Unus pro omnibus: Multi-Client Searchable Encryption via Access Control

Jiafan Wang
Data61, CSIRO
Sydney, NSW, Australia
jiafan.wang@data61.csiro.au

Sherman S. M. Chow
Department of Information Engineering
The Chinese University of Hong Kong
Shatin, N.T., Hong Kong

***Abstract*—Searchable encryption lets an untrusted cloud server store keyword-document tuples encrypted by *writers* and conduct keyword searches with tokens from *readers*. Multi-writer schemes naturally offer broad applicability; however, it is unclear how to achieve the distinctive features of single-writer systems, namely, *optimal search* traversing only the result set and *forward privacy* invalidating old search tokens against any new data. Cutting-edge results by Wang and Chow (Usenix Security 2022) incur extra traversal over existing keywords and weaken forward privacy that only invalidates previous-issued search tokens periodically.**

We propose delegatable searchable encryption (DSE) with optimal search time for the multi-writer multi-reader setting. Beyond forward privacy, DSE supports security measures countering new integrity threats by malicious clients and keyword-guessing attacks inherent to public-key schemes. These are simultaneously made conceivable via one-time delegations of updating and/or searching power from the data owner and our tailored notion of shiftable multi-recipient counter encryption. DSE also benefits from the hybrid searchable encryption idea of Wang and Chow but at a microscopic level. Our evaluation confirms the order-of-magnitude improvement in search time over real-world datasets.

I. INTRODUCTION

Secure multiparty collaborative applications, often cloud-based, enable secure and managed collaboration on projects by mutually distrustful entities with different roles. Managing personnel, or the project owner, establishes a project folder and grants specific access rights to authorized users, ensuring only they can read and/or write to files in the folder. Collaboration can be from software development teams on a complex codebase or managerial/compliance staff reviewing financial records. In the first example, programmers get both read and write privileges, allowing them to contribute source code and review logs. On the other hand, auditors may have read-only access but can write logs after reviewing. Access control policies can be granular, say, based on keywords.

Sherman Chow (corresponding, 0000-0001-7306-453X) is supported by GRF 14210217, 14209918, and 14210621 from RGC. Jiafan Wang contributed to this work mostly while at CUHK. We are grateful to Russell W. F. Lai for his help during the early stages, Alexandra Boldyreva for her comments on a related Ph.D. thesis and kind invitation to the "Encryption for Secure Search and other Algorithms" workshop, and the anonymous reviewers of NDSS.

Network and Distributed System Security (NDSS) Symposium 2024
26 February - 1 March 2024, San Diego, CA, USA
ISBN 1-891562-93-2
https://dx.doi.org/10.14722/ndss.2024.23288
www.ndss-symposium.org

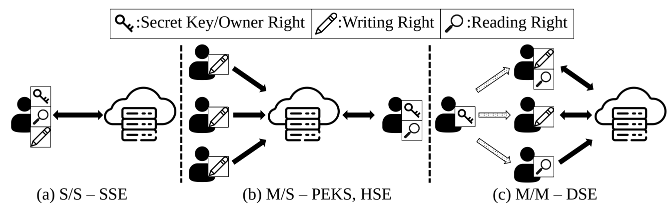


Fig. 1: Architectures of Searchable Encryption

Storage servers are often untrustworthy, especially amidst recurrent data breach incidents. Encryption is thus vital, but it precludes any operations such as indexing. Searchable symmetric encryption (SSE) [32], [16], [24] has then emerged. It is designed to cater to a single client as the sole writer (for updating) and reader (for searching), who uses the secret key to generate tokens. The storage server helps search or update specific keywords with these tokens. To ensure forward privacy [33], [8], [27], *i.e.*, old search tokens cannot compromise the privacy of future updates, these tokens are generated differently depending on prior searches and updates or the updating state of the encrypted database in general. Directly applying SSE to multi-client scenarios requires synchronizing the distribution of different versions of tokens, and hence frequent interactions among the clients, let alone the lack of resilience against threats caused by corrupt clients. Despite its single-client limitations, SSE achieves optimal search with symmetric-key operations linear in the search result size.

Multi-writer searchable encryption is typically realized by public-key encryption with keyword search (PEKS) [6]. Anyone who knows the public key of the data owner can *independently* generate PEKS ciphertexts. As a result, there is *no* global structure indexing them. Indeed, the syntax of PEKS mandates that "searching" is done by *testing* a search token against *each* ciphertext, making the search time *linear in the database size*. Its public-key nature also hinders the state maintenance for forward privacy. Specifically, encryptors have no way to get informed of the need for encrypting "differently" to invalidate the old search tokens. PEKS thus faces inherent difficulty in achieving sublinear search and forward privacy.

Categorizing SSE and PEKS, they are "single-writer/single-reader" (S/S) and "multi-writer/single-reader" (M/S), respec-

TABLE I: Comparison of M/$*$ Searchable Encryption Notions

Scheme	Read	Write	Fwd. Pri.	Integrity	Model
HSE [36]	$\mathcal{O}(W+r)$	$O(1)$	Epoch	N/A	M/S
DSE	$\mathcal{O}(r)$	$\mathcal{O}(W)$	Regular	Yes	M/M

W: the number of keywords ever written; r: the size of the search result.

tively, as depicted in Fig. 1. "Single-writer/multi-reader" (S/M) schemes can be obtained from S/S, as observed [16], [11]. The *"multi-writer/multi-reader" (M/M)* architecture is the most flexible and challenging to achieve. This paper aims to address the limitations inherent in SSE and PEKS, providing the first solution to the open problem of devising a searchable encryption scheme that offers *sublinear searches, standard forward privacy, and multi-client support.*

Hybrid searchable encryption (HSE) recently proposed by Wang and Chow [36] makes a significant step towards achieving sublinear search and forward privacy simultaneously in the M/S setting. However, it incurs an additive overhead of public-key operations that scales linearly with the number of active keywords, *i.e.*, keywords ever written in the database. Moreover, HSE only offers epoch-based forward privacy [36]: while updates cannot be identified by search tokens of past epochs, updates within the same epoch as the search token generation remain searchable. It also relies on a global clock to supply epoch information for search and update token generation. Finally, all writers must refresh (part of) their own encrypted databases for the reader at each epoch (hence their tagline "*Omnes pro uno*"[1]). Failure to do so results in the loss of the new updates or the forward privacy over them.[2]

To achieve the best of SSE and PEKS, we formulate *Delegatable Searchable Encryption (*DSE*)*, a new framework for M/M searchable encryption that supports keyword search and update of keyword-document tuples (w, id) through delegating permissions to search and update. That is, to enable clients to search for w or add a new tuple (w, id), a database owner grants the corresponding *searching or updating right* to them.

Anyone can act as an owner by setting up their DSE instances (for projects with separate access control) with simple management: conferring or withholding keys. The delegation is only needed *once per client*. Searching or updating needs no further client-owner interaction. The owner can go offline, similar to running identity-/attribute-based encryption (I/ABE) for access control [14]. DSE thus embodies the spirit of *"Unus pro omnibus."*[1] Delegation opens an implicit *keyword-specific channel* with which eligible clients collaboratively maintain states and indices of the ciphertexts and a door to optimal search and forward privacy. Table I compares DSE and HSE.[3]

[1] *"Unus pro omnibus, omnes pro uno"* is a Latin phrase meaning "One for all, all for one." Inspired by it, DSE asks the owner to set up the keys for all.

[2] The complications stem from the core design of having the writer set up a writer-specific SSE database and store encryption of its SSE search tokens in the encrypted database of the reader. As a search token is ever-changing across epochs for epoch-based forward privacy, the number of ciphertexts for them is also ever-growing. The assistance from each writer is to rebuild this part of the encrypted database to ease the burden of the reader.

[3] Appendix A discusses other so-called "multi-client/user" schemes.

A. Design Overview

DSE essentially upgrades the inverted index of dynamic SSE [24], [8], [27] to the M/M setting. In SSE, the sole client maintains a search counter sct_w and an update counter uct_w for any keyword w as its state, keeping track of the number of searches and updates performed on w. It is crucial to keep uct_w secret, or it leaks whether w is being updated, breaking forward privacy. Meanwhile, sct_w is allowed to be inferrable by the server in SSE schemes [8], [27] to enable efficient searches. These counters form the basis for forward privacy.

Update and Search in SSE. To facilitate updates, the SSE client generates a pseudorandom function (PRF) key. It is used to derive the PRF value of (w, sct_w) as an index key IK_w. When *an update on* w is performed, it will be stored at the address determined by the PRF value of uct_w under key IK_w, which remains pseudorandom until IK_w is revealed. After each update, uct_w is incremented, preparing for the next update.

To *search for* w, the client provides the server with IK_w, which helps locate the addresses storing ciphertexts associated with w until the last address, which is the PRF value of uct_w under key IK_w. After each search on w, sct_w is incremented. It mandates subsequent operations to use new IK_w for new sct_w, rendering keys granted to the server for older sct_w useless.

M/M Upgrades. The DSE database owner delegates IK_w to any client who can perform searches and updates on w. A global state containing sct_w will be published by the server.

To securely maintain uct_w among multiple clients, we tailor a primitive called ***shiftable multi-recipient encryption*** (SME). SME is an efficient public-key encryption scheme for multiple recipients, corresponding to counters uct_w for each keyword in DSE. It allows homomorphic shifting on the ciphertext of uct_w to change its value with each update. After shifting, encrypted update counters *for all keywords* are re-randomized to avoid leaking the updated one. The encrypted uct_w is published via the global state. Only eligible clients with the SME secret key of w delegated can access uct_w for searches or updates on w.

To distinguish readers from writers, the database owner sets up an IBE instance for each keyword. Its identity-based secret key is only delegated to eligible readers. Each update on w is then IBE-encrypted under the "identity" sct_w, invalidating previous keys similar to forward-private SSE.

Optimizing Public-Key Operations. All M/M solutions must differentiate different writers' updates; public-key operations seem unavoidable. DSE further reduces their uses via "microscopic" uses of hybrid encryption. That is, a writer IBE-encrypts the secret key material once, which will then be used for encrypting subsequent updates within the encrypted index of SSE. The traversal only takes one IBE decryption to retrieve the first result from each writer. The remaining operations in the asymptotical-optimal search use only symmetric keys. In contrast, HSE uses hybrid encryption to first encrypt an SSE token via an IBE extension (identity-coupling key-aggregate encryption [36]) before SSE encryption. This makes DSE $240\times$ faster than HSE for the widely-evaluated Enron dataset.

B. Security Overview

Oblivious Access Control via SME**.** The M/M setting poses new threats in the presence of corrupt clients. As an example, access control on the state for different keywords should be enforced in an oblivious manner. Otherwise, the keywords being updated will be revealed. SME is a lightweight tool tailored for securely maintaining counters. Notably, SME is concretely more efficient than generic multi-client oblivious primitives without non-colluding assumptions [15].

Integrity. Integrity is instrumental in many distributed systems with numerous clients. Our DSE could require each client to provide well-formed proof of the operation, including the state modification via SME. Our SME instantiation is as simple as multi-recipient ElGamal encryption. Together with the simple structure of other (encrypted) states, our DSE instantiation provides efficient integrity protection using the recent succinct arguments of knowledge for bilinear group arithmetic [28], in contrast to heavyweight circuit-based proofs.

Non-Committedness. Similar to SSE, DSE and its building block need to be non-committing (see Section II-C). In particular, the security proof of SME involves intrinsic generic-group analysis, unlike the random-oracle-based ElGamal encryption.

Extended Defense. A line of research is devoted to studying the leakage of SSE. Existing generic enhancements for SSE can be easily adapted in DSE since it largely follows the SSE index design, *e.g.*, hiding deleted updates for backward privacy [9] and hiding search result sizes for volume hiding [22].

Keyword-Guessing Mitigation. The delegation process of DSE could further enhance write-access control by delegating keyword-specific secrets required for encrypting and searching [34]. DSE thus resolves the vulnerability to keyword guessing, an inherent weakness of PEKS [6] and HSE [36].

Security for M/M Constructions. Security definitions of SSE are often parameterized by leakage functions [16], [24], *e.g.*, *search pattern* (the repetition of searched keywords) and *access pattern* (the accessed document identifiers). The definition naturally follows real-world versus ideal-world formalization. Yet, the notable M/S notion, PEKS, has no leakage-based definition. Security for DSE requires new definitional efforts.

Moreover, the server in SSE is *the only corrupt party* (essentially an honest-but-curious server). In contrast, a DSE adversary can corrupt not only the server but also *many clients* who have searching or updating rights of *different keywords*, demanding a dedicated formulation *with no direct SSE counterpart*. Meanwhile, it could decide to only corrupt some clients but not the server, and it is not weaker since a DSE scheme might then feature a different leakage profile. Our definitions capture what can be achieved in either case.

Corruption becomes more versatile in an M/M architecture. In particular, we formally define integrity to capture the guarantees that updates from honest clients are always searchable by those authorized, even when malicious clients try to corrupt the encrypted database. In short, the pursuit of DSE is not confined to achieving practical efficiency but also the foundation of a new cryptographic notion and new primitives.

II. Preliminaries

A. Notations

Basic Notations. Polynomial and negligible functions in the security parameter λ are denoted by $\mathsf{poly}(\lambda)$ and $\mathsf{negl}(\lambda)$, respectively. $[n] = \{1, \ldots, n\}$. For a set S, $x \leftarrow\!\!\$\ S$ samples $x \in S$ uniformly at random. For a probabilistic polynomial-time (PPT) algorithm A, $x \leftarrow \mathsf{A}(\cdot)$ executes A and assigns its output to the variable x. A deterministic assignment of y to x is denoted by $x := y$. Algorithms can abort by outputting $\perp$. Empty sets and strings are denoted by $\emptyset$ and ϵ, respectively.

$\mathsf{Alg}\mathcal{O}$ denotes an oracle in the security game regarding algorithm Alg, except $\mathsf{Corr}\mathcal{O}$ is for secret revelation/corruption.

Vectors and Indexed Sets are in or start with capital letters, *e.g.*, Γ, Uctr. The entry-wise multiplication and addition between vectors are denoted by "$\circ$" and "$+$", respectively. All sets considered in this work are indexed. Let $S = \{s_w\}_{w \in \mathcal{W}}$ be a set indexed by an index set $\mathcal{W}$. We often view S as a vector of length $|\mathcal{W}|$. Let $W \subseteq \mathcal{W}$. $S|_W := \{s_w\}_{w \in W}$ is the restriction of S by W. $S[w]$ is the entry of S at position w.

$\Gamma_{W:\mathcal{W}}$ is the ***characteristic vector*** of W with respect to $\mathcal{W}$. $\Gamma_{W:\mathcal{W}}[w] = 1$ if $w \in W$; 0, otherwise. "$: \mathcal{W}$" is omitted if it is clear. These notations are crucial for, *e.g.*, shifting counters in SME ($\mathsf{Uctr}|_W = \{5, 0\}$ for $W = \{w_2, w_3\}$ in Fig. 15.)

Implicit Cyclic Group Notation. Let $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_t$ be cyclic groups of order q with pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_t$. For $i \in \{1, 2, t\}$, we denote the generator of $\mathbb{G}_i$ by $[1]_i$. $[1]_t := [1]_1[1]_2$ is a generator of $\mathbb{G}_t$. For $x, y \in \mathbb{Z}_q$, elements in group $\mathbb{G}$ with discrete logarithm x with respect to $[1]$ is denoted by $[x]$; group operations are denoted additively, *e.g.*, $[x] + [y] := [x + y]$; $[x]$ to the power of y is denoted by $[xy] := y \cdot [x]$. Pairing operations are expressed multiplicatively as $[x]_1[y]_2 := [xy]_t$.

B. Non-Committing Primitives

Definition 1 (Non-Committing Pseudorandom Functions)**.** *A PRF family $F : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$ is non-committing pseudorandom if there exist* PPT *simulators $\mathcal{S}_1, \mathcal{S}_2, \mathcal{S}_3$ such that for any sequence of inputs $\{x_i\}_{i=1}^{\ell}$, $\ell \in \mathsf{poly}(\lambda)$, the distributions below are computationally indistinguishable:*
$$\left\{ \begin{array}{c} \mathsf{td} \leftarrow \mathcal{S}_1(1^\lambda); \\ (K, y_1, \ldots, y_\ell) : (y_i, \mathsf{td}) \leftarrow \mathcal{S}_2(\mathsf{td}, i)\ \forall\ i \in [\ell]; \\ K \leftarrow \mathcal{S}_3(\mathsf{td}, \{x_i\}_{i=1}^{\ell}) \end{array} \right\} \text{ and}$$
$$\{(K, y_1, \ldots, y_\ell) : K \leftarrow\!\!\$\ \mathcal{K}; y_i \leftarrow F(K, x_i)\ \forall\ i \in [\ell]\}.$$

It is easy to show that any function $F : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$ is non-committing pseudorandom if F is a random oracle. For the i-th query on a PRF value, the simulator randomly selects $y_i \leftarrow\!\!\$\ \mathcal{Y}$ as the answer and records (i, y_i). After all ℓ queries, the simulator receives $\{x_i\}_{i=1}^{\ell}$, randomly selects $K \leftarrow\!\!\$\ \mathcal{K}$, and programs the random oracle with $y_i = F(K, x_i)$ for $i \in [\ell]$.

Definition 2 (Identity-Based Encryption [7])**.** *An* IBE *scheme is a tuple of* PPT *algorithms defined as follows. Below also recall Boneh–Franklin* IBE *[7], using cryptographic hash functions $H_1 : \{0,1\}^* \to \mathbb{G}_1$ and $H_2 : \mathbb{G}_t \to \{0,1\}^\lambda$.*

$(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KG}(1^\lambda)$*: It outputs a master key pair* $(\mathsf{pk}, \mathsf{sk})$*.*

Output $(\mathsf{pk}, \mathsf{sk}) := ([s]_2, s)$*, where $s \leftarrow\!\!\$\ \mathbb{Z}_q$.*

$\mathsf{Real\text{-}NC}^{\mathsf{IBE}}_{\mathcal{A}}(1^\lambda)$	$\mathsf{Ext}\mathcal{O}(w)$	$\mathsf{Enc}\mathcal{O}(w,m)$
$(\mathsf{pk},\mathsf{sk}) \leftarrow \mathsf{KG}(1^\lambda)$	$\mathsf{dk}_w \leftarrow \mathsf{Ext}(\mathsf{sk},w)$	$c \leftarrow \mathsf{Enc}(\mathsf{pk},w,m)$
return $b \leftarrow \mathcal{A}^{\mathsf{Ext}\mathcal{O},\mathsf{Enc}\mathcal{O}}(\mathsf{pk})$	**return** dk_w	**return** c

Fig. 2: Real Experiment for Non-Committing IBE

$\mathsf{Ideal\text{-}NC}^{\mathsf{IBE}}_{\mathcal{A},\mathcal{S}}(1^\lambda)$	$\mathsf{Ext}\mathcal{O}(w)$
$D :=$ Empty dictionary	$(\mathsf{dk}_w,\mathsf{td}) \leftarrow \mathcal{S}(\mathsf{td},w,D[w]), \mathsf{Corr} := \mathsf{Corr} \cup \{w\}$
$\mathsf{Corr} := \emptyset$	**return** dk_w
$(\mathsf{pk},\mathsf{td}) \leftarrow \mathcal{S}(1^\lambda)$	$\mathsf{Enc}\mathcal{O}(w,m)$
$b \leftarrow \mathcal{A}^{\mathsf{Ext}\mathcal{O},\mathsf{Enc}\mathcal{O}}(\mathsf{pk})$	**if** $w \in \mathsf{Corr}$ **then** $c \leftarrow \mathsf{Enc}(\mathsf{pk},w,m)$
return b	**else** $(c,\mathsf{td}) \leftarrow \mathcal{S}(\mathsf{td}), D[w] := D[w] \cup \{(m,c)\}$
	return c

Fig. 3: Ideal Experiment for Non-Committing IBE

$\mathsf{dk}_w \leftarrow \mathsf{Ext}(\mathsf{sk},w)$*: The algorithm inputs a secret key* sk *and an identity* w*. It outputs a decryption key* dk_w *for* w*.*

Output $\mathsf{dk}_w := [d]_1 = \mathsf{sk}[h]_1$*, where* $[h]_1 := H_1(w)$*.*

$\mathsf{ctx} \leftarrow \mathsf{Enc}(\mathsf{pk},w,m)$*: The algorithm inputs* pk*, an identity* w*, and a message* m*. It outputs a ciphertext* ctx*.*

Pick $r \leftarrow\!\!\$\ \mathbb{Z}_q$*. Parse* pk *as* $[s]_2$*. Compute* $[h]_1 = H_1(w)$*. Output* $\mathsf{ctx} := ([c_0]_2 := [r]_2, c_1 := H_2(r[h]_1[s]_2) \oplus m)$*.*

$m \leftarrow \mathsf{Dec}(\mathsf{dk}_w,\mathsf{ctx})$*: The algorithm inputs a key* dk_w *and a ciphertext* ctx*. It outputs the decrypted result* m*.*

Parse ctx *as* $([c_0]_2, c_1)$*. Output* $m := c_1 \oplus H_2(\mathsf{dk}_w[c_0]_2)$*.*

Definition 3 (Non-Committing IBE)**.** *An* IBE *scheme is (recipient) non-committing if, for any* PPT *adversary* $\mathcal{A}$*, there exists a* PPT *simulator* $\mathcal{S}$ *s.t. the quantity is negligible:*

$$\left|\Pr[\mathsf{Real\text{-}NC}^{\mathsf{IBE}}_{\mathcal{A}}(1^\lambda)=1] - \Pr[\mathsf{Ideal\text{-}NC}^{\mathsf{IBE}}_{\mathcal{A},\mathcal{S}}(1^\lambda)=1]\right|.$$

$\mathsf{Real\text{-}NC}^{\mathsf{IBE}}_{\mathcal{A}}$ *and* $\mathsf{Ideal\text{-}NC}^{\mathsf{IBE}}_{\mathcal{A},\mathcal{S}}$ *are defined in Figs. 2 and 3.*

Unlike existing definitions for non-committing encryption that work with one public key, *e.g.*, [20], Definition 3 allows many ciphertexts under different identities. We formulate an encryption oracle that outputs a ciphertext c without using message m and identity w, and records (m,c) in a dictionary entry $D[w]$, used by the simulated extraction oracle $\mathsf{Ext}\mathcal{O}$. This simplifies the housekeeping for multi-recipient encryption (*e.g.*, Definition 5): When some recipients can be compromised (put into Corr), simulation without the message is not possible.

To show that Boneh–Franklin IBE is non-committing in the random oracle model, the simulator outputs a random tuple $([c_0]_2 = [r]_2, c_1)$ to simulate a ciphertext. When the identity w and message m are later revealed, the simulator samples $k \in \mathbb{Z}_q$. It programs $H_1(w) := [k]_1$ and $H_2(r[k]_1[s]_2) := c_1 \oplus m$, and outputs the identity decryption key as $k[s]_1$. The simulation is perfect, modulo hash collisions.

C. Non-Committedness and Security of Searchable Encryption

Non-committedness is necessary for realizing adaptive security in traditional SSE schemes. For an update tuple (w,id), an SSE server stores a symmetric-key encryption $\mathsf{Enc}(\mathsf{sk}_w,\mathsf{id})$ with sk_w derived from a PRF. The simulator needs to simulate $\mathsf{Enc}(\mathsf{sk}_w,\mathsf{id})$ without knowing w or id. Not until the adversary requests to search for w, the simulator is given all $\{\mathsf{id}_i\}$ where $\{(w,\mathsf{id}_i)\}$ were supposed to be previously updated. Now, to "explain" all the ciphertexts $\{c_i\}$ previously simulated SSE simulators [16], [9] will patch the query-response of the random oracle to output sk such that $\mathsf{Dec}(\mathsf{sk},c_i) = \mathsf{id}_i$, which effectively realizes non-committing encryption $\mathsf{Enc}(\mathsf{sk}_w,\mathsf{id})$ of input id and non-committing PRF $F(\mathsf{sk}_w,w)$ of input w.

The DSE simulator needs to do more. It "explains" ciphertexts and keys not only when receiving requests for searching but also when *the corruption of clients with related rights* happens. The simulation can be realized given the well-defined leakage and the non-committing property of underlying primitives, *i.e.*, PRF, IBE, and our SME in Section III.

III. Shiftable Multi-Recipient Encryption (SME)

Shiftable multi-recipient encryption (SME) is our tailored notion of homomorphic encryption over a set of messages, each in a slot for a recipient. We will use SME to encrypt the update counters in the global state of DSE, with each recipient corresponding to a keyword. SME features two properties.

Shiftable: Each slot features homomorphic addition over the message space $(\mathcal{M},+)$. Namely, an SME-encrypted message m in a slot can be *shifted* by an offset m' via encrypting m' for the same slot and adding up these two ciphertexts.

Expandable: SME allows expanding a multi-recipient ciphertext to include more recipients by using their secret keys.

A. Syntax

Definition 4 (Shiftable Multi-Recipient Encryption)**.** *An* SME *scheme for message space* $(\mathcal{M},+)$*, ciphertext space* $(\mathcal{C},+)$*, and recipient space* $\mathcal{R}$ *is a tuple of* PPT *algorithms:*

$\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$*: It generates public parameters* pp*.*

$(\mathsf{pk},\mathsf{sk}) \leftarrow \mathsf{KG}(1^\lambda)$*: It generates a key pair* $(\mathsf{pk},\mathsf{sk})$*.*

$\mathsf{ctx}_\emptyset \leftarrow \mathsf{Init}(1^\lambda)$*: The ciphertext initialization algorithm generates an initial ciphertext* $\mathsf{ctx}_\emptyset \in \mathcal{C}$ *associated with the empty set of recipients. In general, the ciphertext* ctx_A *associated with the recipient set* $A \subseteq \mathcal{R}$ *encrypts messages in* $\mathcal{M}^A$*.*

$\mathsf{ctx}_{A\cup B} \leftarrow \mathsf{Expand}(\mathsf{SK}|_A,\mathsf{ctx}_B)$*: This algorithm expands the ciphertext* $\mathsf{ctx}_B \in \mathcal{C}$ *associated with the set* $B \subseteq \mathcal{R}$ *of recipients to the ciphertext* $\mathsf{ctx}_{A\cup B} \in \mathcal{C}$ *associated with* $A \cup B$*, where* $A \cap B = \emptyset$*, given the decryption keys* $\mathsf{SK}|_A$ *of* A*. Each expanded slot encrypts the identity* $0_\mathcal{M} \in \mathcal{M}$*.*

$\mathsf{ctx}_A \leftarrow \mathsf{Enc}(\mathsf{PK}|_A, M|_A)$*: This algorithm inputs a set of encryption keys* $\mathsf{PK}|_A$ *and a set of message* $M|_A$ *associated with the recipient set* $A \subseteq \mathcal{R}$*. It outputs a ciphertext* $\mathsf{ctx}_A \in \mathcal{C}$*.*

$\mathsf{ctx}''_A \leftarrow \mathsf{Shift}(\mathsf{ctx}_A,\mathsf{ctx}'_A)$*: This algorithm inputs two ciphertexts* $\mathsf{ctx}_A, \mathsf{ctx}'_A \in \mathcal{C}$ *that encrypt some messages* $M|_A, M'|_A \in \mathcal{M}$ *for the recipient set* $A \subseteq \mathcal{R}$*, respectively. It outputs a new ciphertext* ctx''_A *that supposedly encrypts the shifted messages* $M + M'|_A$*, where "+" is done entry-wise.*

$M|_A \leftarrow \mathsf{Dec}(\mathsf{SK}|_A,\mathsf{ctx}_B)$*: The decryption algorithm inputs a set of decryption keys* $\mathsf{SK}|_A$ *and a ciphertext* $\mathsf{ctx}_B \in \mathcal{C}$ *with* $A \subseteq B$*. It outputs the messages* $M|_A$*.*

$\mathsf{Real\text{-}NC}^{\mathsf{SME}}_{\mathcal{A}}(1^\lambda)$

$D, D' :=$ Empty dictionary
$\nu, \mu := 0,\ D[\nu] := \emptyset,\ D'[\mu] := \emptyset$
$\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda), \mathsf{ctx}_\nu \leftarrow \mathsf{Init}(1^\lambda)$
for $i \in \mathcal{R}$ **do** $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{KG}(1^\lambda)$
$b \leftarrow \mathcal{A}^{\mathbb{O}}(\mathsf{PK}|_{\mathcal{R}}, \mathsf{ctx}_\nu)$
return b

$\mathsf{Expand}\mathcal{O}(j, A)$

ensure $j \in [\nu] \cup \{0\}, D[j] \cap A = \emptyset$
$\nu := \nu + 1, D[\nu] := D[j] \cup A$
$\mathsf{ctx}_\nu \leftarrow \mathsf{Expand}(\mathsf{SK}|_A, \mathsf{ctx}_j)$
return ctx_ν

$\mathsf{Enc}\mathcal{O}(j, M|_A)$

ensure $j \in [\nu], A = D[j]$
$\mu := \mu + 1, D'[\mu] := D[j]$
$\mathsf{ctx}'_\mu := \mathsf{Enc}(\mathsf{PK}|_{D[j]}, M|_{D[j]})$
return ctx'_μ

$\mathsf{Shift}\mathcal{O}(j, t)$

ensure $j \in [\nu], t \in [\mu]$
ensure $D'[t] = D[j]$
$\nu := \nu + 1, D[\nu] := D[j]$
$\mathsf{ctx}_\nu := \mathsf{ctx}_j + \mathsf{ctx}'_t$
return ctx_ν

$\mathsf{Corr}\mathcal{O}(i^*)$

return sk_{i^*}

Fig. 4: Real Experiment for Non-Committing SME

$\mathsf{Ideal\text{-}NC}^{\mathsf{SME}}_{\mathcal{A},\mathcal{S}}(1^\lambda)$

$D, D', L, L' :=$ Empty dictionary
$\nu, \mu := 0, D[\nu] := \emptyset, D'[\mu] := \emptyset$
$\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda), \mathsf{ctx}_\nu \leftarrow \mathcal{S}(\mathtt{Init}, 1^\lambda)$
for $i \in \mathcal{R}$ **do** $\mathsf{pk}_i \leftarrow \mathcal{S}(\mathtt{KGen}, 1^\lambda)$
$b \leftarrow \mathcal{A}^{\mathbb{O}}(\mathsf{PK}|_{\mathcal{R}}, \mathsf{ctx}_\nu)$
return b

$\mathsf{Enc}\mathcal{O}(j, M|_A)$

ensure $j \in [\nu], A = D[j]$
parse $M|_A$ **as** $\{m_i\}_{i \in A}$
$\mu := \mu + 1, D'[\mu] := D[j]$
for $i \in D'[\mu]$ **do** $L'[i][\mu] := m_i$
$\mathsf{ctx}'_\mu \leftarrow \mathcal{S}(\mathtt{Enc}, \{m_i\}_{i \in \mathsf{Corr}})$
return ctx'_μ

$\mathsf{Expand}\mathcal{O}(j, A)$

ensure $j \in [\nu] \cup \{0\}, D[j] \cap A = \emptyset$
$\nu := \nu + 1,\ D[\nu] := D[j] \cup A$
for $i \in D[j]$ **do** $L[i][\nu] := L[i][j]$
for $i \in A$ **do** $L[i][\nu] := 0_{\mathcal{M}}$
$\mathsf{ctx}_\nu \leftarrow \mathcal{S}(\mathtt{Expand})$
return ctx_ν

$\mathsf{Shift}\mathcal{O}(j, t)$

ensure $j \in [\nu], t \in [\mu]$
ensure $D'[t] = D[j]$
$\nu := \nu + 1, D[\nu] := D[j]$
for $i \in D[\nu]$
do $L[i][v] := L[i][j] + L'[i][t]$
$\mathsf{ctx}_\nu := \mathsf{ctx}_j + \mathsf{ctx}'_t$
return ctx_ν

$\mathsf{Corr}\mathcal{O}(i^*)$

$\mathsf{sk}_{i^*} \leftarrow \mathcal{S}(\mathtt{Corr}, L[i^*], L'[i^*]), \mathsf{Corr} := \mathsf{Corr} \cup \{i^*\},$ **return** sk_{i^*}

Fig. 5: Ideal Experiment for Non-Committing SME

An SME scheme for message space $(\mathcal{M}, +)$, ciphertext space $\mathcal{C}$, and recipient space $\mathcal{R}$ is correct if for any λ, $\mathsf{pp} \in \mathsf{Setup}(1^\lambda)$, $(\mathsf{pk}, \mathsf{sk}) \in \mathsf{KG}(1^\lambda)$, and $A \subseteq \mathcal{R}$, we have:

- $\mathsf{Dec}(\mathsf{SK}|_{A \cup B}, \mathsf{Expand}(\mathsf{SK}|_B, \mathsf{ctx}_A)) = \mathsf{Dec}(\mathsf{SK}|_A, \mathsf{ctx}_A) \cup 0_{\mathcal{M}}|_B$, for any $\mathsf{ctx}_A \in \mathcal{C}$ and $B \subseteq \mathcal{R}$ s.t. $A \cap B = \emptyset$, where $0_{\mathcal{M}}|_B$ is a set of copies of $0_{\mathcal{M}}$ at slots indexed by B.
- $\mathsf{Dec}(\mathsf{SK}|_A, \mathsf{ctx}_A) = M|_A$ for any $\mathsf{ctx}_A \in \mathsf{Enc}(\mathsf{PK}|_A, M|_A)$.
- $\mathsf{Dec}(\mathsf{SK}|_A, \mathsf{ctx}''_A) = \mathsf{Dec}(\mathsf{SK}|_A, \mathsf{ctx}_A) + \mathsf{Dec}(\mathsf{SK}|_A, \mathsf{ctx}'_A) = M|_A + M'|_A$ for any $\mathsf{ctx}_A \in \mathsf{Enc}(\mathsf{PK}|_A, M|_A)$, $\mathsf{ctx}'_A \in \mathsf{Enc}(\mathsf{PK}|_A, M'|_A)$, and $\mathsf{ctx}''_A \in \mathsf{Shift}(\mathsf{ctx}_A, \mathsf{ctx}'_A)$.

Shift Non-Committing. Traditional non-committing encryption [20] requires the existence of an efficient simulator $\mathcal{S}$. On start-up, $\mathcal{S}$ simulates a public key pk. Subsequently, $\mathcal{S}$ simulates an encryption oracle outputting a ciphertext ctx for each message m, given that m is chosen by an adversary $\mathcal{A}$ and unknown to $\mathcal{S}$. Upon the request of $\mathcal{A}$, $\mathcal{S}$ simulates a secret key sk to "explain" the ciphertexts simulated previously.

$\mathsf{KG}(1^\lambda)$

$x \leftarrow\!\$\ \mathbb{Z}_q$
$\mathsf{sk} := x$
$\mathsf{pk} := [x]$
return $(\mathsf{pk}, \mathsf{sk})$

$\mathsf{Init}(1^\lambda)$

$r \leftarrow\!\$\ \mathbb{Z}_q$
$\mathsf{ctx}_\emptyset := ([r], \emptyset)$
return $\mathsf{ctx}_\emptyset$

$\mathsf{Enc}(\mathsf{PK}|_A, M|_A)$

parse $\mathsf{PK}|_A$ **as** $\{[x_i]\}_{i \in A}$
parse $M|_A$ **as** $\{[m_i]\}_{i \in A}$
$r \leftarrow\!\$\ \mathbb{Z}_q$
return $([r], \{r[x_i] + [m_i]\}_{i \in A})$

$\mathsf{Shift}(\mathsf{ctx}_A, \mathsf{ctx}'_A)$

parse ctx_A **as** $([c_0], \{[c_i]\}_{i \in A})$
parse ctx'_A **as** $\left([c'_0], \{[c'_i]\}_{i \in A}\right)$
for $i \in A$ **do** $[c''_i] := [c_i] + [c'_i]$
$\mathsf{ctx}''_A := \left([c_0] + [c'_0], \{[c''_i]\}_{i \in A}\right)$
return ctx''_A

$\mathsf{Expand}(\mathsf{SK}|_A, \mathsf{ctx}_B)$

ensure $A \cap B = \emptyset$
parse $\mathsf{SK}|_A$ **as** $\{x_i\}_{i \in A}$
parse ctx_B **as** $([c_0], \{[c_i]\}_{i \in B})$
for $i \in A$ **do** $[c_i] := [c_0] \cdot x_i$
$\mathsf{ctx}_{A \cup B} := ([c_0], \{[c_i]\}_{i \in A \cup B})$
return $\mathsf{ctx}_{A \cup B}$

$\mathsf{Dec}(\mathsf{SK}|_A, \mathsf{ctx}_B)$

ensure $A \subseteq B$
parse $\mathsf{SK}|_A$ **as** $\{x_i\}_{i \in A}$
parse ctx_B **as** $([c_0], \{[c_i]\}_{i \in B})$
for $i \in A$ **do**
$[m_i] := [c_i] - [c_0] \cdot x_i$
return $\{[m_i]\}_{i \in A}$

Fig. 6: Construction of SME

Shiftable non-committing SME further requires $\mathcal{S}$ to simulate the result of homomorphic shifting on any ciphertext without knowing *which slots are shifted* and the offsets in shifting. This shift non-committing property for multi-recipient encryption has not been formalized before.[4] Note that this notion subsumes the chosen plaintext attack (CPA) security.

Definition 5 (Shift Non-Committing)**.** *An* SME *scheme is shift non-committing, if for any* PPT *adversary* $\mathcal{A}$*, there exists a* PPT *simulator* $\mathcal{S}$ *s.t. the following quantity is negligible:*

$$\left|\Pr[\mathsf{Real\text{-}NC}^{\mathsf{SME}}_{\mathcal{A}}(1^\lambda) = 1] - \Pr[\mathsf{Ideal\text{-}NC}^{\mathsf{SME}}_{\mathcal{A},\mathcal{S}}(1^\lambda) = 1]\right|.$$

$\mathsf{Real\text{-}NC}^{\mathsf{SME}}_{\mathcal{A}}$ *and* $\mathsf{Ideal\text{-}NC}^{\mathsf{SME}}_{\mathcal{A},\mathcal{S}}$ *are defined in Figs. 4 and 5.*

In Figs. 4 and 5, adversary $\mathcal{A}$ can access oracles $\mathbb{O} := \{\mathsf{Expand}\mathcal{O}, \mathsf{Enc}\mathcal{O}, \mathsf{Shift}\mathcal{O}, \mathsf{Corr}\mathcal{O}\}$. ctx_ν is the ciphertext resulted from $\mathsf{Expand}\mathcal{O}$ and $\mathsf{Shift}\mathcal{O}$. ctx'_μ is the ciphertext from $\mathsf{Enc}\mathcal{O}$. $D[j]$ records the recipients of the j-th version of ctx_ν for $j \in [\nu]$, similar to $D'[t]$ for ctx_μ. $L[i][j]$ records the i-th message of the j-th version of ctx_ν, similar to $L'[i][t]$ for ctx_μ.

B. Construction

Let $\mathbb{G}$ be a group of λ-bit prime order q where the decisional Diffie-Hellman (DDH) assumption holds. Our SME scheme starts with $\mathsf{Setup}(1^\lambda)$ that outputs $(\mathbb{G}, [1], q)$. Fig. 6 describes the other algorithms, which equips $\mathsf{Expand}()$ to the multi-recipient ElGamal encryption with randomness reusing [4]. Our shift non-committing property was not explored [4].

Our SME scheme is very efficient. With the encryption slots (*i.e.*, authorized keywords in DSE) known and small-exponent exponentiation (for small counter values in DSE) precomputed,

[4]Function-wise, one might view SME as a set of cryptographic counters [25] However, there is no (shift-)non-committing cryptographic counter.

$\mathsf{Enc}()$ can pick r and perform the exponentiations offline. Online computation is purely modular additions.

A shift non-committing SME is also a non-committing encryption; hence, construction in the standard model is impossible [29].[5] Non-committing encryption can be constructed trivially in the random oracle (RO) model, *e.g.*, by hashing the ephemeral keys produced by a key encapsulation mechanism with an RO. However, such usage of RO destroys the homomorphism of ciphertexts. We thus consider an alternative idealized model, *i.e.*, the generic group model (GGM).[6]

It is straightforward to show that our SME scheme is correct and CPA-secure under DDH [4]. Theorem 1 asserts its shift non-committing property with the proof in Appendix C.

Theorem 1. SME *(Fig. 6) is shift non-committing in GGM.*

IV. Delegatable Searchable Encryption (DSE)

A. System Model

DSE involves three parties: *a server* that provides storage services; *a database owner* that initializes the system and delegates the searching and/or updating rights of specified keyword sets to any client; and *multiple clients* that become readers and/or writers of some keywords after delegations.

A writer could insert keyword-document tuples for any of their update-permitted keywords to the database. A reader could retrieve the identifiers of documents matching any of their search-permitted keywords from the database. Neither writers nor readers *own* the data, similar to the traditional Unix permissions – a file only has one owner, the database owner.

Different from the convention where the server is the *only* adversary, DSE clients with searching and updating rights could also be adversarial. We consider security in cases of an honest-but-curious (or simply *curious*) server and an honest server separately, while client corruption could happen in both cases. Corrupt parties aim to derive information beyond the confined leakage. Meanwhile, malicious clients might intentionally submit faulty requests to tamper with the global state or the database such that updates from honest clients could not be searched. DSE defines integrity against this misbehavior.

B. Syntax

Let the keyword space be $\{0,1\}^*$. When delegating, the data owner specifies two keyword sets $\mathcal{W}_{\mathsf{Q}}, \mathcal{W}_{\text{✎}} \subseteq \{0,1\}^*$ for each client, denoting search-permitted and update-permitted keywords, respectively. Let $\mathcal{W} \subseteq \{0,1\}^*$ be the active keyword space, which is the union of keywords ever delegated by the database owner. When a delegation on $(\mathcal{W}_{\mathsf{Q}}, \mathcal{W}_{\text{✎}})$ involves new keywords (*i.e.*, $(\mathcal{W}_{\mathsf{Q}} \cup \mathcal{W}_{\text{✎}}) \backslash \mathcal{W} \neq \emptyset$), $\mathcal{W}$ will be expanded to ensure $\mathcal{W}_{\mathsf{Q}} \subseteq \mathcal{W}$ and $\mathcal{W}_{\text{✎}} \subseteq \mathcal{W}$ after the delegation.

DSE assumes a global state: the server maintains a public state and refreshes it after delegation (by the database owner) and search or update operations (by the server alone). Any client could synchronize with the latest global state in an offline phase. To search/update, a client enters an online phase to produce the corresponding token. The server operates with the token and refreshes the state. Our syntax follows a *non-interactive* formulation, *i.e.*, no extra roundtrip between any parties except the inevitable one for searches.

Definition 6 (Delegatable Searchable Encryption)**.** *A* DSE *scheme consists of a tuple of* PPT *algorithms:*

$(\mathsf{msk}, \mathsf{st}, \mathsf{EDB}) \leftarrow \mathsf{Setup}(1^\lambda)$*: It is run by the database owner inputting the security parameter* 1^λ*. It outputs a master secret key* msk *to be kept secret, an initial state* st*, and an initially empty encrypted database* EDB*.* st *and* EDB *are forwarded to the server. The server will publish* st*, which serves as an input to the remaining algorithms.*

$(\mathsf{st}', \mathsf{sk}) \leftarrow \mathsf{Delegate}(\mathsf{st}, \mathsf{msk}, \mathcal{W}_{\mathsf{Q}}, \mathcal{W}_{\text{✎}})$*: The delegation algorithm is run by the database owner to delegate searching and/or updating rights to a client. It takes global state* st*, a master secret key* msk*, and two sets of keywords* $(\mathcal{W}_{\mathsf{Q}}, \mathcal{W}_{\text{✎}})$*.*

It outputs an updated global state st' *to be published by the server, and a secret key* sk *to be forwarded to the client. The client with* sk *can search for any keyword in* $\mathcal{W}_{\mathsf{Q}}$ *and encrypt documents with respect to any keyword in* $\mathcal{W}_{\text{✎}}$*.*

$\mathfrak{s} \leftarrow \mathsf{SrchTkn}(\mathsf{st}, \mathsf{sk}, W_{\mathsf{Q}})$*: The algorithm allows a client to generate a search token for keywords s/he could search for. Suppose that a client possesses a secret key* sk *for search-permitted keywords* $\mathcal{W}_{\mathsf{Q}}$*. The algorithm inputs secret key* sk *and a keyword set* $W_{\mathsf{Q}} \subseteq \mathcal{W}_{\mathsf{Q}}$*. It outputs a search token* $\mathfrak{s}$*.*

$(\mathsf{st}', \mathsf{EDB}', \mathsf{ID}|_{W_{\mathsf{Q}}}) \leftarrow \mathsf{Srch}(\mathsf{st}, \mathsf{EDB}, \mathfrak{s})$*: The algorithm lets a server search an encrypted database* EDB *with a valid search token* $\mathfrak{s}$*. It outputs updated state* st'*, (possibly) updated encrypted database* EDB'*, and search results* $\mathsf{ID}|_{W_{\mathsf{Q}}}$*.*

$\mathfrak{u} \leftarrow \mathsf{UpdtTkn}(\mathsf{st}, \mathsf{sk}, \mathsf{ID}|_{W_{\text{✎}}})$*: The algorithm allows a client to generate an update token for keywords s/he could update. Suppose that a client possesses a secret key* sk *for update-permitted keywords* $\mathcal{W}_{\text{✎}}$*. The algorithm inputs the secret key* sk *and a set* $\mathsf{ID}|_{W_{\text{✎}}}$ *of document identifiers indexed by* $W_{\text{✎}} \subseteq \mathcal{W}_{\text{✎}}$ *to be inserted.*[7] *It outputs an update token* $\mathfrak{u}$*.*

$(\mathsf{st}', \mathsf{EDB}') \leftarrow \mathsf{Updt}(\mathsf{st}, \mathsf{EDB}, \mathfrak{u})$*: The algorithm allows a server to update an encrypted database* EDB *with a valid update token* $\mathfrak{u}$*. It outputs updated* st' *and updated* EDB'*.*

A DSE scheme is correct if: for any keyword w and any set of document identifiers ID, if (w, id) for all $\mathsf{id} \in \mathsf{ID}$ has been written by any client with updating rights of w, any client with searching rights of w who searches for w obtains results that contain ID. Integrity (Section IV-D) subsumes correctness.

C. Security

Adaptive security of DSE is considered in two settings – one with a curious server and the other with an honest server, both parameterized by the corresponding leakage function $\mathcal{L}$. Adversary $\mathcal{A}$ could adaptively access oracles of delegation $\mathsf{Delegate}\mathcal{O}$, search $\mathsf{Srch}\mathcal{O}$, update $\mathsf{Updt}\mathcal{O}$, and corruption

[5] It could be possible under a relaxed definition where the simulator is only required to simulate an a priori bounded number of ciphertexts.

[6] GGM has been used in multi-client searchable encryption [2], [26].

[7] A way for deletion is to maintain a second instance for deleted tuples [8].

$\mathsf{Real\text{-}CS}^{\mathsf{DSE}}_{\mathcal{A}}(1^\lambda)$

$I := \emptyset$ // client identifier set
$(\mathsf{msk}, \mathsf{st}, \mathsf{EDB}) \leftarrow \mathsf{DSE.Setup}(1^\lambda)$
$\mathbb{O} := \left\{ \begin{array}{c} \mathsf{Srch}\mathcal{O}, \mathsf{Updt}\mathcal{O}, \\ \mathsf{Delegate}\mathcal{O}, \mathsf{Corr}\mathcal{O} \end{array} \right\}$
return $b \leftarrow \mathcal{A}^{\mathbb{O}}(\mathsf{st}, \mathsf{EDB})$

$\mathsf{Delegate}\mathcal{O}(i, \mathcal{W}_{\mathbf{Q}}, \mathcal{W}_{\mathbf{U}})$

ensure $i \notin I$
$I := I \cup \{i\}$
$(\mathcal{W}_{\mathbf{Q},i}, \mathcal{W}_{\mathbf{U},i}) := (\mathcal{W}_{\mathbf{Q}}, \mathcal{W}_{\mathbf{U}})$
$(\mathsf{st}', \mathsf{sk}_i) \leftarrow$
$\mathsf{Delegate}(\mathsf{st}, \mathsf{msk}, \mathcal{W}_{\mathbf{Q}}, \mathcal{W}_{\mathbf{U}})$
return st'

$\mathsf{Srch}\mathcal{O}\,(i, W_{\mathbf{Q}})$

ensure $i \in I \wedge W_{\mathbf{Q}} \subseteq \mathcal{W}_{\mathbf{Q},i}$
$\mathfrak{s} \leftarrow \mathsf{SrchTkn}\,(\mathsf{st}, \mathsf{sk}_i, W_{\mathbf{Q}})$
return $\mathfrak{s}$

$\mathsf{Updt}\mathcal{O}\left(i, \mathsf{ID}|_{W_{\mathbf{U}}}\right)$

ensure $i \in I \wedge W_{\mathbf{U}} \subseteq \mathcal{W}_{\mathbf{U},i}$
$\mathfrak{u} \leftarrow \mathsf{UpdtTkn}\left(\mathsf{st}, \mathsf{sk}_i, \mathsf{ID}|_{W_{\mathbf{U}}}\right)$
return $\mathfrak{u}$

$\mathsf{Corr}\mathcal{O}(i)$

ensure $i \in I$
return sk_i

Fig. 7: Real Experiment for DSE with A Curious Server

$\mathsf{Ideal\text{-}CS}^{\mathsf{DSE}}_{\mathcal{A},\mathcal{S},\mathcal{L}}(1^\lambda)$

$I := \emptyset$ // client identifier set
$\mathsf{Hist} := \mathtt{Setup}$
$(\mathsf{st}, \mathsf{EDB}) \leftarrow \mathcal{S}(\mathcal{L}(\mathsf{Hist}))$
$\mathbb{O} := \left\{ \begin{array}{c} \mathsf{Srch}\mathcal{O}, \mathsf{Updt}\mathcal{O}, \\ \mathsf{Delegate}\mathcal{O}, \mathsf{Corr}\mathcal{O} \end{array} \right\}$
return $b \leftarrow \mathcal{A}^{\mathbb{O}}(\mathsf{st}, \mathsf{EDB})$

$\mathsf{Delegate}\mathcal{O}(i, \mathcal{W}_{\mathbf{Q}}, \mathcal{W}_{\mathbf{U}})$

ensure $i \notin I$
$I := I \cup \{i\}$
$(\mathcal{W}_{\mathbf{Q},i}, \mathcal{W}_{\mathbf{U},i}) := (\mathcal{W}_{\mathbf{Q}}, \mathcal{W}_{\mathbf{U}})$
$\mathsf{Hist} :=$
$\mathsf{Hist} \| (\mathsf{Delegate}, i, \mathcal{W}_{\mathbf{Q}}, \mathcal{W}_{\mathbf{U}})$
return $\mathsf{st}' \leftarrow \mathcal{S}(\mathcal{L}(\mathsf{Hist}))$

$\mathsf{Srch}\mathcal{O}\,(i, W_{\mathbf{Q}})$

ensure $i \in I \wedge W_{\mathbf{Q}} \subseteq \mathcal{W}_{\mathbf{Q},i}$
$\mathsf{Hist} := \mathsf{Hist} \| \,(\mathsf{Srch}, i, W_{\mathbf{Q}})$
return $\mathfrak{s} \leftarrow \mathcal{S}(\mathcal{L}(\mathsf{Hist}))$

$\mathsf{Updt}\mathcal{O}\left(i, \mathsf{ID}|_{W_{\mathbf{U}}}\right)$

ensure $i \in I \wedge W_{\mathbf{U}} \subseteq \mathcal{W}_{\mathbf{U},i}$
$\mathsf{Hist} := \mathsf{Hist} \| \left(\mathsf{Updt}, i, \mathsf{ID}|_{W_{\mathbf{U}}}\right)$
return $\mathfrak{u} \leftarrow \mathcal{S}(\mathcal{L}(\mathsf{Hist}))$

$\mathsf{Corr}\mathcal{O}(i)$

ensure $i \in I$
$\mathsf{Hist} := \mathsf{Hist} \| (\mathsf{Corr}, i)$
return $\mathsf{sk}_i \leftarrow \mathcal{S}(\mathcal{L}(\mathsf{Hist}))$

Fig. 8: Ideal Experiment for DSE with A Curious Server

$\mathsf{Corr}\mathcal{O}$.[8] In the real experiment, oracle queries are answered by executing DSE algorithms. In the ideal one, they are answered by a simulator $\mathcal{S}$ who is given the leakage of the corresponding history just in time. An $\mathcal{L}$-adaptively-secure DSE requires that no PPT $\mathcal{A}$ can determine whether it is facing a real DSE.

Figs. 7 and 8 present the security experiments with a curious server. The adversary $\mathcal{A}$ can instruct any client to ask for the delegation of any keywords via $\mathsf{Delegate}\mathcal{O}$. With $\mathsf{Srch}\mathcal{O}$ (resp. $\mathsf{Updt}\mathcal{O}$), $\mathcal{A}$ can receive search (resp. update) tokens from any client on specified keywords. With $\mathsf{Corr}\mathcal{O}$, $\mathcal{A}$ can corrupt any honest clients and get their secret keys.

$\mathcal{A}$ might search or update as corrupt clients. If $\mathcal{A}$ is a curious server, it leaks nothing, as $\mathcal{A}$ has possessed both the database and the corrupt secret keys. For $\mathcal{A}$ being an honest server, Appendix D-A discusses its security experiments.

Definition 7 (Adaptive Security of DSE)**.** *A* DSE *scheme is $\mathcal{L}$-adaptively-secure with a(n) {curious, honest} server if, for any* PPT *adversary $\mathcal{A}$, there exists a* PPT *simulator $\mathcal{S}$:*

$$\left| \begin{array}{l} \Pr[\mathsf{Real\text{-}\{CS,HS\}}^{\mathsf{DSE}}_{\mathcal{A}}(1^\lambda) = 1] \\ -\Pr[\mathsf{Ideal\text{-}\{CS,HS\}}^{\mathsf{DSE}}_{\mathcal{A},\mathcal{S},\mathcal{L}}(1^\lambda) = 1] \end{array} \right| \leq \mathsf{negl}(\lambda).$$

Security experiments with a curious server (resp. an honest server) are defined in Figs. 7 and 8 (resp. Figs. 19 and 20).

Leakage Functions. Let $\mathcal{L}$ be any leakage function. We define $\mathcal{L}_{\mathsf{CS}}$ and $\mathcal{L}_{\mathsf{HS}}$ for the cases of *curious* server and *honest* server, respectively. The two settings are not necessarily comparable: DSE with an honest server might be adaptively secure with less leakage. With the notation in the ideal experiments (Figs. 8 and 20), we consider the DSE history as a sequence of events. We define Hist^t as $(\mathsf{Setup}, \epsilon) \| (\mathtt{h}_1, \mathrm{arg}_1) \| \cdots \| (\mathtt{h}_t, \mathrm{arg}_t)$, where for $j \in [t]$,

$$(\mathtt{h}_j, \mathrm{arg}_j) \in \left\{ \begin{array}{c} (\mathsf{Delegate}, (i, \mathcal{W}_{\mathbf{Q}}, \mathcal{W}_{\mathbf{U}})), (\mathsf{Corr}, i), \\ (\mathsf{Srch}, (i, W_{\mathbf{Q}})), (\mathsf{CorrSrch}, (i, W_{\mathbf{Q}})), \\ \left(\mathsf{Updt}, (i, \mathsf{ID}|_{W_{\mathbf{U}}})\right), \left(\mathsf{CorrUpdt}, (i, \mathsf{ID}|_{W_{\mathbf{U}}})\right) \end{array} \right\}.$$

We will omit $\mathsf{Hist}^0 = (\mathsf{Setup}, \epsilon)$ that leaks no information.

$\mathcal{L}(\mathsf{Hist}^t) = (\mathtt{h}_1, \overline{\mathrm{arg}_1}) \| \cdots \| (\mathtt{h}_t, \overline{\mathrm{arg}}_t)$ is the leaked version of Hist^t, where $\overline{\mathrm{arg}}_j$ for $j \in [t]$ captures the leaked information of the j-th event due to $\mathcal{L}$. We denote by $*$ any arguments in $\overline{\mathrm{arg}}_j$ that have not been revealed. Note that the arguments of an event might be hidden when it happens, but later revealed due to other events, *e.g.*, a search reveals the document identifiers hidden in previous updates. The effect of leakage functions could be accumulative, *e.g.*, a sequence of searches might gradually leak some keywords hidden in a prior update. Appendix D-B defines two leakage functions for DSE.

Forward Privacy. Forward privacy ensures that any update reveals no information about the keyword to be updated, even if it has been searched before. For DSE, the notion intuitively confines the leakage regarding the update tuple in the history. Following the convention [8], [27], forward privacy specifies the leakage $\mathcal{L}$ of an $\mathcal{L}$-adaptively-secure DSE.

Forward privacy in the multi-client setting needs to consider corruption [36] – a corrupt client with the secret key of the target keyword could trivially reveal it. We thus consider forward privacy on keywords not granted to corrupt clients.

Given the history Hist^t, we denote by $W^t_{\mathtt{corr}}$ the set of keywords searchable or updatable by all corrupt clients:

$$W^t_{\mathtt{corr}} := \left\{ w : \begin{array}{c} (\mathsf{Delegate}, (i, \mathcal{W}_{\mathbf{Q}}, \mathcal{W}_{\mathbf{U}})) \in \mathsf{Hist}^t \\ \wedge (\mathsf{Corr}, i) \in \mathsf{Hist}^t \wedge w \in \mathcal{W}_{\mathbf{Q}} \cup \mathcal{W}_{\mathbf{U}} \end{array} \right\}.$$

Formally, forward privacy of DSE (Definition 8) limits the leakage on $\left(\mathsf{Updt}, (i, \mathsf{ID}|_{W_{\mathbf{U}}})\right)$ to *at most* the writer identifier i and the document identifiers of the corrupt keywords $\mathsf{ID}|_{W^t_{\mathtt{corr}} \cap W_{\mathbf{U}}}$. Nothing about $W_{\mathbf{U}} \setminus W^t_{\mathtt{corr}}$ is revealed.

Definition 8 (Forward Privacy of DSE)**.** *An $\mathcal{L}$-adaptively-secure* DSE *is forward-private if, for any history* $\mathsf{Hist}^t = \mathsf{Hist}^{t-1} \| (\mathtt{h}_t, \mathrm{arg}_t)$ *with* $(\mathtt{h}_t, \mathrm{arg}_t) = \left(\mathsf{Updt}, (i, \mathsf{ID}|_{W_{\mathbf{U}}})\right)$*, the arguments $\overline{\mathrm{arg}}_t$ of the t-th entry in $\mathcal{L}(\mathsf{Hist}^t)$ is*

$$\overline{\mathrm{arg}}_t = \mathcal{L}'\left(\mathsf{Updt}, (i, \mathsf{ID}|_{W^t_{\mathtt{corr}} \cap W_{\mathbf{U}}})\right),$$

[8] Extra $\mathsf{CorrSrch}\mathcal{O}$ and $\mathsf{CorrUpdt}\mathcal{O}$ are needed with an honest server.

$\mathsf{Integrity}_{\mathcal{A}}^{\mathsf{DSE}}(1^\lambda)$

$I := \emptyset$ // client identifier set

$\hat{\mathsf{ID}} := \emptyset$ // plaintext copy of tuples from $\mathsf{Updt}\mathcal{O}$

$(\mathsf{msk}, \mathsf{st}, \mathsf{EDB}) \leftarrow \mathsf{DSE.Setup}(1^\lambda)$

$\mathbb{O} := \{\mathsf{Srch}\mathcal{O}, \mathsf{Updt}\mathcal{O}, \mathsf{Delegate}\mathcal{O}, \mathsf{Corr}\mathcal{O}\}$

$(i, W_{\mathsf{Q}}) \leftarrow \mathcal{A}^{\mathbb{O}}(\mathsf{st}, \mathsf{EDB})$

$\mathfrak{s} \leftarrow \mathsf{SrchTkn}(\mathsf{st}, \mathsf{sk}_i, W_{\mathsf{Q}})$

$\left(\mathsf{st}', \mathsf{EDB}', \mathsf{ID}|_{W_{\mathsf{Q}}}\right) \leftarrow \mathsf{Srch}(\mathsf{st}, \mathsf{EDB}, \mathfrak{s})$

return $\left(\hat{\mathsf{ID}}\Big|_{W_{\mathsf{Q}}} \not\subseteq \mathsf{ID}|_{W_{\mathsf{Q}}}\right)$

$\mathsf{Delegate}\mathcal{O}(i, \mathcal{W}_{\mathsf{Q}}, \mathcal{W}_{\mathsf{U}})$

ensure $i \notin I$

$I := I \cup \{i\}$

$(\mathcal{W}_{\mathsf{Q},i}, \mathcal{W}_{\mathsf{U},i}) := (\mathcal{W}_{\mathsf{Q}}, \mathcal{W}_{\mathsf{U}})$

$(\mathsf{st}', \mathsf{sk}_i) \leftarrow \mathsf{Delegate}(\mathsf{st}, \mathsf{msk}, \mathcal{W}_{\mathsf{Q}}, \mathcal{W}_{\mathsf{U}})$

return st'

$\mathsf{Srch}\mathcal{O}(i, W_{\mathsf{Q}})$

ensure $i \in I \wedge W_{\mathsf{Q}} \subseteq \mathcal{W}_{\mathsf{Q},i}$

return $\mathfrak{s} \leftarrow \mathsf{SrchTkn}(\mathsf{st}, \mathsf{sk}_i, W_{\mathsf{Q}})$

$\mathsf{Corr}\mathcal{O}(i)$

ensure $i \in I$; **return** sk_i

$\mathsf{Updt}\mathcal{O}\left(i, \mathsf{ID}|_{W_{\mathsf{U}}}\right)$

ensure $i \in I \wedge W_{\mathsf{U}} \subseteq \mathcal{W}_{\mathsf{U},i}$

$\mathfrak{u} \leftarrow \mathsf{UpdtTkn}\left(\mathsf{st}, \mathsf{sk}_i, \mathsf{ID}|_{W_{\mathsf{U}}}\right)$

$(\mathsf{st}', \mathsf{EDB}') \leftarrow \mathsf{Updt}(\mathsf{st}, \mathsf{EDB}, \mathfrak{u})$

for $w \in W_{\mathsf{U}}$ **do** $\hat{\mathsf{ID}}[w] := \hat{\mathsf{ID}}[w] \cup \mathsf{ID}[w]$

return $(\mathsf{st}', \mathsf{EDB}')$

Fig. 9: Integrity Experiment of DSE

with $\mathcal{L}'$ as a stateless function whose output solely depends on the inputs, and $W^t_{\mathtt{corr}}$ is the keyword set of corrupt clients.

Note that Definition 8 benchmarks against standard forward privacy in SSE [8], [27], definitely stronger than the epoch-based verison [36] that only protects updates at a new epoch.

D. Integrity

Integrity (subsuming correctness) is important for some M/M applications. Updates of honest clients will be reflected in the search results of other honest clients, even if malicious clients exist. Its definition (Fig. 9) checks whether the search result of keywords specified by the adversary contains *all* tuples updated via $\mathsf{Updt}\mathcal{O}$. It is not in the real-ideal formulation, as the leakage is not relevant.

Definition 9 (Integrity of DSE)**.** *A* DSE *scheme is said to have integrity if, for any* PPT *adversary $\mathcal{A}$, and* Integrity *as defined in Fig. 9: it holds that* $\Pr[\mathsf{Integrity}_{\mathcal{A}}^{\mathsf{DSE}}(1^\lambda) = 1] \leq \mathsf{negl}(\lambda)$.

V. DSE-$\mathcal{F}$: Forward-Private Framework

Following the idea in Section I-A, we construct DSE-$\mathcal{F}$ from a shiftable multi-recipient encryption scheme SME, an identity-based encryption scheme IBE, and a PRF family F.

Specifically, SME *encrypts the update counters* Uctr[9] of active keywords, which could be shifted per update. IBE *encrypts document identifiers* using the search counter $\mathsf{Sctr}[w]$ of updated keyword w as the identity. DSE-$\mathcal{F}$ generates the secret key $\mathsf{SK}_{\mathsf{EDB}}$ per keyword using F, which, with $\mathsf{Sctr}[w]$, is used to derive the index key $\mathsf{IK}_{\mathsf{EDB}}$. For any keyword w, its $\mathsf{IK}_{\mathsf{EDB}}[w]$ and $\mathsf{Uctr}[w]$ *determine the next address* via F to store the IBE-encrypted identifiers into EDB.

A search on w reveals its IBE decryption key $\mathsf{DK}_{\mathsf{IBE}}[w]$ and index key $\mathsf{IK}_{\mathsf{EDB}}[w]$. As each search increments $\mathsf{Sctr}[w]$ (as in Fig. 15) and which is used to generate the current $\mathsf{DK}_{\mathsf{IBE}}$ and $\mathsf{IK}_{\mathsf{EDB}}$, both keys are only valid for existing updates, ensuring forward privacy. To speed up searches, the server maintains auxiliary dictionaries SSI and CDB, recording respectively the

[9]To be compatible with SME, each Uctr entry is a group element started from the generator [1]. This is shiftable (by +[1]), which gives unique values like a numeric counter. For simplicity, we omit the group notation for Uctr (and so do counter i and "search starting index" SSI).

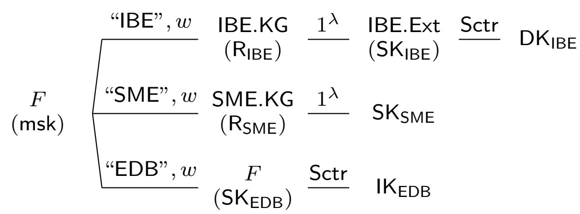


Fig. 10: Key Hierarchy (()/edge label is secret/public)

$\mathsf{Setup}(1^\lambda)$

$\mathsf{msk} \leftarrow\!\!\$\ \{0,1\}^\lambda, \mathcal{W} := \emptyset$

$\mathsf{pp}_{\mathsf{SME}} \leftarrow \mathsf{SME.Setup}(1^\lambda), \mathsf{ctx}_{\mathcal{W}} \leftarrow \mathsf{SME.Init}(1^\lambda)$

$\mathsf{Sctr} := \mathsf{PK}_{\mathsf{IBE}} := \mathsf{PK}_{\mathsf{SME}} := \emptyset$

$\mathsf{EDB} := \mathsf{CDB} := \mathsf{SSI} :=$ Empty dictionary

Initialize st as $((\mathsf{Sctr}, \mathsf{PK}_{\mathsf{IBE}}, \mathsf{PK}_{\mathsf{SME}})|_{\mathcal{W}}, \mathsf{ctx}_{\mathcal{W}})$

return $(\mathsf{msk}, \mathsf{st}, (\mathsf{EDB}, \mathsf{CDB}, \mathsf{SSI}))$

Fig. 11: Setup of DSE-$\mathcal{F}$

starting index of the next search (*i.e.*, $\mathsf{Uctr}[w]+1$) and previous results of a keyword (the latter appeared in existing designs[10]).

The owner uses master secret msk to generate the above keys for clients. Fig. 10 summarizes the hierarchy of keys, *e.g.*, $F(\mathsf{msk}, (\text{"IBE"}, w)) \rightarrow \mathsf{R}_{\mathsf{IBE}}$ and $\mathsf{IBE.KG}(1^\lambda, \mathsf{R}_{\mathsf{IBE}}) \rightarrow \mathsf{SK}_{\mathsf{IBE}}$.

A. Description

Figs. 11 to 14 depict DSE-$\mathcal{F}$ for active keyword space $\mathcal{W}$.[11] For any delegation on $(\mathcal{W}_{\mathsf{Q}}, \mathcal{W}_{\mathsf{U}})$, DSE-$\mathcal{F}$ assumes $\mathcal{W}_{\mathsf{Q}} \subseteq \mathcal{W}_{\mathsf{U}}$, *i.e.*, update-only or search-and-update rights. We discuss how to grant search-only rights in Section V-B.

Setup (Fig. 11). The database owner sets up DSE-$\mathcal{F}$ by picking a key msk and initializing the state st and the encrypted database EDB. $\mathcal{W}$ is empty before any delegation.

State st publishes: 1) search counters $\mathsf{Sctr}|_{\mathcal{W}}$; 2) IBE public keys $\mathsf{PK}_{\mathsf{IBE}}|_{\mathcal{W}}$; 3) SME public keys $\mathsf{PK}_{\mathsf{SME}}|_{\mathcal{W}}$; and 4) SME ciphertext $\mathsf{ctx}_{\mathcal{W}}$, that later encrypts update counters $\mathsf{Uctr}|_{\mathcal{W}}$.

[10]Most efficient SSE schemes reveal this information due to the search pattern. It has been leveraged in a similar way for efficiency [18], [27].

[11]Our description directly refers to the keywords for brevity. To hide the exact keywords from external attackers who access the global state, the owner can derive pseudonyms for the keywords and inform the eligible clients.

$\mathsf{Delegate}(\mathsf{st}, \mathsf{msk}, \mathcal{W}_{\mathbf{Q}}, \mathcal{W}_{\mathbf{E}})$

1 : **parse** st **as** $((\mathsf{Sctr}, \mathsf{PK}_{\mathsf{IBE}}, \mathsf{PK}_{\mathsf{SME}})|_{\mathcal{W}}, \mathsf{ctx}_{\mathcal{W}})$
2 : **ensure** $\mathcal{W}_{\mathbf{Q}} \subseteq \mathcal{W}_{\mathbf{E}}$
3 : **for** $w \in \mathcal{W}_{\mathbf{E}}$ **do** / (re)produce SME, IBE, and PRF keys for w
4 : $\quad \mathsf{R}_{\mathsf{SME}}[w] := F(\mathsf{msk}, (\text{“SME”}, w))$ / SME's seed
5 : $\quad \mathsf{R}_{\mathsf{IBE}}[w] := F(\mathsf{msk}, (\text{“IBE”}, w))$ / IBE's seed
6 : $\quad (\mathsf{PK}_{\mathsf{SME}}[w], \mathsf{SK}_{\mathsf{SME}}[w]) := \mathsf{SME.KG}(1^{\lambda}; \mathsf{R}_{\mathsf{SME}}[w])$
7 : $\quad (\mathsf{PK}_{\mathsf{IBE}}[w], \mathsf{SK}_{\mathsf{IBE}}[w]) := \mathsf{IBE.KG}(1^{\lambda}; \mathsf{R}_{\mathsf{IBE}}[w])$
8 : $\quad \mathsf{SK}_{\mathsf{EDB}}[w] := F(\mathsf{msk}, (\text{“EDB”}, w))$
9 : $\mathsf{ctx}_{\mathcal{W} \cup \mathcal{W}_{\mathbf{E}}} \leftarrow \mathsf{SME.Expand}\left(\mathsf{SK}_{\mathsf{SME}}|_{\mathcal{W}_{\mathbf{E}} \setminus \mathcal{W}}, \mathsf{ctx}_{\mathcal{W}}\right)$
10 : **for** $w \in \mathcal{W}_{\mathbf{E}} \setminus \mathcal{W}$ **do** $\mathsf{Sctr}[w] := 0$ / not-yet-active $\mathcal{W}_{\mathbf{E}} \setminus \mathcal{W}$
11 : $\mathcal{W} := \mathcal{W} \cup \mathcal{W}_{\mathbf{E}}$ / update-permitted keywords are now active
12 : $\mathsf{st}' := ((\mathsf{Sctr}, \mathsf{PK}_{\mathsf{IBE}}, \mathsf{PK}_{\mathsf{SME}})|_{\mathcal{W}}, \mathsf{ctx}_{\mathcal{W}})$
13 : $\mathsf{sk} := \left(\mathsf{SK}_{\mathsf{IBE}}|_{\mathcal{W}_{\mathbf{Q}}}, (\mathsf{SK}_{\mathsf{EDB}}, \mathsf{SK}_{\mathsf{SME}})|_{\mathcal{W}_{\mathbf{E}}}\right)$
14 : **return** $(\mathsf{st}', \mathsf{sk})$

Fig. 12: Delegation of DSE-$\mathcal{F}$

$\mathsf{UpdtTkn}\left(\mathsf{st}, \mathsf{sk}, \mathsf{ID}|_{W_{\mathbf{E}}}\right)$

1 : **parse** st **as** $((\mathsf{Sctr}, \mathsf{PK}_{\mathsf{IBE}}, \mathsf{PK}_{\mathsf{SME}})|_{\mathcal{W}}, \mathsf{ctx}_{\mathcal{W}})$
2 : **parse** sk **as** $\left(\mathsf{SK}_{\mathsf{IBE}}|_{\mathcal{W}_{\mathbf{Q}}}, (\mathsf{SK}_{\mathsf{EDB}}, \mathsf{SK}_{\mathsf{SME}})|_{\mathcal{W}_{\mathbf{E}}}\right)$
3 : **ensure** $W_{\mathbf{E}} \subseteq \mathcal{W}_{\mathbf{E}}$
4 : $\mathsf{Uctr}|_{W_{\mathbf{E}}} \leftarrow \mathsf{SME.Dec}\left(\mathsf{SK}_{\mathsf{SME}}|_{W_{\mathbf{E}}}, \mathsf{ctx}_{\mathcal{W}}\right)$
5 : $\mathsf{ctx}'_{\mathcal{W}} := \mathsf{SME.Enc}\left(\mathsf{PK}_{\mathsf{SME}}, [\Gamma_{W_{\mathbf{E}}}]\right)$
6 : $i := 1, \mathsf{ADDR} := \mathsf{VAL} :=$ Empty dictionary
7 : **for** $w \in W_{\mathbf{E}}$ **do** / derive “random” addresses for encrypted $\mathsf{ID}[w]$
8 : $\quad \mathsf{IK}_{\mathsf{EDB}}[w] := F(\mathsf{SK}_{\mathsf{EDB}}[w], \mathsf{Sctr}[w])$
9 : $\quad \mathsf{ADDR}[i] := F(\mathsf{IK}_{\mathsf{EDB}}[w], \mathsf{Uctr}[w] + 1)$
10 : $\quad \mathsf{VAL}[i] := \mathsf{IBE.Enc}(\mathsf{PK}_{\mathsf{IBE}}[w], \mathsf{Sctr}[w], \mathsf{ID}[w])$
11 : $\quad i := i + 1$
12 : **return** $\mathfrak{u} := (\mathsf{ctx}'_{\mathcal{W}}, \mathsf{ADDR}, \mathsf{VAL})$

$\mathsf{Updt}\,(\mathsf{st}, \mathsf{EDB}, \mathfrak{u})$

1 : **parse** st **as** $((\mathsf{Sctr}, \mathsf{PK}_{\mathsf{IBE}}, \mathsf{PK}_{\mathsf{SME}})|_{\mathcal{W}}, \mathsf{ctx}_{\mathcal{W}})$
2 : **parse** $\mathfrak{u}$ **as** $(\mathsf{ctx}'_{\mathcal{W}}, \mathsf{ADDR}, \mathsf{VAL})$
3 : **for** $i \in \{1, \ldots, |\mathsf{ADDR}|\}$ **do** $\mathsf{EDB}[\mathsf{ADDR}[i]] := \mathsf{VAL}[i]$
4 : $\mathsf{ctx}''_{\mathcal{W}} = \mathsf{SME.Shift}(\mathsf{ctx}_{\mathcal{W}}, \mathsf{ctx}'_{\mathcal{W}})$
5 : $\mathsf{st}' := \left((\mathsf{Sctr}, \mathsf{PK}_{\mathsf{IBE}}, \mathsf{PK}_{\mathsf{SME}})|_{\mathcal{W}}, \mathsf{ctx}''_{\mathcal{W}}\right), \mathsf{EDB}' := \mathsf{EDB}$
6 : **return** $(\mathsf{st}', \mathsf{EDB}')$

Fig. 13: Update of DSE-$\mathcal{F}$

EDB is a server-side dictionary for keyword-document tuples. Alongside EDB, two dictionaries auxiliary to Srch() are maintained: CDB *caches previous search results* of each keyword; SSI records the *starting index of the next search* on each keyword. One can view (CDB, SSI) as part of EDB.

Delegation (Fig. 12). The database owner provides a client with the secret key set sk regarding the delegated keywords. For search-permitted keywords $\mathcal{W}_{\mathbf{Q}}$ and update-permitted keywords $\mathcal{W}_{\mathbf{E}}$, sk includes: 1) IBE secret keys $\mathsf{SK}_{\mathsf{IBE}}|_{\mathcal{W}_{\mathbf{Q}}}$; 2) PRF keys $\mathsf{SK}_{\mathsf{EDB}}|_{\mathcal{W}_{\mathbf{E}}}$; and 3) SME secret keys $\mathsf{SK}_{\mathsf{SME}}|_{\mathcal{W}_{\mathbf{E}}}$.

(Line 3–8) For keyword $w \in \mathcal{W}_{\mathbf{E}}$, $\mathsf{R}_{\mathsf{IBE}}[w]$ and $\mathsf{R}_{\mathsf{SME}}[w]$ are obtained by evaluating $F(\mathsf{msk}, \cdot)$ on (“IBE”$, w$) and (“SME”$, w$), respectively. Then, $\mathsf{SK}_{\mathsf{IBE}}[w]$ and $\mathsf{SK}_{\mathsf{SME}}[w]$ could be derived by evaluating IBE.KG and SME.KG using $\mathsf{R}_{\mathsf{IBE}}[w]$ and $\mathsf{R}_{\mathsf{SME}}[w]$ as randomness, respectively. $\mathsf{SK}_{\mathsf{EDB}}[w]$ is derived from $F(\mathsf{msk}, (\text{“EDB”}, w))$.

(Line 9–11) For the newly delegated keywords $\mathcal{W}_{\mathbf{E}} \setminus \mathcal{W}$, the SME ciphertext is expanded to store their initial update counters; their search counters $\mathsf{Sctr}[w]$ are initialized to 0. The active keyword space $\mathcal{W}$ now includes the new keywords. Fig. 15 includes $W_{\mathbf{E}} = \{w_3\}$ into $\mathcal{W} = \{w_1, w_2\}$.

Update (Fig. 13). The writer specifies $\mathsf{ID}|_{W_{\mathbf{E}}}$, an identifier set of documents matching keywords in $W_{\mathbf{E}}$ for generating the update token via UpdtTkn(). The result includes a set of address-value tuples (ADDR, VAL) to be inserted into EDB via Updt(), and an SME ciphertext $\mathsf{ctx}'_{\mathcal{W}}$ for shifting its prior.

(Line 4–5 in UpdtTkn()) The writer decrypts the $W_{\mathbf{E}}$ slots of the SME ciphertext $\mathsf{ctx}_{\mathcal{W}}$ to get the update counters $\mathsf{Uctr}|_{W_{\mathbf{E}}}$. The writer encrypts the offset $[\Gamma_{W_{\mathbf{E}}}]$ as a new SME ciphertext $\mathsf{ctx}'_{\mathcal{W}}$. Recall that element of the characteristic vector $\Gamma_{W_{\mathbf{E}}}[w]$ is defined as 1 if $w \in W_{\mathbf{E}}$; 0, otherwise.

$\mathsf{SrchTkn}\,(\mathsf{st}, \mathsf{sk}, W_{\mathbf{Q}})$

1 : **parse** st **as** $((\mathsf{Sctr}, \mathsf{PK}_{\mathsf{IBE}}, \mathsf{PK}_{\mathsf{SME}})|_{\mathcal{W}}, \mathsf{ctx}_{\mathcal{W}})$
2 : **parse** sk **as** $\left(\mathsf{SK}_{\mathsf{IBE}}|_{\mathcal{W}_{\mathbf{Q}}}, (\mathsf{SK}_{\mathsf{EDB}}, \mathsf{SK}_{\mathsf{SME}})|_{\mathcal{W}_{\mathbf{E}}}\right)$
3 : **ensure** $W_{\mathbf{Q}} \subseteq \mathcal{W}_{\mathbf{Q}}$
4 : $\mathsf{Uctr}|_{W_{\mathbf{Q}}} \leftarrow \mathsf{SME.Dec}\left(\mathsf{SK}_{\mathsf{SME}}|_{W_{\mathbf{Q}}}, \mathsf{ctx}_{\mathcal{W}}\right)$
5 : **for** $w \in W_{\mathbf{Q}}$ **do**
6 : $\quad \mathsf{DK}_{\mathsf{IBE}}[w] := \mathsf{IBE.Ext}(\mathsf{SK}_{\mathsf{IBE}}[w], \mathsf{Sctr}[w])$
7 : $\quad \mathsf{IK}_{\mathsf{EDB}}[w] := F(\mathsf{SK}_{\mathsf{EDB}}[w], \mathsf{Sctr}[w])$
8 : **return** $\mathfrak{s} := \left((\mathsf{DK}_{\mathsf{IBE}}, \mathsf{IK}_{\mathsf{EDB}}, \mathsf{Uctr})|_{W_{\mathbf{Q}}}\right)$

$\mathsf{Srch}\,(\mathsf{st}, (\mathsf{EDB}, \mathsf{CDB}, \mathsf{SSI}), \mathfrak{s})$

1 : **parse** st **as** $((\mathsf{Sctr}, \mathsf{PK}_{\mathsf{IBE}}, \mathsf{PK}_{\mathsf{SME}})|_{\mathcal{W}}, \mathsf{ctx}_{\mathcal{W}})$
2 : **parse** $\mathfrak{s}$ **as** $\left((\mathsf{DK}_{\mathsf{IBE}}, \mathsf{IK}_{\mathsf{EDB}}, \mathsf{Uctr})|_{W_{\mathbf{Q}}}\right)$
3 : $\mathsf{ID}|_{W_{\mathbf{Q}}} := \emptyset$
4 : **for** $w \in W_{\mathbf{Q}}$ **do**
5 : $\quad$ **if** $\mathsf{Sctr}[w] = 0$ **then** $\mathsf{SSI}[w] := 1$ / start of w's first search
6 : $\quad$ **else** $\mathsf{ID}[w] := \mathsf{CDB}[w]$ / obtain previous search results of w
7 : $\quad$ **for** $i \in \{\mathsf{SSI}[w], \ldots, \mathsf{Uctr}[w]\}$ **do**
8 : $\quad\quad \mathsf{addr} \leftarrow F(\mathsf{IK}_{\mathsf{EDB}}[w], i)$
9 : $\quad\quad \mathsf{val} \leftarrow \mathsf{IBE.Dec}(\mathsf{DK}_{\mathsf{IBE}}[w], \mathsf{EDB}[\mathsf{addr}])$
10 : $\quad\quad \mathsf{ID}[w] := \mathsf{ID}[w] \cup \mathsf{val}$
11 : $\quad \mathsf{SSI}[w] := \mathsf{Uctr}[w] + 1$ / start of w's next search
12 : $\quad \mathsf{CDB}[w] := \mathsf{ID}[w]$ / cache the current search result of w
13 : $\mathsf{EDB}' := \mathsf{EDB}, \mathsf{CDB}' := \mathsf{CDB}, \mathsf{SSI}' := \mathsf{SSI}$
14 : $\mathsf{st}' := \left((\mathsf{Sctr} + \Gamma_{W_{\mathbf{Q}}}, \mathsf{PK}_{\mathsf{IBE}}, \mathsf{PK}_{\mathsf{SME}})|_{\mathcal{W}}, \mathsf{ctx}_{\mathcal{W}}\right)$
15 : **return** $\left(\mathsf{st}', (\mathsf{EDB}', \mathsf{CDB}', \mathsf{SSI}'), \mathsf{ID}|_{W_{\mathbf{Q}}}\right)$

Fig. 14: Search of DSE-$\mathcal{F}$

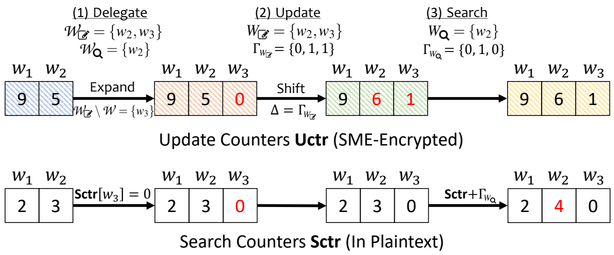


Fig. 15: Example of DSE-$\mathcal{F}$ Counters ($\mathcal{W} = \{w_1, w_2\}$)

(Line 7–11) Let i be a running index of ADDR. For $w \in W_{\text{✎}}$, address $\mathsf{ADDR}[i]$ for storing $\mathsf{VAL}[i]$ (an IBE-encryption of $\mathsf{ID}[w]$ under $\mathsf{Sctr}[w]$) is $F(\mathsf{IK}_{\mathsf{EDB}}[w], \mathsf{Uctr}[w] + 1)$, where $\mathsf{IK}_{\mathsf{EDB}}[w]$ is determined by $\mathsf{SK}_{\mathsf{EDB}}[w]$ and current $\mathsf{Sctr}[w]$.

The update token $\mathfrak{u}$ consists of $(\mathsf{ctx}'_{\mathcal{W}}, \mathsf{ADDR}, \mathsf{VAL})$.

(Line 3–5 in Updt()) The server sets $\mathsf{EDB}[\mathsf{ADDR}[i]]$ as $\mathsf{VAL}[i]$ for $i \in \{1, \ldots, |\mathsf{ADDR}|\}$. It obtains new $\mathsf{ctx}''_{\mathcal{W}}$ for st by shifting $\mathsf{ctx}_{\mathcal{W}}$ with $\mathsf{ctx}'_{\mathcal{W}}$, which essentially increases $\mathsf{Uctr}|_{W_{\text{✎}}}$ by 1. $\mathsf{Uctr}|_{\mathcal{W} \setminus W_{\text{✎}}}$ is unchanged but with its slots refreshed. Part (2) of Fig. 15 illustrates the case for $W_{\text{✎}} = \{w_2, w_3\}$.

Search (Fig. 14). Via SrchTkn(), the reader prepares for the server the IBE decryption keys and index keys for keywords $W_{\text{🔍}}$ to be searched, with respect to the global state st.

(Line 4 in SrchTkn()) The reader decrypts the $W_{\text{🔍}}$ slots of the SME ciphertext $\mathsf{ctx}_{\mathcal{W}}$ for the update counters $\mathsf{Uctr}|_{W_{\text{🔍}}}$.

(Line 5–8) For $w \in W_{\text{🔍}}$, with current $\mathsf{Sctr}[w]$, the reader derives $\mathsf{DK}_{\mathsf{IBE}}[w]$ via $\mathsf{IBE.Ext}(\mathsf{SK}_{\mathsf{IBE}}[w], \mathsf{Sctr}[w])$ and generates $\mathsf{IK}_{\mathsf{EDB}}[w]$ via $F(\mathsf{SK}_{\mathsf{EDB}}[w], \mathsf{Sctr}[w])$. With $\mathsf{Uctr}|_{W_{\text{🔍}}}$, they form the search token $\mathfrak{s}$ for documents that match $w \in W_{\text{🔍}}$.

(Line 3–6 in Srch()) The server initializes an identifier set $\mathsf{ID}|_{W_{\text{🔍}}}$. For $w \in W_{\text{🔍}}$, if this is the first search on w, it initializes $\mathsf{SSI}[w] := 1$ and starts from the first position; otherwise, results from prior searches on w are obtained from the cache $\mathsf{CDB}[w]$ and put into $\mathsf{ID}[w]$.

(Line 7–13) For $w \in W_{\text{🔍}}, i \in \{\mathsf{SSI}[w], \ldots, \mathsf{Uctr}[w]\}$, the server gets the dictionary entry at $F(\mathsf{IK}_{\mathsf{EDB}}[w], i)$. The server decrypts such an entry with $\mathsf{DK}_{\mathsf{IBE}}[w]$ and inserts the result into $\mathsf{ID}[w]$. $\mathsf{ID}|_{W_{\text{🔍}}}$ will be returned to the reader.

After retrieving each $w \in W_{\text{🔍}}$, the server updates $\mathsf{SSI}[w]$ to $\mathsf{Uctr}[w] + 1$ as the starting index of the next search on w. The server also caches the searched entries per (pseudonym of) keyword by updating $\mathsf{CDB}[w]$ with $\mathsf{ID}[w]$. SSI and CDB help avoid retrieving from scratch. Any reader has no need to reset $\mathsf{Uctr}[w]$ via shifting (only $\mathsf{IK}_{\mathsf{EDB}}[w]$ will be changed with $\mathsf{Sctr}[w]$ for forward privacy).

(Line 14) The server refreshes the global state st by incrementing the search counters of $W_{\text{🔍}}$ (as in Fig. 15).

B. Additional Considerations

Our description is under the basic setting of a single database owner. We discuss here easy generalizations addressing concerns in a multi-user system, such as revocation and concurrency control. Apart from employing existing orthogonal techniques, we describe specific aspects of our scheme.

Enforcing Search-Only Right. DSE-$\mathcal{F}$, as described, grants the updating right with the searching right since a client who can search for keyword w has obtained all secrets required to update w. To decouple for the search-only right, the database owner could additionally grant a signing key (specific to w) to those who could update but not those who could only search.

Distributed Owners. Multiple owners can run multiple DSE instances. If a "root" owner overseeing all owners is desired, IBE can be replaced with hierarchical IBE. Other keys are PRF-derived as Fig. 10 shows. Threshold cryptography can distribute the power of a single owner. Our PRF instantiation (Section VII) features a simple threshold extension.

Revocation. We can revoke via twisting/upgrading our building blocks related to the key material of each entity. PRF keys are for locating/creating dictionary entries (reader and writer). They are, in turn, derived from PRF. When deriving the PRF keys, a client identity can be taken as input. Constrained PRF could be of help. SME keys are for counters (reader and writer), and IBE keys are for payloads (only reader). We can employ techniques in updatable encryption and revocable IBE.

Synchronizing "Simultaneous" Updates. When multiple writers update different keywords simultaneously, they do not need to synchronize because they write to distinct pseudorandom locations, and the needed SME shifting can be executed in an arbitrary order. However, for two writers holding the same update counter value for the same keyword, they instruct the server to update its encrypted database at the same address. To resolve this potential clash, the server can designate one of the updates as the first and instruct the other to re-derive the address using the now-incremented counters. (It is worth noting that the SME component of the update token can still be utilized.) Unfortunately, such clash identification degrades privacy. Resolving the tension between oblivious updates and synchronization is left as future work. That said, our optimized scheme, to be described in Section V-D, can reduce some of the synchronization requirements.

C. Security Analysis and Extensions

DSE-$\mathcal{F}$ is adaptively forward-private against a curious or an honest server (both with a set of corrupt clients) with leakage functions $\mathcal{L}_{\mathsf{CS}}$ and $\mathcal{L}_{\mathsf{HS}}$ (defined in Appendix D-C), respectively. Theorem 2 asserts the security of DSE-$\mathcal{F}$ with its proof in Appendix D-C. In either case, the delegation only leaks pseudonyms of new active keywords from global states.

For $\mathcal{L}_{\mathsf{CS}}$, corruption on any client exposes their secret key, leaking documents matching the search-permitted keywords and the occasions when the update-permitted keywords are updated. A search exposes the search pattern and matching document identifiers. SSI and CDB cause no extra leakage as the server is allowed to learn such information in a search. For any keywords that are neither searchable nor updatable by corrupt clients, updates on them are put in addresses pseudorandom to the server and remain encrypted until the next event, fulfilling forward privacy.

The information leaked in $\mathcal{L}_{\mathsf{HS}}$ is much less than $\mathcal{L}_{\mathsf{CS}}$ due to the assumption of an honest server. In particular, only corrupt searches leak the identifiers of matching documents.

Theorem 2. *If F,* IBE *and* SME *are all non-committing-secure in their sense,* DSE-$\mathcal{F}$ *is $\mathcal{L}_{\mathsf{CS}}$ (resp. $\mathcal{L}_{\mathsf{HS}}$) adaptively secure and forward private with a curious (resp. an honest) server.*

Anonymity. As DSE-$\mathcal{F}$ tokens do not involve the client information, DSE-$\mathcal{F}$ hides the client who performs operations if a client connects using an anonymous channel. It explains $*$ for client i in events related to honest clients in Appendix D, *i.e.*, the client identifier i remains hidden.

Response Hiding. DSE-$\mathcal{F}$ is response-revealing – returning search results in plaintext. For higher security (*e.g.*, avoid leaking the identifiers of matching documents in searches), we could make DSE-$\mathcal{F}$ response-hiding by withholding the IBE decryption key during searches. The server could only return IBE ciphertexts of the search results. The server still archives these encrypted results as per keywords for subsequent searches. Response-hiding DSE-$\mathcal{F}$ hides leakage of document identifiers except those from corruption. It also helps *backward privacy* and is essential for *volume-hiding*.

Appendix B equips DSE-$\mathcal{F}$ with advanced features (*e.g.*, backward-private, volume-hiding) rarely realized for M/M.

D. Efficiency Analysis and Optimization

The database owner only needs to keep the master secret key. The server stores the tuple $(\mathsf{st}, \mathsf{EDB}, \mathsf{CDB}, \mathsf{SSI})$ – st and SSI are linear in the number of active keywords, while EDB and CDB are linear in the plaintext database size. A client maintains one state for each keyword they could search or update, which is common in forward-private SSE [8].

For delegation, the database owner generates secret keys of keywords to be delegated. It also expands the SME ciphertext and the active keyword space if $\mathcal{W}_{\text{✎}} \setminus \mathcal{W} \neq \emptyset$. As $\mathcal{W}_{\text{🔍}} \subseteq \mathcal{W}_{\text{✎}}$, the overhead for delegation is $\mathcal{O}(|\mathcal{W}_{\text{✎}}|)$.

Either searching on $W_{\text{🔍}}$ or updating on $W_{\text{✎}}$ requires the client to decrypt the needed update counters (with an overhead of $\mathcal{O}(|W_{\text{🔍}}|)$ or $\mathcal{O}(|W_{\text{✎}}|)$). For an update, the writer shifts an SME ciphertext in $\mathcal{O}(|\mathcal{W}|)$. The number of active keywords $|\mathcal{W}|$ is typically sublinear in the number of keyword-document tuples in real-world databases. The writer determines addresses and values by PRF and IBE encryption in $\mathcal{O}(|W_{\text{✎}}|)$. The server simply inserts tuples to EDB with an overhead of $\mathcal{O}(|W_{\text{✎}}|)$.

To search, the reader outputs IBE decryption keys and PRF keys in $\mathcal{O}(|W_{\text{🔍}}|)$. Owing to the server-side $(\mathsf{SSI}, \mathsf{CDB})$, the update counters are not reset during searches, avoiding $\mathcal{O}(|\mathcal{W}|)$ for shifting. The server executes PRF evaluation and IBE decryption as many times as the number of current matches (*i.e.*, starting from SSI for $W_{\text{🔍}}$). Its complexity is bounded by the sum of matching updates, *i.e.*, $\mathcal{O}(\sum \mathsf{Uctr}|_{W_{\text{🔍}}})$, the *optimal* search efficiency [8] we strive for.

To search for all identifiers of documents matching a keyword w, the search overheads of DSE-$\mathcal{F}$ and FP-HSE [36] are $\mathcal{O}(\mathsf{Uctr}[w])$ and $\mathcal{O}(\mathsf{Uctr}[w] + \sum_{i \in S} |W_i|)$, respectively, where W_i refers to the keywords ever written by writer i in set S of writers who have updated w. DSE-$\mathcal{F}$ beats FP-HSE in complexity, yet the $\mathcal{O}(\mathsf{Uctr}[w])$ part of HSE only involves symmetric-key operations. Below discusses how to minimize public-key operations in DSE-$\mathcal{F}$ to make its search faster than FP-HSE in both theory and practice.

Less Public-Key Operations via Hybrid Encryption. Viewing DSE as an access control system, it is not difficult to imagine that the writer can use standard tricks such as hybrid encryption for the "content" by using only one public-key operation.

When a writer updates a keyword, the initial entry can be an IBE encryption of a secret K as keying material of SSE, *e.g.*, the writer could use a PRF with key K to derive both the address and the symmetric-key encryption (SKE) key for the SSE encrypted data structure beyond the first entry. Document identifiers can then be encrypted using SKE instead of IBE. Note that each writer needs to IBE-encrypt once; otherwise, writers can unwrap ciphertexts of others if only SKE is used.

Correspondingly, searching performs one IBE decryption for each contributing writer. The search remains sublinear, but the number of public-key operations is significantly reduced.

Readers run either IBE decryption for addresses from the index key $\mathsf{IK}_{\mathsf{EDB}}$ or SKE decryption. For privacy against the server, both IBE and SKE ciphertexts should be pseudorandom. Special elliptic curves [17] should be used for IBE.

This optimization has other benefits. Each client executes a DSE-update only once to settle its key materials. They then *independently* insert tuples using their own keys. Both changes loosen the synchronization requirement.

VI. DSE-$\mathcal{I}$: Instrument for Integrity

A. Description

DSE-$\mathcal{I}$ enhances DSE-$\mathcal{F}$ with integrity, *i.e.*, the global state must be correctly modified so that later honest clients can operate as expected. DSE-$\mathcal{I}$ requires that any token must come with a proof, with which the server checks the token before executing it. DSE-$\mathcal{I}$ uses an equivocable commitment scheme $\varphi.(\mathsf{Setup}, \mathsf{Com})$ [3] and an argument of knowledge $\Pi.(\mathsf{Setup}, \mathsf{Prove}, \mathsf{Vf})$ [28].

Setup and Delegation. The database owner of DSE-$\mathcal{I}$ extra sets up Π and φ with $\Pi.\mathsf{Setup}()$ and $\varphi.\mathsf{Setup}()$, respectively. The state st includes the common reference string crs of Π.

For each $w \in \mathcal{W}_{\text{✎}}$, an additional randomness $\mathsf{R}_\varphi[w]$ is derived from $F(\mathsf{msk}, (\text{“}\varphi\text{”}, w))$. $\mathsf{R}_\varphi[w]$ is used to commit $\mathsf{SK}_{\mathsf{EDB}}[w]$ as $\mathsf{COM}[w] := \varphi.\mathsf{Com}(\mathsf{SK}_{\mathsf{EDB}}[w]; \mathsf{R}_\varphi[w])$. st will include $\mathsf{COM}|_{\mathcal{W}}$. As $\mathsf{SK}_{\mathsf{EDB}}[w]$ is the PRF key (for $\mathsf{IK}_{\mathsf{EDB}}[w]$) with its commitment publicly available. We specifically denote its associated PRF by pkPRF to differentiate it, for the convenience of proving the knowledge of $\mathsf{SK}_{\mathsf{EDB}}$ with Π.

Update. The writer runs $\Pi.\mathsf{Prove}()$ on the language below for a well-formedness proof π of $\mathsf{ctx}'_{\mathcal{W}}$ in $\mathfrak{u}$. $(\mathsf{ADDR}, \mathsf{VAL})$ is omitted as it will not affect the global data structure; *e.g.*, ADDR is related to how many entries a writer can occupy, which can be enforced by (anonymous) rate-limiting.

$$\left\{\begin{array}{l}(\mathsf{st},\mathsf{ctx}'_{\mathcal{W}}):\exists\left(\mathsf{SK}_{\mathsf{SME}}|_{W_{✎}},\Gamma_{W_{✎}},r\right)\ \text{s.t.}\\ \left\{\begin{array}{l}1.\ \vec{0}_{\mathcal{W}}=\Gamma_{W_{✎}}\circ(\Gamma_{W_{✎}}-\vec{1}_{\mathcal{W}})\\ 2.\ [\vec{0}_{\mathcal{W}}]=\Gamma_{W_{✎}}\circ(\mathsf{PK}_{\mathsf{SME}}-[\mathsf{SK}_{\mathsf{SME}}])\\ 3.\ \mathsf{ctx}'_{\mathcal{W}}=\mathsf{SME.Enc}(\mathsf{PK}_{\mathsf{SME}},[\Gamma_{W_{✎}}];r)\end{array}\right.\end{array}\right\}$$

The first two relations ensure the binary vector nature of $\Gamma_{W_{✎}}$ and the knowledge of the SME secret keys. The third ensures that $\mathsf{ctx}'_{\mathcal{W}}$ encrypts offset $[\Gamma_{W_{✎}}]$. The server executes $\Pi.\mathsf{Vf}()$ on π. After verifying, it shifts $\mathsf{ctx}_{\mathcal{W}}$ with $\mathsf{ctx}'_{\mathcal{W}}$.

Search. The reader uses Π to output a well-formedness proof π of the index key $\mathsf{IK}_{\mathsf{EDB}}|_{W_{\mathsf{Q}}}$ in the search token $\mathfrak{s}$:

$$\left\{\begin{array}{l}\left(\mathsf{st},\mathsf{IK}_{\mathsf{EDB}}|_{W_{\mathsf{Q}}}\right):\exists\left((\mathsf{SK}_{\mathsf{EDB}},\mathsf{R}_{\varphi})|_{W_{\mathsf{Q}}}\right)\ \text{s.t.}\ \forall w\in W_{\mathsf{Q}},\\ \quad\left\{\begin{array}{l}1.\ \mathsf{COM}[w]=\varphi.\mathsf{Com}(\mathsf{SK}_{\mathsf{EDB}}[w];\mathsf{R}_{\varphi}[w])\\ 2.\ \mathsf{IK}_{\mathsf{EDB}}[w]=\mathsf{pkPRF}(\mathsf{SK}_{\mathsf{EDB}}[w],\mathsf{Sctr}[w])\end{array}\right.\end{array}\right\}.$$

It ensures that for $w\in W_{\mathsf{Q}}$, $\mathsf{IK}_{\mathsf{EDB}}[w]$ is a valid index key derived from $\mathsf{SK}_{\mathsf{EDB}}[w]$ regarding current $\mathsf{Sctr}[w]$. $\mathsf{DK}_{\mathsf{IBE}}|_{W_{\mathsf{Q}}}$ can be verified by IBE operations. $\mathsf{Uctr}|_{W_{\mathsf{Q}}}$ is auxiliary and omitted: the server can search with $\mathsf{IK}_{\mathsf{EDB}}|_{W_{\mathsf{Q}}}$ till an invalid address. After verifying, the server adds $\Gamma_{W_{\mathsf{Q}}}$ to Sctr.

Efficiency. The complexity of an argument system is often upper bounded by the size of the statement and the witness. Thus, the overheads caused by Π in DSE-$\mathcal{I}$ are $\mathcal{O}(|W_{\mathsf{Q}}|)$ for searching and $\mathcal{O}(|\mathcal{W}|)$ for updating. DSE-$\mathcal{I}$ retains the same update/search complexities and enjoys the same optimization (Section V-D) as DSE-$\mathcal{F}$.

B. Security and Integrity Analyses

DSE-$\mathcal{I}$ is adaptively secure with the same leakage as DSE-$\mathcal{F}$, as defined in Appendix D. The proof of Theorem 3 involves two more hybrids than Theorem 2: 1) call φ simulator to generate COM and use φ simulator to invert $\mathsf{COM}[w]$ with respect to $\mathsf{SK}_{\mathsf{EDB}}[w]$, when $(\mathsf{SK}_{\mathsf{EDB}}[w],\mathsf{R}_{\varphi}[w])$ needs to be revealed later (*e.g.*, $\mathsf{Corr}\mathcal{O}$); 2) replace crs with its trapdoor variant and invoke Π simulator (with the knowledge of trapdoors and statements) to simulate π.

Theorem 3. *If F, IBE, and SME are all non-committing in their sense, φ is equivocable, Π is a zero-knowledge argument of knowledge, DSE-$\mathcal{I}$ is $\mathcal{L}_{\mathsf{CS}}$ (resp. $\mathcal{L}_{\mathsf{HS}}$) adaptively secure and forward private with a curious (resp. an honest) server.*

In DSE-$\mathcal{I}$, the server only executes well-formed update tokens and search tokens. It guarantees that update counters Uctr (encrypted as $\mathsf{ctx}'_{\mathcal{W}}$) and search counters Sctr are modified correctly. In this way, any update from an honest client could be retrieved by others, ensuring integrity. Theorem 4 asserts the integrity of DSE-$\mathcal{I}$. The core idea of its proof is to embed the challenge from φ or SME to one of the $|\mathcal{W}|$ keywords and use the extractor of Π to break the security of either.

Theorem 4. *If Π is an argument of knowledge, φ is hiding and binding, and SME is CPA-secure, DSE-$\mathcal{I}$ has integrity.*

TABLE II: Delegation Time (s) for $(|\mathcal{W}_{\mathsf{Q}}|,|\mathcal{W}_{✎}|)=(\ell,\ell)$

Scheme \ ℓ	200	400	600	800	1000
DSE-$\mathcal{F}$	1.27	2.62	3.81	5.10	6.34
DSE-$\mathcal{I}$	1.61	3.19	4.90	6.39	8.13

VII. Experiments

We implement DSE-$\mathcal{F}$ and DSE-$\mathcal{I}$ in Python with Charm-Crypto library[12] for cryptographic and group operations (with MNT224 curve). We use a desktop with Intel Core i7-4790 3.60GHz CPU and 16GB RAM. Our implementations use Pedersen commitment, our SME, Boneh–Franklin IBE [7], HMAC-SHA-256 (for PRF), the argument system from Lai *et al.* [28], and the hybrid technique in Section V-D. We set $\mathsf{pkPRF}(\mathsf{sk},m)$ as $H(m)^{\mathsf{sk}}$, where H is a full domain hash built upon HMAC-SHA-256. Note that our evaluation involves *no* CDB-*cache* for clarity and fairness.

A. Evaluation with Synthetic Datasets

Table II reports delegation time for varying $\mathcal{W}_{\mathsf{Q}}$ and $\mathcal{W}_{✎}$.

Search. Fig. 16a shows the time of searching for a keyword that matches $|\mathsf{ID}|_{W_{\mathsf{Q}}}|\in\{2000,4000,6000\}$ documents uniformly contributed by 10 writers, where the database size varies from 2^{16} to 2^{20} with a fixed active keyword space size $|\mathcal{W}|=1000$. Fig. 16b shows the search time of a keyword with the same results as Fig. 16a, under a varying $|\mathcal{W}|\in\{200,400,600,800,1000\}$ and a fixed database size 2^{16}. The search time of DSE-$\mathcal{F}$/DSE-$\mathcal{I}$ is *independent* of the database size and $|\mathcal{W}|$.

Fig. 16c measures the searches on a keyword that match 9000 documents uniformly contributed by 10 to 50 writers. The search time is linear in the number of contributive writers.

Fig. 16d shows the influence of the size of search results $\mathsf{ID}|_{W_{\mathsf{Q}}}$ (uniformly written by 10 writers) on the search time. We vary $|\mathsf{ID}|_{W_{\mathsf{Q}}}|$ from $2\cdot10^3$ to 10^4. DSE search time grows linear with the number of matches, like most SSE schemes.

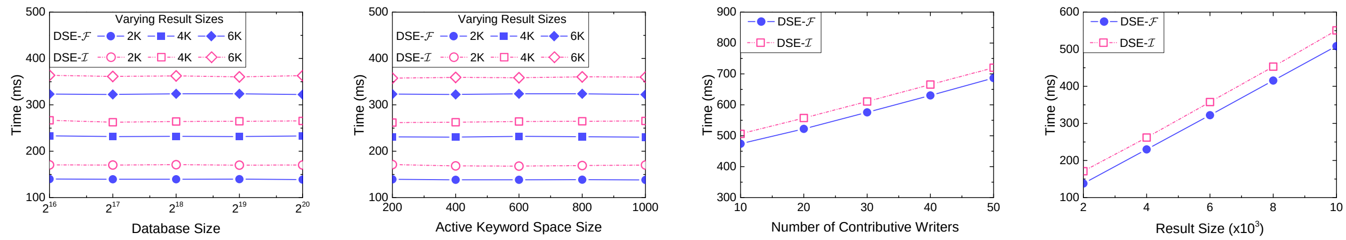
Note that any of the above searches are done within 1s, which is quite efficient. The gap between DSE-$\mathcal{F}$ and DSE-$\mathcal{I}$ in Fig. 16 reflects the time needed by Π in DSE-$\mathcal{I}$ for integrity.

Update. We measure the time of updating multiple keywords in one batch (with $|W_{✎}|\in\{1,5,10,15\}$) over varying active keyword space $|\mathcal{W}|\in\{200,400,600,800,1000\}$. Each keyword of the updates is related to 1000 document identifiers to be inserted, *e.g.*, the result of the orange curve in Fig. 17 should be divided by 15000 for the update time of a single tuple (that is, 0.08ms for DSE-$\mathcal{F}$ and 6.46ms for DSE-$\mathcal{I}$).

Fig. 17a shows that the update time of DSE-$\mathcal{F}$ grows with the number of update tuples and $|\mathcal{W}|$. Fig. 17b indicates that a DSE-$\mathcal{I}$ update is dominated by $|\mathcal{W}|$ due to Π. The time needed by Π is almost independent of the number of tuples in a single update. DSE-$\mathcal{I}$ could be faster per tuple for a larger batch.

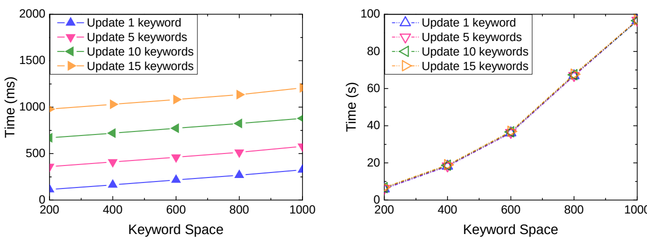
As indicated by our evaluations over synthetic datasets, DSE-$\mathcal{F}$ is efficient for daily cloud storage usage expecting real-time retrieval of frequent-updated data. DSE-$\mathcal{I}$ is more applicable for archiving-and-auditing applications, batching a large volume of data beforehand. For integrity, the complexity

[12]https://jhuisi.github.io/charm



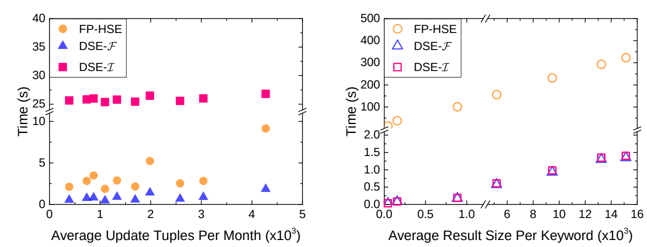
(a) Varying Database Size (b) Varying Active Keyword Space (c) Varying Number of Writers (d) Varying Result Size

Fig. 16: Search Performance (Synthetic Datasets)



(a) DSE-$\mathcal{F}$ (b) DSE-$\mathcal{I}$

Fig. 17: Update Performance (Synthetic Datasets)



(a) Update Time Per Client (b) Search Time Per Keyword

Fig. 18: Performance over Enron Dataset

of zero-knowledge proof is inherent, *e.g.*, 10s appears to be the baseline considering the performance of the latest malicious-secure sharing system [13] using multiple powerful servers.

B. Evaluation with the Enron Dataset

We evaluate DSE over Enron dataset[13] as both daily cloud storage (*i.e.*, email) and archive (*i.e.*, auditing happened on it), compared with FP-HSE [36], the state-of-the-art M/S solution.

Dataset. We pick around 0.5M emails sent by 146 clients from Jan 1999 to Jun 2002. We extract the top 500 most frequent ones as the active keyword space, generating around 4.1M keyword-email tuples. Averagely, a client has 460 keywords, 3483 emails, and 27864 keyword-email tuples. Tagging each email with its date, each client uploads emails to the server monthly and searches are regularly executed per half year.

Update. We delegate the updating rights to each client before measuring update time. The average delegation time per client is 2.77s (resp. 3.42s) for DSE-$\mathcal{F}$ (resp. DSE-$\mathcal{I}$).

Fig. 18a picks 10 random clients and plots their average update tuples and update time per month (excluding months with no email). For epoch-based forward privacy, each FP-HSE client needs to rebuild according to the search schedule, taking linear time in the number of ever-written keywords by a client. We amortize it into the update time of FP-HSE, accounting for its large fluctuation and latency compared to DSE-$\mathcal{F}$. DSE-$\mathcal{I}$ requires more update time, yet each DSE-$\mathcal{I}$ client completes updates within 30s, which is affordable for daily backups.

Search. We search for 10 random keywords every six months over the entire encrypted database. For each search, Fig. 18b shows the search time and result size averaged over keywords.

DSE-$\mathcal{F}$ and DSE-$\mathcal{I}$ are significantly faster than FP-HSE. The number of public-key operations in the (optimized) DSE search is only linear in the number of *writers* updating the searched keywords, while FP-HSE has to make decryption attempts on a token set linear in the number of ever-written *keywords* by all writers. Concretely, the search time of DSE-$\mathcal{F}$ and DSE-$\mathcal{I}$ is more than $240\times$ shorter than that of FP-HSE.

VIII. Concluding Remarks

Searchable encryption has been gaining commercial interest. However, limitations such as linear search time in PEKS and the lack of multi-client support in SSE pose hindrances to adoption. This paper bridges the gaps between these two classical notions by proposing delegatable searchable encryption (DSE), a kind of keyword-based access control system tailored for security and efficiency concerns in multi-client encrypted search, notably, achieving optimal search and forward privacy.

Multiple writers in DSE write to the same linked list in the encrypted database for optimal search across multiple readers. In contrast, hybrid searchable encryption (HSE), the only prior sublinear multi-writer scheme, links the linked lists of different writers with ciphertexts of identity-coupling key-aggregate encryption (ICKAE), which requires public-key operations for trial decryption across different encrypted keywords. Nevertheless, ICKAE allows the reader to confine the search scope to an arbitrary writer subset. Also, anyone can be a writer in HSE without contacting the reader beforehand. It seems to preclude state synchronization tricks for forward privacy. HSE circumvents it by relying on a global clock for epoch-based forward privacy, which is weaker than DSE.

Conclusively, DSE presents a more secure and efficient alternative for applications where one-time delegation is required or acceptable. We expect the new DSE notion could inspire the future development of M/M searchable encryption.

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

References

[1] S. Agrawal, R. Garg, N. Kumar, and M. Prabhakaran, "A practical model for collaborative databases: Securely mixing, searching and computing," in *ESORICS Part I*, 2020.

[2] E. Aronesty, D. Cash, Y. Dodis, D. H. Gallancy, C. Higley, H. Karthikeyan, and O. Tysor, "Encapsulated search index: Public-key, sub-linear, distributed, and delegatable," in *PKC*, 2022.

[3] D. Beaver, "Adaptive zero knowledge and computational equivocation (extended abstract)," in *STOC*, 1996.

[4] M. Bellare, A. Boldyreva, K. Kurosawa, and J. Staddon, "Multirecipient encryption schemes: How to save on bandwidth and computation without sacrificing security," *IEEE TIT*, vol. 53, no. 11, 2007.

[5] M. Bellare, A. Boldyreva, and A. O'Neill, "Deterministic and efficiently searchable encryption," in *CRYPTO*, 2007, pp. 535–552.

[6] D. Boneh, G. D. Crescenzo, R. Ostrovsky, and G. Persiano, "Public key encryption with keyword search," in *EUROCRYPT*, 2004.

[7] D. Boneh and M. K. Franklin, "Identity-based encryption from the Weil pairing," in *CRYPTO*, 2001.

[8] R. Bost, "$\sum o\varphi o\varsigma$: Forward secure searchable encryption," in *CCS*, 2016.

[9] R. Bost, B. Minaud, and O. Ohrimenko, "Forward and backward private searchable encryption from constrained cryptographic primitives," in *CCS*, 2017.

[10] J. W. Byun, H. S. Rhee, H. Park, and D. H. Lee, "Off-line keyword guessing attacks on recent keyword search schemes over encrypted data," in *SDM*, 2006.

[11] J. G. Chamani, Y. Wang, D. Papadopoulos, M. Zhang, and R. Jalili, "Multi-user dynamic searchable symmetric encryption with corrupted participants," *IEEE TDSC*, vol. 20, no. 1, pp. 114–130, 2023.

[12] M. Chase and S. Kamara, "Structured encryption and controlled disclosure," in *ASIACRYPT*, 2010, pp. 577–594.

[13] W. Chen, T. Hoang, J. Guajardo, and A. A. Yavuz, "A metadata-hiding file-sharing system with malicious security," in *NDSS*, 2022.

[14] S. S. M. Chow, "New privacy-preserving architectures for identity-/attribute-based encryption," Ph.D. dissertation, New York University, USA, 2010.

[15] S. S. M. Chow, K. Fech, R. W. F. Lai, and G. Malavolta, "Multi-client oblivious RAM with poly-logarithmic communication," in *ASIACRYPT Part II*, 2020.

[16] R. Curtmola, J. A. Garay, S. Kamara, and R. Ostrovsky, "Searchable symmetric encryption: improved definitions and efficient constructions," in *CCS*, 2006.

[17] M. Fadavi and R. R. Farashahi, "Uniform encodings to elliptic curves and indistinguishable point representation," *DCC*, vol. 88, no. 8, 2020.

[18] F. Hahn and F. Kerschbaum, "Searchable encryption with secure and efficient updates," in *CCS*, 2014.

[19] A. Hamlin, A. shelat, M. Weiss, and D. Wichs, "Multi-key searchable encryption, revisited," in *PKC Part I*, 2018.

[20] B. Hemenway, R. Ostrovsky, and A. Rosen, "Non-committing encryption from Φ-hiding," in *TCC Part I*, 2015.

[21] S. Jarecki, C. S. Jutla, H. Krawczyk, M. Rosu, and M. Steiner, "Outsourced symmetric private information retrieval," in *CCS*, 2013.

[22] S. Kamara and T. Moataz, "Computationally volume-hiding structured encryption," in *EUROCRYPT Part II*, 2019.

[23] S. Kamara, T. Moataz, A. Park, and L. Qin, "A decentralized and encrypted national gun registry," in *S&P*, 2021.

[24] S. Kamara, C. Papamanthou, and T. Roeder, "Dynamic searchable symmetric encryption," in *CCS*, 2012.

[25] J. Katz, S. A. Myers, and R. Ostrovsky, "Cryptographic counters and applications to electronic voting," in *EUROCRYPT*, 2001.

[26] A. Kiayias, O. Oksuz, A. Russell, Q. Tang, and B. Wang, "Efficient encrypted keyword search for multi-user data sharing," in *ESORICS Part I*, 2016.

[27] R. W. F. Lai and S. S. M. Chow, "Forward-secure searchable encryption on labeled bipartite graphs," in *ACNS*, 2017.

[28] R. W. F. Lai, G. Malavolta, and V. Ronge, "Succinct arguments for bilinear group arithmetic: Practical structure-preserving cryptography," in *CCS*, 2019.

[29] J. B. Nielsen, "Separating random oracle proofs from complexity theoretic proofs: The non-committing encryption case," in *CRYPTO*, 2002.

[30] S. Patel, G. Persiano, and K. Yeo, "Symmetric searchable encryption with sharing and unsharing," in *ESORICS Part II*, 2018.

[31] C. V. Rompay, R. Molva, and M. Önen, "Multi-user searchable encryption in the cloud," in *ISC*, 2015.

[32] D. X. Song, D. A. Wagner, and A. Perrig, "Practical techniques for searches on encrypted data," in *S&P*, 2000.

[33] E. Stefanov, C. Papamanthou, and E. Shi, "Practical dynamic searchable encryption with small leakage," in *NDSS*, 2014.

[34] Q. Tang and L. Chen, "Public-key encryption with registered keyword search," in *EuroPKI*, 2009.

[35] J. Wang and S. S. M. Chow, "Simple storage-saving structure for volume-hiding encrypted multi-maps," in *DBSec*, 2021.

[36] ——, "*Omnes pro uno*: Practical multi-writer encrypted database," in *USENIX Security*, 2022.

[37] ——, "Forward and backward-secure range-searchable symmetric encryption," *Proc. Priv. Enhancing Technol.*, no. 1, pp. 28–48, 2022.

[38] P. Xu, Q. Wu, W. Wang, W. Susilo, J. Domingo-Ferrer, and H. Jin, "Generating searchable public-key ciphertexts with hidden structures for fast keyword search," *IEEE TIFS*, vol. 10, no. 9, pp. 1993–2006, 2015.

Appendix A
More Related Work

Many *"multi-client/user"* searchable encryption schemes actually consider "single-writer/multi-reader" (S/M). It can be realized by sharing search tokens of SSE via broadcast encryption [16] or interactions between the writer and readers [21]. Schemes for static databases have been well-studied [26], [30]. A "multi-key" scheme (still S/M) [19] requires the writer to replicate the index for each reader or rely on a heavyweight obfuscation. A recent S/M solution [11] achieves forward privacy by tolerating either the inefficiency of oblivious primitives or large reader-side local storage.

Deterministic encryption [5] can realize M/M searchable encryption but with significantly weakened security. Another approach is to *introduce a third party* that performs ciphertext/query transformation. It is still as inefficient as PEKS and might incur multiple rounds of communication [31].

SPCHS of Xu *et al.* [38] is an M/S scheme with optimal asymptotic search efficiency. However, its underlying atomic operation is pairing, hampering search efficiency for very large databases. More importantly, it lacks forward privacy.

HSE of Wang and Chow [36] is the state-of-the-art M/S solution. It is unclear how to upgrade it to the M/M setting due to its epoch-based constraint of search tokens, which would force the data owner to remain online and keep issuing renewed tokens across epochs to multiple readers.

Multi-client ORAM potentially supports secure search in the M/M setting. However, existing solutions [15] require the server computation to be inefficiently linear in the number of ciphertexts. With two non-colluding servers, recent works [1], [23] design (secret-shared) databases contributed by multiple writers. They could answer (function) queries from the readers. Non-dynamic scheme [1] renders forward privacy a non-issue. The non-colluding assumption also makes generic solutions, *e.g.*, multi-client ORAM [15], more efficient.

Encapsulated search index has been recently proposed [2] as a public-key "searchable encryption system" featuring sublinear search. Its encrypted index and search tokens are *document-specific*, in great contrast to the *universal* index over multiple documents (from *different writers*) we considered.

APPENDIX B
ON MORE SECURITY PROPERTIES AND EXTENSIONS

Backward Privacy [9] prevents any search from revealing any deleted document identifier, provided it has not been searched before deletion. To realize it, DSE-$\mathcal{F}$ can adopt the two-roundtrip approach from FIDES [9]. It requires DSE-$\mathcal{F}$ to be response-hiding, as illustrated in Section V-C.

The writer attaches "DEL" flags to the document identifiers to be deleted before encrypting the tuples as VAL via IBE and uploading them to the server. When searching, $\mathsf{DK}_{\mathsf{IBE}}$ will not be included in the token. The server only retrieves and returns a collection of IBE ciphertexts. The reader could decrypt them locally with $\mathsf{DK}_{\mathsf{IBE}}$ and remove those with "DEL." The above adaption prevents the server from learning the identifiers of deleted documents. Like FIDES [9], it takes one more roundtrip to get back the actual documents, and the server cannot reclaim the space for deleted data unless non-deleted identifiers are re-uploaded via updates. For backward privacy without extra roundtrip [9], puncturable encryption can confine the search capacity (over deleted documents) of the token. How to realize it in DSE is an interesting problem.

Volume Hiding conceals the result size of any search. Prior solutions [22], [35] usually pad dummy ciphertexts until each keyword has (roughly) the same volume of search results.

To hide the volume, DSE should be first made response-hiding so that the server cannot observe dummy payloads. DSE makes the encrypted update counters available to eligible clients. A possible way is to instruct writers to check during updates whether their update-permitted keywords have the same volume. If not, a writer makes dummy updates for keywords with fewer volumes and batches them with real updates.

To alleviate the workload of clients, the database owner can take responsibility for hiding the volume. As DSE does not differentiate between different data sources, the database owner could periodically check the global state and update as a writer to pad dummies for keywords with fewer volumes.

Keyword Guessing. PEKS [6] (and general M/S searchable encryption, *e.g.*, [36]) is inherently vulnerable to *offline keyword-guessing attacks* [10]. Search tokens issued by the reader can be tested over ciphertexts of different keywords, which the server can create offline without interacting with the reader. The server can use a token with a known keyword to check if any (future) ciphertext has the same keyword.

Forward Privacy and Pseudonyms. DSE-$\mathcal{F}$ significantly reduces the damage of offline testing issued search tokens. Recall that the IBE ciphertexts (*i.e.*, VAL from $\mathfrak{u}$ in Fig. 13) and the decryption keys (*i.e.*, $\mathsf{DK}_{\mathsf{IBE}}$ from $\mathfrak{s}$ in Fig. 14) in DSE-$\mathcal{F}$ are generated with respect to the search counter of the keyword. By forward privacy, the server cannot test any token against any IBE ciphertext in subsequent updates. Thus, the server is unable to learn yet-to-be-searched keywords.

In DSE-$\mathcal{F}$, the "keywords" delegated to clients do not need to be actual keywords but *pseudonyms* of them. Thus, even if the server could learn the relation between these pseudonyms and some searches, it can, at most, infer statistical information about the actual keywords, *e.g.*, the search frequency of keywords. Provided the fact that such statistical information has already been leaked by the search and access patterns in most practical searchable encryption schemes (*e.g.*, [8]), the keyword guessing attack will not cause extra leakage.

Write Access Control on IBE Ciphertexts. In retrospect, Tang and Chen [34] considered a model called public-key encryption with registered keyword search (PERKS), where their registration is somewhat similar to the delegation in DSE. Technically, the PERKS scheme can be viewed as a simple extension of IBE [7], where the (single) reader distributes secret keyword-specific values beforehand to the eligible writers for encrypting with respect to these keywords. In this way, the adversarial server cannot arbitrarily encrypt searchable ciphertexts to do keyword guessing. While DSE enforces the write access control on the index, PERKS limits the writing ability of searchable ciphertexts. Obviously, if instantiating the IBE scheme with PERKS and distributing the keyword-specific secrets during delegation, DSE-$\mathcal{F}$ achieves the same write access control as PERKS on the generation of searchable ciphertexts, eliminating keyword guessing.

Structured Data. Like HSE, DSE is an encrypted keyword-based index. We can build upon existing techniques to extend DSE for supporting structured data, such as matrices [12], graphs [12], [27], and ranges [37].

APPENDIX C
PROOF FOR SHIFT MULTI-RECIPIENT ENCRYPTION

Proof of Theorem 1. Recall that an adversary $\mathcal{A}$ in the GGM can only see any group element via its representation of distinct random $\lceil \log q \rceil$-bit strings, which are sampled independent of the structure of $\mathbb{G}$. The representation is essentially a map $[\cdot] : \mathbb{Z}_q[\mathcal{Z}] \to \{0,1\}^{\lceil \log q \rceil}$, where $\mathcal{Z}$ is a large enough (larger than the runtime of $\mathcal{A}$) polynomial-size set of symbols, and $\mathbb{Z}_q[\mathcal{Z}]$ is the ring of polynomials over the symbols in $\mathcal{Z}$.

$\mathcal{A}$ is given at the beginning a random bit string $[1]$ representing the generator. To perform group operations on $[a]$ and $[b]$, the adversary must query a group operation oracle, which on input $[a]$ and $[b]$, checks if $[a+b]$ (where $a+b$ is computed over $\mathbb{Z}_q[\mathcal{Z}]$) is already assigned. If not, it assigns $[a+b]$ to a random bit string. The sampled bit-string representation is maintained in a map. In any case, the oracle returns $[a+b]$. In an experiment that involves a simulator, *e.g.*, the ideal experiment, the map is maintained by the simulator. Due to the restriction of the group operation oracle, only affine polynomials of the form $a_0 + \sum_{Z \in \mathcal{Z}} a_Z Z$ in $\mathbb{Z}_q[\mathcal{Z}]$ will ever be assigned.

For the analysis below, observing that for the k-th query to $\mathsf{Expand}\mathcal{O}/\mathsf{Shift}\mathcal{O}$ with input $(j, \ldots)$, the dependency of k on j forms a tree, which can be labeled as follows. The root node is labeled 0. Supposing the k-th query to $\mathsf{Expand}\mathcal{O}/\mathsf{Shift}\mathcal{O}$ is with input j, a node labeled k is created as a child of the node labeled j. We use $\mathsf{Path}(k)$ to denote the set of nodes on the path from the root (exclusive) to k (inclusive).

We construct a stateful simulator $\mathcal{S}$. $\mathcal{S}$ maintains an invariant that x_i is set to some value in $\mathbb{Z}_q$ for all $i \in \mathsf{Corr}$.

On input $(\texttt{Init}, 1^\lambda)$, $\mathcal{S}$ samples $r_0 \leftarrow\$ \mathbb{Z}_q$, assigns r_0 to a random unused bit string, and outputs $[r_0]$ as ctx_0.

On input $(\texttt{KGen}, 1^\lambda)$, $\mathcal{S}$ picks arbitrary distinct unused symbols $X_i \in \mathcal{Z}$, samples random bit strings χ_i, and assigns $[X_i] := \chi_i$ for all $i \in \mathcal{R}$. It outputs $\mathsf{PK}|_{\mathcal{R}} := \{\chi_i\}_{i\in\mathcal{R}}$.

On the ℓ-th call to $\mathcal{S}(\texttt{Expand})$ with (j, A) as the input of $\mathsf{Expand}\mathcal{O}$, $\mathcal{S}$ performs the following steps.

- Parse ctx_j as $(\gamma_{0,j}, \ldots)$ and $\gamma_{0,j} = [r_j]$.
- For all $i \in A \cap \mathsf{Corr}$, set $\gamma_{i,\ell} := [r_j x_i]$.
- For $i \in A \setminus \mathsf{Corr}$, pick an unused symbol $C_{i,\ell} \in \mathcal{Z}$, and assign $C_{i,\ell}$ to a random unused bit string $\gamma_{i,\ell}$.
- Output $\mathsf{ctx}_\ell := \mathsf{ctx}_j \cup \{\gamma_{i,\ell}\}_{i\in A}$.

On the ℓ-th call to $\mathcal{S}(\texttt{Enc}, \{m_i\}_{i\in\mathsf{Corr}})$ with $(j, M|_A = \{m_i\}_{i\in A})$ as the input of $\mathsf{Enc}\mathcal{O}$, $\mathcal{S}$ does:

- Sample $r_\ell \leftarrow\$ \mathbb{Z}_q$.
- Assign r_ℓ and $r_\ell x_i + m_i$ to random unused bit strings for all $i \in \mathsf{Corr}$.
- Set $\gamma'_{0,\ell} := [r_\ell]$ and $\gamma'_{i,\ell} := [r_\ell x_i + m_i]$ for all $i \in \mathsf{Corr}$.
- For $i \in A \setminus \mathsf{Corr}$, pick an unused symbol $C_{i,\ell} \in \mathcal{Z}$, and assign $C_{i,\ell}$ to a random unused bit string $\gamma'_{i,\ell}$.
- Output $\mathsf{ctx}'_\ell := \{\gamma'_{i,\ell}\}_{i\in A\cup\{0\}}$.

On the ℓ-th call to $\mathsf{Shift}\mathcal{O}(j, t)$, retrieve $\mathsf{ctx}'_t = \{\gamma'_{i,t}\}_{i\in D'[t]\cup\{0\}}$ from $\mathsf{Enc}\mathcal{O}$ and $\mathsf{ctx}_j = \{\gamma_{i,j}\}_{i\in D[j]\cup\{0\}}$ from $\mathsf{Expand}\mathcal{O}$ or $\mathsf{Shift}\mathcal{O}$, where $D'[t] = D[j]$. For $i \in D[j] \cup \{0\}$, add up the preimages of $\gamma'_{i,t}$ and $\gamma_{i,j}$, and assign the result to a random unused string $\gamma_{i,\ell}$. Output $\mathsf{ctx}_\ell = \{\gamma_{i,\ell}\}_{i\in D[j]\cup\{0\}}$.

On input $(\texttt{Reveal}, i^*, L[i^*], L'[i^*])$ for some $i^* \in D[\nu]$, if $i^* \in \mathsf{Corr}$, then x_{i^*} was already set and $\mathcal{S}$ simply returns x_{i^*}. Otherwise, $\mathcal{S}$ performs the following steps. Below we analyze the simulation of ctx_ν from $\mathsf{Expand}\mathcal{O}$ and $\mathsf{Shift}\mathcal{O}$. The simulation of ctx_μ from $\mathsf{Enc}\mathcal{O}$ can be conducted similarly.

- Sample $x_{i^*} \leftarrow\$ \mathbb{Z}_q$.
- Replace all assignments involving χ_{i^*} and $\gamma_{i^*,k}$ for all $k \in [\nu]$. That is, if $e + c_{i^*} X_{i^*} + \sum_{k\in[\nu]} d_{i^*,k} C_{i^*,k}$ was assigned to some string α for some expression $e \in \mathbb{Z}_q[\mathcal{Z} \setminus \{X_{i^*}, C_{i^*,k}\}_{k\in[\nu]}]$ and some $c_{i^*}, d_{i^*,1}, \ldots, d_{i^*,\nu} \in \mathbb{Z}_q$ not all zero, check if $e + c_{i^*} x_{i^*} + \sum_{k\in[\nu]} d_{i^*,k}\left(r_k x_i + \sum_{j\in\mathsf{Path}(k)} m_{i,j}\right)$ was assigned, and abort the simulation if so.
- If the simulation is not aborted, remove the entry $e + c_{i^*} X_{i^*} + \sum_{k\in[\nu]} d_{i^*,k} C_{i^*,k}$ and assign $e + c_{i^*} x_{i^*} + \sum_{k\in[\nu]} d_{i^*,k}\left(r_k x_i + \sum_{j\in\mathsf{Path}(k)} m_{i,j}\right)$ to α. $r_k x_i + \sum_{j\in\mathsf{Path}(k)} m_{i,j}$ is assigned to $\gamma_{i,k}$ for all $k \in [\nu]$.
- Output x_i.

If $\mathcal{S}$ does not abort, it simulates the view of $\mathcal{A}$ in the real experiment perfectly. Thus, it suffices to show that $\mathcal{S}$ only aborts with negligible probability. Recall that $\mathcal{S}$ aborts, if and only if on input $(\texttt{Reveal}, i, L[i], L'[i])$ for some $i \in D[\nu] \setminus \mathsf{Corr}$, the expression $e + c_{i^*} x_{i^*} + \sum_{k\in[\nu]} d_{i^*,k}\left(r_k x_i + \sum_{j\in\mathsf{Path}(k)} m_{i,j}\right)$ was already assigned for some $i^* \notin \mathsf{Corr}$, some expression $e \in \mathbb{Z}_q[\mathcal{Z} \setminus \{X_{i^*}, C_{i^*,k}\}_{k\in[\nu]}]$, and some $c_{i^*}, d_{i^*,1}, \ldots, d_{i^*,\nu} \in \mathbb{Z}_q$ not all zero. Supposing this event happens, since only affine polynomials will ever be assigned, we can re-write the expression that was already assigned as

$$a + \sum_{k\in[\nu]} b_k r_k + c_{i^*} x_{i^*} + d_{i^*,k}\left(r_k x_{i^*} + \sum_{j\in\mathsf{Path}(k)} m_{i^*,k}\right) + \sum_{i\notin\mathsf{Corr}\cup\{i^*\}} c_i X_i + \sum_{\substack{i\notin\mathsf{Corr}\cup\{i^*\}\\ k\in[\nu]}} d_{i,k} C_{i,k}$$

where $a, b_k, c_i, d_{i,k} \in \mathbb{Z}_q$. The above can be written as:

$$a' + \sum_{k\in[\nu]} b'_k r_k + \sum_{i\notin\mathsf{Corr}} c'_i X_i + \sum_{\substack{i\notin\mathsf{Corr}\\ k\in[\nu]}} d'_{i,k} C_{i,k}$$

for some $a', b'_k, c'_i, d'_{i,k} \in \mathbb{Z}_q$. For our expression above to be zero, it boils down to having the expression below become zero for some $a'', b''_k, c''_i, d''_{i,k} \in \mathbb{Z}_q$. Since x_{i^*} and r_k are uniformly random and information-theoretically hidden from $\mathcal{A}$, by the Schwartz-Zippel lemma, the probability that

$$a'' + \sum_{k\in[\nu]} b''_k r_k + c_{i^*} x_{i^*} + d_{i^*,k}\left(r_k x_{i^*} + \sum_{j\in\mathsf{Path}(k)} m_{i^*,k}\right)$$

is not a zero polynomial is upper bounded by $1/q = \mathsf{negl}(\lambda)$. Suppose it is indeed a zero polynomial; then, by extracting the expressions of all coefficients, we obtain $c_{i^*} = d_{i^*,k} = 0$ for all $k \in [\nu]$, contradicting the assumption that they are not all zero. As there are only polynomially many entries in the map maintained by $\mathcal{S}$ at any given point of the experiment, we conclude that $\mathcal{S}$ only aborts with negligible probability.

Appendix D
Security Foundation of DSE-$\mathcal{F}$

A. DSE *Security with An Honest Server*

Figs. 19 and 20 present the security experiments of DSE with an honest server (and a set of corrupt clients), where $\mathsf{Delegate}\mathcal{O}$ remains the same as the case of a curious server. Their major difference from the curious-server case is that the encrypted database EDB is withheld from adversary $\mathcal{A}$ since initialization. As the server is honest, $\mathcal{A}$ can no longer get the search (resp. update) token from an honest client via $\mathsf{Srch}\mathcal{O}$ (resp. $\mathsf{Updt}\mathcal{O}$). $\mathcal{A}$ only observes the refreshed global state st'.

In this case, $\mathsf{CorrSrch}\mathcal{O}$ is the only way that $\mathcal{A}$ receives the information of the encrypted database regarding its chosen operations ($\mathsf{CorrUpdt}\mathcal{O}$ does not return results except the updated state). The security experiments thus define a set C to record the corrupt clients in $\mathsf{Corr}\mathcal{O}$ and handle the operations by honest and corrupt clients differently.

B. *Leakage Functions* RevealID *and* RevealKW

With $\mathcal{L}(\mathsf{Hist}^t) = (\mathtt{h}_1, \overline{\mathrm{arg}}_1)\|\cdots\|(\mathtt{h}_t, \overline{\mathrm{arg}}_t)$ being the current leaked history, we introduce two leakage functions. Both functions extend naturally to a keyword set W.

1) $\mathsf{RevealID}_w(\mathcal{L}(\mathsf{Hist}^t))$ reveals identifiers of documents matching keyword w. Suppose $\mathsf{RevealID}_w(\mathcal{L}(\mathsf{Hist}^t)) = (\mathtt{h}_1, \overline{\mathrm{arg}}'_1)\|\cdots\|(\mathtt{h}_t, \overline{\mathrm{arg}}'_t)$. For $j \in [t]$ and $\mathtt{h}_j = \mathsf{Updt}$,

$\mathsf{Real\text{-}HS}^{\mathsf{DSE}}_{\mathcal{A}}(1^\lambda)$

$I := \emptyset$ / client identifier set

$C := \emptyset$ / corrupt client identifier set

$(\mathsf{msk}, \mathsf{st}, \mathsf{EDB}) \leftarrow \mathsf{DSE.Setup}(1^\lambda)$

$\mathbb{O} := \left\{ \begin{array}{c} \mathsf{Srch}\mathcal{O}, \mathsf{Updt}\mathcal{O}, \\ \mathsf{Delegate}\mathcal{O}, \mathsf{Corr}\mathcal{O}, \\ \mathsf{CorrSrch}\mathcal{O}, \mathsf{CorrUpdt}\mathcal{O} \end{array} \right\}$

return $b \leftarrow \mathcal{A}^{\mathbb{O}}(\mathsf{st})$

$\mathsf{Delegate}\mathcal{O}(i, \mathcal{W}_{\mathsf{Q}}, \mathcal{W}_{\text{\faEdit}})$

ensure $i \notin I$

$I := I \cup \{i\}$

$(\mathcal{W}_{\mathsf{Q},i}, \mathcal{W}_{\text{\faEdit},i}) := (\mathcal{W}_{\mathsf{Q}}, \mathcal{W}_{\text{\faEdit}})$

$(\mathsf{st}', \mathsf{sk}_i) \leftarrow \mathsf{Delegate}(\mathsf{st}, \mathsf{msk}, \mathcal{W}_{\mathsf{Q}}, \mathcal{W}_{\text{\faEdit}})$

return st'

$\mathsf{Srch}\mathcal{O}\,(i, W_{\mathsf{Q}})$

ensure $i \in I \wedge i \notin C \wedge W_{\mathsf{Q}} \subseteq \mathcal{W}_{\mathsf{Q},i}$

$\mathfrak{s} \leftarrow \mathsf{SrchTkn}\,(\mathsf{st}, \mathsf{sk}_i, W_{\mathsf{Q}})$

$(\mathsf{st}', \mathsf{EDB}', \mathsf{ID}|_{W_{\mathsf{Q}}}) \leftarrow \mathsf{Srch}(\mathsf{st}, \mathsf{EDB}, \mathfrak{s})$

return st'

$\mathsf{Updt}\mathcal{O}\left(i, \mathsf{ID}|_{W_{\text{\faEdit}}}\right)$

ensure $i \in I \wedge i \notin C \wedge W_{\text{\faEdit}} \subseteq \mathcal{W}_{\text{\faEdit},i}$

$\mathfrak{u} \leftarrow \mathsf{UpdtTkn}\left(\mathsf{st}, \mathsf{sk}_i, \mathsf{ID}|_{W_{\text{\faEdit}}}\right)$

$(\mathsf{st}', \mathsf{EDB}') \leftarrow \mathsf{Updt}(\mathsf{st}, \mathsf{EDB}, \mathfrak{u})$

return st'

$\mathsf{Corr}\mathcal{O}(i)$

ensure $i \in I$; $C := C \cup \{i\}$; **return** sk_i

$\mathsf{CorrSrch}\mathcal{O}(i, W_{\mathsf{Q}})$

ensure $i \in C \wedge W_{\mathsf{Q}} \subseteq \mathcal{W}_{\mathsf{Q},i}$

$\mathfrak{s} \leftarrow \mathsf{SrchTkn}\,(\mathsf{st}, \mathsf{sk}_i, W_{\mathsf{Q}})$

$(\mathsf{st}', \mathsf{EDB}', \mathsf{ID}|_{W_{\mathsf{Q}}}) \leftarrow \mathsf{Srch}(\mathsf{st}, \mathsf{EDB}, \mathfrak{s})$

return $(\mathsf{st}', \mathsf{ID}|_{W_{\mathsf{Q}}})$

$\mathsf{CorrUpdt}\mathcal{O}(i, \mathsf{ID}|_{W_{\text{\faEdit}}})$

ensure $i \in C \wedge W_{\text{\faEdit}} \subseteq \mathcal{W}_{\text{\faEdit},i}$

$\mathfrak{u} \leftarrow \mathsf{UpdtTkn}\left(\mathsf{st}, \mathsf{sk}_i, \mathsf{ID}|_{W_{\text{\faEdit}}}\right)$

$(\mathsf{st}', \mathsf{EDB}') \leftarrow \mathsf{Updt}(\mathsf{st}, \mathsf{EDB}, \mathfrak{u})$

return st'

Fig. 19: Real Experiment for DSE with An Honest Server

$\mathsf{Ideal\text{-}HS}^{\mathsf{DSE}}_{\mathcal{A},\mathcal{S},\mathcal{L}}(1^\lambda)$

$I := \emptyset$ // client identifier set

$C := \emptyset$ // corrupt client identifier set

$\mathsf{Hist} := \mathtt{Setup}$

$(\mathsf{st}, \mathsf{EDB}) \leftarrow \mathcal{S}(\mathcal{L}(\mathsf{Hist}))$

$\mathbb{O} := \left\{ \begin{array}{c} \mathsf{Srch}\mathcal{O}, \mathsf{Updt}\mathcal{O}, \\ \mathsf{Delegate}\mathcal{O}, \mathsf{Corr}\mathcal{O}, \\ \mathsf{CorrSrch}\mathcal{O}, \mathsf{CorrUpdt}\mathcal{O} \end{array} \right\}$

return $b \leftarrow \mathcal{A}^{\mathbb{O}}(\mathsf{st})$

$\mathsf{Delegate}\mathcal{O}(i, \mathcal{W}_{\mathsf{Q}}, \mathcal{W}_{\text{\faEdit}})$

ensure $i \notin I$

$I := I \cup \{i\}$, $(\mathcal{W}_{\mathsf{Q},i}, \mathcal{W}_{\text{\faEdit},i}) := (\mathcal{W}_{\mathsf{Q}}, \mathcal{W}_{\text{\faEdit}})$

$\mathsf{Hist} := \mathsf{Hist} \| (\mathsf{Delegate}, i, \mathcal{W}_{\mathsf{Q}}, \mathcal{W}_{\text{\faEdit}})$

return $\mathsf{st}' \leftarrow \mathcal{S}(\mathcal{L}(\mathsf{Hist}))$

$\mathsf{Srch}\mathcal{O}\,(i, W_{\mathsf{Q}})$

ensure $i \in I \wedge i \notin C \wedge W_{\mathsf{Q}} \subseteq \mathcal{W}_{\mathsf{Q},i}$

$\mathsf{Hist} := \mathsf{Hist} \| \,(\mathsf{Srch}, i, W_{\mathsf{Q}})$

return $\mathsf{st}' \leftarrow \mathcal{S}(\mathcal{L}(\mathsf{Hist}))$

$\mathsf{Updt}\mathcal{O}\left(i, \mathsf{ID}|_{W_{\text{\faEdit}}}\right)$

ensure $i \in I \wedge i \notin C \wedge W_{\text{\faEdit}} \subseteq \mathcal{W}_{\text{\faEdit},i}$

$\mathsf{Hist} := \mathsf{Hist} \| \left(\mathsf{Updt}, i, \mathsf{ID}|_{W_{\text{\faEdit}}}\right)$

return $\mathsf{st}' \leftarrow \mathcal{S}(\mathcal{L}(\mathsf{Hist}))$

$\mathsf{Corr}\mathcal{O}(i)$

ensure $i \in I$

$\mathsf{Hist} := \mathsf{Hist} \| (\mathsf{Corr}, i)$; $C := C \cup \{i\}$

return $\mathsf{sk}_i \leftarrow \mathcal{S}(\mathcal{L}(\mathsf{Hist}))$

$\mathsf{CorrSrch}\mathcal{O}(i, W_{\mathsf{Q}})$

ensure $i \in C \wedge W_{\mathsf{Q}} \subseteq \mathcal{W}_{\mathsf{Q},i}$

$\mathfrak{s} \leftarrow \mathsf{SrchTkn}\,(\mathsf{st}, \mathsf{sk}_i, W_{\mathsf{Q}})$

$\mathsf{Hist} := \mathsf{Hist} \| (\mathsf{CorrSrch}, i, W_{\mathsf{Q}})$

return $(\mathsf{st}', \mathsf{ID}|_{W_{\mathsf{Q}}}) \leftarrow \mathcal{S}(\mathcal{L}(\mathsf{Hist}))$

$\mathsf{CorrUpdt}\mathcal{O}(i, W_{\text{\faEdit}})$

ensure $i \in C \wedge W_{\text{\faEdit}} \subseteq \mathcal{W}_{\text{\faEdit},i}$

$\mathfrak{u} \leftarrow \mathsf{UpdtTkn}\left(\mathsf{st}, \mathsf{sk}_i, \mathsf{ID}|_{W_{\text{\faEdit}}}\right)$

$\mathsf{Hist} := \mathsf{Hist} \| (\mathsf{CorrUpdt}, i, W_{\text{\faEdit}})$

return $\mathsf{st}' \leftarrow \mathcal{S}(\mathcal{L}(\mathsf{Hist}))$

Fig. 20: Ideal Experiment for DSE with An Honest Server

$\overline{\arg}'_j := \overline{\arg}_j \cup (*, \mathsf{ID}|_{\{w\}})$, if $\arg_j = (i, \mathsf{ID}|_{W_{\text{\faEdit}}}) \wedge w \in W_{\text{\faEdit}}$. Arguments of other events are unchanged ($\overline{\arg}'_j := \overline{\arg}_j$).

2) $\mathsf{RevealKW}_w(\mathcal{L}(\mathsf{Hist}^t))$ reveals the occasions when keyword w is updated. Suppose $\mathsf{RevealKW}_w(\mathcal{L}(\mathsf{Hist}^t)) = (\mathtt{h}_1, \overline{\arg}'_1) \| \cdots \| (\mathtt{h}_t, \overline{\arg}'_t)$. For $j \in [t]$ and $\mathtt{h}_j = \mathsf{Updt}$, $\overline{\arg}'_j := \overline{\arg}_j \cup (*, *|_{\{w\}})$, if $\arg_j = (i, \mathsf{ID}|_{W_{\text{\faEdit}}}) \wedge w \in W_{\text{\faEdit}}$. Arguments of other events are unchanged ($\overline{\arg}'_j := \overline{\arg}_j$).

C. Security Proof of DSE-$\mathcal{F}$

For our analysis, we denote the identifier set of corrupt clients for history Hist^t by $C = \left\{i : (\mathsf{Corr}, i) \in \mathsf{Hist}^t\right\}$. Moreover, $W^t_{s,\mathtt{corr}}$ and $W^t_{u,\mathtt{corr}}$ denote the sets of keywords searchable and updatable by any corrupt client, respectively:

$$W^t_{s,\mathtt{corr}} = \left\{ w : \begin{array}{c} (\mathsf{Delegate}, (i, \mathcal{W}_{\mathsf{Q}}, \mathcal{W}_{\text{\faEdit}})) \in \mathsf{Hist}^t \\ \wedge i \in C \wedge w \in \mathcal{W}_{\mathsf{Q}} \end{array} \right\},$$

$$W^t_{u,\mathtt{corr}} = \left\{ w : \begin{array}{c} (\mathsf{Delegate}, (i, \mathcal{W}_{\mathsf{Q}}, \mathcal{W}_{\text{\faEdit}})) \in \mathsf{Hist}^t \\ \wedge i \in C \wedge w \in \mathcal{W}_{\text{\faEdit}} \end{array} \right\}.$$

$W^t_{\mathtt{corr}} = W^t_{s,\mathtt{corr}} \cup W^t_{u,\mathtt{corr}}$ is the keyword set of corrupt clients. We also use $\mathcal{W}_{\mathsf{Q},i}$ and $\mathcal{W}_{\text{\faEdit},i}$ to represent the sets of keywords searchable and updatable by client i, respectively.

We consider $\mathcal{L}_{\mathsf{CS}}(\mathsf{Hist}^t)$ for different cases of $(\mathtt{h}_t, \arg_t)$.

- Case $(\mathtt{h}_t, \arg_t) = (\mathsf{Delegate}, (i, \mathcal{W}_{\mathsf{Q}}, \mathcal{W}_{\text{\faEdit}}))$. It only leaks the pseudonyms of new active keywords. Therefore, $\mathcal{L}_{\mathsf{CS}}(\mathsf{Hist}^t) := \mathcal{L}_{\mathsf{CS}}(\mathsf{Hist}^{t-1}) \| (\mathsf{Delegate}, (*, *, \mathcal{W}_{\text{\faEdit}} \backslash \mathcal{W}))$, where $\mathcal{W}$ is the active keyword space before the event.
- Case $(\mathtt{h}_t, \arg_t) = (\mathsf{Corr}, i)$. Corrupting client i leaks document identifiers matching any keyword searchable by i, and the occasions when any keyword updatable by i was updated. Formally, $\mathcal{L}_{\mathsf{CS}}(\mathsf{Hist}^t)$ can be written as $\mathsf{RevealKW}_{\mathcal{W}_{\text{\faEdit},i}}\big(\mathsf{RevealID}_{\mathcal{W}_{\mathsf{Q},i}}\big(\mathcal{L}_{\mathsf{CS}}\big(\mathsf{Hist}^{t-1}\big)\big)\big) \| (\mathsf{Corr}, i)$.
- Case $(\mathtt{h}_t, \arg_t) = \left(\mathsf{Updt}, (i, \mathsf{ID}|_{W_{\text{\faEdit}}})\right)$. For $w \in W_{\text{\faEdit}}$, the leakages of DSE-$\mathcal{F}$ depend on how w was delegated:
  - if no corrupt client can update or search on w, *i.e.*, $w \notin W^t_{\mathtt{corr}}$, nothing is leaked;
  - if w is updatable but not searchable by any corrupt

client, *i.e.*, $w \in W^t_{u,\mathtt{corr}} \setminus W^t_{s,\mathtt{corr}}$, w is revealed; and
  - if w is searchable by any corrupt client, *i.e.*, $w \in W^t_{s,\mathtt{corr}}$, the search result of w is revealed.

  Formally, $\mathcal{L}_{\mathsf{CS}}(\mathsf{Hist}^t)$ could be written as $\mathcal{L}_{\mathsf{CS}}(\mathsf{Hist}^{t-1}) \| \Big(\mathsf{Updt}, (*, *|_{W_{\text{✎}} \cap (W^t_{u,\mathtt{corr}} \setminus W^t_{s,\mathtt{corr}})} \cup \mathsf{ID}|_{W_{\text{✎}} \cap W^t_{s,\mathtt{corr}}})\Big)$. Since $W^t_{\mathtt{corr}} = W^t_{s,\mathtt{corr}} \cup W^t_{u,\mathtt{corr}}$, we have $W_{\text{✎}} \cap (W^t_{u,\mathtt{corr}} \setminus W^t_{s,\mathtt{corr}}) \subseteq W_{\text{✎}} \cap W^t_{\mathtt{corr}}$ and $W_{\text{✎}} \cap W^t_{s,\mathtt{corr}} \subseteq W_{\text{✎}} \cap W^t_{\mathtt{corr}}$. The update leakage of DSE-$\mathcal{F}$ is no more than the document identifiers of corrupt keywords, which satisfies forward privacy (Definition 8).
- Case $(\mathtt{h}_t, \arg_t) = (\mathsf{Srch}, (i, W_{\mathsf{Q}}))$. For $w \in W_{\mathsf{Q}}$, it leaks the fact that w is being searched and the identifiers of documents matching w. We define $\mathcal{L}_{\mathsf{CS}}(\mathsf{Hist}^t)$ as $\mathsf{RevealID}_{W_{\mathsf{Q}}}\left(\mathcal{L}_{\mathsf{CS}}\left(\mathsf{Hist}^{t-1}\right)\right) \| (\mathsf{Srch}, (*, W_{\mathsf{Q}}))$.

We analyze $\mathcal{L}_{\mathsf{HS}}(\mathsf{Hist}^t)$ and differentiate it from $\mathcal{L}_{\mathsf{CS}}$.

- Case $(\mathtt{h}_t, \arg_t) = (\mathsf{Delegate}, (i, \mathcal{W}_{\mathsf{Q}}, \mathcal{W}_{\text{✎}}))$. The leakage remains the same as $\mathcal{L}_{\mathsf{CS}}$. That is, $\mathcal{L}_{\mathsf{HS}}(\mathsf{Hist}^t) := \mathcal{L}_{\mathsf{HS}}(\mathsf{Hist}^{t-1}) \| (\mathsf{Delegate}, (*, *, \mathcal{W}_{\text{✎}} \setminus \mathcal{W}))$.
- Case $(\mathtt{h}_t, \arg_t) = (\mathsf{Corr}, i)$. Unlike $\mathcal{L}_{\mathsf{CS}}(\mathsf{Hist}^t)$, without the view of the server, it will not leak the identifiers of documents with keywords searchable by i. The corruption on client i only reveals the occasions when keywords updatable or searchable by i were updated. Formally, $\mathcal{L}_{\mathsf{HS}}(\mathsf{Hist}^t)$ can be written as $\mathsf{RevealKW}_{\mathcal{W}_{\text{✎},i} \cup \mathcal{W}_{\mathsf{Q},i}}\left(\mathcal{L}_{\mathsf{HS}}\left(\mathsf{Hist}^{t-1}\right)\right) \| (\mathsf{Corr}, i)$.
- Case $(\mathtt{h}_t, \arg_t) = \Big(\mathsf{Updt}, (i, \mathsf{ID}|_{W_{\text{✎}}})\Big)$. For $w \in W_{\text{✎}}$, it has two different leakages based on how keywords are delegated to corrupt clients: if w is neither searchable nor updatable by any corrupt client, *i.e.*, $w \notin W^t_{\mathtt{corr}}$, it leaks nothing; otherwise, w is leaked. Formally, we define $\mathcal{L}_{\mathsf{HS}}(\mathsf{Hist}^t) := \mathcal{L}_{\mathsf{HS}}\left(\mathsf{Hist}^{t-1}\right) \| \Big(\mathsf{Updt}, \Big(*, *|_{W_{\text{✎}} \cap W^t_{\mathtt{corr}}}\Big)\Big)$.
- Case $(\mathtt{h}_t, \arg_t) = (\mathsf{Srch}, (i, W_{\mathsf{Q}}))$. For $w \in W_{\mathsf{Q}}$, it only leaks the fact that w is being searched without the view of the server. Formally, $\mathcal{L}_{\mathsf{HS}}(\mathsf{Hist}^t) := \mathcal{L}_{\mathsf{HS}}\left(\mathsf{Hist}^{t-1}\right) \| (\mathsf{Srch}, (*, W_{\mathsf{Q}}))$.
- Case $(\mathtt{h}_t, \arg_t) = \Big(\mathsf{CorrUpdt}, (i, \mathsf{ID}|_{W_{\text{✎}}})\Big)$. No more information about Hist^{t-1} is revealed. Formally, we have $\mathcal{L}_{\mathsf{HS}}(\mathsf{Hist}^t) := \mathcal{L}_{\mathsf{HS}}(\mathsf{Hist}^{t-1}) \| \Big(\mathsf{CorrUpdt}, (i, \mathsf{ID}|_{W_{\text{✎}}})\Big)$.
- Case $(\mathtt{h}_t, \arg_t) = (\mathsf{CorrSrch}, (i, W_{\mathsf{Q}}))$. The results matching W_{Q} are leaked. Formally, $\mathcal{L}_{\mathsf{HS}}(\mathsf{Hist}^t)$ can be written as $\mathsf{RevealID}_{W_{\mathsf{Q}}}\left(\mathcal{L}_{\mathsf{HS}}(\mathsf{Hist}^{t-1})\right) \| (\mathsf{CorrSrch}, (i, W_{\mathsf{Q}}))$. Note that $\mathcal{L}_{\mathsf{CS}}$ has the same leakage upon (Corr, i).

Proof of Theorem 2. We derive a game sequence from Real-CS (or **Game** G_0) to Ideal-CS (or **Game** G_5).

- **Game** G_1: Instead of calling $F(\mathsf{msk}, \cdot)$ as in G_0, G_1 picks a random string when it meets a new PRF input and stores it to answer the same query. If the adversary could distinguish between G_0 and G_1, we can build a reduction to distinguish between PRF and a truly random function.
- **Game** G_2: G_2 maintains an internal record Uctr^* of adversarial updates. That is, upon receiving any query from a corrupt client, G_2 modifies Uctr^* accordingly.

  When generating update tokens for $\mathsf{Updt}\mathcal{O}$, $w \in W^t_{\mathtt{corr}}$ from these updates will be leaked. For these keywords, G_2 increases Uctr^* accordingly to obtain Uctr instead of decrypting it from $\mathsf{ctx}_{\mathcal{W}}$. $\mathsf{ctx}_{\mathcal{W}}$ could be shifted according to the correct offset. By the correctness of SME, any adversary cannot distinguish between G_1 and G_2.
- **Game** G_3: G_3 simulates $\mathsf{DK}_{\mathsf{IBE}}$ for $\mathsf{Srch}\mathcal{O}$, VAL for $\mathsf{Updt}\mathcal{O}$, and $\mathsf{SK}_{\mathsf{IBE}}$ for $\mathsf{Corr}\mathcal{O}$.
  - If the searching right of keyword w has not been delegated to any corrupt client from $\mathsf{Corr}\mathcal{O}$, G_3 simply calls the IBE simulator with trapdoors to simulate $\mathsf{VAL}[w]$.
  - When $\mathsf{DK}_{\mathsf{IBE}}[w]$ needs to be revealed for $w \in W_{\mathsf{Q}}$, G_3 gets the leakage of occasions when updates on w and identifiers of documents matching w. G_3 recalls previous addresses storing IBE ciphertexts of w (*i.e.*, VAL with respect to the "identity" $\mathsf{Sctr}[w]$) and invokes the IBE simulator to generate $\mathsf{DK}_{\mathsf{IBE}}[w]$ regarding $\mathsf{Sctr}[w]$.
  - When $\mathsf{SK}_{\mathsf{IBE}}[w]$ needs to be revealed as the searching right of w is granted to a corrupt client from $\mathsf{Corr}\mathcal{O}$, G_3 invokes the IBE simulator for its trapdoors/secret keys after related $\mathsf{DK}_{\mathsf{IBE}}$ elements were simulated as above.

  With non-committing IBE, no $\mathcal{A}$ distinguishes G_3 and G_2.
- **Game** G_4: For $\mathsf{IK}_{\mathsf{EDB}}$ in $\mathsf{Srch}\mathcal{O}$ and $\mathsf{SK}_{\mathsf{EDB}}$ in $\mathsf{Corr}\mathcal{O}$:
  - When $\mathsf{IK}_{\mathsf{EDB}}[w]$ needs to be revealed for $w \in W_{\mathsf{Q}}$ since previous addresses regarding $\mathsf{IK}_{\mathsf{EDB}}[w]$ are leaked, G_4 invokes the PRF simulator to simulate the corresponding PRF key, and returns it as $\mathsf{IK}_{\mathsf{EDB}}[w]$.
  - When $\mathsf{SK}_{\mathsf{EDB}}[w]$ needs to be revealed as the searching right of w is granted to a corrupt client from $\mathsf{Corr}\mathcal{O}$, G_4 returns the PRF key simulated by the PRF simulator after related $\mathsf{IK}_{\mathsf{EDB}}$ elements were simulated as above.

  With non-committing F, no $\mathcal{A}$ distinguishes G_4 from G_3.
- **Game** G_5: G_5 simulates $\mathsf{ctx}_{\mathcal{W}}$ for $\mathsf{Delegate}\mathcal{O}$ and $\mathsf{Updt}\mathcal{O}$. G_5 also simulates $\mathsf{SK}_{\mathsf{SME}}$ for $\mathsf{Corr}\mathcal{O}$.
  - Call the SME simulator ($\mathsf{Expand}\mathcal{O}$) to simulate the expanded entries of $\mathsf{ctx}_{\mathcal{W}}$ in $\mathsf{Delegate}\mathcal{O}$.
  - Call the SME simulator ($\mathsf{Enc}\mathcal{O}$ and $\mathsf{Shift}\mathcal{O}$) to simulate the ciphertext $\mathsf{ctx}'_{\mathcal{W}}$ for shifting $\mathsf{ctx}_{\mathcal{W}}$ to $\mathsf{ctx}''_{\mathcal{W}}$ in $\mathsf{Updt}\mathcal{O}$.
  - Call the SME simulator ($\mathsf{Corr}\mathcal{O}$) to simulate $\mathsf{SK}_{\mathsf{SME}}$ in $\mathsf{Corr}\mathcal{O}$. As keywords related to $\mathsf{SK}_{\mathsf{SME}}$ to be revealed are now updatable and/or searchable by corrupt clients, the update counters of these keywords leak, with which SME simulator could complete the simulation.

With non-committedness, no $\mathcal{A}$ distinguishes G_5 from G_4. We thus conclude the game transitions and security proof.

For forward privacy, recall the leakage $\mathcal{L}_{\mathsf{CS}}$ analyzed at the beginning: when the keyword $w \in W_{\text{✎}}$ of these update tuples is neither searchable nor updatable by any corrupt client (*i.e.*, $w \notin W^t_{\mathtt{corr}}$), nothing about w will be revealed. DSE-$\mathcal{F}$ fulfills the requirement in Definition 8 that $\mathsf{Updt}\mathcal{O}$ leaks nothing more than document identifiers of corrupt keywords.

For security with an honest server, its proof is similar, except that it does not need to simulate the server's view. Simulations of matches in $\mathsf{Srch}\mathcal{O}$ and $\mathsf{Corr}\mathcal{O}$ are deferred to $\mathsf{CorrSrch}\mathcal{O}$.