Secure Dynamic Searchable Symmetric Encryption With Constant Client Storage Cost

Kun He, Jing Chen, *Member, IEEE*, Qinxi Zhou, Ruiying Du, and Yang Xiang, *Fellow, IEEE*

***Abstract*— Dynamic Searchable Symmetric Encryption (DSSE) enables users to search on the encrypted database stored on a semi-trusted server while keeping the search and update information under acceptable leakage. However, most existing DSSE schemes are not efficient enough in practice due to the complex structures and cryptographic primitives. Moreover, the storage cost on the client side grows linearly with the number of keywords in the database, which induces unaffordable storage cost when the size of keyword set is large. In this article, we focus on secure dynamic searchable symmetric encryption with constant client storage cost. Our framework is boosted by fish-bone chain, a novel two-level structure which consists of Logical Keyword Index Chain (LoKIC) and Document Index Chain (DIC). To instantiate the proposed framework, we propose a forward secure DSSE scheme, called CLOSE-F, and a forward and backward secure DSSE scheme, called CLOSE-FB. Experiments showed that the computation cost of CLOSE-F and CLOSE-FB are as efficient as the state-of-the-art solutions, while the storage costs on the client side are constant in both CLOSE-F and CLOSE-FB, which are much smaller than existing schemes.**

***Index Terms*— Searchable symmetric encryption, forward security, backward security.**

I. Introduction

As CLOUD services are widely used by individuals and enterprises to store their important data, users prefer to encrypt sensitive data before storing them on the cloud to protect their privacy. Though encryption provides a strong security guarantee, it also prevents cloud servers from performing useful operations at the same time, such as search and calculation. *Symmetric Searchable Encryption* (SSE) was then introduced by the research community to realize keyword search in the ciphertext, which is one of the most basic data operations [1].

Early research focused on the *static* case, in which a user outsources his/her private documents to a semi-trusted server, and later issues search tokens to the server to perform keyword search without revealing sensitive information [2]–[6]. However, static SSE does not support many document operations, i.e., creation, update, and deletion, which are required in many practical applications. The researchers then draw their attention to *Dynamic* SSE (DSSE) [7]–[11].

A DSSE scheme includes a structure and the operations over it for storing, searching, and updating keyword/document pairs [12]. Although a lot of works are devoted to the security and efficiency of DSSE, existing schemes still face significant challenges. 1) Many solutions employed ingenious structures (e.g., inverted index), whose operations have good asymptotic performance in theory. However, the performance is poor in practice because of the heavy cryptographic operations [9], [10] or constrained update problems [11]. 2) The state stored on the client for searching and updating those structures grows linearly with the number of keywords which may introduce a heavy burden on the client [13], [14]. 3) Those (even encrypted) state on the client side may leak some information, such as the keywords and the number of keywords. In short, there is still a lack of an efficient structure with practical operations over it for search and update processes in DSSE that can be securely stored without leaking additional information.

In this article, we aim to design a framework which implements **C**onstant c**L**ient st**O**rage cost for dynamic **S**earchable symmetric **E**ncryption, named CLOSE. Instead of the traditional structure, such as inverted index, our construction is boosted by a novel two-level structure, named *fish-bone chain*, which binds all keywords to a single state. The first level is a logical structure of search tokens for a keyword, called *Logical Keyword Index Chain* (LoKIC). With this structure, we can reduce the storage cost on the client side, which solves the second and third challenges. To solve the first challenge, we designed another structure at the second level, called *Document Index Chain* (DIC) for optimizing the computation cost and reducing the rebuilding overhead.

Recently, file-injection attacks [15] have shown that many DSSE schemes are not as secure as expected if an adversary has some knowledge about the added documents. To defend such attacks, the basic security requirement for a DSSE scheme is *forward security*, which means that the server

Manuscript received October 28, 2019; revised June 16, 2020 and August 20, 2020; accepted October 11, 2020. Date of publication October 23, 2020; date of current version December 11, 2020. This work was supported in part by the National Natural Science Foundation of China under Grant 61702379, Grant U1836202, and Grant 61772383; in part by the China Postdoctoral Science Foundation under Grant 2019T120685; and in part by the Joint Fund of Ministry of Education of China for Equipment Pre-Research under Grant 6141A02033341. The associate editor coordinating the review of this manuscript and approving it for publication was Dr. Lejla Batina. *(Corresponding author: Jing Chen.)*

Kun He and Qinxi Zhou are with the Key Laboratory of Aerospace Information Security and Trusted Computing, Ministry of Education, School of Cyber Science and Engineering, Wuhan University, Wuhan 430072, China.

Jing Chen is with the School of Cyber Science and Engineering, Wuhan University, Wuhan 430072, China, and also with the Shenzhen Institute, Wuhan University, Wuhan 430072, China (e-mail: chenjing@whu.edu.cn).

Ruiying Du is with the Key Laboratory of Aerospace Information Security and Trusted Computing, Ministry of Education, School of Cyber Science and Engineering, Wuhan University, Wuhan 430072, China, and also with the Collaborative Innovation Center of Geospatial Technology, Wuhan University, Wuhan 430072, China.

Yang Xiang is with the School of Software and Electrical Engineering, Swinburne University of Technology, Hawthorn, VIC 3122, Australia.

Digital Object Identifier 10.1109/TIFS.2020.3033412

1556-6013 © 2020 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See https://www.ieee.org/publications/rights/index.html for more information.

Authorized licensed use limited to: Wuhan University. Downloaded on August 01,2023 at 12:04:40 UTC from IEEE Xplore. Restrictions apply.

TABLE I
COMPARISON OF SEVERAL SSE SCHEMES

Scheme	Computation		Communication			Client Storage
	Search	Update	Search	Round	Update	
SPS [17]	$O(\min\{a_w + \log N, n_w \log^3 N\})$	$O(\log^2 N)$	$O(n_w + \log N)$	1	$O(\log N)$	$O(N^\alpha)$
TWORAM [9]	$\tilde{O}(a_w \log N + \log^3 N)$	$\tilde{O}(\log^2 N)$	$\tilde{O}(a_w \log N + \log^3 N)$	2	$\tilde{O}(\log^3 N)$	$O(1)$
Sophos [10]	$O(a_w)$	$O(1)$	$O(n_w)$	1	$O(1)$	$O(m)$
KKL+ [18]	$O(a_w)$	$O(1)$	$O(n_w)$	1	$O(1)$	$O(m)$
Fides [11]	$O(a_w)$	$O(1)$	$O(a_w)$	2	$O(1)$	$O(m)$
$\text{Diana}_{\text{del}}$ [11]	$O(a_w)$	$O(\log a_w)$	$O(n_w + n_w \log a_w)$	2	$O(1)$	$O(m)$
Janus [11]	$O(n_w \cdot d_w)$	$O(1)$	$O(n_w)$	1	$O(1)$	$O(m)$
Khons [19]	$O(n_w)$	$O(1)$	$O(n_w)$	2	$O(1)$	$O(m + n)$
FAST [20]	$O(a_w)$	$O(1)$	$O(n_w)$	1	$O(1)$	$O(m)$
EKP [21]	$O(a_w)$	$O(1)$	$O(n_w)$	1	$O(1)$	$O(m + n)$
Mitra [22]	$O(a_w)$	$O(1)$	$O(a_w)$	2	$O(1)$	$O(m)$
SD_a [14]	$O(a_w + \log N)$	$O(\log N)$	$O(a_w + \log N)$	1	$O(\log N)$	$O(1)$
SD_d [14]	$O(a_w + \log N)$	$O(\log^3 N)$	$O(a_w + \log N)$	1	$O(\log^3 N)$	$O(1)$
CLOSE-F (Sec. IV)	$O(a_w + CLen)$	$O(CLen)$	$O(a_w)$	1	$O(1)$	$O(1)$
CLOSE-FB (Sec. V)	$O(a_w + CLen)$	$O(CLen)$	$O(a_w)$	2	$O(1)$	$O(1)$

N is the number of keyword/document pairs (i.e., total entries) in the database. n is the number of documents and m is the number of distinct keywords. n_w is the size of the search result set for keyword w, a_w is the number of entries that matching keyword w was added to the database, and d_w is the number of entries that matching keyword w was deleted from the database (i.e., $n_w = a_w - d_w$). $CLen$ is a constant of the maximum number of hash function computations. The $\tilde{O}$ notation hides the $\log \log N$ factors. Update complexities are given per update keyword/document pair. All schemes in the table achieve forward security and the schemes in [11], [19], [22], [14], and CLOSE-FB achieve backward security. The server's storage complexities are optimal $O(N)$.

cannot learn whether the newly added documents match the previous search tokens (until new related search tokens are generated) [16]. To this end, we propose a forward secure DSSE scheme based on a carefully instantiated fish-bone chain structure, called CLOSE-F.

Another significant security property is *backward security*, which indicates that the server is not able to figure out whether the deleted documents contain the keyword that is being searched [17]. Although backward security was not formally defined until [11], researchers have considered it as an important direction [11], [22]. To achieve backward security, we modify the fish-bone chain in CLOSE-F and propose a forward and backward secure scheme, called CLOSE-FB.

Our main contributions are summarized as follows.

- To the best of our knowledge, CLOSE is the first efficient and secure DSSE framework with constant client storage cost. Furthermore, the computation complexity of CLOSE-F and CLOSE-FB in the search and update processes are nearly optimal in both theory and practice. The comparison of CLOSE-F and CLOSE-FB with previous ones is shown in Table I.
- We design a novel two-level structure, called fish-bone chain, which comprises of the proposed Logical Keyword Index Chain (LoKIC) and Document Index Chain (DIC). With this structure, users can generate forward (and backward) secure search tokens without maintaining a state list for every keyword.
- We prove that CLOSE-F and CLOSE-FB are secure against semi-trusted adversaries. We also implement CLOSE-F and CLOSE-FB, and then compare their performance with previous works. The experimental results show that our schemes are more efficient than the state-of-the-art solutions.

The rest of this article is organized as follows. We introduce the preliminaries in Section II. In Section III, we present our two-level structure, fish-bone chain, for our CLOSE framework. Then, the forward secure scheme CLOSE-F and the forward and backward secure scheme CLOSE-FB are detailed in Section IV and V, respectively. Experiments and evaluation are detailed in Section VI. Finally, we state the related work in Section VII, and conclude our paper in Section VIII.

II. PRELIMINARIES

A. Notations

For a finite set X, $x \xleftarrow{\$} X$ means that x is uniformly selected from X. $|X|$ denotes the cardinality of the set X. Operator $\|$ denotes the concatenation of strings. $\{0,1\}^l$ denotes the set of all binary strings of length l and $\{0,1\}^*$ is the set of all binary strings of finite length.

$\lambda \in \mathbb{N}$ denotes the security parameter. Unless specified explicitly, the symmetric keys are strings of λ bits, and the key generation algorithm uniformly samples a key from $\{0,1\}^\lambda$. We only consider (probabilistic) algorithms and protocols running in polynomial time in λ. Particularly, adversaries are Probabilistic Polynomial-Time (PPT) algorithms.

A function $negl : \mathbb{N} \rightarrow \mathbb{N}$ is negligible in λ, if for all positive polynomial $p(\cdot)$ and all sufficiently large λ, we have $negl(\lambda) < 1/p(\lambda)$.

A two-party protocol P between a client and a server is denoted by $\mathsf{P}(cin; sin) \rightarrow (cout; sout)$, which means the client takes cin as input and outputs $cout$, and the server takes sin as input and outputs $sout$.

B. Dynamic Searchable Symmetric Encryption (DSSE)

A document $doc = (ind, W_{ind})$ is labeled by an identifier $ind \in \{0,1\}^l$ and consists of a set of keywords $W_{ind} \subseteq \{0,1\}^*$.

Authorized licensed use limited to: Wuhan University. Downloaded on August 01,2023 at 12:04:40 UTC from IEEE Xplore. Restrictions apply.

TABLE II
DESCRIPTION OF NOTATIONS IN OUR DSSE SCHEMES

Notation	Description
doc	a document
ind	the identifier of the document
W_{ind}	the set of keywords which are consisted in document ind
DB	document database
EDB	encrypted document database
W	the set of keywords in the whole DB
$DB(w)$	the set of documents containing the keyword w
n	the number of documents in DB
kt_w	the keyword token for the keyword w
st_w	the search token for the keyword w
$CLen$	the system constant
H	the hash function
F	the pseudo-random function

Then, the database DB can be defined as $(ind_i, W_{ind_i})_{i=1}^n$, where n is the number of documents in the database DB. The set of keywords in the whole DB is $W = \bigcup_{i=1}^n W_{ind_i}$, and the set of documents containing a keyword w is denoted by $DB(w) = \{ind_i \mid w \in W_{ind_i}\}$. In this article, N is the number of keyword/document pairs, and $m = |W|$ is the total number of keywords. Obviously, N can be written as $N = \sum_{i=1}^n |W_{ind_i}| = \sum_{w\in W} |DB(w)|$. The notations in our DSSE schemes are shown in Table II

A *dynamic searchable symmetric encryption* (DSSE) scheme $\Pi = \{\mathsf{Setup}, \mathsf{Search}, \mathsf{Update}\}$ is comprised of three protocols between a client and a server.

- $\mathsf{Setup}(\lambda; \perp) \rightarrow (\sigma; EDB)$ is a setup protocol. The client takes a security parameter λ as input and creates a state σ. The server initializes an encrypted database EDB.
- $\mathsf{Search}(\sigma, w; EDB) \rightarrow (\sigma', DB(w); EDB')$ is a search protocol. The client takes the state σ and a keyword w as input, and the server takes the encrypted database EDB as input. After the execution of this protocol, the client gets the search result $DB(w)$ and the updated state σ'. The server gets the updated encrypted database EDB'.
- $\mathsf{Update}(\sigma, DOC, op; EDB) \rightarrow (\sigma'; EDB')$ is a update protocol. The client takes the state σ, the documents set DOC, and the operation op as input, and gets the new state σ' after the update. The documents set DOC consists of a number of keyword/document pairs. The operation op is taken from the set $\{add, del\}$ which respectively means the addition and deletion of document set DOC. Note that all update operations act on the keyword/document pair (w, ind) rather than the document (ind, W_{ind}). The server takes the encrypted database EDB as input and finally gets the updated encrypted database EDB'.

C. System Model

The system model of CLOSE is shown in Fig. 1. There are only two entities in the system: a client and a server. The client can upload/update the encrypted documents (which are stored in the fish-bone chain structure in our constructions) to the server. Then, the client can generate a search token for a particular keyword and sends it to the server to obtain the search results which consists of encrypted documents containing the specific keyword. When the client receives the encrypted documents, it can decrypt them and obtain the plain documents.

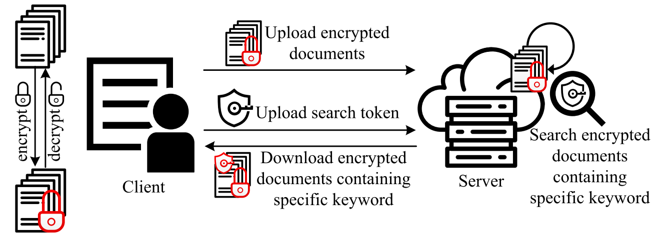


Fig. 1. The system model of CLOSE.

CLOSE is designed for the system in which the encrypted documents are periodically updated. For example, the client may retrieve emails from a mail server daily and add them into the encrypted documents. Periodical update can be performed on powerful devices such as personal computers. On the other hand, searching needs to be performed at any time on any device such as smartphones.

1) Threat Model: We consider a threat model that the server is honest-but-curious [10]. In this model, the server will honest perform the protocols designed in our schemes, but it tries to obtain additional information from the encrypted documents and from each protocol. For example, the server may try to determine that whether newly added documents match a previous search token, i.e., violating forward security.

D. Security Definitions

Besides those schemes based on expensive and powerful techniques (e.g., multi-party computation, fully homomorphic encryption, and oblivious RAM), all existing SSE schemes leak more or less information, such as the number of documents in the result [1], [10], [17]. Therefore, as in most SSE schemes, we do not want the adversary to learn anything about the database and queries (i.e., search and update) beyond some explicit leakage. Formally, the security of SSE scheme is parameterized by a collection of *leakage functions*

$$\mathcal{L} = (\mathcal{L}_{\mathsf{Setup}}, \mathcal{L}_{\mathsf{Search}}, \mathcal{L}_{\mathsf{Update}}),$$

which describes the information that the protocols leak to the adversary.

The standard security definition of an DSSE scheme uses the real world versus ideal world formalization [2], [4], [7]. Formally, we define two games: $Real_{\mathcal{A}}^{\Pi}$ for the real world and $Ideal_{\mathcal{A}}^{\Pi}$ for the ideal world. A scheme Π is secure if the two games are indistinguishable, which means that the adversary learns no more than the output of the leakage functions.

- In $Real_{\mathcal{A}}^{\Pi}$, the DSSE scheme is executed honestly. The adversary $\mathcal{A}$ is given EDB generated by $\mathsf{Setup}(\lambda; \perp)$ as in the real case. The adversary can choose the keyword w and receives the transcripts generated from the protocol $\mathsf{Search}(\sigma, w; EDB)$. Moreover, the adversary can choose a documents set DOC and an operation op to receive the transcripts generated from the protocol

Authorized licensed use limited to: Wuhan University. Downloaded on August 01,2023 at 12:04:40 UTC from IEEE Xplore. Restrictions apply.

$\mathsf{Update}(\sigma, DOC, op; EDB)$. Eventually, the adversary outputs a bit $b \in \{0, 1\}$.

- In $Ideal^{\Pi}_{\mathcal{A}}$, the adversary gets a simulated transcript in place of the real transcripts of the protocols. The simulated transcripts are generated by a PPT simulator $\mathcal{S}$ who knows the output of leakage functions. The encrypted database given to the adversary is generated by $\mathcal{S}(\mathcal{L}_{\mathsf{Setup}}(\lambda; \perp))$. The adversary receives the transcripts generated from $\mathcal{S}(\mathcal{L}_{\mathsf{Search}}(w))$ and $\mathcal{S}(\mathcal{L}_{\mathsf{Update}}(DOC, op))$ when performing search and update operations. Eventually, the adversary outputs a bit $b \in \{0, 1\}$.

Definition 1: A dynamic searchable symmetric encryption scheme Π *is* $\mathcal{L}$-adaptively-secure *if for all PPT adversary* $\mathcal{A}$, *there exists a PPT simulator* $\mathcal{S}$, *such that*

$$|\Pr[Real^{\Pi}_{\mathcal{A}}(\lambda) = 1] - \Pr[Ideal^{\Pi}_{\mathcal{A}}(\lambda) = 1]| \leq negl(\lambda).$$

1) Common Leakage Function: The leakage function $\mathcal{L}$ keeps the track of search query list $\mathcal{Q}$. Each entry of $\mathcal{Q}$ is a pair (t, w). The integer t is a counter, which is initially set to 0, and incremented at each query, and w is the keyword in the search query. For each keyword w, *search pattern* $sp(w)$ is defined as [10]

$$sp(w) = \{t \mid (t, w) \in \mathcal{Q}\}.$$

In this article, we also use the notation of $HistDB(w)$ as in [10]. It is the list of documents containing keyword w historically added to DB, in the order of insertion. Particularly, once the documents are added, the document identifiers are included in the list whether they are deleted or not. For example, $HistDB(w) = \{ind_1, ind_2\}$ means that the ind_1 and ind_2 are historically added to DB, but we do not know if they have been deleted.

2) Forward Security: Forward security means that addition operations do not leak whether a previously searched keyword is in the added documents. We follow the formal definition in [10].

Definition 2: A dynamic searchable symmetric encryption scheme Π *is* forward security *if the update leakage function* $\mathcal{L}_{\mathsf{Upadate}}$ *can be written as*

$$\mathcal{L}_{\mathsf{Update}}(DOC, op) = \mathcal{L}'(\{ind_i, |W_{ind_i}|\}, op).$$

Definition 2 means that the forward secure scheme leaks less than operation, the identifiers, and the number of keywords in the latest updated documents.

3) Backward Security: Let $\mathcal{U}$ be the track of update list maintained by the leakage function $\mathcal{L}$, whose entry is a triple $(t, op, (w, ind))$. t is a counter as mentioned above, $op \in \{add, del\}$ is the operation acted on the keyword/document pair (w, ind). Then, as in [23], $TimeDB(w) = \{(t, ind) \mid (t, add, (w, ind)) \in U \, and \, \forall t', (t', del, (w, ind)) \notin U)\}$ denotes the list of all documents that contain the keyword w and not deleted, and $Updates(w) = \{t \mid (t, add, (w, ind)) \, or \, (t, del, (w, ind)) \in U\}$ denotes the list of the timestamp (the counter) of all updates on w. Backward security means that the search queries on keyword w do not reveal the documents which have been already deleted. We follow the formal definition in [23].

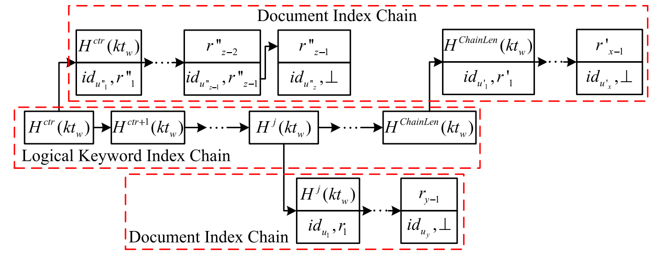


Fig. 2. An illustration of fish-bone chain.

Definition 3: A dynamic searchable symmetric encryption scheme Π *is* backward security *if the update leakage function* $\mathcal{L}_{\mathsf{Upadate}}$ *and the search leakage function* $\mathcal{L}_{\mathsf{Search}}$ *can be written as*

$$\mathcal{L}_{\mathsf{Update}}(DOC, op) = \mathcal{L}'(\{ind_i, |W_{ind_i}|\}, op),$$
$$\mathcal{L}_{\mathsf{Search}}(w) = \mathcal{L}''(TimeDB(w), Updates(w)).$$

III. Fish-Bone Chain: A Two-Level Structure

To design the CLOSE framework, we need a new structure that is different from previous works, in which size of the state is independent with the number of keywords. To this end, we propose a two-level structure, called *fish-bone chain*, which consists of Logical Keyword Index Chain (LoKIC) and Document Index Chain (DIC). We call this two-level structure fish-bone chain because that LoKIC looks like spine and DIC looks like ribs for a fish. This structure represents the encrypted database and is the core building block of our constructions, including CLOSE-F and CLOSE-FB. With the fish-bone chain structure, the state stored on the client side can be reduced to a constant. An illustration of fish-bone chain is shown in Fig. 2.

A. Logical Keyword Index Chain (LoKIC)

As explained in Section I, forward security is the basic requirements for a DSSE scheme. Therefore, the first step towards our structure design is to satisfy forward security. In most existing forward secure DSSE schemes, users need to maintain an *inverted index* locally, which stores the current state of each keyword for generating search tokens [13]. These states will be updated when keyword/document pairs are added or deleted. Therefore, the storage cost on the client side grows linearly with the number of keywords in the database.

To solve this problem, we design a Logical Keyword Index Chain (LoKIC) instead of the inverted index, which only stores a secret key and a global counter on the client. With this logical structure, users can generate the latest search tokens for any keyword without knowing the current state of that keyword. In other words, CLOSE achieves constant client storage cost via the design of LoKIC.

Let $H(\cdot)$ be a cryptographic hash function, a LoKIC for a keyword w consists of $CLen$ search tokens

$$H^{CLen}(kt_w), H^{CLen-1}(kt_w), \ldots, H(kt_w),$$

Authorized licensed use limited to: Wuhan University. Downloaded on August 01,2023 at 12:04:40 UTC from IEEE Xplore. Restrictions apply.

where kt_w is a secret *keyword token* for the keyword w, and

$$H^j(kt_w) = \begin{cases} H(kt_w), & j = 1, \\ H(H^{j-1}(kt_w)), & j \geq 2. \end{cases}$$

Given a search token $H^{ctr}(kt_w)$ for the keyword w and the global counter $1 \leq ctr \leq CLen$, the server can compute all the search tokens on LoKIC before $H^{ctr}(kt_w)$ by iteratively invoking the hash function. That means the server can obtain $CLen - ctr + 1$ search tokens.

$$H^{ctr}(kt_w), \cdots, H^{CLen}(kt_w).$$

However, because of one-way attribute of the hash function, the server cannot obtain any search token after $H^{ctr}(kt_w)$ on LoKIC, e.g., $H^{ctr-1}(kt_w)$.

The computation of hash function for ctr times on the client and $CLen - ctr$ times on the server makes extra search computation cost. Fortunately, the cost of extra computation is acceptable in practice as shown in Section VI.

B. Document Index Chain (DIC)

If each search token only corresponds to a single keyword/document pair as in [11], the global counter ctr will soon exhaust when the number of keyword/document pairs is large. Obviously, this approach incurs heavy rebuilding cost.

To tackle this problem, we design another structure on the top of LoKIC, called Document Index Chain (DIC). Roughly speaking, each node (i.e., search token) on LoKIC corresponds to a (maybe empty) DIC, and each node on DIC encodes a keyword/document pair for a certain keyword in one update (how to encode a keyword/document pair depends on the security requirement of the DSSE scheme). With this DIC structure, we can update database for $CLen$ times no matter how many documents are included in one update.

Since the LoKIC is only a logic structure, the encrypted database EDB is actually the set of DICs which store all keyword/document pairs.

C. Construction of Fish-Bone Chain

Let $lookup(DICT, key)$ be a function that returns the element labeled by key in a dictionary $DICT$. If there is no element labeled by key, this function will return a special element $\perp$. A fish-bone chain scheme consists of two algorithms (CSearch, CUpdate) as follows.

1) Chain Update Algorithm CUpdate$(st, dic, msg) \rightarrow dic'$*:* This algorithm takes as input a search token st, a DIC dic with respect to st, and a message msg, and updates the DIC dic' that has msg encoded. The detail is shown in Algorithm 1.

In our construction, the head node of a DIC with respect to a search token st is always labeled by $H(st\|0)$. Therefore, the algorithm first checks whether the DIC is empty by retrieving the node labeled by $H(st\|0)$ (Line 1-2). If dic is empty, a head node is created and msg is encoded into this node, where $\perp$ means that this node is the last node on dic (Line 4-5); otherwise, a new head node will replace the old one and these two nodes are connected by a random token rt (Line 7-10). Finally, the updated dic' is returned (Line 12).

Algorithm 1 CUpdate$(st, dic, msg) \rightarrow dic'$

```
1: key ← H(st||0)
2: value ← lookup(dic, key)
3: if value = ⊥ then
4:     value ← H(st||1) ⊕ (msg||⊥)
5:     dic ← dic ∪ {(key, value)}
6: else
7:     dic ← dic \ {(key, value)}
8:     rt ←$ {0, 1}^λ
9:     dic ← dic ∪ {(key, H(st||1) ⊕ (msg||rt))}
10:    dic ← dic ∪ {(H(rt||0), H(rt||1) ⊕ H(st||1) ⊕ value)}
11: dic' ← dic
12: return dic'
```

Algorithm 2 CSearch$(EDB, CLen, ctr, st) \rightarrow res$

```
1: j ← ctr;  RES ← ∅
2: while j ≤ CLen do
3:     key ← H(st||0)
4:     value ← lookup(EDB, key)
5:     if value ≠ ⊥ then
6:         (msg||rt) ← value ⊕ H(st||1)
7:         while rt ≠ ⊥ do
8:             res ← res ∪ {msg}
9:             value ← lookup(EDB, H(rt||0))
10:            (msg||rt) ← value ⊕ H(rt||1)
11:        res ← res ∪ {msg}
12:    j ← j + 1
13:    st ← H(st)
14: return res
```

2) Chain Search Algorithm CSearch$(EDB, CLen, ctr, st) \rightarrow res$*:* This algorithm takes as input an encrypted database EDB (the set of DICs), a system constant $CLen$, a global counter ctr, and a search token st, and outputs a search result res with respect to st. The detail is shown in Algorithm 2.

For a given search token st, this algorithm traverses from the $(CLen - ctr + 1)$-th node on LoKIC to the first one (Line 2-13). More specifically, this algorithm computes $CLen - ctr$ search tokens $H(st), \cdots, H^{CLen-ctr}(st)$ and uses these search tokens along with st to obtain all non-empty DICs. For each non-empty DIC whose head node is labeled by $H(st\|0)$, this algorithm traverses this DIC and retrieves all encoded msg (Line 6-11). The traversal of a DIC depends on the random token rt encoded in each node on the DIC. Finally, a set res of all msg is returned (Line 14).

IV. Forward Secure DSSE: CLOSE-F

In this section, we instantiate our CLOSE framework and propose a DSSE scheme which achieves constant client storage cost, forward security, and nearly optimal computational efficiency whether in theory or practice, called CLOSE-F.

A. Overview

CLOSE-F consists of three protocols: Setup, Update, and Search, as described in Section II-B. In Setup, the client

Authorized licensed use limited to: Wuhan University. Downloaded on August 01,2023 at 12:04:40 UTC from IEEE Xplore. Restrictions apply.

Algorithm 3 Setup$(\lambda, CLen; \perp) \rightarrow (\sigma; EDB, CLen)$

Client:
1: $k \xleftarrow{\$} \{0,1\}^\lambda$
2: $ctr \leftarrow CLen$
3: $\sigma \leftarrow (k, ctr)$
4: $EDB \leftarrow \emptyset$
5: Send $EDB, CLen$ to the Server.

initializes the DSSE system with some initial values and structures used in other protocols. In Update, keyword/document pairs are encoded by the client for further search. More precisely, for each keyword in the update, a DIC is constructed based on the current global counter and stored on the server. In Search, the client generates a search token based on the keyword and current global counter and sends the search token to the server. Then, the server uses the search token to produce all required nodes on LoKIC and searches for the documents which are stored in DIC. The three protocols of CLOSE-F is sketched in Fig. 3.

B. Construction of CLOSE-F

Let $F : \{0,1\}^\lambda \times \{0,1\}^* \rightarrow \{0,1\}^\lambda$ be a pseudo-random function. The three protocols of our scheme CLOSE-F are described as follows.

1) The Setup Protocol: The formal description of Setup is shown in Algorithm 3. In our scheme, the length of LoKIC is set to a constant $CLen$ in advance, therefore we use Setup$(\lambda, CLen; \perp) \rightarrow (\sigma; EDB, CLen)$ in CLOSE-F instead of Setup$(\lambda; \perp) \rightarrow (\sigma; EDB)$.

Since the state consists of the secret key k (for generating the keyword token) and the global counter ctr (for generating the search token), the storage cost on the client side is constant and extremely small. The client creates an empty EDB for the fish-bone chain and sends it to the server.

2) The Update Protocol: Algorithm 4 describes Update formally. In our forward secure scheme, we only consider the addition operation, and therefore we use Add$(\sigma, DOC; EDB)$ to denote Update$(\sigma, DOC, add; EDB)$ for simplicity.

To add a document set DOC into EDB, we perform the operation on every keyword/document pair. The client first takes a document doc out (Line 4-5), and then processes each keyword of that document (Line 7-16). If w is not in the dictionary KW, the client computes the search token $st_w = H^{ctr}(kt_w)$ for that keyword, where $kt_w = F(k, w)$ is the keyword token (Line 12). Then, the client inserts (w, st_w) into KW and initializes an empty DIC dic_w (Line 13-14). The chain update algorithm CUpdate is invoked to encode the document identifier ind into the DIC dic_w (Line 16). Finally, the client updates the state and sends the addition dictionary ADD to the server (Line 17-18) who adds ADD to the encrypted database EDB (Line 19).

The storage, computation, and communication complexities of the Add protocol on the server side are related to the number of keyword/document pairs in the document set DOC. All these complexities are optimal as in [10], [18]. The constant $CLen$ means that we can only take the add operation for $CLen$ times. After that, we have to rebuild the fish-bone chain with a new secret key and a new counter. To this end, the client first obtains the encrypted database EDB from the server and recovers the document set DOC from EDB through searching all the keywords locally. Then, the client invokes Algorithm 4 with $\sigma = (k^*, CLen)$, where k^* is the new secret key. This rebuilding process will not introduce additional leakage and the amortized cost is $O((CLen \times m + N)/N)$, where m is the number of distinct keywords and N is the number of keyword/document pairs in the database.

Algorithm 4 Add$(\sigma, DOC; EDB) \rightarrow (\sigma'; EDB')$

Client:
1: Parse σ as (k, ctr)
2: $ADD \leftarrow \emptyset$; $KW \leftarrow \emptyset$
3: **while** $|DOC| \neq 0$ **do**
4: $\quad doc \xleftarrow{\$} DOC$
5: $\quad DOC \leftarrow DOC \setminus \{doc\}$
6: $\quad$ Parse doc as (ind, W_{ind})
7: $\quad$ **while** $|W_{ind}| \neq 0$ **do**
8: $\quad\quad w \xleftarrow{\$} W_{ind}$
9: $\quad\quad W_{ind} \leftarrow W_{ind} \setminus \{w\}$
10: $\quad\quad st_w \leftarrow lookup(KW, w)$
11: $\quad\quad$ **if** $st_w = \perp$ **then**
12: $\quad\quad\quad kt_w \leftarrow F(k, w)$; $st_w \leftarrow H^{ctr}(kt_w)$
13: $\quad\quad\quad KW \leftarrow KW \cup (w, st_w)$
14: $\quad\quad\quad dic_w \leftarrow \emptyset$; $ADD \leftarrow ADD \cup dic_w$
15: $\quad\quad dic'_w \leftarrow$ CUpdate(st_w, dic_w, ind)
16: $\quad\quad ADD \leftarrow ADD \cup dic'_w$
17: $\sigma' \leftarrow (k, ctr - 1)$
18: Send ADD to the Server

Server:
19: $EBD' \leftarrow EDB \cup ADD$

3) The Search Protocol: Search is described formally in Algorithm 5. To find the documents containing a keyword w in the encrypted database EDB, the client computes the search token st_w by hashing the keyword token $kt_w = F(k, w)$ for ctr times (Line 2). Then, the client sends st_w together with the global counter ctr to the server (Line 4), who then invokes the chain search algorithm CSearch to obtain the document set $DB(w)$ (Line 5).

The computation complexity of the Search operation is $O(CLen + |DB(w)|)$, which is a little greater than the optimal one $O(|DB(w)|)$. The communication complexity is the optimal $O(|DB(w)|)$.

C. Security Analysis

In this section, we analysis the security of CLOSE-F. Intuitively, since the hash function is one-way, the server cannot decrypt the identifiers stored in a DIC unless the client produces preceding search token for that DIC. Therefore, CLOSE-F achieves forward security, which is formalized in the following theorem.

Theorem 1: Let F be a pseudo-random function, and H be a cryptographic hash function. CLOSE-F

Authorized licensed use limited to: Wuhan University. Downloaded on August 01,2023 at 12:04:40 UTC from IEEE Xplore. Restrictions apply.

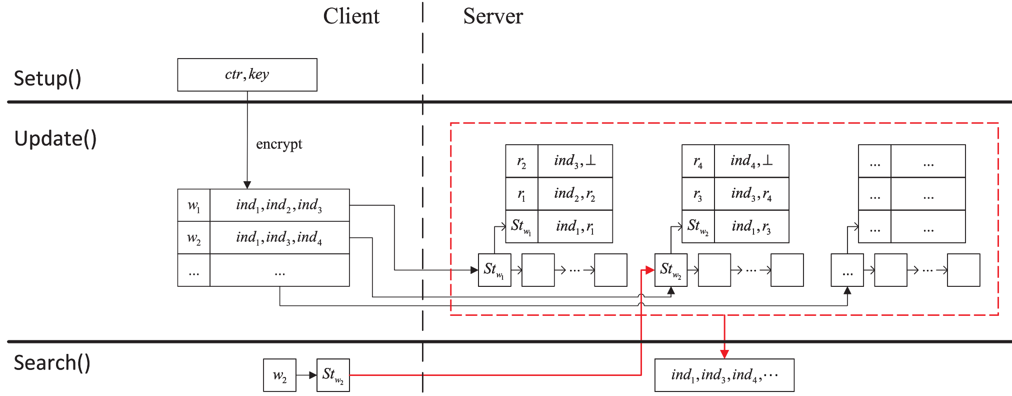


Fig. 3. The framework of CLOSE-F. This figure represents the logical view of DICs stored on the server. From the perspective of the server, all DICs are mixed up and it is even impossible to determine whether two nodes belong to the same DIC.

Algorithm 5 $\mathsf{Search}((\sigma, w); EDB) \rightarrow ((\sigma', DB(w)); EDB')$

Client:
1: Parse σ as (k, ctr)
2: $kt_w \leftarrow F(k, w)$; $st_w \leftarrow H^{ctr}(kt_w)$
3: $\sigma' \leftarrow \sigma$
4: Send ctr, st_w to the Server

Server:
5: $DB(w) \leftarrow \mathsf{CSearch}(EDB, CLen, ctr, st_w)$
6: $EDB' \leftarrow EDB$
7: Send $DB(w)$ to the Client

is $\mathcal{L}$-adaptive-secure in the random oracle model, where the collection of leakage functions $\mathcal{L}^{CLOSE-F} = (\mathcal{L}^{CLOSE-F}_{\mathsf{Setup}}, \mathcal{L}^{CLOSE-F}_{\mathsf{Search}}, \mathcal{L}^{CLOSE-F}_{\mathsf{Update}})$ *is defined as follows.*

$$\mathcal{L}^{CLOSE-F}_{\mathsf{Setup}}(CLen, \lambda) = \perp,$$
$$\mathcal{L}^{CLOSE-F}_{\mathsf{Search}}(w) = (\mathsf{sp}(w), \mathsf{HistDB}(w)),$$
$$\mathcal{L}^{CLOSE-F}_{\mathsf{Update}}(DOC, op) = (\sum_{w \in W} |DB(w)|, op).$$

Proof: Our proof uses the hybrid argument which consists of a series of games. The first game $\mathbf{Game_0}$ is exactly the same with the real world SSE game, while the last game $\mathbf{Game_3}$ is exactly the same with the ideal world SSE game.

$\mathbf{Game_0}$. This game is the real world SSE security game $Real$. Therefore, we have

$$\Pr[Real^{CLOSE-F}_{\mathcal{A}}(\lambda) = 1] = \Pr[Game_0 = 1].$$

$\mathbf{Game_1}$. In this game, we maintain a table $\mathtt{Key}$ to perform pseudo-random function, which is indexed by the keyword w. For each keyword w, $\mathtt{Key}$ records a random string that is binded to the keyword. That is, the system chooses a random keyword token kt_w rather than computing the keyword token from the pseudo-random function F. Precisely, the system picks a new random string if it is confronted to a new w, and stores the sting in a table $\mathtt{Key}$ so that it can be reused when w is queried again. If an adversary can distinguish $Game_1$ from $Game_0$, we can distinguish F from a real random function. That is, we can build an efficient adversary $\mathcal{B}_1$ such that:

$$\Pr[Game_0 = 1] - \Pr[Game_1 = 1] \leq Adv^{prf}_{F,\mathcal{B}_1}(\lambda).$$

$\mathbf{Game_2}$. In this game, we maintain three tables $\mathtt{H}$, $\mathtt{H_1}$, and $\mathtt{H_2}$ to answer the random oracle query, where $\mathtt{H}$ records the response to $H^{ctr}(kt_w)$, and $\mathtt{H_1}$ and $\mathtt{H_2}$ record the response to $H(st_w\|0)$ and $H(st_w\|1)$, respectively. Because we only consider the addition operation, the Update leakage function is also defined as $\mathcal{L}_{\mathsf{Add}}(DOC) = \sum_{w \in W} |DB(w)|$ which only leaks the number of keyword/document pairs. The search token st_w in the Add protocol is generated as random string instead of calling hash function H for ctr times. Also, the hash function $h_1 = H(st_w\|0)$ and $h_2 = H(st_w\|1)$ in the progress of generating tokens are replaced by random strings. Then the random oracle H is programmed to ensure that $H^{ctr}(kt_w) = st_w$, $H(st_w\|0) = h_1$, and $H(st_w\|1) = h_2$ in the search protocol. Every time to call hash function, we keep the track of transcripts via the tables $\mathtt{H}$, $\mathtt{H_1}$, and $\mathtt{H_2}$ respectively. If an adversary can distinguish $Game_2$ from $Game_1$, we will be able to distinguish H from real random function. Formally, we can build an efficient adversary $\mathcal{B}_2$ such that:

$$\Pr[Game_1 = 1] - \Pr[Game_2 = 1] \leq Adv^{hash}_{H,\mathcal{B}_2}(\lambda).$$

$\mathbf{Game_3}$. In this game, we maintain a table $\mathtt{Update}$ for generating search token, which is indexed by the keyword. In the search protocol, $Game_3$ generates the search token st_w by hashing ctr times instead of a used one. We use the immediate table $\mathtt{Update}$ which maps the keyword to the update counter instead of mapping the keyword to the value picked from DIC. Hence, we have

$$\Pr[Game_2 = 1] = \Pr[Game_3 = 1].$$

Simulator. $Game_3$ and $Ideal_{\mathcal{S},\mathcal{L}}$ are identical except that $Simulator$ in $Ideal_{\mathcal{S},\mathcal{L}}$ uses the counter $\bar{w} = \min sp(w)$ uniquely using the leakage function instead of the key w itself. Hence, we have

$$\Pr[Game_3 = 1] = \Pr[Ideal^{CLOSE-F}_{\mathcal{A}}(\lambda) = 1].$$

Conclusion. Combining all games, by stating hash function H is a one-way function and F is a pseudo-random function, there exists two adversaries $\mathcal{B}_1$, $\mathcal{B}_2$ such that

$$\Pr[Real^{CLOSE-F}_{\mathcal{A}}(\lambda) = 1] - \Pr[Ideal^{CLOSE-F}_{\mathcal{A}}(\lambda) = 1] \leq Adv^{prf}_{F,\mathcal{B}_1=1}(\lambda) + Adv^{hash}_{H,\mathcal{B}_2}(\lambda).$$

Then, the theorem is proved. □

Authorized licensed use limited to: Wuhan University. Downloaded on August 01,2023 at 12:04:40 UTC from IEEE Xplore. Restrictions apply.

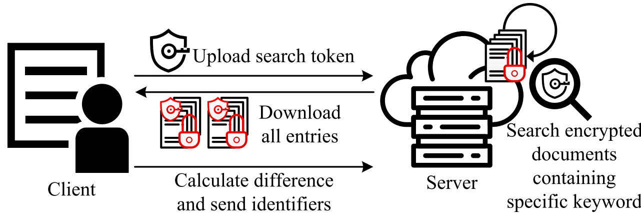


Fig. 4. An illustration of CLOSE-FB.

Note that the forward security preserves even if the adversary has additional background knowledge, such as the most frequent keyword that has been searched before. That is because the update is indistinguishable from a random string of the same length.

V. Forward and Backward Secure DSSE: CLOSE-FB

To achieve forward and backward security simultaneously, we propose another DSSE scheme in this section, called CLOSE-FB, which is also an instantiation of our CLOSE framework. Though CLOSE-FB achieves backward security compared with CLOSE-F, it needs two round of communication in the search protocol which will increase the computation and communication costs on the client side.

A. Overview

As the same with CLOSE-F, CLOSE-FB also consists of three protocols: Setup, Update, and Search. One difference between CLOSE-F and CLOSE-FB is that instead of ind, we encode the $(ind||op)$ in each node of the DIC to support deletion and backward security. Another difference is that CLOSE-FB needs to take two rounds to perform the search protocol. As shown in Fig. 4, when the client sends a search token to the server, the server searches the encrypted documents for the specific keyword and returns all entries which contains identifiers of added and deleted documents to the client. Finally, the client calculates the difference of two parts and sends the identifiers of documents to the server for the final search results.

B. Construction of CLOSE-FB

Let (Enc, Dec) be the encryption and decryption algorithms of some symmetric encryption scheme (e.g., AES). The three protocols of our scheme CLOSE-FB are described as follows.

1) The Setup Protocol: The formal description of Setup for CLOSE-B is shown in Algorithm 6. Different from CLOSE-F, we need two secret keys in CLOSE-FB. One is used to generate keyword tokens, the other one is used to encrypt $(ind\|op)$. As a result, we generate k_1 and k_2 in the setup protocol and add both two secret keys into the client state (Line 3). Finally, the client initiates EDB and sends to the server (Line 4-5).

Since the state only consists of two secret keys k_1, k_2 and the global counter ctr, the local storage cost is still constant and extremely small as the same with CLOSE-F.

Algorithm 6 Setup$(CLen, \lambda; \perp) \rightarrow (\sigma; EDB)$

Client:
1: $k_1 \xleftarrow{\$} \{0,1\}^\lambda$; $k_2 \xleftarrow{\$} \{0,1\}^\lambda$
2: $ctr \leftarrow CLen$
3: $\sigma \leftarrow (k_1, k_2, ctr)$
4: $EDB \leftarrow \emptyset$
5: Send EDB to the Server.

Algorithm 7 Update$(\sigma, DOC, op; EDB) \rightarrow (\sigma'; EDB')$

Client:
1: Parse σ as (k_1, k_2, ctr)
2: $UPDATE \leftarrow \emptyset$; $KW \leftarrow \emptyset$
3: **while** $|DOC| \neq 0$ **do**
4: $\quad doc \xleftarrow{\$} DOC$
5: $\quad DOC \leftarrow DOC \setminus \{doc\}$
6: $\quad$ Parse doc as (ind, W_{ind})
7: $\quad$ **while** $|W_{ind}| \neq 0$ **do**
8: $\quad\quad w \xleftarrow{\$} W_{ind}$
9: $\quad\quad W_{ind} \leftarrow W_{ind} \setminus \{w\}$
10: $\quad\quad st_w \leftarrow lookup(KW, w)$
11: $\quad\quad$ **if** $st_w = \perp$ **then**
12: $\quad\quad\quad kt_w \leftarrow F(k_1, w)$; $st_w \leftarrow H^{ctr}(kt_w)$
13: $\quad\quad\quad KW \leftarrow KW \cup (w, st_w)$
14: $\quad\quad\quad dic_w \leftarrow \emptyset$; $UPDATE \leftarrow UPDATE \cup dic_w$
15: $\quad\quad indop \leftarrow Enc(k_2, ind\|op)$
16: $\quad\quad dic'_w \leftarrow$ CUpdate$(st_w, dic_w, indop)$
17: $\quad\quad UPDATE \leftarrow UPDATE \cup dic'_w$
18: $\sigma' \leftarrow (k_1, k_2, ctr - 1)$
19: Send $UPDATE$ to the Server.
Server:
1: $EBD' \leftarrow EDB \cup UPDATE$

2) The Update Protocol: Details of Update are described in Algorithm 7. To support deletion operation and achieve backward security, we encrypt the entry $(ind\|op)$ with secret key k_2 and store the encrypted form of the entry in the node on DIC (Line 15). The rest of this algorithm is similar to Algotihm 4.

With the encrypted entry $(ind\|op)$, we can recognize what operation is taken to the documents. Thus, we can not only add documents to EDB, but also delete documents from EDB. Encrypted forms of entry prevent adversary from eavesdropping the information of update operation.

3) The Search Protocol: We describe Search of CLOSE-FB in Algorithm 8 specifically. The difference between this algorithm and Algorithm 5 is that the server in this algorithm only obtains encrypted entry $(ind\|op)$ from the chain search algorithm CSearch. Therefore, the server sends all encrypted entries to the client for the exact identifiers of corresponding documents. The client decrypts every encrypted entry with secret key k_2. Then, it add documents identifiers whose op is equal to add to $DB(w)$ and delete documents identifiers whose op is equal to del. Finally, the client sends all identifiers to the server to obtain the specific documents.

Authorized licensed use limited to: Wuhan University. Downloaded on August 01,2023 at 12:04:40 UTC from IEEE Xplore. Restrictions apply.

Algorithm 8 Search$((\sigma, w); EDB) \rightarrow ((\sigma', DB(w)); EDB')$

Client:
1: Parse σ as (k_1, k_2, ctr)
2: $kt_w \leftarrow F(k_1, w)$; $st_w \leftarrow H^{ctr}(kt_w)$
3: $\sigma' \leftarrow \sigma$
4: Send ctr, st_w to the Server.

Server:
1: $res \leftarrow$ CSearch$(EDB, CLen, ctr, st_w)$
2: $EDB' \leftarrow EDB$
3: Send res to the Client.

Client:
1: $DB(w) \leftarrow \emptyset$
2: **while** $|res| \neq \perp$ **do**
3: $\quad indop \xleftarrow{\$} RES$
4: $\quad res \leftarrow res \setminus \{indop\}$
5: $\quad (ind \| op) \leftarrow Dec(k_2, indop)$
6: $\quad$ **if** $op == add$ **then**
7: $\quad\quad DB(w) \leftarrow DB(w) \cup \{ind\}$
8: $\quad$ **else**
9: $\quad\quad DB(w) \leftarrow DB(w) \setminus \{ind\}$

C. Security Analysis

CLOSE-FB achieves both forward security (Definition 2) and backward security (Definition 3). The proof is similar to the one in Section IV-C. The only difference between CLOSE-F and CLOSE-FB is that the server can directly obtain ind when it receives the search token in CLOSE-F, while in CLOSE-FB the server can only obtain the encrypted form of ind. Therefore, the simulator needs to replace ind with random strings in the game. Roughly speaking, every time we update EDB, we need to generate new search tokens for new entries. Therefore, there is no doubt that entries that server observe during the update are indistinguishable from random. In addition, the server does not know the type of operation. The only thing the update protocol leaks is the number of entries. For backward security, the server has the knowledge of the number of entries related to the keyword w and the time every update operation for keyword w took place. The search protocol reveals nothing more to the server. Particularly, the server can not figure out which deletion deletes which addition.

VI. Evaluation

In this section, we evaluate our schemes (i.e., CLOSE-F and CLOSE-FB) and compare them with related schemes. All schemes are implemented in Python 3 with the Python Cryptography Toolkit (pycrypto). The pseudo-random function F was instantiated using HMAC. We ran our experiments on a server with two Intel Xeon E5-2630 v3 CPUs and 128GB of RAM that running on Linux Ubuntu 14.04.5.

A. Dataset

We adopted the well-known Enron email dataset to evaluate our experiments, which is 1.32 GB when decompressed [24]. We exploited RAKE, a python implementation of the rapid automatic keyword extraction, to extract keywords from the Enron email dataset. After discarding some files in BASE64 encoding, we got 517,080 plain-text files. From those files, we extracted 390,423 keywords, and the total number of keyword/document pairs was 22,900,317. To analyze the impact of database size, we also derived three databases from the Enron email dataset. The detailed information of all databases used in our experiments is shown in Table III.

TABLE III
Databases Sizes

	n	m	N
DB1	4,000	15,879	127,723
DB2	20,000	58,310	874,604
DB3	100,000	168,793	5,493,993
Enron	517,080	390,423	22,900,317

As defined in II-B, n is the number of documents, m is the number of keywords, and N is the number of keyword/document pairs.

TABLE IV
Storage Cost on the Client Side

	DB1	DB2	DB3	Enron
Sophos	7,830KB	30,393KB	92,221KB	209,564KB
KKL+	5,177KB	19,063KB	-	-
Mitra	236KB	933KB	2800KB	6898KB
SD_a	1KB	1KB	1KB	1KB
SD_d	1KB	1KB	1KB	1KB
CLOSE-F	2KB	2KB	2KB	2KB
CLOSE-FB	3KB	3KB	3KB	3KB

B. Evaluation of Client Storage Cost

The client storage comparison of CLOSE-F and CLOSE-FB with previous schemes (i.e., Sophos [10], KKL+ [18], Mitra [22], SD_a [14], and SD_d [14]) is shown in Table IV. Note that, running KKL+ on DB3 and Enron needs more than a week (see Section VI-C), therefore, we omit these two experiments.

From the table, we can conclude that the storage costs of Sophos, KKL+, and Mitra increase with the size of database, while the storage costs of SD_a, SD_d, CLOSE-F, and CLOSE-FB are constant. Precisely, the storage cost on the client side grows with the number of keywords in the first three schemes. That is because those schemes employed an inverted index to keep the state for every keyword, while our schemes only store a secret key and a global counter. In SD_a and SD_d, the client can only possess a single master key, at the cost of increasing the computation and communication costs in the update protocols.

C. Evaluation of Computation Cost

Now, we discuss the computation costs of CLOSE-F, CLOSE-FB, Sophos [10], KKL+ [18], Mitra [22], SD_a [14], and SD_d [14].

Table V shows the update time of all schemes on four different databases. One can find that the update time of

Authorized licensed use limited to: Wuhan University. Downloaded on August 01,2023 at 12:04:40 UTC from IEEE Xplore. Restrictions apply.

TABLE V
COMPUTATION TIME IN THE UPDATE PROTOCOL

	DB1	DB2	DB3	Enron	$CLen$
Sophos	0.44h	3.28h	21.24h	91.86h	-
KKL+	3.78h	101.36h	-	-	-
Mitra	7.38s	50.02s	5.35min	21.63min	-
SD$_a$	58.98s	7.81min	54.03min	3.98h	-
SD$_d$	2.14min	16.42min	1.99h	8.84h	-
CLOSE-F	18.32s	1.38min	5.10min	16.78min	200
	41.19s	2.68min	9.19min	25.68min	500
	1.28min	5.02min	16.20min	41.05min	1000
	2.59min	9.48min	29.35min	1.20h	2000
CLOSE-FB	18.29s	1.36min	5.24min	18.07min	200
	39.63s	2.65min	9.49min	25.60min	500
	1.26min	5.15min	16.15min	42.57min	1000
	2.50min	9.01min	29.81min	1.15h	2000

CLOSE-F, CLOSE-FB, Mitra, SD$_a$, and SD$_d$ is much better than the other two schemes, especially KKL+. Sophos needs to carry out many exponentiation to generate search tokens and update tokens in the update process, and therefore is inefficient in practice. In KKL+, the state of inverted index and dual dictionary needs to be updated frequently, which makes it even more inefficient than Sophos. CLOSE-F and CLOSE-FB only compute hash functions, and Mitra only uses hash functions and exclusive OR to update the whole database. Thus, these three schemes are efficient in practice. The computation costs of SD$_a$ and SD$_d$ are higher than CLOSE-F, CLOSE-FB, and Mitra due to their frequent rebuilding process. Moreover, the rebuilding process also introduces a greater communication overhead in the update protocol. We also investigated the impact of the length of LoKIC on efficiency in our scheme. As shown in Table V, the update time grows with $CLen$ since the length of LoKIC determines the times of hash function. Even $CLen$ is 2000, the computation cost of our scheme is acceptable in practice.

Then, we evaluated the computation cost in the search protocol. Fig. 5 shows the performance of four schemes on the same database. Since Sophos and CLOSE-F only support addition operation, the number of matching entries in the picture denotes the number of matching documents for these two schemes. While Mitra and CLOSE-FB support addition and deletion operations for the same time, the search time for these two schemes are associated with the number of matching entries. It is obvious that search time of the proposed two schemes in this article grows slowly with the number of matching entries, which stands out in all schemes. Since the computation cost of KKL+ is huge (1.33s when the number of matching documents is 1), we omit its experimental results in all the rest figures.

Fig. 6 describes the effects of the length of LoKIC. With the growth of number of matching entries, the search time of CLOSE-F and CLOSE-FB increases slowly. The increase rate of search time with different $CLen$ is similar. The results indicate that the search time is related to the length of LoKIC, i.e., growing with $CLen$.

Fig. 7 depicts the search performance on four databases for CLOSE-F. The results indicate that the search time is

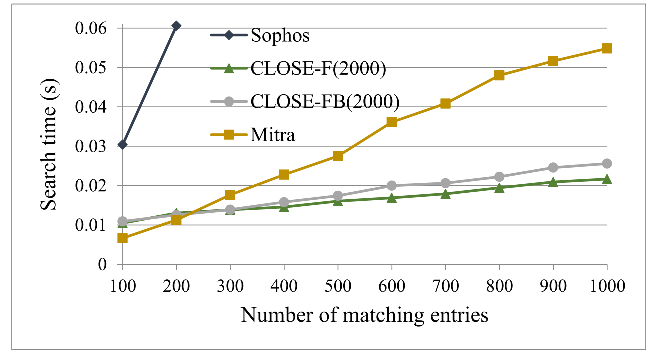
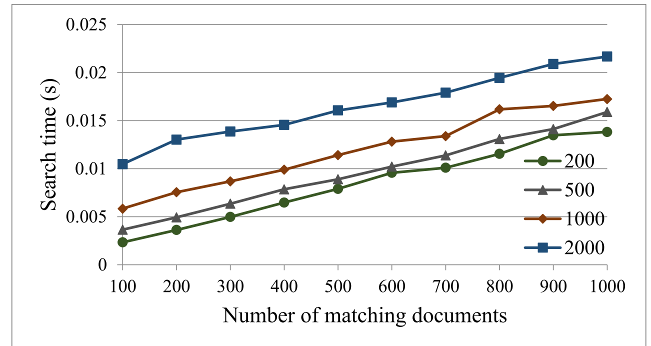
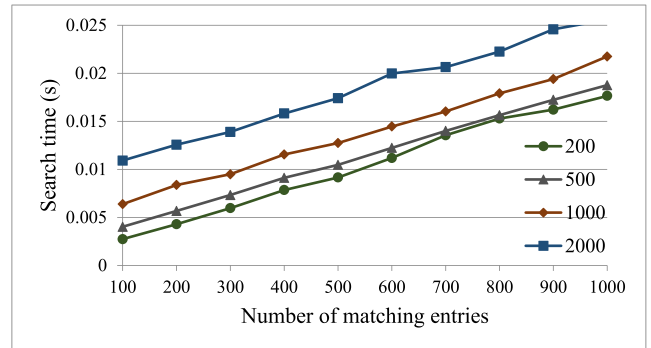


Fig. 5. Comparison of different schemes in the search process.



(a) CLOSE-F.



(b) CLOSE-FB.

Fig. 6. Comparison of our schemes with different length of LoKIC (i.e., $CLen$) in the search process.

irrelevant to the database size. Since the number of matching documents grows when the database gets larger, CLOSE-F is more suitable for large databases. As the same, the search time of CLOSE-FB is irrelevant to the database size too.

D. Summary of Experiments

CLOSE-F, CLOSE-FB, Sophos, KKL+, and Mitra are all forward secure SSE schemes. Among these five schemes, CLOSE-FB and Mitra achieve backward security. Although KKL+ achieves optimal computation complexity in theoretical analysis (see Table I), the performance in practice is extremely poor. This fact also prompted us to design an efficient SSE scheme in practice rather than in theory. Sophos is also inefficient in practice since it employed public-key primitives.

Authorized licensed use limited to: Wuhan University. Downloaded on August 01,2023 at 12:04:40 UTC from IEEE Xplore. Restrictions apply.

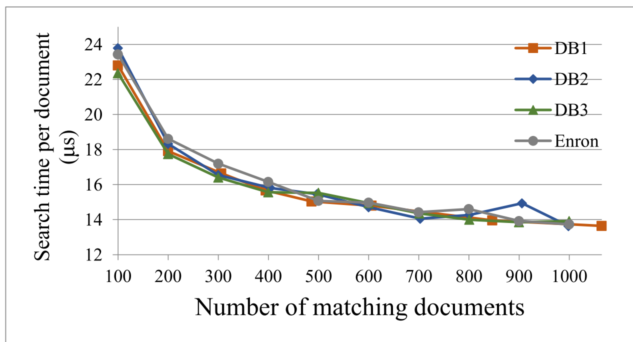


Fig. 7. Comparison of CLOSE-F in different database in the search process.

The performance of Mitra is similar to our scheme. However, Mitra performs a little worse than CLOSE-FB since there is one more operation exclusive OR in the search protocol. To conclude, CLOSE-F is the first efficient forward secure DSSE scheme in practice, and CLOSE-FB is the first efficient DSSE scheme with both forward and backward security.

VII. Related Work

Searchable Symmetric Encryption (SSE) was first introduced by Song *et al.*, and a construction with linear search time was also proposed [1]. Curtmola *et al.* then gave the first construction with sub-linear search time by employing an *inverted index*, which maintains a list of document identifiers per keyword [2]. Subsequent researches also focused on the *static* case, where the encrypted database is no longer changed after it is stored on the cloud server [3], [4], [25], [26].

To support database update, researchers then introduced *dynamic SSE* (DSSE), which allows users to add and/or delete keyword/document pairs in the database [16], [27], [28]. Many researchers focused on the improvement of computation complexity in the search protocol. The first DSSE scheme with sub-linear search time was proposed by Kamara *et al.* [7], but their solution reveals the hashes of keywords contained in the updated documents. Kamara and Papamanthou later fixed this problem by increasing the space complexity [8]. Nevertheless, all of those solutions suffer from more sophisticated attacks, such as the leakage-abuse attack [29] and the file-injection attack [15]. These attacks make researchers realize that *forward security*, which was formally introduced in [17], is an essential property for DSSE.

Although Stefanov *et al.* proposed a forward secure DSSE scheme in [17], their solution has poor computation complexity since it rebuilds the level of data structure in the update protocol. Some schemes achieve forward security via using complex cryptographic technology (e.g., oblivious RAM [30]), however, suffer from heavy computation costs [9], [31]. Schemes with both optimal computation complexity in theory and forward security also have more or less drawbacks [10], [11]. Sophos generates search tokens with inverted index and sends to the server for further search without leaking the real keywords themselves [10]. Sophos makes use of trapdoor permutation to create connection between the search tokens and update tokens, which provides better theoretical computation complexity than [31]. Nevertheless, Sophos is inefficient in practice since the trapdoor permutation is based on public-key primitives.

Backward security was informally mentioned in [17]. Then Bost *et al.* gave the formal definition of backward security and constructions which achieve this property [11]. The scheme $\text{Diana}_{\text{del}}$ in [11] suffers from the limitation of keyword/document pairs, and Janus suffers from the inefficiency of puncturable encryption. After that, Chamani *et al.* [22] proposed three schemes. Mitra achieves both optimal computational and communicational complexity, while needs two round communication. The left two schemes sacrifice the cost of computation and communication for backward security. Sun *et al.* [23] proposed symmetric puncturable encryption which improves the puncturable encryption in [11].

VIII. Conclusion

In this article, we focused on DSSE in practice and proposed the first efficient DSSE framework with constant client storage cost, named CLOSE. To achieve the computational efficiency and security, we designed a novel two-level structure, called fish-bone chain, that consists of LoKIC and DIC. We instantiate two schemes, CLOSE-F and CLOSE-FB, under the CLOSE framework. Then, we implemented CLOSE-F, CLOSE-FB and three previous schemes to evaluate the performance. The experimental results showed that our schemes perform better than the best existing schemes.

Acknowledgment

The authors thank the anonymous reviewers for their comments.

References

[1] D. Xiaoding Song, D. Wagner, and A. Perrig, "Practical techniques for searches on encrypted data," in *Proc. IEEE Symp. Secur. Privacy. S&P*, May 2000, pp. 44–55.

[2] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky, "Searchable symmetric encryption: Improved definitions and efficient constructions," in *Proc. 13th ACM Conf. Comput. Commun. Secur. CCS*, 2006, pp. 79–88.

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

[4] D. Cash, S. Jarecki, C. S. Jutla, H. Krawczyk, M. Rosu, and M. Steiner, "Highly-scalable searchable symmetric encryption with support for Boolean queries," in *Proc. CRYPTO*, 2013, pp. 353–373.

[5] D. Cash and S. Tessaro, "The locality of searchable symmetric encryption," in *Proc. EUROCRYPT*, 2014, pp. 351–368.

[6] G. Asharov, M. Naor, G. Segev, and I. Shahaf, "Searchable symmetric encryption: Optimal locality in linear space via two-dimensional balanced allocations," in *Proc. 48th Annu. ACM SIGACT Symp. Theory Comput. STOC*, 2016, pp. 1101–1114.

[7] S. Kamara, C. Papamanthou, and T. Roeder, "Dynamic searchable symmetric encryption," in *Proc. ACM Conf. Comput. Commun. Secur. - CCS*, 2012, pp. 965–976.

[8] S. Kamara and C. Papamanthou, "Parallel and dynamic searchable symmetric encryption," in *Proc. FC*, 2013, pp. 258–274.

[9] S. Garg, P. Mohassel, and C. Papamanthou, "TWORAM: Efficient oblivious RAM in two rounds with applications to searchable encryption," in *Proc. CRYPTO*, 2016, pp. 563–592.

[10] R. Bost, "$\sum o\varphi o\varsigma$: Forward secure searchable encryption," in *Proc. CCS*, 2016, pp. 1143–1154.

[11] R. Bost, B. Minaud, and O. Ohrimenko, "Forward and backward private searchable encryption from constrained cryptographic primitives," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2017, pp. 1465–1482.

Authorized licensed use limited to: Wuhan University. Downloaded on August 01,2023 at 12:04:40 UTC from IEEE Xplore. Restrictions apply.

[12] R. W. Lai and S. S. Chow, "Forward-secure searchable encryption on labeled bipartite graphs," in *Proc. ACNS*, 2017, pp. 478–497.
[13] R. Bost and P.-A. Fouque, "Security-efficiency tradeoffs in searchable encryption—Lower bounds and optimal constructions," in *Proc. PoPETS*, 2019, pp. 132–151.
[14] I. Demertzis, J. G. Chamani, D. Papadopoulos, and C. Papamanthou, "Dynamic searchable encryption with small client storage," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2020, p. 1227.
[15] 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. USENIX Secur.*, 2016, pp. 707–720.
[16] Y. Chang and M. Mitzenmacher, "Privacy preserving keyword searches on remote encrypted data," in *Proc. ACNS*, 2005, pp. 442–455.
[17] E. Stefanov, C. Papamanthou, and E. Shi, "Practical dynamic searchable encryption with small leakage," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2014, pp. 72–75.
[18] 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. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2017, pp. 1449–1463.
[19] J. Li *et al.*, "Searchable symmetric encryption with forward search privacy," *IEEE Trans. Depend. Sec. Comput.*, early access, Jan. 22, 2019, doi: 10.1109/TDSC.2019.2894411.
[20] X. Song, C. Dong, D. Yuan, Q. Xu, and M. Zhao, "Forward private searchable symmetric encryption with optimized I/O efficiency," *IEEE Trans. Depend. Sec. Comput.*, vol. 17, no. 5, pp. 912–927, Sep. 2020.
[21] M. Etemad, A. Küpçü, C. Papamanthou, and D. Evans, "Efficient dynamic searchable encryption with forward privacy," *Proc. Privacy Enhancing Technol.*, vol. 2018, no. 1, pp. 5–20, Jan. 2018.
[22] J. Ghareh Chamani, D. Papadopoulos, C. Papamanthou, and R. Jalili, "New constructions for forward and backward private symmetric searchable encryption," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Jan. 2018, pp. 1038–1055.
[23] S.-F. Sun *et al.*, "Practical backward-secure searchable encryption from symmetric puncturable encryption," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Jan. 2018, pp. 763–780.
[24] *Enron Email Dataset*. Accessed: May 8, 2015. [Online]. Available: https://www.cs.cmu.edu/~./enron/
[25] B. Wang, W. Song, W. Lou, and Y. T. Hou, "Inverted index based multi-keyword public-key searchable encryption with strong privacy guarantee," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, Apr. 2015, pp. 2092–2100.
[26] J. Chen *et al.*, "EliMFS: Achieving efficient, leakage-resilient, and multi-keyword fuzzy search on encrypted cloud data," *IEEE Trans. Services Comput.*, early access, Oct. 23, 2019, doi: 10.1109/TSC.2017.2765323.
[27] E. Goh, "Secure indexes," IACR Cryptol. ePrint Arch., NV, USA, Tech. Rep. 216, 2003.
[28] S. Hu, C. Cai, Q. Wang, C. Wang, X. Luo, and K. Ren, "Searching an encrypted cloud meets blockchain: A decentralized, reliable and fair realization," in *Proc. IEEE Infocom Conf. Comput. Commun.*, Apr. 2018, pp. 792–800.
[29] D. Cash, P. Grubbs, J. Perry, and T. Ristenpart, "Leakage-abuse attacks against searchable encryption," in *Proc. 22nd ACM SIGSAC Conf. Comput. Commun. Secur. - CCS*, 2015, pp. 668–679.
[30] E. Stefanov *et al.*, "Path ORAM: An extremely simple oblivious RAM protocol," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur. - CCS*, 2013, pp. 299–310.
[31] M. Naveed, "The fallacy of composition of oblivious RAM and searchable encryption," IACR Cryptol. ePrint Arch., Tech. Rep. 668, 2015.

Kun He received the Ph.D. degree in computer science from the Computer School, Wuhan University. He is currently an Associate Professor with Wuhan University. He has published more than 20 research papers in many international journals and conferences, such as IEEE TRANSACTIONS ON DEPENDABLE AND SECURE COMPUTING (TDSC), IEEE TRANSACTIONS ON MOBILE COMPUTING (TMC), IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS (TPDS), IEEE TRANSACTIONS ON COMPUTERS (TC), USENIX Security, and INFOCOM. His research interests include cryptography, network security, mobile computing, and cloud computing.

Jing Chen (Member, IEEE) received the Ph.D. degree in computer science from the Huazhong University of Science and Technology, Wuhan. He is currently a Full Professor with the School of Cyber Science and Engineering, Wuhan University. He has published more than 120 research papers in many international journals and conferences, including IEEE TRANSACTIONS ON DEPENDABLE AND SECURE COMPUTING (TDSC), IEEE TRANSACTIONS ON INFORMATION FORENSICS AND SECURITY (TIFS), IEEE TRANSACTIONS ON MOBILE COMPUTING (TMC), IEEE TRANSACTIONS ON COMPUTERS (TC), IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS (TPDS), IEEE TRANSACTIONS ON SERVICES COMPUTING (TSC), and INFOCOM. His research interests include network security, cloud security, and mobile security.

Qinxi Zhou received the B.S. degree in information security from Northeastern University in 2016 and the M.S. degree in information security from Wuhan University in 2019. Her research interest includes searchable encryption.

Ruiying Du received the B.S., M.S., Ph.D. degrees in computer science from Wuhan University, Wuhan, China, in 1987, 1994, and 2008, respectively. She is currently a Professor with the School of Cyber Science and Engineering, Wuhan University. Her research interests include network security, wireless networks, cloud computing, and mobile computing. She has published more than 80 research papers in many international journals and conferences, such as IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, *International Journal of Parallel and Distributed Systems and Networks*, INFOCOM, SECON, TrustCom, and NSS.

Yang Xiang (Fellow, IEEE) received the Ph.D. degree in computer science from Deakin University, Australia, in 2007. He is currently the Dean of the Digital Research and Innovation Capability Platform, Swinburne University of Technology, Australia. In particular, he is leading his team developing active defense systems against large-scale distributed network attacks. He is the chief investigator of several projects in network and system security, funded by the Australian Research Council (ARC). He has published more than 200 research papers in many international journals and conferences. His research interests include network and system security, distributed systems, and networking. He has served as the program/general chair and a PC member for more than 60 international conferences in distributed systems, networking, and security. He is the Coordinator of Asia for the IEEE Computer Society Technical Committee on Distributed Processing (TCDP).

Authorized licensed use limited to: Wuhan University. Downloaded on August 01,2023 at 12:04:40 UTC from IEEE Xplore. Restrictions apply.