Journal Pre-proof


# Verifiable Database Supporting Keyword Searches with Forward Security

Meixia Miao, Yunling Wang, Jianfeng Wang, Xinyi Huang

PII: S0920-5489(20)30378-0
DOI: https://doi.org/10.1016/j.csi.2020.103491
Reference: CSI 103491

To appear in: *Computer Standards & Interfaces*

Received date: 3 August 2020
Revised date: 3 October 2020
Accepted date: 20 October 2020

Please cite this article as: Meixia Miao, Yunling Wang, Jianfeng Wang, Xinyi Huang, Verifiable Database Supporting Keyword Searches with Forward Security, *Computer Standards & Interfaces* (2020), doi: https://doi.org/10.1016/j.csi.2020.103491

1. We propose a novel VDB framework for satisfying forward-secure keyword search. The proposed construction can achieve verifiability of the search result and database content simultaneously.
2. We present a concrete keyword-based VDB scheme by incorporating delegating polynomial functions and verifiable forward-secure SSE. Our proposed scheme can not only achieve forward-secure keyword search over VDB, but also support all updating operations (i.e., replacement/insertion/deletion).

# Verifiable Database Supporting Keyword Searches with Forward Security

Meixia Miao[a,*], Yunling Wang[a], Jianfeng Wang[b], Xinyi Huang[c]

[a]*School of Cyberspace Security,*
*Xi'an University of Posts and Telecommunications, Xi'an 710121, China*
[b]*State Key Laboratory of Integrated Service Networks,*
*Xidian University, Xi'an 710071, China*
[c]*School of Mathematics and Computer Science,*
*Fujian Normal University, Fuzhou 350117, China*

**Abstract**

The primitive of verifiable database (VDB) enables the integrity of database when a resource-limited client outsources a huge dynamic database on an untrusted cloud server. Meanwhile, the client could perform various types of queries on the database such as index-based queries and keyword-based queries. In the real-world applications, it is more suitable to consider VDB schemes supporting full updating operations (i.e., replacement, insertion, and deletion). Inherently, forward/backward-secure searchable encryption schemes should be adopted to incorporate VDB schemes. In this paper, we first attempt to address the problem of how to perform forward-secure keyword searching on VDB schemes. Moreover, we present a concrete VDB construction that can support forward-secure keyword search and full updating operations. Security and efficiency analysis demonstrate that the proposed VDB schemes can achieve the desired security properties with high efficiency.

*Keywords:* Cloud computing, Verifiable database, Data Updating, Forward-secure Searchable encryption

*Corresponding author
*Email address:* miaofeng415@163.com (Meixia Miao )

## 1. Introduction

In the paradigm of Database-as-a-Service [1, 2, 12, 16, 17, 34], the resource-constrained clients have to outsource their huge database to a cloud server and are incapable of managing the database. As a result, it is very important to guarantee the integrity of the database since the cloud server is not fully trusted. If the database is tempered by the cloud server without being detected, then it is meaningless for the clients to perform any operation on the database. In the last decades, plenty of researchers have devoted great efforts to solve this problem, especially in the scenario of dynamic database. Most of these constructions are based on accumulators [8, 9] or authenticated data structures [22, 23, 24, 29], and thus either rely some strong cryptographic assumptions or expensive operations. This disadvantage makes them impractical for real-world applications.

How to guarantee the integrity of the outsourced dynamic database is firstly considered by Benabbas et al. [4], who introduced the notion of verifiable database with efficient update (VDB). Specifically, the client can detect any malicious temper with the database. To achieve this goal, they use the subgroup membership assumption in composite order bilinear groups to propose the first practical VDB scheme. However, it only supports the private verification by the client. Catalano et al. [10] designed a new VDB scheme supporting publicly verification. That is, a publicly verifiable VDB scheme ensures that any third party has an ability to verify the integrity of the database and it ensures the auditing of the database management. Recently, many flexible VDB schemes [11, 13, 14, 19, 20] have been proposed based on the two initial constructions. Note that VDB is closely related to other two primitives called PDP [3] and POR [18]. The main difference is that PDP and POR only focus on the integrity of dataset instead of huge database.

In the initial constructions of VDB, it only supports the queries on a certain index. That is, given a query index $x$, the server returns the corresponding data value $v_x$ along with the proofs that ensures the validity of $v_x$. However, in the most real-world scenarios, the client may perform various types of queries on the database [27, 30, 31, 32] such as exact match queries, range queries, and the most common keyword-based

queries (or keyword search). Miao et al. [21] firstly introduced how to incorporate the VDB scheme and symmetric searchable encryption (SSE in short). Besides, they proposed a concrete VDB scheme that supports efficient keyword searches. However, this construction utilizes traditional VDB schemes that only support the replacement operation, rather than other updating operations such as insertion or deletion. In the real applications, it is more suitable to use VDB schemes that support all kinds of updating operations [19, 20]. As a result, in the case of inserting/deleting a data record, forward/backward-secure searchable encryption schemes should be adopted to resist the file injection attack [35].[1] To the best of our knowledge, by far there is no research work on VDB constructions that can support forward-secure keyword searches and full updating operations.

### *1.1. Our Contribution*

In this paper, we first address the challenge of performing forward-secure keyword searches on verifiable database with full updating operations. The contributions of this paper are summarised as follows:

- We first propose a novel VDB framework for satisfying forward-secure keyword search. The proposed construction can achieve verifiability of the search result and database content simultaneously. Furthermore, we show that the construction can be instantiated with any forward-secure SSE and VDB scheme.
- We present a concrete keyword-based VDB scheme by incorporating delegating polynomial functions and verifiable forward-secure SSE. Our proposed scheme can not only achieving forward-secure keyword search over VDB, but also supporting all updating operations (i.e., replacement/insertion/deletion).

### *1.2. Related Work*

**Verifiable Database with Updates**. Benabbas et al. [4] proposed the first practical VDB construction by using a cryptographic primitive called delegating polynomial

[1] In section 3, we explain why we could only consider the case of forward-secure searchable encryption.

functions. Nevertheless, their solution only supports the private verifiability since verifying the proof need the client's secret key. To support publicly verify the proof, Catalano and Fiore [10] introduced the first publicly verifiable VDB scheme based on the primitive of vector commitment. However, Chen et al. [13] pointed out that there exists a forward automatic update attack by the malicious server in this scheme. To solve this problem, Chen et al. [14] presented a new VDB scheme with incremental updates (Inc-VDB). We need to mention that all the above VDB schemes [4, 10, 13, 14] only support the update type of data replacement, rather than insertion and deletion. Miao et al. [20] firstly constructed a new efficient VDB schemes based on the primitive of hierarchical vector commitment. Their scheme supports all types of updating operations such as insertion, deletion and replacement. However, if some data records are frequently inserted into the same location, the number of the layer of hierarchical vector commitment is linear increased in their construction. Recently, to solve this problem, Miao et al. [19] firstly presented a new primitive of Merkel sum hash tree which is an extension of Merkel hash tree, and then applied this primitive to construct a new efficient VDB scheme supporting all update operations.

**Forward-secure Symmetric Searchable Encryption**. Stefanov et al. [26] first introduced the notion of forward and backward security for dynamic SSE schemes. Also, they proposed the first ORAM-inspired forward secure SSE scheme. Motivated by the file-injection attack [35], plenty of efforts have been made to designing forward-secure SSE schemes. Bost [6] proposed a new forward-secure SSE scheme which can reduce the computational overhead of search and update in the ORAM-based constructions. However, the construction utilized a public-key cryptographic primitive called the trapdoor permutation as a building block. Consequently, Song et al. [25] gave a new construction which used only the symmetric cryptographic primitive. Recently, Demertzis et al. [15] considered how to reduce the storage overhead on the user side. Note that all the above forward-secure SSE constructions support only single keyword search. To rich the expressiveness, some researchers focused on conjunctive keyword search [33], and range query [37] while preserving forward security. Zhang et al. [36] focused on the malicious server model and constructed a verifiable forward-secure SSE scheme based on multi-set hash functions. For the case of database deletion, Bost et

al. [7] formalized the notion of the backward security for SSE scheme and constructed a non-interactive backward-secure SSE scheme. Sun et al. [28] pointed out that their scheme is impractical for real-world applications because of using the public-key puncturable encryption. To address this problem, they introduced the notion of symmetric puncturable encryption and designed an efficient backward-secure SSE scheme.

### *1.3. Organization*

The rest of this paper is organized as follows. We present some preliminaries in Section 2. In section 3, we give the proposed VDB scheme supporting keyword searches with forward security. Then, the security and efficiency analysis of our proposed VDB scheme are presented in Section 4. Finally, the concluding remarks are given in Section 5.

## 2. Preliminaries

In this section, we introduce some preliminaries such as bilinear pairings over composite order groups [5], verifiable dynamic symmetric searchable encryption [36], and verifiable database.

### *2.1. Bilinear Pairings over Composite Order Groups*

Let $p_1, p_2$ be two large primes. Let $\mathbb{G}$ and $\mathbb{G}_T$ be two cyclic multiplicative groups of order $N = p_1p_2$. Let $g$ be a generator of $\mathbb{G}$. A bilinear pairing is a map $e : \mathbb{G}\times\mathbb{G} \rightarrow \mathbb{G}_T$ that satisfies the following properties

1. Bilinear: $e(u^a, v^b) = e(u, v)^{ab}$ for all $u, v \in \mathbb{G}$, and $a, b \in \mathbb{Z}_N$.
2. Non-degenerate: $e(g, g) \neq 1$ is a generator of $\mathbb{G}_T$.
3. Computable: It is efficient to compute $e(u, v)$ for all $u, v \in \mathbb{G}$.

The subgroup decision problem in $(\mathbb{G}, \mathbb{G}_T)$ is defined as follows:

**Definition 1.** *Subgroup Decision Problem: Let $\mathcal{G} = (\mathbb{G}, \mathbb{G}_T, e)$ be bilinear groups of order $N = p_1p_2$. Given an element $x \in \mathbb{G}$, output '1' if the order of $x$ is $p_1$ and output '0' otherwise; That is, without knowing the factorization of $N$, decide whether*

*an element $x \in \mathbb{G}$ is in a subgroup of $\mathbb{G}$ or not. We say that the subgroup decision assumption holds if for every PPT distinguisher $\mathcal{A}$ there exists a negligible function neg(.) such that for all $\lambda$,*

$$|Pr[\mathcal{A}(N, \mathcal{G}, x) = 1] - Pr[\mathcal{A}(N, \mathcal{G}, x^{p_2}) = 1]| \leq neg(\lambda).$$

### *2.2. Verifiable Dynamic Symmetric Searchable Encryption*

Given an encrypted database $DB = (x, v_x)$ which consists of non-empty values $v_x$ for each cell index $1 \leq x \leq q$. Let $\mathsf{ind}_x$ be the distinct indices of values $v_x$ (trivially there is a bijection between $\mathsf{ind}_x$ and $v_x$). Let $\mathsf{W}_x$ be a set of keywords that matching $v_x$. Then, we can define an inherent database $\mathsf{DB} = \{(\mathsf{ind}_x, \mathsf{W}_x) : 1 \leq x \leq q\}$.

**Definition 2.** *A verifiable dynamic searchable symmetric encryption (VDSSE) [36] scheme is a triple* $\Pi = (\mathsf{VDSSE.Setup}, \mathsf{VDSSE.Search}, \mathsf{VDSSE.Update}, \mathsf{VDSSE.Verify})$ *that consists of one algorithm and three protocols:*

- $\mathsf{VDSSE.Setup}(\lambda, \perp; \perp)$*: In this protocol, the client takes the security parameter $\lambda$ as input, and outputs the secret keys $k_s$, $k_r$ and an empty map $\Sigma$. The server outputs an empty map* $\mathrm{T}$.
- $\mathsf{VDSSE.Search}(k_s, w, \Sigma; \mathrm{T})$*: A protocol between the client and server. Given the $k_s$, $w$, and $\Sigma$, the client generates the search token $(t_w, st_c)$ and sends to the server. The server performs search in the* $\mathrm{T}$ *and returns the search result* $\mathrm{R}$ *and a corresponding proof $proof$.*
- $\mathsf{VDSSE.Update}(k_s, k_r, \mathrm{ind}, w, op, \Sigma; \mathrm{T})$*: A protocol between the client and server. Given an operation* $\mathsf{op}$ *of insertion or deletion with its input* $\mathrm{ind}, w$*, the input of client and server is $(k_s, k_r, \mathrm{ind}, w, op, \Sigma)$ and* $\mathrm{T}$*, separately. The output of this protocol is $\Sigma'$ in the client side and $\mathrm{T}'$ in the server side.*
- $\mathsf{VDSSE.Verify}(k_r, w, \Sigma, \mathrm{R}, proof)$*: This algorithm is to verify the correctness and the completeness of the search result* $\mathrm{R}$*. If the* $\mathrm{R}$ *passes the verification, this algorithm outputs $Accept$. Otherwise, it outputs $Reject$.*

**Security**: Loosely speaking, the security of VDSSE means that the adversary could not obtain any information about the database and queries beyond some explicit leakage. Formally, we have the following definition.

**Definition 3.** *A verifiable dynamic searchable symmetric encryption (VDSSE) is $\mathcal{L}$-adaptively-secure, with respect to a leakage function $\mathcal{L} = (\mathcal{L}^{\mathrm{Stp}}, \mathcal{L}^{\mathrm{Srch}}, \mathcal{L}^{\mathrm{Updt}}, \mathcal{L}^{\mathrm{Verify}})$, if for any polynomial-time adversary issuing polynomial number of queries $\Omega(\lambda)$, there exists a probabilistic polynomial-time simulator $\mathcal{S}$ such that*

$$|\Pr[\mathbf{Real}^{\Sigma}_{\mathcal{A}}(\lambda, \Omega) = 1] - \Pr[\mathbf{Ideal}^{\Sigma}_{\mathcal{A},\mathcal{S},\mathcal{L}}(\lambda, \Omega) = 1]| \leq negl(\lambda),$$

*where $\mathbf{Real}^{\Sigma}_{\mathcal{A}}(\lambda, \Omega)$ and $\mathbf{Ideal}^{\Sigma}_{\mathcal{A},\mathcal{S},\mathcal{L}}(\lambda, \Omega)$ are defined as follows:*

- $\mathbf{Real}^{\Sigma}_{\mathcal{A}}(\lambda, \Omega)$*: Adversary $\mathcal{A}$ chooses a security parameter $\lambda$, the experiment runs* $\mathsf{VDSSE.Setup}(\lambda, \perp; \perp)$ *and returns $\Sigma$ and $\mathrm{T}$ to the $\mathcal{A}$. Then, $\mathcal{A}$ adaptively performs the search, verify and update queries and receives the transcript generated by running* $\mathsf{VDSSE.Search}(k_s, w, \Sigma; \mathrm{T})$, $\mathsf{VDSSE.Verify}(k_r, w, \Sigma, \mathrm{R}, proof)$ *and* $\mathsf{VDSSE.Update}(k_s, k_r, \mathrm{ind}, w, op, \Sigma; \mathrm{T})$*. Finally, $\mathcal{A}$ outputs a bit $b$ based on the real transcripts of all operations.*

- $\mathbf{Ideal}^{\Sigma}_{\mathcal{A},\mathcal{S},\mathcal{L}}(\lambda, \Omega)$*: Adversary $\mathcal{A}$ chooses a security parameter $\lambda$ and obtains $\Sigma$ and $\mathrm{T}$ generated by the simulator $\mathcal{S}(\mathcal{L}^{\mathrm{Stp}}(\lambda))$. Then, $\mathcal{A}$ adaptively performs the search, verify and update operations and receives the transcript generated by running $\mathcal{S}(\mathcal{L}^{\mathrm{Srch}}(q))$, $\mathcal{S}(\mathcal{L}^{\mathrm{Verify}}(q))$ and $\mathcal{S}(\mathcal{L}^{\mathrm{Updt}}(q))$. Finally, $\mathcal{A}$ outputs a bit $b$ based on the all simulated transcripts of all operations.*

**Definition 4.** *(Forward Security). An $\mathcal{L}$-adaptively-secure searchable symmetric encryption scheme is forward secure if for an update query $q_i = (\mathrm{ind}_i, w_i, op_i)$, the leakage function $\mathcal{L}^{\mathrm{Updt}}(q) = (i, op_i, \mathrm{ind}_i)$.*

Forward security is a key requirement for the dynamic SSE, which can resist the devastating file injection attack. Generally speaking, forward security requires that the updated document cannot leak the information of the queried keyword. That is, one cannot use the previous search token to perform search in the new updated document.

**Definition 5.** *A verifiable database scheme* $\mathsf{VDB} = (\mathsf{Setup}, \mathsf{Query}, \mathsf{Verify}, \mathsf{Update})$ *consists of four algorithms defined below.*

- $\mathsf{Setup}(1^k, \mathsf{DB})$*: On input the security parameter* $k$*, the client generates the secret key* $\mathsf{SK}$ *and* $\mathsf{PK}$*. Then the client encrypts the database* $\mathsf{DB}$ *as* $\mathsf{S}$ *which is outsource to the server.*
- $\mathsf{Query}(\mathsf{PK}, \mathsf{S}, x)$*: On input the cell index* $x$*, the server returns* $\tau = (v_x, \pi)$*, where* $v_x$ *is the record value for the cell index* $x$ *and* $\pi$ *is the corresponding proof.*
- $\mathsf{Verify}(\mathsf{PK}/\mathsf{SK}, \tau, x)$*: The client use* $\tau = (v_x, \pi)$ *to verify the correct of the* $v_x$*. If* $v_x$ *is correct with respect to* $x$*, this algorithm outputs* $Accept$*. Otherwise, it outputs* $Reject$*.*
- $\mathsf{Update}(\mathsf{SK}, \mathsf{PK}, x, v')$*: The client uses the secret key* $\mathsf{SK}$ *to generate a token* $t'_x$*, and sends* $(t'_x, v')$ *to the server. The server updates the record to* $v'$ *in cell index* $x$*, and update the public key* $\mathsf{PK}$ *based on* $t'_x$*.*

## 3. New Efficient VDB Scheme Supporting Forward-secure Keyword Searches

### *3.1. High Description*

Miao et al. [21] firstly addressed the challenge of keyword-based search on VDB schemes. However, VDB schemes used in this framework only support the replacement operations, rather than other updating operations such as insertion and deletion. Moreover, if we consider to insert or delete a data record in VDB schemes supporting keyword searches, then forward/backward-secure SSE should be adopted in order to avoid the file injection attack. Fortunately, it is sufficient that we only need consider forward-secure SSE in VDB schemes. We explain the details as follows:

Note that in all exiting VDB schemes that support full updating operations (i.e., replacement, insertion, and deletion) [11, 19, 20], the operations of insertion and deletion could be viewed as a particular kind of replacement. More precisely, we first observe the deletion operation in VDB schemes. Note that deleting a record $v$ is equal

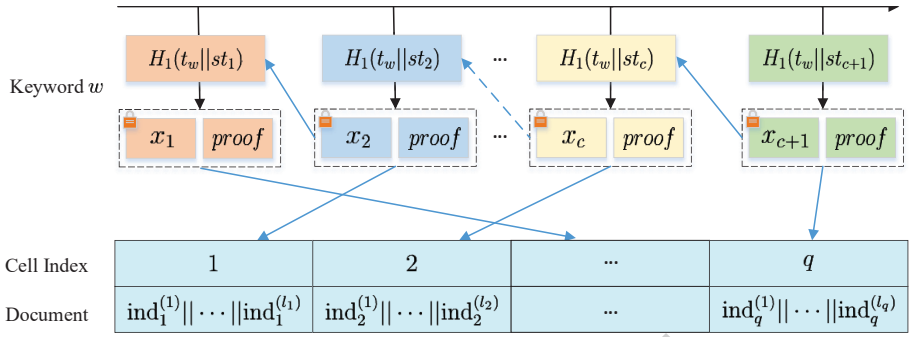


Figure 1: Overview of our scheme

to replace $v$ with a special empty value $\emptyset$, thus the total number of data records in the database (which equals to number of query index) remains unchanged. Trivially, this is different from the file deletion in SSE schemes. Therefore, we need not consider backward-secure SSE in VDB schemes supporting keyword searches. While for the case of insertion, the situation is totally different. After inserting a new data record, the total number of data records in the database is also increase by 1. As a result, forward-secure SSE schemes have to be adopted to resist the file injection attack.

To achieve forward-secure keyword search in VDB scenario, a trivial solution is to simply combine traditional forward-secure SSE with VDB. That is, a distinct keyword $w$ is assigned to a specific position $i$ (i.e., $i$-th cell of VDB) and all the documents containing $w$ are stored into the corresponding position. Note that all the documents containing $w$ are stored together. So the server can determine whether two update operations are acted on the identical document by observing the updated position of VDB. It is clear that the above method cannot achieve forward secrecy.

The key insight is that each keyword is binding to a specific position in the above-mentioned solution. It will leak the relations of newly updated data and the previously searched keyword. To deal with this, as shown in Fig. 1, we can replace the document identifier of SSE with the index $x$, and then the keyword-index $(w, x)$ pair is used to generate the searchable index in SSE, where the documents containing $w$ are spread over different positions. Note that the document update operation in VDB is only

performed when the first keyword-index $(w, x)$ pair for a given document is inserted. That means that the subsequent keyword-index pairs insertion will have no effect on VDB. What's more, each keyword-index pair for a given keyword will act on different positions in VDB. It makes the server cannot learn the relationship between the newly updated data and the previously queried keyword. Thus forward privacy is achieved.

### *3.2. A Concrete VDB Scheme with Keyword Search*

In this section, we present a novel VDB construction supporting forward secure keyword-based search. We need to mention that if the client performs a index-based search in the VDB structure, the Merkel sum hash tree is necessary to locate the index. While if the client performs a keyword-based search in the VDB structure, we use the search token to locate the index, rather than the Merkel sum hash tree. In this section, we mainly focus on how to perform keyword-based search in the VDB structure.

- $\mathbf{Setup}(1^n, 1^\lambda, \mathsf{DB})$: Given an encrypted NoSQL database DB with the form of $(x, \mathrm{ind}_x) \in [q] \times \mathbb{Z}_{n-1}$, there are two steps in this algorithm. The first step is to setup the VDB structure, and the second one is to setup the index for forward secure keyword search.

  1. This step is to generate the public key $\mathsf{PK}$ and the secrete key $\mathsf{SK}$ of the VDB structure. Given a security parameter $n$, gen erate two groups $\mathbb{G}$, $\mathbb{G}_T$ of order $N = p_1 p_2$, where $p_1, p_2$ are primes in the range $[2^{n-1}, 2^n - 1]$, and a bilinear pairing $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$. Let $F$ be an algebraic pseudorandom function (PRF) and randomly choose two keys $k_1$, $k_2$ for it, $\mathcal{H}$ be a hash function, $\{0,1\}^* \to \mathbb{Z}_{n-1}$. $\mathbb{G}_1$ and $\mathbb{G}_2$ is the subgroups of $\mathbb{G}$ of order $p_1$ and $p_2$, respectively. Generate $g_1, h_1 \in_R \mathbb{G}_1$, $g_2, h_2, u_2 \in_R \mathbb{G}_2$, and $a, b \in_R \mathbb{Z}_N$. For each $i \in \{1, ..., q\}$, set $r_i = F_{k_1}(i), \omega_i = F_{k_2}(i, 1)$, and $s_i = 1$. Define

$$t_i = g_1^{r_i + a\mathcal{H}(\mathrm{ind}_i) + bs_i} g_2^{\omega_i}, \omega = \sum_{i=1}^{q} \omega_i, T_\omega \leftarrow e(g_2, u_2)^\omega,$$

$$\hat{t_0} = u_2, \hat{t_1} = h_1 h_2.$$

Finally, the public key is $\mathsf{PK} = \big((\hat{t_0}, \hat{t_1}, s_1, t_1, ..., s_q, t_q), \mathsf{DB}\big)$, and the private key is $\mathsf{SK} = (a, T_\omega, k_1, k_2)$.

2. This step is to generate the search index for keyword search. Given the security parameter $\lambda$, it firstly runs $\mathrm{FSSE.Setup}(\lambda, \perp; \perp)$ to initiate the search index, and then generates the search index for the database $\mathsf{DB}$ as follows.
   - For each document $\mathrm{ind}_x$ in the $\mathsf{DB}$
     - For each $w$ in the $\mathsf{W}_x$
       - $\mathrm{FSSE.Update}(k_s, k_r, x, w, \Sigma; \mathrm{T})$
     - End For
   - End For

- $\mathbf{Query}(k_s, w, \Sigma, \mathrm{T}, \mathsf{PK}, \mathsf{DB})$: Suppose the client would like to search all the documents that contain a given keyword $w$. Firstly, the client generates the search token $(t_w, st_c)$ and sends it to the cloud server. Then the cloud server performs search in the search index T, and gets the search result R and the corresponding proof. Secondly, the cloud server returns the real documents in the VDB based on the search result R. The details are as follows:

  1. The client and the server perform the algorithm $\mathrm{FSSE.Search}(k_s, w, \Sigma; \mathrm{T})$. In this algorithm, the client uses $k_s, w, \Sigma$ to generate the search token $(t_w, st_c)$ and send it to the server. The server performs search in T and generates $(\mathrm{R}, proof)$, where R is the search result and the $proof$ can be used to verify the correctness and completeness of R.
  2. For each search result $x \in \mathrm{R}$, the server retrieves the real document and generates the proof as follows. With the input of the $x$ and public key $\mathsf{PK}$, the server computes

$$T = e(t_x, \hat{t_1}) \cdot e\Big(\prod_{i=1, i\neq x}^{q} t_i, \hat{t_0}\Big).$$

  Without loss of generality, we define $\boldsymbol{v}_x = \mathrm{ind}_x^{(1)} || \mathrm{ind}_x^{(2)} || \cdots || \mathrm{ind}_x^{(l_x)}$. Finally, output $\tau = (T, x, \boldsymbol{v}_x)$.

- **Verify**$(k_r, w, \Sigma, \mathrm{R}, proof, \mathsf{SK}, x, \tau)$: There are two steps to verify the search result. The first one is to use the $(k_r, w, \Sigma, \mathrm{R}, proof)$ to verify the correctness and the completeness of the search result R. The second one is to use $(\mathsf{PK}, x, \tau)$ to verify the returned documents $v_x$ matches with the $x$.

  1. The client performs FSSE.Verify$(k_r, w, \Sigma, R, proof)$. If the output is $Accept$, the client would perform the second step. Otherwise, the client would finish this **Verify** algorithm, and output $Reject$.
  2. The client takes as input the private key $\mathsf{SK}$, the query index $x$, and $\tau = (T, x, \boldsymbol{v}_x)$. The client checks

  $$T \stackrel{?}{=} T_\omega \cdot e(g_1^{r_x + a \cdot \mathcal{H}(\boldsymbol{v}_x) + b \cdot s_x} g_2, h_1 u_2^{-\omega_x} h_2^{\omega_x}),$$

  where $r_x = F_{k_1}(x)$, $\omega_x = F_{k_2}(x, s_x)$. If the equality holds, the client outputs $Accept$. Otherwise, output $Reject$.

- **Update**$(\mathsf{SK}, \mathrm{ind}_x^*)$: There are three types of update, i.e., insertion, replacement and deletion.

  – **Insertion**: Assume that the client would like to insert a new document $\mathrm{ind}_x^*$, one should update the search index firstly and then the VDB structure.

  1. For each $w \in \mathrm{W}_{\mathrm{ind}_x^*}$, the client and the server perform the following protocol FSSE.Update$(k_s, k_r, x, w, add, \Sigma; \mathrm{T})$.
  2. The client firstly retrieves $\tau$ from the VDB structure in position $x$ from the server side. If the output of the verification of $\tau$ is $Accept$, the client computes $\boldsymbol{v}'_x$ by inserting $\mathrm{ind}_x^*$ in $\boldsymbol{v}_x$. Specifically, if $\boldsymbol{v}_x = \mathrm{ind}_x^{(1)} || \mathrm{ind}_x^{(2)} || \cdots || \mathrm{ind}_x^{(l_x)}$, then $\boldsymbol{v}'_x = \mathrm{ind}_x^{(1)} || \mathrm{ind}_x^{(2)} || \cdots || \mathrm{ind}_x^{(l_x)} || \mathrm{ind}_x^*$. Then, set

  $$\omega'_x = F_{k_2}(x, s_x + 1) - F_{k_2}(x, s_x), T_\omega \leftarrow T_\omega \cdot e(g_2^{\omega'_x}, u_2),$$

  $$t'_x \leftarrow g_1^{a(\mathcal{H}(\boldsymbol{v}'_x) - \mathcal{H}(\boldsymbol{v}_x)) + b} g_2^{\omega'_x}.$$

  The server is then given $t'_x$, and updates the public key by setting $t_x \leftarrow t_x \cdot t'_x$, $s_x \leftarrow s_x + 1$ and $l_x \leftarrow l_x + 1$.

– **Replacement**: Assume that the client will replace a document $\mathrm{ind}_x$ with $\mathrm{ind}_x^*$, there are also two steps to update the search index and VDB structure.

1. If there are some new keyword $w$ to be update, the client and the sever perform the following protocol $\mathrm{FSSE.Update}(k_s, k_r, x, w, add, \Sigma; \mathrm{T})$.
2. This step is to update the VDB structure. The client firstly retrieves $\tau$ from the VDB structure in position $x$ from the server side. If the output of the verification of $\tau$ is $Accept$, the client computes as follows. The client computes $\boldsymbol{v}'_x$ by replacing $\mathrm{ind}_x$ with $\mathrm{ind}_x^*$. Specifically, if $\boldsymbol{v}_x = \mathrm{ind}_x^{(1)}||\cdots||\mathrm{ind}_x||\cdots||v_x^{(l_x)}$, then $\boldsymbol{v}'_x = \mathrm{ind}_x^{(1)}||\cdots||\mathrm{ind}_x^*||\cdots||\mathrm{ind}_x^{(l_x)}$. Set

$$\omega'_x = F_{k_2}(x, s_x + 1) - F_{k_2}(x, s_x), T_\omega \leftarrow T_\omega \cdot e(g_2^{\omega'_x}, u_2),$$

$$t'_x \leftarrow g_1^{a(\mathcal{H}(\boldsymbol{v}'_x) - \mathcal{H}(\boldsymbol{v}_x)) + b} g_2^{\omega'_x}.$$

With $t'_x$, The server updates the public key by setting $t_x \leftarrow t_x \cdot t'_x$ and $s_x \leftarrow s_x + 1$.

– **Deletion**: Assume that the client will delete a document $\mathrm{ind}_x$, the operation can be review as a special case of replacement. That is, the client sets $\mathrm{ind}_x = 0$. The only difference is that we do not need to update the search index in the server side.

## 4. Security and Efficiency Analysis

In this section, we present the security and efficiency analysis.

### *4.1. Security Analysis*

**Theorem 4.1.** *The proposed VDB scheme supporting keyword search is correct.*

**Proof 1.** *There are two steps in the proposed scheme to return the searched documents. In the first step, the server searches in the search index based on the search token and finds out the matched cell indexes. Then the server uses the cell indexes to retrieve the corresponding documents. For the first step, the client uses $proof$ to verify the*

*correctness and completeness of the matched cell indexes. The correct of this step comes from the property of Multiset Hash Functions in [36]. Specifically, the client computes the hash value by the returned cell indexes, and compare the computed hash value with the one keeps on the client side. Since the hash value on the client side is the latest one, the* $proof$ *would pass the verification only when the returned cell indexes are correct and complete. This guarantee that the cell indexes are the whole search result. In the second step, the server retrieve the documents based on the cell indexes and returned the corresponding proof* $\tau = (T, x, \upsilon_x)$*. If the documents is the desired ones, the proof will pass the verification. That is,*

$$\begin{aligned} T &= e(t_x, \hat{t_1}) e(\prod_{i \neq x} t_i, \hat{t_0}) \\ &= e(g_1^{r_x + a\mathcal{H}(\boldsymbol{v}_x) + bs_x} g_2^{w_x}, h_1 h_2) \prod_{i \neq x} e(g_1^{r_i + a\mathcal{H}(\boldsymbol{v}_i) + bs_i} g_2^{\omega_i}, u_2) \\ &= e(g_1^{r_x + a\mathcal{H}(\boldsymbol{v}_x) + bs_x} g_2, h_1 u_2^{-\omega_x} h_2^{\omega_x}) \cdot T_\omega \end{aligned}$$

**Theorem 4.2.** *The proposed VDB scheme supporting keyword search is secure.*

**Proof 2.** *In our scheme, if a client performs search based on a cell index, the VDB construction is the same in [19]. Thus, please refer to [19] for the details of the security of the scheme. While if a client performs keyword search in the VDB, the client should perform keyword search in the search index first, and then performs search based on cell index in the VDB structure. In this setting, we need to analyze the security of the keyword search in the search index, and then the VDB structure. We give the details as follows.*

*Similar to [36], the keyword search index is* $\mathcal{L}$*-adaptively-secure. The leakage function* $\mathcal{L}$ *in our scheme includes search pattern* $\mathbf{sp}(w)$*, update history* $\mathbf{uh}(w)$ *and proof history* $\mathbf{ph}(w)$*. Specifically, search pattern is defined as* $\mathbf{sp}(w) = \{i \mid$ *for each*$(i, w)$ *in* $Q\}$*, and the* $\mathbf{uh}(w) = \{(i, op_i, ind_i) \mid$ *for each*$(i, ind_i, w, op_i)$ *in* $\mathrm{Hist}\}$*, and* $\mathbf{ph}(w) = \{(i, proof_i) \mid$ *for each* $(i, ind_i, w_i, op_i)$ *in* $\mathrm{Hist}\}$*. Here the query history* $\mathrm{Hist} = (\mathrm{DB_i}, q_i)_{i=0}^{Q}$ *which contains the query* $q_i$ *and the corresponding response* $\mathrm{DB}_i$*, where* $q_i = (i, w)$ *for a search query, or* $q_i = (i, ind, w, op)$ *for a update query, or* $q_i = (i, w, \mathrm{R}, proof)$ *for a verify query. The idea to prove the security of search index is*

*to use a sequence of indistinguishable hybrids. The first hybrid is the real game. In the second hybrid, we replace the PRFs by tables respectively. In the third hybrid, we replace the hash functions by random oracles and then program them during the search algorithm. The last hybrid is the ideal-world game. Because the latter hybrid is indistinguishable from the prior one, the ideal-world game is distinguishable from the real-world game. Therefore, the the keyword search index is $\mathcal{L}$-adaptively-secure. The more details can be referred to [36]. For the security of the VDB structure, we can prove that it satisfies the property of security under the assumption of subgroup membership in composite order bilinear groups [19]. Note that for the keyword search in the VDB structure, the Merkel sum hash tree is unnecessary. For the above analysis, our prosed VDB scheme supporting keyword search is secure.*

**Theorem 4.3.** *The proposed VDB scheme supporting keyword search is efficient.*

**Proof 3.** *Our proposed scheme can support keyword search by constructing the keyword search index. Note that it is very efficient to perform the keyword-based algorithms, because we only use the symmetric cryptographic primitives. Besides, the communication overhead is also acceptable which is benefit from the incremental property of the multiset hash functions. Specifically, the user uses the update data and the latest owned $proof$ to incrementally update the $proof$. It is unnecessary to download all the previous data on the server side to generate the $proof$. In this case, the computation and communication overhead in the proposed VDB scheme supporting keyword search is acceptable.*

### *4.2. Comparison*

In this section, we give the comparison of computational cost between our proposed scheme with Miao et al.'s VDB scheme. In our proposed scheme, the VDB structure can support keyword search, while allowing to update the documents with forward security. We remark that our scheme can support keyword search by introducing some slightly computation overload. That is, the SSE structure is very efficient to construct, which is based on the symmetric cryptographic primitives. In the setup phase, it is

necessary for our scheme to construct the keyword search index based on the pseudo-random functions and hash functions. In the verify algorithm, we firstly verify the correctness and completeness of the search result based on the search index, and then verify that the returned document related to the search result is not modified based on the VDB structure. We would like to mention that the extra verification of the search result is very efficient, which only needs some hash functions.

Table 1: Efficiency Comparison

| Schemes | Scheme [19] | Our Scheme |
|---|---|---|
| Model | Amortized | Amortized |
| Assumption | Subgroup Member | Subgroup Member |
| Accountability | $\surd$ | $\surd$ |
| Keyword Search | $\times$ | $\surd$ |
| Forward Security | - | $\surd$ |
| Setup Cost | $2qF + 2M_1 + 3E + P$ | $(2q+d)F + 2M_1 + 3E + P$ $+(2F + 5H)\|\mathrm{DB}\|$ |
| Query Cost (Server) | $(q-1)M_1 + 2P$ | $((q-1)M_1 + 2P + 2H)\|\mathrm{DB}(w)\|$ |
| Update Cost (Client) | $1M_1 + 1M_2 + 4E + 2F + 1P$ | $1M_1 + 1M_2 + 4E + 2F + 1P$ $+(2F + 5H)\|\mathrm{W}\|$ |
| Update Cost (Server) | $1M_1 + 1E$ | $1M_1 + 1E$ |
| Verify Cost (Client) | $3M_1 + 1M_2 + 3E + 2F$ $+1P + \log_2 q \cdot H$ | $3M_1 + 1M_2 + 3E + 3F$ $+1P + (2 + H\|R\|)$ |

We present the comparison on the computation cost between Miao et al.'s scheme and our proposed scheme in Table 1. We mainly focus on the computation overhead in data setup, update and verify phase. In the following, we introduce some notations. We denote by $M_1$ and $M_2$ the multiplication operation in group $\mathbb{G}_1$ and $\mathbb{G}_2$, $E$ a modular exponentiation in $\mathbb{G}_1$, $P$ a pairing operation, $H$ an operation of hash, and $F$ a pseudo-random function. Besides, we refer to $|\mathrm{DB}|$ as the size of database, $|\mathrm{DB}(w)|$ as the number of documents containing $w$ and $|\mathrm{W}|$ as the number of keywords in the database.

## 5. Conclusion

The notion of VDB provides an elegant solution to the challenge of verifiable auditing on a dynamic huge database. Due to the file injection attack, traditional SSE cannot be adopted to VDB schemes supporting insertion/deletion operations. In this paper, we first consider the problem how to incorporate the two primitives of forward-secure SSE and VDB with full updating operations. Besides, we propose a concrete VDB scheme that also supports forward-secure keyword searches. The security and efficiency analysis show that the proposed VDB scheme can achieve the desired security properties and is efficient for real-world applications.

## Acknowledgement

This work is supported by National Natural Science Foundation of China (No. 61902315).

## References

[1] Agrawal, D., El Abbadi, A., Emekçi, F., Metwally, A., 2009. Database management as a service: Challenges and opportunities, in: Proceedings of the 25th International Conference on Data Engineering, ICDE 2009, IEEE. pp. 1709–1716.

[2] Applebaum, B., Ishai, Y., Kushilevitz, E., 2010. From secrecy to soundness: Efficient verification via secure computation, in: Proceedings of the 37th International Colloquium on Automata, Languages and Programming, ICALP 2010, Springer. pp. 152–163.

[3] Ateniese, G., Burns, R.C., Curtmola, R., Herring, J., Kissner, L., Peterson, Z.N.J., Song, D.X., 2007. Provable data possession at untrusted stores, in: Proceedings of the 14th ACM Conference on Computer and Communications Security, CCS 2007, ACM. pp. 598–609.

[4] Benabbas, S., Gennaro, R., Vahlis, Y., 2011. Verifiable delegation of computation over large datasets, in: Proceedings of the Advances in Cryptology, CRYPTO 2011, Springer. pp. 111–131.

[5] Boneh, D., Goh, E., Nissim, K., 2005. Evaluating 2-dnf formulas on ciphertexts, in: Proceedings of the 2nd Theory of Cryptography Conference, TCC 2005, Springer. pp. 325–341.

[6] Bost, R., 2016. $\sum o\varphi o\varsigma$: Forward secure searchable encryption, in: Proceedings of the 23rd ACM SIGSAC Conference on Computer and Communications Security, CCS 2016, ACM. pp. 1143–1154.

[7] Bost, R., Minaud, B., Ohrimenko, O., 2017. Forward and backward private searchable encryption from constrained cryptographic primitives, in: Proceedings of the 24th ACM Conference on Computer and Communications Security, CCS 2017, ACM. pp. 1465–1482.

[8] Camenisch, J., Kohlweiss, M., Soriente, C., 2009. An accumulator based on bilinear maps and efficient revocation for anonymous credentials, in: Proceedings of the 12th International Conference on Practice and Theory in Public Key Cryptography, PKC 2009, Springer. pp. 481–500.

[9] Camenisch, J., Lysyanskaya, A., 2002. Dynamic accumulators and application to efficient revocation of anonymous credentials, in: Proceedings of the Advances in Cryptology, CRYPTO 2002, Springer. pp. 61–76.

[10] Catalano, D., Fiore, D., 2013. Vector commitments and their applications, in: Proceedings of the 16th International Conference on Practice and Theory in Public-Key Cryptography, PKC 2013, Springer. pp. 55–72.

[11] Chen, X., Li, H., Li, J., Wang, Q., Huang, X., Susilo, W., Xiang, Y., . Publicly verifiable databases with all efficient updating operations. IEEE Transactions on Knowledge and Data Engineering, doi: 10.1109/TKDE.2020.2975777 .

[12] Chen, X., Li, J., Huang, X., Li, J., Xiang, Y., Wong, D.S., 2014. Secure outsourced attribute-based signatures. IEEE Trans. Parallel Distrib. Syst. 25, 3285–3294.

[13] Chen, X., Li, J., Huang, X., Ma, J., Lou, W., 2015. New publicly verifiable databases with efficient updates. IEEE Trans. Dependable Secur. Comput. 12, 546–556.

[14] Chen, X., Li, J., Weng, J., Ma, J., Lou, W., 2016. Verifiable computation over large database with incremental updates. IEEE Trans. Computers 65, 3184–3195.

[15] Demertzis, I., Chamani, J.G., Papadopoulos, D., Papamanthou, C., 2020. Dynamic searchable encryption with small client storage, in: Proceedings of the 27st Annual Network and Distributed System Security Symposium, NDSS 2020, The Internet Society.

[16] Gennaro, R., Gentry, C., Parno, B., 2010. Non-interactive verifiable computing: Outsourcing computation to untrusted workers, in: Proceedings of the Advances in Cryptology, CRYPTO 2010, Springer. pp. 465–482.

[17] Huang, H., Chen, X., Wang, J., 2020. Blockchain-based multiple groups data sharing with anonymity and traceability. Sci. China Inf. Sci. 63.

[18] Juels, A., Jr., B.S.K., 2007. Pors: proofs of retrievability for large files, in: Proceedings of the 14th ACM Conference on Computer and Communications Security, CCS 2007, ACM. pp. 584–597.

[19] Miao, M., Ma, J., Huang, X., Wang, Q., 2018. Efficient verifiable databases with insertion/deletion operations from delegating polynomial functions. IEEE Trans. Information Forensics and Security 13, 511–520.

[20] Miao, M., Wang, J., Ma, J., Susilo, W., 2017. Publicly verifiable databases with efficient insertion/deletion operations. J. Comput. Syst. Sci. 86, 49–58.

[21] Miao, M., Wang, J., Wen, S., Ma, J., 2019. Publicly verifiable database scheme with efficient keyword search. Inf. Sci. 475, 18–28.

[22] Papamanthou, C., Tamassia, R., 2007. Time and space efficient algorithms for two-party authenticated data structures, in: Proceedings of the 9th International

Conference on Information and Communications Security, ICICS 2007, Springer. pp. 1–15.

[23] Papamanthou, C., Tamassia, R., Triandopoulos, N., 2008. Authenticated hash tables, in: Proceedings of the 15th ACM Conference on Computer and Communications Security, CCS 2008, ACM. pp. 437–448.

[24] Papamanthou, C., Tamassia, R., Triandopoulos, N., 2016. Authenticated hash tables based on cryptographic accumulators. Algorithmica 74, 664–712.

[25] Song, X., Dong, C., Yuan, D., Xu, Q., Zhao, M., 2018. Forward private searchable symmetric encryption with optimized i/o efficiency. IEEE Transactions on Dependable and Secure Computing .

[26] Stefanov, E., Papamanthou, C., Shi, E., 2014. Practical dynamic searchable encryption with small leakage, in: Proceedings of the 21st Annual Network and Distributed System Security Symposium, NDSS 2014, The Internet Society. pp. 72–75.

[27] Sun, S., Liu, J.K., Sakzad, A., Steinfeld, R., Yuen, T.H., 2016. 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 2016, Springer. pp. 154–172.

[28] Sun, S., Yuan, X., Liu, J.K., Steinfeld, R., Sakzad, A., Vo, V., Nepal, S., 2018. Practical backward-secure searchable encryption from symmetric puncturable encryption, in: Proceedings of the 25th ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Springer. pp. 763–780.

[29] Tamassia, R., 2003. Authenticated data structures, in: Proceedings of the 11th Annual European Symposium on Algorithms, ESA 2003, Springer. pp. 2–5.

[30] Wang, J., Chen, X., Li, J., Zhao, J., Shen, J., 2017. Towards achieving flexible and verifiable search for outsourced database in cloud computing. Future Generation Comp. Syst. 67, 266–275.

[31] Wang, J., Chen, X., Sun, S., Liu, J.K., Au, M.H., Zhan, Z., 2018. Towards efficient verifiable conjunctive keyword search for large encrypted database, in: Proceedings of the 23rd European Symposium on Research in Computer Security, ESORICS 2018, Springer. pp. 83–100.

[32] Wang, Y., Sun, S.F., Wang, J., Liu, J.K., Chen, X., 2020. Achieving searchable encryption scheme with search pattern hidden. IEEE Transactions on Services Computing doi:10.1109/TSC.2020.2973139.

[33] Wu, Z., Li, K., 2019. Vbtree: forward secure conjunctive queries over encrypted data for cloud computing. VLDB J. 28, 25–46.

[34] Yuan, H., Chen, X., Li, J., Jiang, T., Wang, J., Deng, R., 2019. Secure cloud data deduplication with efficient re-encryption. IEEE Transactions on Services Computing doi:10.1109/TSC.2019.2948007.

[35] Zhang, Y., Katz, J., Papamanthou, C., 2016. All your queries are belong to us: The power of file-injection attacks on searchable encryption, in: Proceedings of the 25th USENIX Security Symposium, USENIX Security 2016, USENIX Association. pp. 707–720.

[36] Zhang, Z., Wang, J., Wang, Y., Su, Y., Chen, X., 2019. Towards efficient verifiable forward secure searchable symmetric encryption, in: Proceedings of the 24th European Symposium on Research in Computer Security, ESORICS 2019, Springer. pp. 304–321.

[37] Zuo, C., Sun, S., Liu, J.K., Shao, J., Pieprzyk, J., 2018. Dynamic searchable symmetric encryption schemes supporting range queries with forward (and backward) security, in: Proceedings of the 23rd European Symposium on Research in Computer Security, ESORICS 2018, Springer. pp. 228–246.

**Conflict of Interest**

We declare that we do not have any commercial or associative interest that represents a conflict of interest in connection with the work submitted, the manuscript entitled "Verifiable Database Supporting Keyword Searches with Forward Security".