Information Sciences 475 (2019) 18–28

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

Publicly verifiable database scheme with efficient keyword search

Meixia Miao[a], Jianfeng Wang[a,b,*], Sheng Wen[c], Jianfeng Ma[a]

[a] State Key Laboratory of Integrated Service Networks, Xidian University, Xi'an 710071, China
[b] State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100093, China
[c] Swinburne University of Technology, Melbourne 3122, Australia

ARTICLE INFO

Article history:
Received 9 July 2018
Revised 28 August 2018
Accepted 30 September 2018
Available online 3 October 2018

Keywords:
Verifiable database
Searchable encryption
Vector commitment
Cloud computing

ABSTRACT

The primitive of verifiable database (VDB) enables a resource-limited client to securely outsource a large and dynamic database on an untrusted server. Meanwhile, any misbehavior that attempts to tamper with the database can be detected undoubtedly. However, it seems that all existing VDB constructions only satisfy the basic query and update operations for a certain index performed by the client. In this paper, we first attempt to address the challenge of keyword-based search on VDB scheme. Specifically, we propose a concrete VDB construction supporting efficient keyword search based on the enhanced vector commitment, where each position of vector commitment is tied to a distinct keyword. Furthermore, we show how to extend the basic construction to support conjunctive keyword search. Security and efficiency analysis demonstrate that the proposed VDB schemes can achieve the desired security goals with high efficiency.

1. Introduction

With the adventure of cloud computing and big data, more and more resource-constrained enterprises or clients prefer to move their own data into the cloud. This is known as the paradigm of Database-as-a-Service (or outsourcing storage) [1,2,19,27,28,35]. However, because the cloud server cannot be full trusted, this paradigm also introduces many security challenges. One of the most critical challenges is how to ensure the integrity of the database. If the cloud server could tamper with the data records without being detected, it may return invalid results for any query on the database. As a result, the database services provided by the cloud server are meaningless. For the case of a static database, a trivial solution to ensure the integrity of the database is to use the technique of message authentication code (MAC). For example, the client could generate a signature on each data record of the database and then upload the data record and the resulting signature together. Given a query by the client, the cloud server outputs the data record together with the corresponding signature. Nevertheless, this solution will not work if the database is a dynamic one.

Benabbas et al. [5] firstly proposed the primitive of verifiable database with efficient update (VDB, for short) to address the above challenge of database outsourcing. To be specific, a resource-limited client could not only outsource his large database into the cloud but also retrieve and update any database record. Besides, any misbehavior of attempting to tamper with database records will be detected by the client. The first practical VDB construction was proposed based on the

* Corresponding author.
E-mail address: jfwang@xidian.edu.cn (J. Wang).

https://doi.org/10.1016/j.ins.2018.09.067
0020-0255/

subgroup membership assumption in composite order bilinear groups. However, it only supports the property of private verifiability. That is, only the client (i.e., data owner) can check the correctness of the proofs with his private key. In order to achieve the public verifiability of the outsourced database, Catalano et al. [11] presented an elegant solution to design public verifiable VDB scheme with a new primitive called vector commitment. The proposed scheme can support the public verifiability under the computational Diffie-Hellman (CDH) assumption. Thus, any interested third party could verify the integrity of the database. Motivated by the two initial constructions, plenty of VDB schemes have been proposed in the recent years [12–14,30,31,39].

Note that traditional VDB schemes only focus on the type of NoSQL databases. That is, the database consists of the pair of (i, v_i), where v_i is the corresponding data value indexed by i. Based on the vector commitment technique, the client can query a certain index i and retrieve the data v_i along with the corresponding proof, which can be used to verify the integrity of data value v_i. However, the client can only perform the basic query with a certain index. Note that in the real-world applications, the client may require various types of search, such as range queries, exact match queries, and the most common keyword-based queries (or keyword search). Obviously, the queries on the index in VDB schemes can be viewed as a special kind of keyword search if we view the index as a specified keyword. However, all the existing VDB schemes never considered the general keyword-based queries, e.g., searching all of data records for which the value is equal to 1000. On the other hand, some existing searchable encryption schemes considered the case of performing keyword searches on *dynamic* databases. However, none of them focused on the integrity and verifiability of database. Recently, Zhang et al. [38] presented a new searchable encryption scheme with verifiability via blockchain. We argue that it only support the verifiability of searching result, not the database itself. Specifically, it can only ensure all the matched document identities are returned, but cannot check the integrity of the document content. If the client cannot verify whether the malicious cloud server performed the keyword search on the latest updated database or not, the search results provided by the server were meaningless. To the best of our knowledge, there is no VDB construction that can simultaneously support efficient keyword search.

1.1. Our contribution

In this paper, we first address the challenge of performing efficient keyword search on verifiable and dynamic database. The contributions of this paper are summarised as follows:

- We propose the first VDB framework that supports efficient keyword search by incorporating the primitives of VDB and searchable symmetric encryption (SSE). Besides, the framework can ensure the (public or private) verifiability of dynamic database and thus preserve all properties of VDB schemes.
- The proposed new framework can simultaneously achieve verifiability of search result and integrity of database based on the binding of document identifiers and database indexes. Due to its distinguishable property, the resulting VDB scheme supporting keyword search is almost as efficient as the traditional VDB scheme.
- We show that our proposed construction can be easily extended to expressive search, i.e., conjunctive query. That is, any searchable encryption scheme can be used as a building block of our framework in a black-box manner.

1.2. Related work

1.2.1. Verifiable database

All existing VDB schemes are constructed with the cryptographic primitives of delegating polynomial functions or vector commitment. Benabbas et al. [5] proposed the first practical VDB construction with delegating polynomial functions. Their solution could only support private verifiability because the secret key is involved in verifying the proof. Catalano and Fiore [11] presented the first publicly verifiable VDB scheme from vector commitment. Nevertheless, Chen et al. [12] argued that the proposed VDB scheme [11] could not resist the forward automatic update (FAU) attack by the malicious server. In order to reduce the computation overhead on the client side, Chen et al. [14] presented a new primitive called VDB with incremental updates (Inc-VDB), which was more efficient in the case that the database confronts frequent but slight modifications. Note that all the VDB schemes mentioned above [5,11,12,14] only support data replacement operation. Miao et al. [31] firstly considered to design an efficient VDB scheme that supported all types of updating operations. They also presented a concrete VDB construction supporting all updating operations by using a new primitive called hierarchical vector commitment. However, the number of layer of hierarchical vector commitment in their construction increased linearly in the case when some data records were continually inserted at the same position of the database. Following that, Miao et al. [30] introduced a new primitive called Merkel sum hash tree and then used to construct an efficient VDB scheme supporting all types of update operations. However, it could not support the property of public verifiability.

1.2.2. Searchable encryption

The notion of searchable encryption could be divided into two categories. One is called searchable asymmetric encryption (or known as public-key encryption keyword search), and the other is called searchable symmetric encryption (SSE for short). For the efficiency of real-world applications, we only focus on SSE in this paper, which was firstly introduced by Song et al. [32]. However, the searching complexity in Song et al.'s work was linear to the size of ciphertext. In order to

address the efficiency problem of SSE, plenty of research work have been done in recent years [9,15,18,26,29]. Curtmola et al. [15] constructed an inverted index to improve the search efficiency of SSE. The first dynamic and sub-linear SSE was first proposed by Kamara et al. [21]. Cash et al. [9] presented a new dynamic SSE scheme that can support a large database. Stefanov et al. [33] developed the first dynamic SSE scheme satisfied the forward security. Recently, some dynamic SSE schemes were proposed to support both forward and backward security [7,22]. However, all these SSE schemes were constructed in the honest-but-curious server model. The dishonest server may return incorrect search results in order to save the computational resources. Kurosawa and Ohtaki [24] proposed the first verifiable SSE scheme which can verify the correctness of the search results. Wang et al. [36,37] designed a Bloom Filter Tree to solve the problem on how to verify when the search result was empty in both static and dynamic database. Azraoui et al. [3] proposed a verifiable SSE scheme which can achieve conjunctive keyword search. Recently, Bost et al. [6] proposed a verifiable dynamic SSE scheme based on the primitive of verifiable hash table. On the other hand, in order to enrich the search expression, plenty of research work focused on designing multi-keyword searchable encryption schemes [10,16,17,20,23,25].

1.3. Organization

The rest of this paper is organized as follows. Some preliminaries are presented in Section 2. The proposed VDB scheme supporting keyword search is given in Section 3. We present the security and efficiency analysis of our proposed VDB scheme in Section 4. The experimental evaluation is presented in Section 5. Finally, we conclude this paper in Section 6.

2. Preliminaries

In this section, we introduce some useful preliminaries which will be used in this paper.

2.1. Mathematical assumption

Definition 1. The Computational Diffie-Hellman (CDH) Problem: Given a triple (g, g^a, g^b), where g is the generator of $\mathbb{G}$ and $a, b \in_R \mathbb{Z}_p$, outputs g^{ab}. The CDH assumption means that, for any probabilistic polynomial time (PPT) adversary $\mathcal{A}$, the probability to output g^{ab} is negligible, i.e., $\Pr[\mathcal{A}(1^k, g, g^a, g^b) = g^{ab}] \leq \mathsf{negl}(k)$, where $\mathsf{negl}(\cdot)$ is a negligible function with security parameter k.

A variant of CDH problem is the square computational Diffie-Hellman (Squ-CDH) problem. The definition is as follows: given a tuple (g, g^a), where g is a generator of $\mathbb{G}$ and $a \in_R \mathbb{Z}_p$, outputs g^{a^2}. Note that the Squ-CDH problem is equivalent to traditional CDH problem, which has been proved in [4].

2.2. Searchable symmetric encryption

Song et al. [32] firstly introduced the primitive of searchable symmetric encryption to address encrypted data search. In the following, we present the formal definition of SSE.

Definition 2. A searchable symmetric encryption (SSE) [32] scheme is a triple $\Pi = (\mathsf{EDBSetup}, \mathsf{TokenGen}, \mathsf{Search})$ consists of three algorithms:

- $\mathsf{EDBSetup}(\lambda, \mathsf{DB})$: Let λ be the security parameter. Given a database $\mathsf{DB} = \{(\mathsf{ind}_i, \mathsf{W}_i) : i \in [1, d]\}$ with $\mathsf{ind}_i \in \{0,1\}^\lambda$ and $\mathsf{W}_i \subseteq \{0,1\}^*$. The data owner runs this algorithm and outsources the database DB into the server. It takes λ, DB as input and generates the secret key sk and outsources the encrypted database EDB to the server[1]
- $\mathsf{TokenGen}(sk, q)$: It takes as input secret key sk and search query q, outputs the search token t_q.
- $\mathsf{Search}(t_q, \mathsf{EDB})$: On receiving the query token t_q, the server performs search over the EDB and returns all matched results R to the client.

2.3. Tuple set

Cash et al. [10] proposed a new primitive named tuple set, or TSet, which can be viewed as an inverted index to build a single keyword-based SSE scheme. Roughly speaking, TSet is a table structure, where each row is indexed by a keyword and stored all the document identifiers that contain the given keyword. Upon receiving a keyword-related token, the server is enabled to perform search with the TSet. In the following, we present the formal definition of TSet with the following syntax:

- $\mathsf{TSetSetup}(\lambda, DB)$: It takes as input security parameter λ and the database DB, outputs the encrypted database TSet.
- $\mathsf{TSetGetTag}(K_T, w)$: It takes as input private key K_T and a query keyword w, outputs the corresponding search token $stag$.

[1] Here, EDB contains the encrypted version of the database and the corresponding index.

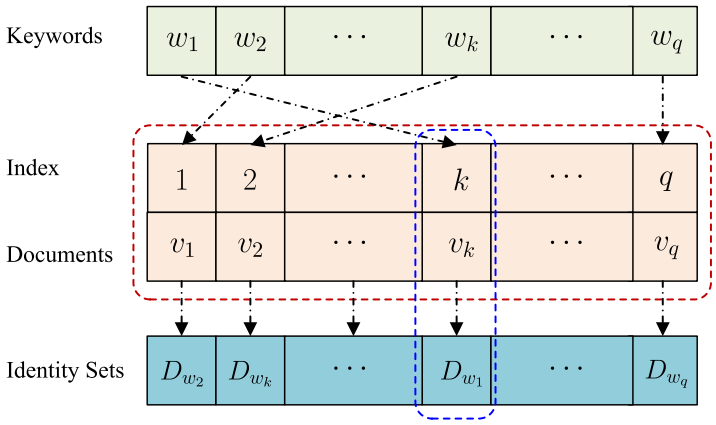

Fig. 1. The main idea of the proposed VDB scheme with keyword search.

- $\mathsf{TSetRetrieve}(\mathsf{TSet}, stag)$: It takes as input a queried token $stag$ and TSet, outputs the encrypted document indemnifiers t.

Definition 3. Let $\Sigma = (\mathsf{TSetSetup}, \mathsf{TSetGetTag}, \mathsf{TSetRetrieve})$ be a TSet instantiation, and $\mathcal{A}$ be an polynomial adversary and $\mathcal{S}$ be a simulator. We define the security of TSet via the following two probabilistic experiments:

- $\mathbf{Real}^{\Sigma}_{\mathcal{A}}(\lambda)$: Adversary $\mathcal{A}$ chooses a keyword set W, the experiment runs $(\mathsf{TSet}, K_T) \leftarrow \mathsf{TSetSetup}(\lambda, W)$ and sends TSet to adversary $\mathcal{A}$. He repeatedly generates queries $w \in W$, and sends $stag \leftarrow \mathsf{TSetGetTag}(K_T, w)$ to $\mathcal{A}$. Finally, a binary bit b is output as the final output.
- $\mathbf{Ideal}^{\Sigma}_{\mathcal{A},\mathcal{S}}(\lambda)$: The experiment initializes an empty list q and sets a counter $i = 0$. Adversary $\mathcal{A}$ selects a keyword set W, the experiment runs $\mathcal{S}(\mathcal{L}_{\mathsf{TSet}})$ and sends TSet to adversary $\mathcal{A}$. He repeatedly generates queries $w \in W$, stores w in $q[i]$ with increments i, and sends the outputs of $\mathcal{S}(\mathcal{L}_{\mathsf{TSet}}, q)$ to $\mathcal{A}$. Finally, a binary bit b is output as the final output.

We say the Σ is $\mathcal{L}_{\mathcal{T}}$-adaptively secure TSet construction if there exist an simulator $\mathcal{S}$ for all probabilistic polynomial-time adversary $\mathcal{A}$, such that: $|\Pr[\mathbf{Real}^{\Sigma}_{\mathcal{A}}(\lambda) = 1] - \Pr[\mathbf{Ideal}^{\Sigma}_{\mathcal{A},\mathcal{S}}(\lambda)]| \leq negl(\lambda)$.

3. New efficient VDB scheme with support for keyword search

3.1. High description

In traditional VDB schemes [5,11,12,14], the client can only perform the simple query on a certain index i (one position of the database vector). The main reason is that it does not have a keyword-based index that can be used to achieve expressive search on the whole database. A trivial solution is that we can regard each document identity as one component of vector commitment. Then, the server can provide the corresponding proof according to the position of the matched document identity. However, the main drawback is that the number of the documents must be fixed beforehand, which contradicts the original purpose of VDB, i.e., the client can perform the data updating operations in a verifiable manner.

In order to fill the gap between VDB and keyword search, we present a novel VDB scheme to support keyword search. Our main idea is to insert an additional layer for the traditional vector commitment, where the distinct keyword set is assigned into the vector commitment. That is, each position of vector commitment is used to index a distinct keyword w at the additional layer of vector commitment (as shown in Fig. 1). More precisely, each position x of vector commitment can be used to search all the document identifiers DB_w that contain a given keyword w. Similar with [10,34], a new TSet instance is constructed to perform keyword-based search, which supports efficient uploading for each key-document identifier pair in an "on-the-fly" fashion.

Once a token of keyword w is received, the server can firstly perform search over the searchable keyword-index to obtain a pair of position/document identifiers. Then the position can be used to verify the integrity of documents based on vector commitment [12]. Note that v_i in Fig. 1 represents the document F_i. In the real applications, the exact data that is placed into the commitment is the hash value of encrypted version F_i.

3.2. A Concrete VDB Scheme with keyword search

In this section, we present a novel VDB construction supporting efficient keyword-based search based on the primitives of vector commitment and searchable encryption.

We firstly present some notations. Assuming that k is the system security parameter. A database $DB = (id_i, W_i)_{i=1}^{d}$ consists of document identifier/keyword set pairs, where d denotes the documents number of he whole database. Let DB_w refers to a

set of document identifiers that contain keyword w. Let F: $\{0,1\}^{\lambda} \times \{0,1\}^{\lambda} \mapsto \{0,1\}^{\lambda}$ is a pseudorandom function. In addition, Let $\mathbb{G}_1$ and $\mathbb{G}_2$ be two groups where $|\mathbb{G}_1| = |\mathbb{G}_2| = q$ and g is a generator of $\mathbb{G}_1$. Let e be a bilinear map, $\mathcal{H}$ be a secure hash function in $\mathbb{G}_1$, and P be a permutation in range $[1, q]$.

- $\mathsf{Setup}(1^k, DB)$: The client randomly chooses q elements $z_i \in_R \mathbb{Z}_p$ and computes $h_i = g^{z_i}$, $h_{i,j} = g^{z_i z_j}$, where $1 \le i \ne j \le q$. Then, he randomly chooses secret key $y \in_R \mathbb{Z}_p$ and then computes $Y = g^y$. The public parameter $\mathsf{PP} = (p, q, \mathbb{G}_1, \mathbb{G}_2, \mathcal{H}, e, g, \{h_i\}_{1\le i\le q}, \{h_{i,j}\}_{1\le i,j\le q, i\ne j})$, and the message space $\mathcal{M} = \mathbb{Z}_p$.
  1. The client initializes an array TSet that each element consists of a pair of $(\mathsf{key}, \mathsf{value})$, where key refers to an identifier of element and the value represents the ciphertext of keyword/document identifier pair. Let $W = \cup_{i=1}^{d} W_i$ be the distinct keyword set of DB, K_S and K_T be the keys of PRF F. The detail is presented as follows:
     - For each keyword $w \in W$ and the corresponding document identifiers DB_w, the client computes the following values: $K_e = F(K_S, w)$, $stag_w = F(K_T, w)$ and $e_w = Enc(K_e, x||DB_w))$ where x is the assigned index for keyword w by $P(ord(w))$.[2] Then he assigns $\mathsf{TSet}[stag_w] = e_w$.
     - The client generates a Bloom filter BF and inserts all keyword tokens $stag$ into it, which can ensure the verifiability of search.
     - Assuming that n_i is the number of document identifiers in DB_{w_i}. A Merkle hash tree (MHT) is generated, where each leaf node is allocated with a hash value $h(x_i||w_i||n_i)$. We denote the of MHT as ϕ root of MHT as ϕ.
  2. Assume that $T = 0$ is the initial counter, the client generates the commitment $C_R = \prod_{i=1}^{q} h_i^{v_i}$ on the original database vector $(v_1||DB_{w_{P(1)}}, v_2||DB_{w_{P(2)}}, \cdots, v_q||DB_{w_{P(q)}})$, where $DB_{w_{P(i)}}$ represents all document identifiers for keyword $w_{P(i)}$. Given $C^{(T)}$ to be the commitment on the current database vector that is obtained from the original database by updating T times. Specially, $C^{(0)} = C_R$ and $C_{-1} = C_R$. The client computes $H_0 = \mathcal{H}(C_{-1}, C^{(0)}, 0)^y$ and sends it to the server. The server first checks the validity of H_0, the server generates $C_0 = H_0 C^{(0)}$. Also, the server inserts the tuple $(H_0, C_{-1}, C^{(0)}, 0)$ into aux.

  Finally, the public key $\mathsf{PK} = (\mathsf{PP}, Y, C_R, C_0, \mathsf{BF}, \mathsf{MHT}, \phi)$ and the auxiliary information $\mathsf{S} = (\mathsf{PP}, \mathsf{aux}, DB, \mathsf{TSet})$ are uploaded to the server and the private key $\mathsf{SK} = \{y, K_S, K_T\}$ is stored locally by the client.
- $\mathsf{Query}(\mathsf{PK}, \mathsf{S}, w, x)$: Suppose the client wants to search all the documents that contain a given keyword w. When input the key (K_S, K_T) and w, the client firstly computes $stag_w = F(K_T, w)$ and sends it to the server. The detailed procedure will be performed as follows:
  1. Once the server receives the search token $stag_w$, it performs search over TSet and returns $t = \mathsf{TSet}[stag_w]$ to the client. Then the client computes $K_e = F(K_S, w)$ and obtains $x||DB_w = Dec(K_e, t)$.
  2. Assume that $\mathsf{PK} = (\mathsf{PP}, Y, C_R, C_T, \mathsf{BF}, \mathsf{MHT}, \phi)$ is the latest public key. Upon receiving the queried index x, the server computes $\pi_x^{(T)} = \prod_{1\le j\le q, j\ne x} h_{x,j}^{v_j^{(T)}}$ and sends back the proof $\tau = (v_x^{(T)}, \pi_x^{(T)}, H_T, C_{T-1}, C^{(T)}, T, \mathsf{BF}, \phi)$.
- $\mathsf{Verify}(\mathsf{PK}, x, \tau)$: The verify can be performed in the two phases:
  1. Firstly, the client verifies the integrity of search result, i.e., DB_w. There are two steps needed to perform:
     - When the search result is empty, the client checks whether $\mathsf{BF}(stag_w) = 1$ holds. If not, the process terminates and the client accepts the search result.
     - When the result is not empty, the client can verifies the completeness by checking $h(x||w||n_x)$ with the root of MHT ϕ, where n_x is obtained by decrypting ciphertext e. Note that the correctness of search result can be achieved by the properties of vector commitment in the following process.
  2. The client parses the proofs $\tau = (v_x^{(T)}, \pi_x^{(T)}, H_T, C_{T-1}, C^{(T)}, T)$. Any verifier (including himself) could verify the correctness of τ by checking the two equations[3]: $e(H_T, g) = e(\mathcal{H}(C_{T-1}, C^{(T)}, T), Y)$ and $e(C_T / H_T h_x^{v_x^{(T)}}, h_x) = e(\pi_x^{(T)}, g)$. If so, the verifier outputs $v_x^{(T)}$. Otherwise, a terminator $\perp$ is returned.
- $\mathsf{Update}(\mathsf{SK}, w, x, v_x')$: Given a keyword w, the client firstly obtains the corresponding indicator x, and then retrieves all the corresponding document identifiers DB_w. For each $x \in DB_w$, the server returns the latest data record v_x and the corresponding proof τ to the client. If $\mathsf{Verify}(\mathsf{PK}, x, \tau) = v_x \ne \perp$ holds, the client adds 1 to T and computes $C^{(T)} = \frac{C_{T-1}}{H_{T-1}} h_x^{v_x' - v_x}$ and $t_x' = H_T = \mathcal{H}(C_{T-1}, C^{(T)}, T)^y$. Finally, (t_x', v_x') is sent to the server. If t_x' is valid, the server generates $C_T = H_T C^{(T)}$ and updates the current public key as $\mathsf{PK} = (\mathsf{PP}, Y, C_R, C_T, \mathsf{BF}, \mathsf{MHT}, \phi)$. Finally, the server replaces v_x by v_x' at the position of x, i.e., $DB_x \leftarrow v_x'$. and inserts $(t_x' = H_T, C_{T-1}, C^{(T)}, T)$ into aux.

3.3. The enhanced construction with conjunctive query

In the above-mentioned construction, the client can perform single keyword search over VDB scheme, which makes it more useful in real-world scenarios. Furthermore, we extend the basic construction to multi-keyword setting by adopting OXT protocol in [10]. The details of the proposed construction can be describing as follows:

- $\mathsf{Setup}(\mathsf{SK}, \mathsf{PP}, DB)$: On input the security parameter λ, the client (data owner) randomly selects keys K_S, K_T for PRF F and K_X, K_I, K_Z for PRF F_p (F_p: $\{0,1\}^* \to Z_p^*$). Then he randomly chooses q elements $z_i \in_R \mathbb{Z}_p$ and computes $h_i = g^{z_i}$, $h_{i,j} = g^{z_i z_j}$,

[2] The $ord(w)$ refers to the numeric order of w in the whole keyword set W.

[3] If the verifier is the data owner, then he requires only to check whether the equation $H_T = \mathcal{H}(C_{T-1}, C^{(T)}, T)^y$ holds.

where $1 \le i \ne j \le q$. Furthermore, he randomly chooses secret key $y \in_R \mathbb{Z}_p$ and computes the corresponding public key $Y = g^y$. Finally, The public parameter of system can be described as $\mathsf{PP} = (p, q, \mathbb{G}_1, \mathbb{G}_2, \mathcal{H}, e, g, \{h_i\}_{1 \le i \le q}, \{h_{i,j}\}_{1 \le i,j \le q, i \ne j})$ and the secret key is $\mathsf{SK} = (y, K_S, K_T, K_X, K_I, K_Z)$. The client generates the encrypted database VDB with the following Algorithm 1.

Algorithm 1 $\mathsf{Setup}(\mathsf{SK}, \mathsf{PP}, \mathsf{DB})$.

Input SK, PP, DB
Output VDB, PK, S

$\mathsf{TSet}, \mathsf{XSet}, \mathsf{Token} \leftarrow \phi$
for $w \in \mathsf{W}$ **do**
  $C_w \leftarrow \phi$; $c \leftarrow 1$; $T \leftarrow 0$
  $K_e \leftarrow F(K_S, w)$; $\mathsf{T}_w \leftarrow F(K_T, w)$; $\mathsf{Token} \leftarrow \mathsf{Token} \cup \{\mathsf{T}_w\}$
  for $\mathrm{ind} \in DB_w$ **do**
    $\mathrm{xind} \leftarrow F_p(K_I, \mathrm{ind})$; $z \leftarrow F_p(K_Z, w \parallel c)$
    $l \leftarrow F(\mathsf{T}_w, c)$; $e \leftarrow \mathrm{Enc}(K_e, x_w || \mathrm{ind})$; $y \leftarrow \mathrm{xind} \cdot z^{-1}$
    $\mathsf{TSet}[l] = (e, y)$; $C_w \leftarrow C_w \cup \{e\}$
    $\mathrm{xtag}_w \leftarrow g^{F_p(K_X, w) \cdot \mathrm{xind}}$; $\mathsf{XSet} \leftarrow \mathsf{XSet} \cup \{\mathrm{xtag}_w\}$
    $c \leftarrow c + 1$
  end for
  Compute Bloom filter value: $\mathsf{BF}_\mathsf{w}(C_w)$
  $l \leftarrow F(\mathsf{T}_w, 0)$, $\mathsf{TSet}[l] \leftarrow (\mathsf{BF}_\mathsf{w}(C_w), H_{K_e}(|C_w|, \mathsf{BF}_\mathsf{w}(C_w)))$
end for
Compute Bloom filter values: $\mathsf{BF}_\mathsf{X}(\mathsf{XSet})$, $\mathsf{BF}_\mathsf{T}(\mathsf{Token})$
Compute vector commitment:
$C_R \leftarrow \prod_{i=1}^{q} h_i^{v_i}$, where $v_i || DB_{w_{P(i)}}$ and $i \in [1, q]$
$C^{(0)} = C_R$, $C_{-1} = C_R$, $H_0 \leftarrow \mathcal{H}(C_{-1}, C^{(0)}, 0)^y$
$\mathsf{aux} \leftarrow (H_0, C_{-1}, C^{(0)}, 0)$
Set $\mathsf{VDB} \leftarrow \{\mathsf{TSet}, \mathsf{XSet}, \mathsf{Token}, \mathsf{BF}_\mathsf{X}(\mathsf{XSet}), \mathsf{BF}_\mathsf{T}(\mathsf{Token})\}$
$\mathsf{PK} = (\mathsf{PP}, Y, C_R, C_0, \mathsf{BF}, \phi)$
$\mathsf{S} = (\mathsf{PP}, \mathsf{aux}, DB, \mathsf{TSet})$
return {VDB, PK, S}

- $\mathsf{TokenGen}(\bar{\mathsf{w}}, \mathsf{SK})$: Assume that the client would like to perform search over a given keyword set $\bar{\mathsf{w}} = (w_1, w_2, \cdots, w_d)$. For simplicity, w_1 can be seen as the sterm, the search token $T_{\bar{\mathsf{w}}}$ is generated in Algorithm 2.

Algorithm 2 $\mathsf{TokenGen}(\bar{\mathsf{w}}, \mathsf{SK})$.

Input $\mathsf{T}_{\bar{w}}$, SK
Output $\mathsf{T}_{\bar{w}}$

On input the key $(K_S, K_X, K_I, K_Z, K_T)$ and the queried keyword $\bar{w} = (w_1, \cdots, w_d)$,the client generates the search token as follows:
$T_q \leftarrow F(K_S, w_1)$
for $c = 1, 2, \ldots$ until the server stops **do**
  for $i = 2, \ldots, d$ **do**
    $\mathrm{xtoken}[c, i] \leftarrow g^{F_p(K_Z, w_1 || c) \cdot F_p(K_X, w_i)}$
  end for
  $\mathrm{xtoken}[c] = \mathrm{xtoken}[c, 2], \ldots, \mathrm{xtoken}[c, d]$
end for
$\mathsf{T}_{\bar{w}} \leftarrow (T_q, \mathrm{xtoken}[1], \mathrm{xtoken}[2], \ldots)$

- $\mathsf{Query}(\mathsf{T}_{\bar{w}}, \mathsf{VDB}, \mathsf{PK})$: On receiving the client's search token $T_{\bar{\mathsf{w}}}$, the server performs search over VDB and returns the result R along with the corresponding proof proof in Algorithm 3.
- $\mathsf{Verify}(\mathsf{R}_{w_1}(\mathsf{R}), \mathsf{proof}, \tau)$: The verification of search result can be divided into two steps: Firstly, the verifier check the completeness and correctness of document identifiers; Secondly, the integrity of document contents will be done. The details of this process is shown in Algorithm 4.
- $\mathsf{Update}(\mathsf{SK}, w, x, v_x')$: This algorithm is the same as the one in the basic construction. Thus, we omit it here.

Algorithm 3 $\mathsf{Query}(\mathsf{T}_{\tilde{w}}, \mathsf{VDB}, \mathsf{PK})$.

Input $\mathsf{T}_{\tilde{w}}$, VDB, PK
Output R_{w_1}, R, proof

R_{w_1}, R, B, proof $\leftarrow \phi$
$l \leftarrow F(T_{w_1}, 0)$
$(\mathsf{BF}_{\mathsf{w}}(C_{w_1}), H(K_e, |C_{w_1}|, \mathsf{BF}_{\mathsf{w}}(C_{w_1}))) \leftarrow \mathsf{TSet}[l])$
for $c = 1, 2, \cdots$ **do**
  $l \leftarrow F(T_{w_1}, c)$
  $(e_c, y_c) \leftarrow \mathsf{TSet}[l]$
  $\mathsf{R}_{w_1} \leftarrow \mathsf{R}_{w_1} \cup \{e_c\}$
end for
if $\mathsf{R}_{w_1} = \phi$ **then**
  $\mathsf{proof}_1 \leftarrow \mathsf{BF}_{\mathsf{T}}(\mathsf{Token})$
  return proof_1 and exit
else
  $x_w \leftarrow Dec(K_e, e_c)$ $(\forall e_c \in \mathsf{R}_{w_1})$
  $\pi_{x_w}^{(T)} \leftarrow \prod_{1 \le j \le q, j \ne x_w} h_{x_w,j}^{v_j^{(T)}}$
  $\tau = (v_{x_w}^{(T)}, \pi_{x_w}^{(T)}, H_T, C_{T-1}, C^{(T)}, T)$
end if
for $c = 1, \cdots, |\mathsf{R}_{w_1}|$ **do**
  for $i = 2, \cdots, d$ **do**
    $b[c, i] \leftarrow \mathsf{xtoken}[c, i]^{y_c}$
  end for
  if $\forall i = 2, \ldots, d : b[c, i] \in \mathsf{XSet}$ **then**
    $\mathsf{R} \leftarrow \mathsf{R} \cup \{e_c\}$; $\mathsf{B} \leftarrow \mathsf{B} \cup \{b[c, i]\}$
  end if
end for
if $\mathbf{R} \ne \phi$ **then**
  $\mathsf{proof}_2 \leftarrow \mathsf{BF}_{\mathsf{X}}(\mathsf{XSet})$
end if
$\mathsf{proof} \leftarrow \{\mathsf{BF}_{\mathsf{w}}(C_w), \{\mathsf{BF}_{\mathsf{X}}(\mathsf{XSet}), \{\mathsf{proof}_i\}_{i=1,2}\}$
return $(\mathsf{R}_{w_1}, \mathsf{R}, \tau, \mathsf{proof})$

Algorithm 4 $\mathsf{Verify}(\mathsf{R}_{w_1}(\mathsf{R}), \mathsf{proof}, \tau)$.

Input $\mathsf{R}_{w_1}(\mathsf{R})$, proof, τ
Output *Accept* or *Reject*

Verifiability of Document Identities:
Case 1: $\mathsf{R}_{w_1} = \Phi$
$\mathsf{BF}_{\mathsf{w}_1}(T_w) = 1$
Case 2: $\mathsf{R}_{w_1} \ne \Phi$
$H_{K_e}(|C_{w_1}|, \mathsf{BF}_{\mathsf{w}}(C_{w_1})) \stackrel{?}{=} H_{K_e}(|\mathsf{R}_{w_1}|, \mathsf{BF}_{\mathsf{w}}(\mathbf{C}_{w_1}))$
for $i = 1, 2, \cdots, |\mathsf{R}_{w_1}|$ **do**
  if $\mathsf{BF}_{\mathsf{w}}(ind_i) \ne 1$ **then**
    Reject and exit
  end if
end for
if $\mathsf{R} = \Phi$ **then**
  for $i = 1, 2, \cdots, |\mathsf{R}_{w_1}|$ **do**
    $\mathsf{xind} \leftarrow F_p(K_I, \mathrm{ind}_i)$
    for $j = 2, \cdots, d$ **do**
      $\mathsf{xtag}[i, j] \leftarrow g^{F_p(K_X, w_j) \cdot \mathsf{xind}}$
      $\mathsf{xtag}[i] \leftarrow \mathsf{xtag}[i] \cup \mathsf{xtag}[i, j]$
    end for
    if $\mathbf{BF_X}(\mathsf{xtag[i]}) = 1$ **then**
      return *Reject* and exit
    end if
  end for
else
  The client select all $ind \in \mathsf{R}_{w_1} - \mathsf{R}$
  The remaining operations are the same as the above case.
end if
Verifiability of Document Contents:
The verifier checks the following two equations:
$e(H_T, g) = e(\mathcal{H}(C_{T-1}, C^{(T)}, T), Y)$
$e(C_T / H_T h_x^{v_x^{(T)}}, h_x) = e(\pi_x^{(T)}, g)$
return *Accept* or *Reject*

4. Security and efficiency analysis

4.1. Security analysis

Theorem 4.1. *The Tset instantiation* Σ *used in our scheme is* $\mathcal{L}_T$*-adaptively secure by assuming that F is secure pseudorandom function.*

Proof 1. Similar to [10], we present the security analysis by designing a simulator. That is, we prove that $\mathbf{Real}_{\mathcal{A}}^{\Sigma}(\lambda)$ is computationally indistinguishable from $\mathbf{Ideal}_{\mathcal{A},\mathcal{S}}^{\Sigma}(\lambda)$. In other words, it means that a simulator $\mathcal{S}$ can exactly simulate the whole protocol with only the leakage information, thus completes the proof.

The simulator algorithm $\mathcal{S}$ can be performed as follows: $\mathcal{S}$ takes as input the leakage $\mathcal{L}_T = (|DB_w|, N = \sum_{w \in W} |DB_w|)$ and all the transcripts of adversary's query $T[q]$. The initialization $\mathcal{S}(\mathcal{L}_T)$ generates *TSet* as an N array just like the TSetSetup algorithm of Real Game. What is different is that $\mathcal{S}$ fills all the element of array with the same length random bitstrings. More specifically, $\mathcal{S}$ generates a two-dimensional array $T[w, c]$ to store all the N records. (It is used to make a response of $\mathcal{A}$.) Then, $\mathcal{S}$ sends *TSet* to $\mathcal{A}$. For each query q from $\mathcal{A}$, $\mathcal{S}$ retrieves a list $t = (T[q, 1], \cdots, T[q, |DB_q|])$. If the element $F(stag_w, c)$ was queried before assigning value, $\mathcal{S}$ aborts. Note that the F is a pseudo random function, it is indistinguishable between the Real Game and the modified one where $stag_w$ are replaced by random elements from the range of F. On the other hand,

Table 1
Efficiency comparison.

Schemes	Scheme [11]	Scheme [12]	Our scheme
Model	Amortized	Amortized	Amortized
Assumption	CDH	CDH	CDH
ResistFAUAttack	×	√	√
Accountability	×	√	√
KeywordSearch	×	×	√
querycost(Server)	$(q-1)(E+M)$	$(q-1)(E+M)$	$(q-1)(E+M)+D+H$
Verifycost(Verifier)	$E+M+I+2P$	$E+2M+I+4P$	$2E+2M+I+4P+O(\log_2 q)H$
Updatecost(Client)	$E+M$	$2E+2M+I$	$2E+2M+I+2H$

because any given $T[w, c]$ is assigned randomly to the *TSet*, the view of $\mathcal{A}$ of each $T[w, c]$ is identical to the Real Game. Thus, we prove that simulator $\mathcal{S}$ can perfectly simulate the protocol.

Theorem 4.2. *The proposed VDB scheme can achieve secure keyword search with the Squ-CDH assumption.*

Proof 2. Similar to [11], we present the security proof by using reduction method. Suppose that the proposed VDB scheme cannot achieve verifiable update in the case of keyword search, which means that there exists a adversary $\mathcal{A}$ that can convince the client on a false opening with a non-negligible advantage ε. Thus, we can build a efficient algorithm $\mathcal{A}'$ that can use $\mathcal{A}$ as a subroutine to break the Squ-CDH assumption. That is, on input (g, g^a), $\mathcal{A}'$ can output g^{a^2}.

$\mathcal{A}'$ firstly selects a keyword w and then chooses the corresponding index $i \in_R \mathbb{Z}_q$ as a guess for the index i, where $\mathcal{A}$ can break the position binding. Then, $\mathcal{A}'$ randomly chooses $z_j \in \mathbb{Z}_p$ for each $j \in [1, q]$ and $j \neq i$, and computes the following values:

$\forall j \in [1, q], j \neq i, h_j = g^{z_j}, h_{i,j} = (g^a)^{z_j}, h_i = g^a$

$\forall k, j \in [1, q], k, j \neq i, h_{k,j} = g^{z_k z_j}$

$\mathcal{A}'$ sets $\mathsf{PP} = (p, q, \mathbb{G}_1, \mathbb{G}_2, \mathcal{H}, e, g, \{h_i\}, \{h_{i,j}\})$, where $j \in [1, q]$ and $j \neq i$. In addition, $\mathcal{A}'$ randomly selects private key SK as $y \in_R \mathbb{Z}_q$ and generates $Y = g^y$. Given a database DB with the keyword set W, $\mathcal{A}'$ generates the corresponding keyword-based index by simply running the $\mathsf{TSetSetup}$ algorithm. Then, he generates the Bloom filter BF and Merkel hash tree MHT with the information on keyword and the corresponding document identities. Finally, $\mathcal{A}'$ computes the root commitment $C_R = \prod_{i=1}^{q} h_i^{v_i}$, the signature $H_0 = \mathcal{H}(C_R, C_R, 0)^y$ and the current public key $C_0 = H_0 C_R$. Set $\mathsf{PK} = (\mathsf{PP}, Y, C_R, C_0, \mathsf{BF}, \mathsf{MHT}, \phi)$ and $\mathsf{S} = (\mathsf{PP}, \mathsf{aux}, DB, \mathsf{TSet})$. $\mathcal{A}'$ sends it to $\mathcal{A}$. Note that PK and S have the same distribution as the real protocol. $\mathcal{A}'$ simply runs the real $\mathsf{Query}(\mathsf{PK}, \mathsf{S}, w, i)$ and $\mathsf{Update}(\mathsf{SK}, w, i, v_i')$ algorithms to answer the `verify` and `update` queries of $\mathcal{A}$. Due that the $\mathsf{Update}(\mathsf{SK}, w, x, v_i')$ algorithm needs to keep the private key, anyone cannot perform the update operation except the $\mathcal{A}'$. Thus, it can resist on the FAU attack as stated in [12].

Assume that (i^*, τ^*) is the returned tuple by the adversary $\mathcal{A}$, where $\tau^* = (v^*, \pi^*, \Sigma_T)$. If $\mathcal{A}$ can win in breaking the security of VDB scheme, we have $v^* \neq \perp$, $v^* \neq v_{i^*}^{(T)}$ and $e(C^{(T)}, h_{i^*}) = e(h_{i^*}^{v_{i^*}^{(T)}}, h_{i^*}) e(\pi_{i^*}^{(T)}, g) = e(h_{i^*}^{v^*}, h_{i^*}) e(\pi^*, g)$.

If $i \neq i^*$, $\mathcal{A}'$ aborts the simulation. Otherwise, with $h_i = g^a$, we can compute

$$g^{a^2} = (\pi^* / \pi_{i^*}^{(T)})^{(v_{i^*}^{(T)} - v^*)^{-1}}.$$

The success probability of $\mathcal{A}'$ is ϵ/q.

4.2. Comparison

We give the comparison of computational cost among the proposed scheme, Catalano-Fiore's scheme [11] and Chen et al.'s scheme [12]. In Setup phase, all the above mentioned schemes need to perform some expensive operations for generating system parameters. Although our scheme introduces slight computation overload to build the searchable index, it is a one-time work and can be done in an off-line way. Besides, our construction can keep the benefit of resisting on FAU attack as [12], while supporting keyword-based search in VDB setting. The computation cost on client side is independent with the whole size of the oursourced database based on vector commitment technique. More precisely, the client just requires to perform a few (static) computation operations in data update and verify phase, the server invests computation of proof for updated record, which is the most expensive computation overhead (All three schemes hold this property).

We present the comparison on the computation overhead among the abovementioned schemes in Table 1. We mainly focus on the computation cost in data update and verify phase. Some notations are introduced. We denote M as the multiplication operation in group $\mathbb{G}_1$ (or $\mathbb{G}_2$), by E a modular exponentiation in $\mathbb{G}_1$, I an inversion operation in $\mathbb{G}_1$, D a decryption operation of symmetric encryption, H an operation of hash, P a pairing operation, and F a pseudo-random function. Here, some lightweight operations are omitted. We remark that our scheme can achieve keyword search for VDB by introducing some slightly computation overload. That is, the client just needs to a hash operation to generate the search token and a decrypt operation to obtain the indicator (index of vector commitment) in the phrase of Search. In addition, the client performs hash operation on the documents at the beginning of verification.

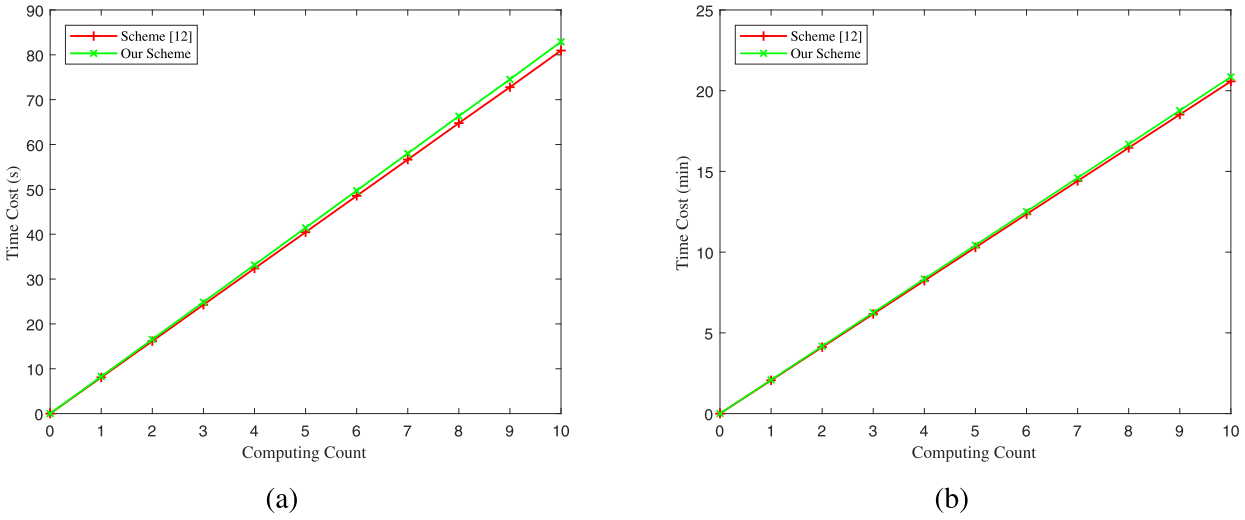

Fig. 2. Time cost of Query for both datasets: (a) Enron dataset; (b) The synthetic dataset.

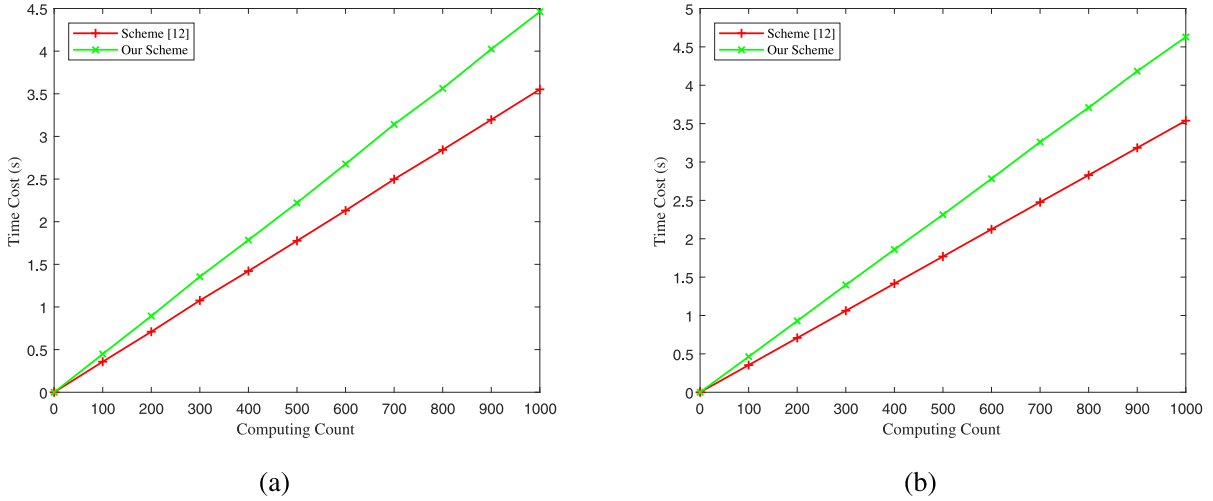

Fig. 3. Time cost of Verify for both datasets: (a) Enron dataset; (b) The synthetic dataset.

5. Performance evaluation

In this section, we provide an overall experimental evaluation of the proposed basic construction. The experiments are deployed by invoking Java Pairing-based Cryptography (JPBC) library and OpenSSL library on a LINUX PC equipped with Intel Xeon E5-1620 3.50 GHz and 16G memory.

Our experiments are conducted based on a real-world dataset: Enron Email Dataset [8] and a synthetic dataset. Specifically, we choose all the keywords that belong to less than 20 documents to construct the experimental database. The total number of keywords is $q = 6459$ and the keyword-document pairs is 65429. For the synthetic dataset, we assume that $q = 100,000$. It means that there are 100,000 distinct keywords in the whole database.

In the following experiments, we mainly focus on computation overhead comparison between our basic construction and Chen et al. scheme [12]. Note that our scheme can be seen as an extension for keyword search based on [12]. We provide the performance evaluation of Query, Verify and Update algorithms for our scheme and Chen et al. [12]. The detailed experimental results are shown in Figs. 2–4, respectively.

Fig. 2 shows the computation overhead of the Query algorithm based on different dataset. It can be clearly seen that the computational overhead of our scheme is slightly more expensive than that of scheme [12]. The reason is that our scheme requires to perform additional keyword search operations. Note that the query overhead increases with the size of vector commitment, i.e., q.

As shown in Fig. 3, the verification cost of both the two schemes is linear with the counting count. Our construction requires more computation cost than scheme [12]. This is because that our construction performs 1 signature verification and

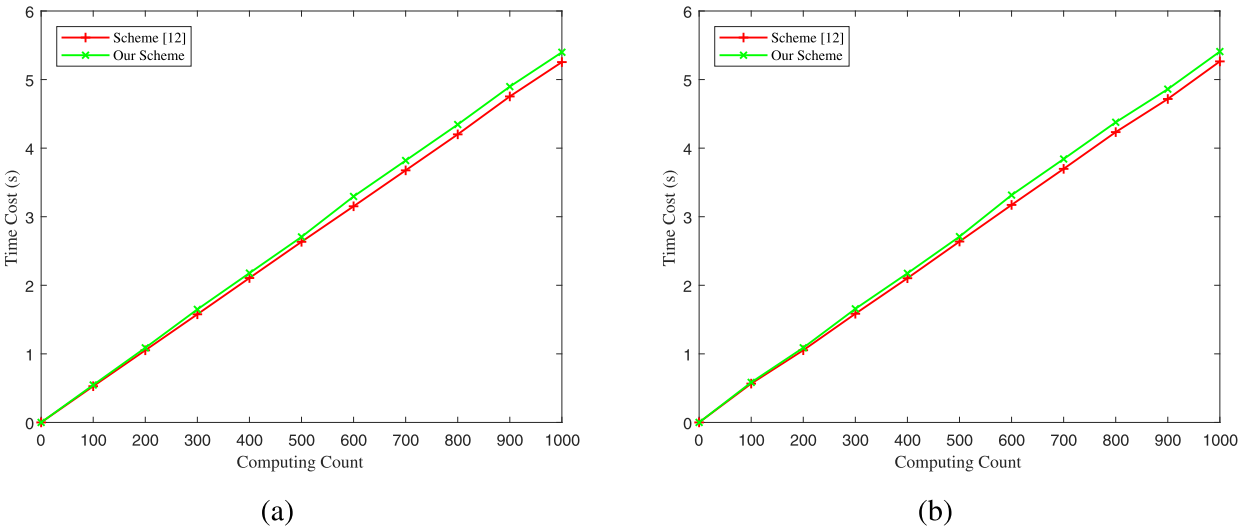

Fig. 4. Time cost of Update for both datasets: (a) Enron dataset; (b) The synthetic dataset.

$O(\log_2 q)$ hash operation to verify the integrity of document identifier. Note that our scheme can support kinds of keyword search, which makes it more suitable to be applied in real-world data outsourcing scenario.

In Fig. 4, we provide the performance comparison of Update operation in client side. The experimental results show that our scheme is as efficient as that of scheme [12]. Actually, our scheme requires just a few additional hash operation and it has almost no effect on the system performance.

6. Conclusion

The notion of VDB is a useful primitive to address the challenge of verifiable outsourcing of storage in cloud computing. Nevertheless, none of the existing solutions can support efficient keyword search on the database. In this paper, we propose the first VDB scheme that also achieved efficient keyword search by incorporating the primitives of VDB and SSE. Furthermore, we show that our construction can be easily extended to multi-keyword setting. Security and efficiency analysis demonstrate that the proposed VDB scheme can not only achieve the desired security properties but also provide a comparable overhead for real applications.

Acknowledgments

This work is supported by National Natural Science Foundation of China (Nos. 61572382, 61702401, and U1405255), China 111 Project (No. B16037), China Postdoctoral Science Foundation (No. 2017M613083), Natural Science Basic Research Plan in Shaanxi Province of China (Nos. 2016JZ021 and 2018JQ6001).

References

[1] D. Agrawal, A. El Abbadi, F. Emekçi, A. Metwally, Database management as a service: challenges and opportunities, in: Proceedings of the 25th International Conference on Data Engineering, ICDE'09, IEEE, 2009, pp. 1709–1716.
[2] B. Applebaum, Y. Ishai, E. Kushilevitz, From secrecy to soundness: efficient verification via secure computation, in: Proceedings of the 37th International Colloquium on Automata, Languages and Programming, ICALP'10, Springer, 2010, pp. 152–163.
[3] M. Azraoui, K. Elkhiyaoui, M. Önen, R. Molva, Publicly verifiable conjunctive keyword search in outsourced databases, in: 2015 IEEE Conference on Communications and Network Security, CNS'15, IEEE, 2015, pp. 619–627.
[4] F. Bao, R.H. Deng, H. Zhu, Variations of diffie-hellman problem, in: Proceedings of the 5th International Conference on Information and Communications Security, ICICS'03, 2003, pp. 301–312.
[5] S. Benabbas, R. Gennaro, Y. Vahlis, Verifiable delegation of computation over large datasets, in: Advances in Cryptology,CRYPTO'11, Springer, 2011, pp. 111–131.
[6] R. Bost, P.-A. Fouque, D. Pointcheval, Verifiable dynamic symmetric searchable encryption: Optimality and forward security, 2016, CryptologyePrint Archive, Report 2016/062). https://eprint.iacr.org/2016/062.
[7] R. Bost, B. Minaud, O. Ohrimenko, Forward and backward private searchable encryption from constrained cryptographic primitives, in: Proceedings of the 24th ACM Conference on Computer and Communications Security, CCS'17, ACM, 2017, pp. 1465–1482.
[8] CALO, Enron Email Dataset, (http://www.cs.cmu.edu/~enron/). Accessed April 10, 2018.
[9] D. Cash, J. Jaeger, S. Jarecki, C.S. Jutla, H. Krawczyk, M. Rosu, M. Steiner, Dynamic searchable encryption in very-large databases: data structures and implementation, in: Proceedings of the 21st Annual Network and Distributed System Security Symposium, NDSS'14, The Internet Society, 2014, pp. 23–26.
[10] D. Cash, S. Jarecki, C.S. Jutla, H. Krawczyk, M. Rosu, M. Steiner, Highly-scalable searchable symmetric encryption with support for boolean queries, in: Advances in Cryptology, CRYPTO'13, Springer, 2013, pp. 353–373.

[11] D. Catalano, D. Fiore, Vector commitments and their applications, in: Proceedings of the 16th International Conference on Practice and Theory in Public-Key Cryptography, PKC'13, Springer, 2013, pp. 55–72.
[12] X. Chen, J. Li, X. Huang, J. Ma, W. Lou, New publicly verifiable databases with efficient updates, IEEE Trans. Dependable Sec. Comput. 12 (5) (2015) 546–556.
[13] X. Chen, J. Li, J. Ma, Q. Tang, W. Lou, New algorithms for secure outsourcing of modular exponentiations, IEEE Trans. Parallel Distrib. Syst. 25 (9) (2014) 2386–2396.
[14] X. Chen, J. Li, J. Weng, J. Ma, W. Lou, Verifiable computation over large database with incremental updates, IEEE Trans. Comput. 65 (10) (2016) 3184–3195.
[15] R. Curtmola, J.A. Garay, S. Kamara, R. Ostrovsky, Searchable symmetric encryption: improved definitions and efficient constructions, in: Proceedings of the 13th ACM Conference on Computer and Communications Security, CCS'06, 2006, pp. 79–88.
[16] S. Faber, S. Jarecki, H. Krawczyk, Q. Nguyen, M. Rosu, M. Steiner, Rich queries on encrypted data: beyond exact matches, in: Proceedings of the 20th European Symposium on Research in Computer Security, Computer Security, ESORICS'15, Springer, 2015, pp. 123–145.
[17] B.A. Fisch, B. Vo, F. Krell, A. Kumarasubramanian, V. Kolesnikov, T. Malkin, S.M. Bellovin, Malicious-client security in blind seer: a scalable private DBMS, in: 2015 IEEE Symposium on Security and Privacy, SP'15, IEEE, 2015, pp. 395–410.
[18] C. Gao, S. Lv, Y. Wei, Z. Wang, Z. Liu, X. Cheng, M-SSE: an effective searchable symmetric encryption with enhanced security for mobile devices, IEEE Access 6 (2018) 38860–38869.
[19] R. Gennaro, C. Gentry, B. Parno, Non-interactive verifiable computing: outsourcing computation to untrusted workers, in: Advances in Cryptology, CRYPTO'10, Springer, 2010, pp. 465–482.
[20] S. Kamara, T. Moataz, Boolean searchable symmetric encryption with worst-case sub-linear complexity, in: Advances in Cryptology, EUROCRYPT'17, Springer, 2017, pp. 94–124.
[21] S. Kamara, C. Papamanthou, Parallel and dynamic searchable symmetric encryption, in: Proceedings of the 17th International ConferenceFinancial Cryptography and Data Security, FC'13, Springer, 2013, pp. 258–274.
[22] K.S. Kim, M. Kim, D. Lee, J.H. Park, W. Kim, Forward secure dynamic searchable symmetric encryption with efficient updates, in: Proceedings of the 24th ACM Conference on Computer and Communications Security, CCS'17, ACM, 2017, pp. 1449–1463.
[23] F. Krell, G. Ciocarlie, A. Gehani, M. Raykova, Low-leakage secure search for boolean expressions, in: Proceedings of the Cryptographers' Track at the RSA Conference 2017 Topics in Cryptology, CT-RSA'17, Springer, 2017, pp. 397–413.
[24] K. Kurosawa, Y. Ohtaki, How to update documents verifiably in searchable symmetric encryption, in: Proceedings of the 12th International Conference on Cryptology and Network Security, CANS'13, Springer, 2013, pp. 309–328.
[25] H. Li, D. Liu, Y. Dai, T.H. Luan, X.S. Shen, Enabling efficient multi-keyword ranked search over encrypted mobile cloud data through blind storage, IEEE Trans. Emerg. Top. Comput. 3 (1) (2015) 127–138.
[26] H. Li, D. Liu, Y. Dai, T.H. Luan, S. Yu, Personalized search over encrypted data with efficient and secure updates in mobile clouds, IEEE Trans. Emerg. Top. Comput. 6 (1) (2018) 97–109.
[27] J. Li, X. Chen, M. Li, J. Li, P.P.C. Lee, W. Lou, Secure deduplication with efficient and reliable convergent key management, IEEE Trans. Parallel Distrib. Syst. 25 (6) (2014) 1615–1625.
[28] J. Li, X. Huang, J. Li, X. Chen, Y. Xiang, Securely outsourcing attribute-based encryption with checkability, IEEE Trans. Parallel Distrib. Syst. 25 (8) (2014) 2201–2210.
[29] J. Li, Z. Liu, X. Chen, F. Xhafa, X. Tan, D.S. Wong, L-Encdb: a lightweight framework for privacy-preserving data queries in cloud computing, Knowl.-Based Syst. 79 (2015) 18–26.
[30] M. Miao, J. Ma, X. Huang, Q. Wang, Efficient verifiable databases with insertion/deletion operations from delegating polynomial functions, IEEE Trans. Inf. Forensics Secur. 13 (2) (2018) 511–520.
[31] M. Miao, J. Wang, J. Ma, W. Susilo, Publicly verifiable databases with efficient insertion/deletion operations, J. Comput. Syst. Sci. 86 (2017) 49–58.
[32] D.X. Song, D.A. Wagner, A. Perrig, Practical techniques for searches on encrypted data, in: 2000 IEEE Symposium on Security and Privacy, SP'00, Springer, 2000, pp. 44–55.
[33] E. Stefanov, C. Papamanthou, E. Shi, Practical dynamic searchable encryption with small leakage, in: Proceedings of the 21st Annual Network and Distributed System Security Symposium, NDSS'14, The Internet Society, 2014, pp. 72–75.
[34] S. Sun, J.K. Liu, A. Sakzad, R. Steinfeld, T.H. Yuen, An efficient non-interactive multi-client searchable encryption with support for boolean queries, in: Proceedings of the 21st European Symposium on Research in Computer Security, ESORICS'16, in: LNCS, 9878, Springer, 2016, pp. 154–172.
[35] H. Wang, Identity-based distributed provable data possession in multicloud storage, IEEE Trans. Serv. Comput. 8 (2) (2015) 328–340.
[36] J. Wang, X. Chen, X. Huang, I. You, Y. Xiang, Verifiable auditing for outsourced database in cloud computing, IEEE Trans. Comput. 64 (11) (2015) 3293–3303.
[37] J. Wang, X. Chen, J. Li, J. Zhao, J. Shen, Towards achieving flexible and verifiable search for outsourced database in cloud computing, Future Gener. Comp. Syst. 67 (2017) 266–275.
[38] Y. Zhang, R.H. Deng, J. Shu, K. Yang, D. Zheng, TKSE: trustworthy keyword search over encrypted data with two-side verifiability via blockchain, IEEE Access 6 (2018) 31077–31087.
[39] Z. Zhang, X. Chen, J. Li, X. Tao, J. Ma, Hvdb: a hierarchical verifiable database scheme with scalable updates, J. Ambient Intell. Human. Comput. (2018) 1–13, doi:10.1007/s12652-017-0523-3.