldots, a_{\phi,j})$.

sets $c_{i,j} \coloneqq \eta_{i,j} \oplus r$ and adds $c_{i,j}$ to $\mathcal{C}_{\mathcal{A},j}$. Finally, S returns $\mathtt{ct} \coloneqq (c_f, s, \{\mathcal{C}_{\mathcal{A},j}\}_{j=1}^{\ell}, c_v)$. It is obvious that s, r, c_f, c_v, and $\eta_{i,j}$ are uniquely determined and uniformly distributed. In particular, we have $\eta_{i_1,j} = \eta_{i_2,j}$ for any $i_1, i_2 \in [n]$ and $j \in [\ell]$ if it holds $(a_{i_1,1}, \ldots, a_{i_1,j}) = (a_{i_2,1}, \ldots, a_{i_2,j})$. This is consistent with the real environment.

Second, we consider the case where D has made at least one $\mathtt{kg}$ query $\mathbf{p}$ such that $\mathsf{P}_{\mathbf{p}}^{(\mathrm{PF})}(\mathcal{A}) = 1$. Indeed, the simulation can be done in almost the same way as above. The only difference is that S adds $(\mathtt{ctr}, f_{\mathsf{id}})$ to $\widehat{\mathcal{F}}$ since S knows f_{id} and $\mathcal{A}$ from $\mathcal{L}_{\mathsf{Enc}}(\mathtt{ctr}, f_{\mathsf{id}}, \mathcal{A})$.

Thus, the real and ideal environments are indistinguishable.

For `add` query $((\mathsf{id}, w), \mathcal{A})$ at `ctr`: The simulation is done in a similar way to the `enc` query. Let $\mathcal{A} = \{(a_{i,1}, a_{i,2}, \ldots, a_{i,\ell})\}_{i=1}^{n}$. In $\mathsf{Exp}^{\mathsf{Real}}_{\mathsf{D},\mathcal{P}^{(\mathrm{PF})}}(\kappa)$, the clients computes $\mathsf{trans}_1 \coloneqq (\mathsf{id}, t, \{\mathcal{T}_{\mathcal{A},j}\}_{j=1}^{\ell}, \mathsf{tag}_v)$ and sends D it. D stores it in EDB. We list important observations to simulate the transcript in $\mathsf{Exp}^{\mathsf{Ideal}}_{\mathsf{D},\mathsf{S},\mathcal{L},\mathcal{P}^{(\mathrm{PF})}}(\kappa)$ as follows.

- As in the `enc` query, a seed $k_{i,j} \coloneqq \mathsf{H}_\kappa(\mathsf{msk} \| a_{i,1} \| \cdots \| a_{i,j} \| \star \| \cdots \| \star)$ of $\mathsf{tag}_{i,j} = \mathsf{H}_\kappa(\mathsf{H}_\kappa(k_{i,j} \| w) \| t) \oplus \gamma$ is indeed equivalent to $\mathsf{sk}_{\mathbf{p}}$ for $\mathbf{p} = (a_{i,1} \| \cdots \| a_{i,j} \| \star \| \cdots \| \star)$. Hence, S has to consistently maintain $k_{i,j}$ (with the list $\mathsf{HList}_{\mathsf{sk}}$).
- A trapdoor $\mathsf{trpdr}_{i,j} \coloneqq \mathsf{H}_\kappa(k_{i,j} \| w)$ of $\mathsf{tag}_{i,j} = \mathsf{H}_\kappa(\mathsf{trpdr}_{i,j} \| t) \oplus \gamma$ is repeatedly used when searching for w with $\mathsf{sk}_{\mathbf{p}}$ such that $\mathsf{sk}_{\mathbf{p}} = k_{i,j}$. That is, the corresponding tag $\mathsf{tag}_{i,j}$ has to be computed by the same trapdoor $\mathsf{trpdr}_{i,j}$ used for the previous search for w with $\mathsf{sk}_{\mathbf{p}} = k_{i,j}$. Although it makes the simulation complicated, fortunately, the trapdoor $\mathsf{trpdr}_{i,j}$ is not disclosed to D at this point. Therefore, to handle this, S creates $\mathsf{tag}_{i,j}$ by random sampling from $\{0,1\}^\kappa$ but does not create $\mathsf{trpdr}_{i,j}$ at this point, and computes and associates it with the tag later, depending on other queries. S properly handles the above with the list $\mathsf{HList}_{\mathsf{tag}}$.
- Each of t and γ is unique to each entry (id, w) except for negligible probability. Therefore, all hash values $\mathsf{H}_\kappa(\mathsf{trpdr}_{i,j} \| t)$ for all $i \in [n]$ and $j \in [\ell]$ are uniquely determined to each entry $(\mathsf{id}, w))$ and uniformly distributed. Moreover, tag_v $(= \mathsf{H}_\kappa(\gamma))$ is also unique to each added entry (id, w) and uniformly distributed. They are handled with $\mathsf{HList}_{\mathsf{tag}}$.
- S has to consistently relate $\mathsf{tag}_{i,j} \in \mathcal{T}_{\mathcal{A},j}$ to $\mathsf{sk}_{\mathbf{p}}$ so that D can correctly obtain a search result for w with the secret key since D knows which

tag $\mathsf{tag}_{i,j}$ should satisfy the relation $\mathbf{p} = (a_{i,1}, \ldots, a_{i,j}, \star, \ldots, \star)$. (This property is handled with random oracle queries.)

Based on the above observations, S simulates trans_1 in $\mathsf{Exp}^{\mathsf{Ideal}}_{\mathsf{D},\mathsf{S},\mathcal{L},\mathcal{P}^{(\mathrm{PF})}}(\kappa)$ as follows. First, we consider the case where $\mathsf{P}^{(\mathrm{PF})}_{\mathbf{p}}(\mathcal{A}) = 0$ for all $(\cdot, \mathbf{p}) \in \widehat{\mathcal{Q}}_{\mathtt{kg}}$ and all $(\cdot, \cdot, w) \notin \mathcal{Q}_{\mathtt{srch}}$, i.e., D has no secret key $\mathsf{sk}_{\mathbf{p}}$ for $\mathsf{P}^{(\mathrm{PF})}_{\mathbf{p}}(\mathcal{A}) = 1$ and has not been made any $\mathtt{srch}$ query $(\mathbf{p}, w)$. S knows id and $\mathcal{A}$ from $\mathcal{L}_{\mathsf{Add}}(\mathtt{ctr}, (\mathsf{id}, w), \mathcal{A})$. S simulates trans_1 so that D will be able to obtain a correct search result for w under secret keys $\mathsf{sk}_{\mathbf{p}}$ such that $\mathsf{P}^{(\mathrm{PF})}_{\mathbf{p}}(\mathcal{A}) = 1$ holds. To this end, S randomly chooses $t, \gamma, \mathsf{tag}_v \xleftarrow{\$} \{0,1\}^\kappa$ and sets $\mathsf{HList}_{\mathsf{tag}}[t] \coloneqq \gamma$ and $\mathsf{HList}_{\mathsf{tag}}[\gamma] \coloneqq \mathsf{tag}_v$. For all $i \in [n]$ and $j \in [\ell]$, S randomly chooses $\psi_{i,j} \xleftarrow{\$} \{0,1\}^\kappa$. If $\mathsf{HList}_{\mathsf{tag}}[t][(a_{i,1}, \ldots, a_{i,j}, \star, \ldots, \star)]$ is empty, S stores $\psi_{i,j}$ in it and adds $\psi_{i,j} \oplus \gamma$ to $\mathcal{T}_{\mathcal{A},j}$; otherwise, i.e., if a collision between $(a_{i,1}, \ldots, a_{i,j}, a_{i,j+1}, \ldots, a_{i,\ell})$ and $(a_{\phi,1}, \ldots, a_{\phi,j}, a'_{\phi,j+1}, \ldots, a'_{\phi,\ell})$ in $\mathcal{A}$ occurs, S discards $\psi_{i,j}$ (a single $\psi_{\phi,j}$ is sufficient to construct $\mathcal{T}_{\mathcal{A},j}$). Finally, S sends $\mathsf{trans}_1 \coloneqq (\mathsf{id}, t, \{\mathcal{T}_{\mathcal{A},j}\}_{j=1}^{\ell}, \mathsf{tag}_v)$ to D (via the client), and adds $(\mathtt{ctr}, \mathsf{id}, \mathcal{A})$ to $\widehat{\mathcal{Q}}_{\mathtt{add}}$, which is a set of the $\mathtt{add}$ queries that S maintains.[6] D stores it in EDB. It is obvious that t, γ, tag_v, and $\psi_{i,j}$ are uniquely determined and uniformly distributed. In particular, we have $\psi_{i_1,j} = \psi_{i_2,j}$ for any $i_1, i_2 \in [n]$ and $j \in \ell$ if it holds $(a_{i_1,1}, \ldots, a_{i_1,j}) = (a_{i_2,1}, \ldots, a_{i_2,j})$. This is consistent with the real environment.

For the case where there exists $(\cdot, \mathbf{p}) \in \widehat{\mathcal{Q}}_{\mathtt{kg}}$ such that $\mathsf{P}^{(\mathrm{PF})}_{\mathbf{p}}(\mathcal{A}) = 1$, i.e., D already has a secret key $\mathsf{sk}_{\mathbf{p}}$ for $\mathsf{P}^{(\mathrm{PF})}_{\mathbf{p}}(\mathcal{A}) = 1$, or the case where there exists $(\cdot, \cdot, w) \in \mathcal{Q}_{\mathtt{srch}}$, the simulation can be done in the same way as the above. Besides, S adds $(\mathtt{ctr}, w)$ to $\widehat{\mathcal{W}}$ since S knows (id, w) and $\mathcal{A}$ from $\mathcal{L}_{\mathsf{Add}}(\mathtt{ctr}, (\mathsf{id}, w), \mathcal{A})$.

Therefore, the real and ideal environments are indistinguishable.

For $\mathtt{srch}$ query $(\mathbf{p}, q)$ at $\mathtt{ctr}$: In $\mathsf{Exp}^{\mathsf{Real}}_{\mathsf{D},\mathcal{P}^{(\mathrm{PF})}}(\kappa)$, the clients computes and sends $\mathsf{trans}_1 \coloneqq (j, \mathsf{trpdr})$ to D. D fixes $j \in [\ell]$, and does the following for every $(\mathsf{id}, t, \{\mathcal{T}_{\mathcal{A},i}\}_{i=1}^{\ell}, \mathsf{tag}_v) \in \mathsf{EDB}$: for every $\mathsf{tag} \in \mathcal{T}_{\mathcal{A},j}$, D computes $\gamma = \mathsf{H}_\kappa(\mathsf{trpdr} \| t) \oplus \mathsf{tag}$ and adds id to $\mathcal{X}_{q,\mathbf{p}}$ if $\mathsf{H}_\kappa(\gamma) = \mathsf{tag}_v$. D sets $\mathsf{trans}_2 \coloneqq \mathcal{X}_{q,\mathbf{p}}$. We list important observations to simulate $(\mathsf{trans}_1, \mathsf{trans}_2)$ in $\mathsf{Exp}^{\mathsf{Ideal}}_{\mathsf{D},\mathsf{S},\mathcal{L},\mathcal{P}^{(\mathrm{PF})}}(\kappa)$ as follows.

- If q was previously searched under $\mathsf{sk}_{\mathbf{p}}$, S has to use trpdr used at the previous search again. S can handle this issue with the search pattern $\mathsf{SP}_{w,\mathbf{p}}$.

[6] $\widehat{\mathcal{Q}}_{\mathtt{add}}$ is not equivalent to $\mathcal{Q}_{\mathtt{add}} = \{(\mathtt{ctr}_i, (\mathsf{id}_i, w_i), \mathcal{A}_i)\}_{i \in [|\mathcal{Q}_{\mathtt{add}}|]}$.

– If we have $\mathsf{id} \in \mathcal{X}_{q,\mathbf{p}}$, there exist $(\mathsf{id}, t, \{\mathcal{T}_{\mathcal{A},i}\}_{i=1}^{\ell}, \mathsf{tag}_v) \in \mathsf{EDB}$ and a single tag $\mathsf{tag} \in \mathcal{T}_{\mathcal{A},j}$ satisfying $\mathsf{H}_\kappa(\mathsf{tag} \oplus \mathsf{H}_\kappa(\mathsf{trpdr} \| t)) = \mathsf{tag}_v$.
– If we have $\mathsf{id} \notin \mathcal{X}_{q,\mathbf{p}}$ for $(\mathsf{id}, t, \{\mathcal{T}_{\mathcal{A},i}\}_{i=1}^{\ell}, \mathsf{tag}_v) \in \mathsf{EDB}$, there are no tags $\mathsf{tag} \in \mathcal{T}_{\mathcal{A},j}$ satisfying $\mathsf{H}_\kappa(\mathsf{tag} \oplus \mathsf{H}_\kappa(\mathsf{trpdr} \| t)) = \mathsf{tag}_v$.

Based on the above observations, S simulates the transcripts (trans_1, trans_2) in $\mathsf{Exp}^{\mathsf{Ideal}}_{\mathsf{D},\mathsf{S},\mathcal{L},\mathcal{P}^{(\mathrm{PF})}}(\kappa)$ as follows. Here, there are two cases depending on whether D has a secret key $\mathsf{sk}_{\mathbf{p}}$ for $\mathbf{p}$. First, we consider the case where D does not have $\mathsf{sk}_{\mathbf{p}}$. S then knows $\mathbf{p}$, $\mathsf{SP}_{q,\mathbf{p}}$, and $\mathsf{AP}_{q,\mathbf{p}}$ from $\mathcal{L}_{\mathsf{Srch}}(\mathtt{ctr}, \mathbf{p}, q)$. If $\mathsf{SP}_{q,\mathbf{p}} = \emptyset$, the corresponding trapdoor trpdr appear for the first time from D's point of view. Therefore, S randomly chooses $\mathsf{trpdr} \leftarrow \{0,1\}^\kappa$, and stores it to $\mathsf{HList}_{\mathsf{trpdr}}[\mathbf{p}][\mathtt{ctr}]$. Otherwise, i.e., if $\mathsf{SP}_{q,\mathbf{p}} \neq \emptyset$, the trapdoor trpdr should be already generated when the previous search for q under $\mathbf{p}$. Therefore, S retrieves trpdr from $\mathsf{HList}_{\mathsf{trpdr}}[\mathbf{p}][\mathtt{ctr}']$, where some $\mathtt{ctr}' \in \mathsf{SP}_{w,\mathcal{A}}$, and adds trpdr to $\mathsf{HList}_{\mathsf{trpdr}}[\mathbf{p}][\mathtt{ctr}]$ (for the record). Since an appropriate index $j \in [\ell]$ can be identified from $\mathbf{p}$, S can set $\mathsf{trans}_1 := (j, \mathsf{trpdr})$. Receiving trans_1, for every $(\mathsf{id}, t, \{\mathcal{T}_{\mathcal{A},i}\}_{i=1}^{\ell}, \mathsf{tag}_v) \in \mathsf{EDB}$, D does the following procedures. D makes random oracle queries $\mathsf{trpdr} \| t$, and S returns $\psi := \mathsf{tag}_{\mathbf{p}} \oplus \gamma$ to D, where S changes how to choose $\mathsf{tag}_{\mathbf{p}}$ depending on $\mathsf{AP}_{q,\mathbf{p}}$ below.

– If $(\mathtt{ctr}', \mathsf{id}) \in \mathsf{AP}_{q,\mathbf{p}}$, S retrieves $\mathsf{tag}_{\mathbf{p}}$ from $\mathsf{HList}_{\mathsf{tag}}[t][\mathbf{p}]$.
– If $(\mathtt{ctr}', \mathsf{id}) \notin \mathsf{AP}_{q,\mathbf{p}}$, S retrieves a dummy $\mathsf{tag}_{\mathbf{p}}$ from $\mathsf{HList}_{\mathsf{tag}}[t][\mathbf{p}][\mathsf{trpdr}]$. If it is empty, S randomly chooses $\mathsf{tag}_{\mathbf{p}} \xleftarrow{\$} \{0,1\}^\kappa$ and stores it to $\mathsf{HList}_{\mathsf{tag}}[t][\mathbf{p}][\mathsf{trpdr}]$.

For every $\mathsf{tag}_i \in \mathcal{T}_{\mathcal{A},j}$, D makes a random oracle query γ_i, where $\gamma_i := \mathsf{tag}_i \oplus \psi$. If $\mathsf{HList}_{\mathsf{tag}}[\gamma_i]$ is not empty, S retrieves ξ_i; otherwise, S randomly chooses ξ_i and sets $\mathsf{HList}_{\mathsf{tag}}[\gamma_i] := \xi_i$. S returns ξ_i to D.

We explain why the above simulation is perfect with the following four conditions of an arbitrarily fixed entry $(\mathsf{id}, t, \{\mathcal{T}_{\mathcal{A},i}\}_{i=1}^{\ell}, \mathsf{tag}_v) \in \mathsf{EDB}$, where $\mathcal{A} = \{(a_{i,1}, \ldots, a_{i,\ell})\}_{i=1}^{n}$.

(1) $\underline{(\mathtt{ctr}', \mathsf{id}) \in \mathsf{AP}_{q,\mathbf{p}} \text{ and } \mathsf{P}_{\mathbf{p}}^{(\mathrm{PF})}(\mathcal{A}) = 1}$. In this case, id should be added to $\mathcal{X}_{q,\mathbf{p}}$. Recall that each tag $\mathsf{tag}_{i,j} \in \mathcal{T}_{\mathcal{A},j}$ was stored in $\mathsf{HList}_{\mathsf{tag}}[t][(a_{i,1}, \ldots, a_{i,j}, \star, \ldots, \star)]$ at $\mathtt{ctr}'$. As described above, S retrieved $\psi \oplus \gamma$ from $\mathsf{HList}_{\mathsf{tag}}[t][\mathbf{p}]$ and returned ψ to D. Since $\mathsf{P}_{\mathbf{p}}^{(\mathrm{PF})}(\mathcal{A}) = 1$ holds, there exists some $(a_{k,1}, \ldots, a_{k,j}, \star, \ldots, \star)$ such that $\mathbf{p} = (a_{k,1}, \ldots, a_{k,j}, \star, \ldots, \star)$, and hence, we have $\mathsf{HList}_{\mathsf{tag}}[t][\mathbf{p}] = \mathsf{HList}_{\mathsf{tag}}[t][(a_{k,1}, \ldots, a_{k,j}, \star, \ldots, \star)]$. Therefore, S receives γ $(= \mathsf{tag}_{k,j} \oplus \psi)$ and returns $\mathsf{tag}_v = \mathsf{HList}_{\mathsf{tag}}[\gamma]$ to D. Thus, D adds id to $\mathcal{X}_{q,\mathbf{p}}$.

(2) $(\mathtt{ctr}', \mathsf{id}) \in \mathsf{AP}_{q,\mathbf{p}}$ and $\mathsf{P}_{\mathbf{p}}^{(\mathrm{PF})}(\mathcal{A}) = 0$. In this case, id should be added to $\mathcal{X}_{q,\mathbf{p}}$, but the operation should not be done with this entry $(\mathsf{id}, t, \{\mathcal{T}_{\mathcal{A},i}\}_{i=1}^{\ell}, \mathsf{tag}_v)$. There must be another entry $(\mathsf{id}, t', \{\mathcal{T}'_{\mathcal{A}',i}\}_{i=1}^{\ell}, \mathsf{tag}'_v) \in \mathsf{EDB}$ satisfying the condition (1). Indeed, since $\mathsf{P}_{\mathbf{p}}^{(\mathrm{PF})}(\mathcal{A}) = 0$ holds, we have $\mathbf{p} \neq (a_{i,1}, \ldots, a_{i,j}, \star, \ldots, \star)$ for any $i \in [n]$. Therefore, we also have $\mathsf{HList}_{\mathsf{tag}}[t][\mathbf{p}] \neq \mathsf{HList}_{\mathsf{tag}}[t][(a_{i,1}, \ldots, a_{i,j}, \star, \ldots, \star)]$ for any $i \in [n]$. Then, for any $i \in [n]$, it holds $\gamma \neq \mathsf{tag}_{i,j} \oplus \psi$, and hence, D receives $\xi_i = \mathsf{HList}_{\mathsf{tag}}[\mathsf{tag}_{i,j} \oplus \psi]$ such that $\mathsf{tag}_v \neq \xi_i$. D does not add id to $\mathcal{X}_{q,\mathbf{p}}$ (at this point).

(3) $(\mathtt{ctr}', \mathsf{id}) \notin \mathsf{AP}_{q,\mathbf{p}}$ and $\mathsf{P}_{\mathbf{p}}^{(\mathrm{PF})}(\mathcal{A}) = 1$. In this case, id should not be added to $\mathcal{X}_{q,\mathbf{p}}$, though it holds $\mathbf{p} = (a_{k,1}, \ldots, a_{k,j}, \star, \ldots, \star)$ for some $(a_{k,1}, \ldots, a_{k,j}, \star, \ldots, \star)$. This means that the entry $(\mathsf{id}, t, \{\mathcal{T}_{\mathcal{A},i}\}_{i=1}^{\ell}, \mathsf{tag}_v)$ was added by a $\mathtt{add}$ query (id, q') such that $q' \neq q$. Note that S does not know the keywords q and q' themselves but gets to know that they are different. As described above, even if it holds $\mathbf{p} = (a_{k,1}, \ldots, a_{k,j}, \star, \ldots, \star)$, S retrieves the dummy $\mathsf{tag}'_{\mathbf{p}}$ from $\mathsf{HList}_{\mathsf{tag}}[t][\mathbf{p}][\mathsf{trpdr}]$ and sent D $\psi' \coloneqq \mathsf{tag}'_{\mathbf{p}} \oplus \gamma$, not $\psi \coloneqq \mathsf{tag}_{\mathbf{p}} \oplus \gamma$.[7] Thus, for any $i \in [n]$, it holds $\gamma \neq \mathsf{tag}_{i,j} \oplus \psi'$, and hence, D receives $\xi_i = \mathsf{HList}_{\mathsf{tag}}[\mathsf{tag}_{i,j} \oplus \psi']$ such that $\mathsf{tag}_v \neq \xi_i$. D does not add id to $\mathcal{X}_{q,\mathbf{p}}$.

(4) $(\mathtt{ctr}', \mathsf{id}) \notin \mathsf{AP}_{q,\mathbf{p}}$ and $\mathsf{P}_{\mathbf{p}}^{(\mathrm{PF})}(\mathcal{A}) = 0$. We omit the details since it is straightforward from the above discussion.

The same simulation works well for the case where D has $\mathsf{sk}_{\mathbf{p}}$. The difference from the case where D does not have $\mathsf{sk}_{\mathbf{p}}$ is whether or not S obtains the searched keyword q. If D has $\mathsf{sk}_{\mathbf{p}}$, then D may make a random oracle query $\mathsf{sk}_{\mathbf{p}} \| q$, and S has to send the corresponding trapdoor trpdr to D. The leakage of q allows S to correctly respond to the query $\mathsf{sk}_{\mathbf{p}} \| q$. Suppose two leakage functions $\mathcal{L}_{\mathsf{Srch}}(\mathtt{ctr}, \mathbf{p}, q) = (\mathbf{p}, \mathsf{SP}_{q,\mathbf{p}}, \mathsf{AP}_{q,\mathbf{p}})$ and $\mathcal{L}_{\mathsf{Srch}}(\mathtt{ctr}', \mathbf{p}, q') = (\mathbf{p}, \mathsf{SP}_{q',\mathbf{p}}, \mathsf{AP}_{q',\mathbf{p}})$. Therefore, two trapdoors trpdr and trpdr' are used in the simulation of each Search algorithm. Without the knowledge of q, S cannot determine the appropriate trapdoor.

Thus, the real and ideal environments are indistinguishable.

[7] This captures the following situation: In $\mathsf{Exp}^{\mathsf{Real}}_{\mathsf{D},\mathcal{P}^{(\mathrm{PF})}}(\kappa)$, D receives $\mathsf{trpdr} = \mathsf{H}_\kappa(\mathsf{H}_\kappa(\mathsf{msk} \| \mathbf{p}) \| q)$. On the other hand, for the entry $(\mathsf{id}, t, \{\mathcal{T}_{\mathcal{A},i}\}_{i=1}^{\ell}, \mathsf{tag}_v)$, its appropriate trapdoor trpdr' should satisfy $\mathsf{trpdr}' = \mathsf{H}_\kappa(\mathsf{H}_\kappa(\mathsf{msk} \| \mathbf{p}) \| q')$. Since it holds $\mathsf{H}_\kappa(\mathsf{trpdr} \| t) \neq \mathsf{H}_\kappa(\mathsf{trpdr}' \| t)$, S has to use the dummy $\mathsf{tag}'_{\mathbf{p}}$ to make the value of $\mathsf{tag}'_{\mathbf{p}} \oplus \gamma$ different from that of $\mathsf{tag}_{\mathbf{p}} \oplus \gamma$.

Random oracle queries from D: In addition to the above simulations, D may issue the following random oracle queries.[8] Note that S has to correctly respond to the following queries if and only if D already has $\mathsf{sk}_{\mathbf{p}}$ (via the $\mathtt{kg}$ query), since D cannot correctly guess $\mathsf{sk}_{\mathbf{p}}$ except for negligible probability.

- S simulates D's queries to decrypt $\mathsf{ct} = (c_f, s, \mathcal{C}_{\mathcal{A}}, c_v)$ as follows.
  - On receiving a query $\mathsf{sk}_{\mathbf{p}} \| s$ to compute $r = \mathsf{H}_\kappa(\mathsf{sk}_{\mathbf{p}} \| s) \oplus c$, S retrieves η from $\mathsf{HList}_{\mathsf{ct}}[s][\mathsf{sk}_{\mathbf{p}}]$ and returns it to D.
  - On receiving a query $r \| 1$ to check $\mathsf{H}_\kappa(r\|1) = c_v$, S retrieves c_v from $\mathsf{HList}_{\mathsf{ct}}[r][1]$ and returns it to D.
  - On receiving a query $r \| 0$ to compute $\mathsf{H}_{|f_{\mathsf{id}}|}(r\|0)$, S retrieves c_f from $\mathsf{HList}_{\mathsf{ct}}[r][0]$ and returns $k_f \coloneqq c_f \oplus f_{\mathsf{id}}$ to D. Note that S knows $\mathtt{ctr}'$ when r is chosen and hence can find $(\mathtt{ctr}', f_{\mathsf{id}}) \in \widehat{\mathcal{F}}$ since D has made the $\mathtt{kg}$ query $\mathbf{p}$.
- S simulates D's queries to run the server-side search algorithm $\mathsf{Search}_{\mathrm{S}}$ as follows.
  - On receiving a query $\mathsf{sk}_{\mathbf{p}} \| q$ to compute $\mathsf{trpdr} = \mathsf{H}_\kappa(\mathsf{sk}_{\mathbf{p}} \| q)$, S retrieves the appropriate trapdoor trpdr by using $\mathcal{L}_{\mathsf{Srch}}(\mathtt{ctr}, \mathbf{p}, q) = (\mathbf{p}, q, \mathsf{SP}_{q,\mathbf{p}}, \mathsf{SP}_{q,\mathbf{p}})$ and returns it to D. Specifically, S knows when entries related to q and $\mathbf{p}$ are added from $\widehat{\mathcal{W}}$. Let $\{\mathtt{ctr}'_i\}_{i=1}^k$ be counters when the $\mathtt{add}$ queries $\{((\star, q), \mathcal{A}_i)\}_{i=1}^k$ such that $\mathsf{P}_{\mathbf{p}}^{(\mathrm{PF})}(\mathcal{A}) = 1$ are made. If there exists a previous access pattern $\mathsf{AP}_{q,\mathbf{p}} = \{(\mathtt{ctr}''_j, \mathsf{id}_j)\}$, S finds $\mathtt{ctr}''_{i^\star}$ such that $\mathtt{ctr}''_{i^\star} \in \{\mathtt{ctr}'_1, \ldots, \mathtt{ctr}'_k\}$, which means that an entry $(\mathsf{id}_{i^\star}, q)$ with $\mathcal{A}_{i^\star}$ was added at $\mathtt{ctr}''_{i_\star}$. In other words, the trapdoor for q and $\mathbf{p}$ was made and used in the search. Therefore, S retrieves the latest counter $\mathtt{ctr}^\star$ from $\mathsf{SP}_{q,\mathbf{p}}$, and returns $\mathsf{trpdr} = \mathsf{HList}_{\mathsf{trpdr}}[\mathbf{p}][\mathtt{ctr}^\star]$.
  - On receiving a query $\mathsf{trpdr} \| t$ to compute $\mathsf{H}_\kappa(\mathsf{trpdr}\|t)$, S finds the corresponding predicate $\mathbf{p}$ and randomness γ. S then retrieves tag and returns $\psi \coloneqq \mathsf{tag} \oplus \gamma$ as in the above simulation of the Search algorithm.
  - On receiving a query γ to check $\mathsf{H}_\kappa(\gamma) = \mathsf{tag}_v$, S retrieves and returns ξ_i from $\mathsf{HList}_{\mathsf{tag}}[\gamma_i]$ as in the simulation of the Search algorithm.

It completes the proof. □

[8] We implicitly assume S can identify what kind of random oracle queries are issued. This can be easily done by appending a label lab to the prefix of the input of the random oracle, e.g., $\mathsf{tag}_v \coloneqq \mathsf{H}_\kappa(\mathsf{lab}\|\gamma)$.

Table 1: Statistical information about datasets used in our experiments

#files	#keywords	#entries
300	10,117	49,764
600	14,670	102,940
1,000	18,238	163,772

4 Experiments

4.1 Experimental Settings

Implementation. We give a performance evaluation of the proposed scheme by Python software implementation. These experiments were done in an MacBook Air (M1, 2020) with 8-core CPU, 16GB RAM. We used SHA-256 implemented with Python's library hashlib as the hash function. Note that both the client and server run locally, and communication costs are not taken into account.

Dataset. We used a part of the Enron Email dataset [41] (May 7, 2015 version), commonly and widely used for SSE experiments. As a preprocessing, after removing header information, symbols, and URL information from the dataset, we applied the Porter stemming algorithm [34] from the NLTK (Natural Language ToolKit) library to them. Table 1 shows the statistics about the number of files, the number of unique keywords, and the number of entries (i.e., (w, id) pairs).

Parameter Setting. We used an integer list to implement the predicates and attributes. In order to compare the performance of different attributes, we experimented with the following two attributes ($\ell = 3$).

- ***Attr*1**: $\mathcal{A}_1 = (a_1, a_2, a_3)$. It has only one attribute, and the corresponding predicates (i.e., tags) are (a_1, a_2, a_3), $(a_1, a_2, \star)$, $(a_1, \star, \star)$, and $(\star, \star, \star)$.
- ***Attr*2**: $\mathcal{A}_2 = ((a_1, a_2, a_3), (a_1, a_4, a_5), (a_1, a_6, a_7), (a_1, a_6, a_8))$. It has four attributes, and the corresponding predicates (i.e., tags) are nine: there are (a_1, a_4, a_5), $(a_1, a_4, \star)$, (a_1, a_6, a_7), (a_1, a_6, a_8), and $(a_1, a_6, \star)$ in addition to the above four predicates.

Also, the predicate of the user who runs Search and Dec is $\mathbf{p} = (a_1, a_2, \star)$. That is, Search (or Dec) can be run on documents added (or encrypted) with either attribute ***Attr*1** or ***Attr*2**.

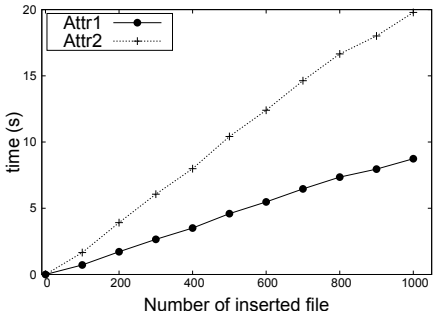

Fig. 1: Addition cost.

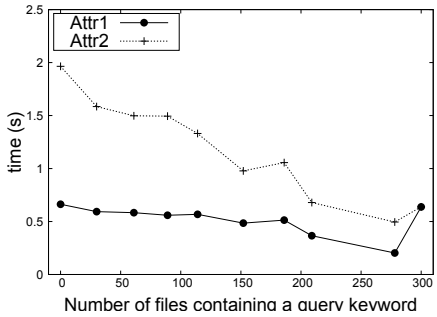

Fig. 2: Search cost.

4.2 Experimental Results

Addition Cost. The Add algorithm computes $\mathcal{O}(n \cdot \ell \cdot d)$ elements for n attribute vectors whose degree is at most ℓ and a file that contains d distinct keywords. We give the addition costs in Fig. 1. In both attributes ***Attr*1** and ***Attr*2**, the addition cost increases linearly with number of inserted files. Specifically, ***Attr*1** and ***Attr*2** take less than 0.9 s and 2.0 s, respectively, to add 1,000 files.

Search Cost. The Search algorithm requires a kind of exhaustive search, so it seems inefficient in the asymptotic sense. Fig. 2 shows the experimental results on search costs for the proposed scheme, which provides reasonably efficient search costs. Specifically, in the case where queried keywords are contained in more than 200 files, both the Search algorithm for ***Attr*1** or ***Attr*2** requires less than a second. The reason why the search time decreases as the size of the search result is due to the search procedure; if the algorithm finds an entry about the queried keyword q and a file f_{id}, it adds id to the search result and steps into the search for the next file $f_{\mathsf{id}'}$. Therefore, the more files contain the queried keyword, the less search time we have.

Encryption/Decryption Cost. The Enc algorithm computes $\mathcal{O}(n \cdot \ell)$ elements for n attribute vectors whose degree is at most ℓ to encrypt a single file. The Dec algorithm needs to find a sub-ciphertext that contains a randomness, which is used to decrypt the corresponding encrypted file. Therefore, Dec requires $\mathcal{O}(n \cdot \ell)$ computational cost. As can be seen in Figs. 3 and 4, the proposed scheme has reasonable encryption and decryption costs. Specifically, ***Attr*1** and ***Attr*2** take less than 0.63 s and 0.74 s, respectively, to encryption 1,000 files. On the other hand, both ***Attr*1** and ***Attr*2** take less than 0.45 s to decryption 1,000 files.

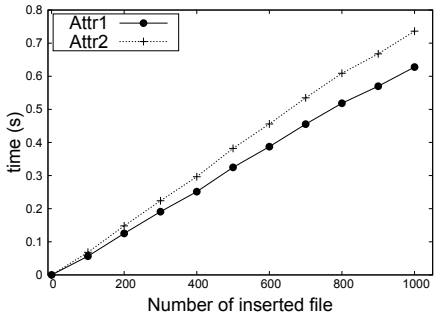

Fig. 3: Encryption cost

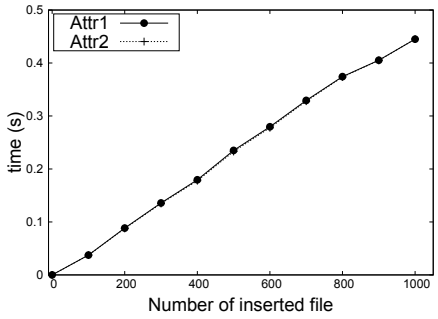

Fig. 4: Decryption cost.

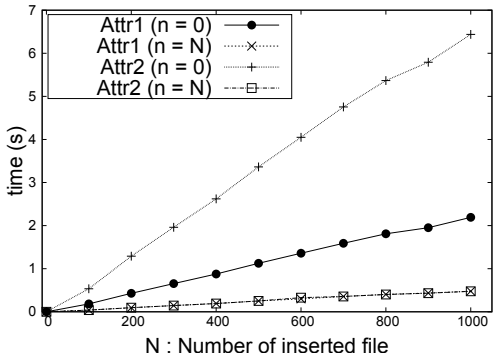

Fig. 5: Search cost with DB size.

Search Cost Associated with DB Size. We show in Fig. 5 the search cost for scaling the number of files to be added from 100 to 1,000. We chose keywords so that the search results are empty (i.e., worst case ($n = 0$)) and all files are in the search results ($n = N$). Surprisingly, in the case where many files are search results ($n = N$), the proposed scheme is reasonably fast, even with a large DB size. Specifically, both ***Attr*1** and ***Attr*2** take less than 0.48 s to search a keyword that 1,000 files contain.

5 Concluding Remarks

We introduced a new model of dynamic multi-user symmetric searchable encryption (MUSE) and its security notion, and showed a concrete dynamic MUSE scheme for prefix-fixing predicates, which are employed in the context of pseudorandom functions. Our dynamic MUSE scheme is constructed from only symmetric-key primitives and is secure in the random oracle model. The experimental results showed that our scheme

achieved a good trade-off among flexibility of access control, security levels, and efficiency.

The proposed dynamic MUSE scheme only supports addition operations; we leave how to support deletion operations as an open problem. It would also be interesting to define and achieve forward and backward privacy for dynamic MUSE.

References

1. Aljabri, J., Michala, A.L., Singer, J.: ELSA: A keyword-based searchable encryption for cloud-edge assisted industrial internet of things. In: IEEE International Symposium on Cluster, Cloud and Internet Computing (CCGrid 2022). pp. 259–268 (2022)
2. Aljabri, J., Michala, A.L., Singer, J.: ELSA: Edge lightweight searchable attribute-based encryption multi-keyword scalability. In: IEEE Conference on Dependable and Secure Computing (DSC 2022). pp. 1–4 (2022)
3. Bag, A., Patranabis, S., Mukhopadhyay, D.: Tokenised multi-client provisioning for dynamic searchable encryption with forward and backward privacy. In: ACM Symposium on Information, Computer and Communications Security, ASIACCS 2024. p. 1691–1707. Association for Computing Machinery (2024)
4. Bakas, A., Dang, H.V., Michalas, A., Zalitko, A.: The cloud we share: Access control on symmetrically encrypted data in untrusted clouds. IEEE Access **8**, 210462–210477 (2020)
5. Boneh, D., Di Crescenzo, G., Ostrovsky, R., Persiano, G.: Public key encryption with keyword search. In: Cachin, C., Camenisch, J. (eds.) Advances in Cryptology – EUROCRYPT 2004. Lecture Notes in Computer Science, vol. 3027, pp. 506–522. Springer Berlin Heidelberg (2004)
6. Boneh, D., Franklin, M.: Identity-based encryption from the Weil pairing. In: Kilian, J. (ed.) Advances in Cryptology – CRYPTO 2001. vol. 2139, pp. 213–229. Springer Berlin Heidelberg (2001)
7. Boneh, D., Waters, B.: Constrained pseudorandom functions and their applications. In: Sako, K., Sarkar, P. (eds.) Advances in Cryptology – ASIACRYPT 2013, Part II. pp. 280–300. Springer Berlin Heidelberg, Berlin, Heidelberg (2013)
8. Bost, R., Minaud, B., Ohrimenko, O.: Forward and backward private searchable encryption from constrained cryptographic primitives. In: ACM SIGSAC Conference on Computer and Communications Security, CCS 2017. pp. 1465–1482. ACM, New York, NY, USA (2017)
9. Boyle, E., Goldwasser, S., Ivan, I.: Functional signatures and pseudorandom functions. In: Krawczyk, H. (ed.) Public Key Cryptography – PKC 2014. vol. 8383, pp. 501–519. Springer Berlin Heidelberg (2014)
10. Cash, D., Grubbs, P., Perry, J., Ristenpart, T.: Leakage-abuse attacks against searchable encryption. In: ACM SIGSAC Conference on Computer and Communications Security, CCS 2015. pp. 668–679. ACM, New York, NY, USA (2015)
11. Cash, D., Jaeger, J., Jarecki, S., Jutla, C., Krawczyk, H., Roşu, M.C., Steiner, M.: Dynamic searchable encryption in very-large databases: Data structures and implementation. In: Network and Distributed System Security Symposium, NDSS 2014. The Internet Society (2014)

12. Chamani, J.G., Papadopoulos, D., Papamanthou, C., Jalili, R.: New constructions for forward and backward private symmetric searchable encryption. In: ACM SIGSAC Conference on Computer and Communications Security, CCS 2018. pp. 1038–1055. ACM, New York, NY, USA (2018)
13. Chamani, J.G., Wang, Y., Papadopoulos, D., Zhang, M., Jalili, R.: Multi-user dynamic searchable symmetric encryption with corrupted participants. IEEE Transactions on Dependable and Secure Computing **20**(1), 114–130 (2023)
14. Curtmola, R., Garay, J., Kamara, S., Ostrovsky, R.: Searchable symmetric encryption: Improved definitions and efficient constructions. In: ACM SIGSAC Conference on Computer and Communications Security, CCS 2006. pp. 79–88. ACM, New York, NY, USA (2006)
15. Curtmola, R., Garay, J.A., Kamara, S., Ostrovsky, R.: Searchable symmetric encryption: Improved definitions and efficient constructions. Journal of Computer Security **19**(5), 895–934 (2011)
16. Dang, H.V., Ullah, A., Bakas, A., Michalas, A.: Attribute-based symmetric searchable encryption. In: Zhou, J., Conti, M., Ahmed, C.M., Au, M.H., Batina, L., Li, Z., Lin, J., Losiouk, E., Luo, B., Majumdar, S., Meng, W., Ochoa, M., Picek, S., Portokalidis, G., Wang, C., Zhang, K. (eds.) Applied Cryptography and Network Security Workshops 2020. pp. 318–336. Springer International Publishing, Cham (2020)
17. Demertzis, I., Chamani, J.G., Papadopoulos, D., Papamanthou, C.: Dynamic searchable encryption with small client storage. In: Network and Distributed System Security Symposium, NDSS 2020. The Internet Society (2020)
18. Goyal, V., Pandey, O., Sahai, A., Waters, B.: Attribute-based encryption for fine-grained access control of encrypted data. In: ACM SIGSAC Conference on Computer and Communications Security, CCS 2006. pp. 89–98. ACM, New York, NY, USA (2006)
19. Hahn, F., Kerschbaum, F.: Searchable encryption with secure and efficient updates. In: ACM SIGSAC Conference on Computer and Communications Security, CCS 2014. pp. 310–320. ACM, New York, NY, USA (2014)
20. Hattori, M., Hirano, T., Ito, T., Matsuda, N., Mori, T., Sakai, Y., Ohta, K.: Ciphertext-policy delegatable hidden vector encryption and its application to searchable encryption in multi-user setting. In: Chen, L. (ed.) Cryptography and Coding. pp. 190–209. Springer Berlin Heidelberg, Berlin, Heidelberg (2011)
21. Jarecki, S., Jutla, C., Krawczyk, H., Rosu, M., Steiner, M.: Outsourced symmetric private information retrieval. In: ACM SIGSAC Conference on Computer and Communications Security, CCS 2013. p. 875–888. ACM, New York, NY, USA (2013)
22. Kamara, S., Papamanthou, C.: Parallel and dynamic searchable symmetric encryption. In: Sadeghi, A.R. (ed.) Financial Cryptography and Data Security, FC 2013. pp. 258–274. Springer Berlin Heidelberg, Berlin, Heidelberg (2013)
23. Kamara, S., Papamanthou, C., Roeder, T.: Dynamic searchable symmetric encryption. In: ACM SIGSAC Conference on Computer and Communications Security, CCS 2012. pp. 965–976. ACM, New York, NY, USA (2012)
24. Kermanshahi, S.K., Liu, J.K., Steinfeld, R., Nepal, S., Lai, S., Loh, R., Zuo, C.: Multi-client cloud-based symmetric searchable encryption. IEEE Transactions on Dependable and Secure Computing **18**(5), 2419–2437 (2021)
25. Kiayias, A., Papadopoulos, S., Triandopoulos, N., Zacharias, T.: Delegatable pseudorandom functions and applications. In: ACM SIGSAC Conference on Computer and Communications Security, CCS 2007. pp. 669–684. CCS '13, ACM, New York, NY, USA (2013)

26. Li, J., Huang, Y., Wei, Y., Lv, S., Liu, Z., Dong, C., Lou, W.: Searchable symmetric encryption with forward search privacy. IEEE Transactions on Dependable and Secure Computing **18**(1), 460–474 (Jan 2021)
27. Meng, L., Chen, L., Tian, Y., Manulis, M., Liu, S.: FEASE: fast and expressive asymmetric searchable encryption. In: Balzarotti, D., Xu, W. (eds.) USENIX Security 2024. USENIX Association (2024)
28. Meng, R., Zhou, Y., Ning, J., Liang, K., Han, J., Susilo, W.: An efficient key-policy attribute-based searchable encryption in prime-order groups. In: Okamoto, T., Yu, Y., Au, M.H., Li, Y. (eds.) Provable Security, ProvSec 2017. pp. 39–56. Springer International Publishing, Cham (2017)
29. Michalas, A.: The lord of the shares: Combining attribute-based encryption and searchable encryption for flexible data sharing. In: ACM/SIGAPP Symposium on Applied Computing, SAC 2019. p. 146–155. ACM, New York, NY, USA (2019)
30. Michalas, A., Bakas, A., Dang, H.V., Zalitko, A.: MicroSCOPE: Enabling access control in searchable encryption with the use of attribute-based encryption and SGX. In: Askarov, A., Hansen, R.R., Rafnsson, W. (eds.) Secure IT Systems, NordSec 2019. pp. 254–270. Springer International Publishing, Cham (2019)
31. Miers, I., Mohassel, P.: IO-DSSE: scaling dynamic searchable encryption to millions of indexes by improving locality. In: Network and Distributed System Security Symposium, NDSS 2017 (2017)
32. Naveed, M., Prabhakaran, M., Gunter, C.: Dynamic searchable encryption via blind storage. In: IEEE Symposium on Security and Privacy, S&P 2014. pp. 639–654 (May 2014)
33. Niu, S., Hu, Y., Zhou, S., Shao, H., Wang, C.: Attribute-based searchable encryption in edge computing for lightweight devices. IEEE Systems Journal **17**(3), 3503–3514 (2023)
34. Porter, M.F.: An algorithm for suffix stripping. Program **14**(3), 130–137 (1980)
35. Sahai, A., Waters, B.: Fuzzy identity-based encryption. In: Cramer, R. (ed.) Advances in Cryptology – EUROCRYPT 2005. Lecture Notes in Computer Science, vol. 3494, pp. 457–473. Springer Berlin Heidelberg (2005)
36. Song, D.X., Wagner, D., Perrig, A.: Practical techniques for searches on encrypted data. In: IEEE Symposium on Security and Privacy, S&P 2000. pp. 44–55 (2000)
37. Stefanov, E., Papamanthou, C., Shi, E.: Practical dynamic searchable encryption with small leakage. In: Network and Distributed System Security Symposium, NDSS 2014. The Internet Society (2014)
38. Sun, S.F., Liu, J.K., Sakzad, A., Steinfeld, R., Yuen, T.H.: An efficient non-interactive multi-client searchable encryption with support for boolean queries. In: Askoxylakis, I., Ioannidis, S., Katsikas, S., Meadows, C. (eds.) Computer Security – ESORICS 2016. pp. 154–172. Springer International Publishing, Cham (2016)
39. Sun, S.F., Zuo, C., Liu, J.K., Sakzad, A., Steinfeld, R., Yuen, T.H., Yuan, X., Gu, D.: Non-interactive multi-client searchable encryption: Realization and implementation. IEEE Transactions on Dependable and Secure Computing **19**(1), 452–467 (2022)
40. Sun, S., Steinfeld, R., Lai, S., Yuan, X., Sakzad, A., Liu, J.K., Nepal, S., Gu, D.: Practical non-interactive searchable encryption with forward and backward privacy. In: Network and Distributed System Security Symposium, NDSS 2021. The Internet Society (2021)
41. The CALO Project: Enron email dataset (may 7, 2015 version). `https://www.cs.cmu.edu/~enron/` (2015), `https://www.cs.cmu.edu/~./enron/`

Experiment: $\mathsf{Exp}^{\mathsf{Corr}}_{\mathsf{A},\mathcal{P}}(\kappa)$

$(\mathsf{msk}, \mathsf{EDB}) \leftarrow \mathsf{Setup}(1^\kappa, \mathcal{P})$
$(f^\star, \mathcal{A}^\star, q^\star, \mathsf{P}^\star) \leftarrow \mathsf{A}^{\mathsf{O}_{\mathtt{kg}}(\cdot),\mathsf{O}_{\mathsf{add}}(\cdot,\cdot),\mathsf{O}_{\mathtt{srch}}(\cdot,\cdot)}(\mathsf{msk}, \mathsf{EDB}^{(0)})$
$\mathsf{ct}^\star \leftarrow \mathsf{Enc}(\mathsf{msk}, f^\star, \mathcal{A}^\star)$
$(\mathcal{X}_{q^\star,\mathsf{P}^\star}; \mathsf{EDB}') \leftarrow \mathsf{Search}(\mathsf{SKList}[\mathsf{P}^\star], q^\star; \mathsf{EDB})$
if $(\mathsf{Dec}(\mathsf{SKList}[\mathsf{P}^\star], \mathsf{ct}^\star) \neq f^\star) \vee (\mathcal{X}_{q^\star,\mathsf{P}^\star} \neq \mathsf{ID}_{q^\star,\mathsf{P}^\star})$ **then**
  return 1
else
  return 0

Fig. 6: A correctness game. $\mathsf{O}_{\mathtt{kg}}$ takes a predicate $\mathsf{P} \in \mathcal{P}$ as input, returns $\mathsf{sk}_\mathsf{P} \leftarrow \mathsf{KeyGen}(\mathsf{msk}, \mathsf{P})$ to A, and adds sk_P to $\mathsf{SKList}[\mathsf{P}]$. $\mathsf{O}_{\mathsf{add}}$ takes $(\mathsf{id}, w) \in \{0,1\}^\ell \times \Lambda$ and $\mathcal{A} \in \mathcal{U}$ as input, runs $\langle(\mathsf{ack}; \mathsf{EDB}'), \mathsf{trans}\rangle \leftarrow \mathsf{Add}(\mathsf{msk}, (\mathsf{id}, w), \mathcal{A}; \mathsf{EDB})$, and returns $(\mathsf{ack}, \mathsf{trans})$. $\mathsf{O}_{\mathtt{srch}}$ takes $\mathsf{P} \in \mathcal{P}$ and $q \in \Lambda$ as input, runs $\langle(\mathcal{X}_{q,\mathsf{P}}; \mathsf{EDB}'), \mathsf{trans}\rangle \leftarrow \mathsf{Search}(\mathsf{SKList}[\mathsf{P}], q; \mathsf{EDB})$, and returns $(\mathcal{X}_{q,\mathsf{P}}, \mathsf{trans})$.

42. Wang, J., Chow, S.S.M.: Omnes pro uno: Practical multi-writer encrypted database. In: Butler, K.R.B., Thomas, K. (eds.) USENIX Security 2022. pp. 2371–2388. USENIX Association (2022)
43. Wang, J., Chow, S.S.M.: Unus pro omnibus: Multi-client searchable encryption via access control. In: Network and Distributed System Security Symposium, NDSS 2024. The Internet Society (2024)
44. Xu, L., Xu, C., Liu, J.K., Zuo, C., Zhang, P.: A multi-client dynamic searchable symmetric encryption system with physical deletion. In: Qing, S., Mitchell, C., Chen, L., Liu, D. (eds.) Information and Communications Security, ICICS 2018. pp. 516–528. Springer International Publishing, Cham (2018)
45. Yang, J., Liu, F., Luo, X., Hong, J., Li, J., Xue, K.: Forward private multi-client searchable encryption with efficient access control in cloud storage. In: GLOBECOM 2022. pp. 3791–3796 (2022)
46. Yavuz, A.A., Guajardo, J.: Dynamic searchable symmetric encryption with minimal leakage and efficient updates on commodity hardware. In: Dunkelman, O., Keliher, L. (eds.) Selected Areas in Cryptography – SAC 2015. pp. 241–259. Springer International Publishing, Cham (2016)
47. Yin, H., Zhang, W., Deng, H., Qin, Z., Li, K.: An attribute-based searchable encryption scheme for cloud-assisted iiot. IEEE Internet of Things Journal **10**(12), 11014–11023 (2023)
48. Zhang, Y., Zhu, T., Guo, R., Xu, S., Cui, H., Cao, J.: Multi-keyword searchable and verifiable attribute-based encryption over cloud data. IEEE Transactions on Cloud Computing **11**(1), 971–983 (2023)

A The Formal Definition of Correctness of Dynamic MUSE

Formally, following Cash et al.'s work [11], we consider an experiment in Fig. 6 and define the correctness of dynamic MUSE as follows.

Definition 3 (Correctness). *A dynamic MUSE scheme Σ is said to be correct if for any PPT algorithm* A*, it holds* $\Pr\left[\mathsf{Exp}^{\mathsf{Corr}}_{\mathsf{A},\mathcal{P}}(\kappa)=1\right] < \mathsf{negl}(\kappa)$.

B Proof of Theorem 1

Let $\mathcal{A}=\{(a_{i,1},a_{i,2},\ldots,a_{i,\ell})\}_{i=1}^{n}$. First, We show that a correctly-encrypted ciphertext $\mathsf{ct}=(c_f,s,\{\mathcal{C}_{\mathcal{A},j}\}_{j=1}^{\ell},c_v)$ can be always decrypted with $\mathsf{sk}_{\mathbf{p}}$ correctly if $\mathsf{P}^{(\mathrm{PF})}_{\mathbf{p}}(\mathcal{A})=1$ holds, which implies that there exists $(a_{i,1},\ldots,a_{i,j},\star,\ldots,\star)$ such that $\mathbf{p}=(a_{i,1},\ldots,a_{i,j},\star,\ldots,\star)$. Since $c_f=f_{\mathsf{id}}\oplus \mathsf{H}_{|f_{\mathsf{id}}|}(r\|0)$, there exists a sub-ciphertext $c_{i,j}=r\oplus\mathsf{H}_\kappa(\mathsf{H}_\kappa(\mathsf{msk}\|(a_{i,1},\ldots,a_{i,j},\star,\ldots,\star))\|s)=r\oplus\mathsf{H}_\kappa(\mathsf{H}_\kappa(\mathsf{msk}\|\mathbf{p})\|s)$. It is obvious that the user who has $\mathsf{sk}_{\mathbf{p}}$ $(=\mathsf{H}_\kappa(\mathsf{msk}\|\mathbf{p}))$ can obtain r by computing $c_{i,j}\oplus\mathsf{H}_\kappa(\mathsf{sk}_{\mathbf{p}}\|s)$, since the outputs of random oracles with different input never collide with each other.

We next show that when the Search algorithm is executed with a secret key $\mathsf{sk}_{\mathbf{p}}$ for any prefix-fixing predicate $\mathsf{P}^{(\mathrm{PF})}_{\mathbf{p}}\in\mathcal{P}^{(\mathrm{PF})}$ and any keyword $q\in\Lambda$, it always outputs a correct search result $\mathcal{X}_{q,\mathbf{p}}=\mathsf{ID}_{q,\mathsf{P}^{(\mathrm{PF})}_{\mathbf{p}}}$. We consider four cases for arbitrarily fixed entry $(\mathsf{id},s,\{\mathcal{T}_{\mathcal{A},j}\}_{j=1}^{\ell},\mathsf{tag}_v)\in\mathsf{EDB}$. First, suppose that the entry was generated by $\mathsf{Add}(\mathsf{msk},(\mathsf{id},q),\mathcal{A};\mathsf{EDB})$ such that $\mathsf{P}^{(\mathrm{PF})}_{\mathbf{p}}(\mathcal{A})=1$. In such a case, the search procedure always adds id to $\mathcal{X}_{q,\mathbf{p}}$. We omit the detail since it can be proved similarly to the decryption correctness shown above. Second, suppose that the entry was generated by $\mathsf{Add}(\mathsf{msk},(\mathsf{id},q),\mathcal{A};\mathsf{EDB})$ such that $\mathsf{P}^{(\mathrm{PF})}_{\mathbf{p}}(\mathcal{A})=0$. In this case, it clearly holds $\mathbf{p}\neq(a_{i,1},\ldots,a_{i,j},\star,\ldots,\star)$ for any $i\in[n]$ and $j\in[\ell]$, and hence, we have $\mathsf{H}_\kappa(\mathsf{sk}_{\mathbf{p}}\|q)\neq\mathsf{H}_\kappa(\mathsf{H}_\kappa(\mathsf{msk}\|(a_{i,1},\ldots,a_{i,j},\star,\ldots,\star))\|q)$. Therefore, the server-side search algorithm $\mathsf{Search}_{\mathrm{S}}$ obtains γ_i such that $\gamma_i\neq\gamma$ for any $i\in[n]$, where γ is randomness such that $\mathsf{H}_\kappa(\gamma)=\mathsf{tag}_v$. Thus, $\mathsf{Search}_{\mathrm{S}}$ does not add id to $\mathcal{X}_{q,\mathbf{p}}$. Third, suppose that the entry was generated by $\mathsf{Add}(\mathsf{msk},(\mathsf{id},w),\mathcal{A};\mathsf{EDB})$ such that $\mathsf{P}^{(\mathrm{PF})}_{\mathbf{p}}(\mathcal{A})=1$. In this case, although there exists $(a_{i,1},\ldots,a_{i,j},\star,\ldots,\star)$ such that $\mathbf{p}=(a_{i,1},\ldots,a_{i,j},\star,\ldots,\star)$, we have $\mathsf{H}_\kappa(\mathsf{sk}_{\mathbf{p}}\|q)\neq\mathsf{H}_\kappa(\mathsf{sk}_{\mathbf{p}}\|w)$ due to the property of random oracles. Since it clearly holds $\mathsf{H}_\kappa(\mathsf{H}_\kappa(\mathsf{sk}_{\mathbf{p}}\|q)\|t)\neq\mathsf{H}_\kappa(\mathsf{H}_\kappa(\mathsf{sk}_{\mathbf{p}}\|w)\|t)$, $\mathsf{Search}_{\mathrm{S}}$ only obtains γ_i such that $\gamma_i\neq\gamma$ for any $i\in[n]$, where γ is randomness such that $\mathsf{H}_\kappa(\gamma)=\mathsf{tag}_v$. Thus, $\mathsf{Search}_{\mathrm{S}}$ does not add id to $\mathcal{X}_{q,\mathbf{p}}$. We omit the proof details for the fourth case where the entry was generated by $\mathsf{Add}(\mathsf{msk},(\mathsf{id},w),\mathcal{A};\mathsf{EDB})$ such that $\mathsf{P}^{(\mathrm{PF})}_{\mathbf{p}}(\mathcal{A})=0$ since it can be proved similarly. □