PAPER *Special Section on Cryptography and Information Security*

No-Dictionary Searchable Symmetric Encryption*

Wakaha OGATA[†a)], ***Member and*** **Kaoru KUROSAWA**[††b)], ***Fellow***

SUMMARY In the model of *no-dictionary* searchable symmetric encryption (SSE) schemes, the client does not need to keep the list of keywords $\mathcal{W}$. In this paper, we first show a generic method to transform any passively secure SSE scheme to a *no-dictionary* SSE scheme such that the client can verify search results even if $w \notin \mathcal{W}$. In particular, it takes only $O(1)$ time for the server to prove that $w \notin \mathcal{W}$. We next present a no-dictionary SSE scheme such that the client can hide even the search pattern from the server.
key words: *searchable symmetric encryption, dictionary, verifiable, search pattern*

1. Introduction

1.1 Background

The notion of searchable symmetric encryption (SSE) schemes was introduced by Song et al. [34]. In the store phase, a client encrypts a set of files and an index table by a symmetric encryption scheme, and then stores them on an untrusted server. In the search phase, he can efficiently retrieve the matching files for a search keyword w keeping the keyword and the files secret.

Since then, single keyword search SSE schemes [15], [16], [19], [24], [26], dynamic SSE schemes [13], [21], [22], [25], [27], [30], verifiable SSE schemes [24]–[27], [35], multiple keyword search SSE schemes [1], [7], [12], [20], [23], [36] and more [14] have been studied extensively by many researchers.

Curtmola, et al. [16], [17] gave a rigorous definition of privacy against honest but curious servers. Kurosawa and Ohtaki [24], [26] showed a definition of reliability against malicious servers who may return incorrect search results to the client, or may delete some encrypted files to save her memory space. An SSE scheme is called verifiable if it satisfies both privacy and reliability.

Let $\mathcal{D} = \{D_1, \ldots, D_N\}$ be the set of files and $\mathcal{W} = \{w_1, \ldots, w_m\}$ be the set of keywords, where each keyword w is contained in some file(s). We call $\mathcal{W}$ a dictionary.

Let $\mathcal{ID}(w) = \{j \mid D_j \text{ contains } w\}$. Then an index table $\mathcal{T}$ is defined as $\mathcal{T} = (\mathcal{ID}(w_1), \ldots, \mathcal{ID}(w_m))$, where $w_i \in \mathcal{W}$. Let $\mathcal{I}$ be an encryption of $\mathcal{T}$. In the store phase, the client sends $\mathcal{I}$ and an encryption of $\mathcal{D}$ to the server.

We say that an SSE scheme is a *no-dictionary* SSE scheme if the client does not need to keep $\mathcal{W}$. In usual SSE schemes, the client does not need to keep $\mathcal{W}$. However, there are some exceptional cases. In this paper, we study two cases in which it is non-trivial to design an efficient no-dictionary SSE scheme. (The notion of no-dictionary SSE schemes was first studied by Taketani and Ogata [35] in the setting of verifiable SSE schemes.)

1.2 No-Dictionary SSE with Search Pattern Hiding

The search pattern is the information on which past queries are the same as the current one, where a query is an encryption of a search word w. In usual SSE schemes, the search pattern is leaked to the server.

If the client keeps a dictionary $\mathcal{W}$, we can construct a search pattern hiding SSE scheme by using the technique of private information retrieval (PIR) [29], [32][†] (The cost for it is that the communication complexity and the computation complexity increase.).

In the store phase, the client stores an encrypted index table $\mathcal{I}_0 = (\mathcal{I}_0[1], \ldots, \mathcal{I}_0[m])$ such that $\mathcal{I}_0[i]$ is an encryption of $\mathcal{T}[i](= \mathcal{ID}(w_i))$, where $w_i \in \mathcal{W}$ for each i. In the search phase, by using PIR, he obtains $\mathcal{I}_0[i]$ from the server without revealing any information on the search word $w_i \in \mathcal{W}$. This means that the search pattern is hidden from the server. He finally retrieves encryptions of all D_j such that $j \in \mathcal{T}[i]$ from the server.

If the client does not want to keep $\mathcal{W}$ (i.e. no-dictionary SSE), there is a simple way to modify the above scheme. Let b be the bit length of the longest keyword in $\mathcal{W}$, and let $\pi : \{0,1\}^{\leq b} \to \{0,1\}^{\ell}$ be an injection for some ℓ. The client constructs an extended index table $\mathcal{T}_e$ of size 2^{ℓ} such that $\mathcal{T}_e[\pi(w)] = \mathcal{ID}(w)$. Then he stores $\mathcal{I}_e = (\mathcal{I}_e[1], \ldots, \mathcal{I}_e[2^{\ell}])$ such that $\mathcal{I}_e[i]$ is an encryption of $\mathcal{T}_e[i]$ to the server, and keeps only (b, π). In this way, we can obtain a no-dictionary search-pattern hiding SSE scheme. However, $\mathcal{I}_e$ is much larger than $\mathcal{I}_0$ because $2^{\ell} \gg |\mathcal{W}|$ in general.

Manuscript received March 20, 2018.
Manuscript revised June 19, 2018.
[†]The author is with Tokyo Institute of Technology, Tokyo, 152-8552 Japan.
[††]The author is with Ibaraki University, Hitachi-shi, 316-8511 Japan.
*A part of this paper was published at Financial Cryptography and Data Security 2017 [31].
a) E-mail: ogata.w.aa@m.titech.ac.jp
b) E-mail: kaoru.kurosawa.kk@vc.ibaraki.ac.jp
DOI: 10.1587/transfun.E102.A.114

[†]The connection between SSE and PIR was suggested by Curtmola et al. [16], [17].

1.3 No-Dictionary Verifiable SSE

Consider a verifiable SSE scheme such as follows. The client stores $\mathcal{I}_1 = ((a_1, b_1, c_1), \ldots, (a_m, b_m, c_m))$ to the server such that

$$(a_i, b_i, c_i) = (F_{k_1}(w_i), F_{k_2}(w_i) + \mathcal{ID}(w_i), MAC(a_i, b_i))$$

for each $w_i \in \mathcal{W}$, where F is a pseudorandom function and k_1, k_2 are keys. To search on w, the client sends

$$(a', b') = (F_{k_1}(w), F_{k_2}(w))$$

to the server. The server finds i such that $a' = a_i$ and returns the search result with $MAC(a_i, b_i)$.

Is it a *no-dictionary* verifiable SSE scheme? The answer is no because a malicious server can cheat by saying that $a' \notin \{a_1, \ldots, a_m\}$ (namely $w \notin \mathcal{W}$) even if $a' \in \{a_1, \ldots, a_m\}$. The client has no way to check this.

We can prevent this cheating by using the extended index table $\mathcal{T}_e$ defined in Sect. 1.2. However, the encrypted $\mathcal{I}_e$ gets much larger than $\mathcal{I}_1$ (see Sect. 1.2).

For this problem, Taketani and Ogata [35] showed a *no-dictionary* verifiable SSE scheme such that the encrypted index table is almost the same size as $\mathcal{I}_1$. In this scheme, however, the server takes $O(N \log(Nm))$ time to prove that $w \notin \mathcal{W}$, where $N = |\mathcal{D}|$ and $m = |\mathcal{W}|$.

1.4 Our Contribution

In this paper, we first show a generic method to transform any passively secure SSE scheme to a *no-dictionary* verifiable SSE scheme. In the transformed scheme, the encrypted index table is only a few times larger than that of the underlying SSE scheme, and the server takes only $O(1)$ time to prove that $w \notin \mathcal{W}$, which is more efficient than the scheme in [35]. The search time for $w \in \mathcal{W}$ remains almost the same as that of the original SSE scheme. We also prove that the transformed scheme is UC-secure in Appendix similarly to [24], [26].

We next present a no-dictionary search-pattern hiding SSE scheme such that the encrypted index table is only a few times larger than $\mathcal{I}_0$ (As in the corresponding dictionary SSE scheme, the cost for it is that the communication complexity and the computation complexity increase.)[†].

We use Cuckoo Hashing [33] in both our results as a main technical tool.

1.5 Remark

In the verifiable SSE schemes of [24]–[27], the set of keywords is defined as $\mathcal{W} = \{0, 1\}^\ell$. In reality, however, keywords have various length. Therefore we must use the technique of Sect. 1.2 in practice.

[†]This part was not written in the conference version [31] of this paper.

If we use an oblivious RAM (ORAM) in a dynamic SSE scheme [18] (in which the client can update files), we can hide the search pattern and the access pattern. In such a scheme, however, the client must keep the dictionary (or a corresponding list). The communication cost is also large.

2. Verifiable Searchable Symmetric Encryption

In this section, we define a no-dictionary (verifiable) SSE scheme and its security. Basically, we follow the notation used in [12], [24], [26].

- Let $\mathcal{D} = \{D_1, \ldots, D_N\}$ be the set of files.
- Let $\mathcal{W}$ be the set of keywords, where each keyword w is contained in some file(s).
- For $w \in \{0, 1\}^*$, define as follows:

$$\mathcal{D}(w) = \begin{cases} \{D_i \mid D_i \text{ contains } w\} & \text{if } w \in \mathcal{W} \\ \emptyset & \text{otherwise} \end{cases}$$

- Let $C = \{C_1, \ldots, C_N\}$, where C_i is a ciphertext of D_i.
- Let

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

Note that $C(w) = \emptyset$ if $w \notin \mathcal{W}$.

If X is a bit string, $|X|$ denotes the bit length of X. If X is a set, $|X|$ denotes the cardinality of X. "PPT" refers to probabilistic polynomial time, and "PT" refers to polynomial time.

2.1 Model

An SSE scheme has two phases, the store phase (which is executed only once) and the search phase (which is executed a polynomial number of times). In the store phase, the client encrypts all files in $\mathcal{D}$ and stores them on the server. In the search phase, the client sends a ciphertext of a word w, and the server returns $C(w)$. If there is a mechanism to verify the validity of $C(w)$, the scheme is called a verifiable SSE (vSSE).

Formally, a vSSE scheme consists of the following four polynomial-time algorithms $\texttt{vSSE} = (\texttt{Setup}, \texttt{Trpdr}, \texttt{Search}, \texttt{Dec})$ as follows:

- $(K, \mathcal{I}, C) \leftarrow \texttt{Setup}(1^\lambda, \mathcal{D}, \mathcal{W}, \{(w, \mathcal{D}(w)) \mid w \in \mathcal{W}\})$: a PPT algorithm that generates a key K, an encrypted index $\mathcal{I}$, and the set of encrypted files $C = \{C_1, \ldots, C_N\}$, where λ is a security parameter.
This algorithm is run by the client in the store phase. He then stores $(\mathcal{I}, C)$ on the server.
- $t(w) \leftarrow \texttt{Trpdr}(K, w)$: a PPT algorithm that outputs a trapdoor $t(w)$ for $w \in \{0, 1\}^*$.
This algorithm is run by the client in the search phase. $t(w)$ is sent to the server.
- $(C^*, \mathsf{Proof}) \leftarrow \texttt{Search}(\mathcal{I}, C, t(w))$: a PT algorithm that outputs the search result C^* and Proof for the validity check.

This algorithm is run by the server in the search phase. She then returns (C^*, Proof) to the client.

- $\mathcal{D}^*/\perp \leftarrow \mathtt{Dec}(K, t(w), C^*, \mathsf{Proof})$: a PT algorithm that decrypts C^* and verifies its validity based on Proof. If not valid, output is $\perp$. This algorithm is run by the client in the search phase.

We say that a vSSE satisfies correctness if the following holds for any $K, \mathcal{D}, \mathcal{W}, \{(w, \mathcal{D}(w)) \mid w \in \mathcal{W}\}$ and any word $w \in \{0,1\}^*$.

- If

$$\begin{aligned}(K, \mathcal{I}, \mathcal{C}) &\leftarrow \mathtt{Setup}(1^\lambda, \mathcal{D}, \mathcal{W},\\ &\qquad \{(w, \mathcal{D}(w)) \mid w \in \mathcal{W}\}),\\ t(w) &\leftarrow \mathtt{Trpdr}(K, w),\\ (C^*, \mathsf{Proof}) &\leftarrow \mathtt{Search}(\mathcal{I}, \mathcal{C}, t(w)),\\ \mathcal{D}^* &\leftarrow \mathtt{Dec}(K, t(w), C^*, \mathsf{Proof}),\end{aligned}$$

then

$$\mathcal{D}^* = \mathcal{D}(w).$$

We assume that C^* is equal to $C(w) (\subset \mathcal{C})$ as in most existing schemes.

An (not verifiable) SSE scheme is defined by omitting Proof.

2.2 Security Definition

We next define the security of vSSE schemes. Note that a search word w does not need to belong to the set $\mathcal{W}$.

Privacy. In a (v)SSE, the server should learn almost no information on $\mathcal{D}, \mathcal{W}$, and the search word w. Let $L_1(\mathcal{D}, \mathcal{W})$ denote the information that the server can learn in the store phase, and let $L_2(\mathcal{D}, \mathcal{W}, \mathbf{w}, w)$ denote that in the search phase, where w is the current search word and $\mathbf{w} = (w_1, w_2, \ldots)$ is the list of the past search words queried so far.

In most existing SSE schemes, $L_1(\mathcal{D}, \mathcal{W}) = (|D_1|, \ldots, |D_N|, |\mathcal{W}|)$, and $L_2(\mathcal{D}, \mathcal{W}, \mathbf{w}, w)$ consists of $\{j \mid D_j \in \mathcal{D}(w)\}$ and the search pattern

$$\mathtt{SPattern}((w_1, \ldots, w_{q-1}), w) = (sp_1, \ldots, sp_{q-1}),$$

where

$$sp_j = \begin{cases} 1 & \text{if } w_j = w,\\ 0 & \text{if } w_j \neq w.\end{cases}$$

The search pattern reveals which past queries are the same as w.

Let $L = (L_1, L_2)$. The client's privacy is defined by using two games: a real game $\mathbf{Game}_{real}$ and a simulation game $\mathbf{Game}^L_{sim}$, as shown in Figs. 1 and 2, respectively. $\mathbf{Game}_{real}$ is played by a challenger $\mathbf{C}$ and an adversary $\mathbf{A}$, and $\mathbf{Game}^L_{sim}$ 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

$$\begin{aligned}&|\Pr[\mathbf{A} \text{ outputs } b = 1 \text{ in } \mathbf{Game}_{real}]\\ &- \Pr[\mathbf{A} \text{ outputs } b = 1 \text{ in } \mathbf{Game}^L_{sim}]| \end{aligned} \tag{2}$$

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

1. Adversary $\mathbf{A}$ chooses $(\mathcal{D}, \mathcal{W})$ and sends them to challenger $\mathbf{C}$.
2. $\mathbf{C}$ generates $(K, \mathcal{I}, \mathcal{C}) \leftarrow \mathtt{Setup}(1^\lambda, \mathcal{D}, \mathcal{W}, \{(w, \mathcal{D}(w)) \mid w \in \mathcal{W}\})$ and sends $(\mathcal{I}, \mathcal{C})$ to $\mathbf{A}$.
3. For $i = 1, \ldots, q$, do:
   a. $\mathbf{A}$ chooses a word $w_i \in \{0,1\}^*$ and sends it to $\mathbf{C}$.
   b. $\mathbf{C}$ sends the trapdoor $t(w_i) \leftarrow \mathtt{Trpdr}(K, w_i)$ back to $\mathbf{A}$.
4. $\mathbf{A}$ outputs bit b.

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

1. Adversary $\mathbf{A}$ chooses $(\mathcal{D}, \mathcal{W})$ and sends them to challenger $\mathbf{C}$.
2. $\mathbf{C}$ sends $L_1(\mathcal{D}, \mathcal{W})$ to simulator $\mathbf{S}$.
3. $\mathbf{S}$ computes $(\mathcal{I}, \mathcal{C})$ from $L_1(\mathcal{D}, \mathcal{W})$, and sends them to $\mathbf{C}$.
4. $\mathbf{C}$ relays $(\mathcal{I}, \mathcal{C})$ to $\mathbf{A}$.
5. For $i = 1, \ldots, q$, do:
   a. $\mathbf{A}$ chooses $w_i \in \{0,1\}^*$ and sends it to $\mathbf{C}$.
   b. $\mathbf{C}$ sends $L_2(\mathcal{D}, \mathcal{W}, \mathbf{w}, w_i)$ to $\mathbf{S}$, where $\mathbf{w} = (w_1, \ldots, w_{i-1})$.
   c. $\mathbf{S}$ computes $t(w_i)$ from $L_2(\mathcal{D}, \mathcal{W}, \mathbf{w}, w_i)$ and sends it to $\mathbf{C}$.
   d. $\mathbf{C}$ relays $t(w_i)$ to $\mathbf{A}$.
6. $\mathbf{A}$ outputs bit b.

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

Reliability. In an SSE scheme, a malicious server might cheat a client by returning a false result $\tilde{C}^* (\neq C(w))$ during the search phase. (Weak) reliability guarantees that the client can detect such a malicious behavior. Formally, reliability is defined by game $\mathbf{Game}_{reli}$ shown in Fig. 3, which is played by an adversary $\mathbf{B} = (\mathbf{B}_1, \mathbf{B}_2)$ (malicious server) and a challenger $\mathbf{C}$. $\mathbf{B}_1$ and $\mathbf{B}_2$ are assumed to be able to communicate freely.

Definition 2 (Reliability): We say that $\mathbf{B}$ wins in $\mathbf{Game}_{reli}$ if $\mathbf{B}_1$ receives $\mathcal{D}^*_i$ such that $\tilde{\mathcal{D}}^*_i \notin \{\mathcal{D}(w_i), \perp\}$ for some i. We say that a vSSE scheme satisfies reliability if for any PPT adversary $\mathbf{B}$,

$$\Pr[\mathbf{B} \text{ wins in } \mathbf{Game}_{reli}]$$

is negligible.

For SSE schemes in which $C^* = C(w)$ is assumed to be returned as a search result, strong reliability was also defined in [26]. In strong reliability, the server has to answer a wrong pair $(\tilde{C}^*, \widetilde{\mathsf{Proof}}) (\neq (C(w), \mathsf{Proof}))$ that will be accepted in the search phase to win the game.

(Store phase)

1. $\mathbf{B}_1$ chooses $(\mathcal{D}, \mathcal{W})$ and sends them to $\mathbf{C}$.
2. $\mathbf{C}$ generates $(K, \mathcal{I}, C) \leftarrow \mathtt{Setup}(1^\lambda, \mathcal{D}, \mathcal{W}, \{(w, \mathcal{D}(w)) \mid w \in \mathcal{W}\})$, and sends $(\mathcal{I}, C)$ to $\mathbf{B}_2$.

(Search phase) For $i = 1, \ldots, q$, do

1. $\mathbf{B}_1$ chooses $w_i \in \{0,1\}^*$ and sends it to $\mathbf{C}$.
2. $\mathbf{C}$ sends the trapdoor $t(w_i) \leftarrow \mathtt{Trpdr}(K, w_i)$ to $\mathbf{B}_2$.
3. $\mathbf{B}_2$ returns $(\tilde{C}_i^*, \widetilde{\mathsf{Proof}_i})$ to $\mathbf{C}$.
4. $\mathbf{C}$ computes

$$\tilde{\mathcal{D}}_i^* \leftarrow \mathtt{Dec}(K, t(w_i), \tilde{C}_i^*, \widetilde{\mathsf{Proof}_i})$$

and returns $\tilde{\mathcal{D}}_i^*$ to $\mathbf{B}_1$. $\tilde{\mathcal{D}}_i^*$ can be $\perp$.

Fig. 3 **Game**$_{reli}$.

Definition 3 (Strong Reliability): We say that $\mathbf{B}$ strongly wins in **Game**$_{reli}$ if there exists i, such that both $\mathtt{Dec}(K, t(w_i), \tilde{C}_i^*, \widetilde{\mathsf{Proof}_i}) \neq \perp$ and $(\tilde{C}_i^*, \widetilde{\mathsf{Proof}_i}) \neq (C(w_i), \mathsf{Proof}_i)$ hold. We say that a vSSE scheme satisfies strong reliability if for any PPT adversary $\mathbf{B}$,

$$\Pr[\mathbf{B} \text{ strongly wins in } \mathbf{Game}_{reli}]$$

is negligible.

3. Building Blocks

3.1 Cuckoo Hashing

Cuckoo Hashing [33] is a hashing algorithm with the advantage that the search time is constant. To store n keys, it uses two tables T_1 and T_2 of size m, and two independent random hash functions h_1 and h_2 with the range $\{1, \ldots, m\}$. Every key x is stored at one of two positions, $T_1(h_1(x))$ or $T_2(h_2(x))$. So we need to inspect at most two positions to search x.

It can happen that both possible places $T_1(h_1(x))$ and $T_2(h_2(x))$ of a given key x are already occupied. This problem is solved by allowing x to throw out the key (say y) occupying the position $T_1(h_1(x))$. Next, we insert y at its alternative position $T_2(h_2(y))$. If it is already occupied, we repeat the above steps until we find an empty position. If we failed after some number of trials, we choose new hash functions and rebuild the data structure.

Let $n = m(1-\epsilon)$ for some $\epsilon \in (0,1)$. Then the above algorithm succeeds with probability $1 - c(\epsilon)/m + O(1/m^2)$ for some explicit function $c(\cdot)$ [28]. The expected construction time of (T_1, T_2) is bounded above by [28]

$$2n \frac{1 - e^{\epsilon - 1}}{(1 - e^{\epsilon-1}) + \epsilon}. \tag{3}$$

3.2 Pseudo-Random Function

Let $\mathcal{R}$ be a family of all functions $f : \{0,1\}^* \to \{0,1\}^n$. We say that $F : \{0,1\}^\ell \times \{0,1\}^* \to \{0,1\}^n$ is a pseudo-random function if for any PPT distinguisher $\mathbf{D}$,

$$\left| \Pr[k \xleftarrow{\$} \{0,1\}^\ell : \mathbf{D}^{F(k,\cdot)} = 1] - \Pr[f \xleftarrow{\$} \mathcal{R} : \mathbf{D}^{f(\cdot)} = 1] \right|$$

is negligibly small.

It is well known that a pseudo-random function works as a MAC which is existentially unforgeable against chosen message attack.

4. Generic Transformation from SSE to vSSE

In this section, we show a generic method to transform any SSE which satisfies privacy to a no-dictionary verifiable SSE. In the transformed scheme, the encrypted index table is only a few times larger than that of the underlying SSE scheme, and the server takes only $O(1)$ time to prove that $w \notin \mathcal{W}$. The search time for $w \in \mathcal{W}$ remains almost the same as that of the original SSE scheme. We also prove that the transformed scheme is UC-secure in Appendix similarly to [24], [26].

4.1 Construction

Let $\mathtt{SSE}_0 = (\mathtt{Setup}_0, \mathtt{Trpdr}_0, \mathtt{Search}_0, \mathtt{Dec}_0)$ be an SSE scheme. We construct a no-dictionary verifiable SSE $\mathtt{vSSE}_1 = (\mathtt{Setup}_1, \mathtt{Trpdr}_1, \mathtt{Search}_1, \mathtt{Dec}_1)$ as follows. Let F be a pseudo-random function.

- $\mathtt{Setup}_1(1^\lambda, \mathcal{D}, \mathcal{W}, \{(w, \mathcal{D}(w)) \mid w \in \mathcal{W}\})$: Let $\mathcal{W} = \{w_1, w_2, \ldots, w_{|\mathcal{W}|}\}$.
  1. Run $\mathtt{Setup}_0(1^\lambda, \mathcal{D}, \mathcal{W}, \{(w, \mathcal{D}(w)) \mid w \in \mathcal{W}\})$ to obtain $(K_0, \mathcal{I}_0, C)$. Note that $C_i \in C$ is a ciphertext of each file $D_i \in \mathcal{D}$.
  2. Randomly choose a key k of F. We write $F_k(x)$ instead of $F(k, x)$.
  3. Compute $key_j \leftarrow F_k(0 \| w_j)$ for all $w_j \in \mathcal{W}$.
  4. Construct cuckoo hash tables (T_1', T_2') of size $|\mathcal{W}|+1$ which store $\{key_j\}_{j=1}^{|\mathcal{W}|}$. Let (h_1, h_2) be the hash functions which are used to construct (T_1', T_2'). This means that
  $$T_1'(h_1(key_j)) = key_j \text{ or } T_2'(h_2(key_j)) = key_j$$
  for each key_j. When failing in constructing tables, go back to step 2.
  5. Construct two tables (T_1, T_2) of size $|\mathcal{W}|+1$ as follows:
  For $a = 1, 2$ and $i = 1, \ldots, |\mathcal{W}|+1$, if $T_a'(i) = key_j$ for some $key_j = F_k(0 \| w_j)$, then
  $$T_a(i) \leftarrow \langle key_j, F_k(a \| i \| key_j), F_k(3 \| key_j \| C(w_j)) \rangle.$$
  Otherwise
  $$T_a(i) \leftarrow \langle null, F_k(a \| i \| null), null \rangle.$$
  6. Output $(K = (K_0, k), \mathcal{I} = (\mathcal{I}_0, T_1, T_2, h_1, h_2), C)$.

The client sends $(\mathcal{I}, C)$ to the server, and keeps K secret.

For each $key_j = F_k(0\|w_j)$, it holds that

$$T_1(h_1(key_j)) = \langle key_j, F_k(1\|h_1(key_j)\|key_j), F_k(3\|key_j\|C(w_j))\rangle$$

or

$$T_2(h_2(key_j)) = \langle key_j, F_k(2\|h_2(key_j)\|key_j), F_k(3\|key_j\|C(w_j))\rangle.$$

- $\texttt{Trpdr}_1((K_0, k), w)$: Compute $key \leftarrow F_k(0\|w)$ and $t_0(w) \leftarrow \texttt{Trpdr}_0(K_0, w)$. Output $t(w) = (key, t_0(w))$.

The client sends $t(w)$ to the server, where w is a search word.

- $\texttt{Search}_1((\mathcal{I}_0, T_1, T_2, h_1, h_2), C, t(w) = (key, token))$: Retrieve

$$\langle\alpha_1, \beta_1, \gamma_1\rangle \leftarrow T_1(h_1(key)),$$
$$\langle\alpha_2, \beta_2, \gamma_2\rangle \leftarrow T_2(h_2(key)).$$

Let

$$C^* \leftarrow \begin{cases} \texttt{Search}_0(\mathcal{I}_0, C, token) & \text{if } key \in \{\alpha_1, \alpha_2\} \\ \emptyset & \text{otherwise} \end{cases}$$

$$\mathsf{Proof} \leftarrow \begin{cases} \gamma_1 & \text{if } key = \alpha_1 \\ \gamma_2 & \text{if } key = \alpha_2 \\ (\alpha_1, \beta_1, \alpha_2, \beta_2) & \text{otherwise} \end{cases}$$

Output (C^*, Proof).

The server returns (C^*, Proof) to the client.

- $\texttt{Dec}_1((K_0, k), t(w) = (key, token), C^*, \mathsf{Proof})$:
(Case 1) $\mathsf{Proof} = \gamma$.
If $\gamma \neq F_k(3\|key\|C^*)$, then output $\perp$.
(Case 2) $\mathsf{Proof} = (\alpha_1, \beta_1, \alpha_2, \beta_2)$.
If $C^* \neq \emptyset$ or $key \in \{\alpha_1, \alpha_2\}$ or $\beta_1 \neq F_k(1\|h_1(key)\|\alpha_1)$ or $\beta_2 \neq F_k(2\|h_2(key)\|\alpha_2)$, then output $\perp$.
Otherwise, compute $\mathcal{D}^* \leftarrow \texttt{Dec}_0(K_0, token, C^*)$ and output $\mathcal{D}^*$.

The client obtains $\perp$ or $\mathcal{D}^*$.

4.2 Example

Suppose that there are 7 keywords $\mathcal{W} = \{w_1, \ldots, w_7\}$ and 8 ciphertexts $C = \{C_1, \ldots, C_8\}$ such that $C(w_j)$ are given in Table 1. In the same table, $h_1(key_j)$ and $h_2(key_j)$ are the hash values which are used to construct the cuckoo hash tables (T_1', T_2') for the set $\{key_j = F_k(0\|w_j) \mid j = 1, \ldots, 7\}$.

Then T_1 and T_2 are constructed as shown in Table 2. Note that the size of each table is $8 = |\mathcal{W}| + 1$.

(Case 1) Suppose that a client searches for a keyword $w_3 \in \mathcal{W}$.

1. The client sends trapdoor $(key_3, t_0(w_3))$ to the server.
2. Since $h_1(key_3) = 6$ and $h_2(key_3) = 4$, the server retrieves

$$\langle\alpha_1, \beta_1, \gamma_1\rangle = T_1(6) = \langle key_3, F_k(1\|6\|key_3), F_k(3\|key_3\|C_1, C_4)\rangle,$$
$$\langle\alpha_2, \beta_2, \gamma_2\rangle = T_2(4) = \langle key_2, F_k(2\|4\|key_2), F_k(3\|key_2\|C_2)\rangle$$

from T_1 and T_2.
Because $\alpha_1 = key_3$, the server obtains the search result

$$C^* = (C_1, C_4) \leftarrow \texttt{Search}_0(\mathcal{I}_0, C, t_0(w_3)),$$
$$\mathsf{Proof} = \gamma_1 = F_k(3\|key_3\|C_1, C_4),$$

and returns (C^*, Proof) to the client.
3. The client verifies if $\gamma_1 = F_k(3\|key_3\|C^*)$.

(Case 2) Suppose that the client searches for $w \notin \mathcal{W}$.

1. The client computes $key \leftarrow F_k(0\|w)$ and $t_0(w) \leftarrow \texttt{Trpdr}_0(K_0, w)$. He sends $t(w) = (key, t_0(w))$ to the server.
2. Suppose that $h_1(key) = 5$ and $h_2(key) = 3$. Then the server retrieves

$$\langle\alpha_1, \beta_1, \gamma_1\rangle = T_1(5) = \langle null, F_k(1\|5), null\rangle,$$
$$\langle\alpha_2, \beta_2, \gamma_2\rangle = T_2(3) = \langle key_4, F_k(2\|3\|key_4), F_k(3\|key_4\|C_1, C_3, C_7)\rangle.$$

Because $key \notin \{\alpha_1, \alpha_2\}$, the server returns $C^* = \emptyset$ and $\mathsf{Proof} = (\alpha_1, \beta_1, \alpha_2, \beta_2) = (null, F_k(1\|5), key_4, F_k(2\|3\|key_4))$.

Table 1 Example.

keyword w_j	$C(w_j)$	$h_1(key_j)$	$h_2(key_j)$
w_1	C_1, C_4, C_5, C_8	6	1
w_2	C_2	2	4
w_3	C_1, C_4	6	4
w_4	C_1, C_3, C_7	6	3
w_5	C_2, C_6	7	8
w_6	C_5, C_8	7	6
w_7	C_1	2	8

Table 2 Cuckoo hash tables (T_1, T_2).

i	$T_1(i)$	i	$T_2(i)$
1	$\langle$ null , $F_k(1\|1)$, null $\rangle$	1	$\langle key_1, F_k(2\|1\|key_1), F_k(3\|key_1\|C_1, C_4, C_5, C_8)\rangle$
2	$\langle key_7, F_k(1\|2\|key_7), F_k(3\|key_7\|C_1)\rangle$	2	$\langle$ null , $F_k(2\|2)$, null $\rangle$
3	$\langle$ null , $F_k(1\|3)$, null $\rangle$	3	$\langle key_4, F_k(2\|3\|key_4), F_k(3\|key_4\|C_1, C_3, C_7)\rangle$
4	$\langle$ null , $F_k(1\|4)$, null $\rangle$	4	$\langle key_2, F_k(2\|4\|key_2), F_k(3\|key_2\|C_2)\rangle$
5	$\langle$ null , $F_k(1\|5)$, null $\rangle$	5	$\langle$ null , $F_k(2\|5)$, null $\rangle$
6	$\langle key_3, F_k(1\|6\|key_3), F_k(3\|key_3\|C_1, C_4)\rangle$	6	$\langle$ null , $F_k(2\|6)$, null $\rangle$
7	$\langle key_6, F_k(1\|7\|key_6), F_k(3\|key_6\|C_5, C_8)\rangle$	7	$\langle$ null , $F_k(2\|7)$, null $\rangle$
8	$\langle$ null , $F_k(1\|8)$, null $\rangle$	8	$\langle key_5, F_k(2\|8\|key_5), F_k(3\|key_5\|(C_2, C_6))\rangle$

3. The client verifies if $key \notin \{\alpha_1, \alpha_2\}$, $\beta_1 = F_k(1\|h_1(key)\|\alpha_1)$, and $\beta_2 = F_k(2\|h_2(key)\|\alpha_2)$.

4.3 Efficiency

The efficiency of our transformed scheme $\mathtt{vSSE}_1$ is estimated as follows:

- In the store phase, $|\mathcal{W}|$ keys are stored in two tables, where each table has size $m = |\mathcal{W}| + 1$. Therefore, the client takes the expected time $O(|\mathcal{W}|) + time(\mathtt{Setup}_0)$ to run $\mathtt{Setup}_1$ from Eq. (3).
- In the search phase, the search time for $w \in \mathcal{W}$ is almost the same as that of the original scheme.
- The server takes only $O(1)$ time to prove that $w \notin \mathcal{W}$ because the search time is constant in cuckoo hashing.

To prove that $w \notin \mathcal{W}$, in the method of [35], the server takes $O(N \log N|\mathcal{W}|)$ time. In the concrete method (Algorithm 1+2) in [6], it takes $O(\log|\mathcal{W}|) + time(\mathtt{Search}_0)$.

4.4 Security

Theorem 1: If the underlying scheme $\mathtt{SSE}_0$ has $L = (L_1, L_2)$-privacy and F is a pseudorandom function, then our scheme $\mathtt{vSSE}_1$ has $L' = (L'_1, L'_2)$-privacy such that

$$\begin{aligned} L'_1(\mathcal{D}, \mathcal{W}) &= L_1(\mathcal{D}, \mathcal{W}) \cup \{|\mathcal{W}|\}, \\ L'_2(\mathcal{D}, \mathcal{W}, \mathbf{w}, w_i) &= L_2(\mathcal{D}, \mathcal{W}, \mathbf{w}, w_i) \\ &\quad \cup \{\mathtt{SPattern}(\mathbf{w}, w_i), [w_i \in \mathcal{W}]\}. \quad (4) \end{aligned}$$

In the all existing SSE schemes, $|\mathcal{W}| \in L_1(\mathcal{D}, \mathcal{W})$ and $\{\mathtt{SPattern}(\mathbf{w}, w_i), [w_i \in \mathcal{W}]\} \subseteq L_2(\mathcal{D}, \mathcal{W}, \mathbf{w}, w_i)$. (There may be some exceptions which use oblivious RAM. But such SSE schemes are inefficient.) So, the client's privacy in our vSSE scheme has the same level as that of the underlying SSE scheme.

(Proof) Let $\mathbf{S}_0$ be a simulator of the underlining SSE scheme which has (L_1, L_2)-privacy. We construct a simulator $\mathbf{S}$ of our vSSE scheme which achieves (L'_1, L'_2)-privacy as follows.

(Store phase) In $\mathbf{Game}_{sim}$, $\mathbf{S}$ takes $L'_1(\mathcal{D}, \mathcal{W}) = L_1(\mathcal{D}, \mathcal{W}) \cup \{|\mathcal{W}|\}$ as an input. $\mathbf{S}$ runs $\mathbf{S}_0(L_1(\mathcal{D}, \mathcal{W}))$ and gets its output $(\mathcal{I}_0, C)$. Next $\mathbf{S}$ constructs T_1 and T_2 as follows. Note that the size of each T_1, T_2 is $m = |\mathcal{W}| + 1$.

- Choose $key'_1, \ldots, key'_{|\mathcal{W}|}$ randomly, where key'_i is the simulated value of $key_j = F_K(0\|w_j)$ such that $\{key'_1, \ldots, key'_{|\mathcal{W}|}\} = \{key_1, \ldots, key_{|\mathcal{W}|}\}$.
- Construct the cuckoo hash tables (T'_1, T'_2) which store $(key'_{\pi(1)}, \ldots, key'_{\pi(|\mathcal{W}|)})$, where π is a random permutation. Let h_1, h_2 be the two hash functions which are used to construct (T'_1, T'_2).
- For $a = 1, 2$ and $i = 1, \ldots, |\mathcal{W}| + 1$, if $T'_a(i) = key'_j$ for some j, then choose two random strings r and r', and $T_a(i) \leftarrow \langle key'_j, r, r'\rangle$. Otherwise, choose a random string r and $T_a(i) \leftarrow \langle null, r, null\rangle$.

$\mathbf{S}$ sends $(\mathcal{I}_0, T_1, T_2, h_1, h_2)$ and C to the challenger. Let $cntr \leftarrow 1$, where $cntr$ will denote the number of distinct keywords which the client has queried.

(Search phase) In the ith search phase, $\mathbf{S}$ takes $L'_2(\mathcal{D}, \mathcal{W}, \mathbf{w}, w^*) = L_2(\mathcal{D}, \mathcal{W}, \mathbf{w}, w^*) \cup \{\mathtt{SPattern}(\mathbf{w}, w^*), [w^* \in \mathcal{W}]\}$ as an input. $\mathbf{S}$ first obtains $t_0(w^*)$ by running $\mathbf{S}_0(L_2(\mathcal{D}, \mathcal{W}, \mathbf{w}, w^*))$, and sets

$$key^*_i \leftarrow \begin{cases} key'_{cntr} & \text{if } sp_j = 0 \text{ for all } j \text{ and } w^* \in \mathcal{W}, \\ key^*_j & \text{if } sp_j = 1 \text{ for some } j, \\ \text{random} & \text{otherwise.} \end{cases}$$

$$cntr \leftarrow \begin{cases} cntr + 1 & \text{if } sp_j = 0 \text{ for all } j \text{ and } w^* \in \mathcal{W}, \\ cntr & \text{otherwise.} \end{cases}$$

$\mathbf{S}$ outputs $(key^*_i, t_0(w^*))$ as a simulated trapdoor.

We will prove that there is no adversary $\mathbf{A}$ who can efficiently distinguish between $\mathbf{Game}_{real}$ and $\mathbf{Game}_{sim}$. We consider a game sequence $(\mathbf{Game}_{real}, \mathbf{Game}_{mid}, \mathbf{Game}_{sim})$. $\mathbf{Game}_{mid}$ is the same as $\mathbf{Game}_{real}$ except that all values of $F_k(\cdot)$ are replaced with random strings. For $i \in \{real, mid, sim\}$, define

$$P_i = \Pr[\ \mathbf{A} \text{ outputs } b = 1 \text{ in } \mathbf{Game}_i].$$

Then $|P_{real} - P_{mid}|$ is negligible because F is a pseudorandom function. We can also see that $|P_{mid} - P_{sim}|$ is negligible from the (L_1, L_2)-privacy of the underlying SSE scheme. Consequently, $|P_{real} - P_{sim}|$ is negligible. □

Theorem 2: Our vSSE scheme $\mathtt{vSSE}_1$ satisfies strong reliability if F is a pseudorandom function.

(Proof) We look at the pseudorandom function F as a MAC.

Suppose that there exists an adversary $\mathbf{B} = (\mathbf{B}_1, \mathbf{B}_2)$ who can break the strong reliability of our vSSE scheme, and $\mathbf{B}$ runs the search phase q times. Let $(\tilde{C}^*_i, \widetilde{\mathsf{Proof}_i})$ be $\mathbf{B}_2$'s response to $t(w_i) = (key_i, t_0(w_i))$ in the ith search phase, and let

$$(C(w_i), \mathsf{Proof}_i) = \mathtt{Search}_1(\mathcal{I}, C, t(w_i)).$$

From the definition, $\mathbf{B}$ strongly wins if there exists $i \in \{1, \ldots, q\}$ such that

$$\begin{aligned} &(\tilde{C}^*_i, \widetilde{\mathsf{Proof}_i}) \neq (C(w_i), \mathsf{Proof}_i) \quad \text{and} \\ &\mathtt{Dec}_1(K, (key_i, t_0(w_i)), \tilde{C}^*_i, \widetilde{\mathsf{Proof}_i}) \neq \perp. \end{aligned} \quad (5)$$

By using $\mathbf{B}$, we will construct a forger $\mathbf{F}$ against the MAC, where $\mathbf{F}$ has oracle access to F_k.

First, $\mathbf{F}$ randomly chooses $J \in \{1, \ldots, q\}$. Then, $\mathbf{F}$ runs $\mathbf{B}$ by playing the role of the challenger $\mathbf{C}$ (see Fig. 3) until the $(J-1)$th search phase. During this simulation, when $\mathbf{C}$ needs to compute $F_k(x)$ for some x, $\mathbf{F}$ queries x to its oracle F_k.

In the Jth search phase, there are three cases:

(1) $\widetilde{\mathsf{Proof}_J} = \tilde{\gamma}$.

In this case, $\mathbf{F}$ outputs $m' = (3\|key_J\|\tilde{C}^*_J)$ and $tag' = \tilde{\gamma}$

as a forgery of the MAC F.

(2) $\mathsf{Proof}_J = \gamma$ and $\widetilde{\mathsf{Proof}}_J = (\tilde{\alpha}_1, \tilde{\beta}_1, \tilde{\alpha}_2, \tilde{\beta}_2)$.
Since $\mathsf{Proof}_J = \gamma$, there exists $a \in \{1, 2\}$ such that $T_a(h_a(key_J)) = \langle key_J, F_k(a\|h_a(key_J)\|key_J), \ldots\rangle$. For this a, **F** outputs $m' = (a\|h_a(key_J)\|\tilde{\alpha}_a)$ and $tag' = \tilde{\beta}_a$ as a forgery.

(3) $\mathsf{Proof}_J = (\alpha_1, \beta_1, \alpha_2, \beta_2)$ and $\widetilde{\mathsf{Proof}}_J = (\tilde{\alpha}_1, \tilde{\beta}_1, \tilde{\alpha}_2, \tilde{\beta}_2)$.
If there exists $a \in \{1, 2\}$ s.t. $(\alpha_a, \beta_a) \neq (\tilde{\alpha}_a, \tilde{\beta}_a)$, then, **F** outputs $m' = (a\|h_a(key_J)\|\tilde{\alpha}_a)$ and $tag' = \tilde{\beta}_a$ as a forgery. Otherwise **F** outputs "fail."

Now **F** succeeds in forgery if **B** strongly wins and **F** correctly predicts i which satisfies Eq. (5), i.e., Eq. (5) holds in $i = J$. Since **F** predicts J correctly with probability $1/q$, we obtain that

$$\Pr[\mathbf{F} \text{ succeeds in forgery}] \geq \Pr[\mathbf{B} \text{ strongly wins in } \mathbf{Game}_{reli}] \times \frac{1}{q}.$$

□

We prove the UC-security of $\mathtt{vSSE}_1$ in Appendix.

5. Search-Pattern Hiding

As mentioned before, the existing no-dictionary SSE schemes leak search pattern. Namely, they have (L_1, L_2)-privacy (Def. 1) such that L_2 includes search pattern.

In this section, we show a no-dictionary search-pattern hiding SSE scheme such that the encrypted index table is only a few times larger than $\mathcal{I}_0$ which is defined in Sect. 1.2.

We consider a model such that the search phase consists of two subprotocols. In the first subprotocol, the client obtains

$$\mathcal{ID}(w) = \{i \mid D_i \text{ contains } w \text{ as a keyword}\}$$

for the search word w. In the second subprotocol, he obtains

$$C(w) = \{C_i \mid i \in \mathcal{ID}(w)\}.$$

We focus on the first subprotocol, in which the search pattern should be hidden. The definition of privacy is the same as Def. 1.

If we use PIR in the second subprotocol in addition, we can hide even the access pattern.

5.1 PIR

PIR is a two party protocol between a sender and a receiver such as follows. The sender has a database $\mathcal{M} = (m_1, \ldots, m_N)$. The receiver wants to obtain m_{idx} without revealing the index idx. A trivial solution is that the sender sends the entire $\mathcal{M}$ to the receiver. In PIR, this must be realized with less amount of communication. There exists a PIR scheme such that the communication overhead is $O((\log N)^2)$ [29], [32].

A PIR scheme consists of four algorithms $(\mathtt{Gen}_{PIR}, \mathtt{Query}_{PIR}, \mathtt{Ans}_{PIR}, \mathtt{Dec}_{PIR})$, where the first two are PPT algorithms and the last two are PT algorithms.

- $(pk, sk) \leftarrow \mathtt{Gen}_{PIR}(1^\lambda)$: The receiver runs this algorithm, and sends pk to the sender. He keeps sk secret.
- $Q^{idx} \leftarrow \mathtt{Query}_{PIR}(sk, idx)$: The receiver runs this algorithm when he wants to obtain m_{idx}, and sends Q^{idx} to the sender.
- $rsp \leftarrow \mathtt{Ans}_{PIR}(pk, \mathcal{M}, Q^{idx})$: The sender runs this algorithm, and sends rsp back to the receiver.
- $res \leftarrow \mathtt{Dec}_{PIR}(sk, rsp)$: The receiver runs this algorithm, and obtains $res = m_{idx}$.

The sender should learn no information on idx from (pk, Q^{idx}).

More formally, a PIR scheme has to satisfy the following property; For any idx and idx', (pk, Q^{idx}) and $(pk, Q^{idx'})$ are computationally indistinguishable.

5.2 No-Dictionary Search-Pattern Hiding

We show our no-dictionary SSE scheme, $\mathtt{SSE}_2$, which can hide even the search pattern. For each $w_j \in \mathcal{W}$, let $\mathcal{ID}(w_j) = \{id_1, \ldots, id_{k_j}\}$.

$\mathtt{SSE}_2 = (\mathtt{Setup}_2, \mathtt{Trpdr}_2, \mathtt{Search}_2, \mathtt{Dec}_2)$

- $\mathtt{Setup}_2$:

1. Generate two PIR key pairs $(sk_1, pk_1), (sk_2, pk_2)$.
2. Choose a key K' of a symmetric encryption scheme $(\mathtt{Enc}, \mathtt{Dec})$ randomly.
3. For each $D_i \in \mathcal{D}$, compute $C_i \leftarrow \mathtt{Enc}_{K'}(D_i)$ and set $C = (C_1, \ldots, C_N)$.
4. Compute $\mathcal{ID}'(w_j) \leftarrow \mathtt{Enc}_{K'}(id_1\|\cdots\|id_{k_j}\|00\cdots00)$ for all $w_j \in \mathcal{W}$, where 0s are padded so that $|\mathcal{ID}'(w_1)| = |\mathcal{ID}'(w_2)| = \cdots = |\mathcal{ID}'(w_{|\mathcal{W}|})|$.
5. Choose a key k of pseudo-random function F randomly, and compute $key_j \leftarrow F_k(w_j)$ for all $w_j \in \mathcal{W}$.
6. Construct cuckoo hash tables (T_1, T_2) that stores $\langle key_j, \mathcal{ID}'(w_j)\rangle$. Note that
$$T_1(h_1(key_j)) = \langle key_j, \mathcal{ID}'(w_j)\rangle$$
or
$$T_2(h_2(key_j)) = \langle key_j, \mathcal{ID}'(w_j)\rangle$$
holds.
7. Output $((K', sk_1, sk_2, k), (T_1, T_2, pk_1, pk_2), C)$.

The client sends (T_1, T_2, pk_1, pk_2) and C to the server, and keeps (K', sk_1, sk_2, k) secret.

- $\mathtt{Trpdr}_2$:

1. Compute $key \leftarrow F_k(w)$.
2. Compute $Q_b \leftarrow \mathtt{Query}_{PIR}(sk_b, h_b(key))$ for $b = 1, 2$.
3. Output $t(w) = (Q_1, Q_2)$

The client sends $t(w) = (Q_1, Q_2)$ to the server, where w is a search word.

- $\mathtt{Search}_2$:

1. Compute $rsp_b \leftarrow \mathtt{Ans}_{PIR}(pk_b, T_b, Q_b)$ for $b = 1, 2$.
2. Output (rsp_1, rsp_2).

The server returns (rsp_1, rsp_2) to the client.

- $\mathtt{Dec}_2$:

1. Compute $res_b \leftarrow \mathtt{Dec}_{PIR}(sk_b, rsp_b)$ for $b = 1, 2$.
2. If $res_1 = \langle F_k(w), \mathcal{ID}'_1\rangle$, then decrypt $\mathcal{ID}'_1$ and obtain $\mathcal{ID}(w)$.
3. If $res_2 = \langle F_k(w), \mathcal{ID}'_2\rangle$, then decrypt $\mathcal{ID}'_2$ and obtain $\mathcal{ID}(w)$.
4. Otherwise output $\mathcal{ID}(w) = \emptyset$, which means that "$w \notin \mathcal{W}$."

The client obtains $\mathcal{ID}(w)$ even if $w \notin \mathcal{W}$.

If $w = w_j$, the trapdoor $t(w) = (Q_1, Q_2)$ is a pair of queries to retrieve $T_1(h_1(key_j))$ and $T_2(h_2(key_j))$. Therefore, either of res_1 and res_2 is equal to $\langle key_j, \mathcal{ID}'(w_j)\rangle$ from the property of cuckoo hashing and PIR.

We can use arbitrary encoding methods to represent $\mathcal{ID}(w)$. For example, $\mathcal{ID}(w) = \{2, 4, 5\}$ can be encrypted as $\mathcal{ID}'(w) = \mathtt{Enc}_{K'}(010110\cdots)$. In this case, padding is unnecessary because the length of plaintext is constant. This encoding is more efficient when hit rate is relatively large.

The following theorem shows that $\mathtt{vSSE}_2$ does not leak the search pattern.

Theorem 3: Define

$$L''_1(\mathcal{D}, \mathcal{W}) = (|\mathcal{W}|, |D_1|, \ldots, |D_N|, L_{max}),$$
$$L''_2(\mathcal{D}, \mathcal{W}, \mathbf{w}, w_i) = (),$$

where

$$L_{max} = \max_{w_i \in \mathcal{W}} |\mathcal{ID}(w_i)|.$$

If

- $(\mathtt{Gen}_{PIR}, \mathtt{Query}_{PIR}, \mathtt{Ans}_{PIR}, \mathtt{Dec}_{PIR})$ is a secure PIR scheme,
- F is a pseudorandom function, and
- $(\mathtt{Enc}, \mathtt{Dec})$ is an IND-CPA secure symmetric encryption scheme,

then our scheme $\mathtt{SSE}_2$ has $L = (L''_1, L''_2)$-privacy.

(Proof) We construct a simulator $\mathbf{S}_2$ which achieves (L''_1, L''_2)-privacy as follows.

(Store phase)

On input $L''_1(\mathcal{D}, \mathcal{W}) = (|\mathcal{W}|, |D_1|, \ldots, |D_N|, L_{max})$, $\mathbf{S}_2$ computes $(T'_1, T'_2, pk'_1, pk'_2)$ and C' as follows.

1. As in $\mathtt{Setup}_2$, generate two PIR key pairs $(sk'_1, pk'_1), (sk'_2, pk'_2)$, and choose K'.
2. For each $i \in \{1, \ldots, N\}$, compute $C'_i \leftarrow \mathtt{Enc}_{K'}(0^{|D_i|})$ and set $C' = (C'_1, \ldots, C'_N)$.
3. Compute $\mathcal{ID}''_j \leftarrow \mathtt{Enc}_{K'}(0^{L_{max}})$ for all $j \in \{1, \ldots, |\mathcal{W}|\}$.
4. Choose a random string key'_j for all $j \in \{1, \ldots, |\mathcal{W}|\}$ as the simulated value of $F_k(w_j)$.
5. Construct cuckoo hash tables (T'_1, T'_2) that stores $\langle key'_j, \mathcal{ID}''_j\rangle$.

$\mathbf{S}_2$ sends $(T'_1, T'_2, pk'_1, pk'_2)$ and C' as the simulated values of (T_1, T_2, pk_1, pk_2) and C to the challenger.

(Search phase)

$\mathbf{S}_2$ outputs $t'(w) = (Q'_1, Q'_2)$, where

$$Q'_b \leftarrow \mathtt{Query}_{\mathrm{PIR}}(sk_b, 1).$$

We will prove that there is no adversary who can efficiently distinguish between $\mathbf{Game}_{real}$ and $\mathbf{Game}_{sim}$. We consider a game sequence $(\mathbf{Game}_{real}, \mathbf{Game}_1, \mathbf{Game}_2, \mathbf{Game}_{sim})$.

$\mathbf{Game}_1$ is the same as $\mathbf{Game}_{real}$ except that all queries Q_b in search phases are replaced with $Q'_b \leftarrow \mathtt{Query}_{\mathrm{PIR}}(sk_b, 1)$. From the security of PIR, $\mathbf{Game}_{real}$ and $\mathbf{Game}_1$ are indistinguishable.

$\mathbf{Game}_2$ is the same as $\mathbf{Game}_1$ except that all values of $F_k(w_j)$ are replaced with random strings key'_j as in $\mathbf{Game}_{sim}$. From the pseudorandomness of F, $\mathbf{Game}_1$ and $\mathbf{Game}_2$ are indistinguishable.

The difference between $\mathbf{Game}_2$ and $\mathbf{Game}_{sim}$ is that

- In $\mathbf{Game}_2$, $C_i = \mathtt{Enc}_{K'}(D_i)$ and $\mathcal{ID}'(w_j) = \mathtt{Enc}_{K'}(\mathcal{ID}(w_j))$, where $\mathcal{ID}(w_j)$ are padded so that $|\mathcal{ID}(w_j)| = L_{max}$.
- In $\mathbf{Game}_{sim}$, $C'_i = \mathtt{Enc}_{K'}(0^{|D_i|})$ and $\mathcal{ID}''(w_j) = \mathtt{Enc}_{K'}(0^{L_{max}})$.

Therefore, $\mathbf{Game}_1$ and $\mathbf{Game}_2$ are indistinguishable from IND-CPA security of $(\mathtt{Enc}, \mathtt{Dec})$.

Consequently, $|P_{real} - P_{mid}|$ is negligibly small. □

The above theorem shows that $\mathtt{SSE}_2$ leaks no information in the search phase. However, if a user downloads the hit files $C_i \in C(w)$ without using PIR, the server may learn some information about the search result. In such a case, total leakage becomes $L''_2(\mathcal{D}, \mathcal{W}, \mathbf{w}, w) = \mathcal{ID}(w)$.

In general, efficiency must be sacrificed to obtain search-pattern hiding with/without dictionary.

- The search process needs two round-trip communication to complete keyword search process.
- In general, PIR is built by using asymmetric technique. So, the scheme needs high computation/communication cost.

5.3 How to Add Reliability

By using the same idea as in Sect. 4, we can add the reliability to the above scheme. The client generates cuckoo hash tables (T_1, T_2) such that

$$T_1(h_1(key_j)) \\ = \langle key_j, \mathcal{ID}'(w_j), F_k(1\|h_1(key_j)\|key_j\|\mathcal{ID}'(w_j))\rangle$$

or

$$T_2(h_2(key_j)) = \langle key_j, \mathcal{ID}'(w_j), F_k(2\|h_2(key_j)\|key_j\|\mathcal{ID}'(w_j))\rangle$$

holds, where $key_j = F_k(0\|w_j)$. Then the client checks the validity of the answer from the server in the same way as in Sect. 4.

6. Conclusion

In this paper, we studied two cases in which construction of efficient no-dictionary SSE schemes is not trivial, and showed that the cuckoo hashing technique can be used to solve the problem in both cases.

First, we proposed a generic transformation from any passively secure SSE scheme to a no-dictionary verifiable SSE scheme. The efficiency of the transformed scheme is almost the same as the underlying SSE scheme.

We next presented a no-dictionary search-pattern hiding SSE scheme that has a compact encrypted index table. In addition, we showed that our no-dictionary search-pattern hiding scheme can be modified to a verifiable scheme with small cost.

References

[1] L. Ballard, S. Kamara, and F. Monrose, "Achieving efficient conjunctive keyword searches over encrypted data," 7th International Conference on Information and Communication Security (ICICS 2005), pp.414–426, 2005.
[2] N. Baric and B. Pfitzmann, "Collision-free accumulators and fail-stop signature schemes without trees," EUROCRYPT 1997, LNCS, vol.1233, pp.480–494, 1997.
[3] M. Bellare, A. Desai, E. Jokipii, and P. Rogaway, "A concrete security treatment of symmetric encryption," FOCS 1997, pp.394–403, 1997.
[4] M. Bellare, R. Guerin, and P. Rogaway, "XOR MACs: New methods for message authentication using finite pseudorandom functions," CRYPTO 1995, LNCS, vol.963, pp.15–28, 1995.
[5] S. Bellovin and W. Cheswick, "Privacy-enhanced searches using encrypted bloom filters," Technical Report 2004/022, IACR ePrint Cryptography Archive, https://eprint.iacr.org/2004/022, 2004.
[6] R. Bost, P.-A. Fouque, and D. Pointcheval, "Verifiable dynamic symmetric searchable encryption: Optimality and forward security," Technical Report 2016/62, IACR ePrint Cryptography Archive, https://eprint.iacr.org/2016/062, 2016.
[7] J.W. Byun, D.H. Lee, and J. Lim, "Efficient conjunctive keyword search on encrypted data storage system," EuroPKI, LNCS, vol.4043, pp.184–196, 2006.
[8] J. Camenisch and A. Lysyanskaya, "Dynamic accumulators and application to efficient revocation of anonymous credentials," CRYPTO 2002, LNCS, vol.2442, pp.61–76, 2002.
[9] R. Canetti, "Universally composable security: A new paradigm for cryptographic protocols," FOCS 2001, pp.136–145, 2001.
[10] R. Canetti, "Universally composable signatures, certification and authentication," Technical Report 2003/239, IACR ePrint Cryptography Archive, https://eprint.iacr.org/2003/239, 2003.
[11] Full version of [9]: Technical Report 2000/067, IACR ePrint Cryptography Archive, https://eprint.iacr.org/2000/067, last revised 16 July 2013.
[12] D. Cash, S. Jarecki, C.S. Jutla, H. Krawczyk, M. Rosu, and M. Steiner, "Highly-scalable searchable symmetric encryption with support for Boolean queries," CRYPTO 2013, Part I, LNCS, vol.8042, pp.353–373, 2013.
[13] D. Cash, J. Jaeger, S. Jarecki, C.S. Jutla, H. Krawczyk, M.-C. Rosu, and M. Steiner, "Dynamic searchable encryption in very-large databases: Data structures and implementation," Symposium on Network and Distributed Systems Security (NDSS 2014), 2014.
[14] D. Cash and S. Tessaro, "The locality of searchable symmetric encryption," EUROCRYPT 2014, LNCS, vol.8441, pp.351–368, 2014.
[15] Y. Chang, M. Mitzenmacher, "Privacy preserving keyword searches on remote encrypted data," Applied Cryptography and Network Security (ACNS 2005), LNCS, vol.3531, pp.442–455, 2005.
[16] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky, "Searchable symmetric encryption: Improved definitions and efficient constructions," ACM Conference on Computer and Communications Security 2006, pp.79–88, 2006.
[17] Full version of [16]: Technical Report 2006/210, IACR ePrint Cryptography Archive, https://eprint.iacr.org/2006/210, 2006.
[18] S. Garg, P. Mohassel, and C. Papamanthou, "TWORAM: Efficient oblivious RAM in two rounds with applications to searchable encryption," CRYPTO 2016, Part III, LNCS, vol.9816, pp.563–592, 2016.
[19] E.-J. Goh, "Secure indexes," Technical Report 2003/216, IACR ePrint Cryptography Archive, https://eprint.iacr.org/2003/216, 2003.
[20] P. Golle, J. Staddon, B.R. Waters, "Secure Conjunctive Keyword Search over Encrypted Data," Applied Cryptography and Network Security (ACNS 2004), LNCS, vol.3089, pp.31–45, 2004.
[21] S. Kamara and C. Papamanthou, "Parallel and dynamic searchable symmetric encryption," Financial Cryptography and Data Security (FC 2013), LNCS, vol.7859, pp.258–274, 2013.
[22] S. Kamara, C. Papamanthou, and T. Roeder, "Dynamic searchable symmetric encryption," ACM Conference on Computer and Communications Security 2012, pp.965–976, 2012.
[23] K. Kurosawa, "Garbled searchable symmetric encryption," Financial Cryptography and Data Security (FC 2014), LNCS, vol.8437, pp.234–251, 2014.
[24] K. Kurosawa and Y. Ohtaki, "UC-secure searchable symmetric encryption," Financial Cryptography and Data Security (FC 2012), LNCS, vol.8437, pp.285–298, 2012.
[25] K. Kurosawa and Y. Ohtaki, "How to update documents verifiably in searchable symmetric encryption," Cryptology and Network Security (CANS 2013), LNCS, vol.8257, pp.309–328, 2013.
[26] The final version of [24]. Technical Report 2015/251, IACR ePrint Cryptography Archive, https://eprint.iacr.org/2015/251, 2015.
[27] K. Kurosawa, K. Sasaki, K. Ohta, and K. Yoneyama, "UC-secure dynamic searchable symmetric encryption scheme," Advances in Information and Computer Security (IWSEC 2016), LNCS, vol.9836, pp.73–90, 2016.
[28] R. Kutzelnigg, "Bipartite random graphs and cuckoo hashing," Fourth Colloquium on Mathematics and Computer Science Algorithms, Trees, Combinatorics and Probabilities, DMTCS Proceedings, pp.403–406, 2006.
[29] H. Lipmaa, "An oblivious transfer protocol with log-squared communication," Information Security (ISC 2005), LNCS, vol.3650, pp.314–328, 2005.
[30] M. Naveed, M. Prabhakaran, and C. Gunter, "Dynamic searchable encryption via blind storage," IEEE Symposium on Security and Privacy 2014, pp.639–654, 2014.
[31] W. Ogata and K. Kurosawa, "Efficient no-dictionary verifiable searchable symmetric encryption," Financial Cryptography and Data Security (FC 2017), LNCS, vol.10322, pp.498–516, 2017.
[32] R. Ostrovsky and W.E. Skeith, III, "A survey of single-database private information retrieval: Techniques and applications," Public Key Cryptography 2007, LNCS, vol.4450, pp.393–411, 2007.
[33] R. Pagh and F.F. Rodler, "Cuckoo hashing," J. Algorithms, vol.51, no.2, pp.122–144, 2004.
[34] D. Song, D. Wagner, and A. Perrig, "Practical techniques for searches on encrypted data," IEEE Symposium on Security and Privacy 2000, pp.44–55, 2000.
[35] S. Taketani and W. Ogata, "Improvement of UC secure searchable symmetric encryption scheme," The 10th International Workshop on

Security (IWSEC 2015), LNCS, vol.9241, pp.135–152, 2015.
[36] P. Wang, H. Wang, and J. Pieprzyk, "Keyword field-free conjunctive keyword searches on encrypted data and extension for dynamic group," Cryptology and Network Security (CANS 2008), LNCS, vol.5339, pp.178–195, 2008.

Appendix: UC-Security for No-Dictionary vSSE

If a protocol is secure in the universally composable (UC) security framework, its security is maintained even if the protocol is combined with other protocols [9]–[11]. The UC security is defined based on *ideal functionality* $\mathcal{F}$. Kurosawa and Ohtaki introduced an ideal functionality of vSSE [24], [26]. Taketani and Ogata [35] generalized it in order to handle the general leakage functions $L = (L_1, L_2)$ as shown in Fig. A·1.

In the no-dictionary verifiable SSE setting, the real world is described as follows. We assume a real adversary, $\mathbf{A}^{\mathrm{uc}}$, can control the server arbitrarily, and the client is always honest. For simplicity, we ignore session id.

In the store phase, an environment, $\mathbf{Z}$, chooses $(\mathcal{D}, \mathcal{W})$ and sends them to the client. The client computes $(K, \mathcal{I}, C) \leftarrow \mathtt{Enc}(1^\lambda, K, \mathcal{D}, \mathcal{W}, \{(w, \mathcal{D}(w)) \mid w \in \mathcal{W}\})$, and sends $(\mathcal{I}, C)$ to the server. The client stores $K^\dagger$ and the server stores $(\mathcal{I}, C)$. In the search phase, $\mathbf{Z}$ chooses a word $w \in \{0,1\}^*$ and sends it to the client. The client computes $t(w) \leftarrow \mathtt{Trpdr}(K, w)$ and sends it to the server. The server, who may be controlled by real adversary $\mathbf{A}^{\mathrm{uc}}$, returns $(\tilde{C}^*, \widetilde{\mathsf{Proof}})$ to the client. The client computes $\tilde{\mathcal{D}}(w) \leftarrow \mathtt{Dec}(K, t(w), \tilde{C}^*, \widetilde{\mathsf{Proof}})$ and sends $\tilde{\mathcal{D}}(w)$ to $\mathbf{Z}$. Note that $\tilde{\mathcal{D}}(w)$ can be $\perp$. After repeating several searches, $\mathbf{Z}$ outputs a bit b.

On the other hand, the ideal world is described as follows: In the store phase, $\mathbf{Z}$ sends $(\mathcal{D}, \mathcal{W})$ to the dummy client. The dummy client sends (**store**, $\mathcal{D}, \mathcal{W}$) to functionality $\mathcal{F}^L_{vSSE}$ (see Fig. A·1). In the search phase, $\mathbf{Z}$ sends w to the dummy client. The dummy client sends (**search**, w) to $\mathcal{F}^L_{vSSE}$, and receives $\mathcal{D}(w)$ or $\perp$ (according to ideal adversary $\mathbf{S}^{\mathrm{uc}}$'s decision), which is relayed to $\mathbf{Z}$. At last, $\mathbf{Z}$ outputs a bit b

In both worlds, $\mathbf{Z}$ can communicate with $\mathbf{A}^{\mathrm{uc}}$ (in the real world) or $\mathbf{S}^{\mathrm{uc}}$ (in the ideal world) in an arbitrary way.

Store: Upon receiving the input (**store**, $sid, D_1, \ldots, D_N, \mathcal{W}$) from the dummy client, verify that this is the first input from the client with (**store**, sid).
If it is, then store $\mathcal{D} = \{D_1, \ldots, D_N\}$, and send $L_1(\mathcal{D}, \mathcal{W})$ to $\mathbf{S}^{\mathrm{uc}}$. Otherwise, ignore this input.
Search: Upon receiving (**search**, sid, w) from the client, send $L_2(\mathcal{D}, \mathcal{W}, \mathbf{w}, w)$ to $\mathbf{S}^{\mathrm{uc}}$. Note that in a no-dictionary vSSE scheme, the client may send $w \notin \mathcal{W}$. If $\mathbf{S}^{\mathrm{uc}}$ returns `accept`, then send $\mathcal{D}(w)$ to the client. If $\mathbf{S}^{\mathrm{uc}}$ returns `reject`, then send $\perp$ to the client.

Fig. A·1 Ideal functionality $\mathcal{F}^L_{vSSE}$.

†He may forget $\mathcal{D}, \mathcal{W}, C, \mathcal{I}$.

UC-security of no-dictionary vSSE scheme is defined as follows.

Definition 4 (UC-security with leakage L): We say that a given no-dictionary vSSE scheme has universally composable (UC) security with leakage L against non-adaptive adversaries, if for any PPT real adversary $\mathbf{A}^{\mathrm{uc}}$, there exists a PPT ideal adversary (simulator) $\mathbf{S}^{\mathrm{uc}}$, and for any PPT environment $\mathbf{Z}$,

$$\begin{aligned}&|\Pr[\mathbf{Z} \text{ outputs 1 in the real world}]\\ &\quad - \Pr[\mathbf{Z} \text{ outputs 1 in the ideal world}]|\end{aligned}$$

is negligible.

We can show the following theorem.

Theorem 4: If a no-dictionary vSSE scheme satisfies L-privacy and strong reliability for some L, it has UC security with leakage L against non-adaptive adversaries.

(Proof) Assume that the scheme satisfies L-privacy and strong reliability.

We consider four games $\mathbf{Game}_0, \ldots, \mathbf{Game}_3$. Let

$$p_i = \Pr[\mathbf{Z} \text{ outputs 1 in } \mathbf{Game}_i]$$

for a fixed $\mathbf{A}^{\mathrm{uc}}$. $\mathbf{Game}_0$ is equivalent to the real world in the definition of UC security. So,

$$p_0 = \Pr[\mathbf{Z} \text{ outputs 1 in the real world}].$$

$\mathbf{Game}_1$ is different from $\mathbf{Game}_0$ in the following points.

- In the store phase, the client records $(\mathcal{D}, \mathcal{W}, \mathcal{I})$ as well as the key K.
- In the search phase, if $\mathbf{A}^{\mathrm{uc}}$ instructs the server to return $(\tilde{C}^*, \widetilde{\mathsf{Proof}})$ such that $(\tilde{C}^*, \widetilde{\mathsf{Proof}}) \neq (C^*, \mathsf{Proof}) \leftarrow \mathtt{Search}(\mathcal{I}, C, t(w))$, then the server returns `reject` to the client. Otherwise the server returns `accept`.
- If the client receives `accept` from the server, he sends $\mathcal{D}(w)$ to $\mathbf{Z}$. Otherwise, he sends $\perp$ to $\mathbf{Z}$.

$\mathbf{Game}_1$ is the same as $\mathbf{Game}_0$ until $\mathbf{A}^{\mathrm{uc}}$ instructs the server to return $(\tilde{C}^*, \widetilde{\mathsf{Proof}})$ such that

$$\begin{aligned}&\mathtt{Dec}(K, t(w), \tilde{C}^*, \widetilde{\mathsf{Proof}}) \neq \perp \quad \text{and}\\ &(\tilde{C}^*, \widetilde{\mathsf{Proof}}) \neq (C^*, \mathsf{Proof}).\end{aligned}$$

The above condition is the (strongly) winning condition of $\mathbf{B}$ in $\mathbf{Game}_{reli}$. So, we can obtain

$$|p_0 - p_1| \leq \max_{\mathbf{B}} \Pr[\mathbf{B} \text{ strongly wins in } \mathbf{Game}_{reli}].$$

From the assumption, $|p_0 - p_1|$ is negligibly small.

In $\mathbf{Game}_2$, we split the client into two entities, client1 and client2, as follows: (See Fig. A·2(a).)

- Both client1 and client2 receive all input from $\mathbf{Z}$.
- In the store phase, only client2 sends $(\mathcal{I}, C)$ to the server.
- In the search phase, only client2 sends $t(w)$ to the server. Then, only client1 receives `accept/reject` from the

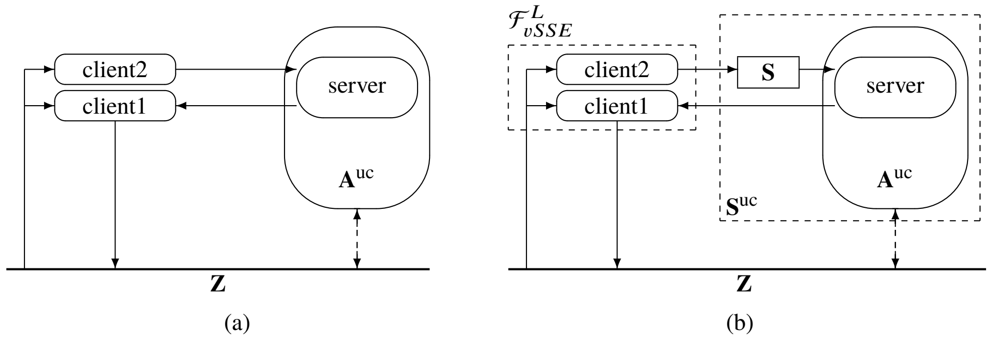

Fig. A·2 (a) $\mathbf{Game}_2$, (b) $\mathbf{Game}_3$.

server, and sends $\mathcal{D}(w)/\perp$ to $\mathbf{Z}$.

This change is conceptual only. Therefore $p_2 = p_1$.

Now, we look at ($\mathbf{Z}$, client1, server, $\mathbf{A}^{\mathrm{uc}}$) and client2 as an adversary $\mathbf{A}$ and a challenger $\mathbf{C}$ in the real game of privacy, respectively. Then, from the assumption, there exists a simulator $\mathbf{S}$ such that Eq. (2) is negligible.

In $\mathbf{Game}_3$, client2 plays the role of the challenger in the simulation game of privacy; he sends $L_1(\mathcal{D}, \mathcal{W})$ or $L_2(\mathcal{D}, \mathcal{W}, \mathbf{w}, w)$ to the simulator $\mathbf{S}$, and then $\mathbf{S}$ sends its outputs (the simulated message) to the server. (See Fig. A·2(b).) Again, we look at ($\mathbf{Z}$, client1, server, $\mathbf{A}^{\mathrm{uc}}$) as $\mathbf{A}$. Then $\mathbf{Game}_3$ is the simulation game and $\mathbf{Game}_2$ is the real game. Therefore

$$|p_3 - p_2| \leq |\Pr[\mathbf{A} \text{ outputs 1 in } \mathbf{Game}_{real}] - \Pr[\mathbf{A} \text{ outputs 1 in } \mathbf{Game}^{L}_{sim}]|,$$

and it is negligible from the assumption.

In $\mathbf{Game}_3$, (client1, client2) behaves exactly the same way as $\mathcal{F}^{L}_{vSSE}$ in the ideal world. So, considering ($\mathbf{S}$, server, $\mathbf{A}^{\mathrm{uc}}$) as a simulator $\mathbf{S}^{\mathrm{uc}}$, we obtain

$$p_3 = \Pr[\mathbf{Z} \text{ outputs 1 in the ideal world}]$$

for this simulator. Consequently, we can say that for any $\mathbf{A}^{\mathrm{uc}}$ there exists $\mathbf{S}^{\mathrm{uc}}$ such that $|p_0 - p_3| = |\Pr[\mathbf{Z} \text{ outputs 1 in the real world}] - \Pr[\mathbf{Z} \text{ outputs 1 in the ideal world}]|$ is negligible. □

Corollary 1: If SSE_0 has $L = (L_1, L_2)$-privacy and F is a pseudorandom function, the vSSE scheme vSSE_1 obtained from SSE_0 using the transformation in Sect. 4 is UC-secure with leakage $L' = (L'_1, L'_2)$ where L and L' are given in Theorem 1.

Wakaha Ogata received the B.S., M.E. and D.E. degrees in electrical and electronic engineering in 1989, 1991 and 1994, respectively, from Tokyo Institute of Technology. From 1995 to 2000, she was an Assistant Professor at Himeji Institute of Technology. Since 2000 she has been working for Tokyo Institute of Technology, and now she is a Professor from 2013. Her current interests are cryptography and information security.

Kaoru Kurosawa received the B.E. and Dr. Eng. degrees in electrical engineering in 1976 and 1981, respectively, from Tokyo Institute of Technology. From 1997 to 2001, he was a Professor in Tokyo Institute of Technology. He is currently a Professor in the Department of Computer and Information Sciences at Ibaraki University. His current research interest is cryptography. He was Program Chair for Asiacrypt 2007, PKC 2013 and some other conferences. Dr. Kurosawa is a member of IEEE, ACM, IACR, IEICE. He received the excellent paper award of IEICE in 1981, the young engineer award of IEICE in 1986, Telecom System Scientific Award of Telecommnucations Avdancement Foundation in 2006 and Achievement Award of IEICE in 2007.