Improvement of UC Secure Searchable Symmetric Encryption Scheme

Shunsuke Taketani and Wakaha Ogata(✉)

Tokyo Institute of Technology, Meguro, Japan
{taketani.s.aa,ogata.w.aa}@m.titech.ac.jp

Abstract. Searchable symmetric encryption refers to a system whereby clients store encrypted documents in a server that can be searched by keywords without revealing private information. In this paper, we demonstrate that the UC-secure SSE scheme proposed by Kurosawa and Ohtaki is inefficient under certain scenarios, and we propose a modified scheme. Our scheme has reliability and privacy, where privacy is slightly weaker than the original Kurosawa-Ohtaki scheme. Therefore, our scheme offers UC-security with slightly weaker privacy. More precisely, the additional information our scheme leaks is only the size of a set of keywords. On the other hand, the index size for our scheme is much smaller than the original scheme when the set of keywords is a very sparse subset of l-bit strings for some l. The UC-secure Kurosawa-Ohtaki scheme is improved with the proposed scheme by introducing a new tag for proving "non-existence." The proposal is an example of how an SSE scheme can be effectively converted into a verifiable SSE scheme.

1 Introduction

In recent years, many storage services have become available with which clients can store documents or files on the service provider's server. By using such services, clients can access their information at any time and from anywhere and any device. If the number of stored files increases, a keyword search is desirable to find particular files. On the other hand, the client can encrypt files to avoid leaking confidential information to the service provider. Searchable symmetric encryption (SSE) enables the client to search a large number of encrypted files with encrypted keywords.

The concept for SSE was introduced by Song et al. [21], and many SSE schemes have since been proposed [2,9,11,12,14,18]. Most SSE schemes offer privacy; e.g., the server cannot learn anything about the stored files or keywords. However, such schemes do not have any mechanism to verify search results; that is, it is assumed that the server is honest and that it always follows protocol. Thus, Kurosawa and Ohtaki [18] introduced enhanced security notions, *reliability* and universally composable security (UC-security) for SSE schemes. Reliability ensures the validity of search results even if the server is malicious. Kurosawa

DOI: 10.1007/978-3-319-22425-1_9

and Ohtaki showed that an SSE scheme that has both privacy and reliability has UC-security[1], and proposed a concrete SSE scheme which has UC-security.

In this paper, we evaluate the scheme proposed by Kurosawa and Ohtaki [18], and show that it is inefficient under some scenarios. We then propose a modified scheme for which privacy is somewhat weaker than the original Kurosawa-Ohtaki scheme. Therefore, our scheme provides UC-security with slightly weaker privacy. More precisely, the additional information our scheme leaks is merely the size of a set of keywords. Yet, the index size for our scheme is much smaller than the original scheme when the set of keywords is a very sparse subset of l-bit strings for some l. The index size is reduced by eliminating dummy elements from the original index and introducing a new tag that proves that the eliminated elements do not exist in the reduced index.

Related works. Various SSE schemes have been proposed. Some support additional search functions, such as multi-keyword searching [1,13], ranked searching [7], and fuzzy searching [3,22]. Others support adding and removing documents [15–17,19].

In [10], a verifiable SSE scheme is proposed. The scheme has two modes: "privacy preferred" and "efficiency preferred." However, the former mode, which is relatively more secure in terms of privacy, requires a very large index. Moreover, no formal security proof is provided for this scheme.

[19] proposed another verifiable SSE scheme. However, their scheme does not assume that a client will query a keyword that is not contained in the set of keywords used to build the index. Assume that a client generates an (encrypted) index based on a certain keyword set $\mathcal{W}$, forgets $\mathcal{W}$, and then searches for $w \notin \mathcal{W}$. In this case, the server has no choice but to return "no document hits" without any proof. This means that the server can forge the search results by answering "no document hits"at any time.

2 Verifiable Searchable Symmetric Encryption

In this section, we define a (verifiable) SSE scheme and its security. Basically, we follow the notation used in [8,18,20].

- Let $\mathcal{D} = \{D_0, \ldots, D_{N-1}\}$ be a set of documents.
- Let $\mathcal{W}$ be a set of keywords.
- For a keyword $w \in \mathcal{W}$, let $\mathcal{D}(w)$ denote the set of documents that contain w.

We consider a system model that has two components: a client and a server. Roughly speaking, in SSE schemes, clients encrypt all documents in $\mathcal{D}$ before storing them on a server. Clients can then search through these documents using a keyword $w \in \mathcal{W}$ from the set $\mathcal{D}$, the output for which is derived as follows:

$$\mathcal{C}(w) = \{C_i \mid C_i \text{ is a ciphertext of } D_i \in \mathcal{D}(w)\}. \tag{1}$$

[1] In [20], it was shown that *strong* reliability rather than ordinary reliability is required to be US-security.

In response to a search query, the server returns $\mathcal{C}(w)$. If there is a mechanism to verify the validity of the response, the scheme is a verifiable SSE (vSSE).

Hereafter, $|X|$ denotes the bit length of X for a bit string X, and $|X|$ denotes the cardinality of X for a set X. Furthermore, "PPT" refers to the probabilistic polynomial time.

2.1 System Model

Formally, a vSSE scheme has two phases: the store phase (executed only once) and the search phase (executed a polynomial number of times). Such a scheme consists of the following six polynomial-time algorithms:[2]

$$\texttt{vSSE} = (\texttt{Gen}, \texttt{Enc}, \texttt{Trpdr}, \texttt{Search}, \texttt{Dec}, \texttt{Verify})$$

such that

- $K \leftarrow \texttt{Gen}(1^\lambda)$: a probabilistic algorithm that generates a key K, where λ is a security parameter. This algorithm is run by the client during the store phase, and K is kept secret.
- $(\mathcal{I}, \mathcal{C}) \leftarrow \texttt{Enc}(K, \mathcal{D}, \mathcal{W})$: a probabilistic encryption algorithm that outputs an encrypted index $\mathcal{I}$ and $\mathcal{C} = \{C_0, \ldots, C_{N-1}\}$, where C_i is the ciphertext for D_i. This algorithm is run by the client during the store phase, and $(\mathcal{I}, \mathcal{C})$ are sent to the server.
- $t(w) \leftarrow \texttt{Trpdr}(K, w)$: an algorithm that outputs a trapdoor $t(w)$ for a keyword w. This is run by the client during the search phase, and $t(w)$ is sent to the server.
- $(\tilde{\mathcal{C}}(w), \mathit{Proof}) \leftarrow \texttt{Search}(\mathcal{I}, \mathcal{C}, t(w))$: a deterministic search algorithm, where $\tilde{\mathcal{C}}(w)$ is the search result and *Proof* is its proof. This algorithm is run by the server during the search phase, and $(\tilde{\mathcal{C}}(w), \mathit{Proof})$ is sent to the client.
- $\texttt{accept}/\texttt{reject} \leftarrow \texttt{Verify}(K, t(w), \tilde{\mathcal{C}}(w), \mathit{Proof})$: a deterministic verification algorithm that determines the validity of $\tilde{\mathcal{C}}(w)$ based on *Proof*. This algorithm is run by the client.
- $D \leftarrow \texttt{Dec}(K, C)$: a deterministic decryption algorithm. The client uses this algorithm for all $C \in \tilde{\mathcal{C}}(w)$, when $\texttt{Verify}(K, t(w), \tilde{\mathcal{C}}(w), \mathit{Proof}) = \texttt{accept}$.

Correctness entails the following from the scheme for the set of documents $\mathcal{D}$ and a keyword $w \in \mathcal{W}$:

- $D_i = \texttt{Dec}(K, C_i)$ if $\mathcal{C} = \{C_0, \ldots, C_{N-1}\}$ is the output of $\texttt{Enc}(K, \mathcal{D}, \mathcal{W})$.
- $\texttt{Verify}(K, t(w), \tilde{\mathcal{C}}(w), \mathit{Proof}) = \texttt{accept}$, if $(\mathcal{I}, \mathcal{C})$ is outputted by $\texttt{Enc}(K, \mathcal{D}, \mathcal{W})$, $t(w)$ is outputted by $\texttt{Trpdr}(K, w)$, and $(\tilde{\mathcal{C}}(w), \mathit{Proof})$ is outputted by $\texttt{Search}(\mathcal{I}, \mathcal{C}, t(w))$.

2.2 Security Definition

We next define some security conditions that should be satisfied by a vSSE scheme.

[2] If the search result does not need to be verified, *Proof* and `Verify` can be omitted.

- Adversary $\mathbf{A}$ chooses $(\mathcal{D}, \mathcal{W})$ and sends them to challenger $\mathbf{C}$.
- $\mathbf{C}$ generates $K \leftarrow \texttt{Gen}(1^k)$ and sends $(\mathcal{I}, \mathcal{C}) \leftarrow \texttt{Enc}(K, \mathcal{D}, \mathcal{W})$ to $\mathbf{A}$.
- For $i = 0, \ldots, q-1$, do:
  1. $\mathbf{A}$ chooses a keyword $w_i \in \mathcal{W}$ and sends it to $\mathbf{C}$.
  2. $\mathbf{C}$ sends the trapdoor $t(w_i) \leftarrow \texttt{Trpdr}(K, w_i)$ back to $\mathbf{A}$.
- $\mathbf{A}$ outputs bit b.

Fig. 1. Real game $\texttt{Game}_{real}$

- Adversary $\mathbf{A}$ chooses $(\mathcal{D}, \mathcal{W})$ and sends them to challenger $\mathbf{C}$.
- $\mathbf{C}$ sends $L_1(\mathcal{D}, \mathcal{W})$ to simulator $\mathbf{S}$.
- $\mathbf{S}$ computes $(\mathcal{I}', \mathcal{C}')$ from $L_1(\mathcal{D}, \mathcal{W})$, and sends them to $\mathbf{C}$.
- $\mathbf{C}$ relays $(\mathcal{I}', \mathcal{C}')$ to $\mathbf{A}$
- For $i = 0, \ldots, q-1$, do:
  1. $\mathbf{A}$ chooses $w_i \in \mathcal{W}$ and sends it to $\mathbf{C}$.
  2. $\mathbf{C}$ sends $L_2(\mathcal{D}, \mathcal{W}, \mathbf{w}, w_i)$ to $\mathbf{S}$, where $\mathbf{w} = (w_1, \ldots, w_{i-1})$.
  3. $\mathbf{S}$ computes $t'(w_i)$ from $L_2(\mathcal{D}, \mathcal{W}, \mathbf{w}, w_i)$ and sends it to $\mathbf{C}$.
  4. $\mathbf{C}$ relays $t'(w_i)$ to $\mathbf{A}$.
- $\mathbf{A}$ outputs bit b.

Fig. 2. Simulation game $\texttt{Game}^L_{sim}$

Privacy. In a vSSE, the server should learn as little information as possible regarding $\mathcal{D}, \mathcal{W}$, and the queried keyword w. Let $L = (L_1, L_2)$ be a pair of leakage functions, such that $L_1(\mathcal{D}, \mathcal{W})$ (and respectively, $L_2(\mathcal{D}, \mathcal{W}, \mathbf{w}, w)$) denote the information the user permits the server to learn during the store phase (and respectively, the search phase). Here, $\mathbf{w} = (w_1, w_2, \ldots)$ is the list of keywords queried in past searches, and w is the keyword queried now. The client's privacy is defined by using two games: a real game $\texttt{Game}_{real}$, and a simulation game $\texttt{Game}^L_{sim}$, as shown in Figs. 1 and 2, respectively. $\texttt{Game}_{real}$ is played by a challenger $\mathbf{C}$ and an adversary $\mathbf{A}$, and $\texttt{Game}^L_{sim}$ is played by $\mathbf{C}$, $\mathbf{A}$ and a simulator $\mathbf{S}$.

Definition 1 (L-privacy). *We say that a vSSE scheme has L-privacy, if there exists a PPT simulator* $\mathbf{S}$ *such that*

$$|\Pr(\mathbf{A} \textit{ outputs } b = 1 \textit{ in } \texttt{Game}_{real}) - \Pr(\mathbf{A} \textit{ outputs } b = 1 \textit{ in } \texttt{Game}^L_{sim})| \qquad (2)$$

is negligible for any PPT adversary $\mathbf{A}$.

In most existing SSE schemes, $L_1(\mathcal{D}, \mathcal{W})$ includes $(|D_0|, \ldots, |D_{N-1}|)$ and some information about $\mathcal{W}$, such as $|\mathcal{W}|$ or the length of the keywords. On the other hand, $L_2(\mathcal{D}, \mathcal{W}, \mathbf{w}, w)$ consists of

$$\texttt{List}(w) = \{j \mid D_j \in \mathcal{D} \text{ contains } w\}$$

(Store phase)

- $\mathbf{A}_1$ chooses $(\mathcal{D}, \mathcal{W})$ and sends them to $\mathbf{C}$.
- $\mathbf{C}$ generates $K \leftarrow \texttt{Gen}(1^\lambda)$, and sends $(\mathcal{I}, \mathcal{C}) \leftarrow \texttt{Enc}(K, \mathcal{D}, \mathcal{W})$ to $\mathbf{A}_2$.

(Search phase)

- For $i = 0, \ldots, q-1$, do
  1. $\mathbf{A}_1$ chooses a keyword w_i and sends it to $\mathbf{C}$.
  2. $\mathbf{C}$ sends the trapdoor $t(w_i) \leftarrow \texttt{Trpdr}(K, w_i)$ to $\mathbf{A}_2$.
  3. $\mathbf{A}_2$ returns $(\tilde{\mathcal{C}}(w_i)^*, \mathit{Proof}_i^*)$ to $\mathbf{C}$.
  4. $\mathbf{C}$ computes
  $$\texttt{accept/reject} \leftarrow \texttt{Verify}(K, t(w_i), \tilde{\mathcal{C}}(w_i)^*, \mathit{Proof}_i^*)$$
  and returns a set $\tilde{\mathcal{D}}(w_i)^*$ of plaintexts of documents in $\tilde{\mathcal{C}}(w_i)^*$ to $\mathbf{A}_1$ if the result is `accept`, otherwise sends `reject` to $\mathbf{A}_1$.
- If there exists i, such that both $\texttt{Verify}(K, t(w_i), \tilde{\mathcal{C}}(w_i)^*, \mathit{Proof}_i^*) = \texttt{accept}$ and $(\tilde{\mathcal{C}}(w_i)^*, \mathit{Proof}_i^*) \neq (\mathcal{C}(w_i), \mathit{Proof}_i)$ hold, then $\mathbf{A}$ (strongly) wins; otherwise $\mathbf{A}$ loses.

Fig. 3. $\texttt{Game}_{reli}$

and the search pattern

$$\texttt{SPattern}((w_1, \ldots, w_{q-1}), w) = (sp_1, \ldots, sp_{q-1}), \quad sp_j = \begin{cases} 1 & \text{if } w_j = w \\ 0 & \text{if } w_j \neq w \end{cases}$$

that reveals the past queries that are the same as w.

Reliability. In an SSE scheme, a malicious server should not cheat a client by returning a false result $\tilde{\mathcal{C}}(w)^*(\neq \mathcal{C}(w))$ during the search phase. We generally call this notion (weak) reliability. In [20], *strong reliability* was also defined, and a relation between strong reliability and universal composability was discussed. Strong reliability is formulated by considering the game $\texttt{Game}_{reli}$ shown in Fig. 3. This game is played by an adversary $\mathbf{A} = (\mathbf{A}_1, \mathbf{A}_2)$ (malicious server) and a challenger $\mathbf{C}$. We assume that $\mathbf{A}_1$ and $\mathbf{A}_2$ can communicate freely.

Definition 2 ((Strong)Reliability). *We say that a vSSE scheme satisfies (strong) reliability if for any PPT adversary* $\mathbf{A}$*,* $\Pr(\mathbf{A}$ *wins*$)$ *is negligible for any* $(\mathcal{D}, \mathcal{W})$ *and any search queries* $w_0, \ldots, w_{q-1}$.

From now on, we will say just *Reliability* for what we mean *Strong Reliability.*

Universally Composable Security. It is known that if protocol Σ is secure in the universally composable (UC) security framework, then the security of Σ is maintained even if it is combined with other protocols. The security in the

Store: Upon receiving the input $(\textbf{store}, sid, D_0, \ldots, D_{N-1}, \mathcal{W})$ from the (dummy) client, verify that this is the first input from the client with $(\textbf{store}, sid)$.
If it is, then store $\mathcal{D} = \{D_0, \ldots, D_{N-1}\}$, and send $L_1(\mathcal{D}, \mathcal{W})$ to $\mathbf{S}$. Otherwise, ignore this input.

Search: Upon receiving $(\textbf{search}, sid, w)$ from the client, send $L_2(\mathcal{D}, \mathcal{W}, \mathbf{w}, w)$ to $\mathbf{S}$.

1. If $\mathbf{S}$ returns "OK," then send $\mathcal{D}(w)$ to the client.
2. If $\mathbf{S}$ returns $\perp$, then send $\perp$ to the client.

Fig. 4. Ideal functionality $\mathcal{F}^L_{vSSE}$

UC framework is defined by associating it with a given *ideal functionality* $\mathcal{F}$. Refer to [4–6] for the formal definition of the UC framework. Kurosawa and Ohtaki introduced an ideal functionality of vSSE [18,20]. Here, we generalize the definition in order to handle the general leakage functions $L = (L_1, L_2)$ as shown in Fig. 4. Note that the server does not interact with $\mathcal{F}^L_{vSSE}$, because it does not have its own input and output.

Definition 3 (UC-Security with Leakage L). *We say that vSSE scheme has universally composable (UC) security with leakage L, if it realizes*[3] *the ideal functionality $\mathcal{F}^L_{vSSE}$.*

The following theorem can be proved in the same way as the theorem in [20].

Theorem 1. `vSSE` *has UC security with leakage L against non-adaptive adversaries if the* `vSSE` *scheme satisfies L-privacy and reliability.*

2.3 Kurosawa-Ohtaki Scheme (KO-Scheme)

We next review the UC-secure SSE scheme proposed by Kurosawa and Ohtaki, KO-scheme [18]. In this scheme, the set of searchable keywords is $\mathcal{W} \subseteq \{0,1\}^l$ for some l.

Let $\texttt{SKE} = (G, E, E^{-1})$ be a symmetric encryption scheme, where G is a key-generation algorithm, E is an encryption algorithm, and E^{-1} is the corresponding decryption algorithm. For a security parameter λ, let $\pi : \{0,1\}^\lambda \times \{0,1\}^{l+1+\log N} \to \{0,1\}^{l+1+\log N}$ be a pseudorandom permutation, where N denotes the number of documents in $\mathcal{D}$, and let $\texttt{MAC} : \{0,1\}^\lambda \times \{0,1\}^* \to \{0,1\}^n$ be a tag-generation function. For simplicity, we write $y = \pi(x)$ rather than $y = \pi(K, x)$, and $\texttt{MAC}(m)$ rather than $\texttt{MAC}(K, m)$, where K is a key.

The KO-scheme proceeds as follows:

[3] That is, if there does not exist any environment $\mathcal{Z}$ that can distinguish the real world and the ideal world by interacting with the real-world adversary or the ideal-world adversary.

$\texttt{Gen}(1^\lambda)$: Run G to generate a key K_0 for $\texttt{SKE}$. Randomly choose a key $K_1 \in \{0,1\}^\lambda$ for π and a key $K_2 \in \{0,1\}^\lambda$ for $\texttt{MAC}$. Output $K = (K_0, K_1, K_2)$.

$\texttt{Enc}(K, \mathcal{D}, \mathcal{W})$: First, compute $C_i = E(K_0, D_i)$ for each $D_i \in \mathcal{D}$, and let $\mathcal{C} = \{C_0, \ldots, C_{N-1}\}$. Let $\mathcal{I}$ be an array of size $2 \times 2^l N$. We write $\mathcal{I}[i]$ for the i-th element of $\mathcal{I}$.

1. Let
$$\mathcal{I}[i] \leftarrow (\texttt{dummy}, \texttt{MAC}(i \| \texttt{dummy}))$$
for all $i = 0, \ldots, 2 \times 2^l N - 1$.
2. For each $w \in \{0,1\}^l$, suppose that $\mathcal{D}(w) = (D_{s_1}, \ldots, D_{s_m})$. Then for $j = 1, \ldots, m$, let
$$\begin{aligned} addr &= \pi(0, w, j) \\ tag_{w,j} &= \texttt{MAC}(addr \| C_{s_j}) \\ \mathcal{I}[addr] &\leftarrow (s_j, tag_{w,j}). \end{aligned}$$
3. For each $D_k \in \mathcal{D}$, suppose that document number k appears N_k times in $\mathcal{I}$. Then for $j = 1, \ldots, 2^l - N_k$, let
$$\begin{aligned} addr &= \pi(1, j, k) \\ tag_{j,k} &= \texttt{MAC}(addr \| C_k) \\ \mathcal{I}[addr] &\leftarrow (k, tag_{j,k}). \end{aligned}$$

Finally, output $(\mathcal{I}, \mathcal{C})$.

$\texttt{Trpdr}(K, w)$: Output
$$t(w) = (\pi(0, w, 0), \ldots, \pi(0, w, N-1))$$

$\texttt{Search}(\mathcal{I}, \mathcal{C}, t(w))$: Parse $t(w)$ as $t(w) = (addr_0, \ldots, addr_{N-1})$. Suppose that
$$\mathcal{I}[addr_i] = (s_i, tag_i)$$
for $i = 0, \ldots, N-1$. First, set $\tilde{\mathcal{C}}(w) \leftarrow \texttt{empty}$. Then, for $i = 0, \ldots, N-1$, add C_{s_i} to $\tilde{\mathcal{C}}(w)$, if $s_i \neq \texttt{dummy}$. Set $Proof = (tag_0, \ldots, tag_{N-1})$. Finally, output $(\tilde{\mathcal{C}}(w), Proof)$.

$\texttt{Verify}(K, t(w), \tilde{\mathcal{C}}(w), Proof)$: Parse $t(w), \tilde{\mathcal{C}}(w)$, and $Proof$ as
$$\begin{aligned} t(w) &= (addr_0, \ldots, addr_{N-1}) \\ \tilde{\mathcal{C}}(w) &= (\tilde{C}_0, \ldots, \tilde{C}_{m-1}) \\ Proof &= (tag_0, \ldots, tag_{N-1}). \end{aligned}$$

Then, verify the validity of the result with the following steps.

1. Let $X_i \leftarrow \tilde{C}_i$ for $i = 0, \ldots, m-1$.
2. Let $X_i \leftarrow \texttt{dummy}$ for all $i = m, \ldots, N-1$.
3. If $tag_i = \texttt{MAC}(addr_i \| X_i)$ for all $i = 0, \ldots, N-1$, then output $\texttt{accept}$. Otherwise output $\texttt{reject}$.

$\texttt{Dec}(K, C)$: Output a document $D = E^{-1}_{K_0}(C)$ for a ciphertext C.

Proposition 1. *For $\mathcal{W} \subseteq \{0,1\}^l$, let*

$$\begin{aligned} L^{KO} &= (L_1^{KO}, L_2^{KO}) \\ L_1^{KO}(\mathcal{D}, \mathcal{W}) &= (|D_0|, \ldots, |D_{N-1}|, l) \\ L_2^{KO}(\mathcal{D}, \mathcal{W}, \mathbf{w}, w) &= (\texttt{List}(\mathcal{D}, w), \texttt{SPattern}(\mathbf{w}, w)) \end{aligned}$$

The above scheme has L^{KO}-privacy and reliability. Therefore, it also has UC security with leakage L^{KO}.

Remark 1. In [18], Kurosawa and Ohtaki claimed that only $\texttt{List}(\mathcal{D}, w)$ is leaked during the search phase. However, it is obvious that the trapdoor leaks a search pattern, since `Trpdr` is deterministic.

2.4 Inefficiency of KO-Scheme

With the KO-scheme, the index size is $\mathcal{O}(2^l N)$, where l denotes the bit-length of the keywords. Here we consider a case where $\mathcal{W}$ is a set of English words that includes a long word, namely "indistinguishability." The bit-length of this set of keywords is $l = 8 \times 20$. With such a keyword set, then, the index will become very large.

Let l be the maximum length of the keywords expressed as bit strings, and $\mathcal{W}_0 = \{0,1\}^l$. Then, $\mathcal{W} \subseteq \mathcal{W}_0$ holds. In general, the KO-scheme is inefficient whenever $|\mathcal{W}| \ll |\mathcal{W}_0|$, and it is not uncommon.

An easy solution to this problem is to transform each word into a short bit string as follows: Let $l' = \lceil \log_2 |\mathcal{W}| \rceil$. First, the client numbers the keywords in $\mathcal{W}$ from 0 to $|\mathcal{W}| - 1$. That is, $\mathcal{W} = \{w_0, \ldots, w_{|\mathcal{W}|-1}\}$. For the `Enc` algorithm, the client does not use the keyword $w_i \in \mathcal{W}$, but rather its index $i \in [0, 2^{l'} - 1]$. Then, the index size is $\mathcal{O}(2^{l'} N) \approx \mathcal{O}(|\mathcal{W}|N)$, even if $|\mathcal{W}| \ll 2^l$.

For this solution, however, the client must keep $\mathcal{W}$ on hand in order to translate the keyword w into its index i when searching w.

In the next section, we provide another solution to reduce the size of the index.

3 Improvement of KO-Scheme

In this section, we propose a new vSSE scheme.

The idea of reducing the index size is elimination of `dummy` elements from the index, and introduction of a new tag that proves that the eliminated elements do not exist in the constructed index.

Let $M = |\mathcal{W}|$ be the number of keywords. The index of our scheme is much smaller than the index of the KO-scheme, if $M \ll 2^l$. This means that the computation cost to generate an index and to search it is also reduced.

3.1 Concrete Description of Our Scheme

Here, $\texttt{SKE} = (G, E, E^{-1}), \mathcal{D}, \pi$, and $\texttt{MAC}$ follow the denotations from Sect. 2.3. Our vSSE scheme is as follows:

$\texttt{Gen}(1^\lambda)$: Run G to generate a key K_0 for $\texttt{SKE}$. Randomly choose a key $K_1 \in \{0,1\}^\lambda$ for π and a key $K_2 \in \{0,1\}^\lambda$ for $\texttt{MAC}$. Output $K = (K_0, K_1, K_2)$.

$\texttt{Enc}(K, \mathcal{D}, \mathcal{W})$: First, compute $C_i = E(K_0, D_i)$ for each $D_i \in \mathcal{D}$ and let $\mathcal{C} = \{C_0, \dots, C_{N-1}\}$.

Our index $\mathcal{I}$ is an array of size $MN + 1$, and each element of $\mathcal{I}$ has four fields:

$$(\texttt{addr}, \texttt{ID}, \texttt{tag}, \texttt{Ntag}).$$

Hereafter, $\mathcal{I}[i]$ denotes the i-th element in $\mathcal{I}$, and $\mathcal{I}[i].\texttt{addr}$ denotes the $\texttt{addr}$ field of the i-th element in $\mathcal{I}$. Furthermore, we will use the same notation for the other three fields. $\mathcal{I}$ is constructed as follows.

1. Set

$$\mathcal{I}[MN].\texttt{addr} \leftarrow 2^{l+1+\log N}$$
$$\mathcal{I}[MN].\texttt{ID} \leftarrow \texttt{dummy}$$
$$\mathcal{I}[MN].\texttt{tag} \leftarrow \texttt{dummy}.$$

2. Let

$$p_{i,j} = \begin{cases} \pi(0, w_i, j) & \text{if } D_j \in \mathcal{D}(w_i) \\ \pi(1, w_i, j) & \text{if } D_j \notin \mathcal{D}(w_i) \end{cases}$$

for $w_i \in \mathcal{W}$ and $D_j \in \mathcal{D}$, and set

$$\mathcal{I}[Ni + j].\texttt{addr} \leftarrow p_{i,j}$$
$$\mathcal{I}[Ni + j].\texttt{ID} \leftarrow j$$
$$\mathcal{I}[Ni + j].\texttt{tag} \leftarrow \texttt{MAC}(0 \| p_{i,j} \| C_j)$$

for $i = 0, \dots, M-1$ and $j = 0, \dots, N-1$. At this time, each element of $\mathcal{I}$ is

$$\begin{array}{llrll}
\mathcal{I}[0] & = (p_{0,0}, & 0, & \texttt{MAC}(0\|p_{0,0}\|C_0), & \texttt{undefined}) \\
\mathcal{I}[1] & = (p_{0,1}, & 1, & \texttt{MAC}(0\|p_{0,1}\|C_1), & \texttt{undefined}) \\
 & \vdots & & & \\
\mathcal{I}[N-1] & = (p_{0,N-1}, & N-1, & \texttt{MAC}(0\|p_{0,N-1}\|C_{N-1}), & \texttt{undefined}) \\
\mathcal{I}[N] & = (p_{1,0}, & 0, & \texttt{MAC}(0\|p_{1,0}\|C_0), & \texttt{undefined}) \\
 & \vdots & & & \\
\mathcal{I}[2N-1] & = (p_{1,N-1}, & N-1, & \texttt{MAC}(0\|p_{1,N-1}\|C_{N-1}), & \texttt{undefined}) \\
 & \vdots & & & \\
\mathcal{I}[MN-1] & = (p_{M-1,N-1}, & N-1, & \texttt{MAC}(0\|p_{M-1,N-1}\|C_{N-1}), & \texttt{undefined}).
\end{array}$$

Note that all values in the $\texttt{addr}$ field are distinct, because π is a permutation.

3. Sort $\mathcal{I}[0], \ldots, \mathcal{I}[MN]$ based on the addr field, such that

$$\mathcal{I}[0.\mathtt{addr} < \mathcal{I}[1].\mathtt{addr} < \cdots < \mathcal{I}[MN].\mathtt{addr}.$$

4. For $r = 0, \ldots, MN$, compute

$$Ntag_r = \begin{cases} \mathtt{MAC}(1\|0\|\mathcal{I}[r].\mathtt{addr}) & \text{if } r = 0 \\ \mathtt{MAC}(1\|\mathcal{I}[r-1].\mathtt{addr}\|\mathcal{I}[r].\mathtt{addr}) & \text{if } r \neq 0 \end{cases}$$

and set

$$\mathcal{I}[r].\mathtt{Ntag} \leftarrow Ntag_r.$$

Finally, output $(\mathcal{I}, \mathcal{C})$.

$\mathtt{Trpdr}(K, w)$: Output

$$t(w) = (\pi(0, w, 0), \ldots, \pi(0, w, N-1)).$$

$\mathtt{Search}(\mathcal{I}, \mathcal{C}, t(w))$: First set $\tilde{\mathcal{C}}(w) \leftarrow \mathtt{empty}$. Parse $t(w)$ as $t(w) = (addr_0, \ldots, addr_{N-1})$. For $i = 0, \ldots, N-1$, search $addr_i$ from the addr field in $\mathcal{I}$, and follow the steps below:

- If $addr_i = \mathcal{I}[r_i].\mathtt{addr}$ for some $r_i \in [0, MN-1]$, then set

$$\tilde{\mathcal{C}}(w) \leftarrow \tilde{\mathcal{C}}(w) \cup C_{\mathcal{I}[r_i].\mathtt{ID}}$$
$$pr_i \leftarrow \mathcal{I}[r_i].\mathtt{tag}.$$

- If $addr_i < \mathcal{I}[0].\mathtt{addr}$, set

$$pr_i \leftarrow (0\|\mathcal{I}[0].\mathtt{addr},\ \mathcal{I}[0].\mathtt{Ntag}).$$

- If $\mathcal{I}[r_i - 1].\mathtt{addr} < addr_i < \mathcal{I}[r_i].\mathtt{addr}$ for some $r_i \in [1, MN]$, then set

$$pr_i \leftarrow (\mathcal{I}[r_i - 1].\mathtt{addr}\|\mathcal{I}[r_i].\mathtt{addr},\ \mathcal{I}[r_i].\mathtt{Ntag}).$$

Set $Proof = (pr_0, \ldots, pr_{N-1})$. Finally, output $(\tilde{\mathcal{C}}(w), Proof)$.

$\mathtt{Verify}(K, t(w), \tilde{\mathcal{C}}(w), Proof)$: Parse $\tilde{\mathcal{C}}(w)$, $Proof$, and $t(w)$ as

$$\tilde{\mathcal{C}}(w) = (\tilde{C}_0, \ldots, \tilde{C}_{m-1})$$
$$Proof = (pr_0, \ldots, pr_{N-1})$$
$$t(w) = (addr_0, \ldots, addr_{N-1})$$

and follow the steps below to verify the validity of the search result.

1. If m is not equal to the number of pr_js that are tags—meaning that they do not consist of a pair of $addr$ and $Ntag$—then output reject.
2. For each pr_j that is a tag, if there does not exist a distinct $i \in \{0, \ldots, m-1\}$, such that

$$\mathtt{MAC}(0\|addr_j\|\tilde{C}_i) = pr_j, \tag{3}$$

then output reject.

3. For a pr_j that is not a tag but rather a pair of $addr$ and $Ntag$, assume that $pr_j = (addr'_{j,1} \| addr'_{j,2}, Ntag_j)$. If the following two statements are true for all such j, then output `accept`.

$$addr'_{j,1} < addr_j < addr'_{j,2} \tag{4}$$
$$\texttt{MAC}(1 \| addr'_{j,1} \| addr'_{j,2}) = Ntag_j \tag{5}$$

Otherwise, output `reject`.

$\texttt{Dec}(K, C)$: Output document $D = E^{-1}_{K_0}(C)$ for a ciphertext C.

3.2 Security

Let

$$L^{new} = (L_1^{new}, L_2^{new})$$
$$L_1^{new}(\mathcal{D}, \mathcal{W}) = L_1^{KO}(\mathcal{D}, \mathcal{W}) \cup |\mathcal{W}| = (|D_0|, \ldots, |D_{N-1}|, l, |\mathcal{W}|)$$
$$L_2^{new}(\mathcal{D}, \mathcal{W}, \mathbf{w}, w) = L_2^{KO}(\mathcal{D}, \mathcal{W}, \mathbf{w}, w) = (\texttt{List}(\mathcal{D}, w), \texttt{SPattern}(\mathbf{w}, w))$$

We can prove that the above scheme satisfies L^{new}-privacy and reliability, and therefore UC-security with leakage L^{new}.

Theorem 2. *If the symmetric encryption scheme* $\texttt{SKE} = (G, E, E^{-1})$ *is secure in terms of indistinguishability against chosen-plaintext attacks (IND-CPA), and if* π *is a pseudorandom permutation, then the proposed scheme satisfies* L^{new}*-privacy.*

Proof. We construct the simulator $\mathbf{S}$ as follows. First, $\mathbf{S}$ receives $L_1(\mathcal{D}, \mathcal{W}) = (|D_0|, \ldots, |D_{N-1}|, l, |\mathcal{W}|)$.

1. $\mathbf{S}$ runs $\texttt{Gen}(1^\lambda)$ to generate key $K = (K_0, K_1, K_2)$.
2. Let $C'_j = E(K_0, 0^{|D_j|})$ for $j = 0, \ldots, N-1$, and let $\mathcal{C}' = \{C'_0, \ldots, C'_{N-1}\}$.
3. Choose $w'_0, \ldots, w'_{M-1}$ from $\{0,1\}^l$ randomly, and set $\mathcal{W}' = \{w'_0, \ldots, w'_{M-1}\}$. Compute the index $\mathcal{I}'$ as if all of the documents $D_0, \ldots, D_{N-1} \in \mathcal{D}$ include all of the keywords in $\mathcal{W}'$. That is, for $i = 0, \ldots, M-1$ and $j = 0, \ldots, N-1$,

$$\mathcal{I}'[r'_{i,j}] = (\pi(0, w'_i, j), j, tag'_{i,j}, \texttt{undefined}) \tag{6}$$
$$\mathcal{I}'[MN] = (2^{l+1+\log N}, \texttt{dummy}, \texttt{MAC}(0 \| \texttt{dummy}), \texttt{undefined})$$

where

$$r'_{i,j} = Ni + j$$
$$tag'_{i,j} = \texttt{MAC}(0 || \pi(0, w'_i, j) || C'_j).$$

Next, sort the elements based on the `addr` field, and set

$$\mathcal{I}'[r].\texttt{Ntag} \leftarrow \texttt{MAC}(1 \| \mathcal{I}'[r-1].\texttt{addr} \| \mathcal{I}'[r].\texttt{addr}).$$

4. Return $(\mathcal{I}',\mathcal{C}')$.

During the i-th search iteration, **S** is given

$$\texttt{List}(w_i) = \{s_0, \ldots, s_{m-1}\}$$

and

$$\texttt{SPattern}(\mathbf{w}, w_i) = (sp_1, \ldots, sp_{i-1})$$

(but neither w_i nor $\mathbf{w}$). **S** simulates the trapdoor as follows.

1. If $sp_j = 1$ for some $j < i$, then **S** sets $t'_i = t'_j$ and returns t'_i.
2. If $\texttt{List}(w_i) = \emptyset$, then **S** randomly chooses $w' \in \{0,1\}^l \backslash (\mathcal{W}' \cup \mathcal{W}_{\text{used}})$[4], otherwise, **S** randomly chooses $w' \in \mathcal{W}' \backslash \mathcal{W}_{\text{used}}$, where $\mathcal{W}_{\text{used}}$ is initially empty. Then, **S** sets

$$\mathcal{W}_{\text{used}} = \mathcal{W}_{\text{used}} \cup \{w'\}$$
$$addr'_j = \begin{cases} \pi(0, w', j) & \text{if } j \in \texttt{List}(w_i) \\ \pi(1, w', j) & \text{if } j \notin \texttt{List}(w_i) \end{cases}$$

for $j = 0, \ldots, N-1$, and returns

$$t'_i = (addr'_0, \ldots, addr'_{N-1}).$$

We will prove that there is no adversary **A** who can distinguish the games $\texttt{Game}_{real}$ and $\texttt{Game}_{sim}$ by using six games $\texttt{Game}_0, \ldots, \texttt{Game}_5$. Let $\texttt{Game}_0 = \texttt{Game}_{real}$. Hereafter, we write

$$P_i = \Pr(\mathbf{A}\,\text{outputs}\,b = 1\,\text{in}\,\texttt{Game}_i)$$

for simplicity.

– $\texttt{Game}_1$ is equivalent to $\texttt{Game}_0$, except that each $C_j = E(K_0, D_j)$ is replaced with $C'_j = E(K_0, 0^{|D_j|})$ for $j = 0, \ldots, N-1$. From the assumption for SKE, $|P_0 - P_1|$ is negligible.
– $\texttt{Game}_2$ uses a real random permutation π_2 for computing *addr*s rather than pseudorandom permutation π as with $\texttt{Game}_1$. Then, $|P_1 - P_2|$ is negligible, owing to the pseudorandomness of π.
– $\texttt{Game}_3$ is equivalent to $\texttt{Game}_2$, except that the set of keywords is changed from $\mathcal{W}$ to $\mathcal{W}'(|\mathcal{W}'| = M)$, and the random permutation is changed to π_3, whose output for a keyword $w_i \in \mathcal{W}$ is the same as the output of π_2 for input $w'_i \in \mathcal{W}'$, and the output for $w'_i \in \mathcal{W}$ is the same as the output of π_2 for $w_i \in \mathcal{W}$ for all i. Then, π_3 is a random permutation, as is π_2, and the constructed indexes for $\texttt{Game}_2$ and $\texttt{Game}_3$ are identical. Hence, $|P_2 - P_3|$ is negligible.

[4] If $\{0,1\}^l \backslash (\mathcal{W}' \cup \mathcal{W}_{\text{used}}) = \emptyset$, w' is chosen from $\mathcal{W}' \backslash \mathcal{W}_{\text{used}}$.

– $\texttt{Game}_4$ is equivalent to $\texttt{Game}_3$, except that the $\texttt{List}(w')$ in $\texttt{Game}_3$ is replaced by $\texttt{List}'(w') = \{0, \ldots, M-1\}$ for all $w' \in \mathcal{W}'$, and π_3 is replaced by π_4, which satisfies

$$\pi_4(0\|w'\|j) = \begin{cases} \pi_3(0\|w'\|j) & \text{if } j \in \texttt{List}(w') \\ \pi_3(1\|w'\|j) & \text{if } j \notin \texttt{List}(w') \end{cases}$$

$$\pi_4(1\|w'\|j) = \begin{cases} \pi_3(1\|w'\|j) & \text{if } j \in \texttt{List}(w') \\ \pi_3(0\|w'\|j) & \text{if } j \notin \texttt{List}(w') \end{cases}$$

for all $j = 0, \ldots, N-1$ and all w'. Then, π_4 is also a random permutation, and the constructed indexes for $\texttt{Game}_3$ and $\texttt{Game}_4$ are identical. Hence, $|P_3 - P_4|$ is negligible.

– In $\texttt{Game}_5$, we use pseudorandom permutation π, rather than π_4, and this is the only difference between $\texttt{Game}_5$ and $\texttt{Game}_4$. Because π_4 is a random permutation, $|P_4 - P_5|$ is negligible, owing to the pseudorandomness of π.

From the above, $|P_0 - P_5|$ is negligible. Since it is obvious that $\texttt{Game}_5 = \texttt{Game}_{sim}$, $\texttt{Game}_{real}$ and $\texttt{Game}_{sim}$ are indistinguishable for any adversary $\mathbf{A}$. □

Theorem 3. *If* MAC *is existentially unforgeable against chosen-message attacks, our scheme satisfies reliability.*

Proof. Suppose that for $(\mathcal{D}, \mathcal{W})$ and search queries $w_0, \ldots, w_{q-1}$, there exists an adversary $\mathbf{A} = (\mathbf{A}_1, \mathbf{A}_2)$ who can break the reliability. We show that a forger $\mathbf{B}$ against MAC can be constructed using $\mathbf{A}$.

$\mathbf{B}$ behaves like a client. When $\mathbf{B}$ receives $(\mathcal{D}, \mathcal{W})$ from $\mathbf{A}_1$ during the store phase, it creates $\mathcal{I}$ and $\mathcal{C}$ ordinarily, except that $\mathbf{B}$ does not choose the key for MAC, but rather uses its own MAC oracle to compute $\mathcal{I}$. Here, $\mathbf{B}$ will send queries to its MAC oracle only when constructing $\mathcal{I}$.then $\mathbf{B}$ sends $(\mathcal{I}, \mathcal{C})$ to $\mathbf{A}_2$. We note that $\mathbf{B}$ will send queries to its MAC oracle only when constructing $\mathcal{I}$.

In the search phase, $\mathbf{A}_1$ sends w_i to $\mathbf{B}$ for q times. $\mathbf{B}$ calculates a trapdoor $t(w_i)$ for w_i normally and sends it to $\mathbf{A}_2$. $\mathbf{A}_2$ outputs $(\tilde{\mathcal{C}}(w_i)^*, \mathit{Proof}_i^*)$ and sends it back to $\mathbf{B}$. While this step, $\mathbf{B}$ also runs the $\texttt{Search}$ algorithm and gets $(\mathcal{C}(w_i), \mathit{Proof}_i)$ for its own.

For each i, $\mathbf{A}_2$'s output $(\tilde{\mathcal{C}}(w_i)^*, \mathit{Proof}_i^*)$ is either of the following three types:

Type 1 $(\tilde{\mathcal{C}}(w_i)^*, \mathit{Proof}_i^*) = (\mathcal{C}(w_i), \mathit{Proof}_i)$.
Type 2 $(\tilde{\mathcal{C}}(w_i)^*, \mathit{Proof}_i^*) \neq (\mathcal{C}(w_i), \mathit{Proof}_i)$ and:
  Type 2-1 the `Verify` algorithm outputs `reject`.
  Type 2-2 the `Verify` algorithm outputs `accept`.

For each output of **Type 1**, $\mathbf{B}$ returns $\mathcal{D}(w_i)$ to $\mathbf{A}_1$.

For each output of **Type 2**, $\mathbf{B}$ has to return `reject` if it is **Type 2-1**, and a plaintext $\tilde{\mathcal{D}}(w_i)^*$ of $\tilde{\mathcal{C}}(w_i)^*$ if it is **Type 2-2**. However, $\mathbf{B}$ cannot distinguish **Type 2-1** and **Type 2-2**, since $\mathbf{B}$ does not have the key for the MAC itself.

For this problem, $\mathbf{B}$ randomly chooses J from $[1, q]$ at the beginning of the search phase. This J is the prediction by $\mathbf{B}$ of i such that i-th output is the first **Type 2-2** output. Based on this J, $\mathbf{B}$ performs as follows.

For outputs of $\mathbf{A}_2$ before the J-th output in the search phase, **B** considers all **Type 2** outputs to be **Type 2-1**, and returns `reject` to $\mathbf{A}_1$.

If the J-th output of $\mathbf{A}_2$ is not **Type 2**, **B** fails to forge a MAC and aborts. Otherwise, **B** considers it as **Type 2-2**, and determines its output as below.

Case a: $\mathcal{C}(w_J) \neq \tilde{\mathcal{C}}(w_J)^*$

Suppose that

$$\begin{aligned} t(w_J) &= (addr_0, \ldots, addr_{N-1}) \\ \mathcal{C}(w_J) &= (C_{q_0}, \ldots, C_{q_{m-1}}) \\ \tilde{\mathcal{C}}(w_J)^* &= (C_0^*, \ldots, C_{m^*-1}^*) \\ Proof_J^* &= (pr_0^*, \ldots, pr_{N-1}^*). \end{aligned}$$

Note that $addr_j = \pi(0, w_J, j)$, and that **B** knows all of the above values. Because $\tilde{\mathcal{C}}(w_J)^* \neq \mathcal{C}(w_J)$, we need to consider only the following three cases.
Case a-1: $m^* = m$, and there exists C_i^* in $\tilde{\mathcal{C}}(w_J)^*$ but not in $\mathcal{C}(w_J)$
Case a-2: $m^* > m$
Case a-3: $m^* < m$.
In both **Cases a-1** and **a-2**, there exists C_i^* in $\tilde{\mathcal{C}}(w_J)^*$ but not in $\mathcal{C}(w_J)$. Then, **B** randomly chooses $pr_j^* \in Proof_J^*$ that is a *tag*, and outputs a forged message-tag pair $((0\|addr_j\|C_i^*), pr_j^*)$. Here we assume that **A** wins in $\texttt{Game}_{reli}$ and that **B** successfully predicts J, that is,

$$\texttt{Verify}(K, t(w_J), \tilde{\mathcal{C}}(w_J)^*, Proof_J^*) = \texttt{accept} \tag{7}$$

holds with a non-negligible probability. We show that **B**'s output shown above is a valid forgery against `MAC` with a non-negligible probability. This means that $pr_j^* = \texttt{MAC}(0\|addr_j\|C_i^*)$, and that **B** did not send the query $(0\|addr_j\|C_i^*)$ to its own MAC oracle.

First, from Eqs. (3) and (7), there exists $pr_{j'}^*$ such that $\texttt{MAC}(0\|addr_{j'}\|C_i^*) = pr_{j'}^*$, in $Proof_J^*$. $j = j'$ holds with at least probability $1/m$. Next, we can see that **B** has never queried $(0\|addr_{j''}\|C_i^*)$ for any j'' to its own MAC oracle when computing $\mathcal{I}$, because $C_i^* \notin \mathcal{C}(w_J)$. Therefore, **B** has succeeded in forging a valid tag with non-negligible probability.

In **Case a-3**, there exists C_{q_i} in $\mathcal{C}(w_J)$, but not in $\tilde{\mathcal{C}}(w_J)^*$. Then, $pr_{q_i}^*$ consists of a pair of *addr*s and *Ntag*. Let $pr_{q_i}^* = (addr_1^*\|addr_2^*, Ntag^*)$. **B** outputs $((1\|addr_1^*\|addr_2^*), Ntag^*)$ as a forgery. Assume that **A** wins in $\texttt{Game}_{reli}$. From Eqs.(4) and (5),

$$\begin{gathered} addr_1^* < addr_{q_i} < addr_2^* \\ \texttt{MAC}(1\|addr_1^*\|addr_2^*) = Ntag^* \end{gathered}$$

hold. Further, $C_{q_i} \in \mathcal{C}(w_J)$ implies that $addr_{q_i}$ appears in the `addr` field of $\mathcal{I}$. Therefore, **B** did not query $(1\|addr'_1\|addr'_2)$ to the MAC oracle for any $(addr'_1, addr'_2)$ such that $addr'_1 < addr_{q_i} < addr'_2$. This means that if **A** wins in $\texttt{Game}_{reli}$ with non-negligible probability, then **B** succeeds with a non-negligible probability. This contradicts the assumption about the security of $\mathtt{MAC}$.

Case b: $\mathcal{C}(w_J) = \tilde{\mathcal{C}}(w_J)^*$
From $(\mathcal{C}(w_J), \mathit{Proof}_J) \neq (\tilde{\mathcal{C}}(w_J)^*, \mathit{Proof}^*_J)$, it is obvious that $\mathit{Proof}_J \neq \mathit{Proof}^*_J$.
Suppose that

$$\mathit{Proof}_J = (pr_1, \dots, pr_N),$$
$$\mathit{Proof}^*_J = (pr^*_1, \dots, pr^*_N)$$

Then, there exists an i *s.t.* $pr_i \neq pr^*_i$. Since pr_i and pr^*_i are either a *tag* or a pair $(addr_1 \| addr_2, Ntag)$ of *addr*s and $Ntag$, every case will be either of the following four cases.
Case b-1: both pr_i and pr^*_i are *tag*s
Case b-2: both pr_i and pr^*_i are pairs $(addr_1 \| addr_2, Ntag)$
Case b-3: pr_i is a *tag*, and pr^*_i is a pair $(addr_1 \| addr_2, Ntag)$
Case b-4: pr_i is a pair $(addr_1 \| addr_2, Ntag)$, and pr^*_i is a *tag*.
In **Case b-1**, **B** knows the C_j which satisfies $pr_i = \mathtt{MAC}(0 \| addr_i \| C_j)$. So it chooses another $C_{j'}$ from $\mathcal{C}(w_J)$ randomly, and outputs $(0 \| addr_i \| C_{j'})$ and pr^*_i.

In **Cases b-2** and **b-3**, **B** outputs $((1 \| addr^*_1 \| addr^*_2), Ntag^*)$, where $pr^*_i = (addr^*_1 \| addr^*_2, Ntag^*)$.

In **Case b-4**, if there exists an $i' (\neq i)$ where **Case b-3** is occurring, then **B** applies exactly the same method as in **Case b-3** to $pr^*_{i'}$ instead of pr^*_i[5].

If **A** succeeds in breaking the reliability, **B** successfully predicts J in probability $1/q$. When **A** wins and **B** successfully predicts J, then **B** successfully forges a MAC with probability at least $1/N$. Therefore, we obtain

$$\Pr(\mathbf{B}\,\text{succeeds}) \geq \Pr(\mathbf{A}\,\text{wins in}\,\mathtt{Game}_{reli}) \times \frac{1}{qN}.$$

Note that q and N are polynomials of security parameter λ.

As a result, our scheme satisfies reliability if MAC is unforgeable against chosen message attack. □

From Theorems 1, 2, and 3, our scheme is UC-secure.

Corollary 1. *If the symmetric encryption scheme* $\mathtt{SKE} = (G, E, E^{-1})$ *is IND-CPA secure, and if* π *is a pseudorandom permutation, and if* $\mathtt{MAC}$ *is existentially unforgeable against chosen-message attacks, then the above scheme is UC-secure with leakage* L^{new}.

[5] When Proof^*_J is accepted by `Verify`, such i' will always exist because the `Verify` algorithm starts with a step to check whether the number of the *tag*s in *Proof* is equal to the numbers of encrypted documents in the search result $\mathcal{C}(w_J) = \mathcal{C}(w_J)$, and output `reject` if not.

Table 1. Comparison of the efficiency of proposed scheme with the KO-scheme

		KO-scheme [18]	Proposed scheme
Index Size(bits)		$\mathcal{O}(2^l N \log N)$	$\mathcal{O}(MN \log N)$
Complexity	`Enc`	$\mathcal{O}(2^l N)$	$\mathcal{O}(MN)$
	`Search`	$\mathcal{O}(N)$	$\mathcal{O}(N \log MN)$
	`Verify`	$\mathcal{O}(N)$	$\mathcal{O}(N + m^2)$

N : number of documents
l : (maximum) length of keywords
M : number of keywords $|\mathcal{W}|$ $(M \leq 2^l)$
m : number of documents in the search result

The leaked information under our scheme is slightly more than the leakage from the KO-scheme. In particular, our scheme leaks the number of keywords to the server.

Should the client prefer to avoid leaking the exact number of keywords, dummy keywords can be added to $\mathcal{W}$. Of course, the more dummy keywords that are added, the larger the index grows. This is constitutes a trade-off between security and computational costs.

Nevertheless, the proposed scheme can modify the maximum length of a keyword l by replacing the permutation π with a collision resistant hash function.

3.3 Comparison

Table 1 shows the index size and the computational cost for each algorithm, comparing the KO-scheme with the proposed scheme. In the estimation on the cost of `Enc` algorithm, we ignored the cost of sorting, which we see it as negligible compared to `MAC`. We eliminated the rows for algorithms `Gen`, `Trpdr`, `Dec`, inasmuch as they are exactly the same in both schemes.

We can see that our `Enc` algorithm is much more efficient and that the index size is much smaller than the KO-scheme, when $|\mathcal{W}| \ll 2^l$. However, our `Search` and `Verify` algorithms are less efficient than those in the KO-scheme, because our scheme requires an extra step to search the *addr*s from the `addr` field in `Search`, and to search the C_is corresponding to *tag*s in `Verify`.

4 Conclusion

In this paper, we provided generalized definitions for the privacy and UC-security of SSE schemes, and we proposed a vSSE scheme as a modified version of the KO-scheme. Whereas the privacy of the proposed scheme is slightly weaker than the original, the index size is much smaller when the set of keywords is a very sparse subset of l-bit strings for some l.

Importantly, the idea of $Ntag$ to prove "non-existence" can be applied widely. For example, the weak point of the (dynamic) vSSE scheme [19] can be overcome by adding an $Ntag$ after sorting the elements in $\mathcal{I}$ based on `label`. Similarly, most SSE schemes might be converted to vSSE schemes by including a tag and an $Ntag$.

References

1. Ballard, L., Kamara, S., Monrose, F.: Achieving efficient conjunctive keyword searches over encrypted data. In: Qing, S., Mao, W., López, J., Wang, G. (eds.) ICICS 2005. LNCS, vol. 3783, pp. 414–426. Springer, Heidelberg (2005)
2. Bellare, M., Boldyreva, A., O'Neill, A.: Deterministic and efficiently searchable encryption. In: Menezes, A. (ed.) CRYPTO 2007. LNCS, vol. 4622, pp. 535–552. Springer, Heidelberg (2007)
3. Boldyreva, A., Chenette, N.: Efficient Fuzzy search on encrypted data. IACR Cryptology ePrint Archive 2014/235
4. Canetti, R.: Universally composable security: "A New Paradigm for Cryptographic," protocols. Revision 1 of ECCC Report TR01-016 (2001)
5. Canetti, R.: Universally composable signatures, certification and authentication. Cryptology ePrint Archive, Report 2003/239 (2003). http://eprint.iacr.org/
6. Canetti, R.: Universally composable security: a new paradigm for cryptographic protocols. Cryptology ePrint Archive, Report 2000/067 (2005). http://eprint.iacr.org/
7. Cao, N., Wang, C., Li, M., Ren, K., Lou, W.: Privacy-preserving multi-keyword ranked search over encrypted cloud data. IEEE Trans. Parallel Distrib. Syst. **25**, 222–233 (2014)
8. Cash, D., Jarecki, S., Jutla, C., Krawczyk, H., Roşu, M.-C., Steiner, M.: Highly-scalable searchable symmetric encryption with support for Boolean queries. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part I. LNCS, vol. 8042, pp. 353–373. Springer, Heidelberg (2013)
9. Chang, Y.-C., Mitzenmacher, M.: Privacy preserving keyword searches on remote encrypted data. In: Ioannidis, J., Keromytis, A.D., Yung, M. (eds.) ACNS 2005. LNCS, vol. 3531, pp. 442–455. Springer, Heidelberg (2005)
10. Chai, Q., Gong, G.: Verifiable symmetric searchable encryption for semi-honest-but-curious cloud servers. In: 2012 IEEE International Conference on Communications (ICC), pp. 917–922 (2012)
11. Curtmola, R., Garay, J.A., Kamara, S., Ostrovsky, R.: Searchable symmetric encryption: improved definitions and efficient constructions. In: ACM Conference on Computer and Communications Security, pp. 79–88 (2006). Full version: Cryptology ePrint Archive, Report 2006/210. http://eprint.iacr.org/
12. Goh, E.-J.: Secure indexes. Technical Report 2003/216, IACR ePrint Cryptography Archive (2003)
13. Golle, P., Staddon, J., Waters, B.: Secure conjunctive keyword search over encrypted data. In: Jakobsson, M., Yung, M., Zhou, J. (eds.) ACNS 2004. LNCS, vol. 3089, pp. 31–45. Springer, Heidelberg (2004)
14. Kamara, S., Lauter, K.: Cryptographic cloud storage. In: Sion, R., Curtmola, R., Dietrich, S., Kiayias, A., Miret, J.M., Sako, K., Sebé, F. (eds.) RLCPS, WECSR, and WLC 2010. LNCS, vol. 6054, pp. 136–149. Springer, Heidelberg (2010)
15. Kamara, S., Papamanthou, C., Roeder, T.: CS2: a searchable cryptographic cloud storage system. MSR Technical Report no. MSR-TR-2011-58. Microsoft (2011)

16. Kamara, S., Roeder, T.: Dynamic searchable symmetric encryption. In: Proceedings of the 2012 ACM Conference on Computer and Communications Security, pp. 965–976 (2012)
17. Kamara, S., Papamanthou, C.: Parallel and dynamic searchable symmetric encryption. In: Sadeghi, A.-R. (ed.) FC 2013. LNCS, vol. 7859, pp. 258–274. Springer, Heidelberg (2013)
18. Kurosawa, K., Ohtaki, Y.: UC-secure searchable symmetric encryption. In: Keromytis, A.D. (ed.) FC 2012. LNCS, vol. 7397, pp. 285–298. Springer, Heidelberg (2012)
19. Kurosawa, K., Ohtaki, Y.: How to update documents *Verifiably* in searchable symmetric encryption. In: Abdalla, M., Nita-Rotaru, C., Dahab, R. (eds.) CANS 2013. LNCS, vol. 8257, pp. 309–328. Springer, Heidelberg (2013)
20. Kurosawa, K., Ohtaki, Y.: How to construct UC-secure searchable symmetric encryption scheme. Cryptology ePrint Archive, Report 2015/251 (2015). http://eprint.iacr.org/2015/251
21. Song, D., Wagner, D., Perrig, A.: Practical techniques for searches on encrypted data. In: IEEE Symposium on Security and Privacy, pp. 44–55 (2000)
22. Wang, C., Ren, K., Yu, S., Urs, K.M.R.: Achieving usable and privacy-assured similarity search over outsourced cloud data. In: Proceedings of INFOCOM 2012, pp. 451–459 (2012)