Achieving Practical Symmetric Searchable Encryption With Search Pattern Privacy Over Cloud

Yandong Zheng, Rongxing Lu, *Senior Member, IEEE*, Jun Shao, Fan Yin, and Hui Zhu, *Senior Member, IEEE*

Abstract—Dynamic symmetric searchable encryption (SSE), which enables a data user to securely search and dynamically update the encrypted documents stored in a semi-trusted cloud server, has received considerable attention in recent years. However, the search and update operations in many previously reported SSE schemes will bring some additional privacy leakages, e.g., search pattern privacy, forward privacy and backward privacy. To the best of our knowledge, none of the existing dynamic SSE schemes preserves the search pattern privacy, and many backward private SSE schemes still leak some critical information, e.g., the identifiers containing a specific keyword currently in the database. Therefore, aiming at the above challenges, in this article, we design a practical SSE scheme, which not only supports the search pattern privacy but also enhances the backward privacy. Specifically, we first leverage the k-anonymity and encryption to design an obfuscating technique. Then, based on the obfuscating technique, pseudorandom function and pseudorandom generator, we design a basic dynamic SSE scheme to support single keyword queries and simultaneously achieve search pattern privacy and enhanced backward privacy. Furthermore, we also extend our proposed scheme to support more efficient boolean queries. Security analysis demonstrates that our proposed scheme can achieve the desired privacy properties, and the extensive performance evaluations also show that our proposed scheme is indeed efficient in terms of communication overhead and computational cost.

Index Terms—Dynamic SSE, search pattern privacy, enhanced backward privacy, boolean query

1 Introduction

As the data generated by the Internet of Things (IoT), social media and the web continue to increase, the global big data market will grow from $18.3bn in 2014 to an incredible $92.2bn by 2026 [1]. Such an explosive growth of data motivates an increasing number of individuals and organizations to outsource data to the powerful cloud [2], [3]. Meanwhile, as the data in some fields (e.g., eHealthcare) contain some private information and at the same time the cloud servers may not be fully trusted, data should be encrypted before being outsourced to the cloud. However, directly outsourcing encrypted data inevitably hides the characteristics of the data such as the keywords in the documents, and thus makes it challenging for the data user to search the outsourced documents meeting some criteria (e.g., "*return all documents containing keyword* w"). Although the data user can download each encrypted document and check whether it satisfies the search criteria or not, this approach is inefficient and impractical in terms of computational cost and communication overhead.

In order to achieve much more efficient search over outsourced encrypted documents, Song *et al.* [4] proposed the first searchable encryption scheme. After that, Goldwasser *et al.* [5] and Garg *et al.* [6] respectively introduced the fully homomorphic encryption and ORAM based searchable encryption schemes. Although both of the two schemes can achieve highly secure searchable encryption, the huge computational cost in the ORAM technique and fully homomorphic encryption technique makes the search efficiency in such schemes not desirable. Then, in order to balance the security and search efficiency of the searchable encryption schemes, symmetric searchable encryption (SSE) was proposed, which improves the search efficiency at the cost of small leakage including access pattern and search pattern. The access pattern reveals which documents are returned in a query and the search pattern leaks which search queries refer to the same keyword. After the first SSE was proposed in [4], Curtmola *et al.* [7] defined the security model of SSE, i.e., adaptive security of SSE, and introduced the first inverted index based SSE scheme. The inverted index technique builds a map from each keyword to the documents' identifiers matching it, which is an important basis for many subsequent works including our work in this paper.

- *Y. Zheng and R. Lu are with the Faculty of Computer Science, University of New Brunswick, Fredericton, NB E3B 5A3, Canada. E-mail: {yzheng8, rlu1}@unb.ca.*
- *J. Shao is with the School of Computer and Information Engineering, Zhejiang Gongshang University, Hangzhou, Zhejiang 310018, China. E-mail: chn.junshao@gmail.com.*
- *F. Yin is with the Information Security and National Computing Grid Laboratory, Southwest Jiaotong University, Chengdu, Sichuan 610031, China. E-mail: yinfan519@gmail.com.*
- *H. Zhu is with the State Key Laboratory of Integrated Services Networks, Xidian University, Xi'an, Shaanxi 710071, China. E-mail: zhuhui@xidian.edu.cn.*

Manuscript received 21 Oct. 2019; revised 2 Mar. 2020; accepted 27 Apr. 2020. Date of publication 4 May 2020; date of current version 15 June 2022. (Corresponding author: Rongxing Lu.) Recommended for acceptance by J. Liu. Digital Object Identifier no. 10.1109/TSC.2020.2992303

Although the aforementioned proposals can support efficient search over encrypted data, most of them are static SSE and thus they cannot support the dynamic update of the outsourced encrypted data. Aiming at the dynamic update, Kamara *et al.* [8] introduced the first dynamic SSE scheme with sub-linear search time and followed by several other dynamic SSE schemes [9], [10]. However, due to the update operations, the dynamic SSE schemes have more potential leakages than the static SSE schemes. In specific, the addition operation may reveal the information regarding whether the current updated keyword was searched in previous queries, i.e., forward privacy. Similarly, the current search query may reveal the documents that match the current search keyword but have been deleted before, i.e., backward privacy.

Forward privacy was first introduced in [11] and it became increasingly important since Zhang *et al.* presented a file-injection attack in [12]. Such an attack is particularly effective, especially when the dynamic SSE schemes are not forward private [12]. However, as pointed out in [12], it is difficult to conduct the attack in the "closed systems", as all documents in the "closed systems" are outsourced by the client and it is almost impossible for an adversary to inject documents into the systems. As for the backward privacy, it was first formally defined by Bost *et al.* [13]. In specific, they defined three types of backward privacy, i.e., type-I, type-II, and type-III backward privacy. Among them, the type-I backward privacy is the most secure one, and for a specific keyword w, it only leaks the number of previous updates associated with w, the identifiers containing w currently in the database, and when each of such documents was inserted. At the same time, Bost *et al.* and Chamani *et al.* respectively constructed a type-I backward private SSE scheme in [13] and [14].

Nevertheless, two challenges in the dynamic SSE schemes still have not been well addressed. The first one is that none of the existing dynamic SSE schemes preserves the search pattern privacy in an efficient way. Though ORAM or fully homomorphic encryption based searchable encryption schemes can achieve search pattern privacy, both of them are inefficient and impractical in terms of communication overhead and computational cost. The second one is that the current backward private SSE schemes still leak some critical information as mentioned above even if they have achieved type-I backward privacy. Thus, aiming at the above challenges, in this paper, we design a practical SSE scheme with search pattern privacy, forward privacy and enhanced backward privacy. Besides these privacy properties, our proposed scheme can be easily extended to support efficient boolean queries, Specifically, our contributions are four-fold as follows.

- First, we leverage k-anonymity technique and encryption technique to design an obfuscating technique, which serves as the core idea of our proposed scheme for privacy preservation.
- Second, based on the obfuscating technique, pseudorandom function and pseudorandom generator, we design a basic dynamic SSE scheme to support single keyword queries and achieve search pattern privacy, forward privacy and enhanced backward privacy, where the enhanced backward privacy leaks much less information than type-I backward privacy.

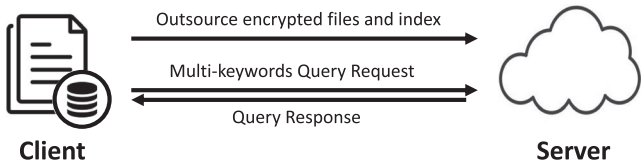


Fig. 1. System model under consideration.

- Third, based on our basic scheme, we propose an extended SSE scheme supporting efficient boolean queries.
- Finally, we analyze the security of our proposed scheme and show that it achieves desired privacy properties. And at the same time, we conduct extensive experiments to evaluate its performance, and the results show that it is indeed efficient in terms of communication overhead and computational cost.

The remainder of this paper is organized as follows. In Section 2, we introduce our system model, security model and design goal. Then, we describe some preliminaries in Section 3. In Section 4, we present our scheme, followed by security analysis and performance evaluation in Sections 5 and 6, respectively. Finally, we describe the related work in Section 7 and draw our conclusion in Section 8.

2 Models and Design Goal

In this section, we formalize our system model, security model, and identify our design goal.

2.1 System Model

In our system model, we consider a typical dynamic SSE scheme with support for both single keyword queries and boolean queries, which consists of two entities, i.e., one client and one (cloud) server, as shown in Fig. 1.

• Client: The client collects N documents with identifiers $\mathbb{ID} = \{id_1, id_2, \ldots, id_N\}$. Each document id_i contains a set of keywords $\mathbb{W}_i$, which is a subset of the collection of all keywords $\mathbb{W} = \{w_1, w_2, \ldots, w_d\}$, i.e., $\mathbb{W}_i \subseteq \mathbb{W}$. Due to the limited computational capability and storage space, the client stores his/her documents in the cloud server. As these documents may contain some sensitive information and the cloud server is not fully trusted, the client tends to encrypt them before outsourcing them to the server. Then, the client can access these documents by performing single keyword queries or boolean queries, e.g., a single keyword query may be "*return all documents that contain keyword* w_i", and a boolean query may be a conjunctive query like "*return documents that simultaneously contain a set of keywords, e.g.,* $\{w_1, w_2\}$".

A straightforward method to deal with keyword queries is to traverse all encrypted documents and return documents that meet the queried criteria. However, the overhead of this method is too large. In order to improve the query efficiency, in this paper, the client builds an inverted index database DB for his/her keywords. As shown in Fig. 2, each data record in the index database DB corresponds to a keyword $w_i \in \mathbb{W}$ and a collection of documents' identifiers containing w_i, denoted by $\mathbb{ID}_i$. After that, the client will encrypt DB and outsource it to the cloud server. With this index database, the client can efficiently access and maintain the documents in the cloud server. At the same time, both the

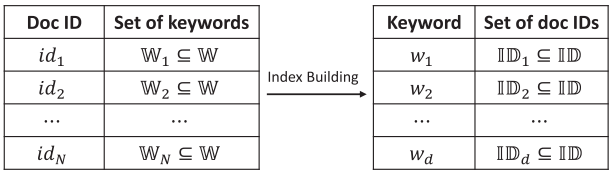


Fig. 2. Inverted index database building.

encrypted documents and the inverted index database can support users' dynamic update, i.e., adding or deleting some keywords from a given document.

• Server: The (cloud) server is powerful in both computational capability and storage space. It is responsible for storing encrypted documents along with encrypted inverted index database outsourced by the client, and processing the keyword query requests from the client. In specific, on receiving a keyword query request, the cloud server finds out all documents' identifiers satisfying the queried criteria and returns them to the client.

2.2 Security Model

In the security model of our dynamic SSE scheme, we consider the client is *honest*, i.e., he/she will follow the protocol sincerely. At the same time, the cloud server is considered to be *honest-but-curious*. That is, the cloud server will honestly follow the protocol to store the encrypted documents together with the inverted index database and deal with the keyword query requests from the client, while it may be curious about some private information, such as the plaintexts of documents and inverted index database stored in the server, as well as the queried keywords in the query requests. Besides, our system is designed to be a "closed system" and each data record must be encrypted by the client with the secret key, so the cloud server cannot launch the file-injection attack of [12] due to the difficulty of injecting attack documents. Besides the above security requirements, we aim to preserve three types of privacy in our dynamic SSE scheme, i.e., search pattern privacy, forward privacy and enhanced backward privacy. The details on these three types of privacy will be described in Section 3. Note that there may be other active attacks such as data pollution attack and deniable of service attack, however, as we focus on privacy preservation in this work, those attacks are beyond the scope of this paper, and will be discussed in our future work.

2.3 Design Goal

In this work, our goal is to design a practical and privacy-preserving dynamic SSE scheme. Specifically, the following objectives should be satisfied.

- *Privacy preservation:* The basic security requirement of our proposed scheme is privacy preservation. That is, the data stored in the cloud server including encrypted documents and encrypted inverted index database should be privacy-preserving, and the queried keywords in the query requests should be privacy-preserving. At the same time, the proposed scheme should satisfy search pattern privacy, forward privacy and enhanced backward privacy, as described in Section 2.2.
- *Efficiency:* In order to achieve the above privacy requirement, additional communication overhead and computational cost will be incurred. Specifically, keyword search queries over encrypted inverted index database will increase the computational overhead at both the client side and the server side, and bring additional communication overhead between the client and the server. Thus, in the proposed scheme, we aim to minimize the communication overhead and computational cost during the keyword search queries.

3 Preliminaries

In this section, we first introduce the formal definition and privacy concerns of dynamic SSE schemes. Then, we briefly review the pseudorandom function and pseudorandom generator techniques, which will be used in our proposed scheme.

3.1 Dynamic SSE

A dynamic SSE scheme $\Sigma = (Setup, Search, Update)$ is run between the client and the server, and consists of one algorithm $Setup$ and two protocols $Search$, $Update$.

$Setup(\lambda)$: Given a security parameter λ, the setup algorithm generates a secret key K for the client and an encrypted index database EDB for the cloud server. At the same time, it outputs the local state of the client σ.

$Search(K, \sigma, w_i; \mathrm{E}DB)$: The search protocol is used for performing keyword queries over the database and is run between the client and the server. In this protocol, the client and the server respectively inputs (K, σ, w_i) and EDB, which denotes that the client intends to search all documents that contain keyword w_i over EDB. As the output, the server protocol returns the collection of documents' identifiers containing w_i, i.e., $\mathbb{ID}_i$, to the client.

$Update(K, \sigma, id_j, w_i, op; \mathrm{E}DB)$: The update protocol, run between the client and the server, is used to update the database EDB. In this protocol, the client inputs $(K, \sigma, id_j, w_i, op)$ to update a keyword w_i in the document id_j and the update operation op is either addition or deletion.

Correctness: Generally speaking, a dynamic SSE scheme is correct if and only if all search queries can return correct query results, e.g., the search query for w_i can return exact $\mathbb{ID}_i$. As for the formal definition of the correctness of the dynamic SSE scheme, we refer readers to [10] for details.

Security: As described in [13], the security of the dynamic SSE scheme is measured by a leakage function $\mathcal{L} = (\mathcal{L}^{Setup}, \mathcal{L}^{Search}, \mathcal{L}^{Update})$, where $\mathcal{L}^{Setup}$, $\mathcal{L}^{Search}$ and $\mathcal{L}^{Update}$ are leakages with respect to the setup algorithm, search protocol and update protocol. Informally, a dynamic SSE scheme is secure if and only if it reveals nothing to the adversarial server except for the leakage function $\mathcal{L}$. This is formally captured by a real-world and ideal-world experiment defined as follows.

Definition 1 (Adaptive Security of Dynamic SSE [13]). *A dynamic SSE scheme $\Sigma = (Setup, Search, Update)$ is adaptively secure with respect to the leakage function $\mathcal{L}$ iff for all PPT adversaries Adv issuing polynomial number of queries q, there exists a PPT simulator Sim such that $|Pr[Real^{\Sigma}_{Adv}(\lambda, q) = 1] - \Pr[Ideal^{\Sigma}_{Adv,Sim,\mathcal{L}}(\lambda, q) = 1]|$ is negligible in λ,*

where $\mathrm{Real}^{\Sigma}_{Adv}(\lambda, q)$ *and* $\mathrm{Ideal}^{\Sigma}_{Adv,Sim,\mathcal{L}}(\lambda, q)$ *are two games defined as follows.*

- $\mathrm{Real}^{\Sigma}_{Adv}(\lambda, q)$: In the real-world game, the adversary Adv first gets back an index database EDB generated by running $Setup(\lambda)$ algorithm. Then, Adv can search/update the database q times by calling the search/update protocol, where q is a polynomial number. At the same time, Adv observes the real transcripts generated in the search/update protocol and outputs a bit b.
- $\mathrm{Ideal}^{\Sigma}_{Adv,Sim,\mathcal{L}}(\lambda, q)$: In the ideal-world game, the adversary Adv obtains an EDB generated by a simulator Sim with leakage $\mathcal{L}$. Then, Adv performs the same q search/update operations as that in the real-world game by calling the simulator's search/update protocol, where q is a polynomial number. At the same time, Adv observes the simulated transcripts generated in the search/update protocol and outputs a bit b.

3.2 Privacy Concerns of Dynamic SSE

In dynamic SSE, search pattern captures the information that which queries refer to the same keyword and it is a common leakage in the existing dynamic SSE schemes [13], [14], [15]. Let $QList$ be a list of search queries and each query in $QList$ is in the form of (t, w_i), which denotes that the keyword $w_i \in \mathbb{W}$ is searched in timestamp t. Then, as described in [15], the search pattern over keyword w_i can be defined as

$$sp(w_i) = \{t | (t, w_i) \in QList\}.$$

Definition 2 (Search Pattern Privacy). *An $\mathcal{L}$-adaptively secure SSE scheme keeps search pattern privacy* iff *$\mathcal{L}^{Search}$ does not leak the search pattern information, i.e., $\{sp(w_i) | w_i \in \mathbb{W}\}$.*

Forward privacy of dynamic SSE schemes was first proposed in [11] and attracted great attention after the file-injection attack was proposed [12]. Generally speaking, the forward privacy guarantees that the keyword related to the current update operation should not be linked to keywords that were searched before. Suppose that $\mathrm{L}astTime(w_i)$ leaks the timestamp when w_i was last searched/updated. At the same time, $\mathrm{L}astTime(\mathbb{W}) = \{\mathrm{L}astTime(w_1), \ldots, \mathrm{L}astTime(w_d)\}$ leaks the last updated/searched timestamp of each $w_i \in \mathbb{W}$, but the one-to-one relationship between $\mathrm{L}astTime(w_i)$ and w_i is hidden. In other words, with $\mathrm{L}astTime(\mathbb{W})$, it is difficult for the adversary to distinguish which one is the last updated/searched timestamp of a specific keyword w_i. Then, the forward privacy can be formally defined as follows.

Definition 3 (Forward Privacy). *An $\mathcal{L}$-adaptively secure SSE scheme keeps forward privacy* iff *the update leakage function $\mathcal{L}^{Update}$ can be written as: $\mathcal{L}^{Update}(op, w_i, id_j) = \mathcal{L}'(op, id_j, \mathrm{L}astTime(\mathbb{W}))$, where op is addition or deletion and $\mathcal{L}'$ is a stateless function.*

Backward privacy of dynamic SSE ensures that search queries on keyword w_i cannot be linked to the documents that contain w_i but have been deleted previously. Bost *et al.* first formally defined the backward privacy and introduced three types of backward privacy with different leakage functions in [13], where the type-I backward privacy leaks the least information. In specific, it leaks the number and type of previous updates matching each keyword w_i, the documents' identifiers matching w_i in the current database, and when each such document was inserted [13]. In this work, we aim to further reduce the leakage of backward privacy and achieve enhanced backward privacy with less leakage than the type-I backward privacy. Specifically, the enhanced backward privacy only leaks each update operation, updated document's identifier, the documents' identifiers matching the searched keyword and $\mathrm{L}astTime(\mathbb{W})$. However, different from the leakages defined in the type-I backward privacy, the leakages in the enhanced backward privacy cannot be matched to a specific keyword, which makes these leakages become trivial for adversary. Thus, the enhanced backward privacy leaks less information than the type-I backward privacy and can be formally defined as follows.

Definition 4 (Enhanced Backward Privacy). *An $\mathcal{L}$-adaptively secure SSE scheme keeps the enhanced backward privacy* iff *the leakage function $\mathcal{L}$ satisfies that $\mathcal{L}^{Update}(op, w_i, id_j) = \mathcal{L}'(op, id_j, \mathrm{L}astTime(\mathbb{W}))$, and $\mathcal{L}^{Search}(w_i) = \mathcal{L}''(\mathbb{ID}_i, \mathrm{L}astTime(\mathbb{W}))$, where $\mathcal{L}'$ and $\mathcal{L}''$ are two stateless functions, and $\mathbb{ID}_i$ contains the documents' identifiers matching w_i.*

3.3 Pseudorandom Function

The pseudorandom function (PRF) is a kind of random encryption function, which was first proposed by Goldreich *et al*. [16]. Let $F : \mathcal{K} \times \mathcal{X} \rightarrow \mathcal{Y}$ denote a PRF family from $\mathcal{X}$ to $\mathcal{Y}$, where $\mathcal{K}$ denotes the key space $\{0,1\}^{\lambda}$. Then, without knowing the secret key K, it is difficult to distinguish $F(K, \cdot)$ from $F'(K, \cdot)$, where K is randomly chosen from $\mathcal{K}$ and $F'(K, \cdot)$ is a random function from $\mathcal{X}$ to $\mathcal{Y}$. That is, for all PPT adversaries Adv, $|Pr[\mathrm{A}dv^{F(K,\cdot)}(1^{\lambda}) = 1] - \Pr[\mathrm{A}dv^{F'(K,\cdot)}(1^{\lambda}) = 1]|$ is negligible in λ.

3.4 Pseudorandom Generator

Let $G : \mathcal{X} \rightarrow \mathcal{Y}$ denote a pseudorandom generator (PRG). Given a seed $x \in \mathcal{X}$, G can generate a long random number $G(x) \in \mathcal{Y}$. At the same time, for all PPT adversaries Adv, it is difficult to distinguish $G(x)$ from a random number x', i.e., $|Pr[\mathrm{A}dv(G(x)) = 1] - \Pr[\mathrm{A}dv(x') = 1]|$ is negligible, where x' has the same length with $G(x)$.

4 Our Proposed Scheme

In this section, we present a basic SSE scheme supporting single keyword queries. Then, based on the basic SSE scheme, we propose an extended SSE scheme supporting boolean queries. Before delving into the details, we first introduce an obfuscating technique, which serves as the core idea of both our basic SSE scheme and the extended one.

4.1 The Obfuscating Technique

Suppose that the encrypted index database (i.e., EDB) is stored as a map in the server. Each data record in EDB is a key-value pair (loc_i, c_i) associated with a specific w_i, where c_i denotes the encrypted $\mathbb{ID}_i$ and loc_i denotes c_i's location in EDB. Then, we can design an obfuscating technique to access EDB with search pattern privacy. The main idea is to access the data record with k-anonymity technique, and re-encrypt the accessed data record with a new key-value pair.

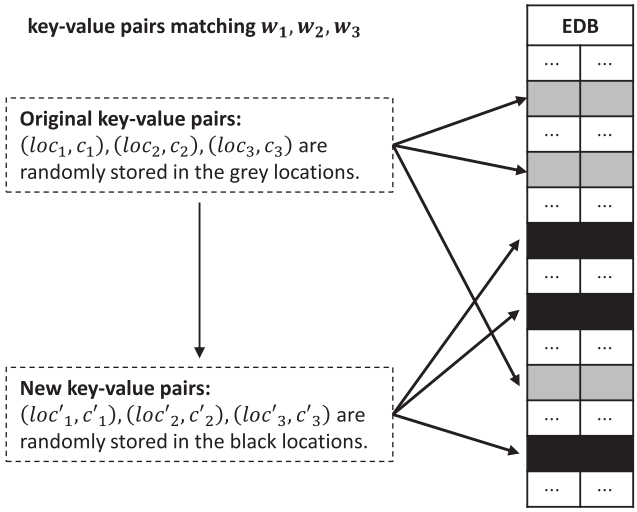


Fig. 3. An example of the obfuscating technique with $k = 3$.

In specific, when the client intends to access a data record associated with a specific keyword w_i, he/she will simultaneously access k data records matching w_i and other $k-1$ random keywords. Suppose that $\{(loc_1, c_1), \ldots, (loc_k, c_k)\}$ denotes the key-value pairs of the k accessed data records in EDB. Then, after the client accesses these data records, he/she will generate k new key-value pairs $\{(loc'_1, c'_1), \ldots, (loc'_k, c'_k)\}$ and restore them to EDB. These key-value pairs satisfy the obfuscation property. That is, without the secret key, the adversary has no idea on the one-to-one match between new key-value pairs $\{(loc'_1, c'_1), \ldots, (loc'_k, c'_k)\}$ and original key-value pairs $\{(loc_1, c_1), \ldots, (loc_k, c_k)\}$. Fig. 3 is an example of the obfuscating technique when $k = 3$. From Fig. 3, we can see that original key-value pairs $\{(loc_1, c_1), (loc_2, c_2), (loc_3, c_3)\}$ and new key-value pairs $\{(loc'_1, c'_1), (loc'_2, c'_2), (loc'_3, c'_3)\}$ look like they are randomly stored in EDB and the one-to-one match between them is hidden. In other words, the obfuscating technique preserves the privacy that which grey location (loc_i, c_i) is stored and which black location (loc'_i, c'_i) is stored. At the same time, the obfuscating technique preserves the privacy that which two locations including a grey location and a black location correspond to a same keyword w_i for $i = 1, 2, 3$. Furthermore, the obfuscating technique can well preserve the search pattern privacy and the details will be introduced in Section 5.

4.2 Basic SSE Scheme Supporting Single Keyword Queries

Based on the obfuscating technique, we present our basic SSE scheme supporting single keyword queries, which is comprised of one algorithm $Setup$, as well as two protocols $Search$ and $Update$.

4.2.1 Setup

The client is responsible for bootstrapping the scheme by running the setup algorithm, as shown in Algorithm 1. In specific, given a security parameter λ, the client first generates a secret key K for the pseudorandom function F. Then, he/she initializes two empty maps $FileCnt$ and $DictW$, which respectively store the total number of times that each keyword is accessed and the encrypted index database EDB. In the setup phase, each keyword's value in $FileCnt$ is set to be 0, i.e., $\{FileCnt[w_i] = 0 | w_i \in \mathbb{W}\}$. At the same time, for each data record $(w_i, \mathbb{ID}_i) \in EDB$, the client first represents $\mathbb{ID}_i$ to be an N-bit bitmap $B_i = b_{i,1} b_{i,2} \cdots b_{i,N}$, where N is set to be the maximum number of documents in the system and each bit $b_{i,j}$ is associated with the document id_j for $j = 1, 2, \ldots, N$. If id_j belongs to $\mathbb{ID}_i$, $b_{i,j}$ is set to be 1. Otherwise, $b_{i,j} = 0$. Then, the client encrypts (w_i, B_i) as $c_i = G(F_K(w_i, FileCnt[w_i]||1)) \oplus B_i$ and places it in $DictW$ at the location of $loc_i = F_K(w_i, FileCnt[w_i]||0)$, i.e., $DictW[loc_i] = c_i$, where G is a pseudorandom generator and generates N binary bits. Note that, $\mathbb{ID}_i$ is empty in the setup phase, so each bit $b_{i,j}$ in bitmap $B_i = b_{i,1} b_{i,2} \cdots b_{i,N}$ is set to be 0, i.e., $b_{i,j} = 0$ for $j = 1, 2, \ldots, N$. Finally, the client sends $DictW$ to the server and locally keeps K and $FileCnt$, where $DictW$ and $FileCnt$ are respectively regarded as the encrypted index database EDB and the client's local state σ.

Algorithm 1. $Setup(\lambda)$

1: $K \leftarrow Gen(1^\lambda)$
2: $FileCnt, DictW \leftarrow$ empty map
3: **for** $i = 1$ to d **do**
4: $\quad FileCnt[w_i] = 0$
5: $\quad$ Bits $B_i = b_{i,1} b_{i,2} \cdots b_{i,N} = 00 \cdots 0$
6: $\quad loc_i = F_K(w_i, FileCnt[w_i]||0)$
7: $\quad c_i = G(F_K(w_i, FileCnt[w_i]||1)) \oplus B_i$
8: $\quad DictW[loc_i] = c_i$
9: **end for**
10: $\sigma \leftarrow FileCnt$
11: $EDB \leftarrow DictW$
12: return EDB to the server

4.2.2 Search

As shown in Algorithm 2, the client can search the documents containing a specific keyword w_i as the following steps.

- *Step-1:* Given a search keyword w_i, the client randomly chooses $k-1$ distinct keywords from $\mathbb{W} \backslash \{w_i\}$. In this case, counting the queried keyword w_i, the client has k keywords in total, denoted by $\{w_{r_1}, \ldots, w_{r_k}\}$. For each keyword w_{r_l}, the client computes its location in EDB as $loc_l = F_K(w_{r_l}, FileCnt[w_{r_l}]||0)$ for $l = 1, 2, \ldots, k$. Then, the client sends these k locations $\{loc_1, loc_2, \ldots, loc_k\}$ to the server.
- *Step-2:* On receiving locations $\{loc_1, loc_2, \ldots, loc_k\}$ from the client, the server finds out the encrypted value of each location in $DictW$, i.e., $DictW[loc_l]$ for $l = 1, 2, \ldots, k$, and puts them in CList. Then, it returns CList to the client.
- *Step-3:* On receiving the CList containing k encrypted values from the server, the client uses secret key K to recover each B_{r_l} as $B_{r_l} = \mathrm{C}List[l] \oplus G(F_K(w_{r_l}, FileCnt[w_{r_l}]||1)$ for $l = 1, 2, \ldots, k$. If r_l is equal to i, the client recovers $\mathbb{ID}_i$ from B_{r_l} (i.e., B_i) to obtain all documents' identifiers matching w_i, i.e., the desirable query result.
- *Step-4:* The client updates each w_{r_l}'s value in $FileCnt$, i.e., $FileCnt[w_{r_l}] = FileCnt[w_{r_l}] + 1$. Then, with the

updated $FileCnt[w_{r_l}]$, the client generates a new key-value pair (loc_l, c_l) for (w_{r_l}, B_{r_l}), where $loc_l = F_K(w_l, FileCnt[w_{r_l}]||0)$ and $c_l = G(F_K(w_l, FileCnt[w_{r_l}]||1)) \oplus B_{r_l}$. After that, the client sends these k new key-value pairs to the server.

- *Step-5:* Finally, the server stores each new key-value pair in $DictW$, i.e., $\{DictW[loc_l] = c_l | l = 1, 2, \ldots, k\}$.

Algorithm 2. $Search(K, w_i, \sigma, k; EDB)$

Client:
1: Randomly choose $k-1$ distinct keywords from $\mathbb{W}\backslash\{w_i\}$
2: w_i and the chosen keywords are denoted by $\{w_{r_1}, \ldots, w_{r_k}\}$
3: $\mathrm{L}oclist = \{\}$
4: **for** $l = 1$ to k **do**
5: $loc_l = F_K(w_{r_l}, FileCnt[w_{r_l}]||0)$
6: $\mathrm{L}oclist = \mathrm{L}oclist \cup loc_l$
7: **end for**
8: Send $\mathrm{L}oclist = \{loc_1, \ldots, loc_k\}$ to the server
Server:
9: $\mathrm{C}List = \{\}$
10: **for** $l = 1$ to $\mathrm{L}oclist.size$ **do**
11: $\mathrm{C}List = \mathrm{C}List \cup DictW[loc_l]$
12: **end for**
13: Send $\mathrm{C}List$ to the Client
Client:
14: **for** $l = 1$ to $\mathrm{C}List.size$ **do**
15: $B_{r_l} = \mathrm{C}List[l] \oplus G(F_K(w_{r_l}, FileCnt[w_{r_l}]||1)$
16: **if** $r_l == i$ **then**
17: Recover $\mathbb{ID}_i$ from B_{r_l} (B_i), i.e., the desirable
18: result
19: **end if**
20: **end for**
21: // Generate new key-value pairs
22: LocList = {}, CList = {}
23: **for** $l = 1$ to k **do**
24: $FileCnt[w_{r_l}]$++
25: $loc_l = F_K(w_{r_l}, FileCnt[w_{r_l}]||0)$
26: $c_l = G(F_K(w_l, FileCnt[w_{r_l}]||1)) \oplus B_{r_l}$
27: $\mathrm{L}ocList = \mathrm{L}ocList \cup loc_l$
28: $\mathrm{C}List = \mathrm{C}List \cup c_l$
29: **end for**
30: Send LocList and CList to the server
Server:
31: **for** $l = 1$ to k **do**
32: $DictW[\mathrm{L}ocList[l]] = \mathrm{C}List[l]$
33: **end for**

4.2.3 Update

The update protocol is designed for updating the documents stored on the server, where the update operations include addition and deletion. That is, the client can add/delete a keyword w_i from a given document id_j as follows.

The first two steps in the update protocol are the same as that in the search protocol, so the client and the server first run the first two steps of the search protocol, which returns CList to the client. Then, the client uses the secret key K to recover each B_{r_l} as $B_{r_l} = CList[l] \oplus G(F_K(w_{r_l}, FileCnt[w_{r_l}]||1)$ for $l = 1, 2, \ldots, k$. After that, based on the update operation op, the client adds/deletes w_i from id_j by updating w_i's bitmap B_{r_l} (B_i). If op is addition, set $b_{r_l,j} = 1$. Otherwise, $b_{r_l,j} = 0$. Finally, the client and the server follow step 4 and step 5 of the search protocol to finish the remaining update protocol. That is, the client generates a new key-value pair for each (w_{r_l}, B_{r_l}) and sends them to the server. On receiving new key-value pairs from the client, the server stores them in EDB.

Algorithm 3. $Update(K, op, (w_i, id_j), \sigma, k; EDB)$

1: The client and the server run the first two steps of the search protocol, which returns CList to the client.
Client:
2: **for** $l = 1$ to $\mathrm{C}List.size$ **do**
3: $B_{r_l} = \mathrm{C}List[l] \oplus G(F_K(w_{r_l}, FileCnt[w_{r_l}]||1)$
4: **if** $r_l == i$ **then**
5: **if** $op == addition$ **then**
6: set $b_{r_l,j} = 1$ in $B_{r_l} = b_{r_l,1} b_{r_l,2} \cdots b_{r_l,N}$
7: **else if** $op == deletion$ **then**
8: set $b_{r_l,j} = 0$ in $B_{r_l} = b_{r_l,1} b_{r_l,2} \cdots b_{r_l,N}$
9: **end if**
10: **end if**
11: **end for**
12: The client and the server follow step 4 and step 5 of the search protocol to finish the remaining update protocol.

Note that, in the update protocol, one update operation allows the client to update one keyword for a given document. Then, when the client intends to add/remove a document with u keywords, the client needs to run the update protocol u times in total, i.e., add/remove these u keywords one by one. In addition, the anonymous parameter k in the update protocol is closely related to the security level of our proposed scheme, i.e., the security is strengthened as k becomes larger. At the same time, it can be dynamically changed among different search queries or update operations according to different security requirements.

4.3 Extended SSE Scheme Supporting Boolean Queries

In this subsection, we extend our basic SSE scheme to support boolean queries, i.e., search the documents whose keywords satisfy a specific boolean function $BF(w_{q_1}, \ldots, w_{q_u})$, where $\{w_{q_1}, \ldots, w_{q_u}\}$ is a set of queried keywords. In specific, the boolean function $BF(w_{q_1}, \ldots, w_{q_u})$ can support the "AND", "OR" and "NOT" operations between keywords $\{w_{q_1}, \ldots, w_{q_u}\}$. When the client intends to do such kind of boolean queries, he/she first chooses an anonymous parameter k such that k is satisfied with the security requirement and $k > u$. With k, the client selects $(k-u)$ random keywords and uses them together with u queried keywords to construct a search query. Then, he/she follows the search protocol to obtain a set of bitmaps $\{B_{q_1}, \ldots, B_{q_u}\}$ matching the queried keywords $\{w_{q_1}, \ldots, w_{q_u}\}$. With these bitmaps, the client computes the boolean function $BF(B_{q_1}, \ldots, B_{q_u})$ and the result is the bitmap of the documents' identifiers matching $BF(w_{q_1}, \ldots, w_{q_u})$. For example, when $BF(w_1, w_2, w_3) = w_1 \wedge w_2 \wedge w_3$ and the bitmaps matching $\{w_1, w_2, w_3\}$ are $\{B_1, B_2, B_3\}$, the client can compute $BF(B_1, B_2, B_3) = B_1 \wedge B_2 \wedge B_3$, which is the bitmap of documents' identifiers satisfying $w_1 \wedge w_2 \wedge w_3$. Furthermore, the client recovers the documents' identifiers from the bitmap $B_1 \wedge B_2 \wedge B_3$. At the same time, he/she follows the search protocol to generate k new key-value pairs for the k searched keywords and sends them back to the server. Since the overall procedure of the

boolean queries is the same as that of single keyword queries, the boolean queries in our extended SSE scheme are as efficient as single keyword queries in our basic SSE scheme.

Discussion. Note that both our basic scheme and extended scheme can support multiple-keyword update, i.e., the client can update and delete multiple keywords together. In specific, if the client attempts to update u keywords $\{w_{q_1}, \ldots, w_{q_u}\}$, he/she first chooses an anonymous parameter k such that k is satisfied with the security requirement and $k > u$. With k, the client selects $(k-u)$ random keywords and uses them together with u queried keywords to construct update request. Then, he/she follows the update protocol to obtain a set of bitmaps $\{B_{q_1}, \ldots, B_{q_u}\}$ matching the updated keywords $\{w_{q_1}, \ldots, w_{q_u}\}$. Finally, the client updates these bitmaps, generates k new key-value pairs for all k keywords, and sends them back to the server. The multiple-keyword update operation allows the client to add a new document or remove an obsolete document much more efficiently than the single keyword update protocol in our basic scheme because a multi-keyword update operation can simultaneously update multiple keywords instead of one keyword.

5 Security Analysis

In this section, we analyze the security of our basic SSE scheme and extended SSE scheme. In specific, we present that our proposed scheme achieves the privacy concerns described in Section 3.2, i.e., search pattern privacy, forward privacy and enhanced backward privacy.

5.1 Security Analysis of Basic SSE Scheme

• *Search Pattern Privacy:* The search pattern is the information regarding which queries refer to the same queried keyword. Since the obfuscating technique employs the k-anonymous query technique, which processes the queried keyword together with $k-1$ random keywords, it is difficult for the adversary to distinguish which keyword is the real queried keyword. Then, if the adversary attempts to link all queries matching a specific queried keyword $w_i \in \mathbb{W}$, the first step is to collect all queries containing w_i and the second step is to distinguish in which queries w_i is regarded as a random keyword and in which queries w_i is a real queried keyword. In the first step, the adversary can use a backtracking method to track all possible queries involving the keyword w_i.

Suppose that a single keyword query Q involves k keywords including a real queried keyword w_i and $k-1$ random keywords. At the same time, these keywords correspond to k key-value pairs in EDB. Then, based on them, the adversary can backtrack k previous search queries, called Q's adjacent queries, in which the k key-value pairs accessed in Q are respectively updated. For example, if the query Q accesses EDB's encrypted values $\{c_1, c_2, \ldots, c_k\}$, queries $\{Q_1, Q_2, \ldots, Q_k\}$ are Q's adjacent queries when c_j is updated in Q_j, as shown in Fig. 4. Similarly, the adversary can continue to backtrack each Q_i's adjacent queries and Q_i's adjacent queries' adjacent queries until no more queries can be further backtracked. Then, the backtracking process forms a backtracking tree with the query Q as the root node, and only one path from Q to the first query containing w_i is the exact queries sequence containing w_i. If the depth of the backtracking tree is h, the adversary has $\frac{1}{k^h}$ probability to successfully discover the correct queries sequence matching w_i. As shown in the Fig. 4, when $h=2$ and the queries sequence matching w_i is $\{Q_{11} \rightarrow Q_1 \rightarrow Q\}$, the probability of linking them together is $\frac{1}{k^2}$. Then, when $k=10$ and $h=10$, the probability will be 1×10^{-10}. Thus, it is difficult for the adversary to discover all queries involving w_i. Even if he/she discovers all queries involving w_i, the client is still not confident with the result, since w_i may be just a random keyword in some of such queries. Therefore, our basic SSE scheme can well preserve the search pattern privacy. In addition, the search pattern privacy also enables our basic scheme to resist the keyword frequency attack, i.e., the server uses the frequency of each keyword being queried to inference the queried keyword in a specific query.

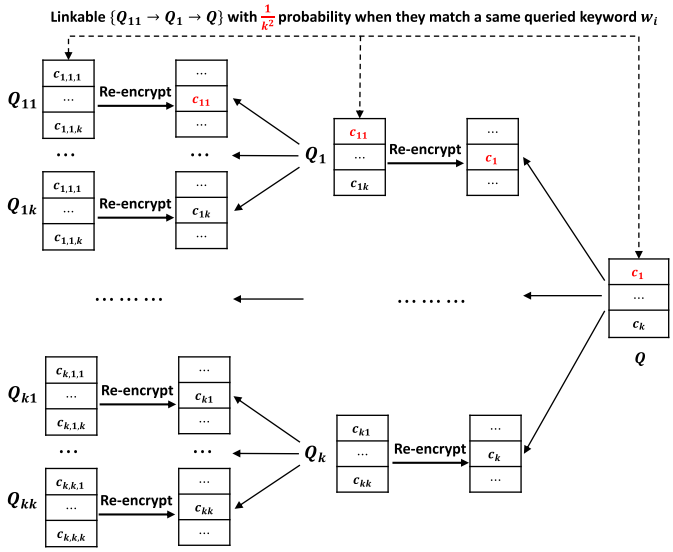


Fig. 4. The backtracking tree of the obfuscating technique when $h=2$: if the queries $\{Q_{11} \rightarrow Q_1 \rightarrow Q\}$ match a same keyword w_i, the probability of linking them together is $\frac{1}{k^2}$.

• *Forward Privacy:* The forward privacy ensures that the current update keyword should not be linked to the keywords that were searched before. In our basic SSE scheme, the update protocol is k-anonymous, i.e., during an update operation, the updated keyword will be updated together with other $k-1$ random keywords. Suppose that w_i is the updated keyword and $\{w_{r_1}, w_{r_2}, \ldots, w_{r_{k-1}}\}$ denotes $k-1$ random keywords. Then, in the update protocol, the client sends an update request together with k locations matching w_i and other $k-1$ random keywords to the server, where these k locations are generated by PRF. The security of PRF guarantees that the server can only observe k locations and has no idea on real keywords matching these locations. Similarly, in the search protocol, the client also has no idea on which k keywords are being searched. Then, it is difficult for the server to break the forward privacy by deducing the real keywords in the current update operation and previous search queries.

Without knowing the real keywords, the server just can observe the relationship of encrypted data records matching the updated/searched keywords to infer the relationship between the current keyword and previous searched

keywords. Given k encrypted data records in the current update operation, the server has $\frac{1}{k}$ probability to successfully guess which one exactly matches the updated keyword w_i. Then, the server can further find the current query's adjacent update/search queries. The adjacent query is either an update query or a search query. In case of the update query, the server has to continue to find adjacent queries' adjacent queries. However, it is more difficult for the server to match them with w_i because an adjacent query has k adjacent queries and each such query also contains k queried keyword. Even if the previous query is a search query, the adversary also cannot confirm whether w_i is regarded as a queried keyword or a random keyword in that query. At the same time, the latter case, i.e., w_i is just regarded as a random keyword, is more likely to appear under the assumption that each keyword is randomly searched. In this case, it is difficult for the server to link previous search queries matching w_i with the current update operation.

Besides, the update operation in our basic SSE scheme only leaks the type of update operation op, the identifier of the updated document id_j and all keywords' last updated timestamps $\mathrm{L}astTime(\mathbb{W})$, i.e., $\mathcal{L}^{Update}(op, w_i, id_j) = \mathcal{L}'(op, id_j, \mathrm{L}astTime(\mathbb{W}))$. Actually, the leakage of op and id_j comes from the update on the real document. In specific, the real updated document and its length change respectively leak the identifier of updated document and the update operation. Thus, based on the definition of forward privacy, our basic SSE scheme can achieve forward privacy.

• *Enhanced Backward Privacy:* The enhanced backward privacy ensures that search queries on keyword w_i cannot be linked to the documents that contain w_i but have been deleted previously. The documents' identifiers that contain w_i but have been deleted previously can be leaked in two ways. The first way is that $\mathbb{ID}_i$ contains such kind of identifiers, but it is almost impossible to happen in our basic SSE scheme. This is because the client will immediately change $\mathbb{ID}_i$ when a document deletes w_i. Thus, $\mathbb{ID}_i$ cannot contain the documents' identifiers that have been deleted. The second way is to link the current search query with previous deletion update queries, which is also difficult in practice. Since the search protocol and update protocol are extremely similar, and both of them conduct k-anonymous queries. Thus, linking an update query with previous search queries matching w_i is as difficult as linking a search query with previous update queries matching w_i. In the above discussion, we have shown that it is difficult for the server to link an update query with previous search queries. Thus, it is difficult to link a search query with previous update queries. Furthermore, it is also difficult to link a search query with previous deletion update queries.

Besides, as discussed above, the leakage function of the update operation is $\mathcal{L}^{Update}(op, w_i, id_j) = \mathcal{L}'(op, id_j, \mathrm{L}astTime(\mathbb{W}))$. At the same time, the search query leaks the documents' identifiers matching each searched keyword and all keywords' last updated timestamps, i.e., $\mathcal{L}^{Search}(w_i) = \mathcal{L}''(\mathbb{ID}_i, \mathrm{L}astTime(\mathbb{W}))$. Actually, the leakage $\mathbb{ID}_i$ comes from the search on real documents. For example, when the client recovers $\mathbb{ID}_i$, he/she needs to request the real documents by sending $\mathbb{ID}_i$ to the server, which will leak $\mathbb{ID}_i$ to the server. Thus, based on the definition of backward privacy, our basic SSE scheme satisfies enhanced backward privacy.

Theorem 1. *Suppose that F and G are secure PRF and PRG, our basic SSE scheme is adaptively-secure with leakage $\mathcal{L}^{Update}(op, w_i, id_j) = \mathcal{L}'(op, id_j, \mathrm{L}astTime(\mathbb{W}))$ and $\mathcal{L}^{Search}(w_i) = \mathcal{L}''(\mathbb{ID}_i, \mathrm{L}astTime(\mathbb{W}))$.*

Proof. In the real-world game, suppose that the server observes q transcripts in total. The transcript observed from the setup algorithm includes the initialized EDB, and each other transcript observed from the update/search protocol includes k original key-value pairs and k updated key-value pairs matching the queried keywords as described in Section 4.2. In the following, we describe our simulator Sim, which is comprised of three algorithms, i.e., $SimSetup$, $SimUpdate$ and $SimSearch$. □

• *SimSetup(λ):* In the setup algorithm, Sim first generates d key-value pairs $\{(loc_i, c_i)|i = 1, 2, \ldots, d\}$, where each loc_i is a random number in the range of F and each c_i is a random binary string with length N. For each key-value pair (loc_i, c_i), Sim sets $DictW[loc_i] = c_i$. Then, it sets EDB to be $DictW$ and sends it to the server. Finally, Sim sets the local state of the simulator to be null, i.e., $\sigma_{\mathcal{S}} = null$, and locally stores each loc_i and the timestamp that it was last updated, i.e., $\{(t, loc_i)|i = 1, 2, \ldots, d\}$.

• *SimUpdate($\sigma_{\mathcal{S}}, \mathcal{L}^{Update}(op, w_i, id_j), \mathrm{E}DB$):* In the update protocol, Sim receives the leakage function $\mathcal{L}^{Update}(op, w_i, id_j) = \mathcal{L}'(op, id_j, \mathrm{L}astTime(\mathbb{W}))$. Suppose that the involved k key-value pairs matching the queried keywords were last updated in timestamps $\{t_1, \ldots, t_k\}$, respectively. Then, Sim randomly chooses k locations $\{loc_1, loc_2, \ldots, loc_k\}$ such that the last update timestamp of each loc_l is t_l for $l = 1, 2, \ldots, k$, and sends them to the server. After receiving the response from the server, Sim randomly generates k new key-value pairs, where each location is a random number in the range of F and each value is a random binary string with length N. Finally, Sim sends them to the server, and locally stores each location and the timestamp that it was last updated.

• *SimSearch($\sigma_{\mathcal{S}}, \mathcal{L}^{Search}(w_i), \mathrm{E}DB$):* In the search protocol, Sim receives the leakage function $\mathcal{L}^{Search}(w_i) = \mathcal{L}'(\mathbb{ID}_i, \mathrm{L}astTime(\mathbb{W}))$. Sim first chooses k random locations with the same method in the SimUpdate protocol and sends them to the server. When receiving the response from the server, Sim randomly generates k new key-value pairs and sends them to the server. At the same time, Sim locally stores each location and the timestamp that it was last updated. Finally, Sim sends the $\mathbb{ID}_i$ to the server.

For the transcript in the setup algorithm, since the d key-value pairs in the real-world game are generated by the PRF F and PRG G, they are indistinguishable from the random key-value pairs in the ideal-world game. For the transcripts in the update/search protocol, the timestamps of the k locations matching the queried keywords in the ideal-world game are the same as that in the real-world game, and the documents' identifiers matching the searched/updated keyword w_i in the ideal-world game are the same as that in the real-world game. Thus, the transcripts of search/update protocol in the ideal-world game are also indistinguishable from those in the real-world game. Consequently, all the q transcripts in the ideal-world game are indistinguishable from those in the real-world game. Therefore, our basic SSE scheme is adaptively-secure with leakage $\mathcal{L}^{Update}$ and $\mathcal{L}^{Search}$. ■

5.2 Security Analysis of Extended SSE Scheme

For our extended SSE scheme, the forward privacy and enhanced backward privacy can be achieved and the detailed security analysis is the same as that in our basic SSE scheme. As for the search pattern privacy, it can be analyzed from two aspects. On the one hand, we consider the search pattern privacy of a single queried keyword, which is the same as that in the basic SSE scheme. Thus, the single keyword's search pattern privacy can be easily achieved. On the other hand, we consider the search pattern privacy when regarding all involved keywords as a whole. In this case, the search pattern means the information that which queries refer to the same set of queried keywords. Since a boolean query contains multiple keywords and the number of possible keywords combinations in a search query is large, about $2^d - \binom{d}{0} = 2^d - 1$ when at least one keyword is queried, where d is the number of keywords, the number of queries containing the same keywords set is relatively small. In specific, a single keyword query in the basic SSE scheme only contains a real queried keyword and has d possibilities, while a boolean query in the extended SSE scheme may contain i keywords $(1 \leq i \leq d)$ and has $\binom{d}{i}$ possible keywords combinations. Then, the probability that different search/update queries contain the same queried keywords set is relatively low. At the same time, even if there exist some such kind of queries, the obfuscating technique guarantees that it is difficult to link them as described in Section 5.1. Thus, from the perspective of probability, it is difficult to deduce the search pattern privacy when regarding all involved keywords as a whole. Therefore, the search pattern privacy of our extended SSE scheme is even stronger than that in our basic SSE scheme.

6 Performance Evaluation

In this section, we evaluate the performance of our proposed scheme with regard to the computational cost and communication overhead in search/update protocol, as well as the storage overhead at the client side. At the same time, we compare the computational cost of search queries between our proposed scheme and the most efficient implementation of a forward and backward private scheme so far (i.e., MITRA) [14]. Since our extended SSE scheme is based on our basic SSE scheme and has the similar performance as our basic SSE scheme, we focus on evaluating the performance of our basic SSE scheme. In specific, we implement our scheme in Java and conduct experiments on an Intel(R) Core(TM) i7-3770 CPU @3.40 GHz Windows Platform with 16 GB RAM. In our experiments, we respectively use HMAC-SHA1 as the PRF and a mersenne twister algorithm based generator as the PRG [17]. The secret key K is set to be a 160-bit random number, i.e., $|K| = 160$. For the MITRA scheme, we evaluate it with the code released by authors [18]. In addition, we evaluate our scheme using synthetic dataset. Specifically, we randomly generate 10^5 keywords and each search/update operation involves k keywords, where the value of k is set to be $10, 20, 30$. At the same time, we set the maximum number of documents in the system, i.e., N, ranges from 10^2 to 10^5. All experiments were conducted 100 times and the average is reported.

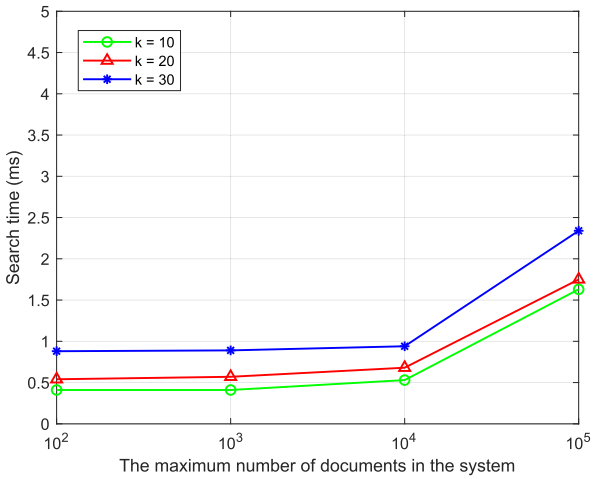
6.1 Computational Cost

Search. As described in Section 4.2, the search protocol in our scheme consists of two rounds and all of the involved computational cost is at the client side. Suppose that C_{HMAC}, C_{PRG} and C_{XOR} respectively denote the computational complexity of HMAC operation, pseudorandom number generation operation and XOR operation. Then, in the first round, the client requires $k \times \mathrm{C}_{HMAC}$ computational complexity to compute the locations associated with the current k queried keywords. In the second round, the client requires $k \times (\mathrm{C}_{HMAC} + \mathrm{C}_{PRG} + \mathrm{C}_{XOR})$ computational complexity to recover the documents' identifiers matching the queried keywords. After that, he/she needs to re-encrypt the documents' identifiers for each queried keyword and generate a new location for each encrypted documents' identifiers with $k \times (2 \cdot C_{HMAC} + \mathrm{C}_{PRG} + \mathrm{C}_{XOR})$. Thus, in the search protocol, the overall computational cost at the client side is $k \times (4 \cdot C_{HMAC} + 2 \cdot C_{PRG} + 2 \cdot C_{XOR})$. Since pseudorandom number generation operation is to generate an N-bit binary string and XOR operation is over two N-bit binary strings, C_{PRG} and C_{XOR} are related to the parameter N. Furthermore, the computational complexity of the search protocol is related to the anonymity parameter k and the maximum number of documents in the system N. In Fig. 5a, we plot the average computational cost of our search protocol versus k and N. From Fig. 5a, we can see that the search protocol is extremely efficient even if the runtime of search linearly increases as k and N. For example, when $k = 30$ and $N = 10^5$, the runtime of each search operation is just 2.4 ms.

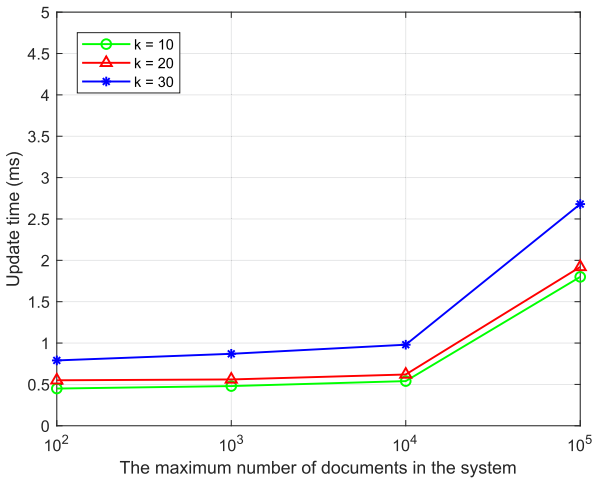
Update. As described in Section 4.2, the update protocol also consists of two rounds and it executes almost the same operations as the search protocol, i.e., retrieve the data records matching the k queried keywords, re-encrypt them and send back to the server. The only difference is that the update protocol needs to additionally add/delete the updated document identifier, but the computational cost in this operation is low compared with other operations. Thus, the computational cost of the update protocol has the same trend but a little larger than that of the search protocol as shown in Fig. 5b.

Comparison With MITRA. In the literature, existing searchable encryption schemes can be divided into two categories, i.e., the ORAM and homomorphic encryption based schemes and SSE schemes. The ORAM and homomorphic encryption based schemes can achieve the same security level with our proposed scheme, i.e., simultaneously satisfy the forward privacy, backward privacy and search pattern privacy. However, compared with most of the SSE schemes, such kinds of schemes are computationally expensive and inefficient. Obviously, our proposed scheme is a SSE scheme and it is much more efficient than the ORAM and homomorphic encryption based schemes. As for the SSE based schemes, most of them cannot achieve backward privacy and none of them can achieve search pattern privacy, so they leak more privacy information than our proposed scheme. Thus, our proposed scheme leaks the least information.

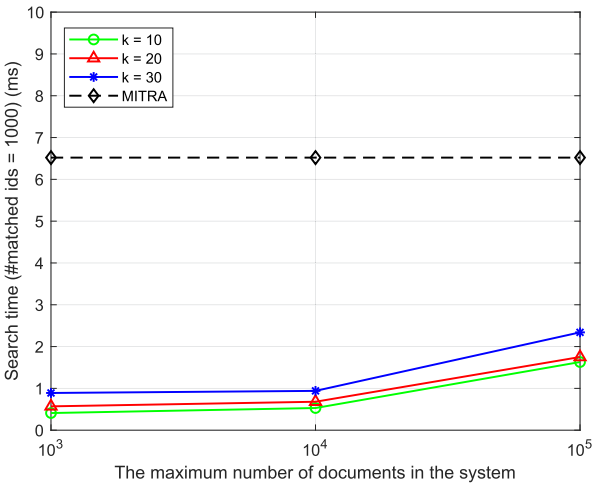
In the following, we show that our proposed scheme is extremely efficient with regard to the search operation by comparing our scheme with the most efficient implementation of a forward and backward SSE scheme, i.e., MITRA. In specific, we compare the search efficiency between our proposed scheme and MITRA when variable N ranges from 10^3 to 10^6 and the size of the matched documents is set to be 1000 as shown in Fig. 5c. From Fig. 5c, we can see that our proposed scheme has better search performance than the



(a) Search time vs the anonymous parameter k and the maximum number of documents N in the system



(b) Update time vs the anonymous parameter k and the maximum number of documents N in the system



(c) Search time comparison between our basic SSE scheme and MITRA when the number of matched documents N is 1000

Fig. 5. Performance evaluation of our basic SSE scheme.

MITRA when the size of the matched documents is 1000. Therefore, our scheme achieves the best search efficiency.

Overall, our scheme not only achieves the best search efficiency, but also leaks the least information.

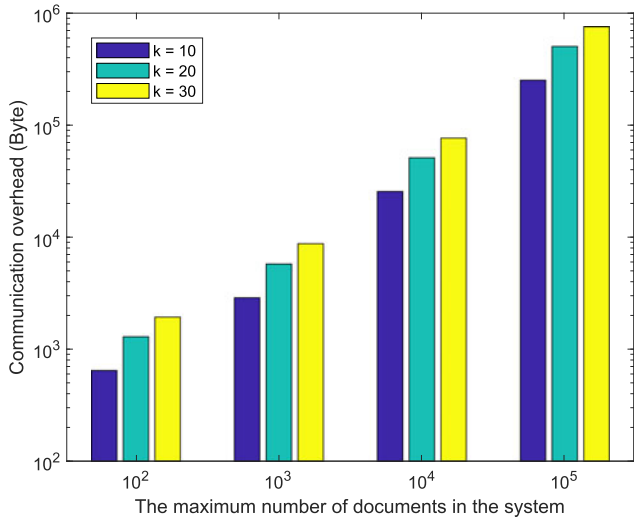


Fig. 6. Communication overhead vs the anonymous parameter k and the maximum number of documents N in the system.

6.2 Communication Overhead

As described in Section 4.2, the search protocol and update protocol are two-round protocols and they involve the same communication overhead. In specific, in the first round, the client sends the locations of the queried keywords to the server. As HMAC-SHA1 is employed as PRF to generate these locations, the size of each location is 160 bits and the overall communication overhead for k locations is $k \times 160$ bits. At the same time, the server needs to return the encrypted values matched the requested locations to the client and each of them is N bits. Hence, the communication overhead for the encrypted values is $k \times N$ bits. In the second round, the client returns k pairs new encrypted values and locations to the server and the communication overhead is $k \times (160 + N)$ bits. Thus, the overall communication overhead in search/update protocol is $k \times (2 \times 160 + 2 \times N)$. Fig. 6 plots the communication overhead of our search/update protocol versus k and N. From Fig. 6, we can see that the communication overhead linearly increases as k and N.

6.3 Client Storage

In our scheme, the client needs to locally store a 160-bit secret key and a map $FileCnt$ containing d key-value pairs, where the value in the $FileCnt$ is the number of times that each keyword is accessed (i.e., searched/updated). Suppose that the maximum number of times that a keyword is accessed is set to be N_{access}. Then, the storage overhead at the client side is $(160 + d \times \log_2 N_{access})$ bits. When $d = 10^4$ and $N_{access} = 10^7$, the client's storage overhead is roughly 8.77 KB.

7 Related Work

With the explosive growth of data and enhancement of privacy awareness, more and more individuals and organizations choose to outsource encrypted data to the cloud. At the same time, in order to efficiently access and maintain the outsourced encrypted data, they expect to deploy a kind of encryption that can achieve privacy-preserving search and update over encrypted data, i.e., searchable encryption, which enables search over outsourced encrypted data and

has attracted considerable attention from academia and industry. For the searchable encryption, security and efficiency are two main concerns and need to be balanced. In the literature, ORAM technique was deployed to design highly secure searchable encryption schemes, which can preserve all security concerns of encrypted data [6], [19]. Similarly, fully homomorphic encryption together with functional encryption also can be used to design highly secure searchable encryption [5]. However, the overhead of such schemes is too large and not practical.

SSE attempts to improve the search efficiency of searchable encryption scheme at the cost of small leakage including search pattern and access pattern. In 2000, Song *et al.* [4] proposed the first symmetric searchable encryption scheme and the search time over one document is linear to the length of the document. Then, Curtmola *et al.* [7] first defined the security of SSE, i.e., adaptive security of SSE and introduced the first inverted index based SSE scheme with sublinear search time. The inverted index technique builds a map from each keyword to the documents' identifiers matching it, which is an important basis for many subsequent works.

However, the aforementioned proposals are static SSE schemes and they cannot support dynamical update of outsourced encrypted data. Compared with static SSE schemes, dynamic SSE schemes have rich functionality and are more desirable in practice. In 2012, Kamara *et al.* [8] introduced the first dynamic SSE scheme with sub-linear search time, but this scheme reveals the updated document's unique keywords. Then, Kamara and Papamanthou [9] designed a new dynamic SSE scheme to overcome the limitation in [8]. At the same time, Cash *et al.* [10] designed a dynamic SSE scheme for very-large databases.

Nevertheless, dynamic SSE schemes have more leakages than static SSE schemes including forward privacy and backward privacy. Forward privacy guarantees that the current update operation cannot be linked to previous search queries, which first introduced in [11]. Then, Stefanov *et al.* [20] introduced an ORAM based forward private scheme. Bost *et al.* [21] presented the formal definition of forward privacy and proposed an insertion-only scheme with optimal search and update complexity. Zhang *et al.* [12] presented a file-injection attack to reveal search queries by injecting few documents, and this attack is particularly effective, especially when the dynamical SSE is not forward private. From then on, the forward privacy becomes increasingly important in dynamic SSE. However, the authors pointed out that although the file-injection attack is effective and can be conducted easily in some system, it may be much more difficult to conduct such attack in the "closed systems". This is because all documents in the "closed systems" are outsourced by the client and it is almost impossible for an adversary to inject documents into the systems, e.g., our proposed scheme.

The backward privacy guarantees that the current search keyword cannot be linked to the documents that match the search keyword but have been deleted before. Stefanov *et al.* [20] first introduced the notion of backward privacy to capture the leakage associated with deleted entries, but they neither described the definition of backward privacy nor presented a backward private SSE. Bost *et al.* presented a formal definition of backward privacy with three different types of leakages from most to least secure, and constructed four backward private SSE schemes that achieve different types of backward privacy [13]. Based on the work in [13], Chamani *et al.* [14] proposed three back-private constructions which improved the results of [13] in several ways. At the same time, Sun *et al.* [15] adopted the symmetric puncturable encryption technique to design a practical and non-interactive backward private SSE scheme. Recently, Zuo *et al.* proposed a dynamic SSE scheme with stronger backward privacy [22].

The above work focuses on the single keyword query. Although the single keyword query based SSE schemes can be easily extended to support complex queries, they are not efficient in complex queries. Thus, much of work on SSE focuses on supporting more complex query expressions such as multi-keyword query, boolean query, conjunctive query, disjunctive query and so on. Cash *et al.* [23] designed a SSE scheme to support conjunctive query and even boolean query. Faber *et al.* [24] extended the scheme in [23] to support more complex queries such as range query, substring query, wildcard query and phrase query. Pappas *et al.* [25] proposed a solution to support a rich query set including arbitrary boolean query and free keyword searches etc. Fisch *et al.* [26] proposed a SSE scheme to support boolean query and range query with sub-linear search time. Kamara *et al.* [27] and Lai *et al.* [28] respectively proposed a solution for disjunctive query and conjunctive query. Recently, Zuo *et al.* [29] designed two dynamic SSE schemes to support range queries and Shao *et al.* [30] proposed a verifiable scheme to support conjunctive and fuzzy queries.

8 Conclusion

In this paper, we have designed a practical SSE scheme with search pattern privacy, forward privacy and enhanced backward privacy, and extended it to support boolean queries. In specific, we leveraged k-anonymity and encryption technique to design an obfuscating technique. Then, based on the obfuscating technique, we deployed pseudorandom function and pseudorandom generator to design a single keyword queries based dynamic SSE scheme, and extended it to support efficient boolean qiueries. At the same time, we analyzed the security of our scheme and showed that the proposed scheme achieves desired security properties, i.e., search pattern privacy, forward privacy, and enhanced backward privacy. Besides, we conducted extensive experiment to evaluate the performance of our proposed scheme and the results showed that the proposed scheme is efficient in terms of communication overhead and computational cost. Our future work is to further improve the efficiency of search and update operations without weakening SSE scheme's privacy preservation.

Acknowledgments

This research was supported in part by NSERC Discovery Grants (04009), ZJNSF LZ18F020003, NSFC U1709217, National Key Research and Development Program of China (2017YFB0802200), National Natural Science Foundation of China (61972304), and Natural Science Foundation of Shaanxi Province (2019ZDLGY12-02).

REFERENCES

[1] "Data-sharing and cloud: A big data match made in heaven," [Online]. Available: https://www.computerweekly.com/blog/Ahead-in-the-Clouds

[2] Y. Zheng, R. Lu, B. Li, J. Shao, H. Yang, and K. R. Choo, "Efficient privacy-preserving data merging and skyline computation over multi-source encrypted data," *Inf. Sci.*, vol. 498, pp. 91–105, 2019.

[3] Y. Zheng, R. Lu, and J. Shao, "Achieving efficient and privacy-preserving K-NN query for outsourced ehealthcare data," *J. Med. Syst.*, vol. 43, no. 5, pp. 123:1–123:13, 2019.

[4] 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.

[5] S. Goldwasser *et al.*, "Multi-input functional encryption," in *Proc. Annu. Int. Conf. Theory Appl. Cryptogr. Techn.*, 2014, pp. 578–602.

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

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

[8] S. Kamara, C. Papamanthou, and T. Roeder, "Dynamic searchable symmetric encryption," in *Proc. Int. Conf. Financial Cryptography Data Secur.*, 2012, pp. 965–976.

[9] S. Kamara and C. Papamanthou, "Parallel and dynamic searchable symmetric encryption," in *Proc. Int. Conf. Financial Cryptography Data Secur.*, 2013, pp. 258–274.

[10] D. Cash *et al.*, "Dynamic searchable encryption in very-large databases: Data structures and implementation," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2014.

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

[12] Y. Zhang, J. Katz, and C. Papamanthou, "All your queries are belong to us: The power of file-injection attacks on searchable encryption," in *Proc. 25th USENIX Conf. Secur. Symp.*, 2016, pp. 707–720.

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

[14] J. G. Chamani, D. Papadopoulos, C. Papamanthou, and R. Jalili, "New constructions for forward and backward private symmetric searchable encryption," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2018, pp. 1038–1055.

[15] S. Sun *et al.*, "Practical backward-secure searchable encryption from symmetric puncturable encryption," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2018, pp. 763–780.

[16] O. Goldreich, S. Goldwasser, and S. Micali, "How to construct random functions," *J. ACM*, vol. 33, no. 4, pp. 792–807, 1986.

[17] M. Matsumoto and T. Nishimura, "Mersenne twister: A 623-dimensionally equidistributed uniform pseudo-random number generator," *ACM Trans. Model. Comput. Simul.*, vol. 8, no. 1, pp. 3–30, 1998.

[18] J. G. Chamani, "Implementation of mitra, orion, horus, fides, and dianadel," 2018. [Online]. Available: https://github.com/jgharehchamani/SSE

[19] O. Goldreich and R. Ostrovsky, "Software protection and simulation on oblivious rams," *J. ACM*, vol. 43, no. 3, pp. 431–473, 1996.

[20] E. Stefanov, C. Papamanthou, and E. Shi, "Practical dynamic searchable encryption with small leakage," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2014.

[21] R. Bost, "$\sum o\varphi o\varsigma$: Forward secure searchable encryption," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2016, pp. 1143–1154.

[22] C. Zuo, S. Sun, J. K. Liu, J. Shao, and J. Pieprzyk, "Dynamic searchable symmetric encryption with forward and stronger backward privacy," in *Proc. Eur. Symp. Res. Comput. Secur.*, 2019, pp. 283–303.

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

[24] S. Faber, S. Jarecki, H. Krawczyk, Q. Nguyen, M. Rosu, and M. Steiner, "Rich queries on encrypted data: Beyond exact matches," in *Proc. Eur. Symp. Res. Comput. Secur.*, 2015, pp. 123–145.

[25] V. Pappas *et al.*, "Blind seer: A scalable private DBMS," in *Proc. IEEE Symp. Secur. Privacy*, 2014, pp. 359–374.

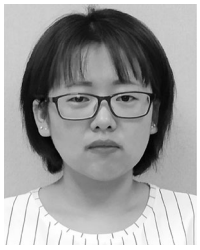
[26] B. A. Fisch *et al.*, "Malicious-client security in blind seer: A scalable private DBMS," in *Proc. IEEE Symp. Secur. Privacy*, 2015, pp. 395–410.

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

[28] S. Lai *et al.*, "Result pattern hiding searchable encryption for conjunctive queries," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2018, pp. 745–762.

[29] C. Zuo, S. Sun, J. K. Liu, J. Shao, and J. Pieprzyk, "Dynamic searchable symmetric encryption schemes supporting range queries with forward (and backward) security," in *Proc. Eur. Symp. Res. Comput. Secur.*, 2018, pp. 228–246.

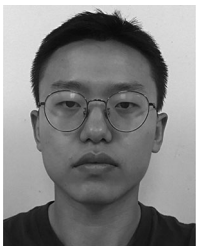
[30] J. Shao, R. Lu, Y. Guan, and G. Wei, "Achieve efficient and verifiable conjunctive and fuzzy queries over encrypted data in cloud," *IEEE Trans. Services Comput.*, 2019, to be published, doi: 10.1109/TSC.2019.2924372.



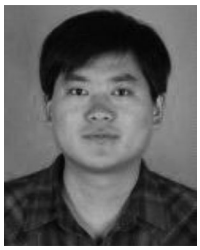
Yandong Zheng received the MS degree from the Department of Computer Science, Beihang University, China, in 2017, and she is currently working toward the PhD degree with the Faculty of Computer Science, University of New Brunswick, Canada. Her research interest includes cloud computing security, big data privacy and applied privacy.

Rongxing Lu (Senior Member, IEEE) received the PhD degree from the Department of Electrical & Computer Engineering, University of Waterloo, Canada, in 2012. He is currently an associate professor with the Faculty of Computer Science (FCS), University of New Brunswick (UNB), Canada. Before that, he worked as an assistant professor with the School of Electrical and Electronic Engineering, Nanyang Technological University (NTU), Singapore from April 2013 to August 2016. He worked as a postdoctoral fellow with the University of Waterloo from May 2012 to April 2013. He was awarded the most prestigious "Governor General's Gold Medal". He won the 8th IEEE Communications Society (ComSoc) Asia Pacific (AP) Outstanding Young Researcher Award, in 2013. He is presently a senior member of IEEE Communications Society. His research interests include applied cryptography, privacy enhancing technologies, and IoT-Big Data security and privacy. He has published extensively in his areas of expertise, and was the recipient of eight best (student) paper awards from some reputable journals and conferences. Currently, He serves as the vice-chair (Publication) of IEEE ComSoc CIS-TC (Communications and Information Security Technical Committee). He is the Winner of 2016-17 Excellence in Teaching Award, FCS, UNB.

Jun Shao received the PhD degree from the Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai, China, in 2008. He was a postdoctoral fellow with the School of Information Sciences and Technology, Pennsylvania State University, Pennsylvania, PA, from 2008 to 2010. He is currently a professor with the School of Computer and Information Engineering, Zhejiang Gongshang University, Hangzhou, China. His current research interests include network security and applied cryptography.



Fan Yin received the BS degree in information security from the Southwest Jiaotong University, Chengdu, China, in 2012. He is currently working toward the PhD degree in information and communication engineering, Southwest Jiaotong University, and also a visiting student at Faculty of Computer Science, University of New Brunswick, Canada. His research interests include searchable encryption, privacy-preserving and security for cloud security and network security.



Hui Zhu (Senior Member, IEEE) received the BSc degree from Xidian University, Xi'an, China, in 2003, the MSc degree from Wuhan University, Wuhan, China, in 2005, and the PhD degree from Xidian University, in 2009. He was a research fellow with the School of Electrical and Electronics Engineering, Nanyang Technological University, Singapore, in 2013. Since 2016, he has been a professor with the School of Cyber Engineering, Xidian University. His current research interests include applied cryptography, data security, and privacy.