Multi-user verifiable searchable symmetric encryption for cloud storage

Xueqiao LIU

Guomin YANG

Guomin YANG
Singapore Management University, gmyang@smu.edu.sg

Citation

LIU, Xueqiao; YANG, Guomin; and YANG, Guomin. Multi-user verifiable searchable symmetric encryption for cloud storage. (2020). *IEEE Transactions on Dependable and Secure Computing*. 17, (6), 1322-1332.
Available at: https://ink.library.smu.edu.sg/sis_research/7300

Multi-User Verifiable Searchable Symmetric Encryption for Cloud Storage

Xueqiao Liu, Guomin Yang, *Senior Member, IEEE*, Yi Mu, *Senior Member, IEEE*, and Robert H. Deng, *Fellow, IEEE*

Abstract—In a cloud data storage system, symmetric key encryption is usually used to encrypt files due to its high efficiency. In order allow the untrusted/semi-trusted cloud storage server to perform searching over encrypted data while maintaining data confidentiality, searchable symmetric encryption (SSE) has been proposed. In a typical SSE scheme, a users stores encrypted files on a cloud storage server and later can retrieve the encrypted files containing specific keywords. The basic security requirement of SSE is that the cloud server learns no information about the files or the keywords during the searching process. Some SSE schemes also offer additional functionalities such as detecting cheating behavior of a malicious server (i.e., verifiability) and allowing update (e.g., modifying, deleting and adding) of documents on the server. However, the previous (verifiable) SSE schemes were designed for single users, which means the searching can only be done by the data owner, whereas in reality people often use cloud storage to share files with other users. In this paper we present a multi-user verifiable searchable symmetric encryption (MVSSE) scheme that achieves all the desirable features of a verifiable SSE and allows multiple users to perform searching. We then define an ideal functionality for MVSSE under the Universally Composable (UC-) security framework and prove that our ideal functionality implies the security requirements of a secure MVSSE, and our multi-user verifiable SSE scheme is UC-secure. We also implement our scheme to verify its high performance based on some real dataset.

Index Terms—Searchable encryption, dynamics, verifiability, multi-user, UC-security

1 Introduction

With the popularity of cloud computing, more and more data are being stored on cloud storage systems. However, it has also brought new challenges for data security and privacy as users don't fully trust cloud storage providers that are outside the users' trusted domains. To protect the confidentiality of sensitive data in a cloud storage system, the clients can pre-process their data using encryption before uploading them to the remote cloud storage. However, storing only encrypted files brings another problem: if the cloud server could not access the contents of the data, it will not be able to process any data searching queries from the clients.

In order to deal with this problem, searchable encryption was introduced. The idea of searchable encryption is that the server could perform searching on encrypted data without decrypting them. In general, searchable encryption could be classified into public key encryption with keyword search (PEKS) and searchable symmetric encryption (SSE). In this paper we focus on the latter which is more efficient in terms of computation overhead.

Most searchable symmetric encryption schemes are in single-user setting where the client is both the data owner and the data user, and the server is responsible for storing encrypted data, performing data search and returning the corresponding result. This kind of SSE suits the need of single users for storing their personal sensitive data in cloud storage. However, one of the benefits provided by cloud storage is the convenience for data sharing and SSE schemes for single users cannot cater for such a need.

1.1 Our Contributions

In this work, we present a multi-user verifiable searchable symmetric encryption (MVSSE) scheme (Fig. 1) that allows efficient search over encrypted data that are shared among multiple users. Compared with the existing SSE schemes that use single-keyword index and are less efficient in multi-keyword search, i.e., they require search for every keyword once followed by logic operations (conjunction, disjunction, etc.) among all the returned sets, our MVSSE utilizes a two-keyword index that can significantly accelerate multi-keyword search with various logic operations. Moreover, our scheme also achieves other desirable features of SSE such as verifiability against a dishonest server and supporting dynamic data operations. The contributions of our work are four-fold:

- To address the data sharing in multi-user setting, we formalize a solution for constructing secure MVSSE

- X. Liu and G. Yang are with the Institute of Cybersecurity and Cryptology, University of Wollongong, Wollongong, NSW 2522, Australia. E-mail: {xl691, gyang}@uow.edu.au.
- Y. Mu is with the Fujian Provincial Key Laboratory of Network Security and Cryptology, Fujian Normal University, Fuzhou 350117, China. E-mail: ymu.ieee@gmail.com.
- R.H. Deng is with the School of Information Systems, Singapore Management University, Singapore. E-mail: robertdeng@smu.edu.sg.

Manuscript received 30 Apr. 2018; revised 10 Oct. 2018; accepted 14 Oct. 2018. Date of publication 18 Oct. 2018; date of current version 12 Nov. 2020.
(Corresponding authors: Guomin Yang and Yi Mu.)
Digital Object Identifier no. 10.1109/TDSC.2018.2876831

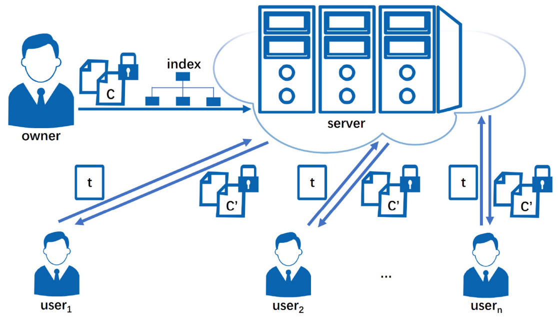


Fig. 1. Multi-user SSE. The owner stores a set of encrypted documents C and the corresponding $index$ which is also encrypted on the server. An authorised user can generate a token t based on the searching keywords and sends it to the server. The server will find the corresponding documents C' and return them to the user.

and formulate the security definitions, namely privacy and reliability, of MVSSE.

- An efficient scheme is presented which utilizes a two-keyword index to reduce the searching time. It is also the first multi-user SSE supporting verifiability and dynamic operations (i.e., adding, deleting and modifying files) on the cloud storage.
- We define the ideal functionality $\mathcal{F}_{MVSSE}$ for MVSSE which can also be used by the future research on this topic in proving the UC-security of any newly proposed schemes.
- We prove the equivalence between UC-security and the security requirements, namely privacy and reliability, of a secure MVSSE. We then prove that our MVSSE scheme is UC-secure against non-adaptive adversaries, i.e., the scheme securely realises our ideal functionality $\mathcal{F}_{MVSSE}$, which implies privacy and reliability.
- We implement our MVSSE scheme and demonstrate its high efficiency in real experiments based on real datasets.

1.2 Related Work

The rudiment of SSE was proposed by Song et al. [1]. In order to search the location of target word W_i, they introduced a solution of sequential scan involving steam cipher and block cipher operations. But the searching time of their scheme is linear to the length of the document.

Secure indexes were introduced by Goh [2]. He designed a secure index which could accelerate searching by the server. He first formulated a security model for indexes named semantic security against adaptive chosen keyword attack, IND-CKA for short. This security model describes the intuition of keeping privacy of document contents: the content of a document can not be deduced from the its index and indexes of other documents which have been queried before. His scheme Z-IDX with a secure index, which is constructed by Bloom filter, achieves $\mathcal{O}(1)$ searching time. In addition, they proved that their scheme is IND-CKA secure.

Chang et al. [3] pointed out that the initial scheme in [2] still leaks some information, namely the number of ‘1’ entries in the Bloom filter of one index. They moved on to propose Privacy Preserving Keyword Searches on Remote Encrypted Data, PPSED for short to depict security requirements that a searchable encryption scheme should meet. Then in a later version of [2], the IND-CKA security model is strengthened by limiting that the two challenging documents could have different lengths. By that it achieves stronger security such that the adversary should not distinguish documents even though they have different file lengths. This modified one is called IND2-CKA for short. They also modified their scheme to make it IND2-CKA secure.

Curtmola et al. [4] formulated the security definitions of SSE: non-adaptive indistinguishability and adaptive indistinguishability, which are all simulation-based definitions. Then they constructed their non-adaptive secure SSE-1 scheme and adaptive secure SSE-2 scheme. They also came up with an extension to realize multi-user SSE by combining a single-user SSE with a broadcast encryption scheme. However, their scheme is not a verifiable one.

Kurosawa et al. [5] provided the definition of verifiable searchable symmetric encryption and its security notions, privacy and reliability. They formulated the ideal functionality $\mathcal{F}$ of verifiable symmetric searchable encryption and proved that UC-security against non-adaptive adversaries[1] is equivalent to their definition of privacy and reliability. They also fixed a flaw in the SSE-2 of [4], proposed a verifiable SSE scheme and proved that it satisfies both privacy and reliability which are equivalent to UC-security. In a subsequent work [6], they also constructed a verifiable SSE that supports file update (i.e., modifying, deleting and adding documents).

Cash et al. implemented an keyword based searchable symmetric encryption scheme OXT which support conjunctive keyword search [7]. The scheme realizes the conjunctive queries with negative keyword search as well. Besides, by converting boolean queries to Searchable Normal Form (SNF for short), it can also support performing boolean queries. Kamara et al. presented a keyword based searchable symmetric encryption scheme IEX which achieves sub-linear complexity in the worst case of disjunctive queries [8]. In addition, it leaks less information than [7] in this worst-case disjunctive search. It also provided an extension scheme $BIEX$ to realize boolean queries and an extension scheme ZMF to satisfy adaptive security.

In some special scenarios, the user desires to selectively retrieve its data. For example, after sending a multi-keyword query to the server, the user requests the server to sort the returning documents by the ascending order of keyword appearing times and return the top k encrypted files. This demands the server to execute computation on the encrypted data. Some schemes [9], [10] have realized several analogous functions with the fully homomorphic encryption (FHE for short) technology.

PEKS, the asymmetric counterpart of SSE, was introduced in [11]. Variants of PEKS were proposed in subsequent works, reducing search complexity in [12], supporting more expressive search mode in [13], resisting inside Keyword Guessing Attack (KGA) in [14], [15] and so on.

1.3 Organization

In Section 2, we retrospect the concepts which will be applied in our scheme. Problem formulation is presented in Section 3. Then the formal syntax and security definitions of

1. Non-adaptive adversaries refer to adversaries that make queries without taking into account the trapdoors or search outcomes of previous queries, while adaptive adversaries choose their queries based on trapdoors and search outcomes from previous queries.

MVSSE are presented in Section 4. A concrete construction of MVSSE is shown in Section 5. Security analysis is given in Section 6. Section 7 contains several variants of our MVSSE system in diverse application scenarios, followed by our scheme perfomance evaluation. Finally, we review all this work and pose an open problem.

2 Preliminaries

2.1 Notation

In this paper, $x, y, z \in_R S$ denotes that all x, y, z are chosen independently and uniformly at random from a finite set S. $|k|$ denotes the length of bits of k. PPT denotes probabilistic polynomial time.

By convention, n denotes the number of stored documents and m denotes the number of keywords. $\mathcal{D} = \{D_1, \cdots, D_n\}$ denotes target document set whose documents will be uploaded to the server and $\mathcal{W} = \{w_1, \cdots, w_m\}$ denotes the keyword set whose keywords are contained in the above document set $\mathcal{D}$. $\texttt{Index} = \{e_{i,j}\}$ is an n column matrix such that:

$$e_{i,j} = \begin{cases} 1 & \text{if } w_i \text{ is contained in } D_j \\ 0 & \text{otherwise} \end{cases}$$

$D(w)$ denotes the set of documents which contain the keyword w and $List(w) = \{i | D_i \text{ contains keyword } w\}$ denotes the set of sequence number of documents.

2.2 UC-Security Framework

Universally Composable (UC-) security, which was introduced in [16], is a powerful tool to measure protocol security. If a protocol is UC-secure, then the security properties of this protocol could be maintained under general protocol composition.

In general, a UC-security framework has three roles: an environment $\mathcal{Z}$, involved parties $P_i (i = 1, \cdots, n)$ and an adversary $\mathbf{A}$. In addition, an ideal functionality $\mathcal{F}$ should be defined in the UC-security framework. $\mathcal{Z}$ is an environment outside $\mathcal{F}$, P_i and $\mathbf{A}$, which gives inputs to parties in the framework, reads outputs from parties, interacts with $\mathbf{A}$ and try to distinguish real world scenarios from ideal world scenarios. Involved parties $P_i (i = 1, \cdots, n)$ in the framework are regarded as dummy ones which could do nothing but forwarding inputs they received from $\mathcal{Z}$ to the functionality $\mathcal{F}$ and forwarding outputs they received from $\mathcal{F}$ to $\mathcal{Z}$ as well. $\mathcal{F}$ is a trusted powerful ideal functionality which could complete any complex computation of the protocol for dummy parties $P_i (i = 1, \cdots, n)$.

We say that Σ securely realizes $\mathcal{F}$ if for any $\mathbf{A}$ there exists an ideal adversary $\mathbf{S}$ such that no environment $\mathbf{Z}$ could tell whether it is interacting with $\mathbf{A}$ and parties running the protocol in real world, or with $\mathbf{S}$ and dummy parties whose computation is undertaken by $\mathcal{F}$ in ideal world.

2.3 RSA Accumulator

Below we briefly review the RSA Accumulator used in [6] and will also be used in our scheme.

Let $P = 2P' + 1$ and $Q = 2Q' + 1$ be two large primes such that P' and Q' are also primes and $|PQ| > 3\lambda$. λ is a security parameter. Let $N = PQ$ and let

$$QR_N = \{a | a = u^2 \bmod N \text{ for some } u \in Z_N^*\}.$$

Then QR_N is a cyclic group of size $(P-1)(Q-1)/4$. Let v be a generator of QR_N. We say that a family of functions $F = \{f : A \to B\}$ is two-universal if $\Pr[f(u_1) = f(u_2)] = 1/|B|$ for all $u_1 \neq u_2$ and for a randomly chosen function $f \in F$. For a set $E = \{y_1, \cdots, y_R\}$ with $y_i \in \{0,1\}^\lambda$, the RSA accumulator works as follows.

1) For each y_i, the prover chooses a prime u_i such that $f(u_i) = y_i$ randomly. Let $prime(y_i)$ denote such a prime u_i. Then computes the accumulated value of $E = \{y_1, \cdots, y_R\}$ as

$$\texttt{Acc}(E) = v^{\prod_{i=1}^{R} prime(y_i)} \bmod N,$$

and sends $\texttt{Acc}(E)$ to the verifier.

2) Later the prover proves that $y_j \in E$ to the verifier as follows. It computes

$$\pi_j = v^{\prod_{i \neq j} prime(y_i)} \bmod N$$

and sends π_j and $prime(y_j)$ to the verifier.

3) The verifier verifies that

$$\texttt{Acc}(E) = (\pi_j)^{prime(y_j)} \bmod N.$$

In fact, when computing $prime(y)$, they will use $PRF_0(k_0, y)$ as the randomness. k_0 is chosen randomly and $PRF_0 : \{0,1\}^{|k_0|} \times \{0,1\}^\lambda \to \{0,1\}^\lambda$ so that they can get the same $prime(y)$ locally.

Proposition 1. *[6] Given N, v, f and $E = y_1, \cdots, y_n$, it is hard to find $y \notin E$ and π, such that*

$$\pi^{prime(y)} = \texttt{Acc}(E) \bmod N \quad (3)$$

under the strong RSA assumption.

3 Problem Formulation

3.1 System Model

As shown in Fig. 1, our system involves three different parties: a data owner who stores documents on the cloud server, a group of data users who are permitted to search the stored documents of the data owner and a cloud server who offers storage service for the data owner and search service for the data users.

3.1.1 Data Owner

The data owner is responsible for generating the system parameters and the private keys for the data users. Then it encrypts documents, generates a related index and corresponding verifiable information. Finally it stores both the encrypted documents and the index on the cloud server, and sends the verifiable information to the system users. It is also able to add, modify and delete documents on the cloud server.

3.1.2 Data Users

A data user is permitted to search the documents stored on the cloud server. When the data user wants to search documents containing some keywords, it generates a trapdoor and communicates with the cloud server to retrieve the target documents. Finally, the data user should also be able to execute validity check on the search results to decide whether to accept the search results or not.

3.1.3 Cloud Server

The cloud server is in charge of storing documents and the index uploaded by the data owner as well as responding the search queries from the valid data users with documents and verification information so that the users can perform validity check on the searching result.

3.2 Threat Model

In this paper, it is assumed that the cloud server is a malicious adversary. It means the cloud server is curious about the documents as well as the corresponding index and keywords and tries to collect more information than the permitted leakage. In addition, the cloud server may also delete or forge documents and only return partial searching results. It is worth noting that although some malicious behaviours such as forging search outcomes or returning fewer results may not be intentionally done by the cloud server, they may still occur due to malware or software bugs [6].

The data owner is assumed to be honest. In other words, it honestly generates parameters, outsources documents together with the related index to the cloud server. At the same time, it also provides the authorized data users with authentic keys and verifiable information.

The data users are also supposed to be honest-but-curious. They honestly generate a trapdoor and communicate with the cloud server during every search.

3.3 Design Goals

A secure MVSSE should satisfy the following features and security requirements.

3.3.1 Multi-User Search Availability

Every authorised data user is permitted to directly search documents stored on the cloud storage by preparing a trapdoor associated with some searching keywords.

3.3.2 Verifiability

The cloud server returns not only searching results but also a proof. The data user can test the correctness and completeness of the searching results with the proof.

3.3.3 Dynamic Data

The system allows the data owner to add, modify and delete documents efficiently. That is, these operations should not influence the index items related to other documents in order to minimize the cost.

3.3.4 Efficiency

The system is able to process multi-keyword queries efficiently.

3.3.5 Data Privacy

The cloud server is not be able to learn either the outsourced documents or the related index.

3.3.6 Keyword Privacy

Given a trapdoor, the server cannot learn the related keyword even though the server can offer search service.

3.3.7 Dynamic Users

When a new data user is added into the system or an existing data user is revoked, other data users should not be influenced.

4 MULTI-USER VERIFIABLE SSE SYSTEM AND SECURITY DEFINITIONS

Below we formalize the MVSSE system model, present the syntax of a multi-user verifiable searchable symmetric encryption scheme and its security definitions.

4.1 Multi-User Verifiable SSE Scheme

A multi-user verifiable searchable symmetric encryption scheme is a protocol which is run among a data owner, a server and a set of data users as follows.

Setup. Upon input security parameter λ, the owner publishes public parameters pp, sends user private keys sk_i to user i and server key k_s to the server, and keep master key mk secret.

Store. Upon input pp, mk, document set $\mathcal{D}$ and keyword set $\mathcal{W}$, the owner sends encrypted index $\mathcal{I}$ and encrypted document set $\mathcal{C} = \{C_1, \cdots, C_n\}$ to the server, and verifiable information A to each user.

TokenGen.

1) Upon input a keyword set $\mathcal{W}_s \subset \mathcal{W}$, user i generates trapdoor $token$ and sends it to the server.
2) The server computes auxiliary information H and sends it to user i.

Search.

1) Upon input H, user i computes search query L and sends it to the server.
2) The server finds encrypted document set $C(\mathcal{W}_s)$ and computes partial verifiable information Tag, then sends them to user i.
3) User i accepts $(C(\mathcal{W}_s), Tag)$ and decrypts $C(\mathcal{W}_s)$ to get result set $D(\mathcal{W}_s)$. Otherwise it outputs `reject`.

Modify.

1) Upon input document sequence number k and modified document $\overline{D}_k$, the owner sends k and encrypted document $\overline{C}_k$ to the server.
2) The server computes partial verifiable information Tag and sends it to the owner.
3) The owner sends verifiable information $\overline{A}$ to each user or outputs `reject`.

Delete. This phase is similar to the phase $Modify$. $\overline{D}_k$ is replaced by $"delete"$.

Add.

1) Upon input new document D_{n+1}, the owner computes update material α and new verifiable information A, then sends (C_{n+1}, α) to the server, sends A to each user.
2) The server stores C_{n+1}, updates $\mathcal{I}$ with α and n to $n+1$.

Real Game ($Game_{real}$)

- In the setup phase, an adversary $\mathbf{A}$ chooses a user set S, for $j = 1, \cdots, q_0$,
  1) $\mathbf{A}$ chooses user sequence number i and sends (S, i) to the challenger $\mathbf{C}$.
  2) $\mathbf{C}$ returns user private key d_i and broadcast header Hdr to $\mathbf{A}$.
- In the store phase, $\mathbf{A}$ chooses $(\mathcal{D}, \mathcal{W})$ and sends them to $\mathbf{C}$. $\mathbf{C}$ returns $(\mathcal{I}, \mathcal{C})$.
- In the tokengen phase, for $j = 1, \cdots, q_1$,
  1) $\mathbf{A}$ chooses keywords $w_{a_j} \in \mathcal{W}$ and $w_{b_j} \in \mathcal{W}$ and sends them to $\mathbf{C}$.
  2) $\mathbf{C}$ returns a trapdoor $token(w_{a_j} \wedge w_{b_j})$ to $\mathbf{A}$.
- In the search phase, for $j = 1, \cdots, q_2$,
  1) $\mathbf{A}$ chooses broadcast header $\overline{Hdr}$, keywords $w_{a_j} \in \mathcal{W}$ and $w_{b_j} \in \mathcal{W}$ and sends them to $\mathbf{C}$.
  2) $\mathbf{C}$ returns entry location and pad string $\{label_j, pad_j\}$ to $\mathbf{A}$.
- In the modify phase, for $j = 1, \cdots, q_3$,
  1) $\mathbf{A}$ sends document sequence number and corresponding document (k, D_k) to $\mathbf{C}$.
  2) $\mathbf{C}$ returns $C_k = E_{k_e}(D_k)$ to $\mathbf{A}$.
- In the delete phase, for $j = 1, \cdots, q_4$,
  1) $\mathbf{A}$ sends k to $\mathbf{C}$.
  2) $\mathbf{C}$ returns $C_k = E_{k_e}(delete)$ to $\mathbf{A}$.
- In the add phase, for $j = 1, \cdots, q_5$,
  1) $\mathbf{A}$ sends D_{n+1} to $\mathbf{C}$.
  2) $\mathbf{C}$ returns $C_{n+1} = E_{k_e}(D_{n+1})$ to $\mathbf{A}$.
- Finally $\mathbf{A}$ outputs a bit b.

Fig. 2. Real game.

3) Each user updates A and n to $n + 1$.

4.2 Security Definitions

We define two games: $\texttt{Game}_{real}$ (Fig. 2) and $\texttt{Game}_{sim}$ (Fig. 3). The former one depicts the process of real interaction between an adversary and a challenger running our protocol while the latter one depicts the process of interaction between an adversary, a simulator and a challenger trying to simulate all the same scenarios as the real protocol.

4.2.1 Privacy

The cloud server is curious about documents and index stored on itself and tries to collect more information than the permitted leakage. [4] formulated this security definitions on verifiable SSE and we apply it to our MVSSE whose real Game and simulation Game are different from theirs.

From intuition, privacy means the server should not learn any more information than permitted leakage such as document size and sequence numbers of desired documents.

Definition 1. *We say that a verifiable SSE satisfies privacy if there exists a PPT simulator* **Sim** *such that*

$$\begin{aligned}\mathsf{Adv}^{pri}_{\mathbf{A}}(\lambda) = |&\Pr(\mathbf{A} \text{ outputs } b = 1 \text{ in } \texttt{Game}_{real}) \\ &- \Pr(\mathbf{A} \text{ outputs } b = 1 \text{ in } \texttt{Game}_{sim})|\end{aligned} \quad (1)$$

is negligible for any PPT adversary **A**.

Simulation Game ($Game_{sim}$)

- In the setup phase, an adversary $\mathbf{A}$ chooses a user set S, for $j = 1, \cdots, q_0$,
  1) $\mathbf{A}$ chooses user sequence number i and sends (S, i) to the challenger $\mathbf{C}$.
  2) $\mathbf{C}$ sends (S, i) to the simulator $\mathbf{Sim}$.
  3) $\mathbf{Sim}$ returns d'_i and broadcast header Hdr' to $\mathbf{C}$, and it replays them to $\mathbf{A}$.
- In the store phase,
  1) $\mathbf{A}$ chooses $(\mathcal{D}, \mathcal{W})$ and sends them to $\mathbf{C}$.
  2) $\mathbf{C}$ sends $|D_1|, \cdots, |D_n|$ and $|\mathcal{W}|$ to simulator $\mathbf{Sim}$, where $\mathcal{D} = \{D_1, \cdots, D_n\}$.
  3) $\mathbf{Sim}$ returns $(\mathcal{I}', \mathcal{C}')$ to $\mathbf{C}$, and it replays them to $\mathbf{A}$.
- In the tokengen phase, for $j = 1, \cdots, q_1$,
  1) $\mathbf{A}$ chooses keywords $w_{a_j} \in \mathcal{W}$, $w_{b_j} \in \mathcal{W}$, and sends them to $\mathbf{C}$.
  2) $\mathbf{C}$ sends $token_j$ to $\mathbf{Sim}$.
  3) $\mathbf{Sim}$ returns $token'_j$ to $\mathbf{C}$, and it replays it to $\mathbf{A}$.
- In the search phase, for $j = 1, \cdots, q_2$,
  1) $\mathbf{A}$ chooses broadcast header $\overline{Hdr}$, keywords $w_{a_j} \in \mathcal{W}$ and $w_{b_j} \in \mathcal{W}$, and sends them to $\mathbf{C}$.
  2) $\mathbf{C}$ sends $List(w_{a_j} \wedge w_{b_j}) = \{k | D_k$ contains both w_{a_j} and $w_{b_j}\}$ to $\mathbf{Sim}$.
  3) $\mathbf{Sim}$ returns $\{label'_j, pad'_j\}$ to $\mathbf{C}$, and it replays it to $\mathbf{A}$.
- In the modify phase, for $j = 1, \cdots, q_3$,
  1) $\mathbf{A}$ sends $(k, \overline{D}_k)$ to $\mathbf{C}$.
  2) $\mathbf{C}$ sends $(k, |\overline{D}_k|)$ to $\mathbf{Sim}$.
  3) $\mathbf{Sim}$ returns $\overline{C}'_k$ to $\mathbf{A}$.
- In the delete phase, for $j = 1, \cdots, q_4$,
  1) $\mathbf{A}$ sends k to $\mathbf{C}$.
  2) $\mathbf{C}$ sends k to $\mathbf{Sim}$.
  3) $\mathbf{Sim}$ returns $\overline{C}'_k(delete)$ to $\mathbf{A}$.
- In the add phase, for $j = 1, \cdots, q_5$,
  1) $\mathbf{A}$ sends D_{n+1} to $\mathbf{C}$.
  2) $\mathbf{C}$ sends $|D_{n+1}|$ to $\mathbf{Sim}$.
  3) $\mathbf{Sim}$ returns C'_{n+1} to $\mathbf{A}$.
- Finally $\mathbf{A}$ outputs a bit b.

Fig. 3. Simulation game.

4.2.2 Reliability

Besides considering the leakage in each phase, the server should not successfully forge a valid returning result $(\texttt{C}(w)^*, Tag^*)$ which could pass the validity check of the client as well. Now let us define this security notion.

For fixed $(\mathcal{D}, \mathcal{W})$ and search queries $w_1, \cdots, w_q \in \mathcal{W}$ arbitrarily, $\mathbf{A}$ wins if it could return $(\texttt{C}(w_i)^*, Tag^*)$ for any query $t(w_i)$ such that $\texttt{C}(w_i)^* \neq \texttt{C}(w_i)$ and $\texttt{Verify}(K, t(w_i), \texttt{C}(w_i)^*, Tag^*) = \texttt{accept}$.

Definition 2. *We say that a verifiable SSE satisfies reliability if for any PPT* **A**,

$$\mathsf{Adv}^{rel}_{\mathbf{A}}(\lambda) = \Pr(\mathbf{A} \text{ wins}) \quad (2)$$

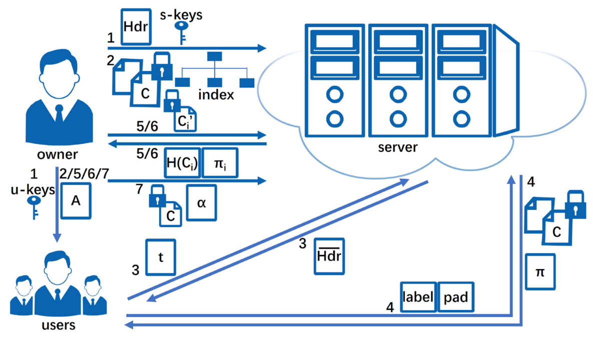


Fig. 4. Scheme model. The scheme consists of seven phases: 1. Setup; 2. Store; 3. TokenGen; 4. Search; 5. Modify; 6. Delete; 7. Add.

is negligible for any $(\mathcal{D}, \mathcal{W})$ and any search queries $w_1, \cdots, w_q$.

5 Our MVSSE Construction

In this section, we present our MVSSE construction. Below we first give an overview of our construction.

5.1 An Overview of Our Construction

We start our system prototype on the foundation of a single-user searchable encryption scheme [6] since it is a dynamic one and our scheme can inherit functionalities of adding, modifying and deleting documents. To transform it to a scheme in multi-user setting, we integrate the broadcast encryption [17] with it, regarding data users as broadcast users.

The *label* which helps locate index items is designed to be computed from two parts: the broadcast plaintext K and the trapdoor t. The former is obtained by decrypting the broadcast ciphertext Hdr and the latter is related to keywords. As shown in Fig. 4, parameters and keys are delivered to in Setup phase, so is Hdr. Then the owner uploads documents and *index* to the server in Store phase. Receiving Hdr, the user decrypts it to get K, generates t from desired keywords and figures out *label* in Search phase. However, once a user obtains K, it needs not to ask the permission to search any more. What we expect is each search is granted in real time. Therefore, K cannot be obtained directly and another round of communication is added. In TokenGen phase, the server processes Hdr with the received t and returns $\overline{Hdr}$. By decrypting $\overline{Hdr}$, the user gets *label* and sends it with *pad* (auxiliary information for decrypting the encrypted index item) to the server. Finally, the server returns target documents C.

Another issue is how to verify searching results. Here verification should be performed on checking two aspects: if returned documents are modified and if more or fewer documents are returned, i.e., the index item is tampered. Two accumulators which are used for checking set members are employed: one is deployed on documents and the other is deployed on encrypted index items. Refering to Fig. 4, the server returns the verifiable information π together with C so that the user computes the accumulators and compares them with A (accumulators received from the owner initially).

To improve multi-keyword query efficiency, we deploy a two-keyword index, i.e., each index item represents documents containing two keywords so that the user can deal with two keywords once. Though a multi-keyword index using more keywords will bring further acceleration than a two-keyword index in a multi-keyword search, it inflates the number of index items and occupies more storage space at the same time. In addition, computation cost on the index accumulator for checking the correctness of the index is also related to the number of index items. Thus, deploying multi-keyword index will also significantly increase the computation cost of various operations involving the index accumulator. To balance the efficiency in different operations, we make a trade-off and adopt the two-keyword index.

5.2 Details of Our Construction

Our proposed MVSSE scheme consists of the following phases.

Setup.

1) $\mathbb{G}$ is a bilinear group of prime order p. There exists a group $\mathbb{G}_1$ and an efficiently computable bilinear map $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_1$. g is a generator of $\mathbb{G}$ and $\alpha \in_R \mathbb{Z}_p$ is randomly chosen. $g_i = g^{(\alpha^i)}$ is computed for $i = 1, 2, \cdots, b, b+2, \cdots, 2b$. The owner randomly chooses $\gamma \in_R \mathbb{Z}_p$ and sets $s = g^\gamma$. The public key is $PK = (g, g_1, \cdots, g_b, g_{b+2}, \cdots, g_{2b}, s)$. $U = \{1, 2, \cdots, b\}$ and the private key for user $i \in U$ is $d_i = g_i^\gamma$. Suppose the user set is $S \subset U$. The owner randomly chooses $t \in_R \mathbb{Z}_p$ and sets $K = e(g_{b+1}, g)^t = e(g_b, g_1)^t$, $Hdr = (c_0, c_1) = (g^t, (s \cdot \prod_{j \in S} g_{b+1-j})^t)$.
2) $PRF_1(k_1, \cdot)$ is a pseudo-random function: $\{0,1\}^{|k_1|} \times \{0,1\}^* \rightarrow \mathbb{Z}_p$. $PRF_2(k_2, \cdot)$ is a pseudo-random function: $\{0,1\}^{|k_2|} \times \{0,1\}^* \rightarrow \{0,1\}^*$. $PRF_3(k_3, \cdot)$ is a pseudo-random function: $\{0,1\}^{|k_3|} \times \mathbb{Z}_p \rightarrow \mathbb{Z}_p$. $SKE = \{G, E, E^{-1}\}$ is a symmetric-key encryption scheme, where G is a key generation algorithm, E is an encryption algorithm and E^{-1} is a decryption algorithm and assume SKE is CPA-secure. $H : \{0,1\}^* \rightarrow \{0,1\}^\lambda$ is a collision-resistant hash function and will be used in following phases. The owner randomly chooses three keys k_1, k_2, k_3 for the above three PRFs respectively and one key k_e for SKE.
3) The owner selects RSA accumulator parameters as Section 2.3. N, v, f, k_0 are public and P, Q are secret.
4) The owner publishes $\{PRF_0, PRF_1, PRF_2, PRF_3, H, f, SKE, N, v, f, k_0, \mathbb{G}, \mathbb{G}_1, p, g, \{g_i\}_{i=1,\cdots,b,b+2,\cdots,2b}, S, b, s\}$, sends $\{k_1, k_2, k_e, d_i\}$ to each user, sends $\{k_3, Hdr\}$ to the server, keeps $\{P, Q, \alpha, \gamma, \sigma, t, K\}$ secret.

Store.

1) $D(w_i \wedge w_j)$ denotes the set of documents which contain both keyword $w_i \in W$ and $w_j \in W$ and $List(w_i \wedge w_j) = \{k | D_k \text{ containsboth } w_i \text{ and } w_j\}$. On input $(\mathcal{D}, W)$, the owner stores $(\mathcal{I}, \mathcal{C})$ to the server, where $\mathcal{D} = \{D_1, D_2, \cdots, D_n\}$ is a set of documents and $\mathcal{W} = \{w_1, w_2, \cdots, w_m\}$ is a set of keywords. Let $\texttt{Index} = \{e_{i,j,k}\}$ be an $\frac{m(m+1)}{2} \times n$ binary matrix such that

$$e_{i,j,k} = \begin{cases} 1 & \text{if } w_i \text{ and } w_j \text{ are both contained in } D_k, \\ 0 & \text{otherwise.} \end{cases}$$

All keywords are arranged in the lexicographical order. For example, $w_i =$ "*common*" is placed before $w_j =$ "*computer*" and $i < j$. There are m rows about one single keyword which could be considered as w_i

with w_j in which $i = j$. The original index $\texttt{Index} = \{e_{i,j,k}\}$ are computed from w_i and w_j in which $i \leq j$. Thus the number of rows is $C_m^2 + m = \frac{m(m+1)}{2}$. That is to say the directory will be in this order $w_1 \wedge w_1, w_1 \wedge w_2, \cdots, w_1 \wedge w_m, w_2 \wedge w_2, w_2 \wedge w_3, \cdots, w_m \wedge w_m$.

2) The owner computes $C_k = E_{k_e}(D_k)$ for each document $D_k \in \mathcal{D}$, $token_l = PRF_1(k_1, w_i||w_j)$ for every two keywords w_i and w_j, and $r_l = PRF_3(k_3, token_l)$, where l is the counter of $token$ corresponding to the counters of keywords i and j. Computes

$$\begin{cases} \overline{label_l} = K^{token_l \cdot r_l} \\ \overline{index_l} = index_l \oplus [PRF_2(k_2, w_i||w_j)]_{1\cdots n} \end{cases}$$

for each two keywords $w_i, w_j \in \mathcal{W}$. Here $||$ denotes concatenation. σ is a random permutation on $\{1, \cdots, \frac{m(m+1)}{2}\}$. Then stores $\mathcal{C} = (C_1, C_2, \cdots, C_n)$, $\mathcal{I} = \{\overline{label_{\sigma(l)}}, \overline{index_{\sigma(l)}}\}$ to the server.

3) $\texttt{Acc}(\cdot)$ is used on $E_C = \{(k, C_k)|k = 1, \cdots, n\}$ and $E_I = \{(\overline{label_l}, k, [\overline{index_l}]_k)|l = 1, \cdots, \frac{m(m+1)}{2}, k = 1, \cdots, n\}$. The owner computes

$$\begin{cases} A_C = v^{\prod_{k=1}^{n} prime(H(k,H(C_k)))} \bmod N \\ A_I = v^{\prod_{l=1}^{\frac{m(m+1)}{2}} \prod_{k=1}^{n} prime(H(\overline{label_l},k,[\overline{index_l}]_k))} \bmod N \end{cases}$$

and sends $\{A_C, A_I\}$ to each user.

TokenGen. Suppose that one user wants to search on two keywords w_a and w_b.

1) The user computes $token_c = PRF_1(k_1, w_a||w_b)$ and sends it to the server.
2) Then the server computes $r_c = PRF_3(k_3, token_c)$ and $\overline{Hdr} = Hdr^{r_c \cdot token_c} = \{c_0^{r_c \cdot token_c}, c_1^{r_c \cdot token_c}\}$, sends $\overline{Hdr}$ to the user.

Search.

1) The user decrypts $\overline{Hdr}$ to get $label_c = K^{r_c \cdot token_c}$ and computes $pad_c = [PRF_2(k_2, w_a||w_b)]_{1\cdots n}$. Then sends $\{label_c, pad_c\}$ to the server.
2) The server access $label_c$ location to get its storing contents, i.e., $\overline{index_c}$. It computes $(e_1, \cdots, e_n) = pad_c \oplus \overline{index_c}$ and sets $C'(w_a \wedge w_b) = \{(k, C_k)|e_k = 1\}$. Then it computes

$$\begin{cases} \pi_C = v^{\prod_{e_k=0} prime(H(k,H(C_k)))} \bmod N, \\ \pi_I = v^{\prod_{l \neq c} \{\prod_{k=1}^{n} prime(H(\overline{label_l},k,[\overline{index_l}]_k))\}} \bmod N. \end{cases}$$

The server returns $(C'(w_a \wedge w_b), \pi_C, \pi_I)$ to the user.

3) The user first computes $X_i = prime(H(i, H(C_i)))$ for each $(i, C_i) \in C'(w_a \wedge w_b)$, and checks if

$$A_C = (\pi_C)^{\prod_{e_i=1} X_i} \bmod N. \quad (3)$$

The user then reconstructs $(e_1, \cdots, e_n)$ from $C'(w_a \wedge w_b)$ and computes $\overline{index_c} = pad_c \oplus (e_1, \cdots, e_n)$. Computes $z_j = prime(H(label_c, j, [\overline{index_c}]_j))$ for $j = 1, \cdots, n$, and checks if

$$A_I = (\pi_I)^{\prod_{j=1}^{n} z_j} \bmod N. \quad (4)$$

If both of the checks succeed, the user decrypts all C_i and outputs the documents $\{D_i|e_i = 1\}$. Otherwise it outputs $\texttt{reject}$.

Modify. Suppose that the owner wants to modify C_i to $\overline{C}_i$.

1) The owner sends $(i, \overline{C}_i)$ to the server.
2) The server computes $\pi_i = v^{\prod_{k \neq i} prime(H(k,H(C_k)))} \bmod N$ and returns $(H(C_i), \pi_i)$ to the owner.
3) The owner computes $X_i = prime(H(i, H(C_i)))$ and checks if

$$A_C = (\pi_i)^{X_i} \bmod N. \quad (5)$$

If the check fails, then he outputs $\texttt{reject}$. Otherwise he computes

$$\begin{aligned} \overline{X}_i &= prime(H(i, H(\overline{C}_i)))), \\ d &= \overline{X}_i / X_i \bmod (P-1)(Q-1), \\ \overline{A}_C &= (A_C)^d = v^{X_1 \cdots \overline{X}_i \cdots X_n} \bmod N. \end{aligned}$$

The owner sends $\overline{A}_C$ to each user for updating A_C.

Delete. Suppose that the owner wants to delete C_i.

1) It first sends (i) to the server.
2) Then applies **Modify** to $\overline{C}_i = E_{k_e}(delete)$.

Add. Suppose that the owner wants to add a document D_{n+1}. Let

$$e_{i,j,n+1} = \begin{cases} 1 & \text{if } w_i \text{ and } w_j \text{ are both contained in } D_{n+1}, \\ 0 & \text{otherwise.} \end{cases}$$

1) The owner computes $C_{n+1} = E_{k_e}(D_{n+1})$ and

$$\overline{A}_C = (A_C)^{prime(H(n+1,H(C_{n+1})))} \bmod N.$$

Then sends them to each user for updating A_C to $\overline{A}_C$.

2) The owner also computes $\alpha_l = [PRF_2(k_2, w_i||w_j)]_{n+1} \oplus e_{i,j,n+1}$ for $l = 1, \cdots, \frac{m(m+1)}{2}$, where $[PRF_2(k_2, w_i||w_j)]_{n+1}$ denotes the $n+1$th bit of $PRF_2(k_2, w_i||w_j)$. It sends $C_{n+1}, (\alpha_{\sigma(1)}, \cdots, \alpha_{\sigma(\frac{m(m+1)}{2})})$ to the server.
3) The server updates $\overline{index_{\sigma(l)}}$ to $\overline{\overline{index_{\sigma(l)}}} = \overline{index_{\sigma(l)}}||\alpha_{\sigma(l)}$ for $l = 1, \cdots, \frac{m(m+1)}{2}$.
4) The owner computes $z_l = prime(H(\overline{label_l}, n+1, \alpha_l))$ and computes $\overline{A}_I = (A_I)^{z_1 \cdots z_{\frac{m(m+1)}{2}}} \bmod N$. It sends it to each user for updating A_I to $\overline{A}_I$ and n to $n+1$.

6 Security

In this section, we prove the security of our MVSSE scheme under the Universally Composable security framework. We first show that UC-security against non-adaptive adversaries implies privacy and reliability (Sections 4.2.1 and 4.2.2), then prove that our scheme is UC-secure.

6.1 Ideal Functionality

Ideal function is used for describing parties' all computation. Here we regard all parties (the owner, the server and the users) as dummy ones who interact with the environment $\mathcal{Z}$ and delegate their computation to the ideal functionality $\mathcal{F}$.

Our ideal functionality $\mathcal{F}$ contains all phases and covers security requirements of each phase. In the setup phase, it

Ideal Functionality $\mathcal{F}$

Running with the dummy owner P_1, the dummy server P_2, the dummy user P_3 and an adversary $\mathbf{S}$.

- Upon receiving $(\mathbf{setup}, sid, S)$ from P_1, verify that this is the first input from P_1 with $(\mathbf{setup}, sid)$. If so, choose public parameters, send Hdr to $\mathbf{S}$ and send d_i to P_3. Otherwise ignore this input.
- Upon receiving $(\mathbf{store}, sid, \mathcal{D}, \mathcal{W}, Index)$ from P_1, verify that this is the first input from P_1 with $(\mathbf{store}, sid)$. If so, store $(n, \mathcal{D}, \mathcal{W}, Index)$, and send $|D_1|, \cdots, |D_n|$ and $|\mathcal{W}|$ to $\mathbf{S}$. Otherwise ignore this input.
- Upon receiving $(\mathbf{tokengen}, sid, w_a, w_b)$ from P_3, send $token_c$ to $\mathbf{S}$, where $w_a \in \mathcal{W}$ and $w_b \in \mathcal{W}$, and send $\overline{Hdr}$ to P_3.
- Upon receiving $(\mathbf{search}, sid, \overline{Hdr}, w_a, w_b)$ from P_3, send $List(w_a \wedge w_b)$ to $\mathbf{S}$, where $w_a \in \mathcal{W}$ and $w_b \in \mathcal{W}$.
    1) If $\mathbf{S}$ returns OK, then send $D(w_a \wedge w_b)$ to P_3.
    2) If $\mathbf{S}$ returns reject, then send `reject` to P_3.
- Upon receiving $(\mathbf{modify}, sid, i, \overline{D}_i)$ from P_1, send $(i, |\overline{D}_i|)$ to $\mathbf{S}$.
    1) If $\mathbf{S}$ returns OK, then replace D_i with $\overline{D}_i$.
    2) If $\mathbf{S}$ returns reject, then send `reject` to P_1.
- Upon receiving $(\mathbf{delete}, sid, i)$ from P_1, send i to S.
    1) If $\mathbf{S}$ returns OK, then let $\overline{D}_i := delete$.
    2) If $\mathbf{S}$ returns reject, then send `reject` to P_1.
- Upon receiving $(\mathbf{add}, sid, D_{n+1})$ from P_1, send $|D_{n+1}|$ to $\mathbf{S}$, add D_{n+1} to $\mathcal{D}$ and update n to $n+1$.

Fig. 5. Ideal Functionality.

leaks Hdr to the ideal adversary $\mathbf{S}$. In store phase, it leaks $|D_1|, \cdots, |D_n|$ and $|\mathcal{W}|$ to $\mathbf{S}$. In the tokengen phase, it leaks $token_c$ to $\mathbf{S}$. In the search phase, it leaks $List(w_a \wedge w_b)$ to $\mathbf{S}$. In the modify phase, it leaks $(i, |\overline{D}_i|)$ to $\mathbf{S}$. In the delete phase, it leaks i to $\mathbf{S}$. In the add phase, it leaks $|D_{n+1}|$ to $\mathbf{S}$. Our ideal functionality is defined in Fig. 5.

We say that our protocol is UC-secure if it securely realizes the ideal functionality $\mathcal{F}$.

6.2 Equivalence

We prove the secure realization of our ideal functionality $\mathcal{F}$ in the UC-framework is equivalent to the definitions of privacy and reliability presented in Sections 4.2.1 and 4.2.2. The necessity and sufficiency are discussed respectively in the follwing two theorems.

Theorem 1. *A MVSSE scheme satisfies privacy and reliability if the corresponding protocol Σ_{MVSSE} is UC-secure against non-adaptive adversaries.*

The proof is given in Appendix A, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TDSC.2018.2876831.

Theorem 2. *Σ_{MVSSE} is UC-secure against non-adaptive adversaries if a MVSSE scheme satisfies privacy and reliability.*

The proof is given in Appendix B, available in the online supplemental material.

6.3 UC-Security of Our Scheme

Because of the equivalence between UC-security and the security requirements, namely privacy and reliability, the UC-security of our scheme implies that our scheme satisfies privacy and reliability.

Theorem 3. *The scheme is UC-secure against non-adaptive adversaries under the strong RSA assumption if broadcast encryption is CPA-secure, SKE is CPA-secure, PRF is a pseudo-random function and H is a collision-resistant hash function.*

The general idea of the proof is that with permitted leakage from the functionality $\mathcal{F}$, our protocol can be simulated perfectly without being detected by the environment $\mathcal{Z}$. It first demonstrates that $\mathcal{Z}$ cannot distinguish running our protocol from interacting with $\mathcal{F}$. Then it shows that the outputs of involved parties in running our protocol are also indistinguishable from the outputs in interacting with $\mathcal{F}$ in the view of $\mathcal{Z}$. The proof is given in Appendix C, available in the online supplemental material.

It is worth noting that our construction is based on the broadcast encryption scheme proposed in [17] that is proven CPA-secure.

7 Extensions

7.1 Boolean Queries

In order to handle boolean queries, such as $w_1 \wedge (w_2 \vee w_3)$, a single keyword index scheme works as follows: the user first executes three queries on each keyword, then performs a disjunctive operation to get $w_2 \vee w_3$ and a conjunctive operation to get the final result sets on the client side.

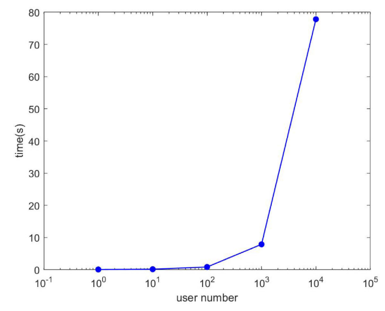
To deal with boolean expressions using our two-keyword index, we can re-write every boolean expression as a disjunctive normal formulation (DNF) whose inner cell expressions are conjunctive expressions. Therefore, for a boolean expression query $w_1 \wedge (w_2 \vee w_3)$, we can perform two searches on $w_1 \wedge w_2$ and $w_1 \wedge w_3$ respectively on the server side and then executes disjunctive operation once on the two returning document sets on the client side. In this example, compared with single keyword index, we can save one searching query on the server side and one conjunctive operation on the client side, and such a saving may vary in different boolean expressions.

7.2 Adding or Revoking Users

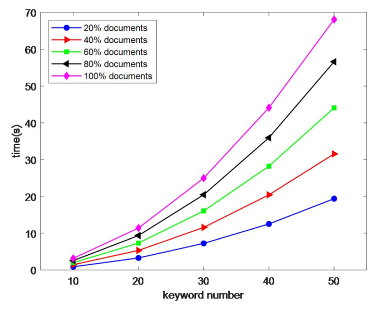
Suppose that the data owner wants to change the target user list for sharing data, including delegating the searching authority to some new users and revoking some users' authority. This could be resolved by adding two phases to our scheme.

Add User. Suppose that the owner would like to add a new user into the target user list S.

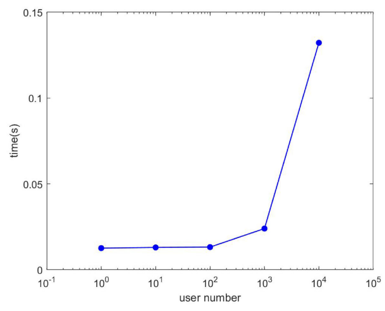
1) Its sequence number i should satisfy $i \in U$ and $i \notin S$ as long as the total number of users does not exceed b yet. The data owner first distributes the private key $d_i = {g_i}^{\gamma}$ to user i.



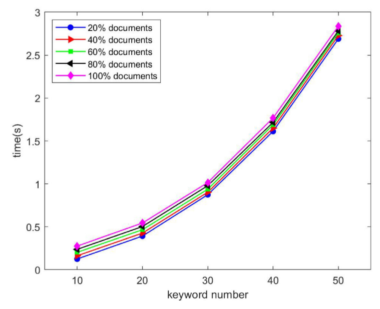
(a) Setup



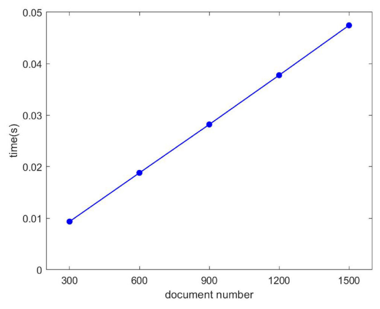
(b) Store



(c) Search (user)



(d) Search (server)



(e) Verify

Fig. 6. Experimental Results.

2) Then the new Hdr should be computed as follows:

$$Hdr = (c_0, c_1') = (c_0, c_1 \cdot g_{b+1-i}^t).$$

The owner sends it to the server for updating.

Revoke User. Suppose that the owner desire to revoke the searching authority of user $i \in S$.

The owner computes the new $Hdr = (c_0, c_1') = (c_0, c_1/g_{b+1-i}^t)$ and sends it to the server for updating.

These two phases only contain the interactions among the owner, the server and the current user. Therefore, other existing users in the target user set will not be influenced at all.

8 Performance Evaluation

In this section, we present the experiment results of our scheme implemented in C++. Our experiments are conducted on a PC with Intel Core i7-4770 CPU (8-core 3.4 GHz) and 16 GB RAM running 64-bit Windows 7 Enterprise.

The system performance is evaluated on NIPS full papers containing 1500 files and Enron Emails containing 39861 files.[2]

8.1 Setup

Fig. 6a illustrates the time cost of Setup phase. As this phase only consists of generating system parameters and multi-user keys, the time cost is linear to the (maximum) number of system users. When there are fewer than 100 system users, the time cost is less than 1 second. When there are 10,000 users,[3] it costs less than 78 seconds, which is still reasonable as Setup is a one-time process.

2. https://archive.ics.uci.edu/ml/machine-learning-databases/bag-of-words/

3. We may use a large value of b in the system setup to accommodate the need of adding extra users after the setup.

8.2 Store

Fig. 6b shows the time cost of Store phase, i.e., constructing the index. For the NIPS dataset, we measure the time of building the index for 20, 40, 60, 80 and 100 percent of the documents in the dataset, respectively. Though Store phase costs more time than other phases, it is a one-time process done by the user before uploading documents onto the server.

The time cost in Store phase exhibits quadratic growth with respect to the number of keywords when fixing the number of documents. When the number of documents is 1500 and the number of keywords is 50, it costs about 68.0954 seconds.

8.3 TokenGen

In the TokenGen phase, the computation that a user undertakes costs about 1 millisecond and the counterpart that the server undertakes costs roughly 6 milliseconds. They are both independent on the number of documents and keywords.

8.4 Search

We devide our Search phase into three steps to measure its time consumption: search on the user side, search on the server side and verification on the user side.

Fig. 6c illustrates the time cost of Search on the user side. It only depends on the number of users. When there are fewer than 100 system users, it costs less than 13 milliseconds. When there are 10,000 users, it costs less than 0.14 second which is still very efficient.

Fig. 6d shows the time cost of Search phase on the server side. The time consumption exhibits quadratic growth with respect to the number of keywords since the server needs to compute witnesses of two accumulators. It takes less than 3 seconds for the NIPS dataset (1500 files) and 50 keywords.

Fig. 6e shows the time cost of verification in Search phase on the user side. The verification process consists

TABLE 1
Search and Verify Time

# of docs	π_C cmpt	π_I cmpt	search (server)	verify (user)
5000	0.4968	3.5692	4.073	0.1602
10000	1.1142	3.559	4.6872	0.3222
20000	2.3674	3.578	5.973	0.6522
39861	4.8204	3.5652	8.4406	1.2558

[1] *All durations are measured in seconds and for 100 keywords.*
[2] *# of docs denotes the number of documents. π_C cmpt and π_I cmpt denote the time on computing π_C and π_I respectively in Search phase on the server side, whose total time cost is denoted by search (server). Verify (user) denotes the time cost of the verification process in Search phase on the user side.*

of verifying the two accumulators and costs less than 50 milliseconds for the NIPS dataset (1500 files).

Due to the use of accumulator in our construction, the computational cost of the search phase on both server and user sides largely depends on the scale of the system, i.e., the number of keywords and documents. To further test the performance of our scheme, we used a larger dataset, the Enron Emails dataset, with 39861 documents and 100 keywords in the experiment. The result is presented in Table 1. From the experimental data, we can see that the cost of computing π_C is approximately proportional to the number of documents and it takes about 4.82 seconds for 39861 documents. For the computation of π_I, we let the server pre-compute all the possible results of the internal product, which makes the real-time computation only related to the number of keywords. For 100 keywords, the computation of π_I takes about 3.6 seconds. For the verification of the searching result, it takes about 1.25 seconds for 39861 documents on the user side. We should note that in the above experiment we used a normal PC to simulate the cloud server while in reality the cloud server is much more powerful.

8.5 Comparison

Table 2 compares some existing SSE schemes with ours.

Most schemes perform only 1 communication round while our scheme needs 2 communication rounds since we embed the broadcast encryption into our scheme to realize the multi-user setting. As our tokens consist of a PRF value and a pair of $(label, \overline{index})$, our trapdoor computation has the optimal performance $\mathcal{O}(1)$ in comparison with r PRF values in [2], $\log(n|\Delta|)$ PRF values in [18] and l keyed hash values in [19]. Our index is a matrix of $\frac{m(m+1)}{2}$ rows, hence its index computation complexity is $\mathcal{O}(m^2)$. The indexes of [3], [4] consist of both a look-up table of m entries and an array of $\sum_{w \in \mathcal{W}} \mathcal{R}(w)$ items. [5] builds an index item for every document regardless whether the document contains the keyword when processing each keyword, so its index computation complexity is $\mathcal{O}(mn)$. [18] uses a binary tree as its index whose nodes correspond to such entries $(w, id, add/del, cnt)$ recording addition or deletion operations. w induces at least $R(w)$ such entries so the number of total entries is more than the number of documnet-keyword pairs, i.e., $\mathcal{O}(n|\Delta|)$. [19] uses an $|\epsilon|$-ary tree as the index to store the set of words, each character of any existing word can be found in a node of the tree and every existing word corresponds to a branch. Thus its index computation complexity is less than the max size of the tree $\frac{|\epsilon|^{l+1}-1}{|\epsilon|-1}$, i.e., $\mathcal{O}(|\epsilon|^l)$. Hence, our index computation complexity is smaller than that of [3], [4], [5], [19] since m is usually much smaller than n. In [6] and our scheme, given a token, the server directly finds the row corresponding to the query keywords, therefore our search complexity is $\mathcal{O}(1)$, better than that of [3], [4], [5] which traverse index items of all n documents or at least $\mathcal{R}(w)$ documents containing the keyword w. The protocol in [18] finds all tuples (l, w, id, add) in all $l \in \{0, 1, \cdots, \lfloor \log(n|\Delta|) \rfloor\}$ level, such that the corresponding deletion tuple (l, w, id, del) does not appear in levels $l' \leq l$. Its search complexity reaches $\mathcal{O}(\mathcal{R}(w)\log^3(n|\Delta|))$ by means of its SkipHole algorithm even in the worst case. [19] stores all sequence numbers of documents containing word w in its leaf node so that its search complexity is also $\mathcal{O}(1)$. Our scheme uses two RSA accumulators to check the validity of the returning documents, and the computation complexity is $\mathcal{O}(n)$ for verifying A_I and $\mathcal{O}(\mathcal{R}(w))$ for verifying A_C, which is relatively more expensive than n MACs in [5], n MACs including $\mathcal{R}(w)$ MACs verifying addition entries and at most $n - \mathcal{R}(w)$ MACs verifying deletion entries at the edge of each deletion region in [18], and $l+1$ verifications including block cipher decryption or keyed hash operations in [19].

Overall, our scheme has a comparable performance with the existing SSE schemes while realizing multi-user searching and some other desirable features such as verifiability and supporting dynamic operations.

TABLE 2
Performance Comparison

	# of rounds	trapdoor computation	index computation	search	multi-user	hide search pattern	adaptive adversary	verifiability	verifiacation cost	update
[1]	1	$\mathcal{O}(1)$	–	$\mathcal{O}(nL)$	✗	✓	–	✗	–	✗
[2]	1	$\mathcal{O}(r)$	$\mathcal{O}(n\|\Delta\|)$	$\mathcal{O}(n)$	✗	✓	✓	✗	–	✗
scheme1[3]	1	$\mathcal{O}(1)$	$\mathcal{O}(n\|\Delta\|)$	$\mathcal{O}(n)$	✗	✓	✗	✗	–	✓
scheme2[3]	2	$\mathcal{O}(1)$	$\mathcal{O}(m)+\mathcal{O}(n\|\Delta\|)$	$\mathcal{O}(n)$	✗	✓	✗	✗	–	✓
SSE-1[4]	1	$\mathcal{O}(1)$	$\mathcal{O}(m)+\mathcal{O}(n\|\Delta\|)$	$\mathcal{O}(\mathcal{R}(w))$	✗	✓	✗	✗	–	✗
SSE-2[4]	1	$\mathcal{O}(n)$	$\mathcal{O}(n\|\Delta\|)$	$\mathcal{O}(\mathcal{R}(w))$	✗	✓	✓	✗	–	✗
[5]	1	$\mathcal{O}(n)$	$\mathcal{O}(mn)$	$\mathcal{O}(\mathcal{R}(w))$	✗	✓	✗	✓	$\mathcal{O}(n)$	✗
[6]	1	$\mathcal{O}(1)$	$\mathcal{O}(m)$	$\mathcal{O}(1)$	✗	✓	✗	✓	$\mathcal{O}(\mathcal{R}(w))+\mathcal{O}(n)$	✓
[18]	1	$\mathcal{O}(\log(n\|\Delta\|))$	$\mathcal{O}(n\|\Delta\|)$	$\mathcal{O}(\mathcal{R}(w)\log^3(n\|\Delta\|))$	✗	✗	–	✓	$\mathcal{O}(n)$	✓
[19]	1	$\mathcal{O}(l)$	$\mathcal{O}(\|\epsilon\|^l)$	$\mathcal{O}(1)$	✗	✗	–	✓	$\mathcal{O}(l)$	✗
our scheme	2	$\mathcal{O}(1)$	$\mathcal{O}(m^2)$	$\mathcal{O}(1)$	✓	✓	✗	✓	$\mathcal{O}(\mathcal{R}(w))+\mathcal{O}(n)$	✓

1. *m is the number of all keywords, $\mathcal{R}(w)$ is the number of documents containing the keyword w, n is the number of stored documents, $|\Delta|$ is the number of distinct keywords per document, r is the Bloom Filter parameter of the scheme, l is the word length of the keyword in current search, $|\epsilon|$ is the size of alphabetic set whose characters are orderly organized to construct each word, L is the average length of one document.*

9 Conclusion and Open Problems

In this paper, we introduced the notion of multi-user verifiable searchable symmetric encryption, which is more practical than single-user SSE in cloud storage systems. We presented the security definitions, i.e., privacy and reliability, for MVSSE and an efficient construction. We also introduced a Universally Composable security framework for proving the security of MVSSE, and proved that our proposed scheme is UC-secure. We also presented several extensions of the proposed scheme and illustrated that the scheme could achieve better performance on conjunctive and boolean queries for both server and client. Our scheme has a limitation that the maximum number of users is determined in the system setup and we leave the construction of a scheme without such a limitation as future work. Another interesting open problem is to explore other keyword index structures that can improve the searching efficiency for multi-keyword boolean queries.

Acknowledgments

We thank the anonymous reviewers for their invaluable comments and suggestions on the paper. The work of Yi Mu is supported by the National Natural Science Foundation of China (61822202, 61872087, 61872089). The work of Robert H. Deng is supported by the AXA Research Fund.

References

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

[2] E. Goh, "Secure indexes," *IACR Cryptology ePrint Archive*, vol. 2003, 2003, Art. no. 216.

[3] Y. Chang and M. Mitzenmacher, "Privacy preserving keyword searches on remote encrypted data," in *Proc. 3rd Int. Appl. Cryptography Netw. Secur.*, 2005, pp. 442–455.

[4] R. Curtmola, J. A. Garay, S. Kamara, and R. Ostrovsky, "Searchable symmetric encryption: Improved definitions and efficient constructions," *J. Comput. Secur.*, vol. 19, no. 5, pp. 895–934, 2011.

[5] K. Kurosawa and Y. Ohtaki, "Uc-secure searchable symmetric encryption," in *Proc. 16th Int. Conf. Financial Cryptography Data Secur.*, 2012, pp. 285–298.

[6] K. Kurosawa and Y. Ohtaki, "How to update documents verifiably in searchable symmetric encryption," in *Proc. 12th Int. Conf. Cryptology Netw. Secur.*, 2013, pp. 309–328.

[7] D. Cash, S. Jarecki, C. Jutla, H. Krawczyk, M.-C. Roşu, and M. Steiner, "Highly-scalable searchable symmetric encryption with support for boolean queries," in *Proc. Annu. Cryptology Conf.*, 2013, pp. 353–373.

[8] S. Kamara and T. Moataz, "Boolean searchable symmetric encryption with worst-case sub-linear complexity," in *Proc., Part III 36th Annu. Int. Conf. Theory Appl. Cryptographic Tech.*, 2017, pp. 94–124.

[9] C. Bösch, Q. Tang, P. H. Hartel, and W. Jonker, "Selective document retrieval from encrypted database," in *Proc. Int. Conf. Inf. Secur.*, 2012, pp. 224–241.

[10] J. Yu, P. Lu, Y. Zhu, G. Xue, and M. Li, "Toward secure multikeyword top-k retrieval over encrypted cloud data," *IEEE Trans. Dependable Secure Comput.*, vol. 10, no. 4, pp. 239–250, Jul./Aug. 2013.

[11] D. Boneh, G. D. Crescenzo, R. Ostrovsky, and G. Persiano, "Public key encryption with keyword search," in *Proc. Int. Conf. Theory Appl. Cryptographic Tech.*, 2004, pp. 506–522.

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

[13] K. Liang, X. Huang, F. Guo, and J. K. Liu, "Privacy-preserving and regular language search over encrypted cloud data," *IEEE Trans. Inf. Forensics Secur.*, vol. 11, no. 10, pp. 2365–2376, 2016.

[14] R. Chen, Y. Mu, G. Yang, F. Guo, and X. Wang, "Dual-server public-key encryption with keyword search for secure cloud storage," *IEEE Trans. Inf. Forensics Secur.*, vol. 11, no. 4, pp. 789–798, 2016.

[15] R. Chen, Y. Mu, G. Yang, F. Guo, X. Huang, X. Wang, and Y. Wang, "Server-aided public key encryption with keyword search," *IEEE Trans. Inf. Forensics Secur.*, vol. 11, no. 12, pp. 2833–2842, Dec. 2016.

[16] R. Canetti, "Universally composable security: A new paradigm for cryptographic protocols," in *Proc. 42nd Annu. Symp. Found. Comput. Sci.*, 2001, pp. 136–145.

[17] D. Boneh, C. Gentry, and B. Waters, "Collusion resistant broadcast encryption with short ciphertexts and private keys," in *Proc. 25th Annu. Int. Cryptology Conf.*, 2005, pp. 258–275.

[18] E. Stefanov, C. Papamanthou, and E. Shi, "Practical dynamic searchable encryption with small leakage," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2014, vol. 14, pp. 23–26.

[19] Q. Chai and G. Gong, "Verifiable symmetric searchable encryption for semi-honest-but-curious cloud servers," in *Proc. IEEE Int. Conf. Commun.*, 2012, pp. 917–922.

Xueqiao Liu received the BS. degree from Hefei University of Technology, China, in 2011 and the MS degree from Jinan University, China, in 2014. She is currently working toward the PhD degree in the School of Computing and Information Technology, University of Wollongong, Australia. Her major research interests include cryptography, data security and privacy in cloud computing and network security.

Guomin Yang received the PhD degree in computer science from the City University of Hong Kong, in 2009. He was a research scientist with the Temasek Laboratories, National University of Singapore from 2009 to 2012. He is currently a senior lecturer with the School of Computing and Information Technology, University of Wollongong. His research mainly focuses on applied cryptography and network security. He received the Australian Research Council Discovery Early Career Researcher Award in 2015. He is a senior member of the IEEE.

Yi Mu received the PhD degree from Australian National University, in 1994. He is currently a full professor with Fujian Normal University, China. Prior to his position with Fujian Normal University, he was a full professor with the University of Wollongong, Australia. His research Interest includes cryptography and information security. He is a senior member of the IEEE.

Robert H. Deng has been a professor with the School of Information Systems, Singapore Management University since 2004. Prior to this, he was principal scientist and manager of Infocomm Security Department, Institute for Infocomm Research, Singapore. His research interests include data security and privacy, multimedia security, network and system security. He has served/is serving on the editorial boards of many international journals in security, such as the *IEEE Transactions on Information Forensics and Security*, the *IEEE Transactions on Dependable and Secure Computing*, the *International Journal of Information Security*, and the *IEEE Security and Privacy Magazine*. He is the chair of the Steering Committee of the ACM Asia Conference on Computer and Communications Security (ASIACCS). He received the University Outstanding Researcher Award from the National University of Singapore in 1999 and the Lee Kuan Yew fellow for research excellence from the Singapore Management University, in 2006. He was named Community Service Star and Showcased Senior Information Security Professional by (ISC)2 under its Asia-Pacific Information Security Leadership Achievements program in 2010. He is a fellow of the IEEE.