

Towards Efficient Verifiable Boolean Search Over Encrypted Cloud Data

Feng Li^{ID}, Jianfeng Ma^{ID}, *Member, IEEE*, Yinbin Miao^{ID}, *Member, IEEE*, Zhiquan Liu^{ID},
Kim-Kwang Raymond Choo^{ID}, *Senior Member, IEEE*,
Ximeng Liu^{ID}, *Member, IEEE*, and Robert H. Deng^{ID}, *Fellow, IEEE*

Abstract—Symmetric Searchable Encryption (SSE) schemes facilitate searching over encrypted data, and have been extensively explored to improve function, efficiency or security. There are, however, additional functions that we need to consider in a real-world setting. For example, forward and backward privacy are required to adequately secure newly added documents and deleted documents in Dynamic SSE (DSSE) schemes, and support boolean search (that allows users to search over encrypted data using basic boolean operations) to achieve improved efficiency and retrieval accuracy. Therefore, in this article we first construct the Verifiable Boolean Search over encrypted data (VBS), and then improve VBS to achieve Forward and Backward privacy (VBS-FB). Finally, we formally prove the security of our proposed schemes, and evaluate their performance using real-world datasets.

Index Terms—Searchable encryption, boolean search, verifiability, forward privacy, backward privacy

1 INTRODUCTION

THE trend of outsourcing encrypted data (e.g., text, audio, and video) to the cloud [1], [2], [3] has partly contributed

- Feng Li and Yinbin Miao are with the School of Cyber Engineering, Xidian University, Xi'an 710071, China, and the Guangxi Key Laboratory of Trusted Software, Guilin University of Electronic Technology, Guilin 541004, China, and also with the Department of Computer Science, City University of Hong Kong, Hong Kong 999077, China. E-mail: fli1996@stu.xidian.edu.cn, ybmiao@xidian.edu.cn.
- Jianfeng Ma is with the School of Cyber Engineering and the Shaanxi Key Laboratory of Network and System Security, Xidian University, Xi'an 710071, China. E-mail: jfma@mail.xidian.edu.cn.
- Zhiquan Liu is with the College of Cyber Security and College of Information Science and Technology, Jinan University, Guangzhou 510632, China. E-mail: zqliu@jnu.edu.cn.
- Kim-Kwang Raymond Choo is with the Department of Information Systems and Cyber Security, University of Texas at San Antonio, San Antonio, TX 78249 USA. E-mail: raymond.choo@fulbrightmail.org.
- Ximeng Liu is with the College of Mathematics and Computer Science, Fuzhou University, Fuzhou 350108, China, and also with Cyberspace Security Research Center, Peng Cheng Laboratory, Shenzhen 518066, China. E-mail: snbnix@gmail.com.
- Robert H. Deng is with the School of Information Systems, Singapore Management University, Singapore 178902, Singapore. E-mail: robertdeng@smu.edu.sg.

Manuscript received 8 Mar. 2021; revised 12 June 2021; accepted 5 Oct. 2021. Date of publication 8 Oct. 2021; date of current version 8 Mar. 2023.

This work was supported by the Foundation for Innovative Research Groups of the National Natural Science Foundation of China under Grant 62121001, the National Natural Science Foundation of China under Grants 62072361, 62072109, U1804263, 61902290, and 62072352, the Fundamental Research Funds for the Central Universities under Grant JB211505, Guangxi Key Laboratory of Cryptography and Information Security under Grant GCIS201917, Guangxi Key Laboratory of Trusted Software under Grant KX202028, Hong Kong Scholar Program under Grant XJ2019038, Henan Key Laboratory of Network Cryptography Technology under Grant LNCT2020-A06, the Key Research and Development Program of Shaanxi under Grant 2020ZDLGY09-06, the Natural Science Foundation of Fujian Province under Grant 2021J06013. The work of Kim-Kwang Raymond Choo was supported only by the Cloud Technology Endowed Professorship. (Corresponding author: Yinbin Miao.)

Recommended for acceptance by W. Zhang.

Digital Object Identifier no. 10.1109/TCC.2021.3118692

to a renewed interest in searchable encryption (SE) schemes, since such schemes allow searching over encrypted data. Existing SE schemes can be either symmetric (i.e., SSE) [4], [5], [6] or asymmetric (i.e., ASE) [7]. However, existing SE schemes, such as those presented in [8], [9], are generally designed to support only a specific function and consequently are not practical. For example, schemes that are designed to support simple search types (e.g., single keyword search, conjunctive keyword search, fuzzy keyword search) are likely to return many irrelevant search results; thus, resulting in poor user experience and network bandwidth wastage. Hence, to mitigate such a limitation we can use boolean search to allow data users to design complex query expressions. For example, we can use SE schemes that support boolean search to find a document that satisfies the following expression: $(word_1 \wedge word_2) \vee (word_3 \wedge word_4 \wedge \neg word_5)$.

In addition to search expressiveness in SE schemes, we also need to consider reliability of the returned results. In other words, the data user should be able to verify integrity (e.g., completeness, correctness, or soundness) of the search results, including those returned by a malicious cloud server. There are many existing SE schemes [10], [11] that consider the cloud server to be honest but curious, in the sense that the cloud server will faithfully perform the protocols but is sufficiently curious about the outsourced data and the queries sent by data users, and attempt to deduce sensitive information from such information. Clearly, the honest but curious cloud server assumption is not reasonable in practice. In addition, the cloud server may only perform a fraction of the search operations and/or return forged results in order to save on computation and storage costs. Thus, practical SE schemes need to support verification of the returned results. That is, the cloud server should be able to prove that the returned results are complete, correct, and sound, as explained below.

- **Completeness:** The cloud server returns a result set $\mathcal{R} = \{R_1, \dots, R_t\}$ for a given query q . For each document

$D_i \in \text{DB}(q)$, if the encrypted form R_i is in the set $\mathcal{R}$, we say that the result set $\mathcal{R}$ is complete.

- **Correctness:** The cloud server returns a result set $\mathcal{R} = \{R_1, \dots, R_t\}$ for a given query q . For each $R_i \in \mathcal{R}$, if the decrypted document D_i has not been tampered with, we say that the document R_i (or D_i) or the result set $\mathcal{R}$ is correct.
- **Soundness:** The cloud server returns a result set $\mathcal{R} = \{R_1, \dots, R_t\}$ for a given query q . For each $R_i \in \mathcal{R}$, if the decrypted document D_i satisfies the query q , we say that the document R_i (or D_i) or the result set $\mathcal{R}$ is sound.

Another function that is important in SE schemes is forward and backward privacy, particularly for dynamic SSE (DSSE) schemes. For example, it has been shown in [12] that file injection attacks on DSSE schemes can result in the recovery of the data user's query. In addition, deleted documents may be retrieved or accessed in subsequent queries. Such an attack reinforces the importance of achieving forward and backward privacy in DSSE schemes.

Given the importance of boolean search, verifiability, and forward and backward privacy in SE schemes, only a small number of existing SE schemes support two of the three functions. Therefore, the focus of this paper is to explore the possibility of designing a scheme that simultaneously supports all these functions securely and efficiently. Specifically, our main contributions in this paper are summarized as follows:

- **Efficient Boolean search.** Our schemes first convert a keyword set into an orthonormal basis, and use the combination of orthonormal vectors as the label of the document to realize boolean search, and subsequently use the inverted index to construct the index structure of the documents. Such an approach allows us to request a boolean query with sub-linear search time.
- **Efficient verifiability of results.** Our schemes exploit a multiset hash function to facilitate completeness verification of the results, which is conducted by the cloud server. The data user only performs inner product operation and produces message digests of the returned results for verifying the soundness and correctness, respectively.
- **Less information leakage.** Our schemes achieve forward privacy and weak backward privacy. We achieve forward privacy by masking the update tokens, and use symmetric puncturable encryption to encrypt the document identity to achieve weak backward privacy, and we formally prove its security.

The rest of the paper is organized as follows. Section 2 reviews several existing SE schemes. Then, in Sections 3 and 4, we introduce the relevant background materials, the system and threat models, the design goals and the security definitions. Our proposed constructions are described in Section 5, and the formal security analysis is presented in Section 6. Finally, in Sections 3.3 and 8, we describe our performance evaluation setup and findings, before concluding this paper.

2 RELATED WORK

Since the first SSE scheme and ASE scheme were proposed by Song *et al.* [13] and Boneh *et al.* [14] respectively, SE has been extensively explored to meet various requirements,

such as multi-keyword search [15], fuzzy keyword search [16], and range search [17].

Boolean search is a practical retrieval method, which allows data users to perform basic boolean searches over encrypted data accurately. The first boolean query on outsourced symmetrically encrypted data was proposed by Cash *et al.* [18], which divides the queried keywords into *s-term* and *x-term* to find the matched results. Subsequently, many schemes [19], [20] have extended [18], aiming to support more general boolean queries in multi-client setting and improve the search efficiency. However, the search time of these schemes is proportional to the number of queried keywords and the documents containing the *s-term*. Xu *et al.* [21] proposed a scheme to support boolean query under public encryption algorithm, which requires more computational cost due to the public encryption adopted. Moataz *et al.* [22] proposed a more general boolean scheme, which is based on the Gram-Schmidt process to orthogonalize the keyword set and determine whether the document satisfies the query through inner product operation. Nevertheless, its search time is proportional to the size of the database, and it fails to take into account possible information leakage during the update process.

The malicious cloud server may execute a fraction of search operations, forge search results and even return some mismatched results for saving bandwidth or computing resources, or for other malicious purposes. Thus, it is necessary to verify the returned results under the assumption that the cloud server is semi-honest or malicious. Chai *et al.* [23] investigated and analyzed the verifiability of returned results under a malicious cloud server. There are many works [24], [25], [26] on the correctness of the results returned by the cloud server, Zhang *et al.* [27] proposed an SSE scheme for verifying the completeness of the results based on the multi-set hash function. Wang *et al.* [28] designed a verifiable method to verify the correctness of returned results based on the message authentication function (MAC). However, these works only focus on one aspect of verifiability, and considers single keyword search only. Etemad *et al.* [29] proposed a verifiable dynamic searchable encryption with boolean search, which provides membership and non-membership proofs for the queried data based on an authenticated skip list. Compared with [22], its search efficiency is relatively low, since it decomposes a boolean search formula to a set of primitive combinations, and then applies a sequence of unions, intersections, and complementations to obtain the final results. This scheme supports the dynamic operations for outsourced data, but does not take into account the forward and backward privacy.

Forward privacy was first explicitly introduced by Stefanov *et al.* [30] and then a specific construction for forward privacy designed by Bost *et al.* [31], named $\Sigma\phi\phi\phi$. After that, Bost *et al.* [32] formally studied stronger security property, called backward privacy, and implemented it with Green-Miers puncturable encryption scheme [33]. Subsequently, many works [34], [35], [36] on forward privacy and backward privacy have emerged, as well as literature combined with other technologies. Zhang *et al.* [27] proposed a verifiable forward-secure SSE scheme, which supports the verifiability of search results and achieves the forward privacy. Guo *et al.* [37] proposed a verifiable and forward-secure SSE scheme based on blockchain technology. However, these existing schemes neither support boolean search,

TABLE 1
A Comparative Summary Among Different Schemes

Schemes	$\mathbb{F}_1$	$\mathbb{F}_2$	$\mathbb{F}_3$	$\mathbb{F}_4$	$\mathbb{F}_5$
[19]	✓*	×	×	×	sub-linear
[27]	×	✓	✓	×	sub-linear
[32]	×	×	✓	✓	sub-linear
[35]	×	×	✓	✓	sub-linear
[18]	✓*	×	×	×	sub-linear
[22]	✓	×	×	×	linear
[29]	✓*	✓	×	×	sub-linear
[27]	×	✓	✓	×	sub-linear
VBS	✓	✓	×	×	sub-linear
VBS-FB	✓	✓	✓	✓	sub-linear

—“*”: Need to perform operations on each keyword in the boolean expression.

— $\mathbb{F}_1$: Boolean search; $\mathbb{F}_2$: Verifiability; $\mathbb{F}_3$: Forward privacy;

$\mathbb{F}_4$: Backward privacy; $\mathbb{F}_5$: Search efficiency;

nor take backward privacy into consideration. To this end, we propose a verifiable boolean search with forward and backward privacy. The comparative summary between our schemes and previous schemes is presented in Table 1.

3 PRELIMINARIES

Here, we briefly introduce the cryptographic primitives and some definitions used in our construction. A summary of the notations used in the paper is shown in Table 2.

3.1 Multiset Hash Function

Multiset hash function [38] operates on multiset (or set), which can map a multiset with finite size into a hash value with fixed length (e.g., SHA-1, MD5). Besides, we can update the hash value if the elements in the multiset have been changed without recalculating the entire multiset.

The MSet-XOR-Hash [38] is the most efficient multiset hash function, which has a triple of PPT algorithms, $(\mathcal{H}, +_{\mathcal{H}}, \equiv_{\mathcal{H}})$. Given a multiset M , MSet-XOR-Hash can be written as follows:

$$\left\{ \begin{array}{l} \mathcal{H}(r, M) = H_k(0, r) \oplus \bigoplus_{m \in M} H_k(1, m); \\ \mathcal{H}(r, M \cup \{x\}) \equiv_{\mathcal{H}} \mathcal{H}(r, M) +_{\mathcal{H}} \mathcal{H}(r, \{x\}) \\ \quad \equiv_{\mathcal{H}} \mathcal{H}(r, M) \oplus H_k(1, x); \\ \mathcal{H}(r, M \setminus \{x\}) \equiv_{\mathcal{H}} \mathcal{H}(r, M) -_{\mathcal{H}} \mathcal{H}(r, \{x\}) \\ \quad \equiv_{\mathcal{H}} \mathcal{H}(r, M) \oplus H_k(1, x), \end{array} \right.$$

where H is a collision resistant hash function such that $\{0, 1\}^{\lambda} \times \{0, 1\}^{*+1} \rightarrow \{0, 1\}^t$ and r is a random nonce.

3.2 Puncturable Pseudorandom Function

A pseudorandom function (PRF) [39], [40] $G: \mathcal{K} \times \mathcal{X} \rightarrow \mathcal{Y}$ is considered to be a *puncturable* PRF, if there is a key space $\mathcal{K}_p$ and a tuple of algorithms $(G.\text{puncture}, G.\text{eval})$, which are defined as follows:

- $G.\text{puncture}(k, x)$ takes the secret key $k \in \mathcal{K}$ and a value $x \in \mathcal{X}$ as input, then it outputs a punctured key $k_x \in \mathcal{K}_p$.
- $G.\text{eval}(k_x, x')$ takes the punctured key k_x and $x' \in \mathcal{X}$ as input. If k_x is the punctured key of x , the algorithm outputs

TABLE 2
Summary of Notations

Notations	Descriptions
$W = \{\omega_1, \dots, \omega_m\}$	Distinct keyword set
EDB	Encrypted database
$S = \{v_1, \dots, v_m\}$	Linearly independent set
$S' = \{u_1, \dots, u_m\}$	Orthogonal set
$S = \{o_1, \dots, o_m\}$	Orthonormal basis
$\text{DB}(q) = \{D_1, \dots, D_t\}$	Set of documents satisfying the query q
$\text{DB}(\omega_i) = \{D_1, \dots, D_t\}$	Set of documents containing ω_i
λ	Security parameter
$ \omega_i = l$	Length of ω_i
$s_1 s_2$	Concatnation between s_1 and s_2
$\delta_{i,j}$	$\delta_{i,j} = 1$ if $i = j$, otherwise, $\delta_{i,j} = 0$
$[1, m]$	Integers between 1 and m
$\langle v_i, v_j \rangle$	Inner product v_i and v_j , that is, $v_i^T v_j$

$$G.\text{eval}(k_x, x') = \begin{cases} G(k, x'), & \text{if } x \neq x'; \\ \perp, & \text{otherwise,} \end{cases}$$

where $\perp \notin \mathcal{Y}$.

Puncturable PRF prevents the punctured x from being calculated as the correct result under the punctured key k_x , and other $x \in \mathcal{X}$ that has not been punctured can be calculated normally.

3.3 Puncturable Encryption

Puncturable encryption is an encryption scheme that encrypts a document with a tag, and the secret key used to encrypt the documents can be *punctured* on a set of tags. The punctured secret key can decrypt ciphertexts except for those attached tags which are punctured on the secret key. The Symmetric Puncturable Encryption (SPE) [35] is based on a puncturable PRF, a symmetric encryption $\text{SE} = (\text{SE.Enc}, \text{SE.Dec})$, and a collision-resistant hash function $H: \mathcal{K} \times \mathbb{N}^+ \rightarrow \mathcal{K}$. SPE consists of four algorithms (KeyGen, Enc, Pun, Dec), a message space $\mathcal{M}$, and a tag space $\mathcal{T}$. We refine SPE as follows:

KeyGen($1^\lambda, d$): The algorithm takes a security parameter λ and an integer $d \in \mathbb{N}^+$ as input, where d denotes the maximum number of tags that can be punctured, then it outputs the secret key sk_0 and the master key $msk = (sk_0, d)$.

Enc(msk, m, t): The algorithm takes the master key msk , a message $m \in \mathcal{M}$, and a tag $t \in \mathcal{T}$ as input, then encrypts m and t together, finally outputs a ciphertext c .

Pun(SK_i, t): The algorithm takes the key SK_i and the tag t that needs to be punctured as input, then it outputs the new punctured key SK_{i+1} . And SK_{i+1} can decrypt all ciphertexts that SK_i can decrypt except for the ciphertext encrypted with the tag t .

Dec(SK_i, c, t): The algorithm takes the key SK_i , the ciphertext c , and the tag t with respect to c as input, then it outputs the message m if the tag t has not been punctured, otherwise outputs $\perp$.

3.4 Symmetric Searchable Encryption

Suppose that the form of the cleartext database (DB) is $\text{DB} = \{(\text{ind}_i, \mathbf{w}_i)_{i=1}^d\}$, where ind_i is the identifier of the document i , $\mathbf{w}_i$ is the keyword set contained in ind_i , and the DB contains d documents. A DSSE scheme is defined as a tuple of three algorithms $\Sigma = (\text{Setup}, \text{Search}, \text{Update})$:

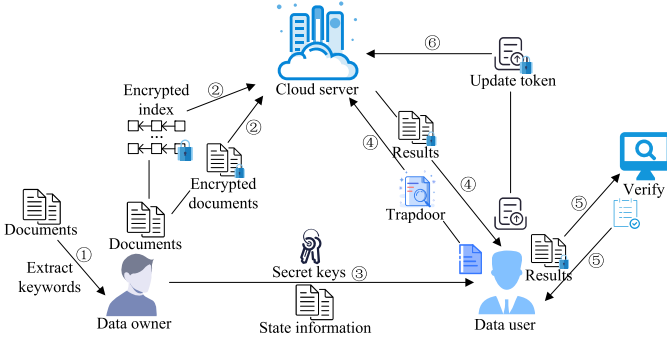


Fig. 1. System model.

- **Setup(DB)** takes a DB as input, then outputs (K, σ, EDB) , where K is a secret key and σ is the state of the user.
- **Search(K, q, σ ; EDB)** takes (K, q, σ) and an EDB as input, where the data user generates a corresponding query trapdoor based on (K, q, σ) and sends it to the cloud server. Then, the server retrieves on the EDB and returns the matched results.
- **Update($K, \sigma, \text{op}, \text{in}$; EDB)** takes $(K, \sigma, \text{op}, \text{in})$ and an EDB as input, where op is an update operation for addition or deletion and in is the pair of (ind_i, w_i) . The user issues an update request and the cloud server returns EDB' and σ' after processing.

4 PROBLEM FORMULATION

In this section, we describe the system model, threat model, design goals and security definitions.

4.1 System Model & Threat Model

The system model of our proposed schemes consists of three entities, namely the data owner, the data user, and the cloud server, as shown in Fig. 1. The role of each entity is as follows:

- *The data owner.* The data owner is responsible for generating the secret keys and building indexes based on the distinct keyword set, and then outsourcing the encrypted indexes and ciphertexts to the cloud server. Besides, the secret keys and the state information which are used to produce the trapdoor will be shared with the authorized user through a secure channel.
- *The data user.* The data user desires to retrieve documents of interest from encrypted data outsourced in the cloud server. He/She first converts the boolean expression into a trapdoor, then sends it to the cloud server and verifies the returned results. Meanwhile, the data user can also produce the update trapdoors by using the shared secret keys and state information.
- *The cloud server.* The cloud server plays the role of data storage and processing. It stores the encrypted indexes and ciphertexts sent by the data owner, and then retrieves the matched ciphertexts according to the trapdoor sent by the data user and returns them to the data user. At the same time, the outsourced data can be updated according to the received update trapdoors.

More specifically, the data owner has a set of documents and wants to outsource them to the cloud server and share it with some trusted users. First, in the step ①, he/she needs to extract a set of distinct keywords from the documents and generate secret keys. Then the corresponding indexes are generated according to the distinct keyword set, and the indexes and documents are encrypted before being sent to the cloud server for storing, in the step ②. After that, the data owner shares the information set (i.e., secret keys, state information, etc.) with the data user through a secure channel. Thus the data user can retrieve and update the data outsourced in the cloud server based on the received information, as shown in the step ③.

The data user can combine the keywords he/she is interested in into a boolean expression, then convert it into a corresponding trapdoor and send it to the cloud server, in the step ④. According to the received trapdoor, the cloud server searches for ciphertexts that satisfy the boolean expression and sends them back to the data user. Later on, in the step ⑤, the data user verifies the completeness, correctness, and soundness of the received results. Finally, the data user is able to update the outsourced data by sending a trapdoor based on the secret keys and the state information (step ⑥).

It is worth noting that the data owner and the data user can also be the same person. Therefore, we will not distinguish between these two entities in the following.

In our scheme, the data owner and the data user are considered two entities that fully trust each other. However, the cloud server is considered to be malicious, which means that it will deduce some sensitive information for various purposes, tamper with returned results, or partially execute the protocols, etc. The leakage of forward and backward privacy during the update process will also cause implicit dangers to data security. Therefore, we also need to prevent the leakage of such information.

4.2 Design Goals

To address the issues mentioned above and the requirements of the data user, we aim to design a more practical and secure DSSE scheme in the paper. The main design goals are as follows.

- *Efficient boolean search.* The Boolean search should allow the data user to use more complex queries and retrieve more accurate results for saving bandwidth resources. In addition, it should not incur heavy computation overhead so that the data user has a better search experience.
- *Efficient result verification.* Our proposed schemes should guarantee the integrity of search results as the malicious cloud server may execute a fraction of search operations, forge search results and even return some mismatched results.
- *Less information leakage.* Our proposed schemes should support dynamic updates without privacy leakage. For example, the cloud server cannot use previous tokens to search newly added files, and the deleted documents cannot be matched.

4.3 Forward & Backward Privacy

Informally, the forward privacy [31] aims to prevent the information about the updated keywords from being

leaked, that is, the cloud server cannot infer whether the updated document matches with the previous queries. And the backward privacy [32] ensures that the cloud server cannot associate the keyword of the current search with a previous update. In other words, whenever a new document/keyword pair (ω, ind) is added and then deleted, if the cloud server cannot know the deleted ind , the DSSE scheme is backward-secure.

Before formally presenting these definitions, we need to introduce some leakage functions. Starting with a common leakage function, the search pattern $\text{sp}(\omega)$, which records each search query (u, ω) via a list Q , where u denotes the timestamp when the keyword ω is searched. Thus, the search pattern can be defined as

$$\text{sp}(\omega) = \{u | (u, \omega) \in Q\},$$

which leaks the timestamps of the search for the same keyword.

In order to capture forward privacy, we introduce the update history $\text{UpHist}(\omega)$ which is the list of storing every update on ω , including the timestamp of the update. Formally, $\text{UpHist}(\omega)$ is defined as

$$\begin{aligned} \text{UpHist}(\omega) = \{u | (u, \text{add}, (\omega, \text{ind})) \in Q \\ \text{or } (u, \text{del}, (\omega, \text{ind})) \in Q\}. \end{aligned}$$

Then we define the leakage functions $\text{TimeDB}(\omega)$ and $\text{DelHist}(\omega)$ for capturing the notion of backward privacy. $\text{TimeDB}(\omega)$ is the list of documents matching keyword ω along with the timestamps of the documents inserted, while excluding the deleted ones. Specifically, $\text{TimeDB}(\omega)$ is defined as

$$\begin{aligned} \text{TimeDB}(\omega) = \{(u, \text{ind}) | (u, \text{add}, (\omega, \text{ind})) \in Q \\ \text{and } \forall (u, \text{del}, (\omega, \text{ind})) \notin Q\}. \end{aligned}$$

The delete history $\text{DelHist}(\omega)$ is the list of timestamp pairs $(u^{\text{add}}, u^{\text{del}})$, where u^{add} represents the timestamp that the entry was inserted into and u^{del} represents the timestamp that the entry was deleted from. Thus, $\text{DelHist}(\omega)$ is defined as

$$\begin{aligned} \text{DelHist}(\omega) = \{(u^{\text{add}}, u^{\text{del}}) | \exists \text{ind s.t. } (u^{\text{del}}, \text{del}, (\omega, \text{ind})) \in Q \\ \text{and } (u^{\text{add}}, \text{add}, (\omega, \text{ind})) \in Q\}. \end{aligned}$$

With these leakage functions in place, we can formally define forward and backward privacy.

Definition 1. (Forward privacy). A $\mathcal{L}$ -adaptively-secure dynamic SSE scheme $\Sigma = (\text{Setup}, \text{Search}, \text{Update})$ is forward privacy iff the update leakage function can be written as

$$\mathcal{L}^{\text{Updt}}(\text{op}, \omega, \text{ind}) = \mathcal{L}'(\text{op}, \text{ind}).$$

Definition 2. (Backward privacy). A $\mathcal{L}$ -adaptively-secure dynamic SSE scheme $\Sigma = (\text{Setup}, \text{Search}, \text{Update})$ is backward privacy iff the search and update leakage functions can be written as

$$\mathcal{L}^{\text{Srch}}(\omega) = \mathcal{L}'(\text{TimeDB}(\omega), \text{DelHist}(\omega)),$$

$$\mathcal{L}^{\text{Updt}}(\text{op}, \omega, \text{ind}) = \mathcal{L}''(\text{op}, \omega).$$

It is worth noticing that backward privacy has three definitions of different strengths in [32], from strong to weak as type-I, type-II, and type-III. In this paper, we consider the weak backward privacy type-III, which is a suitable trade-off between performance and leakage.

5 THE PROPOSED CONSTRUCTION

In this section, we detailedly describe the basic scheme VBS, which supports both the boolean search and results verification. Then, to further improve the security of VBS, we extend it to achieve forward and backward privacy.

5.1 VBS: Basic Construction

Main Idea. In previous works, either the efficient boolean search cannot be achieved [18], [29], or the retrieval efficiency is low [22]. Our basic construction aims to support both efficient boolean search with sub-linear time and verifiability of the results.

Boolean Search. Similar to the existing schemes, we also adopt the inverted index as the index structure of VBS. In addition, we consider each keyword in W as a vector and then convert it into a linearly independent vector set. From such a set, we will obtain an orthonormal basis after the Gram-Schmidt process and normalization, where each keyword corresponds to a vector in the orthonormal basis. We represent each document by the addition of the orthonormal vectors corresponding to the keywords contained in the document. The queries are denoted as vectors in the same way. Therefore, the inner product of the query vector and the document vector can be used to determine whether the document matches the query or not.

Due to the fact that VBS is based on the orthonormal basis derived from the keyword set W , we introduce the pre-processing on the keyword set W at first.

Pre-Procession. Let π be a pseudorandom function such that $\{0, 1\}^\lambda \times \{0, 1\}^* \rightarrow \{0, 1\}^l$. Thus, each keyword ω_i in W can be transformed into a vector of length l , and the vector set is represented as $W = \{\pi(\omega_1), \dots, \pi(\omega_m)\}$.

To obtain the orthonormal basis with respect to keyword set W , the vector set W should be linearly independent. Therefore, we introduce the canonical base of size m , which is denoted by $\{e_1, \dots, e_m\}$, where $|e_i| = m$ for $\forall i \in [1, m]$, and $e_i = (\delta_{i,1}, \dots, \delta_{i,j}, \dots, \delta_{i,m})$. Then we concatenate $\pi(\omega_i)$ with e_i such that $v_i = \pi(\omega_i) \| e_i$ and $|v_i| = l + m$. As a result, the vector set $S = \{v_1, \dots, v_m\}$ is linearly independent. Thus, we can calculate the orthogonal set $S' = \{u_1, \dots, u_m\}$ derived from S according to Gram-Schmidt [41] as follows:

$$\begin{aligned} u_1 &= v_1, \\ u_2 &= v_2 - \text{proj}_{u_1}(v_2), \\ &\vdots \\ u_m &= v_m - \sum_{j=1}^{m-1} \text{proj}_{u_j}(v_m). \end{aligned}$$

Definition 3. The projection operator $\text{proj}_u(v)$ projects the vector v orthogonally onto the line spanned by vector u

$$\text{proj}_u(v) = \frac{\langle v, u \rangle}{\langle u, u \rangle} u.$$

Then we normalize the orthogonal set $S' = \{u_1, \dots, u_m\}$ to an orthonormal basis $\mathcal{S} = \{o_1, \dots, o_m\}$ as follows:

$$o_i = \frac{u_i}{\|u_i\|},$$

which has the following property:

$$\langle o_i, o_j \rangle = o_i^T o_j = \begin{cases} 0, & \text{if } i \neq j; \\ 1, & \text{if } i = j. \end{cases}$$

It is worth noting that the order of the keywords in W impacts the result of an orthonormal basis (e.g., different orders of the keywords in W will generate the different orthonormal bases). Therefore, we need to point out that the order of keywords cannot be changed once the orthonormal basis is generated for a given set of keywords. As a result, we can optionally store the orthonormal basis $\mathcal{S}$ locally or derived from W each time.

Verification. For the sake of results verification, we exploit a multiset hash function to record all elements in each posting list and store them locally for completeness verification. Besides, the Secure Hash Algorithm (SHA) is used to verify the correctness of the documents. Then perform an inner product operation on the returned results and the boolean query to verify the soundness.

Example of Verification. Suppose there are 5 documents $\{1, 2, 3, 4, 5\}$ in the search list and only document 3 matches the boolean query. When the cloud server performs a search, it will use a multiset hash function to record all the searched documents, and then return it to the data user together with document 3. The data user can verify whether the cloud server has completely searched the entire search list by comparing the multiset hash value stored locally and the received multiset hash value. Then the user decrypts the ciphertext and produces its digest to determine whether it has been tampered with. It is worth noting that it is reasonable to store the digests in the cloud server because they are derived from the plaintexts before encryption. Finally, the inner product operation is used to determine that the document 3 is indeed of interest to the data user.

To implement the basic construction VBS, we use vectors in the orthonormal basis to represent documents and queries for efficient boolean search, and then use multiset hash function, SHA, and inner product operation to complete the verifiability of the results.

Detailed Description of VBS. According to the pre-processing of the keyword set W , we introduce the detailed algorithms of VBS as follows:

Setup Phase. Given a security parameter $\lambda \in \mathbb{N}$, the data owner generates a secret key set $K = (\kappa_1, \kappa_2, \kappa_3, \kappa_4, \kappa_5)$, where (κ_1, κ_2) are used for PRF F such that $\{0, 1\}^\lambda \times \{0, 1\}^* \rightarrow \{0, 1\}^t$, κ_3 and κ_4 are used for hash function H_1 and the multiset hash function respectively, and κ_5 is the key of the encryption algorithm.

Each document D_i can be associated with a label ξ_i based on an orthonormal basis $\mathcal{S}$ derived from W , where $\xi_i = \sum_{\omega_j \in W_i} o_j$, o_j is the orthonormal vector in $\mathcal{S}$ corresponding to the keyword ω_j . The key set K and the state table Σ are stored locally, and the EDB is sent to the cloud server. The Setup phase is shown in Algorithm 1.

Algorithm 1. Setup Phase of VBS

Input: $1^\lambda, \text{DB}$
Output: K, Σ, EDB

- 1: Initialize the search table T_s , the file table T_f , and the state table Σ ;
- 2: $\kappa_1, \kappa_2, \kappa_3, \kappa_4, \kappa_5 \xleftarrow{\$} \{0, 1\}^\lambda$;
- 3: **for** $\omega_i \in W$ **do**
- 4: $ST_0 \leftarrow \perp, c \leftarrow 0$;
- 5: $r_\omega \leftarrow F(\kappa_1, \omega_i), \varphi \leftarrow \mathcal{H}(r_\omega, \{\perp\})$;
- 6: **for** $\text{ind}_j \in \text{DB}(\omega_i)$ **do**
- 7: $ST_{c+1} \leftarrow F(\kappa_2, \omega_i \| \text{ind}_j)$;
- 8: $e \leftarrow H_1(\kappa_3, ST_{c+1}) \oplus (ST_c \| \text{ind}_j)$;
- 9: $\xi_j \leftarrow \sum_{\omega_s \in W_j} o_s$;
- 10: $\text{hash} \leftarrow \text{SHA}(D_j)$;
- 11: $T_s[ST_{c+1}] \leftarrow (e, \xi_j, \text{hash}), c \leftarrow c + 1$;
- 12: $C_j \leftarrow \text{Enc}(\kappa_5, D_j), T_f[\text{ind}_j] \leftarrow C_j$;
- 13: $\varphi \leftarrow \varphi + \mathcal{H}(r_\omega, \{\text{ind}_j\})$;
- 14: $\Sigma[\omega_i] \leftarrow (ST_c, c, \varphi)$;
- 15: **return** $K = (\kappa_1, \kappa_2, \kappa_3, \kappa_4, \kappa_5), \Sigma, \text{EDB} = (T_s, T_f)$.

Search Phase. We now depict the Search phase. In general, search queries can be expressed in arbitrary forms: $\mathcal{B}(q) = \diamond \omega_1 \star \diamond \omega_2 \star \dots \star \diamond \omega_n$, where “ $\star$ ” is a binary operator $\star \in \{\vee, \wedge\}$ and “ $\diamond$ ” is a unary operator to indicate that the keyword can be complemented, $\diamond \in \{\emptyset, \neg\}$. Any form of boolean expression can be converted into a disjunctive or conjunctive normal form as follows:

$$\mathcal{B}(q) = \bigvee_{i=1}^k \mathcal{Q}_i = \bigvee_{i=1}^k \bigwedge_{j \in I_i} \diamond \omega_j,$$

where $\bigcup_{i=1}^k I_i = \{1, \dots, n\}$.

Then we explain how to enable the cloud server to securely retrieve documents that satisfy the query or determine whether a document satisfies the query or not. First of all, we introduce a generic keyword that is included in all documents. After orthonormalization, we obtain an orthonormal vector J , and the orthonormal basis is represented as $\mathcal{S} = \{o_1, \dots, o_m, J\}$.

For each conjunctive expression $\mathcal{Q}_i = \bigwedge_{j \in I_i} \diamond \omega_j$, it can be divided into two parts such that $\{P_i, N_i\}$, where P_i and N_i represent a positive keyword set and a negative keyword set, respectively. For each conjunctive expression $\mathcal{Q}_i, i \in [1, k]$, we randomly choose $|P_i|$ positive integers $\{a_{i,j}\}_{1 \leq j \leq |P_i|}$ such that $a_{i,j} \in \mathbb{Z}^+$. Again, we randomly pick $|N_i|$ negative integers $\{c_{i,j}\}_{1 \leq j \leq |N_i|}$ and a positive integer b_i such that $c_{i,j} \in \mathbb{Z}^-$ and $b_i \in \mathbb{Z}^+$. Thus, the conjunctive expression $\mathcal{Q}_i$ can be converted into the following form²:

$$\mathcal{Q}'_i = \frac{1}{b_i + \sum_{\omega_j \in P_i} a_{i,j}} \left(\sum_{\omega_j \in P_i} a_{i,j} o_j + \sum_{\omega_j \in N_i} c_{i,j} o_j + b_i J \right),$$

then we represent the query q as follows:

$$\mathcal{B}(q)' = (\mathcal{Q}'_1, \dots, \mathcal{Q}'_k)^T.$$

Followed by this, the cloud server is able to determine whether the document D_i satisfies the query q with the following implementation:

1. In the algorithm, for the sake of brevity, we don't obviously write the key in the hash function.

2. Notice that the subscript j in the equation is consistent. This is also true in the following equations.

$$f(\mathcal{B}(q)', \xi_i) = \langle \mathcal{B}(q)', \xi_i \rangle = \begin{pmatrix} \langle \mathcal{Q}_1, \xi_i \rangle \\ \vdots \\ \langle \mathcal{Q}_k, \xi_i \rangle \end{pmatrix} = \begin{pmatrix} \frac{1}{b_1 + \sum_{\omega_j \in P_1} a_{1,j}} (\sum_{\omega_j \in P_1 \cap \mathcal{W}_i} a_{1,j} + \sum_{\omega_j \in N_1 \cap \mathcal{W}_i} c_{1,j} + b_1) \\ \vdots \\ \frac{1}{b_k + \sum_{\omega_j \in P_k} a_{k,j}} (\sum_{\omega_j \in P_k \cap \mathcal{W}_i} a_{k,j} + \sum_{\omega_j \in N_k \cap \mathcal{W}_i} c_{k,j} + b_k) \end{pmatrix}.$$

Then we comply with the following rules:

- If $\exists j \in [1, k]$ such that $\langle \mathcal{Q}_j, \xi_i \rangle = 1$, then the document D_i contains all positive keywords in the $\mathcal{Q}_i$ and does not contain any negative keyword. Namely, D_i matches with the query $\mathcal{B}(q)$.
- If $\forall j \in [1, k]$ such that $\langle \mathcal{Q}_j, \xi_i \rangle \neq 1$, then the document D_i does not satisfy the query $\mathcal{B}(q)$.

We note that the conjunctive expression $\mathcal{Q}_i$ is divided into a positive keyword set and a negative keyword set, and then they form the trapdoor $\mathcal{Q}'_i$ together. From the formula, we can know that if the document contains any negative keyword, the inner product result will not be 1. Meanwhile, the $\mathcal{Q}'_i$ for each retrieval is different, even if it is derived from the same $\mathcal{Q}_i$, because the coefficients $a_{i,j}$, $c_{i,j}$, and b_i are randomly selected, which improves the security of the search pattern to some extent. In addition, the generic keyword is to avoid the situation where there are only negative keywords cause the denominator to be 0.

Based on the inverted index, the search time of the VBS is sub-linear. First, the data user selects the keyword with the lowest frequency in each $\mathcal{Q}_i$, that is, the number of the documents containing the keyword is the least (The state table Σ records the number of documents corresponding to each keyword). Then, according to the trapdoor received, the cloud server retrieves documents matching the query in each posting list and deletes the records that have been labeled as $\xi_c = \perp$, since the deletion operation is indicated by setting ξ_c to $\perp$. In this way, the cloud server can avoid repeated retrieval of invalid records and improve efficiency. It will be described in Algorithm 2.

We need to point out that the result $\mathcal{R}$ may contain duplicate encrypted documents, because we will repeatedly add the same documents in different posting lists to $\mathcal{R}$. However, the reason we do this is to reject the result L_r of a certain $\mathcal{Q}_i$ that does not pass the results verification.

Update Phase. The VBS supports dynamic operations without the additional data structures such as the deletion table and the deletion array [10], which are used to achieve danamic operations. For the cloud server, if the operator is deletion, it only needs to set the label $\xi = \perp$ with respect to the deleted document, or add a new record if the operator is addition. The *Update* phase is shown in Algorithm 3.

Verify Phase. The VBS realizes the verification of completeness, correctness, and soundness of the results returned by the cloud server. It is worth noting that storing the digests in the cloud server will not affect the user's verification of the correctness of documents. Since the digests are produced from plaintexts, it is difficult for adversaries to tamper with the documents and the digests at the same time without knowing the secret key. Only when the above properties are met, the algorithm will output "True", otherwise, "False". The *Verify* phase is shown in Algorithm 4.

Algorithm 2. Search Phase of VBS

Input: $\mathcal{B}(q)$, K , Σ , EDB
Output: $\mathcal{R}$
Client:
1: Initialize the lists L_{st} and L_{ms} ;
2: **for** $\mathcal{Q}_i \in \mathcal{B}(q)$ **do**
3: Find the keyword ω_i with the least number of documents in P_i ;
4: $(ST_c, c, \varphi) \leftarrow \Sigma[\omega_i]$;
5: $\mathcal{Q}'_i \leftarrow \mathcal{Q}_i, r_\omega \leftarrow F(\kappa_1, \omega_i)$;
6: $L_{st} \leftarrow (ST_c, \mathcal{Q}'_i, r_\omega)$, $L_{ms} \leftarrow (\varphi, \omega_i)$;
7: Sends $(\mathcal{B}(q)', L_{st}, \kappa_3, \kappa_4)$ to the cloud server;
Server:
8: **for** $(ST_c, \mathcal{Q}'_i, r_\omega) \in L_{st}$ **do**
9: Initialize the list $L_r, \varphi \leftarrow \mathcal{H}(r_\omega, \{\perp\})$;
10: **while** $ST_c \neq \perp$ **do**
11: $(e_c, \xi_c, hash_c) \leftarrow T_s[ST_c]$;
12: $ST_{c-1} || ind_c \leftarrow H_1(\kappa_3, ST_c) \oplus e_c$;
13: **if** $\xi_c = \perp$ **then**
14: **if** $ST_{c+1} = \perp$ **then**
15: $ST_c \leftarrow \perp$, set $ST = ST_{c-1}$;
16: **else**
17: $e_{c+1} \leftarrow H_1(\kappa_3, ST_{c+1}) \oplus (ST_{c-1} || ind_{c+1})$;
18: $T_s[ST_{c+1}] \leftarrow (e_{c+1}, \xi_{c+1}, hash_{c+1})$;
19: $c = c - 1$, **break**;
20: $\varphi \leftarrow \varphi +_{\mathcal{H}} \mathcal{H}(r_\omega, \{ind_c\})$;
21: **if** $\langle \mathcal{Q}'_i, \xi_c \rangle = 1$ **then**
22: $C_c \leftarrow T_f[ind_c]$, $L_r \leftarrow (C_c, hash_c, \xi_c)$;
23: $c = c - 1$;
24: $\mathcal{R} \leftarrow (L_r, \varphi, ST)$;
return $\mathcal{R}$.

Algorithm 3. Update Phase of VBS

Input: (ω, ind) , Σ , op , K
Client:
1: $(ST_c, c, \varphi) \leftarrow \Sigma[\omega]$;
2: $r_\omega \leftarrow F(\kappa_1, \omega)$, $ST \leftarrow F(\kappa_2, \omega || ind)$;
3: **if** $op = \text{add}$ **then**
4: $e \leftarrow H_1(\kappa_3, ST) \oplus (ST_c || ind)$;
5: $\xi \leftarrow \sum_{\omega_s \in \mathcal{W}} o_s$, $hash \leftarrow SHA(D)$;
6: $C \leftarrow Enc(\kappa_5, D)$, $c = c + 1$;
7: $\varphi \leftarrow \varphi +_{\mathcal{H}} \mathcal{H}(r_\omega, \{ind\})$;
8: $\Sigma[\omega] \leftarrow (ST, c, \varphi)$;
9: Sends $((ST, (e, \xi, hash)), (ind, C), op)$ to the cloud server;
10: **if** $op = \text{del}$ **then**
11: $\varphi \leftarrow \varphi -_{\mathcal{H}} \mathcal{H}(r_\omega, \{ind\})$;
12: $\Sigma[\omega] \leftarrow (ST_c, c - 1, \varphi)$;
13: Sends (ST, op) to the cloud server;
Server:
14: **if** $op = \text{add}$ **then**
15: $T[ST_c] \leftarrow (e, \xi, hash)$, $T_f[ind] \leftarrow C$;
16: **else**
17: set $\xi = \perp$ in $T_s[ST]$.

Remark. The basic construction of VBS realizes both boolean search and results verification, but the security has not been fully considered. In particular, when supporting dynamic updates, VBS will introduce more security issues and privacy leakages, such as the leakage of forward and backward privacy.

Algorithm 4. Verify Phase of VBS

Input: $\mathcal{R}, L_{ms}, K, \Sigma, \mathcal{B}(q)'$
Output: $\{\text{True}, \text{False}\}$

- 1: **for** $(L_r, \varphi_r, ST) \in \mathcal{R}$ and $(\varphi, \omega) \in L_{ms}$ **do**
- 2: **if** $ST \neq \perp$ **then**
- 3: Replace ST_c in $\Sigma[\omega]$ with ST ;
- 4: **if** $\varphi_r \neq \varphi$ **then**
- 5: **return** False;
- 6: **for** $(\mathcal{C}, hash, \xi) \in L_r$ **do**
- 7: **if** $SHA(Dec(\kappa_5, \mathcal{C})) \neq hash$ **then**
- 8: **return** False;
- 9: **if** $\langle \xi, \mathcal{Q}'_i \rangle \neq 1$ **then**
- 10: **return** False;
- 11: **return** True.

According to the security definitions, VBS cannot guarantee forward privacy. Since the cloud server is able to infer ST_{c-1} from ST_c when update, which reveals the relationship between the document and the previous queries. That is, the cloud server can learn whether the document contains a keyword we previously searched. And the ind will be exposed to the cloud server when the document is deleted, so VBS does not guarantee backward privacy. These motivate us to extend VBS to be equipped with forward privacy and backward privacy, called VBS-FB.

5.2 VBS-FB: Extended Construction

As aforementioned, VBS is not suitable for being widely deployed on practical platforms due to the security concerns aroused by its dynamic operations. Therefore, we propose an extended construction, VBS-FB, based on the basic construction in this section.

5.2.1 Achieving Forward Privacy

In VBS, the elements in each posting list are stored in a logical location, which is derived from ω and ind. We observe that, however, when the data user wants to insert a document, the logical location ST_c will be sent to the cloud server, which reveals the relationship between previous queries and updates since ST_c is clear text. Therefore, if ST_c is hidden, such a relationship can be protected from being leaked when inserting documents. To this end, we can use a hash function to hide ST_c , such as $F(\kappa, ST_c)$, and the rest is the same as VBS. Algorithm 5 formally describes the *Setup* phase for guaranteeing forward privacy VBS by including additional blue boxed lines.

Algorithm 5. Forward Privacy VBS Setup

- 1: **for** $\omega_i \in \mathcal{W}$ **do**
- 2: $ST_0 \leftarrow \perp, c \leftarrow 0$;
- 3: $r_\omega \leftarrow F(\kappa_1, \omega_i), \varphi \leftarrow \mathcal{H}(r_\omega, \{\perp\})$;
- 4: **for** $\text{ind}_j \in \text{DB}(\omega_i)$ **do**
- 5: $ST_{c+1} \leftarrow F(\kappa_2, \omega_i || \text{ind}_j)$;
- 6: $u \leftarrow H_2(\kappa, ST_{c+1})$;
- 7: $e \leftarrow H_1(\kappa_3, ST_{c+1}) \oplus (ST_c || \text{ind}_j)$;
- 8: $\xi_j \leftarrow \sum_{\omega_s \in \mathcal{W}_j} o_s$;
- 9: $hash \leftarrow SHA(D_j)$;
- 10: $T_s[u] \leftarrow (e, \xi_j, hash)$, $c \leftarrow c + 1$;
- 11: $C_j \leftarrow Enc(\kappa_5, D_j), T_f[\text{ind}_j] \leftarrow C_j$;
- 12: $\varphi \leftarrow \varphi +_{\mathcal{H}} \mathcal{H}(r_\omega, \{\text{ind}_j\})$;
- 13: $\Sigma[\omega_i] \leftarrow (ST_c, c, \varphi)$.

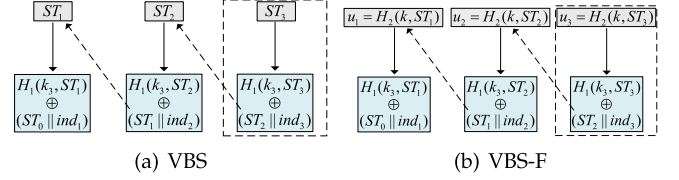


Fig. 2. Example of forward privacy.

Example. As shown in Fig. 2,³ suppose there are two documents ind_1 and ind_2 , and the document ind_3 to be added. Fig. 2a shows the VBS construction. When the data user adds the document ind_3 , the logical location ST_3 will be sent directly to the cloud server. The cloud server can infer ST_2 from ST_3 to reveal the relationship to the previous queries. Therefore, VBS is not a forward-secure scheme. Fig. 2b shows a forward-secure scheme. When a data user adds a document ind_3 , the ST_3 that is masked is sent to the cloud server. The cloud server cannot infer ST_2 based on u_3 , thus it cannot know the relationship between the update and the previous queries.

5.2.2 Achieving Backward Privacy

Obviously, VBS also cannot guarantee backward privacy. Since the delete operation indicates the location of the deleted document, the cloud server knows which document is deleted based on update operations. In this subsection, we present a simple weak backward-secure construction based on the forward-secure VBS.⁴ We note that, the ind of the document is encrypted by SPE, and two VBS instances VBS_{add} and VBS_{del} are used to issue insertion and deletion operations, respectively, in contrast to VBS.

VBS-FB is described in Algorithm 6. In the *Setup* phase, the algorithm initializes two instances. The information stored in the VBS_{add} is similar to that in the VBS, such as the identifiers of documents, while the punctured keys are stored in VBS_{del} . When executing the boolean search, the cloud server runs the *Search* algorithm over VBS_{add} and VBS_{del} , similar to the *Search* phase in VBS. The difference is that inds in the VBS_{add} need to be decrypted using the corresponding punctured keys in VBS_{del} and the deleted document inds cannot be decrypted. It is worth noting that the master secret key needs to be regenerated after each search operation, since the cloud server can decrypt all inds matching ω that have not been punctured by using the keys received before if the master secret key has not been regenerated. However, there is no need to re-encrypt the inds because the cloud server already knows about it.

To insert a new document, the client encrypts the ind with the SPE encryption algorithm with a tag, and the rest operations are similar to those of VBS. To delete a document, the data user punctures the secret key msk' on the corresponding tag, and then he/she updates the msk' in PSK . Finally, the data user adds psk_i to the VBS_{del} .

Example. Fig. 3 shows part of a VBS with backward privacy. Suppose there are two existing documents ind_1 and ind_2 , and a document ind_3 that is deleted after insertion. When the cloud server is searching for these documents, it cannot know that the deleted document is ind_3 , since ind_3 can no longer be decrypted after being punctured by the SPE. Therefore, this construction is backward privacy.

3. The figure only shows part of the key information we need to describe. And VBS-F means that a forward-secure scheme is extended from the VBS.

4. The VBS mentioned later all refer to forward security.

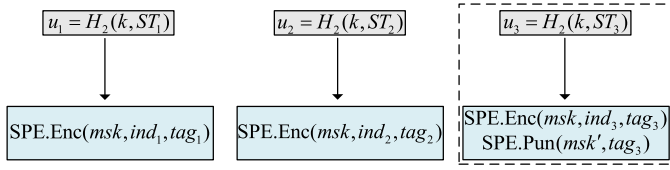


Fig. 3. Example of backward privacy.

Algorithm 6. VBS-FB

Setup(1^λ):

- 1: $(\text{EDB}_{\text{add}}, K_{\text{add}}, \Sigma_{\text{add}}) \leftarrow \text{VBS}_{\text{add}}.\text{Setup}(1^\lambda)$;
- 2: $(\text{EDB}_{\text{del}}, K_{\text{del}}, \Sigma_{\text{del}}) \leftarrow \text{VBS}_{\text{del}}.\text{Setup}(1^\lambda)$;
 // where the EDB and Σ are empty, since there is no DB as input.
- 3: $K_{\text{tag}} \xleftarrow{\$} \{0, 1\}^\lambda$, $\text{MSK}, \text{PSK}, \text{DEL} \leftarrow$ empty map;
Search($\mathcal{B}(q), K, \Sigma, \text{EDB}$):

Client:

- 1: **for** $Q_i \in \mathcal{B}(q)$ **do**
- 2: Find the keyword ω_i with the least number of documents in P_i ;
- 3: $\text{msk}' \leftarrow \text{PSK}[\omega_i]$;
- 4: $d \leftarrow d_{\omega_i}$, $\text{DEL}[\omega_i] \leftarrow d$;
- 5: $\text{msk} \leftarrow \text{SPE}.\text{KeyGen}(1^\lambda, d)$;
- 6: $\text{MSK}[\omega_i] \leftarrow \text{msk}$, $\text{PSK}[\omega_i] \leftarrow \text{msk}'$;
- 7: Send the $\{\text{msk}'\}$ to the cloud server;

Client & Server:

- 8: $\text{VBS}_{\text{del}}.\text{Search}(\mathcal{B}(q), K_{\text{del}}, \Sigma_{\text{del}}, \text{EDB}_{\text{del}})$;
- 9: $\text{VBS}_{\text{add}}.\text{Search}(\mathcal{B}(q), K_{\text{add}}, \Sigma_{\text{add}}, \text{EDB}_{\text{add}})$;
 // Server decrypts each *cind* using punctured keys $(\text{msk}', \text{psk}_1, \dots, \text{psk}_i)$.
- 10: **return** $\mathcal{R}$.

Update($(\omega, \text{ind}), \Sigma, \text{op}, K; \text{EDB}$):

- 1: $\text{tag} \leftarrow F(K_{\text{tag}}, \omega || \text{ind})$;
- 2: $\text{msk} \leftarrow \text{MSK}[\omega]$, $\text{msk}' \leftarrow \text{PSK}[\omega]$, $d \leftarrow \text{DEL}[\omega]$;
- 3: **if** $\text{msk} = \perp$ **then**
- 4: $d \leftarrow d_\omega$, $\text{DEL}[\omega] \leftarrow d$;
- 5: $\text{msk} \leftarrow \text{SPE}.\text{KeyGen}(1^\lambda, d)$;
- 6: $\text{MSK}[\omega] \leftarrow \text{msk}$, $\text{PSK}[\omega] \leftarrow \text{msk}'$;
- 7: **if** $\text{op} = \text{add}$ **then**
- 8: $\text{cind} \leftarrow \text{SPE}.\text{Enc}(\text{msk}, \text{ind}, \text{tag})$;
- 9: $\text{VBS}_{\text{add}}.\text{Update}(\omega, (\text{cind}, \text{tag}), \text{op}, K, \Sigma)$;
- 10: **else**
- 11: $(\text{msk}', \text{psk}_t) \leftarrow \text{SPE}.\text{Pun}(\text{msk}', \text{tag})$;
- 12: $\text{PSK}[\omega] \leftarrow \text{msk}'$;
- 13: $\text{VBS}_{\text{del}}.\text{Update}(\omega, (\text{psk}_t, \text{tag}), \text{op}, K, \Sigma)$.

6 SECURITY ANALYSIS

In this section, we formally analyze the security of VBS-FB and prove it is forward and backward privacy.

The information revealed in our scheme mainly divided into two parts as update leakage profile and search leakage profile. From the view of update, the cloud server only sees addition pairs and deletion pairs, but does not know their specific content except for the update operations. Therefore, the update leakage of VBS-FB can be written as

$$\mathcal{L}_F^{\text{Updt}}(\text{op}, \omega, \text{ind}) = (\text{op}).$$

Leakages incurred by VBS-FB during boolean search are more complex than single keyword. The cloud server

receives the least frequency keyword in each conjunctive query instead of the entire query, since the boolean query is divided into multiple conjunctive queries. Thus, we define the leakage function for boolean query q by overloading notation. For the search pattern, it can be defined as

$$\text{sp}(q) = \bigcup_{i=1}^k \text{sp}(\omega_i).$$

The update history is defined as

$$\text{UpHist}(q) = \bigcup_{i=1}^n \text{UpHist}(\omega_i).$$

For any query q , the leakage function $\text{TimeDB}(q)$ only captures the information about the least frequency keyword in each query

$$\text{TimeDB}(q) = \bigcup_{i=1}^k \text{TimeDB}(\omega_i).$$

Among the documents captured by $\text{TimeDB}(\omega_i)$, the query q needs to be additionally satisfied. And the leakage function $\text{DelHist}(q)$ is defined as

$$\text{DelHist}(q) = \bigcup_{i=1}^k \text{DelHist}(\omega_i).$$

Next, we prove the security of VBS-FB by reduction and we prove that the conjunctive query is forward and backward secure. First, let's assume that there is only one conjunctive query in q .

Theorem 1. Let F be a PRF, H_1 and H_2 are the collision resistant hash functions modeled as random oracles. Given the leakage $\mathcal{L}_F = (\mathcal{L}_F^{\text{Srch}}, \mathcal{L}_F^{\text{Updt}})$ as

$$\mathcal{L}_F^{\text{Srch}}(q) = (\text{sp}(q), \text{UpHist}(q)), \mathcal{L}_F^{\text{Updt}}(\text{op}, \omega, \text{ind}) = (\text{op}),$$

then VBS-FB is $\mathcal{L}_F$ -adaptively-secure with forward privacy.

Proof. Next, we prove the Theorem 1 by constructing a series of successive hybrids from the real world game $\text{SSER}_{\text{REAL}}^{\text{VBS-FB}}(\lambda)$.

Game G_0 . Let G_0 be a real world game $\text{SSER}_{\text{REAL}}^{\text{VBS-FB}}(\lambda)$ such that

$$\Pr[\text{SSER}_{\text{REAL}}^{\text{VBS-FB}}(\lambda) = 1] = \Pr[G_0 = 1].$$

Game G_1 . Instead of using F to generate r_ω , G_1 randomly chooses a string when it receives a new keyword ω , and maintains a table RW to store the pair (ω, r_ω) for reusing when ω is queried in the next time. Similarly, G_1 also maintains a table ST for storing $(\omega || \text{ind}, \text{ST})$ pairs, where ST is a string randomly picked by G_1 after receiving a new string $\omega || \text{ind}$. If there is an adversary who is able to distinguish between G_0 and G_1 , then we can distinguish F from truly random functions with overwhelming probability. Therefore, there is a PPT adversary B_1 such that

$$\Pr[G_0 = 1] - \Pr[G_1 = 1] \leq 2\text{Adv}_{F, B_1}^{\text{prf}}(\lambda).$$

Game G_2 . Instead of using H_2 to generate u , G_2 randomly chooses a string and stores it in the table $\mathbf{U}$ in the *Update* phase.⁵ Then G_2 programs these strings that are consistent

5. It is generally assumed that in a dynamic SSE, all entries are added through the *Update* phase, rather than the *Setup* phase.

Algorithm 7. Game G_2

Setup($\lambda, \perp$):

- 1: Initialize the search table T_s , the file table T_f and the state table Σ ;
- 2: $\kappa_1, \kappa_2, \kappa_3, \kappa_4, \kappa_5, \kappa \xleftarrow{\$} \{0, 1\}^\lambda$;
Update(ω, ind), Σ, op, K ; *EDB*):
 Client:
- 1: $(ST_1, \dots, ST_c, c, \varphi) \leftarrow \Sigma[\omega]$;
- 2: $r_\omega \leftarrow \mathbf{RW}[\omega], ST \leftarrow \mathbf{ST}[\omega] \parallel \text{ind}$;
- 3: **if** $\text{op} = \text{add}$ **then**
- 4: $e \leftarrow H_1(\kappa_3, ST) \oplus (ST_c \parallel \text{ind})$;
- 5: $\xi \leftarrow \sum_{\omega_s \in \mathcal{W}} o_s, \text{hash} \leftarrow \text{SHA}(D)$;
- 6: $\mathcal{C} \leftarrow \text{Enc}(\kappa_5, D), c = c + 1$;
- 7: $\varphi \leftarrow \varphi +_{\mathcal{H}} \mathcal{H}(r_\omega, \{\text{ind}\})$;
- 8: $\Sigma[\omega] \leftarrow (ST_1, \dots, ST_{c-1}, ST, c, \varphi)$;
- 9: $u \xleftarrow{\$} \{0, 1\}^n, \mathbf{U}[ST] \leftarrow u$;
- 10: Sends $((u, (e, \xi, \text{hash})), (\text{ind}, \mathcal{C}), \text{op})$ to the cloud server;
- 11: **if** $\text{op} = \text{del}$ **then**
- 12: $\varphi \leftarrow \varphi -_{\mathcal{H}} \mathcal{H}(r_\omega, \{\text{ind}\})$;
- 13: $\Sigma[\omega] \leftarrow (ST_1, \dots, ST_c, c - 1, \varphi)$;
- 14: $u \leftarrow \mathbf{U}[ST]$;
- 15: Sends (u, op) to the cloud server;
- Server:
- 16: The cloud server performs the same update operation as before;
Search($\mathcal{B}(q), K, \Sigma, \text{EDB}$):
 Client:
- 1: Initialize the lists L_{st} and L_{ms} ;
- 2: **for** $Q_i \in \mathcal{B}(q)$ **do**
- 3: Find the keyword ω_i with the least number of documents in P_i ;
- 4: $(ST_1, \dots, ST_c, c, \varphi) \leftarrow \Sigma[\omega_i]$;
- 5: $Q'_i \leftarrow Q_i, r_\omega \leftarrow \mathbf{RW}[\omega_i]$;
- 6: $L_{st} \leftarrow (ST_c, Q'_i, r_\omega), L_{ms} \leftarrow (\varphi, \omega_i)$;
- 7: **for** $i = 1$ **to** c **do**
- 8: $\mathbf{H}_2(\kappa, ST_i) = \mathbf{U}[ST_i]$;
- 9: Sends $(\mathcal{B}(q)', L_{st}, \kappa_3, \kappa_4)$ to the cloud server;
- Server:
- 10: The cloud server performs search and returns the result $\mathcal{R}$.

with random oracle H_2 such that $H_2(\kappa, ST_c(\omega)) = \mathbf{U}[ST_c(\omega)]$ during the *Search* phase and maintains a table $\mathbf{H}_2$ to keep track of the transcripts. To do so, the state table Σ should keep all ST in $\Sigma[\omega]$ rather than the last record ST_c . The Algorithm 7 describes the game G_2 .

Then we show that G_1 and G_2 are indistinguishable by considering the following. We notice that, $\mathbf{U}[ST_c(\omega)]$ is programmed to table $\mathbf{H}_2$ in the *Search* phase after it is generated in the *Update* phase. Assuming that the adversary does not execute any search query after updating ST_c and then searches for $\mathbf{H}_2$ with ST_c , he will obtain an entry $H_2(\kappa, ST_c(\omega))$ with overwhelming probability which is not equal to $\mathbf{U}[ST_c(\omega)]$ (i.e., $H_2(\kappa, ST_c(\omega)) \neq \mathbf{U}[ST_c(\omega)]$). The next time the same query is made, the adversary will get $\mathbf{U}[ST_c(\omega)]$ in G_2 rather than in the real world (We use $\mathbf{X} = \text{inc}$ to represent this inconsistency). Thus, we have

$$\Pr[G_1 = 1] - \Pr[G_2 = 1] \leq \Pr[\mathbf{X} = \text{inc}].$$

We argue that $\mathbf{X} = \text{inc}$ will occur when a PPT adversary queries ST_c to a random oracle. The probability of

the adversary choosing ST_c is $2^{-\lambda}$, so $\Pr[\mathbf{X} = \text{inc}] \leq \text{poly}(\lambda) \cdot 2^{-\lambda}$ with $\text{poly}(\lambda)$ times at most.

Game G_3 . The only difference between G_3 and G_2 is H_1 . Therefore, we can prove it in the same way that

$$\Pr[G_2 = 1] - \Pr[G_3 = 1] \leq \text{negl}(\lambda).$$

Game G_4 . In game G_3 , φ is a hash value generated by a multiset hash function based on inds for completeness verification. But in G_4 , we use a randomly picked string instead of using a multiset hash function to generate the hash value and keep it in table. Since the multiset hash function is based on the standard hash function, G_3 and G_4 are no different in the view of the adversary. Therefore, we argue that the adversary can not distinguish G_3 and G_4 , then we have

$$\Pr[G_3 = 1] - \Pr[G_4 = 1] = 0.$$

Game G_5 . In G_5 , we introduce a global counter v to record the number of updates. Then we prove that the adversary cannot distinguish between G_4 and G_5 since the strings received by the cloud server during the *Update* phase are indistinguishable, so are the strings received during the *Search* phase. Thus we have

$$\Pr[G_4 = 1] - \Pr[G_5 = 1] = 0.$$

Simulator. The simulator is defined the same as G_5 and described in the Algorithm 8, except for these two. One is to get the update history of the keyword ω from the leakage function $\mathbf{UpHist}$, and the other is to uniquely map the keyword ω to $\bar{\omega} = \min \mathbf{sp}(\omega)$

$$\Pr[G_5 = 1] - \Pr[\text{SSEIDEAL}_{A,S,L_F}^{\text{VBS-FB}}(\lambda) = 1] = 0.$$

According to the above analysis, we can get a distinguishing advantage of a PPT adversary against our scheme

$$\begin{aligned} & \Pr[\text{SSEREAL}_A^{\text{VBS-FB}}(\lambda) = 1] - \Pr[\text{SSEIDEAL}_{A,S,L_F}^{\text{VBS-FB}}(\lambda) = 1] \\ & \leq 2\text{Adv}_{F,B_1}^{\text{prf}}(\lambda) + \text{poly}(\lambda) \cdot 2^{-\lambda} + \text{negl}(\lambda). \end{aligned}$$

Forward security of VBS-FB has been demonstrated based on the assumption of a single conjunctive query. Naturally, the scheme is also forward secure for generic boolean queries. $\square$

Theorem 2. If VBS_{add} and VBS_{del} are $\mathcal{L}_F$ -adaptively-secure constructions, SPE is an IND-sPUN-CPA secure encryption scheme, and F is a PRF, then we say VBS-FB is $\mathcal{L}_B$ -adaptively-secure, with $\mathcal{L}_B = (\mathcal{L}_B^{\text{Srch}}, \mathcal{L}_B^{\text{Updt}})$ defined as

$$\begin{aligned} \mathcal{L}_B^{\text{Srch}}(q) &= (\mathbf{sp}(q), \mathbf{TimeDB}(q), \mathbf{DelHist}(q)), \\ \mathcal{L}_B^{\text{Updt}}(\text{op}, \omega, \text{ind}) &= \text{op}. \end{aligned}$$

Proof. The proof is similar to Theorem 1, which is also proved by constructing a series of hybrids. SPE is an IND-sPUN-CPA secure encryption scheme which has been proved in [35]. Then we prove Theorem 2 as below.

Game G_0 . Let G_0 be a real world game $\text{SSEREAL}_A^{\text{VBS-FB}}(\lambda)$ such that

Algorithm 8. Simulator S .

Setup($\lambda, \perp$):

- 1: Initialize the search table T_s , the file table T_f and the state table Σ ;
- 2: $\kappa_1, \kappa_2, \kappa_3, \kappa_4, \kappa_5, \kappa \xleftarrow{\$} \{0, 1\}^\lambda$;
- 3: $v \leftarrow 0$;
- 4: **RW, ST, U, E**, $\leftarrow$ empty table;

Update(**op**):

Client:

- 1: **if** **op** = **add** **then**
- 2: $\mathbf{U}[v] \leftarrow \{0, 1\}^n$;
- 3: $[v] \leftarrow \{0, 1\}^\lambda$;
- 4: $\mathbf{E}[v] \leftarrow \{0, 1\}^\lambda$;
- 5: $\xi \leftarrow \sum_{\omega_s \in \mathbf{U}} \omega_s$;
- 6: Sends $((u, (e, \xi)), \text{op})$ to the cloud server;
- 7: **if** **op** = **del** **then**
- 8: $\phi[v] \leftarrow \{0, 1\}^\lambda$;
- 9: $u \leftarrow \mathbf{U}[v]$;

10: Sends (u, op) to the cloud server;

11: $v \leftarrow v + 1$;

Server:

12: The cloud server performs the same update operation as before;

Search(**sp**(ω), **UpHist**(ω)):

Client:

- 1: $\bar{\omega} \leftarrow \min \mathbf{sp}(\omega)$;
- 2: $r_\omega \leftarrow \mathbf{RW}[\bar{\omega}]$, $\mathcal{Q}' \leftarrow \mathbf{sp}(\omega)$;
- 3: Parse **UpHist**(ω) as $[(v_1, \text{op}_1, \text{ind}_1), \dots, (v_c, \text{op}_c, \text{ind}_c)]$;
- 4: $c \leftarrow |\mathbf{UpHist}(\omega)|$;
- 5: **if** $c = 0$ **then**
- 6: **return** $\perp$;
- 7: $\mathbf{P}[v_0] \leftarrow \mathbf{H}_3(r_\omega, \{\perp\})$;
- 8: **for** $i = 1$ **to** c **do**
- 9: $\mathbf{ST}_i \leftarrow \mathbf{ST}[\bar{\omega} || \text{ind}_i]$;
- 10: Program $\mathbf{H}_2$ s.t. $\mathbf{H}_2(\kappa, \mathbf{ST}_i) = \mathbf{U}[v_i]$;
- 11: Program $\mathbf{H}_1$ s.t. $\mathbf{H}_1(\kappa, \mathbf{ST}_i) = \mathbf{E}[v_i] \oplus (\mathbf{ST}_{i-1} || \text{ind}_i)$;
- 12: Program $\mathbf{H}_3$ s.t. $\mathbf{H}_3(r_\omega, \text{ind}_i) = \mathbf{P}[v_{i-1}] \oplus \mathbf{H}_3(r_\omega || \text{ind}_i)$;
- 13: $\Sigma[\bar{\omega}] \leftarrow (\mathbf{ST}[\bar{\omega} || \text{ind}_c], c, \mathbf{P}[v_c])$;
- 14: Sends $(\mathcal{Q}', r_\omega, \mathbf{ST}[\bar{\omega} || \text{ind}_c], \kappa_3, \kappa_4)$ to the cloud server;

Server:

15: The cloud server performs search and returns the result $\mathcal{R}$.

$$\Pr[\text{SSER}_{\text{REAL}}^{\text{VBS-FB}}(\lambda) = 1] = \Pr[G_0 = 1].$$

Game G_1 . Instead of using F with key K_{tag} to generate a tag, G_1 randomly picks a string when encountering a new string $\omega || \text{ind}$, and keeps it in table **T** for reusing when $\omega || \text{ind}$ is queried the next time. If there is an adversary who is able to distinguish between G_0 and G_1 , then we can distinguish F from the truly random function with overwhelming probability. More specifically, there is a PPT adversary B_1 such that

$$\Pr[G_0 = 1] - \Pr[G_1 = 1] \leq \text{Adv}_{F, B_1}^{\text{prf}}(\lambda).$$

Game G_2 . In game G_2 , we invoke the simulators S_{add} and S_{del} to replace the VBS_{add} and VBS_{del} , respectively, in the real world. Therefore, G_2 needs to know all the updates and maintain them during the *Update* phase, since it can no longer rely on the cloud server to store them. And then G_2 performs all encryptions and key punctures in

Algorithm 9. Game G_2

Setup(1^λ):

- 1: $\text{EDB}_{\text{add}} \leftarrow S_{\text{add}}.\text{Setup}(1^\lambda)$;
- 2: $\text{EDB}_{\text{del}} \leftarrow S_{\text{del}}.\text{Setup}(1^\lambda)$;
- 3: $u \leftarrow 0$, **Tags**, **Updates**, **DEL** $\leftarrow$ empty map;

Search($\mathcal{B}(q)$, K , Σ , **EDB**):

Client:

- 1: **for** $\mathcal{Q}_i \in \mathcal{B}(q)$ **do**
- 2: Find the keyword ω_i with the least number of documents in P_i ;
- 3: $d \leftarrow d_{\omega_i}$, **DEL**[ω_i] $\leftarrow d$;
- 4: $\text{msk} \leftarrow \text{SPE}.\text{KeyGen}(1^\lambda, d)$;
- 5: Initialize two empty lists L_{add} and L_{del} ;
- 6: **for** $(u_j, \text{op}, \text{ind}_j) \in \text{Updates}[\omega_i]$ **do**
- 7: $\text{tag}_j \leftarrow \text{Tags}[\omega_i || \text{ind}_j]$;
- 8: **if** **op** = **add** **then**
- 9: $\text{cind}_j \leftarrow \text{SPE}.\text{Enc}(\text{msk}, \text{ind}_j, \text{tag}_j)$;
- 10: Append $(u_j, (\text{cind}_j, \text{tag}_j))$ to L_{add} ;
- 11: **else**
- 12: $(\text{msk}', \text{psk}_t) \leftarrow \text{SPE}.\text{Pun}(\text{msk}, \text{tag}_j)$;
- 13: Append $(u_j, (\text{psk}_t, \text{tag}_j))$ to L_{del} ;

14: Send the $\{\text{msk}\}$ to the cloud server;

Client & Server:

- 15: Run the simulator $S_{\text{del}}.\text{Search}(L_{\text{del}})$, the cloud server gets the key shares;
- 16: Run the simulator $S_{\text{add}}.\text{Search}(L_{\text{add}})$, the cloud server gets the ciphertexts and tags;
- 17: **return** $\mathcal{R}$.

Update((ω, ind) , Σ , **op**, K ; **EDB**):

- 1: Append $(u, \text{op}, \text{ind})$ to **Updates**[ω];
- 2: Run $S_{\text{op}}.\text{Update}(\perp)$.

the *Search* phase. The reason we can do this is that VBS_{add} and VBS_{del} are forward-secure and do not leak any useful information during the *Update* phase.

Moreover, G_2 introduces two lists L_{add} and L_{del} store all updates. Specifically, the L_{add} contains information about the documents being added, such as the encrypted identifiers of the documents, the corresponding tags, and the insertion timestamps. Similarly, the L_{del} contains information about the deleted documents, such as the keys that were punctured, the corresponding tags, and the deletion timestamps. We note that L_{add} and L_{del} happen to be update histories of schemes VBS_{add} and VBS_{del} , and are used as inputs to S_{add} and S_{del} , respectively. Algorithm 9 describes the game G_2 .

With these in place, the distinguish advantage between G_1 and G_2 can be written as

$$\Pr[G_1 = 1] - \Pr[G_2 = 1] \leq \text{Adv}_{\text{VBS}_{\text{add}}, B_{\text{add}}, S_{\text{add}}}^{\text{SSE}, \mathcal{E}_F}(\lambda) + \text{Adv}_{\text{VBS}_{\text{del}}, B_{\text{del}}, S_{\text{del}}}^{\text{SSE}, \mathcal{E}_F}(\lambda),$$

where B_{add} and B_{del} are PPT adversaries against schemes VBS_{add} and VBS_{del} respectively, and S_{add} and S_{del} are corresponding simulators.

Game G_3 . G_3 (cf. Algorithm 10) replaces the document identifier with 0 if the inserted document has been deleted before the next search, then encrypts it with SPE.

We notice that only the ciphertext that the tag has been

Algorithm 10. Game G_3 . Only the Modified Parts of G_2 are Displayed

```

Search( $\mathcal{B}(q), K, \Sigma, \text{EDB}$ ):
The previous operations are the same as  $G_2$ ;
6: for  $(u_j, \text{op}, \text{ind}_j) \in \text{Updates}[\omega_i]$  do
7:    $\text{tag}_j \leftarrow \text{Tags}[\omega_i][\text{ind}_j]$ ;
8:   if  $\text{op} = \text{add}$  then
9:     if  $\exists u' \text{ s.t. } (u', \text{del}, \text{ind}_j) \in \text{Updates}[\omega_i]$  then
10:       $\text{cind}_j \leftarrow \text{SPE.Enc}(\text{msk}, 0, \text{tag}_j)$ ;
11:   else
12:      $\text{cind}_j \leftarrow \text{SPE.Enc}(\text{msk}, \text{ind}_j, \text{tag}_j)$ ;
13:   Append  $(u_j, (\text{cind}_j, \text{tag}_j))$  to  $L_{\text{add}}$ ;
14: else
15:    $(\text{msk}', \text{psk}_t) \leftarrow \text{SPE.Pun}(\text{msk}, \text{tag}_j)$ ;
16:   Append  $(u_j, (\text{psk}_t, \text{tag}_j))$  to  $L_{\text{del}}$ ;
The operations after that are the same as  $G_2$ .

```

punctured is modified (i.e., only the deleted document is modified), so the distinguishing advantage between G_2 and G_3 can be reduced to the security of SPE. Assuming that there is an adversary B_2 , we get that

$$\Pr[G_2 = 1] - \Pr[G_3 = 1] \leq \text{Adv}_{\text{SPE}, B_2}^{\text{IND-sPUN-CPA}}(\lambda).$$

Game G_4 . In this game, G_4 (cf. Algorithm 11) first computes the leakage information **TimeDB** and **DelHist** based on the Updates table, and then constructs lists L_{add} and L_{del} from the leakage information. In addition, another difference from G_3 is the way tags are generated. Instead of leveraging the string $\omega[\text{ind}]$ to generate a tag and store it in the Updates table, it is generated on the fly. G_3 is allowed to do this because it assumes that each document can be added and deleted at most once during the Update phase. Therefore, we only need to make sure that the same string $\omega[\text{ind}]$ corresponds to the same tag. We have that

$$\Pr[G_3 = 1] - \Pr[G_4 = 1] = 0.$$

Simulator. We can easily rely on the leakage information $\mathcal{L}_B$ to simulate the final game. Specifically, the simulator can construct the lists L_{add} and L_{del} from the leakage information **TimeDB**(q) and **DelHist**(q), and there is unnecessary to keep track of the updates. Therefore, we have

$$\Pr[G_4 = 1] - \Pr[\text{SSE}_{\text{IDEAL}}^{\text{VBS-FB}}_{A, S, \mathcal{L}_B}(\lambda) = 1] = 0.$$

According to the above analysis, we can get a distinguishing advantage of a PPT adversary against our scheme

$$\begin{aligned} & \Pr[\text{SSER}_{\text{REAL}}^{\text{VBS-FB}}(\lambda) = 1] - \Pr[\text{SSE}_{\text{IDEAL}}^{\text{VBS-FB}}_{A, S, \mathcal{L}_B}(\lambda) = 1] \\ & \leq \text{Adv}_{F, B_1}^{\text{prf}}(\lambda) + \text{Adv}_{\text{VBS}_{\text{add}}, B_{\text{add}}, S_{\text{add}}}^{\text{SSE}, \mathcal{L}_F}(\lambda) \\ & \quad + \text{Adv}_{\text{VBS}_{\text{del}}, B_{\text{del}}, S_{\text{del}}}^{\text{SSE}, \mathcal{L}_F}(\lambda) + \text{Adv}_{\text{SPE}, B_2}^{\text{IND-sPUN-CPA}}(\lambda). \end{aligned}$$

Algorithm 11. Game G_4 . Only the Modified Parts of G_3 are Displayed

```

Search( $\mathcal{B}(q), K, \Sigma, \text{EDB}$ ):
The previous operations are the same as  $G_3$ ;
6: Initialize the empty lists TimeDB and DelHist;
7: for  $(u_j, \text{add}, \text{ind}_j) \in \text{Updates}[\omega_i]$  do
8:   if  $\exists u' \text{ s.t. } (u', \text{del}, \text{ind}_j) \in \text{Updates}[\omega_i]$  then
9:     Append  $(u_j, u')$  to DelHist;
10:  else
11:    Append  $(u_j, \text{ind}_j)$  to TimeDB;
12:  for  $(u_j^{\text{add}}, u_j^{\text{del}}) \in \text{DelHist}$  sorted by increasing  $u_j^{\text{del}}$  do
13:     $\text{tag}_j \leftarrow \{0, 1\}^\lambda$ ;
14:     $\text{cind}_j \leftarrow \text{SPE.Enc}(\text{msk}, 0, \text{tag}_j)$ ;
15:    Append  $(u_j^{\text{add}}, (\text{cind}_j, \text{tag}_j))$  to  $L_{\text{add}}$ ;
16:     $(\text{msk}', \text{psk}_t) \leftarrow \text{SPE.Pun}(\text{msk}, \text{tag}_j)$ ;
17:    Append  $(u_j^{\text{del}}, (\text{psk}_t, \text{tag}_j))$  to  $L_{\text{del}}$ ;
18:  for  $(u_j, \text{ind}_j) \in \text{TimeDB}$  do
19:     $\text{tag}_j \leftarrow \{0, 1\}^\lambda$ ;
20:     $\text{cind}_j \leftarrow \text{SPE.Enc}(\text{msk}, \text{ind}_j, \text{tag}_j)$ ;
15:  Append  $(u_j, (\text{cind}_j, \text{tag}_j))$  to  $L_{\text{add}}$ ;
The operations after that are the same as  $G_3$ .

```

7 PERFORMANCE EVALUATION

In this section, we show the performance of our schemes by comparing with Boolean Symmetric Searchable Encryption (BSSE) [22]. Compared with BSSE, our schemes extend new functionalities and improve security while also speeding up the retrieval efficiency, except in the Setup phase.

7.1 Experimental Settings

Implementation and Setup. We implement our schemes in Python and use the opensource code⁶ of SPE as the puncturable encryption, where d is set to 10. PRFs and hash functions are instantiated with HMAC, and we choose SHA-256 as the underlying message-digest algorithm to digest the documents. For the global security parameter λ , we set its length to 128 bits. In addition, we need to point out that the preprocessing of the keyword set and the encryption of the documents are ignored in the experiments, respectively, because the orthonormal basis can be reused unless the keyword set changes, and the encryption of the document is not the focus of this paper. For comparison, we adopt the same experimental setting for BSSE.

The experiments are performed on an Intel Xeon Gold 6146 CPU 3.2GHz and 128 GB RAM, where the operating system is Ubuntu 18.04, and we run 10 trials for each experiment to get the reliable experimental result.

Dataset. We consider the real-world dataset 20 News-groups⁷ to evaluate the performance of experiments. According to the TF-IDF (Term Frequency-Inverse Document Frequency) technology, 10,000 documents and 1,500 distinct keywords are extracted, of which there are 1,141,081 document/keyword pairs, and the maximum size of posting list that matched a given keyword is 9,950. To assess the impact of the number of documents and the number of keywords contained in the document on the performance of the schemes, we vary the number of

6. <https://github.com/MonashCybersecurityLab/JanusPP>

7. <http://qwone.com/~jason/20Newsgroups/>

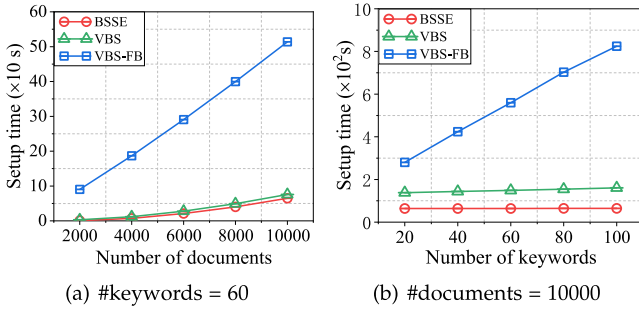


Fig. 4. Performance of the setup.

documents and the number of keywords contained in the document for evaluation. The default number of documents is 10,000, and the number of keywords contained in the document is 60.

7.2 Performance of Experiments

Setup Evaluation. As shown in Figs. 4a and 4b, the index setup time of the three schemes varies with the number of documents and the number of keywords, respectively. The time does not include the transformation of a keyword set to an orthonormal basis.

We observe that as the number of documents or keywords increases, the setup time of these schemes will also increase. Since BSSE only needs to generate a label for each document, the time spent in the *Setup* phase is the least. To support the verifiability of the results, VBS needs additional operations, however, from Fig. 4 we can see that the time spent in the *Setup* phase of VBS and BSSE is very close. The extended scheme VBS-FB supports forward and backward privacy, besides results verification. Therefore, its setup time will be longer than that of the other two schemes due to the SPE encryption algorithm used to implement backward privacy. Fortunately, this will not affect the data user experience, since it only needs to be done once.

Search Evaluation. Here, we analyze the impact of the number of documents and keywords on search efficiency and compare the differences between these three schemes. In Fig. 5, BSSE2 and BSSE4 denote that the query to be searched in BSSE contains 2 or 4 conjunctive expressions, respectively, and the rests are the same.

As shown in Figs. 5a and 5b, we observe a vital performance discrepancy between BSSE and our schemes: the search efficiencies of our schemes are much higher than that of BSSE, since our schemes' search time complexity is $\mathcal{O}(k|DB(\omega)|)$, while BSSE is $\mathcal{O}(n)$, where k is the number of conjunctive expressions in boolean query, and n is the size of the database. Moreover, we find that the search time is constant in BSSE with the increasing keywords contained in the document or conjunctive expressions. This is because no matter how many keywords are contained in the document or conjunctive expressions are included in the query, the search time is only related to the size of the database. As for our schemes, the search time is increasing as the number of keywords or conjunctive expressions, since it depends on the matching documents for the least frequency keyword and the number of conjunctive expressions. At the same time, due to the introduction of forward and backward privacy, the search efficiency of VBS-FB will be slightly lower than that of VBS. Varying the number of documents, the search time

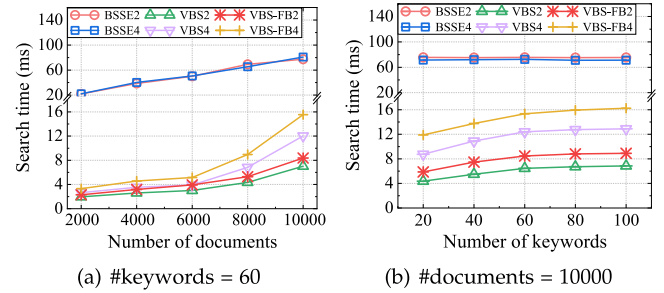


Fig. 5. Performance of the search.

of these schemes will increase since the volume of the dataset increases, as shown in Fig. 5a.

Verifiability Evaluation. The verification of the results includes the completeness, correctness, and soundness, but the time spent in the verification is not much. As for the verification of completeness, the main computation has been completed in the *Search* phase, while for correctness and soundness, the data user only need to compute the returned results, rather than all the retrieved documents. Fig. 6 illustrates our analysis.

The curves of VBS and VBS-FB in Fig. 6 almost overlap, which is reasonable. This is because although VBS-FB has extended VBS to improve the security, this extension does not involve substantial changes to the index structure and will not affect the returned results. Therefore, the verification process is the same as that of VBS. And Fig. 6 shows that the verify time increases as the number of documents returned increases. At the same time, it should be noted that BSSE does not support the verifiability of the results, so BSSE is omitted in this experiment.

Update Evaluation. From the perspective of the update, we evaluate the performance of addition and deletion. The time includes all operations involved in *Update* phase (i.e., token generation, puncture, etc.) and we use 2 or 4 document/keyword pairs for experimentation. Then we compare the three schemes in Figs. 7 and 8.

Addition Evaluation. From Figs. 7a and 7b, we can see that varying the number of documents and keywords does not affect the addition operation, as it does not change with the size of the dataset. Then we notice that VBS takes slightly longer than BSSE since it supports the verifiability of the results, requiring processing of the document's content, ind, and label when adding a document/keyword pair.

As VBS-FB is a combination of VBS and forward and backward privacy, it is necessary to carry out extra processing on update token and ind when adding document/keyword pair to ensure the security we claimed, which

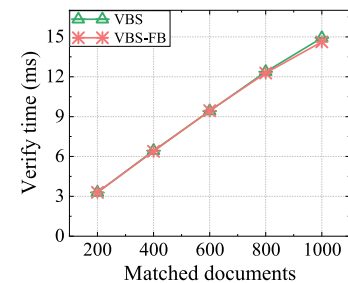


Fig. 6. Performance of the verification.

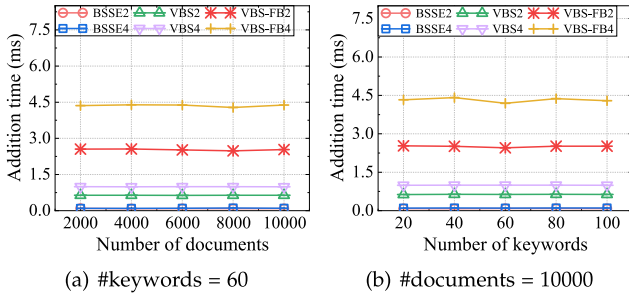


Fig. 7. Performance of the addition operation.

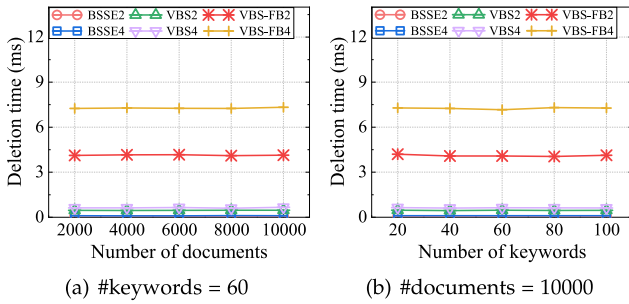


Fig. 8. Performance of the deletion operation.

introduces additional overhead. Nevertheless, this is necessary and acceptable for data security considerations.

Deletion Evaluation. Figs. 8a and 8b show the performance of deletion. Similar to addition operation, the time of deletion operation is constant, which does not change with the size of the dataset. But it will increase as the number of deleted records increases. Compared with addition operation, VBS-FB spends more time on the deletion operation/process, because the ind puncture requires bitwise operation on the tag, so the length of the tag will also affect the puncture time of SPE. In our experiments, the length of the tag is set as 16 bits.

8 CONCLUSION

In this paper, we focused on designing a practical verifiable boolean search over encrypted data, which achieves high search efficiency while supporting results verification. Subsequently, we extended the basic VBS to guarantee forward and backward privacy. Then we formally proved the security of both schemes and evaluated their performance. One limitation of the extended scheme is that the index construction and update incur additional computational costs since SPE is used to encrypt the document identity. And for the efficiency trade-off, we do not consider the information leakage of VBS-FB in search pattern and access pattern. Therefore, in the future, we intend to improve the efficiency of SPE or design a new and more efficient scheme that can achieve full backward privacy.

REFERENCES

- [1] K. Ren *et al.*, "HybridIX: New hybrid index for volume-hiding range queries in data outsourcing services," in *Proc. Int. Conf. Distrib. Comput. Syst.*, 2020, pp. 23–33.
- [2] Y. Guo, M. Wang, C. Wang, X. Yuan, and X. Jia, "Privacy-preserving packet header checking over in-the-cloud middleboxes," *IEEE Internet of Things J.*, vol. 7, no. 6, pp. 5359–5370, Jun. 2020.

- [3] Y. Guo, H. Xie, Y. Miao, C. Wang, and X. Jia, "FedCrowd: A federated and privacy-preserving crowdsourcing platform on blockchain," *IEEE Trans. Services Comput.*, to be published, doi: 10.1109/TSC.2020.3031061.
- [4] W. Xiangyu, J. Ma, M. Yinbin, X. Liu, and Y. Ruikang, "Privacy-preserving diverse keyword search and online pre-diagnosis in cloud computing," *IEEE Trans. Services Comput.*, to be published, doi: 10.1109/TSC.2019.2959775.
- [5] Y. Miao, R. H. Deng, K.-K. R. Choo, X. Liu, J. Ning, and H. Li, "Optimized verifiable fine-grained keyword search in dynamic multi-owner settings," *IEEE Trans. Dependable Secure Comput.*, vol. 18, no. 4, pp. 1804–1820, Jul./Aug. 2021.
- [6] G. S. Poh, J. Chin, W. Yau, K. R. Choo, and M. S. Mohamad, "Searchable symmetric encryption: Designs and challenges," *ACM Comput. Surv.*, vol. 50, no. 3, pp. 40:1–40:37, 2017.
- [7] Q. Tang and X. Chen, "Towards asymmetric searchable encryption with message recovery and flexible search authorization," in *Proc. Symp. Inf. Comput. Commun. Secur.*, 2013, pp. 253–264.
- [8] Y. Miao *et al.*, "Privacy-preserving attribute-based keyword search in shared multi-owner setting," *IEEE Trans. Dependable Secure Comput.*, vol. 18, no. 3, pp. 1080–1094, May/Jun. 2021.
- [9] X. Wang *et al.*, "Search me in the dark: Privacy-preserving boolean range query over encrypted spatial data," in *Proc. IEEE Conf. Comput. Commun.*, 2020, pp. 2253–2262.
- [10] S. Kamara, C. Papamanthou, and T. Roeder, "Dynamic searchable symmetric encryption," in *Proc. Conf. Comput. Commun. Secur.*, 2012, pp. 965–976.
- [11] Y. Miao, R. H. Deng, X. Liu, K.-K. R. Choo, H. Wu, and H. Li, "Multi-authority attribute-based keyword search over encrypted cloud data," *IEEE Trans. Dependable Secure Comput.*, vol. 18, no. 4, pp. 1667–1680, Jul./Aug. 2021.
- [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. USENIX Secur. Symp.*, 2016, pp. 707–720.
- [13] D. X. Song, D. Wagner, and A. Perrig, "Practical techniques for searches on encrypted data," in *Proc. IEEE Symp. Secur. Privacy*, 2000, pp. 44–55.
- [14] D. Boneh, G. Di Crescenzo, R. Ostrovsky, and G. Persiano, "Public key encryption with keyword search," in *Proc. Int. Conf. Theory Appl. Cryptogr. Techn.*, 2004, pp. 506–522.
- [15] N. Cao, C. Wang, M. Li, K. Ren, and W. Lou, "Privacy-preserving multi-keyword ranked search over encrypted cloud data," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 1, pp. 222–233, Jan. 2014.
- [16] Z. Fu, X. Wu, C. Guan, X. Sun, and K. Ren, "Toward efficient multi-keyword fuzzy search over encrypted outsourced data with accuracy improvement," *IEEE Trans. Inf. Forensics Security*, vol. 11, no. 12, pp. 2706–2716, Dec. 2016.
- [17] B. Wang, Y. Hou, M. Li, H. Wang, H. Li, and F. Li, "Tree-based multi-dimensional range search on encrypted data with enhanced privacy," in *Proc. Int. Conf. Secur. Privacy Commun. Netw.*, 2014, pp. 374–394.
- [18] 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. Cryptol. Conf.*, 2013, pp. 353–373.
- [19] L. Sun, C. Xu, and Y. Zhang, "A dynamic and non-interactive boolean searchable symmetric encryption in multi-client setting," *J. Inf. Secur. Appl.*, vol. 40, pp. 145–155, 2018.
- [20] C. Zuo, J. Macindoe, S. Yang, R. Steinfeld, and J. K. Liu, "Trusted boolean search on cloud using searchable symmetric encryption," in *Proc. IEEE Trustcom/BigDataSE/ISPA*, 2016, pp. 113–120.
- [21] P. Xu, S. Tang, P. Xu, Q. Wu, H. Hu, and W. Susilo, "Practical multi-keyword and boolean search over encrypted E-mail in cloud server," *IEEE Trans. Services Comput.*, to be published, doi: 10.1109/TSC.2019.2903502.
- [22] T. Moataz and A. Shikfa, "Boolean symmetric searchable encryption," in *Proc. Symp. Inf. Comput. Commun. Secur.*, 2013, pp. 265–276.
- [23] Q. Chai and G. Gong, "Verifiable symmetric searchable encryption for semi-honest-but-curious cloud servers," in *Proc. Int. Conf. Commun.*, 2012, pp. 917–922.
- [24] Z. Liu, T. Li, P. Li, C. Jia, and J. Li, "Verifiable searchable encryption with aggregate keys for data sharing system," *Future Gener. Comput. Syst.*, vol. 78, pp. 778–788, 2018.

- [25] R. Bost, P.-A. Fouque, and D. Pointcheval, "Verifiable dynamic symmetric searchable encryption: Optimality and forward security," *IACR Cryptol. ePrint Arch.*, vol. 2016, 2016, Art. no. 62.
- [26] J. Wang *et al.*, "Efficient verifiable fuzzy keyword search over encrypted data in cloud computing," *Comput. Sci. Inf. Syst.*, vol. 10, no. 2, pp. 667–684, 2013.
- [27] Z. Zhang, J. Wang, Y. Wang, Y. Su, and X. Chen, "Towards efficient verifiable forward secure searchable symmetric encryption," in *Proc. Eur. Symp. Res. Comput. Secur.*, 2019, pp. 304–321.
- [28] K. Wang, X. Dong, J. Shen, and Z. Cao, "An effective verifiable symmetric searchable encryption scheme in cloud computing," in *Proc. Int. Conf. Inf. Technol.: IoT Smart City*, 2019, pp. 98–102.
- [29] M. Etemad and A. K  p    , "Verifiable dynamic searchable encryption with boolean search," *Turkish J. Electr. Eng. & Comput. Sci.*, vol. 27, no. 4, pp. 2606–2623, 2019.
- [30] E. Stefanov, C. Papamanthou, and E. Shi, "Practical dynamic searchable encryption with small leakage," in *Proc. Annu. Netw. Distrib. Syst. Secur. Symp.*, 2014, pp. 72–75.
- [31] R. Bost, "        : Forward secure searchable encryption," in *Proc. Conf. Comput. Commun. Secur.*, 2016, pp. 1143–1154.
- [32] R. Bost, B. Minaud, and O. Ohrimenko, "Forward and backward private searchable encryption from constrained cryptographic primitives," in *Proc. Conf. Comput. Commun. Secur.*, 2017, pp. 1465–1482.
- [33] M. D. Green and I. Miers, "Forward secure asynchronous messaging from puncturable encryption," in *Proc. IEEE Symp. Secur. Privacy*, 2015, pp. 305–320.
- [34] K. S. Kim, M. Kim, D. Lee, J. H. Park, and W.-H. Kim, "Forward secure dynamic searchable symmetric encryption with efficient updates," in *Proc. Conf. Comput. Commun. Secur.*, 2017, pp. 1449–1463.
- [35] S.-F. Sun *et al.*, "Practical backward-secure searchable encryption from symmetric puncturable encryption," in *Proc. Conf. Comput. Commun. Secur.*, 2018, pp. 763–780.
- [36] J. Ghareh Chamani, D. Papadopoulos, C. Papamanthou, and R. Jalili, "New constructions for forward and backward private symmetric searchable encryption," in *Proc. Conf. Comput. Commun. Secur.*, 2018, pp. 1038–1055.
- [37] Y. Guo, C. Zhang, and X. Jia, "Verifiable and forward-secure encrypted search using blockchain techniques," in *Proc. Int. Conf. Commun.*, 2020, pp. 1–7.
- [38] D. Clarke, S. Devadas, M. Van Dijk, B. Gassend, and G. E. Suh, "Incremental multiset hash functions and their application to memory integrity checking," in *Proc. Int. Conf. Theory Appl. Cryptol. Inf. Secur.*, 2003, pp. 188–207.
- [39] O. Goldreich, S. Goldwasser, and S. Micali, "How to construct random functions," *J. Assoc. Comput. Machinery*, vol. 33, no. 4, pp. 792–807, 1986.
- [40] S. Hohenberger, V. Koppula, and B. Waters, "Adaptively secure puncturable pseudorandom functions in the standard model," in *Proc. Int. Conf. Theory Appl. Cryptol. Inf. Secur.*, 2015, pp. 79–102.
- [41] L. Pursell and S. Trimble, "Gram-schmidt orthogonalization by gauss elimination," *Amer. Math. Monthly*, vol. 98, no. 6, pp. 544–549, 1991.



Feng Li received the BE degree from the School of Cyberspace Security, Hangzhou Dianzi University, Hangzhou, China, in 2018. He is currently working toward the PhD degree in the School of Cyber Engineering, Xidian University, Xi'an, China. His research interests include cloud computing security.



Jianfeng Ma (Member, IEEE) received the PhD degree in computer software and telecommunication engineering from Xidian University, Xi'an, China, in 1995. He was a research fellow with Nanyang Technological University, Singapore, from 1999 to 2001. He is currently a professor and a PhD supervisor with the Department of Cyber Engineering, Xidian University. His research interests include information and network security, wireless and mobile computing systems, and computer networks.



Yinbin Miao (Member, IEEE) received the PhD degree from the Department of Telecommunication Engineering, Xidian University, Xi'an, China, in 2016. He was a post-doctoral fellow with Nanyang Technological University, Singapore, from 2018 to 2019. He is currently a lecturer with the Department of Cyber Engineering, Xidian University. His current research interests include information security and applied cryptography.



Zhiquan Liu received the BS degree from the School of Science, Xidian University, Xi'an, China, in 2012, and the PhD degree from the School of Computer Science and Technology, Xidian University, Xi'an, China, in 2017. He is currently a lecturer with the College of Cyber Security, Jinan University, Guangzhou, China. His current research interests include trust management and privacy preservation in vehicular networks.



Kim-Kwang Raymond Choo (Senior Member, IEEE) received the PhD degree from the Queensland University of Technology, Brisbane, Australia, in 2006. He currently holds the cloud technology endowed professorship at the University of Texas at San Antonio (UTSA). He is the recipient of various awards including the UTSA College of Business Col. Jean Piccione and Lt. Col. Philip Piccione Endowed Research Award for Tenured Faculty in 2018, ESORICS 2015 Best Paper Award. He is an Australian Computer-Society fellow.



Ximeng Liu (Member, IEEE) received the PhD degree from the Department of Telecommunication Engineering, Xidian University, Xi'an, China, in 2015. He is a professor with the Key Laboratory of Information Security of Network Systems, College of Mathematics and Computer Science, Fuzhou University, Fuzhou, China. His current research interests include applied cryptography and big data security.



Robert H. Deng (Fellow, IEEE) is currently AXA chair professor of cybersecurity and professor of information systems with the School of Information Systems, Singapore Management University since 2004. His research interests include data security and privacy, multimedia security, network and system security. He has served on the editorial boards of many international journals, including *TFIS*, *IEEE Transactions on Dependable and Secure Computing*. He has received the Distinguished Paper Award (NDSS 2012), Best Paper Award (CMS 2012), Best Journal Paper Award (IEEE Communications Society 2017).

► For more information on this or any other computing topic, please visit our Digital Library at www.computer.org/csdl.