Dynamic Searchable Symmetric Encryption with Forward and Backward Privacy

Yu Peng†, Qin Liu†*, Yue Tian†, Jie Wu‡, Tian Wang§, Tao Peng††, and Guojun Wang††
†College of Computer Science and Electronic Engineering, Hunan University, P. R. China
‡Department of Computer and Information Sciences, Temple University, Philadelphia, USA
§Institute of Artificial Intelligence and Future Networks, Beijing Normal University & UIC, P. R. China
††School of Computer Science and Cyber Engineering, Guangzhou University, P. R. China
*Correspondence to: gracelq628@hnu.edu.cn

***Abstract*—Dynamic searchable symmetric encryption (DSSE) that enables a client to perform searches and updates on encrypted data has been intensively studied in cloud computing. Recently, forward privacy and backward privacy has engaged significant attention to protect DSSE from the leakage of updates. However, the research in this field almost focused on keyword-level updates. That is, the client needs to know the keywords of the documents in advance. In this paper, we proposed a document-level update scheme, DBP, which supports immediate deletion while guaranteeing forward privacy and backward privacy. Compared with existing forward and backward private DSSE schemes, our DBP scheme has the following merits: 1) *Practicality*. It achieves deletion based on document identifiers rather than document/keyword pairs; 2) *Efficiency*. It utilizes only lightweight primitives to realize backward privacy while supporting immediate deletion. Experimental evaluation on two real datasets demonstrates the practical efficiency of our scheme.**

***Index Terms*—Cloud computing, dynamic searchable symmetric encryption, forward privacy, backward privacy**

I. Introduction

Cloud computing, as a successful paradigm of service-oriented computing, centralizes a great deal of storage and computation sources while enabling convenient and on-demand network access [1], [2]. With the rapid development of cloud computing, an increasing number of users prefer outsourcing their data to the server. To protect the confidentiality of outsourced data, the common solution is to encrypt data using general symmetric encryption before outsourcing. However, encryption destroys the natural structure of data and makes keyword-based search services intractable. To solve this dilemma, searchable symmetric encryption (SSE) [3], [4] has been proposed to perform secure searches over encrypted data. Early SSE schemes were designed for a static setting, which restricts their application. To make the scheme more practical, dynamic searchable symmetric encryption (DSSE) [5]–[8] was proposed to guarantee the update of data.

With the trade of security and efficiency, most of the existing DSSE schemes leak some information in data search and update phases. These leakages, unfortunately, can be used to compromise the privacy of search queries. Recently, the file-injection attack proposed by Zhang et al. [9] shows that user queries can be revealed by adding a small number of carefully designed documents into a database. Consequently, forward privacy (FP) [10]–[13] that prevents the linkability from the newly added documents to previously search tokens becomes a crucial property for DSSE schemes. In a similar vein, backward privacy (BP) [14]–[19] has been concerned by DSSE schemes, which implies that the deleted document is inaccessible for the subsequent search queries.

Nevertheless, most existing forward and backward private schemes focus on keyword-level updates. In other words, the client needs to know all keywords contained in the document to be updated in advance, which is unrealistic for the deletion operation. The work [19] eliminates this assumption but the storage cost at the client is liner with the number of documents. In addition, many forward and backward private schemes perform the deletion in the same way as the addition, which increase both storage space and computational complexity. To solve the above problems, in this paper, we aim to design a forward and backward private scheme which supports document-based immediate deletion. Specifically, we first construct a basic scheme to support document-level update, which the client requires only an identifier of the document in the deletion operation. Then, we propose an advanced scheme to resist external attackers. Our proposed DBP scheme has the following merits: 1) *Practicality.* It eliminates the premise that the client needs to know the keywords of the document to be deleted in advance. 2) *Efficiency.* It utilizes only lightweight cryptographic primitives to realize backward privacy while supporting immediate deletion. Meanwhile, the storage cost at client is liner with the number of keywords instead of documents. The main contributions of this paper can be summarized as follows:

- To the best of our knowledge, it is the first attempt to devise an efficient forward and backward private scheme, which supports document-based immediate deletion.
- Two constructions are provided to achieve enhanced privacy in different adversary models.
- We evaluate our scheme on two real datasets. The results demonstrate that our scheme is efficient.

The remaining sections of this paper are organized as follows. We provide the preliminaries in Section II before giving the definition and security of DSSE in Section III. We present our DBP scheme in Section IV. The security and

experimental analyses are given in Section V and Section VI, respectively. Finally, we introduce related work in Section VII, before concluding this paper in Section Section VIII.

II. Preliminary

A. The System and Adversary Model

System model consists of two entities, a client and a server. The client can upload/update the encrypted documents along with the encrypted indexes to the server. Besides, the client can generate a search token for a particular keyword and sends it to the server. On receiving the search request from the client, the server evaluates the search token on the encrypted indexes and returns the search results to the client.

In our threat model, the server is considered to be honest-but-curious (HBC) [20]. That is, the server will correctly perform the predefined protocols, but it may try to learn additional information from the encrypted documents and from each protocol. Besides, the server may be attacked by the external attackers, who can obtain control over the server. In this condition, the attacker can initiate snapshot attack, which see the encrypted database and the leakages in protocols at one (or more) instant [14].

B. Notations and Cryptographic Preliminaries

Let $\lambda \in \mathbb{N}$ denote a security parameter, $\mathbf{negl}(\lambda)$ denote a negligible function in λ, and $\mathbf{0}$ be a string of 0s with length λ. For $\eta \in \mathbb{N}$, notation $[\eta]$ is used to denote the set of integers $\{1, \ldots, \eta\}$. We denote the set of all binary strings of length η by $\{0,1\}^\eta$ and the set of finite binary strings by $\{0,1\}^*$. The concatenation of η strings $s_1, \ldots, s_\eta$ is denoted by $\langle s_1, \ldots, s_\eta \rangle$. For a finite set X, notation $|X|$ denotes its cardinality, and $(x_1, \ldots, x_\eta) \xleftarrow{\$} X$ means that x_i, $i \in [\eta]$, is sampled uniformly from X. For quick reference, the most relevant notations used in this work are shown in Table I.

Pseudo-Random Function (PRF). PRF, a fundamental primitive for emulating perfect randomness via keyed functions, is a polynomial-time computable two-input function and is indistinguishable from a true random function by any probabilistic polynomial-time (PPT) adversary when the key is kept secret. For a formal definition see [21].

TABLE I
Summary of Notations

Notations	Descriptions
ind	The identifier of the document
DB	A database that includes n documents $\{\mathrm{ind}_1, \ldots, \mathrm{ind}_n\}$
EDB	An encrypted document database
$\mathrm{DB}(\mathrm{ind}_i)$	A set of keywords in document ind_i
$\mathrm{DB}(w)$	A set of documents containing the keyword w
f	A document, $f = (\mathrm{ind}, \mathrm{DB}(\mathrm{ind}))$
W	The set of all keywords in DB, $\mathrm{W} = \cup_{i=1}^{n} \mathrm{DB}(\mathrm{ind}_i)$
m	The total number of keywords in W
N	The total number of document/keyword pairs

III. DSSE Definition and Security Model

A. DSSE Definition

A DSSE scheme consists of four polynomial-time protocols between a client and a server, they are described as follows:

$\mathsf{Setup}(\lambda; \perp) \rightarrow (\sigma; \mathrm{EDB})$: It takes a security parameter λ as input, and outputs a state σ for the client and an encrypted database EDB for the server, respectively.

$\mathsf{Addition}(\sigma, f; \mathrm{EDB}) \rightarrow (\sigma'; \mathrm{EDB}')$: The client takes the state σ and a document f as input, and the server takes the encrypted database EDB as input. At the end of this protocol, the client outputs an updated state σ', and the server outputs an updated encrypted database EDB' including the identifier of document f.

$\mathsf{Deletion}(\sigma, \mathrm{ind}; \mathrm{EDB}) \rightarrow (\perp; \mathrm{EDB}')$: The client takes the state σ and a document identifier ind as input, and the server takes the encrypted database EDB as input. At the end of this protocol, the server outputs an updated database EDB', excluding the identifier ind.

$\mathsf{Search}(\sigma, w; \mathrm{EDB}) \rightarrow (\sigma', \mathrm{DB}(w); \mathrm{EDB}')$: The client takes the state σ and a keyword w as input, and the server takes the encrypted database EDB as input. When the protocol ends, the client outputs an updated state σ' and the search result $\mathrm{DB}(w)$. The server outputs an updated encrypted database EDB'.

B. Security Model

DSSE security is captured by utilizing a real-world versus an ideal-world formalization named $\mathbf{Real}$ and $\mathbf{Ideal}$, respectively. The behavior of $\mathbf{Real}$ is exactly the same as the original DSSE, and $\mathbf{Ideal}$ reflects a behavior of a simulator $\mathcal{S}$, which takes the leakage of the original DSSE as input. The leakage is defined by a function $\mathcal{L} = (\mathcal{L}_{\mathsf{set}}, \mathcal{L}_{\mathsf{add}}, \mathcal{L}_{\mathsf{del}}, \mathcal{L}_{\mathsf{srh}})$, which details what information the adversary $\mathcal{A}$ can know during the execution of Setup, Addition, Deletion, and Search protocols, respectively. We consider the following probabilistic experiments.

- $\mathbf{Real}_{\mathcal{A}}(\lambda)$: Given EDB output by Setup, the adversary $\mathcal{A}$ makes a polynomial number of adaptive queries, and for each query, it receives a search token τ_s by running Search, or an addition token τ_a by running Addition, or a deletion token τ_d by running Deletion. Eventually, $\mathcal{A}$ outputs a bit $b \in \{0,1\}$.
- $\mathbf{Ideal}_{\mathcal{A},\mathcal{S}}(\lambda)$: Given the leakage $\mathcal{L}_{\mathsf{set}}$, the simulator $\mathcal{S}$ generates an encrypted database EDB to the adversary $\mathcal{A}$. $\mathcal{A}$ makes a polynomial number of adaptive queries. Given leakages $\mathcal{L}_{\mathsf{add}}$, $\mathcal{L}_{\mathsf{del}}$, and $\mathcal{L}_{\mathsf{srh}}$, $\mathcal{S}$ returns an appropriate token. Eventually, $\mathcal{A}$ outputs a bit $b \in \{0,1\}$.

Informally, if the adversary $\mathcal{A}$ can distinguish between $\mathbf{Real}$ and $\mathbf{Ideal}$ with only a negligible advantage, the information leakage is limited to $\mathcal{L}$ only. We say that the DSSE scheme achieves adaptive security. Formally, we provide the following definition:

Definition 1 (Adaptive security of DSSE)**.** *A DSSE scheme is $\mathcal{L}$-adaptive-secure if for PPT adversaries $\mathcal{A}$, there exists a PPT simulator $\mathcal{S}$ s.t.* $|\Pr[\mathbf{Real}_{\mathcal{A}}(\lambda) = 1] - \Pr[\mathbf{Ideal}_{\mathcal{A},\mathcal{S}}(\lambda) = 1]| \leq \mathbf{negl}(\lambda)$.

Leakage Function. The leakage functions $\mathcal{L}$ keep a list Q of all queries issued so far as a state. Each entry of Q is in the form of (v, w) for a search query, or $(v, \mathbf{op}, \mathbf{in})$ for an $\mathbf{op}$ update with input $\mathbf{in}$, where w is a queried keyword, the integer v is a timestamp that is initialized to 0 and incremented by 1 at each query, and $\mathbf{op} \in \{\mathsf{add}, \mathsf{del}\}$ denoting the addition and deletion operations. For a list Q, we define a search pattern $\mathsf{SP}(w) = \{v | (v, w) \in Q\}$, which leaks the repetition of search tokens on w issued so far. In addition, we also use notations $\mathsf{TimeDB}(w)$ to capture the leakages in backward private DSSE schemes. $\mathsf{TimeDB}(w) = \{(v, \mathrm{ind}) : (v, \mathsf{add}, (\mathrm{ind}, w)) \in Q \wedge \forall v', (v', \mathsf{del}, (w, \mathrm{ind})) \notin Q\}$ is a list of non-deleted documents matching w along with the timestamps of inserting them into the database.

C. Forward and Backward Privacy

Forward privacy ensures that update queries leak no information about which keywords are involved in the newly added documents. We follow the formal definition in [13].

Definition 2 (Forward privacy of DSSE)**.** *An $\mathcal{L}$-adaptive-secure DSSE scheme achieves forward privacy if there exists a stateless leakage function $\overline{\mathcal{L}}$ s.t $\mathcal{L}_{add}$ can be written as* $\mathcal{L}_{add}(\mathrm{ind}, DB(\mathrm{ind})) = \overline{\mathcal{L}}(\mathrm{ind}, |DB(\mathrm{ind})|)$.

Backward privacy ensures that search queries do not reveal matched documents after they have been deleted. With the leakage functions described above, we provide the following formal definition:

Definition 3 (Backward privacy of DSSE)**.** *An $\mathcal{L}$-adaptive-secure DSSE scheme achieves backward privacy if there exist stateless leakage functions $\overline{\mathcal{L}}$, $\overline{\mathcal{L}}'$ and $\widetilde{\mathcal{L}}$ s.t the update and search leakages can be written as $\mathcal{L}_{add}(\mathrm{ind}) = \overline{\mathcal{L}}(|DB(\mathrm{ind})|)$, $\mathcal{L}_{del}(\mathrm{ind}) = \overline{\mathcal{L}}'(\mathrm{ind})$ and $\mathcal{L}_{srh}(w) = \widetilde{\mathcal{L}}(SP(w), TimeDB(w))$.*

IV. DBP: Document-based Backward Private DSSE

In this section, we first proposed our basic DBP scheme, DBP-B, which efficiently achieves forward and backward privacy while supporting document-based immediate deletion. On this basis, we proposed an advanced DBP scheme DBP-E which can resist the external attackers.

A. Overview

To efficiently update documents, we adopt a map T_e to store the $(\mathrm{key}, \mathrm{value})$ pair for each document, where key is related to a document identifier and value is related to the set of keywords contained in this document. Unlike most previous BP schemes that require the client to know the keywords of a document to be deleted in advance, our DBP scheme just needs the client to send a key to the server, which will directly remove the corresponding entry of T_e while releasing related storage space.

However, forward indexes degrade search efficiency. When the keyword searched for the first time, the search complexity is linear with the number of document/keyword pairs in the database. Inspired by the work in [14], [15], we amortize this cost over multiple searches by letting the server cache the results after each search. Specifically, we also keep a map T_p to store the last search results for each keyword. When searching a keyword, the client will issue two search tokens to evaluate T_e and T_p, respectively. When the search phase ends, the search results will be moved from T_e to T_p. Note that this will not cause security degradation in DBP since the server has already learnt search results in previous search queries according to access pattern [4].

To guarantee forward privacy, DBP adopts a fresh secret key, which is unrelated to previous search tokens, to encrypt each newly inserted data. In terms of backward privacy, DBP-B is implied by immediate document-based deletion. More precisely, the HBC server will honestly execute the deletion protocol and release the related storage spaces once a document is deleted from the system while leaking only the identifier of the deleted document. Once the spaces are freed, the server cannot know which keywords contained in this document in subsequent search queries. Consider the external attackers, who might control the server to initiate snapshot attacks thereby compromising backward privacy. DBP-E resists the attacks at the expense of two roundtrips in the search phase.

B. Construction of DBP-B

Let $F : \{0,1\}^\lambda \times \{0,1\}^* \to \{0,1\}^\lambda$ be a PRF, and let $H_1 : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^{2\lambda}$ be a keyed hash function. The details of DBP-B are shown in Protocol 1 except the pseudocode in boxes.

DBP-B.Setup. The client locally maintains a state $\sigma = (k_1, \mathrm{T}_c)$, where k_1 is the secret key of PRF F and T_c is a local map that stores the secret key $\mathrm{key}^{(w)}$ for each keyword w. The client also generates two maps, T_p is used to store the results of the last search query, and T_e is to store the encrypted indexes. The encrypted database $\mathrm{EDB} = (\mathrm{T_p}, \mathrm{T_e})$ will be sent to the server.

DBP-B.Addition. To add a document $f = (\mathrm{ind}, \mathrm{DB}(\mathrm{ind}))$ into the database, the client sends an addition token $\tau_a = (F_{k_1}(\mathrm{ind}), \mathrm{AddSet})$ to the server, which updates map T_e by storing all elements in AddSet as the value under key $F_{k_1}(\mathrm{ind})$. Specifically, the set AddSet is constructed as follows: For every keyword $w \in \mathrm{DB}(\mathrm{ind})$, the client retrieves the secret key $\mathrm{key}^{(w)}$ corresponding to w from T_c. Then, it encrypts $\langle \mathrm{ind}, \mathbf{0} \rangle$ with $H_1(F_{k_1}(\mathrm{ind}), \mathrm{key}^{(w)})$ and puts the ciphertext e into AddSet.

DBP-B.Search. To find all documents matching keyword w, the client retrieves the current secret key $\mathrm{key}^{(w)}$ from T_c, and then updates $\mathrm{T}_c[w]$ with a string sampled randomly from $\{0,1\}^\lambda$. Next, the client generates the search token $\tau_s = (F_{k_1}(w), \mathrm{key}^{(w)})$, and sends it to the server. Note that a new secret key is used to encrypt documents containing keyword w added later once a search query on w is performed. Therefore, previous search tokens cannot be used to evaluate the newly added documents, and DBP-B satisfies forward privacy. As for the server, it parses τ_s as $(\alpha, \mathrm{key}^{(w)})$ where α is used to search the map T_p and $\mathrm{key}^{(w)}$ is used to search the

Protocol 1 DBP

$\underline{\mathsf{Setup}(\lambda;\perp)\rightarrow(\sigma;\mathrm{EDB})}$

$\mathrm{Client}(\lambda)\rightarrow(\sigma,\mathrm{EDB})$

1: k_1, $\boxed{k_2} \xleftarrow{\$} \{0,1\}^{\lambda}$
2: $(\mathrm{T}_c,\mathrm{T}_e,\mathrm{T}_p) \leftarrow$ empty map
3: $\sigma \leftarrow (k_1,\boxed{k_2},\mathrm{T}_c)$; $\mathrm{EDB} \leftarrow (\mathrm{T_e},\mathrm{T_p})$
4: **Send** EDB **to** Server

$\underline{\mathsf{Addition}(\sigma,f;\mathrm{EDB})\rightarrow(\sigma';\mathrm{EDB}')}$

$\mathrm{Client}(\sigma,\mathrm{ind},\mathrm{DB(ind)})\rightarrow(\sigma',\tau_\mathrm{a})$

1: $\mathrm{AddSet} \leftarrow \emptyset$
2: **for** each $w \in \mathrm{DB(ind)}$ **do**
3:    **if** $\mathrm{T}_c[w]=\perp$ **then**
4:       $\mathrm{key}^{(w)} \xleftarrow{\$} \{0,1\}^{\lambda}$; $\mathrm{T}_c[w] \leftarrow \mathrm{key}^{(w)}$
5:    **else**
6:       $\mathrm{key}^{(w)} \leftarrow \mathrm{T}_c[w]$
7:    $e \leftarrow H_1(F_{k_1}(\mathrm{ind}),\mathrm{key}^{(w)}) \oplus \langle \mathrm{ind},\mathbf{0}\rangle$
8:    $\boxed{e \leftarrow H_1(F_{k_1}(\mathrm{ind}),\mathrm{key}^{(w)}) \oplus \langle Enc(k_2,\mathrm{ind}),\mathbf{0}\rangle}$
9:    $\mathrm{AddSet} \leftarrow \mathrm{AddSet} \cup \mathrm{e}$
10: $\tau_a \leftarrow (F_{k_1}(\mathrm{ind}),\mathrm{AddSet})$
11: **Send** τ_a **to** Server

$\underline{\mathrm{Server}(\mathrm{EDB},\tau_\mathrm{a})\rightarrow(\mathrm{EDB}')}$

12: parse τ_a as (τ_1,AddSet)
13: $\mathrm{T}_e[\tau_1] \leftarrow \mathrm{AddSet}$

$\underline{\mathsf{Search}(\sigma,w;\mathrm{EDB})\rightarrow(\sigma',\mathrm{IND};\mathrm{EDB}')}$

$\mathrm{Client}(\sigma,w)\rightarrow(\sigma',\tau_s)$

1: **if** $\mathrm{T}_c[w]=\perp$ **then**
2:    **return** $\emptyset$
3: $\mathrm{key}^{(w)} \leftarrow \mathrm{T}_c[w]$; $\mathrm{T}_c[w] \xleftarrow{\$} \{0,1\}^{\lambda}$
4: $\tau_s \leftarrow (F_{k_1}(w),\mathrm{key}^{(w)})$
5: **Send** τ_s **to** Server

$\underline{\mathrm{Server}(\mathrm{EDB},\tau_\mathrm{s})\rightarrow(\mathrm{EDB}',\mathrm{Res})}$

6: parse τ_s as $(\alpha,\mathrm{key}^{(w)})$
7: $\mathrm{Res} \leftarrow \emptyset$
8: **for** each non-empty $(\mathrm{key},\mathrm{value})$ pair in T_e **do**
9:    $\tau_1 \leftarrow \mathrm{key}$; $\mathrm{ValueSet} \leftarrow \mathrm{value}$
10:    $mask \leftarrow H_1(\tau_1,\mathrm{key}^{(w)})$
11:    **for** each element $e \in \mathrm{ValueSet}$ **do**
12:       $\langle x,y\rangle \leftarrow e \oplus mask$
13:       **if** $y=\mathbf{0}$ **then**
14:          $\mathrm{Res} \leftarrow \mathrm{Res} \cup \{x\}$; $\mathrm{ValueSet} \leftarrow \mathrm{ValueSet} - e$
15:          $\mathrm{T}_e[\tau_1] \leftarrow \mathrm{ValueSet}$; **break**;
16: $\mathrm{Res} \leftarrow \mathrm{Res} \cup \mathrm{T_p}[\alpha]$; $\mathrm{T}_p[\alpha] \leftarrow \mathrm{Res}$
17: **Send** Res **to** Client

$\underline{\mathrm{Client}(\mathrm{Res})\rightarrow(\mathrm{DB}(w))}$

18: $\mathrm{DB}(w) \leftarrow \emptyset$
19: $\mathrm{DB}(w) \leftarrow \mathrm{Res}$
20: $\boxed{\textbf{for} \text{ each element } r \in \mathrm{Res} \textbf{ do} \quad \mathrm{ind} \leftarrow Dec(k_2,r); \mathrm{DB}(w) \cup \{\mathrm{ind}\}}$

$\underline{\mathsf{Deletion}(\sigma,\mathrm{ind};\mathrm{EDB})\rightarrow(\perp;\mathrm{EDB})}$

$\mathrm{Client}(\sigma,\mathrm{ind})\rightarrow(\tau_d)$

1: $\tau_d \leftarrow (F_{k_1}(\mathrm{ind}),\mathrm{ind})$
2: $\boxed{\tau_d \leftarrow (F_{k_1}(\mathrm{ind}),Enc(k_2,\mathrm{ind}))}$
3: **Send** τ_d **to** Server

$\underline{\mathrm{Server}(\mathrm{EDB},\tau_\mathrm{d})\rightarrow(\mathrm{EDB}')}$

4: parse τ_d as(τ_1,p)
5: $\mathrm{T}_e[\tau_1] \leftarrow \perp$
6: Delete all p in T_p

map T_e. When querying T_e, for every non-empty $(\mathrm{key},\mathrm{value})$ pair, the server performs as follows: It first generates $mask$ by computing $H_1(\mathrm{key},\mathrm{key}^{(\mathrm{w})})$; Then, for each element e in the set ValueSet, it recovers the plaintext $\langle x,y\rangle$ with $mask$, and checks whether y equals **0**. If so, x is the identifier of a matched document. Thus the server puts x into Res, deletes e from ValueSet, and starts to check the next non-empty $(\mathrm{key},\mathrm{value})$ pair in T_e. Otherwise, it goes on checking the next element in ValueSet. When querying T_p, it simply adds the previous search results stored at $\mathrm{T}_p[\alpha]$ into Res, which will be returned to the client. Meanwhile, it updates T_p by setting $\mathrm{T}_p[\alpha]$ with the latest results. Note that, for each matched document, its identifier will be added into T_p after the search phase, and corresponding element can be removed from T_e to avoid unnecessary testing in the subsequent search queries.

DBP-B.Deletion. To delete a document with identifier ind, the client sends $F_{k_1}(\mathrm{ind})$ and ind as the deletion token τ_d to the server, which deletes related contents in the map T_e and map T_p, respectively. For updating T_e, the server can directly remove the entry whose key is $F_{k_1}(\mathrm{ind})$. For updating T_p, the simplest approach is traversing each entry in T_p so as to search and delete ind accordingly. This approach suffers from computational inefficiency for a large database. As a trade-off, the server can amortize the traversal overhead over multiple searches by tracking all deleted document identifiers, and deleting them from T_p in the search phase.

C. Construction of DBP-E

DBP-E not only continues to use the notations and functions of DBP-B, but also introduces other notations. Let (Enc, Dec) be the encryption algorithm and decryption algorithm of some symmetric key encryption (SKE) scheme, respectively. The construction of DBP-E is similar to that of DBP-B except some subtle changes. Therefore, the details of DBP-E are also shown in Protocol 1 and the differences from DBP-B are marked as the boxed pseudocodes.

DBP-E.Setup. This algorithm is almost the same as DBP-B.Setup except that the client will generate two keys (k_1,k_2), where k_1 is the secret key of PRF F and k_2 is the secret key of SKE scheme.

DBP-E.Addition. To add document $f=(\mathrm{ind},\mathrm{DB(ind)})$ into the database, the client acts just like in DBP-B.Addition

except encrypting the identifier of the document. Specifically, the client first encrypts ind with the Enc algorithm of SKE scheme and then employs hash function H_1 to encrypt the encrypted document identifier.

DBP-E.Search. The difference between this protocol and the search protocol in DBP-B is that the server in this protocol only obtains encrypted identifiers from T_e and T_p. Therefore, the server sends all encrypted entries to the client, which will decrypt every encrypted entry with secret key k_2 and download the specific documents from the server.

DBP-E.Deletion. To resist external attackers, the previous search results are stored in T_p as the encrypted forms. Therefore, the main difference is deleting the entries in T_p. To delete a document with identifier ind, the client sends $Enc(k_2, \mathrm{ind})$ instead of ind to the server in addition to $F_{k_1}(\mathrm{ind})$. Upon receiving the deletion token, the server performs like in DBP-B.Deletion.

D. Improvement of Search Speed

The main deficiency of DBP is that its search time increases linearly with the number of document/keyword pairs in the map T_e. To improve the scalability of DBP, we adopt a *dual-key map structure* that uses two keys to identify the elements of T_e. Let $s_{i...j}$ denote a substring of s starting at the i-th bit and ending at the j-th bit. In the improved Addition algorithm, upon receiving the addition token $\tau_a = (F_{k_1}(\mathrm{ind}), \mathrm{AddSet})$, the server updates T_e as follows: For each element $e \in \mathrm{AddSet}$, the server stores a dual-key/value pair in the form of $(\langle \mathrm{key}_1, \mathrm{key}_2 \rangle, \mathrm{value})$ where $\mathrm{key}_1 = \mathrm{F}_{\mathrm{k}_1}(\mathrm{ind})$, $\mathrm{key}_2 = \mathrm{e}_{\lambda+1...2\lambda}$, and $\mathrm{value} = \mathrm{e}_{1...\lambda}$. That is, the concatenation of key_1 and key_2 is used as the key of T_e. In the improved Search algorithm, on receiving the search token $\tau_s = (F_{k_1}(w), \mathrm{key}^{(w)})$, the server first calculates $mask \leftarrow H_1(F_{k_1}(\mathrm{ind}), \mathrm{key}^{(w)})$ for each document identifier ind. Then, the server calculates a dual key $\langle F_{k_1}(\mathrm{ind}), mask_{\lambda+1...2\lambda} \rangle$, and tests whether a corresponding value exists in T_e or not. If so, the server recovers document identifier by calculating $\mathrm{ind} \leftarrow mask_{1...\lambda} \oplus \mathrm{value}$. Therefore, the search complexity can be reduced from $O(N_e)$ to $O(n_e)$, where N_e (resp. n_e) denotes the number of document/keyword pairs (resp. the number of documents) in T_e.

V. Security Analysis

In this section, we give the security analyses of our DBP scheme.

A. The Security Analysis of DBP-B

We provide the theorem regarding the adaptive security of DBP-B as follows:

Theorem 1. *If F is a secure PRF, then DBP-B is $\mathcal{L}$-adaptively-secure in the random oracle model, where the leakage functions $\mathcal{L}$ are defined as follows:*

- $\mathcal{L}_{set}(\lambda) = \emptyset$,
- $\mathcal{L}_{add}(\mathrm{ind}, \mathrm{DB}(\mathrm{ind})) = |\mathrm{DB}(\mathrm{ind})|$,
- $\mathcal{L}_{del}(\mathrm{ind}) = \mathrm{ind}$,
- $\mathcal{L}_{srh}(w) = (SP(w), TimeDB(w))$.

Proof. Given the leakage function collection $\mathcal{L} = (\mathcal{L}_{\mathsf{set}}, \mathcal{L}_{\mathsf{add}}, \mathcal{L}_{\mathsf{del}}, \mathcal{L}_{\mathsf{srh}})$ defined in Theorem 1, we can build a simulator $\mathcal{S}$ as shown in Simulation 2.

• **Random oracles**. The hash functions H_1 behaves like a random oracle. Simulator $\mathcal{S}$ maintains a hash table $\mathbb{H}_1$ that stores input/output pairs $(\mathbf{in}, \mathbf{out}) \in \{0,1\}^* \times \{0,1\}^{2\lambda}$.

• **Setup**. Simulation of Setup is identical to DBP-B.Setup except that it does not generate secret keys k_1 for PRF F, but initializes a counter v to 0.

• **Addition token**. Given leakage $\mathcal{L}_{\mathsf{add}}(\mathrm{ind}, \mathrm{DB}(\mathrm{ind})) = |\mathrm{DB}(\mathrm{ind})|$, an addition token τ_a is simulated as follows: The simulator $\mathcal{S}$ maintains a map FK to store $(\mathrm{ind}, \mathrm{key}_1^{(\mathrm{ind})})$ pairs. If an entry of FK is accessed for the first time, $\mathcal{S}$ sets it to random values. $\mathcal{S}$ also maintains a map Mask, where all values are selected randomly. Note that $\mathcal{S}$ chooses random strings instead of calling the PRF F to generate $\mathrm{key}_1^{(\mathrm{ind})}$. It is trivial to see that the real and simulated values are indistinguishable; otherwise, there exists an adversary $\mathcal{B}$ who can distinguish a PRF from a real random function. Therefore, the probability to distinguish $\mathrm{key}_1^{(\mathrm{ind})}$ from the outputs of PRFs is bounded by $\mathrm{Adv}_{\mathcal{B}}^{\mathrm{prf}}(\lambda)$.

• **Deletion token**. Given leakage $\mathcal{L}_{\mathsf{del}}(\mathrm{ind}) = \mathrm{ind}$, the deletion token τ_d is simulated by querying map FK.

• **Search token**. Given the leakage $\mathcal{L}_{\mathsf{srh}}(w) = (\mathsf{SP}(w), \mathsf{TimeDB}(w))$, a search token τ_s is simulated as follows. Let $\underline{w} = \min(\mathsf{SP}(w))$ denote a keyword, and let $\overline{w} = \max(\mathsf{SP}(w))$ denote the last timestamp when keyword w has been searched.

The local map T_c is only updated in the search phase. If $\mathrm{T}_c[\underline{w}] = \emptyset$, it means that keyword w is searched for the first time, and $\mathrm{T}_c[\underline{w}]$ is initialized with a random string. Let $\mathsf{TimeDB}^t(w)$ denote the partial update history after the t-th query. The random oracle $\mathbb{H}_1$ is programmed s.t. that $\mathbb{H}_1[\mathrm{key}^{(\underline{\mathrm{w}})}, \mathrm{key}_1^{(\mathrm{ind})}] \leftarrow \mathsf{Mask}[\mathrm{v_i}]$.

However, $\mathbb{H}_1$ cannot be updated immediately during the simulation of the addition token. Therefore, there is certain probability for the adversary $\mathcal{A}$ to observe inconsistency while querying the random oracle. Let $\mathbf{poly}(\lambda)$ denote the polynomial function in λ. Note that, the probability for $\mathcal{A}$ to guess $(\mathrm{key}^{(\underline{\mathrm{w}})}, \mathrm{key}_1^{(\mathrm{ind})}])$ is $2^{-\lambda}$ since they are random strings. A PPT adversary can make at most $\mathbf{poly}(\lambda)$ guesses, and the probability that such an event occurs is $\mathbf{poly}(\lambda)/2^{\lambda}$.

• **Conclusion**. By combing all simulation results, we can say that, for any PPT adversary $\mathcal{A}$, there exists a PRF-adversary $\mathcal{B}$ such that

$$\begin{aligned}&|\Pr[\mathbf{Real}_{\mathcal{A}}(\lambda) = 1] - \Pr[\mathbf{Ideal}_{\mathcal{S},\mathcal{A}}(\lambda) = 1]| \\ &\leq \mathrm{Adv}_{\mathcal{B}}^{\mathrm{prf}}(\lambda) + \frac{\mathbf{poly}(\lambda)}{2^{\lambda}}\end{aligned}$$

We thus conclude the resulting probability is $\mathbf{negl}(\lambda)$ by assuming that the PRF is secure. Note that DBP provides backward and forward security because $\mathcal{L}_{\mathsf{add}}$ leaks only $|\mathrm{DB}(\mathrm{ind})|$ and $\mathcal{L}_{\mathsf{srh}}$ leaks only $(\mathsf{SP}(w), \mathsf{TimeDB}(w))$, thus $\overline{\mathcal{L}}'$ and $\widetilde{\mathcal{L}}$ can be defined as $\mathcal{L}_{\mathsf{del}}$ and $\mathcal{L}_{\mathsf{srh}}$, respectively. □

Simulation 2 Simulator $\mathcal{S}$

<u>Programming the random oracles(**in**)</u>

1: **if** $\exists(h_{i,1}, h_{i,2}) \in \mathbb{H}_i$ such that $h_{i,1} = \mathbf{in}$ **then**
2: **return** $h_{i,2}$
3: **else**
4: $\mathbf{out} \xleftarrow{\$} \{0,1\}^\lambda$; $\mathbb{H}_i[\mathbf{in}] \leftarrow \mathbf{out}$
5: **return** **out**

<u>Simulation of Setup</u>

1: $(\mathrm{T}_c, \mathrm{T}_e) \leftarrow$ empty map;
2: $\mathrm{EDB} \leftarrow (\mathrm{T_e})$; $v \leftarrow 0$
3: **return** $(\mathrm{EDB}, \mathrm{T_c})$

<u>Simulation of the addition token</u>

1: **if** $\mathsf{FK}[\mathrm{ind}] = \perp$ **then**
2: $\mathrm{key}_1^{(\mathrm{ind})} \xleftarrow{\$} \{0,1\}^\lambda$; $\mathsf{FK}[\mathrm{ind}] \leftarrow \mathrm{key}_1^{(\mathrm{ind})}$
3: $\mathrm{key}_1^{(\mathrm{ind})} \leftarrow \mathsf{FK}[\mathrm{ind}]$
4: $\mathrm{AddSet} \leftarrow \emptyset$
5: **for** $i \in |\mathrm{DB}(\mathrm{ind})|$ **do**
6: $m_i \xleftarrow{\$} \{0,1\}^{2\lambda}$; $\mathsf{Mask}[v] \leftarrow m_i$
7: $e \leftarrow m_i \oplus \langle \mathrm{ind}, \mathbf{0} \rangle$
8: $\mathrm{AddSet} \leftarrow \mathrm{AddSet} \cup \mathrm{e}$; $v \leftarrow v + 1$
9: **return** $\tau_a = (\mathrm{key}_1^{(\mathrm{ind})}, \mathrm{AddSet})$

<u>Simulation of the deletion token</u>

1: $\mathrm{key}_1^{(\mathrm{ind})} \leftarrow \mathsf{FK}[\mathrm{ind}]$; $\tau_d \leftarrow (\mathrm{key}_1^{(\mathrm{ind})}, \mathrm{ind})$

<u>Simulation of the search token</u>

1: $\underline{w} \leftarrow \min(\mathsf{SP}(w))$; $\overline{w} \leftarrow \max(\mathsf{SP}(w))$
2: **if** $\mathrm{T}_c[\underline{w}] = \emptyset$ **then**
3: $\mathrm{key}^{(\underline{\mathrm{w}})} \leftarrow \{0,1\}^\lambda$; $\mathrm{T}_c[\underline{w}] \leftarrow \mathrm{key}^{(\underline{\mathrm{w}})}$
4: $c \leftarrow |\mathsf{TimeDB}^{>\overline{w}}(w)|$
5: Parse $\mathsf{TimeDB}^{>\overline{w}}(w)$ as $(v_1, \mathrm{ind}_1), \ldots, (v_c, \mathrm{ind}_c)$
6: **if** $c = 0$ **then**
7: **return** $\emptyset$
8: **for** $i \in [c]$ **do**
9: $\mathrm{key}_1^{(\mathrm{ind})} \leftarrow \mathsf{FK}[\mathrm{ind_i}]$; $\mathbb{H}_1[\mathrm{key}^{(\underline{\mathrm{w}})}, \mathrm{key}_1^{(\mathrm{ind})}] \leftarrow \mathsf{Mask}[\mathrm{v_i}]$
10: $\mathsf{nkey}^{(\underline{w})} \leftarrow \{0,1\}^\lambda$; $\mathrm{T}_c[\underline{w}] \leftarrow \mathsf{nkey}^{(\underline{w})}$
11: **return** $\tau_s = \mathrm{key}^{(\underline{\mathrm{w}})}$

B. The Security Analysis of DBP-E

Theorem 2. *If F is a secure PRF and SKE is secure, then DBP-E is $\mathcal{L}$-adaptively-secure in the random oracle model.*

The security analysis is similar to the one in Section V-A. The only difference between two schemes is that the server can directly obtain ind when it receives the search token in DBP-B, while in DBP-E the server can only obtain the encrypted form of ind. Every time we add a document into database, the entries that the attacker observes are indistinguishable from random due to the random oracle and the pseudorandomness of F. Precisely speaking, the only thing that the protocol leaks in the addition phase is the number of keywords in a document. Similarly to the addition phase, the attacker observes during the deletion phase are indistinguishable from random. Besides, the attacker can learn the information that the keywords belong to one document. However, the identifier is the encrypted form. That is, the attacker cannot know any useful information between keywords and documents. In terms of backward privacy, the attacker has the knowledge of the number of entries related to the keyword w in the search phase, and the time every addition/deletion operation for keyword w took place. Beyond this, nothing else is revealed to the server.

VI. Evaluation

In this section, we will report on the performance of our DBP scheme, and the version with improved search speed is denoted by DBP*.

A. Performance Analysis

We theoretically analyze the performance of our scheme in terms of data size and communication/computational complexities. Let $k = |\mathrm{DB}(\mathrm{ind})|$ be the number of keywords in document ind, let a_w be the number of added documents containing keyword w, and let d_w be the number of deleted documents including keyword w. On the client side, the computational complexity for search and deletion is $O(1)$, and for addition is $O(k)$. As for storage costs, the client locally maintains the secret key and a map T_c. Therefore, the space complexity at the client side is $O(m)$. On the server side, the computational complexity for update is $O(k)$. As for searching a keyword, the complexity of DBP (resp. DBP*) is linear with the number of document/keyword pairs (resp. the number of documents) in the map T_e. As for storage costs, the server keeps EDB that consists of a map T_e and T_p. During the search phase, the identifier of matched document will be added into T_p after the search process, and corresponding element can be removed from T_e. Therefore, the size of EDB is $O(N)$. In addition, the addition token is of size $O(k)$, both the deletion token and the search token are of size $O(1)$, and the search results are of size $O(a_w - d_w)$.

TABLE II
Descriptions of datasets

Dataset	Data size(GB)	Documents number	Keywords number	Pairs number
Enron[1]	1.56	577,744	54,021	2,888,720
Wikipedia[2]	10.6	3,506,761	957,920	17,533,805

B. Experiment Environment and Dataset

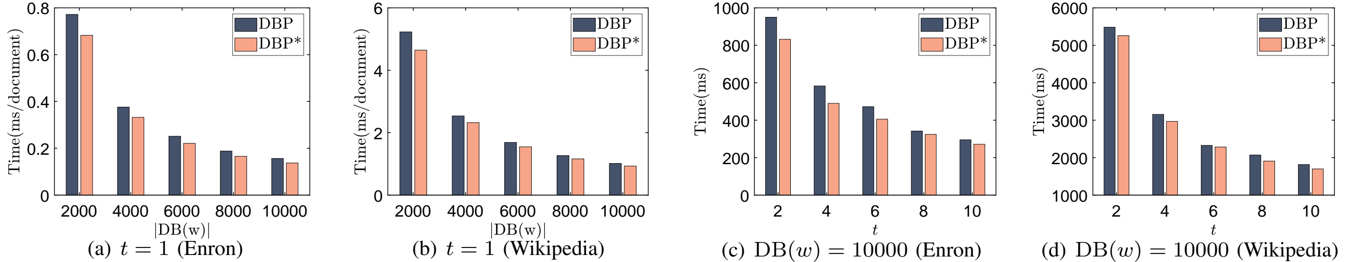
We evaluate the performance of our scheme in a server with an Intel(R) Xeon(R) Gold 5218 CPU of 2.3GHz and 128GB

[1] https://www.cs.cmu.edu/∼./enron/
[2] https://dumps.wikimedia.org/

TABLE III
PERFORMANCE OF DBP SCHEME IN SETUP AND UPDATE.

Dataset	DBP-B					DBP-E				
	Setup			Addition	Deletion	Setup			Addition	Deletion
	Time(ms)	Size1(MB)	Size2(MB)	Time(μs)	Time(μs)	Time(ms)	Size1(MB)	Size2(MB)	Time(μs)	Time(μs)
Enron	8727	2.78	274	15.11	2.25	11471	2.78	274	19.85	8.57
Wikipedia	60073	50.07	1662	18.18	2.30	75955	50.07	1662	21.66	9.16

Size1: the storage cost (σ) at the client side, Size2: the storage cost (EDB) at the server side.



(a) $t = 1$ (Enron) (b) $t = 1$ (Wikipedia) (c) $\mathrm{DB}(w) = 10000$ (Enron) (d) $\mathrm{DB}(w) = 10000$ (Wikipedia)

Fig. 1. Comparison of search performance at the server side. (a) The time for finding out per documents matching keyword w. (b) The time for finding out per documents matching keyword w. (c) The time for finding out all documents matching keyword w. (d) The time for finding out all documents matching keyword w.

memory, running Microsoft Windows Server 2016 Standard operating system. The code is written in Java and compiled using IntelliJ IDEA 11.0.8. During the evaluation, the security parameter λ is set to 256, HMAC is used for PRF F, SHA-512 is used for the keyed hash function H_1 and AES-256 is employed for SKE.

We conduct experiments on two real datasets, Enron Email and Wikipedia Dumps. Before experiments, two datasets are preprocessed by extracting 5 keywords from each document with the TextRank algorithm and TF-IDF model. The statistic properties of the pretreated datasets are shown in Table II.

C. Experiment Results

We provide the experiment results for the data size and execution time. The whole EDB is loaded into the memory before measuring the execution time, thereby I/O time is excluded from our experiments.

Setup. The data size and the client-side execution time for constructing two datasets are shown in Table III. For the same dataset, the data size of both schemes is the same. In terms of execution time on the client side, DBP-E requires utilizing SKE to encrypt the identifier of document, resulting in more execution time than DBP-B. In addition, the data size and the client-side execution time of both schemes increase with the increase of the dataset's size.

Update. Table III also describes the execution time for updating each document at the client side. The results of adding single document are consistent with those in setup phase. As for deleting one document, DBP-E takes more time than DBP-B. The reason is DBP-E needs to encrypt the identifier of the document to be deleted according to SKE scheme.

Search. The main difference of DBP-B and DBP-E is that the latter needs to recover the plaintext identifier at the client side. As for the search phase at the server side, both schemes have the same time overhead. Therefore, we use DBP and DBP* to represent the two schemes and the improved version of two schemes, respectively. As for the time to find all matched documents, DBP (resp. DBP*) asks the server to examine all the document/keyword pairs (resp. all the documents) in the map T_e. Therefore, the execution time in DBP and DBP* depends on the number of document/keyword pairs, N_e, and the number of documents, n_e, respectively.

Fig. 1-(a) shows the impact of $|\mathrm{DB}(w)|$ on the execution time while searching one document from the Enron Email dataset. From this figure, we know that the search time in both DBP and DBP* decreases as $|\mathrm{DB}(w)|$ grows. Furthermore, it is obvious that DBP* improves the performance of DBP. For example, given a keyword contained by 2000 documents, the execution time of DBP and DBP* is about 0.77ms/document and 0.68ms/document, respectively. From Fig. 1-(b), we know that the search time for one document increases compared with Enron Email. For example, given a keyword contained by 2000 documents, the execution time of DBP is about 0.77ms/document and 5.23ms/document in Enron Email dataset and Wikipedia Dumps dataset, respectively. That is caused by the increased data size of Wikipedia Dumps.

To make our scheme more practical, we extend DBP and DBP* into multiple threads setting and the results are shown in Fig. 1-(c) and (d). From these figures, we know that the complexity of both DBP and DBP* is negatively impacted by the number of threads t. For example, given a keyword contained by $10,000$ documents in Wikipedia Dumps dataset, the search time of DBP decreases from 5483ms to 1814ms as t increases from 2 to 10 in Fig. 1-(d).

VII. Related Work

The first SSE scheme was proposed by Song et al. [3], but no rigorous security definition of SSE presented before the work of Curtmola et al. [4]. However, SSE schemes do not allow updating the encrypted database once it has been outsourced. To solve this problem, Kamara et al. [5] proposed the first DSSE scheme with support for sublinear search time. Follow-up work is committed to enriching various functionalities, such as expressive queries [6], rank searches [7] and verifiability [8]. In 2016, file-injection attack proposed by Zhang et al. [9] highlighted the importance of forward privacy in the designing of DSSE schemes. Chang et al. [10] proposed the first FP scheme according to changing search token after each update. The main drawback of their scheme is that the query size is liner with the number of updates on keyword. Stefanov et al. [11] proposed an FP construction based on hierarchical oblivious RAM, which will incur high communication cost. To reduce communication complexity, Bost [12] proposed an FP scheme Sophos based on one-way trapdoor permutation, but fell short of support for actual deletion until the work [13].

Another important security property of DSSE schemes is backward privacy, which was first formally described in [14]. In this work, the authors defined backward privacy at three levels (Type-I to Type-III, ordered from the most to the least secure) and gave some backward private constructions with different security/efficiency trade-offs. Their first scheme Moneta achieves Type-I BP by utilizing obvious RAM, and second scheme Fides realizes Type-II BP on the basis of Sophos [12]. $\mathsf{Diana}_{\mathsf{del}}$ and Janus provide better performance at the cost of security (both are Type-III BP). In [15], Sun et al. proposed a Type-III BP scheme Janus++, which is more efficient than Janus because it is based on symmetric puncturable encryption. In [16], Chamani et al. utilized obvious RAM to achieve a Type-I BP scheme Orion and a Type-III BP scheme Horus, respectively. Besides, they proposed a Type-II BP scheme Mitra, which had better performance than Fides [14] due to symmetric cryptographic primitive. He et al. [17] designed a forward and backward private scheme while reducing client storage according to fish-bone index. Zuo et al. [18] employed symmetric encryption with homomorphic additions to achieve backward privacy. However, the above BP schems [14]–[18] assume that the client needs to know all keyword contained in the document to be deleted in advance. Li et al. [19] proposed a BP scheme which eliminated this assumption but the storage cost at client is liner with the number of documents in dataset.

VIII. Conclusion

In this paper, we proposed a DBP scheme to achieve secure and effective search services in cloud computing. The proposed scheme supports document-based deletion while ensuring forward and backward privacy. Experiment results demonstrate that our scheme is efficient and practical. However, our scheme only supports single keyword queries. As part of our future work, we will make our scheme support rich semantics.

Acknowledgments

This work was supported in part by NSFC grants 61632009, 61872133, 61872130, and 61572181; NSF grants CNS 1824440, CNS 1828363, CNS 1757533, CNS 1629746, CNS 1651947, and CNS 1564128; the CERNET Innovation Project (NGII20190409); the Guangdong Provincial Natural Science Foundation (No. 2017A030308006), and the Hunan Provincial Natural Science Foundation of China (Grant No. 2020JJ3015).

References

[1] X. Liu, Q. Liu, T. Peng, and J. Wu, "Dynamic access policy in cloud-based personal health record (PHR) systems," *Information Sciences*, 2017.

[2] L. Du, K. Li, Q. Liu, Z. Wu, and S. Zhang, "Dynamic multi-client searchable symmetric encryption with support for boolean queries," *Information Sciences*, 2019.

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

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

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

[6] Q. Liu, Y. Peng, J. Wu, T. Wang, and G. Wang, "Secure multi-keyword fuzzy searches with enhanced service quality in cloud computing," *IEEE Transactions on Network and Service Management*, 2020.

[7] Z. Xia, X. Wang, X. Sun, and Q. Wang, "A secure and dynamic multi-keyword ranked search scheme over encrypted cloud data," *IEEE Transactions on Parallel and Distributed Systems*, 2016.

[8] Q. Liu, Y. Tian, J. Wu, T. Peng, and G. Wang, "Enabling verifiable and dynamic ranked search over outsourced data," *IEEE Transactions on Services Computing*, 2019.

[9] Y. Zhang, J. Katz, and C. Papamanthou, "All your queries are belong to us: the power of file-injection attacks on searchable encryption," in *Proc. of USENIX Security*, 2016.

[10] Y.-C. Chang, and M. Mitzenmacher, "Privacy preserving keyword searches on remote encrypted data," in *Proc. of ACNS*, 2005.

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

[12] R. Bost, "Σοφος: Forward secure searchable encryption," in *Proc. of CCS*, 2016.

[13] K. S. Kim, M. Kim, D. Lee, J. H. Park, and W.-H. Kim, "Forward secure dynamic searchable symmetric encryption with efficient updates," in *Proc. of CCS*, 2017.

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

[15] S. Sun, X. Yuan, J. K. Liu, R. Steinfeld, A. Sakzad, V. Vo, and S. Nepal, "Practical backward-secure searchable encryption from symmetric puncturable encryption," in *Proc. of CCS*, 2018.

[16] J. G. Chamani, D. Papadopoulos, C. Papamanthou, and R. Jalili, "New constructions for forward and backward private symmetric searchable encryption," in *Proc. of CCS*, 2018.

[17] K. He, J. Chen, Q. Zhou, R. Du, and Y. Xiang, "Secure dynamic searchable symmetric encryption with constant client storage cost," *IEEE Transactions on Information Forensics and Security*, 2021.

[18] C. Zuo, S. Sun, J. Liu, J. Shao, J. Pieprzyk, "Dynamic searchable symmetric encryption with forward and stronger backward privacy," in *Proc. of ESORICS*, 2019.

[19] J. Li, Y. Huang, Y. Wei and S. Lv, Z. Liu, C. Dong, and W. Lou, "Searchable symmetric encryption with forward search privacy," *IEEE Transactions on Dependable and Secure Computing*, 2021.

[20] Q. Liu, Y. Peng, S. Pei, J. Wu, T. Peng and G. Wang, "Prime inner product encoding for effective wildcard-based multi-keyword fuzzy search," *IEEE Transactions on Services Computing*, 2020.

[21] J. Katz and Y. Lindell, "Introduction to modern cryptography," *CRC press*, 2014.