Enabling Verifiable and Dynamic Ranked Search over Outsourced Data

Qin Liu, *Member, IEEE*, Yue Tian, *Student Member, IEEE*, Jie Wu, *Fellow, IEEE*, Tao Peng, *Member, IEEE*, and Guojun Wang, *Member, IEEE*

Abstract—Cloud computing as a promising computing paradigm is increasingly utilized as potential hosts for users' massive dataset. Since the cloud service provider (CSP) is outside the users' trusted domain, existing research suggests encrypting sensitive data before outsourcing and adopting Searchable Symmetric Encryption (SSE) to facilitate keyword-based searches over the ciphertexts. However, it remains a challenging task to design an effective SSE scheme that simultaneously supports *sublinear search time*, *efficient update and verification*, and *on-demand information retrieval*. To address this, we propose a Verifiable Dynamic Encryption with Ranked Search (VDERS) scheme that allows a user to perform top-K searches on a *dynamic* document collection and verify the correctness of the search results in a secure and efficient way. Specifically, we first provide a basic construction, VDERS^0, where a *ranked inverted index* and a *verifiable matrix* are constructed to enable verifiable document insertion in top-K searches. Then, an advanced construction, VDERS*, is devised to further support document deletion with a reduced communication cost. Extensive experiments on real datasets demonstrate the efficiency and effectiveness of our VDERS scheme.

Index Terms—Searchable symmetric encryption, verifiability, dynamic, top-K searches

1 Introduction

Because of the benefits of low costs and scalability, it has become a prevalent trend for users to outsource their massive datasets to clouds and delegate a cloud service provider (CSP) to manage data storage and offer query services [1]. With the wide application of cloud computing, frequent data leakages have been noticed recently, ranging from personal medical records to private data on social networks [2], [3], [4], [5], [6]. Existing research suggests encrypting data before outsourcing [7]. However, data encryption makes keyword-based searches over ciphertexts a challenging problem. This is even harder for efficient top-K searches in a dynamic and malicious cloud environment [8].

Let us consider the following scenario. Alice outsources archived emails to the cloud, where each email is indexed by the sender's name and ranked in descending order of the receipt date. For example, for a set of emails indexed by keyword *Bob*, the email received on April 2 has a higher rank than the email received on April 1. To keep keyword and document contents secret, Alice uploads them in the encrypted forms to the cloud. There could be hundreds of documents matching a specific keyword, and the consumed costs will be extensive if all the matched documents are returned to and decrypted by the user. Therefore, Alice may want to perform a top-K search to retrieve the most recent emails. Moreover, Alice may want to store only the emails received in the last three months for monetary saving. For example, when entering May, Alice will delete all emails received before February.

In the above application scenario, the adopted encryption scheme should meet the following requirements: (1) *Ranked search*. The user is allowed to perform a top-K search to retrieve the best-matched documents. (2) *Dynamic*. The user is able to update (add and delete) documents stored in the cloud [9]. (3) *Verifiability*. The malicious CSP may delete encrypted documents not commonly used to save memory space, or it may forge the search results to deceive the user. Even if the CSP is *honest*, a virus or worm may tamper with encrypted documents. Therefore, the user should have the ability to verify the correctness of the search results. (4) *Efficiency*. The user can efficiently perform searches, updates, and verifications on a set of encrypted documents.

Although Searchable Symmetric Encryption (SSE) allows a user to retrieve desired documents in a privacy-preserving way, existing SSE schemes only partially address the above requirements. To simultaneously satisfy all these properties, this paper proposes a Verifiable Dynamic Encryption with Ranked Search (VDERS) scheme that allows the user to perform updates and top-K searches on ciphertexts in a verifiable and efficient way. Our main idea is to construct a *verifiable matrix* to record the ranking information and encode it with RSA accumulator [10]. Furthermore, a *ranked inverted index* is built from a collection of documents to facilitate efficient top-K searches and updates. Specifically, we

- Q. Liu and Y. Tian are with the College of Computer Science and Electronic Engineering, Hunan University, Changsha, Hunan Province 410082, P. R. China. E-mail: {gracelq628, tiany}@hnu.edu.cn.
- J. Wu is with the Department of Computer and Information Sciences, Temple University, Philadelphia, PA 19122 USA. E-mail: jiewu@temple.edu.
- T. Peng and G. Wang are with the School of Computer Science and Cyber Engineering, Guangzhou University, Guangzhou, Guangdong Province 510006, P. R. China. E-mail: pengtao_work@163.com, csgjwang@gzhu.edu.cn.

Manuscript received 25 Sept. 2018; revised 30 Apr. 2019; accepted 3 June 2019. Date of publication 11 June 2019; date of current version 4 Feb. 2022. (Corresponding author: Qin Liu.)
Digital Object Identifier no. 10.1109/TSC.2019.2922177

first provide a basic construction, denoted by VDERS^0, which enables verifiable document insertion operations. Then, we provide an advanced construction, denoted by VDERS^*, which not only can support efficient deletion operations, but also can reduce communication costs without outsourcing the verifiable matrix. Our main contributions are summarized as follows:

- We propose a VDERS scheme to achieve dynamic and ranked searches in a cloud environment in an efficient and verifiable way.
- Two constructions are provided to achieve efficient top-K searches with support for verifiable updates.
- We theoretically analyze the security and performance of our scheme and conduct extensive experiments on real datasets to validate its effectiveness.

Paper Organization. We introduce related work in Section 2 and provide the preliminaries in Section 3. After the overview of this work in Section 4, we provide our basic and advanced VDERS constructions in Sections 5 and 6, respectively. We evaluate the proposed scheme in Section 7. Finally, we conclude the paper in Section 8.

2 Related Work

Our work focuses on verifiable ranked queries on dynamic encrypted data. SSE that allows an untrusted server to perform keyword-based searches on ciphertexts can partially address our requirements.

SSE has been widely researched since it was first proposed by Song et al. [11]. As a seminal work in SSE, Curtmola et al. [12] provided a rigorous security definition and constructed two schemes, SSE-1 and SSE-2, based on an inverted index. Compared to SSE-1, SSE-2 is more secure, and it has been proven to be secure against adaptive chosen keyword attacks (CKA2). SSE-1 is secure against non-adaptive chosen-keyword attacks (CKA1), but yields an optimal (sublinear) search time $O(r)$, where r is the number of documents that contain the query keyword. However, neither SSE-1 nor SSE-2 has properties of *dynamism*, *verifiability*, and *ranked search*.

Dynamic SSE (DSSE) allows a user to update the encrypted outsourced data in an efficient and secure way. Kamara et al. [13] constructed a DSSE scheme based on an extended inverted index. Their subsequent work [14] extended it to a parallel search setting by using a red-black tree. Cash et al. [15] constructed a DSSE scheme optimized for super-large datasets, but their scheme supports only efficient document insertion. Naveed et al. [16] put forward a DSSE scheme in which a server worked as a blind storage to decrease leakage at the cost of multiple rounds of interaction. The above schemes are proven to be CKA2-secure, but leak keyword information about the newly added documents. With this leakage, the attackers can reveal the content of a past query by injecting new documents in a dataset [17]. To mitigate such an attack, Stefanov et al. [18] proposed the first DSSE scheme with *forward privacy*, but their scheme suffers from inefficiency. Since then, forward privacy has been the main motivation of recent DSSE schemes [19], [20].

Verifiable SSE (VSSE) allows a user to verify the correctness of search results. Kurosawa et al. [21] constructed a Universally Composable (UC)-secure VSSE scheme, in which a user can detect any malicious server's cheating behavior. While UC-security is stronger than CKA2-security, their construction requires a linear search time. Soleimanian et al. [22] presented a public VSSE scheme, which delegates a third party to accomplish the verification, but fails to support dynamic operations. The subsequent work of Kurosawa et al. [23] extends VSSE to a dynamic environment. Their scheme employs RSA accumulator [10] to generate constant-size digests/proofs, but the verification cost on the client side grows linearly with the total number of documents. To enable verifiable conjunctive keyword search over dynamic encrypted data, Sun et al. [24] exploited the bilinear-map accumulator technique to construct an accumulation tree. Jiang et al. [25] proposed a VDSSE scheme that also utilized an accumulator tree to verify results of boolean queries. The accumulator tree structure is more efficient than RSA accumulator in verification, but consumes more computation time for updating tree structure. Zhu et al. [26] proposed a generic VSSE scheme in a multi-user setting, where the verifiable design can provide result verification for any SSE schemes and support data updates. *However, the above VSSE schemes return all search results and therefore, may be unsuitable for an environment where a lot of documents match a user's query but the user is only interested in best-match documents.*

Ranked SSE (RSSE) allows a user to rank the search results based on different evaluation functions. Cao et al. [27] proposed a dynamic multi-keyword RSSE scheme which utilized the secure KNN technique [28] to rank results according to the number of matched query keywords. The main limitation of their scheme is inefficiency, where the search time increases linearly with the number of documents. Inspired by their work, many multi-keyword VRSSE schemes [29], [30], [31], [32], [33] were put forward. Sun et al. [29] proposed a verifiable RSSE scheme which ranked search results based on cosine similarity while improving search efficiency by an MDB tree structure. Chen et al. [30] proposed a RSSE scheme based on a hierarchical clustering index. Their scheme supports update and verification, but requires the client to re-encrypt search indexes once a document is added into a dataset. Zhang et al. [31] developed a RSSE scheme in a multi-owner model, where the search time grew linearly with the number of queried keywords. Fu et al. [32] leveraged a user interest model to assign weight to query keywords according to the user's search history. Guo et al. [33] put forward a dynamic multi-phrase SSE scheme that allowed the user to rank results locally based on TF-IDF scores. *However, it is hard for existing RSSE schemes to simultaneously support sublinear search time and efficient updates and verification.* The comparison results between our work and previous SSE schemes are shown in Table 1.

3 Preliminary

3.1 System Model

The system consists of three different parties: the CSP, the data owner, and the data user, as illustrated in Fig. 1. The CSP maintains cloud platforms that pool hard and soft resources to provide data storage and query services.

TABLE 1
Comparison with Previous Work

	Rank	Dynamic	Verifiability	Sublinear
Refs. [13], [14], [15]		✓		✓
Refs. [21], [22]			✓	
Refs. [23], [24], [25]		✓	✓	
Ref. [26]		✓	✓	✓
Refs. [27], [31], [32], [33]	✓	✓		
Ref. [29]	✓		✓	
Ref. [30]	✓	✓	✓	
Our VDERS	✓	✓	✓	✓

The data owner first creates ciphertexts **c** for a document collection **d**. Given keywords **w** extracted from **d**, she then builds a secure index I for fast searches, and generates a local evidence Ψ and remote auxiliary information Φ for verifiable searches. After uploading $(\mathbf{c}, \mathrm{I}, \Phi)$ to the cloud, she can perform updates on ciphertexts with an update token $\mathcal{T}_*$ and retrieve documents *on demand* with a search token $\mathcal{T}_W$ in a *verifiable* way. On receiving the search results and a search proof $(\mathcal{R}_W, \Pi)$ from the CSP, the data owner recovers document contents after verifying the correctness of the search results. The data owner can also delegate the search/update/verification ability to authorized data users. In this paper, we do not differentiate between the data owner and the data user, and refer to them as *users*.

3.2 Adversary Model

We assume that the users are fully trusted. The CSP is the potential attacker and is assumed to be honest but curious [7]. That is, the CSP would correctly execute the prespecified protocol, but still attempt to learn extra information about the stored data and the received message [34].

As defined in [12], *access pattern* refers to the outcome of search results, i.e., which documents have been returned; *search pattern* refers to whether two searches have been performed for the same keyword. As a tradeoff between security and efficiency, existing SSE schemes resort to the weakened security guarantee for efficiency concerns. That is, they will reveal the access pattern and the search pattern but nothing else during the search process. Like existing SSE schemes, such information is also available to the CSP in our scheme. Furthermore, in a top-K search, only K highest ranked documents will be returned. Therefore, our scheme will leak information about document ranks besides access pattern and search pattern. It is worth noticing that the leakage of ranking information is inevitable in top-K searches. For example, the adversary first issues a top-1 search and then a top-2 search for keyword W, thereby it can know that the document returned in the first round has the highest rank and the new document returned in the second round is ranked second. If the adversary issues top-$1, 2, \ldots,$ top-K continuously, then the rank of each document will be exposed by comparing search results.

Our scheme mainly aims to preserve the following privacy properties:

- *Confidentiality.* The CSP knows nothing about the document/keyword contents expect search pattern, access pattern, and document ranks.

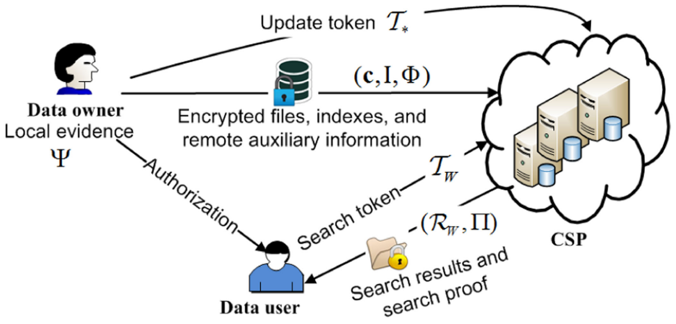

Update token $\mathcal{T}_*$
Data owner
Local evidence
Ψ
$(\mathbf{c}, \mathrm{I}, \Phi)$
Encrypted files, indexes, and remote auxiliary information
Authorization
Search token
$\mathcal{T}_W$
$(\mathcal{R}_W, \Pi)$
CSP
Data user
Search results and search proof

Fig. 1. System model. Communication channels are secured with security protocols such as SSL/SSH.

- *Verifiability.* The CSP cannot forge a search result or falsify the outsourced ciphertexts.

3.3 RSA Accumulator

RSA accumulator [10] is applied to verify the correctness of the search results in our scheme. It provides a constant-size digest for an arbitrarily large set of inputs and a constant-size witness for any element in the set such that it can be used to verify the (non-)membership of the element in this set. Let $\kappa \in \mathbb{N}$ be a security parameter, and let $p = 2p' + 1$ and $q = 2q' + 1$ be two large primes where p', q' are primes such that $|pq| > 3\kappa$. Let $\mathcal{F} = \{f : \{0,1\}^{3\kappa} \rightarrow \{0,1\}^{\kappa}\}$ be a family of two-universal hash functions, let $\mathbf{N} = pq$ and $\varphi(\mathbf{N}) = (p-1)(q-1)$, and let $\mathbb{G}$ be a cyclic group of size $(p-1)(q-1)/4$ where g is a generator of $\mathbb{G}$. For a set of elements $E = \{y_1, \ldots, y_n\}$ with $y_i \in \{0,1\}^{\kappa}$, the RSA accumulator works as follows:

- For each $y_i \in E$, we choose a random prime x_i, denoted by $\mathcal{P}(y_i)$, such that $y_i = f(x_i)$. The accumulator is calculated as $Acc(E) = g^{\prod_{i=1}^{n} \mathcal{P}(y_i)} \bmod \mathbf{N}$.
- For any subset $E' \subseteq E$, a witness $\pi = g^{\prod_{y_i \in E-E'} \mathcal{P}(y_i)} \bmod \mathbf{N}$ is produced.
- The subset test is carried out by checking $Acc(E) = \pi^{\prod_{y_i \in E'} \mathcal{P}(y_i)} \bmod \mathbf{N}$.

Proposition 1. *For any $y \in \{0,1\}^{\kappa}$, we can compute a prime $x \in \{0,1\}^{3\kappa}$ such that $f(x) = y$ with overwhelming probability from the set of inverses $f^{-1}(y)$ by sampling $O(\kappa^2)$ times.*

Strong RSA Assumption. Given $\mathbf{N} = pq$ and a random element $y \in \mathbb{Z}_{\mathbf{N}}$, it is hard to find x and $e > 1$ such that $y = x^e \bmod \mathbf{N}$.

Proposition 2. *Given $\mathbf{N}, g, f$, and $E = \{y_1, \ldots, y_n\}$, it is hard to find $y \notin E$ and π such that $\pi^{\mathcal{P}(y)} = Acc(E) \bmod \mathbf{N}$ under the strong RSA assumption.*

Given Proposition 1, a prime x_i can be computed efficiently for any $y_i \in E$. The security of RSA accumulator is based on *strong RSA assumption* and Proposition 2.

4 Scheme Overview

4.1 Notations

Let $\kappa \in \mathbb{N}$ be a secure parameter of the whole system, and let $\mathbf{0}$ be a string of 0s with length κ. For $t \in \mathbb{N}$, notation $[t]$ is used to denote the set of integers $\{1, \ldots, t\}$. The set of all

TABLE 2
Summary of Notations

$\mathbf{d}$	A sequence of n documents $(D_1, \ldots, D_n)$
$\mathbf{w}$	A sequence of m keywords $(W_1, \ldots, W_m)$
$\mathbf{c}$	A sequence of n ciphertexts $(C_1, \ldots, C_n)$
$\mathbf{d}_W$	A sequence of documents containing keyword W
$\mathbf{w}_j$	A sequence of keywords contained in document D_j
I	A ranked inverted index $(\mathrm{T}_s, \mathrm{A}_s)$
Ψ	The local evidence (ψ_C, ψ_I)
Φ	The remote auxiliary information
Π	The search proof (π_C, π_I)
$\mathcal{T}_W$	A search token generated for keyword W
$\mathcal{T}_*$	An update token generated for document D
$\mathbf{id}_W$	A sequence of document identifiers $(\mathrm{ID}_1, \ldots, \mathrm{ID}_{\lvert\mathbf{d}_W\rvert})$ where ID_j is rank-j document's identifier for keyword W
$\mathbf{s}_W$	A sequence of scores $(\varepsilon_1, \ldots, \varepsilon_{\lvert\mathbf{d}_W\rvert})$ where ε_j is rank-j document's score for keyword W
$\mathbf{c}_W$	A sequence of ciphertexts $(C_{\mathrm{ID}_1}, \ldots, C_{\mathrm{ID}_{\lvert\mathbf{d}_W\rvert}})$ where C_{ID_j} is rank-j document's ciphertext for keyword W
$\mathcal{R}_W$	The search results for search token $\mathcal{T}_W$

binary strings of length t is denoted by $\{0,1\}^t$ and the set of finite binary strings by $\{0,1\}^*$. Given a sequence of elements $\mathbf{v}$, its ith element is denoted by $\mathbf{v}[i]$ or V_i and its total number of elements by $|\mathbf{v}|$. Given a matrix $\mathbf{M}$, the element in its ith row and jth column is denoted by $\mathbf{M}[i][j]$. If S is a set then $|S|$ refers to its cardinality. If s is a string then $||s||$ refers to its bit length. The concatenation of t strings $s_1, \ldots, s_t$ is denoted by $\langle s_1, \ldots, s_t \rangle$.

The data set consists of a sequence of n documents $\mathbf{d} = (D_1, \ldots, D_n)$, where the jth document D_j, associated with identifier j, contains a sequence of keywords $\mathbf{w}_j$ for $j \in [n]$. The ciphertext collection is denoted by $\mathbf{c} = (C_1, \ldots, C_n)$, where C_j is the ciphertext of document D_j for $j \in [n]$. The universal distinct keywords extracted from $\mathbf{d}$ are denoted by $\mathbf{w} = \mathbf{w}_1 \cup \ldots \cup \mathbf{w}_n = (W_1, \ldots, W_m)$. The ith keyword $W_i \in \mathbf{w}$, associated with identifier i, is contained in a sequence of documents $\mathbf{d}_{W_i}$, for $i \in [m]$. We assume that the identifier associated with each document/keyword is independent of its contents, and thus is allowed to be exposed to the CSP. Let ξ be a function outputting the identifier of a document or a keyword. For document $D \in \mathbf{d}$ and keyword $W \in \mathbf{w}$, their identifiers are denoted by $\xi(D)$ and $\xi(W)$, respectively. The most relevant notations in our scheme are shown in Table 2.

4.2 Ranked Inverted Index

In the inverted index [12], a list of nodes pointing to documents containing keyword $W \in \mathbf{w}$ are randomly stored in a search array A_s and the pointer to the head node is stored in a search table T_s. The overwhelming advantage of an inverted index is its search efficiency, i.e., the complexity of searching keyword W is $O(|\mathbf{d}_W|)$, which is not only sub-linear, but also optimal. As an extension, our ranked inverted index I is composed of a search table T_s and a search array A_s. Let -1 denote a $(\log \#\mathrm{A}_s + 1)$-length string of 1s. For each keyword $W \in \mathbf{w}$, a ranked linked list L_W that chains a list of nodes in A_s is defined as follows:

Definition 1 (Ranked linked list). *L_W links $|\mathbf{d}_W|$ nodes $(\mathrm{N}_1, \ldots, \mathrm{N}_{|\mathbf{d}_W|})$ and each node is defined as $\mathrm{N}_j = \langle id_j, \varepsilon_j, addr_s(\mathrm{N}_{j+1})\rangle$, where id_j is identifier of the jth document in $\mathbf{d}_W$, ε_j is the score[1] of the jth document in $\mathbf{d}_W$ for keyword W[2], and $addr_s(\mathrm{N}_{j+1})$ is the address of node N_{j+1} in a search array A_s. In the special case, $addr_s(\mathrm{N}_{|\mathbf{d}_W|+1}) = -1$.*

As for data structures, the search array A_s is an array with $\mathrm{A}_s[i]$ denoting the value stored at location i of A_s and $\mathrm{A}_s[i] \leftarrow v$ denoting storing v at location i of A_s. The search table T_s is a dictionary that stores key-value pairs. If a pair (k, v) exists in T_s, then v is the value associated with key k in T_s. Furthermore, $\mathrm{T}_s[k] \leftarrow v$ denotes storing the value v under key k in T_s. Let $\#\mathrm{A}_s$ and $\#\mathrm{T}_s$ denote the size of A_s and T_s, respectively. Similar to [13], we set $\#\mathrm{T}_s = m + 1$, where the first m entries correspond to keywords in $\mathbf{w}$ and the last entry points to an unused entry in A_s; and we set $\#\mathrm{A}_s = ||\mathbf{c}||/8 + z$ to protect statistical information about $\mathbf{d}$, where $||\mathbf{c}||$ is the bit length of the ciphertext collection and $z \in \mathbb{N}$ is the number of unused entries in A_s.

4.3 Verifiable Matrix

Let $H : \{0,1\}^* \rightarrow \{0,1\}^\kappa$ be a collision-free hash function, let σ be a pseudo-random permutation (PRP) on $[m]$, and let $S : \{0,1\}^\kappa \times \{0,1\}^* \rightarrow \{0,1\}^\kappa$ be a pseudo-random function (PRF) with key k_4. The verifiable matrix $\mathbf{V}$ is an $m \times n$ matrix defined as follows:

$$\mathbf{V}[i][j] = \begin{cases} H(\mathrm{ID}_{j-1}, \mathrm{ID}_j, \mathrm{ID}_{j+1}) \oplus S_{k_4}(W), j \in [|\mathbf{d}_W|] \\ \gamma_{ij}, \qquad otherwise, \end{cases} \tag{1}$$

where $i = \sigma(\xi(W))$, ID_j denotes the identifier of the rank-j document for keyword W, and γ_{ij} is a random string of length κ stored at $\mathbf{V}[i][j]$. Therefore, $\mathbf{V}[i][j]$ records the information of the rank-$(j-1)$, rank-j, and rank-$(j+1)$ documents for keyword W. In the special case, we have $\mathrm{ID}_0 = \mathrm{ID}_{|\mathbf{d}_W|+1} = \mathbf{0}$.

4.4 Scheme Definition

The VDERS scheme consists of the following algorithms:

- $(params, SK) \leftarrow$ GenKey(1^κ): It takes a security parameter κ as input and outputs public parameters $params$ and a secret key SK.
- $(\mathrm{I}, \mathbf{c}) \leftarrow$ Encryption$(SK, \mathbf{d}, \mathbf{w})$: It takes a secret key SK, a sequence of documents $\mathbf{d}$, and a sequence of keywords $\mathbf{w}$ as inputs, and outputs a ranked inverted index I and a sequence of ciphertexts $\mathbf{c}$.
- $(\Phi, \Psi) \leftarrow$ AccGen$(params, SK, \mathbf{d}, \mathbf{w}, \mathbf{c})$: It takes public parameters $params$, a secret key SK, a sequence of documents $\mathbf{d}$, a sequence of keywords $\mathbf{w}$, and a sequence of ciphertexts $\mathbf{c}$ as inputs. It outputs remote auxiliary information Φ and a local evidence Ψ.
- $\mathcal{T}_W \leftarrow$ SrcToken(SK, W): It takes a secret key SK and a keyword $W \in \mathbf{w}$ as inputs to generate a search token $\mathcal{T}_W$.
- $\mathcal{R}_W \leftarrow$ Search$(\mathrm{I}, \mathcal{T}_W)$: It takes a search token $\mathcal{T}_W$ and an index I as inputs to output search results $\mathcal{R}_W$.

1. For the single keyword search, we choose the term frequency (TF) as the relevance score, which is calculated by $\mathrm{TF} = \frac{D_W}{|D|}$, where D_W denotes the frequency of the keyword W appearing in the document D and $|D|$ denotes the number of words in D.

2. The role of scores is to order documents in top-K searches. Therefore, there is no need to use authentic scores for comparison, and all scores will be preprocessed by order preserving functions like [35].

- $\Pi \leftarrow \mathsf{GenProof}(params, \mathbf{c}, \Phi, \mathcal{T}_W, \mathcal{R}_W)$: It takes public parameters $params$, a sequence of ciphertexts $\mathbf{c}$, remote auxiliary information Φ, a search token $\mathcal{T}_W$, and search results $\mathcal{R}_W$ as inputs to generate a search proof Π.
- $\{0,1\} \leftarrow \mathsf{Verify}(params, SK, \Psi, \mathcal{T}_W, \mathcal{R}_W, \Pi)$: It takes public parameters $params$, a secret key SK, a local evidence Ψ, a search token $\mathcal{T}_W$, search results $\mathcal{R}_W$, and a search proof Π as inputs. It outputs 1 if verification is successful, and outputs 0 otherwise. If the output is 1, for each ciphertext in $\mathcal{R}_W$, it takes a secret key SK as input to recover plaintext.
- $(\mathcal{T}_*, \Psi') \leftarrow \mathsf{UpdToken}(params, SK, D, \Psi)$: It takes public parameters $params$, a secret key SK, a document D, and a local evidence Ψ as inputs to generate an update token $\mathcal{T}_*$ and a new local evidence Ψ', where $* \in \{\mathbf{add}, \mathbf{del}\}$.
- $(\mathrm{I}', \mathbf{c}', \Phi') \leftarrow \mathsf{Update}(\mathrm{I}, \mathbf{c}, \Phi, \mathcal{T}_*)$: It takes a ranked inverted index I, a ciphertext collection $\mathbf{c}$, remote auxiliary information Φ, and an update token $\mathcal{T}_*$ as inputs. It generates a new ranked inverted index I', a new ciphertext collection $\mathbf{c}'$, and new remote auxiliary information Φ'.

Definition 2 (Correctness of VDERS). *VDERS is correct if for all $\kappa \in \mathbb{N}$, for all keys generated by the* GenKey *algorithm, for all $(\mathrm{I}, \mathbf{c})$ output by the* Encryption *algorithm, and for all updates, the* Search *algorithm always returns the correct search results.*

4.5 Security Definition

Confidentiality. We follow the approach in [13] that utilizes leakage functions to capture what is being revealed by ciphertexts and tokens. Let Adv be a stateful adversary, let Sim be a stateful simulator, and let $\mathcal{L}_1(\mathbf{d})$,$\mathcal{L}_2(\mathbf{d}, W)$, $\mathcal{L}_3(\mathbf{d}, D)$, and $\mathcal{L}_4(\mathbf{d}, D)$ be stateful leakage algorithms. We consider the following probabilistic experiments.

- $\mathbf{Real}_{\mathrm{Adv}}(\kappa)$: the challenger runs $\mathsf{GenKey}(1^\kappa)$ to generate a secret key SK. Adv outputs $\mathbf{d}$ and receives $(\mathrm{I}, \mathbf{c})$ from the challenger so that $(\mathrm{I}, \mathbf{c}) \leftarrow \mathsf{Encryption}(SK, \mathbf{d}, \mathbf{w})$. Adv makes a polynomial number of adaptive queries, and for each query, it receives from the challenger either a search token $\mathcal{T}_W$ such that $\mathcal{T}_W \leftarrow \mathsf{SrcToken}(SK, W)$ or an update token $\mathcal{T}_*$ such that $(\mathcal{T}_*, \Psi') \leftarrow \mathsf{UdpToken}(params, SK, D, \Psi)$. Finally, Adv returns a bit b that is output by the experiment.
- $\mathbf{Ideal}_{\mathrm{Adv,Sim}}(\kappa)$: Adv outputs $\mathbf{d}$. Given $\mathcal{L}_1(\mathbf{d})$, Sim generates and sends a pair $(\mathrm{I}, \mathbf{c})$ to Adv. Adv makes a polynomial number of adaptive queries. Given either $\mathcal{L}_2(\mathbf{d}, W)$, $\mathcal{L}_3(\mathbf{d}, D)$, or $\mathcal{L}_4(\mathbf{d}, D)$, Sim returns an appropriate token and in the case of an add operation, a ciphtertext C. Finally, Adv returns a bit b that is output by the experiment.

Definition 3 (Confidentiality of VDERS). *VDERS is confidential if for all probabilistic polynomial-time (PPT) adversaries Adv, there exists a PPT simulator Sim such that $|\Pr[\mathbf{Real}_{\mathrm{Adv}}(\kappa) = 1] - \Pr[\mathbf{Ideal}_{\mathrm{Adv,Sim}}(\kappa) = 1]|$ is negligible.*

Intrinsically, the confidentiality of VDERS is equivalent to the CKA2 security of [13], in which Adv's views given by Sim are indistinguishable from those in the real world.

Verifiability. It describes the fact that no adversary can deceive the user to accept incorrect search results. As in [23], we say that the adversary Adv wins if given $(\mathbf{d}, \mathbf{w}, \mathrm{I})$ and search queries Adv can return $(\overline{\mathcal{R}}_W, \overline{\Pi})$ for some query such that $\overline{\mathcal{R}}_W \neq \mathcal{R}_W$ and the user accepts the result.

Definition 4 (Verifiability of VDERS). *VDERS is verifiable if for any PPT adversary Adv, $\Pr(\mathrm{Adv}\ wins)$ is negligible for any $(\mathbf{d}, \mathbf{w}, \mathrm{I})$ and any query.*

5 Basic VDERS Scheme

5.1 Main Idea

The foundational technologies in our VDERS scheme are RSA accumulator (defined in Section 3.3), the ranked inverted index (defined in Section 4.2), and the verifiable matrix (defined in Section 4.3). Our main idea is first building a ranked inverted index to achieve sublinear search time in top-K queries, and then verifying the correctness of search results based on RSA accumulator and the verifiable matrix.

Specially, the search index is constructed as a ranked inverted index. Unlike the inverted index [12] in which each node contains only information about document identifier and the location of the next node in the search array, our ranked inverted index incorporates ranking information in each node to enable top-K queries in cloud computing.

Furthermore, inspired by the work in [23], our VDERS scheme also employs RSA accumulator for dynamic verification. The main difference from their work is that our scheme constructs a verifiable matrix $\mathbf{V}$ to record the document ranking information for each keyword, and calculates the remote auxiliary information Φ and a local evidence Ψ by encoding the elements of $\mathbf{V}$ into RSA accumulator.

5.2 Our Construction

Let $\mathcal{F} = \{f : \{0,1\}^{3\kappa} \to \{0,1\}^\kappa\}$ be a family of two-universal hash functions and let $\mathsf{SKE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ be a symmetric-key encryption (SKE) scheme, where Gen is a key generation algorithm, Enc is an encryption algorithm, and Dec is a decryption algorithm. Let $\mathbb{H} : \{0,1\}^* \to \{0,1\}^*$ be a random oracle and let $H : \{0,1\}^* \to \{0,1\}^\kappa$ be a collision-free hash function. To compute $H(x_1, \ldots, x_n)$, we first transform each input x_i into a bit string $\hat{x}_i$ and then take the concatenation of $\langle \hat{x}_1, \ldots, \hat{x}_n \rangle$ as the inputs of H. Let $F : \{0,1\}^\kappa \times \{0,1\}^* \to \{0,1\}^\kappa$, $G : \{0,1\}^\kappa \times \{0,1\}^* \to \{0,1\}^*$, $P : \{0,1\}^\kappa \times \{0,1\}^* \to \{0,1\}^\kappa$, and $S : \{0,1\}^\kappa \times \{0,1\}^* \to \{0,1\}^\kappa$ be PRFs. VDERS^0 is constructed as follows:

- $(params, SK) \leftarrow \mathsf{GenKey}(1^\kappa)$: Given $(\mathbf{N} = pq, g)$ as shown in Section 3.3, the user randomly chooses a hash function $f \in \mathcal{F}$ and four κ-bit strings k_1, k_2, k_3, k_4 as keys of PRFs[3]. Then, she runs algorithm $\mathsf{SKE.Gen}$ to generate the key k_e of SKE, chooses a PRP σ on $[m]$, and sets the public parameters as $params = (\mathbf{N}, g, f)$ and the secret key as $SK = (p, q, \sigma, k_e, k_1, k_2, k_3, k_4)$.
- $(\mathrm{I}, \mathbf{c}) \leftarrow \mathsf{Encyption}(SK, \mathbf{d}, \mathbf{w})$: Assume that -1 denotes a $(\log \#\mathrm{A}_s + 1)$-length string of 1s and that

3. A prime x is chosen randomly from f^{-1}. A secret key $k_a \in \{0,1\}^\kappa$ shared between the user and the CSP is used as the randomness when computing x, so that they obtain the same outputs from f^{-1}.

free denotes a word not in $\mathbf{w}$. The user runs Algorithm 1 to build a ranked inverted index $\mathrm{I} = (\mathrm{T}_s, \mathrm{A}_s)$ and a sequence of ciphertexts $\mathbf{c} = (C_1, \ldots, C_n)$.

Algorithm 1. VDERS^0.Encryption

```
Input: Secret key $SK$, documents $\mathbf{d}$, and keywords $\mathbf{w}$
OutPut: Ranked inverted index I and ciphertexts $\mathbf{c}$
1:  Initialize the search table $\mathrm{T}_s$ with an empty dictionary of size $m+1$ and initialize the search array $\mathrm{A}_s$ with an empty array of size $||\mathbf{c}||/8 + z$
2:  for each keyword $W \in \mathbf{w}$ do
3:      Create a ranked linked list $\mathrm{L}_W = (\mathrm{N}_1, \ldots, \mathrm{N}_{|\mathbf{d}_W|})$ by randomly choosing $|\mathbf{d}_W|$ unused entries in $\mathrm{A}_s$ where $\mathrm{N}_i = \langle id_i, \varepsilon_i, addr_s(\mathrm{N}_{i+1})\rangle$ is defined in Definition 1
4:      for $i \in [|\mathbf{d}_W|]$ do
5:          Choose a $\kappa$-bit random string $r_i$ and set
                $\mathrm{A}_s[addr_s(\mathrm{N}_i)] \leftarrow (\mathrm{N}_i \oplus \mathbb{H}(P_{k_3}(W), r_i), r_i)$
6:      Encrypt the address of the head node $\mathrm{N}_1$ and set
                $\mathrm{T}_s[F_{k_1}(W)] \leftarrow addr_s(\mathrm{N}_1) \oplus G_{k_2}(W)$
7:  Create a free list $\mathrm{L}_{\mathbf{free}} = (\mathrm{F}_1, \ldots, \mathrm{F}_z)$ by randomly choosing $z$ unused entries in $\mathrm{A}_s$ where $\mathrm{F}_i = \langle \mathbf{0}, \mathbf{0}, addr_s(\mathrm{F}_{i-1})\rangle$ and $addr_s(\mathrm{F}_0) = -1$
8:  for $i = z$ to 1 do
9:      Set $\mathrm{A}_s[addr_s(\mathrm{F}_i)] \leftarrow (\mathrm{F}_i, \mathbf{0})$
10: Store the address of the tail node $\mathrm{F}_z$ in $\mathrm{T}_s$ by setting $\mathrm{T}_s[\mathbf{free}] \leftarrow addr_s(\mathrm{F}_z)$
11: Fill the remaining entries of $\mathrm{A}_s$ with random strings
12: Set $\mathrm{I} = (\mathrm{T}_s, \mathrm{A}_s)$
13: for $j \in [n]$ do
14:     Set $C_j = \mathsf{SKE.Enc}(k_e, D_j)$ and $\mathbf{c} = \mathbf{c} \cup C_j$
15: return $(\mathrm{I}, \mathbf{c})$
```

In Algorithm 1, the user first builds a ranked inverted index I as follows: After initializing the search table T_s and the search array A_s (Line 1), for each keyword $W \in \mathbf{w}$, she creates an encrypted list L_W of nodes storing at random locations of A_s, and stores the encrypted address of the head node in T_s (Lines 2-6). Then, she creates an unencrypted free list $\mathrm{L}_{\mathbf{free}}$ by choosing z unused entries at random in A_s and stores the address of the tail node in T_s (Lines 7-10). Next, she fills the remaining entries of A_s with random strings to hind statistical information about the document collection (Lines 11-12). Finally, she generates a sequence of ciphertexts $\mathbf{c}$ and returns $(\mathrm{I}, \mathbf{c})$ as the outputs of Algorithm 1 (Lines 13-15).

- $(\Phi, \Psi) \leftarrow \mathsf{AccGen}(params, SK, \mathbf{d}, \mathbf{w}, \mathbf{c})$: The user constructs a verifiable matrix $\mathbf{V}$ as defined in Section 4.3, and sets the remote auxiliary information as $\Phi = \mathbf{V}$. Let $\mathcal{P}(y)$ be a random prime x such that $f(x) = y$. She calculates a local evidence $\Psi = (\psi_C, \psi_I)$ with

$$\psi_C = g^{\prod_{i=1}^{n} \mathcal{P}(H(i, H(C_i)))} \bmod \mathbf{N}, \tag{2}$$

$$\psi_I = g^{\prod_{i=1}^{m} \prod_{j=1}^{n} \mathcal{P}(H(i, \mathbf{V}[i][j]))} \bmod \mathbf{N}. \tag{3}$$

- $\mathcal{T}_W \leftarrow \mathsf{SrcToken}(SK, W)$: To retrieve the top-$K$ documents, the user generates a search token $\mathcal{T}_W = (F_{k_1}(W), G_{k_2}(W), P_{k_3}(W), K, \sigma(\xi(W)))$ for keyword $W \in \mathbf{w}$. If the user wants to retrieve all documents containing keyword W, K in $\mathcal{T}_W$ is replaced by notation $*$.
- $\mathcal{R}_W \leftarrow \mathsf{Search}(\mathrm{I}, \mathcal{T}_W)$: For top-$K$ queries, the CSP runs Algorithm 2 to output search results $\mathcal{R}_W = (\mathbf{id}_W^{K+1}, \mathbf{s}_W^{K+1}, \mathbf{c}_W^{K})$, where $\mathbf{id}_W^j$, $\mathbf{s}_W^j$, and $\mathbf{c}_W^j$ denote the first j identifiers, scores, and ciphertexts in $\mathbf{id}_W$, $\mathbf{s}_W$, and $\mathbf{c}_W$, respectively. For normal queries, search results $\mathcal{R}_W$ are set as $(\mathbf{id}_W, \mathbf{s}_W, \mathbf{c}_W)$.

In Algorithm 2, the CSP parses $\mathcal{T}_W$ as $(\tau_1, \tau_2, \tau_3, \tau_4, \tau_5)$, and returns $\emptyset$ if τ_1 is not in T_s (Lines 1-3). Otherwise, the CSP first recovers the address of head node of L_W in A_s and recovers all nodes of L_W in sequence (Lines 4-8). Then, it sorts documents according to their scores and returns identifiers, scores, and ciphertexts of the matched documents as the search results $\mathcal{R}_W$ (Lines 9-17).

Algorithm 2. VDERS^0.Search

```
Input: Ranked inverted index I and search token $\mathcal{T}_W$
OutPut: Search results $\mathcal{R}_W$
1:  Parse $\mathcal{T}_W$ as $(\tau_1, \tau_2, \tau_3, \tau_4, \tau_5)$
2:  if $\tau_1$ is not in $\mathrm{T}_s$ then
3:      return $\emptyset$
4:  Compute $\mathrm{T}_s[\tau_1] \oplus \tau_2$ and recover $addr_s(\mathrm{N}_1)$ the address of the head node $\mathrm{N}_1$ of $\mathrm{L}_W$ in $\mathrm{A}_s$
5:  repeat
6:      Obtain the encrypted node $\mathrm{N}_i$ stored at $\mathrm{A}_s[addr_s(\mathrm{N}_i)]$ and parse $\mathrm{N}_i$ as $(v_i, r_i)$
7:      Compute $v_i \oplus \mathbb{H}(\tau_3, r_i)$ and recover node $\mathrm{N}_i = \langle id_i, \varepsilon_i, addr_s(\mathrm{N}_{i+1})\rangle$
8:  until $addr_s(\mathrm{N}_{i+1}) = -1$
9:  Sort documents according to their scores
10: Let $\mathrm{ID}_j$ and $\varepsilon_j$ be the identifier and score of the rank-$j$ document for $W$, respectively
11: if $\tau_4 \neq *$ then
12:     Set $\mathbf{id}_W^{K+1} = (\mathrm{ID}_1, \ldots, \mathrm{ID}_{K+1})$, $\mathbf{s}_W^{K+1} = (\varepsilon_1, \ldots, \varepsilon_{K+1})$, and $\mathbf{c}_W^K = (C_{\mathrm{ID}_1}, \ldots, C_{\mathrm{ID}_K})$
13:     Set $\mathcal{R}_W \leftarrow (\mathbf{id}_W^{K+1}, \mathbf{s}_W^{K+1}, \mathbf{c}_W^K)$
14: else
15:     Set $\mathbf{id}_W = (\mathrm{ID}_1, \ldots, \mathrm{ID}_{|\mathbf{d}_W|})$, $\mathbf{s}_W = (\varepsilon_1, \ldots, \varepsilon_{|\mathbf{d}_W|})$, and $\mathbf{c}_W = (C_{\mathrm{ID}_1}, \ldots, C_{\mathrm{ID}_{|\mathbf{d}_W|}})$
16:     Set $\mathcal{R}_W \leftarrow (\mathbf{id}_W, \mathbf{s}_W, \mathbf{c}_W)$
17: return $\mathcal{R}_W$
```

- $\Pi \leftarrow \mathsf{GenProof}(params, \mathbf{c}, \Phi, \mathcal{T}_W, \mathcal{R}_W)$: For top-$K$ queries, the CSP parses $\mathcal{T}_W$ as $(\tau_1, \tau_2, \tau_3, \tau_4, \tau_5)$, and sets search proof $\Pi = (\pi_C, \pi_I)$ with

$$\pi_C = g^{\prod_{C_i \in \mathbf{c}, C_i \notin \mathbf{c}_W^K} \mathcal{P}(H(i, H(C_i)))} \bmod \mathbf{N}, \tag{4}$$

$$\pi_I = g^{\prod_{i=1, i \neq \tau_5}^{m} \phi_i \cdot \prod_{j=K+1}^{n} \mathcal{P}(H(\tau_5, \mathbf{V}[\tau_5][j]))} \bmod \mathbf{N}, \tag{5}$$

where $\phi_i = \prod_{j=1}^{n} \mathcal{P}(H(i, \mathbf{V}[i][j]))$. For normal queries, parameter K in both Eqs. (4) and (5) is replaced by $|\mathbf{d}_W|$.
- $\{0,1\} \leftarrow \mathsf{Verify}(params, SK, \Psi, \mathcal{T}_W, \mathcal{R}_W, \Pi)$: Let ID_j be the identifier of the rank-j document containing keyword W. To verify a top-K query, for $i \in [K]$, the user computes x_i with Eq. (6) and checks whether Eq. (7) holds.

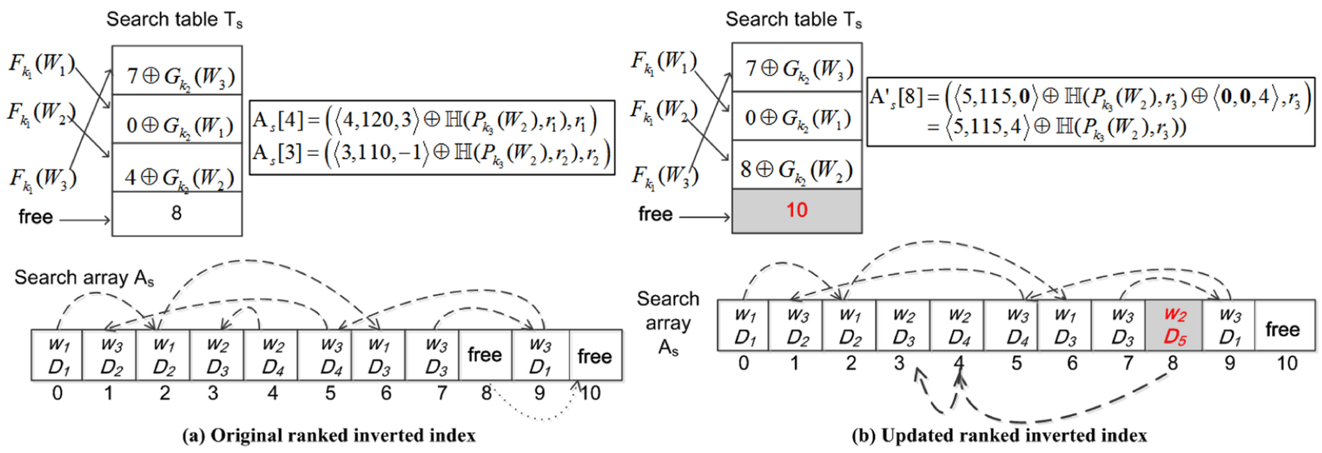

Search table T_s
$F_{k_1}(W_1)$
$F_{k_1}(W_2)$
$F_{k_1}(W_3)$
free
$7\oplus G_{k_2}(W_3)$
$0\oplus G_{k_2}(W_1)$
$4\oplus G_{k_2}(W_2)$
8
$A_s[4]=(\langle 4,120,3\rangle\oplus\mathbb{H}(P_{k_3}(W_2),r_1),r_1)$
$A_s[3]=(\langle 3,110,-1\rangle\oplus\mathbb{H}(P_{k_3}(W_2),r_2),r_2)$
Search array A_s
free
(a) Original ranked inverted index
$8\oplus G_{k_2}(W_2)$
10
$A'_s[8]=(\langle 5,115,\mathbf{0}\rangle\oplus\mathbb{H}(P_{k_3}(W_2),r_3)\oplus\langle\mathbf{0},\mathbf{0},4\rangle,r_3)$
$=\langle 5,115,4\rangle\oplus\mathbb{H}(P_{k_3}(W_2),r_3))$
(b) Updated ranked inverted index

Fig. 2. Samples of ranked inverted indexes.

$$x_i \leftarrow \mathcal{P}(H(\mathrm{ID}_i, H(C_{\mathrm{ID}_i}))). \tag{6}$$

$$\psi_C = (\pi_C)^{\prod_{i=1}^{K} x_i} \bmod \mathbf{N}. \tag{7}$$

Next, she parses $\mathcal{T}_W$ as $(\tau_1, \tau_2, \tau_3, \tau_4, \tau_5)$, and reconstructs the first K elements $\mathbf{V}[\tau_5][1], \ldots, \mathbf{V}[\tau_5][K]$ at the τ_5-row of the verifiable matrix $\mathbf{V}$ with Eq. (1). For $j \in [K]$, she computes y_j with Eq. (8) and checks whether Eq. (9) holds.

$$y_j \leftarrow \mathcal{P}(H(\tau_5, \mathbf{V}[\tau_5][j])). \tag{8}$$

$$\psi_I = (\pi_I)^{\prod_{j=1}^{K} y_j} \bmod \mathbf{N}. \tag{9}$$

If Eqs. (7) and (9) hold, she outputs 1, and 0 otherwise. For normal queries, parameter K in both Eqs. (7) and (9) is replaced by $|\mathbf{d}_W|$. If the output is 1, for each ciphertext $C_j \in \mathcal{R}_W$, the user runs SKE.Dec to recover document D_j.

- $(\mathcal{T}_*, \Psi') \leftarrow$ UpdToken$(params, SK, D, \Psi)$: To delete document D with $\xi(D) = j$, the user sets the update token as $\mathcal{T}_* = \mathcal{T}_{\mathbf{del}} = (j, \mathbf{delete}, C')$, where C' is output of SKE.Enc$(k_e, \mathbf{0})$. Then, she retrieves C_j from the CSP and generates a new local evidence as $\Psi' = (\psi'_C, \psi'_I)$, where $\psi'_I = \psi_I$ and ψ'_C is computed with

$$\psi'_C = (\psi_C)^{\frac{\mathcal{P}(H(j,H(C')))}{\mathcal{P}(H(j,H(C_j)))} \bmod \varphi(\mathbf{N})} \bmod \mathbf{N}. \tag{10}$$

To add document D with $\xi(D) = n+1$, the user sets the update token as $\mathcal{T}_* = \mathcal{T}_{\mathbf{add}} = ((n+1, \mathbf{add}, C_{n+1}), \zeta, \tau_v, \tau_a)$, where C_{n+1} is the output of SKE.Enc(k_e, D_{n+1}), $\zeta = (\gamma_1, \ldots, \gamma_m)$ are a sequence of κ-bit random strings, $\tau_v = (\beta_1, \ldots, \beta_{|\mathbf{w}_{n+1}|})$ and $\tau_a = (\lambda_1, \ldots, \lambda_{|\mathbf{w}_{n+1}|})$. For the ith keyword $W \in \mathbf{w}_{n+1}$, she computes β_i and λ_i as follows:

1) She computes the scores ε_i of document D_{n+1}, chooses a random string r_i of κ bits, and computes $\lambda_i = (\lambda_i[1], \lambda_i[2], \lambda_i[3], \lambda_i[4], \lambda_i[5])$ as:

$$\begin{aligned} &\lambda_i[1] = F_{k_1}(W), \lambda_i[2] = G_{k_2}(W), \lambda_i[3] = r_i, \\ &\lambda_i[4] = \langle n+1, \varepsilon_i, \mathbf{0}\rangle \oplus \mathbb{H}(P_{k_3}(W), \lambda_i[3]), \\ &\lambda_i[5] = \sigma(\xi(W)). \end{aligned} \tag{11}$$

2) She first performs a normal query to retrieve all documents containing keyword W and verifies the correctness of the search results $\mathcal{R}_W$. Then, she determines the rank of D_{n+1}, denoted by J, by comparing its score, ε_i, with those of documents in $\mathcal{R}_W$, and sets $\beta_i = (\beta_i[1], \beta_i[2], \beta_i[3], \beta_i[4], \beta_i[5], \beta_i[6])$ as:

$$\begin{aligned} &\beta_i[1] = H(\overline{\mathrm{ID}}_{\mathrm{J}-2}, \overline{\mathrm{ID}}_{\mathrm{J}-1}, n+1) \oplus S_{k_4}(W), \\ &\beta_i[2] = H(\overline{\mathrm{ID}}_{\mathrm{J}-1}, n+1, \overline{\mathrm{ID}}_{\mathrm{J}}) \oplus S_{k_4}(W), \\ &\beta_i[3] = H(n+1, \overline{\mathrm{ID}}_{\mathrm{J}}, \overline{\mathrm{ID}}_{\mathrm{J}+1}) \oplus S_{k_4}(W), \\ &\beta_i[4] = H(\overline{\mathrm{ID}}_{\mathrm{J}-2}, \overline{\mathrm{ID}}_{\mathrm{J}-1}, \overline{\mathrm{ID}}_{\mathrm{J}}) \oplus S_{k_4}(W), \\ &\beta_i[5] = H(\overline{\mathrm{ID}}_{\mathrm{J}-1}, \overline{\mathrm{ID}}_{\mathrm{J}}, \overline{\mathrm{ID}}_{\mathrm{J}+1}) \oplus S_{k_4}(W), \\ &\beta_i[6] = \mathrm{J}, \end{aligned} \tag{12}$$

where $\overline{\mathrm{ID}}_j$ is the identifier of the rank-j document for keyword W before update, and $\overline{\mathrm{ID}}_{|\mathbf{d}_W|+1} = \overline{\mathrm{ID}}_0 = \mathbf{0}$. In the special case, if $\mathrm{J} = 1$, we set $\beta_i[1] = \beta_i[4] = \perp$; if $\mathrm{J} = |\mathbf{d}_W| + 1$, we set $\beta_i[3] = \beta_i[5] = \perp$. Then, she generates a new local evidence $\Psi' = (\psi'_C, \psi'_I)$ as follows:

$$\psi'_C = (\psi_C)^{\mathcal{P}(H(n+1, H(C_{n+1})))} \bmod \mathbf{N}. \tag{13}$$

For $i \in [|\mathbf{w}_{n+1}|]$, she computes y_i with:

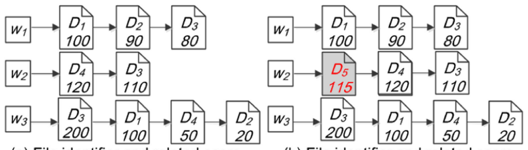
$$y_i = \frac{\prod_{j=1}^{3} \mathcal{P}(H(\lambda_i[5], \beta_i[j]))}{\prod_{j=4}^{5} \mathcal{P}(H(\lambda_i[5], \beta_i[j]))} \bmod \varphi(\mathbf{N}), \tag{14}$$

Note that if $\beta_i[j] = \perp$, $\mathcal{P}(H(\lambda_i[5], \beta_i[j]))$ will be set to 1. Then, she updates ψ_I to ψ'_I with

$$\psi'_I = (\psi_I)^{\prod_{i\in\chi} \mathcal{P}(H(i,\gamma_i)) \prod_{i=1}^{|\mathbf{w}_{n+1}|} y_i} \bmod \mathbf{N}, \tag{15}$$

where $\chi = [m]/\{\lambda_1[5], \ldots, \lambda_{|\mathbf{w}_{n+1}|}[5]\}$ is a set of elements and $\gamma_i \in \zeta$ is a κ-bit random string.

- $(\mathrm{I}', \mathbf{c}', \Phi') \leftarrow$ Update$(\mathrm{I}, \mathbf{c}, \Phi, \mathcal{T}_*)$: The CSP runs algorithm Algorithm 3 to generate the updated versions of ranked inverted index, ciphertext collection, and remote auxiliary information $(\mathrm{I}', \mathbf{c}', \Phi')$. In Algorithm 3, for the deletion operation, the CSP keeps I and $\mathbf{V}$ unchanged, and updates $\mathbf{c}$ by replacing ciphertext C_j with C' (Lines 1-3).



(a) File identifier and related score information before update. (b) File identifier and related score information after update.

Fig. 3. The ranking information of documents.

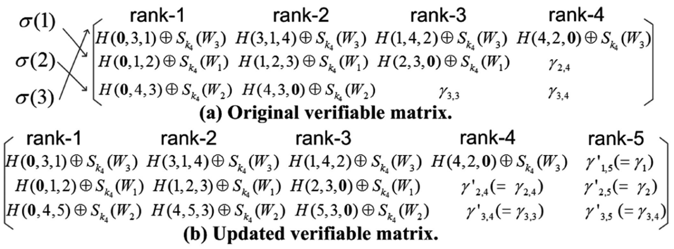
For the addition operation, the CSP first generates $\mathbf{c}'$ by adding C_{n+1} into $\mathbf{c}$ (Lines 4-6). To generate $\mathrm{I}' = (\mathrm{T}_s', \mathrm{A}_s')$, it executes lines 7-13 of Algorithm 3 as follows: For each keyword $W \in \mathbf{w}_{n+1}$, it stores the new node at the last free location ϖ in A_s and sets it as the first node of L_W, and then it updates $\mathrm{T}_s[\lambda_i[1]]$ to point to ϖ and updates $\mathrm{T}_s[\mathbf{free}]$ to point to the second last free location ϖ_{-1} in A_s. Finally, it generates $\Phi' = \mathbf{V}'$ and returns $(\mathrm{I}', \mathbf{c}', \Phi')$ (Lines 14-27).

Algorithm 3. VDERS^0.Update

```
Input: The original ranked inverted index I, the original ciphertexts c, the original remote auxiliary information $\Phi$, and the update token $\mathcal{T}_*$
OutPut: The updated versions of ranked inverted index, ciphertexts, and remote auxiliary information $(\mathrm{I}', \mathbf{c}', \Phi')$
1: Set $\mathrm{I}' = \mathrm{I}$, $\mathbf{c}' = \mathbf{c}$, and $\Phi' = \Phi$ (i.e., $\mathbf{V}' = \mathbf{V}$)
2: if $\mathcal{T}_* = \mathcal{T}_{\mathbf{del}} = (j, \mathbf{delete}, C')$ then
3:     Mark $C_j$ as delete and replace $C_j \in \mathbf{c}'$ with $C'$
4: else
5:     $\mathcal{T}_* = \mathcal{T}_{\mathbf{add}} = ((n+1, \mathbf{add}, C_{n+1}), \zeta, \tau_v, \tau_a)$
6:     Set $\mathbf{c}' = \mathbf{c}' \cup C_{n+1}$
7:     for the $i$-the keyword $W \in \mathbf{w}_{n+1}$ do
8:         Compute $\varpi \leftarrow \mathrm{T}_s[\mathbf{free}]$ and obtain the last free location $\varpi$ in $\mathrm{A}_s$
9:         Compute $((\mathbf{0}, \mathbf{0}, \varpi_{-1}), \mathbf{0}) \leftarrow \mathrm{A}_s[\varpi]$ and obtain the second last free location $\varpi_{-1}$ in $\mathrm{A}_s$
10:        Update the free list to point to the second last free location $\varpi_{-1}$ in $\mathrm{A}_s$ by setting $\mathrm{T}_s[\mathbf{free}] \leftarrow \varpi_{-1}$
11:        Recover the address of the first node $\mathrm{N}_1$ of $\mathrm{L}_W$ in $\mathrm{A}_s$ by computing $\alpha_1 = \mathrm{T}_s(\lambda_i[1]) \oplus \lambda_i[2]$
12:        Store a new node at location $\varpi$ as the first node of $\mathrm{L}_W$ by setting $\mathrm{A}_s[\varpi] \leftarrow (\lambda_i[4] \oplus \langle \mathbf{0}, \mathbf{0}, \alpha_1 \rangle), \lambda_i[3])$
13:        Store the pointer to the first node of $\mathrm{L}_W$ in $\mathrm{T}_s$ by setting $\mathrm{T}_s[\lambda_i[1]] \leftarrow \varpi \oplus \lambda_i[2]$
14:    Add $\zeta$ as the last column of $\mathbf{V}'$
15:    for the $i$th keyword $W \in \mathbf{w}_{n+1}$ do
16:        Set $\mathrm{J} = \beta_i[6]$ and $\tau = \lambda_i[5]$
17:        if $\mathrm{J} < n$ then
18:            for $j = n$ to $\mathrm{J} + 1$ do
19:                Set $\mathbf{V}'[\tau][j+1]$=$\mathbf{V}'[\tau][j]$
20:        if $\mathrm{J} = 1$ then
21:            Set $\mathbf{V}'[\tau][\mathrm{J}] = \beta_i[2]$ and $\mathbf{V}'[\tau][\mathrm{J}+1] = \beta_i[3]$
22:        else
23:            if $\mathrm{J} = |\mathbf{d}_W| + 1$ then
24:                Set $\mathbf{V}'[\tau][\mathrm{J}-1] = \beta_i[1]$ and $\mathbf{V}'[\tau][\mathrm{J}] = \beta_i[2]$
25:            else
26:                Set $\mathbf{V}'[\tau][\mathrm{J}-1] = \beta_i[1]$, $\mathbf{V}'[\tau][\mathrm{J}] = \beta_i[2]$, and $\mathbf{V}'[\tau][\mathrm{J}+1] = \beta_i[3]$
27: return $(\mathrm{I}', \mathbf{c}', \Phi')$
```

5.3 Illustration Example

The correctness of our VDERS scheme can be sketched in the following example.



$$\begin{array}{c|cccc} & \text{rank-1} & \text{rank-2} & \text{rank-3} & \text{rank-4} \\ \sigma(1) & H(\mathbf{0},3,1)\oplus S_{k_4}(W_3) & H(3,1,4)\oplus S_{k_4}(W_3) & H(1,4,2)\oplus S_{k_4}(W_3) & H(4,2,\mathbf{0})\oplus S_{k_4}(W_3) \\ \sigma(2) & H(\mathbf{0},1,2)\oplus S_{k_4}(W_1) & H(1,2,3)\oplus S_{k_4}(W_1) & H(2,3,\mathbf{0})\oplus S_{k_4}(W_1) & \gamma_{2,4} \\ \sigma(3) & H(\mathbf{0},4,3)\oplus S_{k_4}(W_2) & H(4,3,\mathbf{0})\oplus S_{k_4}(W_2) & \gamma_{3,3} & \gamma_{3,4} \end{array}$$

(a) Original verifiable matrix.

$$\begin{array}{ccccc} \text{rank-1} & \text{rank-2} & \text{rank-3} & \text{rank-4} & \text{rank-5} \\ H(\mathbf{0},3,1)\oplus S_{k_4}(W_3) & H(3,1,4)\oplus S_{k_4}(W_3) & H(1,4,2)\oplus S_{k_4}(W_3) & H(4,2,\mathbf{0})\oplus S_{k_4}(W_3) & \gamma'_{1,5}(=\gamma_1) \\ H(\mathbf{0},1,2)\oplus S_{k_4}(W_1) & H(1,2,3)\oplus S_{k_4}(W_1) & H(2,3,\mathbf{0})\oplus S_{k_4}(W_1) & \gamma'_{2,4}(=\gamma_{2,4}) & \gamma'_{2,5}(=\gamma_2) \\ H(\mathbf{0},4,5)\oplus S_{k_4}(W_2) & H(4,5,3)\oplus S_{k_4}(W_2) & H(5,3,\mathbf{0})\oplus S_{k_4}(W_2) & \gamma'_{3,4}(=\gamma_{3,3}) & \gamma'_{3,5}(=\gamma_{3,4}) \end{array}$$

(b) Updated verifiable matrix.

Fig. 4. Samples of verifiable matrices.

(Initial phase) Given a sequence of documents $\mathbf{d} = (D_1, D_2, D_3, D_4)$ and a sequence of keywords $\mathbf{w} = (W_1, W_2, W_3)$, document ranks for each keyword are shown in Fig 3a. Given $(params, SK)$, the user encrypts each document $D_i \in \mathbf{d}$ with SKE to generate ciphertext collection $\mathbf{c} = (C_1, \ldots, C_4)$, and constructs the verifiable matrix $\mathbf{V}$ and the ranked inverted index I as shown in Figs. 4a and 2a, respectively.

She then sets the remote auxiliary information as $\Phi = \mathbf{V}$ and calculates the local evidence $\Psi = (\psi_C, \psi_I)$ as:

$$\psi_C = g^{\prod_{i=1}^{4} \mathcal{P}(H(i,H(C_i)))} \bmod \mathbf{N},$$

$$\psi_I = g^{\prod_{i=1}^{3}\prod_{j=1}^{4} \mathcal{P}(H(i,\mathbf{V}[i][j]))} \bmod \mathbf{N}.$$

Finally, she sends $(params, \mathrm{I}, \mathbf{c}, \Phi)$ to the CSP, and keeps (SK, Ψ) locally.

(Search phase) To retrieve the top-1 document containing keyword W_2, the user sends the search token $\mathcal{T}_W = (F_{k_1}(W_2), G_{k_2}(W_2), P_{k_3}(W_2), 1, 3)$ to the CSP.

The CSP locates $\mathrm{T}_s[F_{k_1}(W_2)] = \mathrm{T}_s[3]$ and recovers the address of the first node in A_s by computing $4 = \mathrm{T}_s[3] \oplus G_{k_2}(W_2)$. It then computes $\mathrm{A}_s[4] \oplus \mathbb{H}(P_{k_3}(W_2), r_1)$ to recover the first node $\mathrm{N}_1 = \langle 4, 120, 3 \rangle$, where 4 is the identifier of the first document, 120 is the score, and 3 is the address of the next node in A_s. In the same way, it recovers the second node $\mathrm{N}_2 = \langle 3, 110, -1 \rangle$, where 3 is the identifier of the second document, 110 is file score, and -1 indicates D_3 is the last document containing keyword W_2. The CSP sorts documents according to their scores, and sets the search results as $\mathcal{R}_W = (\mathbf{id}_W^2, \mathbf{s}_W^2, \mathbf{c}_W^1) = ((4,3), (120, 110), C_4)$.

Next, it calculates the search proof $\Pi = (\pi_C, \pi_I)$ as:

$$\pi_C = g^{\prod_{i=1,2,3} \mathcal{P}(H(i,H(C_i)))} \bmod \mathbf{N},$$

$$\pi_I = g^{\prod_{i=1,2} \phi_i \cdot \prod_{j=2,3,4} \mathcal{P}(H(3,\mathbf{V}[3][j]))} \bmod \mathbf{N},$$

where $\phi_i = \prod_{j=1}^{4} \mathcal{P}(H(i, \mathbf{V}[i][j]))$. The messages returned to the user are set as $(\mathcal{R}_W, \Pi)$.

Once the user receives the messages returned, she computes $x = \mathcal{P}(H(4, H(C_4)))$ and checks if $\psi_C = \pi_C^x \bmod \mathbf{N}$. She then reconstructs $\mathbf{V}[3][1] = H(\mathbf{0}, 4, 3) \oplus S_{k_4}(W_2)$, computes $y = \mathcal{P}(H(3, \mathbf{V}[3][1]))$, and checks if $\psi_I = (\pi_I)^y \bmod \mathbf{N}$. Note that, the following equations hold only when the search results are correct:

$$\begin{aligned} &\pi_C^x \bmod \mathbf{N} \\ &= g^{\prod_{i=1,2,3} \mathcal{P}(H(i,H(C_i))) \cdot \mathcal{P}(H(4,H(C_4)))} \\ &= g^{\prod_{i=1}^{4} \mathcal{P}(H(i,H(C_i)))} = \psi_C \end{aligned}$$

$$\begin{aligned}&(\pi_I)^y \bmod \mathbf{N}\\ &= g^{\prod_{i=1,2}\prod_{j=1}^{4}\mathcal{P}(H(i,\mathbf{V}[i][j]))\cdot\mathcal{P}(H(3,\mathbf{V}[3][1]))\cdot\prod_{j=2,3,4}\mathcal{P}(H(3,\mathbf{V}[3][j]))}\\ &= g^{\prod_{j=1}^{4}\mathcal{P}(H(3,\mathbf{V}[3][j]))\prod_{i=1,2}\prod_{j=1}^{4}\mathcal{P}(H(i,\mathbf{V}[i][j]))}\\ &= g^{\prod_{i=1}^{3}\prod_{j=1}^{4}\mathcal{P}(H(i,\mathbf{V}[i][j]))} = \psi_I.\end{aligned}$$

If all these checks succeed, she can recover document D_2 from ciphertext $C_2 \in \mathcal{R}_W$.

(Update Phase). As shown in Fig. 3b, document D_5, the rank-2 document for keyword W_2, is added. The user generates the update token $\mathcal{T}_* = \mathcal{T}_{\mathbf{add}} = ((5, add, C_5), \zeta, \tau_v, \tau_a)$ as follows: she first generates the ciphertext C_5 of document D_5 with SKE, and sets $\zeta = (\gamma_1, \gamma_2, \gamma_3)$, where γ_i is a random string of κ bits. Then, she sets $\tau_v = \beta$ where $\beta[1] = H(\mathbf{0}, 4, 5) \oplus S_{k_4}(W_2)$, $\beta[2] = H(4,5,3) \oplus S_{k_4}(W_2)$, $\beta[3] = H(5,3,\mathbf{0}) \oplus S_{k_4}(W_2)$, $\beta[4] = H(\mathbf{0},4,3) \oplus S_{k_4}(W_2)$, $\beta[5] = H(4,3,\mathbf{0}) \oplus S_{k_4}(W_2)$, and $\beta[6] = 2$. Furthermore, she chooses a random string r_3 of κ bits and sets $\tau_a = \lambda$ where $\lambda[1] = 3, \lambda[2] = G_{k_2}(W_2), \lambda[3] = r_3$, $\lambda[4] = \langle 5, 115, \mathbf{0}\rangle \oplus \mathbb{H}(P_{k_3}(W_2), \lambda[3])$, and $\lambda[5] = \sigma(2) = 3$.

Next, she computes y as:

$$\begin{aligned}y &= \frac{\prod_{j=1}^{3}\mathcal{P}(H(\lambda[5],\beta[j]))}{\prod_{j=4}^{5}\mathcal{P}(H(\lambda[5],\beta[j]))} \bmod \varphi(\mathbf{N})\\ &= \frac{\prod_{j=1}^{3}\mathcal{P}(H(3,\mathbf{V}'[3][j]))}{\prod_{j=1}^{2}\mathcal{P}(H(3,\mathbf{V}[3][j]))} \bmod \varphi(\mathbf{N}),\end{aligned}$$

Finally, she generates the new local evidence $\Psi' = (\psi'_C, \psi'_I)$ by setting $\psi'_C = (\psi_C)^{\mathcal{P}(H(5,H(C_5)))} \bmod \mathbf{N}$, and $\psi'_I = \psi_I^{\mathcal{P}(H(1,\gamma_1))\cdot\mathcal{P}(H(2,\gamma_2))\cdot y} \bmod \mathbf{N}$.

Given $\mathcal{T}_*$, the CSP generates the updated verifiable matrix $\mathbf{V}'$ and the updated ranked inverted indexes I' as shown in Figs. 4b and 2b, respectively. We leave the construction of document deletion to the readers.

5.4 Security Proof

We first provide the definitions of the leakage functions. The definitions below closely follow those in [13] except that rank information of each document is also exposed.

- $\mathcal{L}_1(\mathbf{d}) = (\#\mathrm{A}_s, \{\xi(W)\}_{W\in\mathbf{w}}, \{\xi(D)\}_{D\in\mathbf{d}}, \{||D||\}_{D\in\mathbf{d}})$. $\#\mathrm{A}_s$ is the size of search array, ξ is the identifier function outputting the identifier of a document D or keyword W, and $||D||$ is the bit length of document D.
- $\mathcal{L}_2(\mathbf{d}, W) = (\xi(W), \mathbf{ACCP}(W))$. $\mathbf{ACCP}(W)$ is the access pattern defined as the sequence of document identifiers $\mathbf{id}_W = (\mathrm{ID}_1, \ldots, \mathrm{ID}_{|\mathbf{d}_W|})$ where ID_j is the identifier of the rank-j document for keyword W.
- $\mathcal{L}_3(\mathbf{d}, D) = (\xi(D), \{\xi(W), \mathbf{apprs}(W)\}_{W\in\mathbf{w}}, ||D||)$ where $\mathbf{apprs}(W)$ is a bit set to 1 if W appears in at least one document in $\mathbf{d}$ and to 0 otherwise.
- $\mathcal{L}_4(\mathbf{d}, D) = (\xi(D))$.

Theorem 1. *If SKE is secure under adaptive chosen-plaintext attacks (adaptive CPA-secure), H is a collision-free hash function, and F, G, P, and S are pseudo-random, then* VDERS^0 *satisfies confidentiality and verifiability under the strong RSA assumption in the random oracle model.*

A proof is given in the appendix, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/TSC.2019.2922177.

6 Advanced VDERS Scheme

6.1 Main Idea

Our basic VDERS construction allows top-K queries in a dynamic and verifiable way, but falls short in the following aspects: (1) *Inefficiency in deletion operation.* It flags the deleted document but keeps corresponding nodes in the search array A_s. Therefore, it has no real meaning to delete a document and limits the number of the documents that can be added. (2) *Inefficiency in verification.* It requires to transmit the verifiable matrix $\mathbf{V}$ to the CSP for the generation of search proofs. The complexity of $\mathbf{V}$ is $O(nm\kappa)$, which may be unacceptable for a large-scale dataset.

To release the space of a deleted document, we define a deletion table T_j for each document $D_j \in \mathbf{d}$ as follows:

Definition 5 (Deletion table). *T_j is a dictionary of size $|\mathbf{w}_j|$. For each keyword $W \in \mathbf{w}_j$, $\mathrm{T}_j[W]$ stores $addr_s(\mathrm{N})$, where N is the node in the ranked linked list L_W corresponding to document D_j, and $addr_s(N)$ is N's location in the search array A_s.*

T_j stores the addresses in A_s for nodes that should be released if document D_j is removed. Let N_{-1} and N_{+1} denote nodes previous to and following N in L_W, respectively. Given T_j, node N can be deleted from L_W for each keyword $W \in \mathbf{w}_j$ by storing N as the last node of the free list $\mathrm{L}_{\mathbf{free}}$ and modifying the pointer of node N_{-1} to point to node N_{+1}. To reduce the communication cost, we encode the verifiable matrix $\mathbf{V}$ into accumulated values so that the CSP can generate search proofs correctly without $\mathbf{V}$.

Algorithm 4. VDERS*.Encryption

```
Input: Secret key $SK$, documents $\mathbf{d}$, and keywords $\mathbf{w}$
OutPut: Ranked inverted index I and ciphertexts $\mathbf{c}^*$
1: Execute lines 1-12 of Algorithm 1 and generate $\mathrm{I} = (\mathrm{T}_s, \mathrm{A}_s)$
2: for each document $D_j \in \mathbf{d}$ do
3:     Initialize $\mathrm{T}_j$ with an empty dictionary of size $|\mathbf{w}_j|$
4:     for each keyword $W \in \mathbf{w}_j$ do
5:         Locate node N corresponding to $D_j$ in $\mathrm{L}_W$
6:         Set $\mathrm{T}_j[W] \leftarrow addr_s(\mathrm{N})$
7:     Set $\widetilde{\mathrm{T}}_j \leftarrow$ SKE.Enc$(k_e, \mathrm{T}_j)$
8:     Set $C_j \leftarrow$ SKE.Enc$(k_e, D_j)$ and $\mathbf{c}^* = \mathbf{c}^* \cup (C_j, \widetilde{\mathrm{T}}_j)$
9: return $(\mathrm{I}, \mathbf{c}^*)$
```

6.2 Our Construction

The main differences between VDERS^0 and VDERS* lie in the following algorithms.

- $(\mathrm{I}, \mathbf{c}^*) \leftarrow$ Encryption$(SK, \mathbf{d}, \mathbf{w})$: As shown in Algorithm 4, the user first executes lines 1-12 of Algorithm 1 and generates the ranked inverted index $\mathrm{I} = (\mathrm{T}_s, \mathrm{A}_s)$. For each document $D_j \in \mathbf{d}$, she creates a deletion table T_j as defined in Definition 5, and separately encrypts T_j and D_j with algorithm SKE.Enc. The ciphertext collection $\mathbf{c}^*$ is set as $(C_j, \widetilde{\mathrm{T}}_j)_{j=1}^{n}$.

- $(\Phi^\star, \Psi^\star) \leftarrow \mathsf{AccGen}(params, SK, \mathbf{d}, \mathbf{w}, \mathbf{c}^\star)$: The user constructs an $m \times n$ verifiable matrix $\mathbf{V}$ as defined in Section 4.3. For each keyword $W \in \mathbf{w}$, she calculates $i = \sigma(\xi(W))$ and $\phi_i = \prod_{j=1}^{|\mathbf{d}_W|} \mathcal{P}(H(i, \mathbf{V}[i][j])) \bmod \varphi(\mathbf{N})$, and sets remote auxiliary information as $\Phi^\star = (\phi_1, \ldots, \phi_m)$. Note that the primary role of $\mathbf{V}$ is to record document ranking information. Since such information has been encoded into $\Phi^\star$, $\mathbf{V}$ can be removed from local storage. Finally, she sets a local evidence as $\Psi^\star = (\psi_C^\star, \psi_I^\star)$ where $\psi_C^\star = \psi_C$ is calculated with Eq. (2) and $\psi_I^\star$ is calculated with

$$\psi_I^\star = g^{\prod_{i=1}^{m} \phi_i} \bmod \mathbf{N}. \quad (16)$$

- $\mathcal{R}_W^\star \leftarrow \mathsf{Search}(\mathrm{I}, \mathcal{T}_W)$: The CSP executes lines 1-10 of Algorithm 2. Let $\mathbf{c}_W^K$ be the first K ciphertexts in $\mathbf{c}_W$. For top-K queries, it sets $\mathcal{R}_W^\star = (\mathbf{id}_W, \mathbf{s}_W, \mathbf{c}_W^K)$. For normal queries, it sets $\mathcal{R}_W^\star = (\mathbf{id}_W, \mathbf{s}_W, \mathbf{c}_W)$.
- $\Pi^\star \leftarrow \mathsf{GenProof}(params, \mathbf{c}^\star, \Phi^\star, \mathcal{T}_W, \mathcal{R}_W^\star)$: The CSP parses $\mathcal{T}_W$ as $(\tau_1, \tau_2, \tau_3, \tau_4, \tau_5)$ and sets the proof as $\Pi^\star = (\pi_C^\star, \pi_I^\star)$, where $\pi_C^\star = \pi_C$ is computed with Eq. (4) and $\pi_I^\star$ is computed based on $\Phi^\star = (\phi_1, \ldots, \phi_m)$:

$$\pi_I^\star = g^{\prod_{i=1, i\neq\tau_5}^{m} \phi_i} \bmod \mathbf{N}. \quad (17)$$

- $\{0,1\} \leftarrow \mathsf{Verify}(params, SK, \Psi^\star, \mathcal{T}_W, \mathcal{R}_W^\star, \Pi^\star)$: To verify a top-$K$ query, for $i \in [K]$, the user computes x_i with Eq. (6) and checks whether Eq. (7) holds. She parses $\mathcal{T}_W$ as $(\tau_1, \tau_2, \tau_3, \tau_4, \tau_5)$, and reconstructs $\mathbf{V}[\tau_5][1], \ldots, \mathbf{V}[\tau_5][|\mathbf{d}_W|]$ with Eq. (1). For $j \in [|\mathbf{d}_W|]$, she computes y_j with Eq. (18) and checks whether Eq. (19) holds:

$$y_j \leftarrow \mathcal{P}(H(\tau_5, \mathbf{V}[\tau_5][j])). \quad (18)$$

$$\psi_I^\star = (\pi_I^\star)^{\prod_{j=1}^{|\mathbf{d}_W|} y_j} \bmod \mathbf{N}. \quad (19)$$

If Eqs. (7) and (19) hold, she outputs 1, and 0 otherwise. For normal queries, parameter K in Eq. (7) is replaced by $|\mathbf{d}_W|$. If the output is 1, for each ciphertext $C_j \in \mathcal{R}_W^\star$, the user runs $\mathsf{SKE.Dec}$ to recover the document D_j.
- $(\mathcal{T}_*^\star, \Psi^{\star\prime}) \leftarrow \mathsf{UpdToken}(params, SK, D, \Psi^\star)$: To delete document D_j with $\xi(D) = j$, the user first retrieves $(C_j, \widetilde{\mathrm{T}}_j)$ from the CSP, and recovers T_j. Then, she sets the update token as $\mathcal{T}_* = \mathcal{T}_{\mathbf{del}} = ((j, \mathbf{delete}), \tau_v, \tau_a)$, where $\tau_v = (\nu_1, \ldots, \nu_{|\mathbf{w}_j|})$ and $\tau_a = (\lambda_1 \ldots, \lambda_{|\mathbf{w}_j|})$. For the ith keyword $W \in \mathbf{w}_j$, she computes ν_i and λ_i as follows:
  1) She performs normal queries to retrieve all documents containing keyword W. For node $\mathrm{N} \in \mathrm{L}_W$ that corresponds to document D_j, the CSP will return $(addr_s(\mathrm{N}_{-1}), addr_s(\mathrm{N}_{+1}))$ together with the search results $\mathcal{R}_W^\star$, where N_{-1} and N_{+1} are the nodes previous to and following N, and $addr_s(\mathrm{N}_{-1})$ and $addr_s(\mathrm{N}_{+1})$ are their locations in A_s, respectively. Then, she computes $addr_s(\mathrm{N}) \leftarrow \mathrm{T}_j[W]$, and sets $\lambda_i = (\lambda_i[1], \lambda_i[2], \lambda_i[3], \lambda_i[4], \lambda_i[5])$ as:

$$\lambda_i[1] = F_{k_1}(W), \lambda_i[2] = addr_s(\mathrm{N}_{-1}),$$
$$\lambda_i[3] = addr_s(\mathrm{N}), \lambda_i[4] = addr_s(\mathrm{N}_{+1}), \lambda_i[5] = \sigma(\xi(W)).$$

  2) After verifying the correctness of $\mathcal{R}_W^\star$, she reconstructs $\mathbf{V}[\lambda_i[5]][1], \ldots, \mathbf{V}[\lambda_i[5]][|\mathbf{d}_W|]$ with Eq. (1), and calculates $\phi_{\lambda_i[5]} = \prod_{j=1}^{|\mathbf{d}_W|} \mathcal{P}(H(\lambda_i[5], \mathbf{V}[\lambda_i[5]][j])) \bmod \varphi(\mathbf{N})$. Then, she determines the rank of D_j, denoted by J, and sets $\beta_i = (\beta_i[1], \beta_i[2], \beta_i[3], \beta_i[4], \beta_i[5], \beta_i[6])$ as:

$$\begin{aligned}
\beta_i[1] &= H(\overline{\mathrm{ID}}_{\mathrm{J}-2}, \overline{\mathrm{ID}}_{\mathrm{J}-1}, \overline{\mathrm{ID}}_{\mathrm{J}+1}) \oplus S_{k_4}(W),\\
\beta_i[2] &= H(\overline{\mathrm{ID}}_{\mathrm{J}-1}, \overline{\mathrm{ID}}_{\mathrm{J}+1}, \overline{\mathrm{ID}}_{\mathrm{J}+2}) \oplus S_{k_4}(W),\\
\beta_i[3] &= H(\overline{\mathrm{ID}}_{\mathrm{J}-2}, \overline{\mathrm{ID}}_{\mathrm{J}-1}, \overline{\mathrm{ID}}_{\mathrm{J}}) \oplus S_{k_4}(W),\\
\beta_i[4] &= H(\overline{\mathrm{ID}}_{\mathrm{J}-1}, \overline{\mathrm{ID}}_{\mathrm{J}}, \overline{\mathrm{ID}}_{\mathrm{J}+1}) \oplus S_{k_4}(W),\\
\beta_i[5] &= H(\overline{\mathrm{ID}}_{\mathrm{J}}, \overline{\mathrm{ID}}_{\mathrm{J}+1}, \overline{\mathrm{ID}}_{\mathrm{J}+2}) \oplus S_{k_4}(W),\\
\beta_i[6] &= \mathrm{J},
\end{aligned}$$

where $\overline{\mathrm{ID}}_j$ is the identifier of the rank-j document for keyword W before update, and $\overline{\mathrm{ID}}_{|\mathbf{d}_W|+1} = \overline{\mathrm{ID}}_0 = \mathbf{0}$. In the special case, if $\mathrm{J} = 1$, we set $\beta_i[1] = \beta_i[3] = \perp$; if $\mathrm{J} = |\mathbf{d}_W|$, we set $\beta_i[2] = \beta_i[5] = \perp$. Then, she calculates $\nu_i = \phi_{\lambda_i[5]} \times x_i \bmod \varphi(\mathbf{N})$, where x_i is calculated as follows:

$$x_i = \frac{\prod_{j=1}^{2} \mathcal{P}(H(\lambda_i[5], \beta_i[j]))}{\prod_{j=3}^{5} \mathcal{P}(H(\lambda_i[5], \beta_i[j]))} \bmod \varphi(\mathbf{N}). \quad (20)$$

Next, she generates the new local evidence as $\Psi^{\star\prime} = (\psi_C^{\star\prime}, \psi_I^{\star\prime})$ with Eqs. (21) and 22, respectively.

$$\psi_C^{\star\prime} = (\psi_C^\star)^{\frac{1}{\mathcal{P}(H(j, H(C_j)))} \bmod \varphi(\mathbf{N})} \bmod \mathbf{N}. \quad (21)$$

$$\psi_I^{\star\prime} = (\psi_I^\star)^{\prod_{i=1}^{|\mathbf{w}_j|} x_i} \bmod \mathbf{N}. \quad (22)$$

To add document D_{n+1} with $\xi(D) = n+1$, the user sets the update token as $\mathcal{T}_*^\star = \mathcal{T}_{\mathbf{add}} = ((n+1, \mathbf{add}, C_{n+1}, \widetilde{\mathrm{T}}_{n+1}), \tau_v, \tau_a)$, where C_{n+1} is output of $\mathsf{SKE.Enc}(k_e, D_{n+1})$, $\widetilde{\mathrm{T}}_{n+1}$ is output of $\mathsf{SKE.Enc}(k_e, \mathrm{T}_{n+1})$[4], $\tau_v = (\nu_1, \ldots, \nu_{|\mathbf{w}_{n+1}|})$ and $\tau_a = (\lambda_1, \ldots, \lambda_{|\mathbf{w}_{n+1}|})$. For the ith keyword $W \in \mathbf{w}_{n+1}$, she calculates ν_i and λ_i as follows:

1) She computes $\lambda_i = (\lambda_i[1], \ldots, \lambda_i[5])$ with Eq. (11).
2) As deletion operations, she reconstructs $\mathbf{V}[\lambda_i[5]][1], \ldots, \mathbf{V}[\lambda_i[5]][|\mathbf{d}_W|]$ with Eq. (1), and calculates $\phi_{\lambda_i[5]} = \prod_{j=1}^{|\mathbf{d}_W|} \mathcal{P}(H(\lambda_i[5], \mathbf{V}[\lambda_i[5]][j])) \bmod \varphi(\mathbf{N})$. Then, she calculates x_i with Eq. (23) and sets $\nu_i = \phi_{\lambda_i[5]} \times x_i \bmod \varphi(\mathbf{N})$:

$$x_i = \frac{\prod_{j=1}^{3} \mathcal{P}(H(\lambda_i[1], \beta_i[j]))}{\prod_{j=4}^{5} \mathcal{P}(H(\lambda_i[1], \beta_i[j]))} \bmod \varphi(\mathbf{N}), \quad (23)$$

where $\beta_i = (\beta_i[1], \ldots, \beta_i[6])$ is calculated with Eq. (12). Next, she generates the new local evidence as $\Psi^{\star\prime} = (\psi_C^{\star\prime}, \psi_I^{\star\prime})$ with Eqs. (13) and 22, respectively. In Eq. (22), where $|\mathbf{w}_j|$ is replaced by $|\mathbf{w}_{n+1}|$.

- $(\mathrm{I}', \mathbf{c}^{\star\prime}, \Phi^{\star\prime}) \leftarrow \mathsf{Update}(\mathrm{I}, \mathbf{c}^\star, \Phi^\star, \mathcal{T}_*^\star)$: If the update token $\mathcal{T}_*^\star = \mathcal{T}_{\mathbf{del}} = ((j, \mathbf{delete}), \tau_v, \tau_a)$, the CSP

4. To construct T_{n+1}, the user can either ask the CSP for the locations of free nodes or locally store the locations of all free nodes.

TABLE 3
Comparison of Computation Cost

Server	Initial	Search	Add	Delete
BASELINE1	$O(1)$	$O(n \cdot m)$	$O(1)$	$O(n)$
VDERS^0	$O(1)$	$O(n \cdot m)$	$O(1)$	$O(1)$
VDERS*	$O(1)$	$O(n + m)$	$O(1)$	$O(1)$
User	**Initial**	**Search**	**Add**	**Delete**
BASELINE1	$O(n \cdot m)$	$O(n + \lvert\mathbf{d}_W\rvert)$	$O(m)$	$O(1)$
VDERS^0	$O(n \cdot m)$	$O(K)$	$O(m)$	$O(1)$
VDERS*	$O(n + m \cdot \lvert\mathbf{d}_W\rvert_{\max})$	$O(K + \lvert\mathbf{d}_W\rvert)$	$O(\lvert\mathbf{w}_j\rvert \cdot \lvert\mathbf{d}_W\rvert_{\max})$	$O(\lvert\mathbf{w}_j\rvert \cdot \lvert\mathbf{d}_W\rvert_{\max})$

TABLE 4
Comparison of Communication Cost

	BASELINE1	BASELINE2	VDERS^0	VDERS*
Initial	$O(n \cdot m)$	$O((\varXi + m) \cdot \kappa)$	$O((\varXi + n \cdot m) \cdot \kappa)$	$O((\varXi + m) \cdot \kappa)$
Search	$O(n)$	$O(\kappa)$	$O(\kappa)$	$O(\kappa)$
Add	$O(m)$	$O(\lvert\mathbf{w}_j\rvert \cdot \kappa)$	$O((m + \lvert\mathbf{w}_j\rvert)) \cdot \kappa$	$O(\lvert\mathbf{w}_j\rvert \cdot \kappa)$
Delete	$O(1)$	$O(\kappa)$	$O(1)$	$O(\lvert\mathbf{w}_j\rvert \cdot \kappa)$

generates $\mathbf{c}^{\star\prime}$ by removing $(\widetilde{\mathrm{T}}_j, C_j)$ from $\mathbf{c}^\star$. It generates $\Phi^{\star\prime}$ by replacing $\phi_{\lambda_i[5]}$ with ν_i for $i \in [|\mathbf{w}_j|]$. To generate $\mathrm{I}' = (\mathrm{T}_s', \mathrm{A}_s')$, for the ith keyword $W \in \mathbf{w}_j$, it parses $\lambda_i = (\lambda_i[1], \lambda_i[2], \lambda_i[3], \lambda_i[4], \lambda_i[5])$ and performs as follows:

1) It computes $\varpi \leftarrow \mathrm{T}_s[\mathbf{free}]$ and finds the last free location ϖ in A_s.
2) It updates the free list to point to node N by setting $\mathrm{T}_s[\mathbf{free}] \leftarrow \lambda_i[3]$.
3) It releases the location of N by setting $\mathrm{A}_s[\lambda_i[3]] \leftarrow (\langle \mathbf{0}, \mathbf{0}, \varpi \rangle, \mathbf{0})$.
4) If $\lambda_i[2] \neq -1$, let $\mathrm{A}_s[\lambda_i[2]] = (v_{-1}, r_{-1})$. It updates N_{-1}'s next pointer by setting $\mathrm{A}_s[\lambda_i[2]] \leftarrow ((\mathbf{0}, \mathbf{0}, \lambda_i[3] \oplus \lambda_i[4] \oplus v_{-1}), r_{-1})$.
5) If N is the first node of L_W, it sets $\mathrm{T}_s[\lambda_i[1]] \leftarrow \lambda_i[4]$.

If $\mathcal{T}_* = \mathcal{T}_{\mathbf{add}} = ((n+1, \mathbf{add}, C_{n+1}, \widetilde{\mathrm{T}}_{n+1}), \tau_v, \tau_a)$, it generates the updated ranked inverted index I' in the same way as VDERS^0.Update. Next, it generates $\mathbf{c}^{\star\prime}$ by adding $(C_{n+1}, \widetilde{\mathrm{T}}_{n+1})$ into $\mathbf{c}^\star$ and generates $\Phi^{\star\prime}$ by replacing $\phi_{\lambda_i[5]}$ with ν_i for $i \in [|\mathbf{w}_{n+1}|]$.

Discussion. In both VDERS^0 and VDERS*, a user needs to search a set of keywords before generating an update token. This will incur additional costs in the update phase. To solve this problem, we can adopt the *lazy update* mechanism [30], which allows to postpone an update operation until a search operation happens. Specifically, to delete document D_j in VDERS*, the user sends $\mathcal{T}_{\mathbf{del}} = (j, \mathbf{del})$ to the CSP, which will send back $\widetilde{\mathrm{T}}_j$ and remove $(C_j, \widetilde{\mathrm{T}}_j)$. Once the user searches any keyword $W \in \mathbf{w}_j$, she can obtain related information for generating (λ_i, ν_i, x_i). With those information, the CSP can update (I, Ψ, Φ) accordingly. After searching all keywords in $\mathbf{w}_j$, she completes all components of the deletion token. Similarly, to add document D_{n+1} in VDERS*, the user keeps the document identifier and related score locally and sends $\mathcal{T}_{\mathbf{add}} = ((n+1, \mathbf{add}, C_{n+1}, \widetilde{\mathrm{T}}_{n+1}), \tau_a))$ to the CSP, which updates $\mathbf{c}^\star$ and I. Once searched any keyword in $\mathbf{w}_{n+1}$, she can obtain related information for modifying Ψ and Φ accordingly. After searching all keywords in $\mathbf{w}_{n+1}$, she completes all components of the add token.

6.3 Security Proof

We assume that the size of deletion table T_j is the same as that of document D_j. Therefore, VDERS* has the same leakages as VDERS^0 except that more information is leaked in the deletion operation. We provide the definition for $\mathcal{L}_4(\mathbf{d}, D)$ as follows:

- $\mathcal{L}_4(\mathbf{d}, D) = (\xi(D), \{addr_s(\mathrm{N}_{-1}), addr_s(\mathrm{N}), addr_s(\mathrm{N}_{+1}), \xi(W)\}_{W \in \mathbf{w}_{\xi(D)}})$ where $(addr_s(\mathrm{N}_{-1}), addr_s(\mathrm{N}), addr_s(\mathrm{N}_{+1}))$ is the information leaked by the deletion token.

Theorem 2. *If SKE is adaptive CPA-secure, H is a collision-free hash function, and F, G, P, and S are pseudo-random, then* VDERS* *satisfies confidentiality and verifiability under the strong RSA assumption in the random oracle model.*

A proof is given in the appendix, available in the online supplemental material.

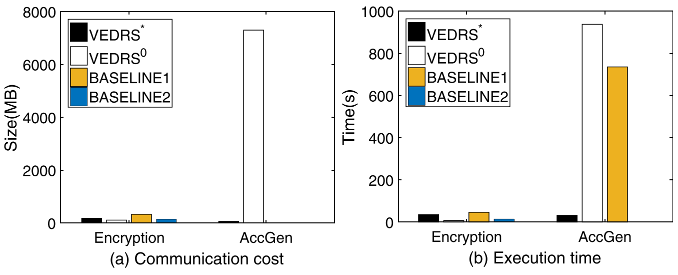


Fig. 5. Comparison in initial phase.

7 Evaluation

7.1 Performance Analysis

We compare the performance of VDERS^0, VDERS*, and the two schemes proposed in [23] and [13] (denoted by BASELINE1 and BASELINE2, respectively) in terms of computational and communication complexities. For computation cost, we only consider the most expensive operations, i.e., the calculation of RSA accumulators, and we analyze the incurred cost from the server side and the user side, respectively. Note that BASELINE2 is not verifiable, and thus, its cost is 0; it will not be listed. Given a collection of n documents with m keywords, the comparison results are shown in Table 3, where K is the parameter for a top-K search, $|\mathbf{d}_W|$ is the number of documents containing keyword W, and $|\mathbf{w}_j|$ is the total number of keywords contained in document D_j.

For communication cost, all schemes encrypt documents with SKE, and the sizes of the ciphertexts are the same. Furthermore, RSA accumulator generates a constant-size search proof. Therefore, we only consider the sizes of a secure index, a verifiable matrix, and a search/update token. The comparison results are shown in Table 4, where $\varXi = \sum |\mathbf{d}_W|$ is the total number of keyword-document pairs.

7.2 Parameter Setting

Experiments were conducted on a local machine running the Microsoft Windows 10 Ultimate operating system with an Inter Core i5 CPU running at 3.4GHz and 8GB memory. The programs were implemented in Java and compiled

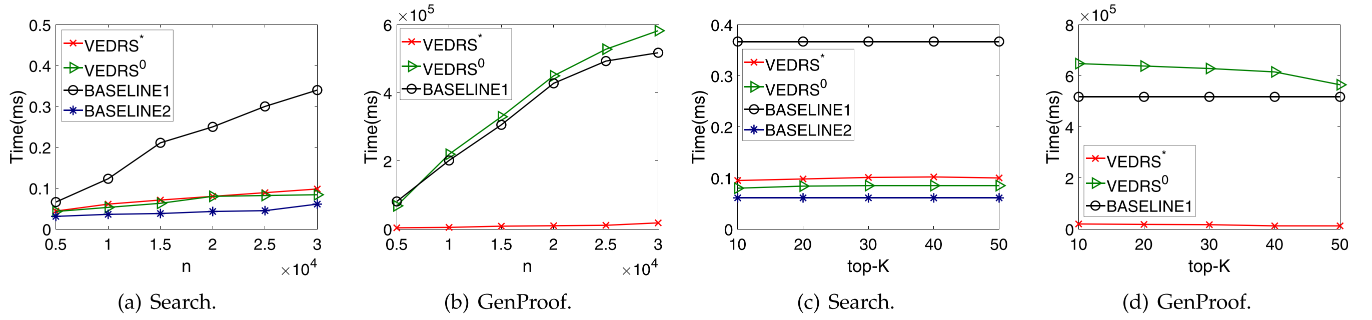


Fig. 6. Execution time in the search phase (server side). Given fixed $K = 20$, the execution time of (a) and (b) changes over the number of documents n; Given fixed $n = 30,000$, the execution time of (c) and (d) changes over the number of returned K documents.

using Eclipse 4.6.0. The cryptographic algorithms were implemented with JPBC library.

To validate the effectiveness and efficiency of the VDERS scheme in practice, we conducted a performance evaluation on a real e-mail data set, the Enron Email Data Set[5]. This data set has 30,000 plaintext e-mails with a total size of about 71.2 MB. The average size of each document is 2.43 KB. We chose $1 \sim 3$ keywords for each document after ranking them by frequency of occurrence, and got 7,922 keywords finally.

7.3 Experiment Results

We compared our VDERS constructions with BASELINE1 and BASELINE2 in terms of the communication cost and the execution time in the initial phase, search phase, and update phase. Since BASELINE2 is unverifiable, it lacks algorithms AccGen, GenProof, and Verify. Furthermore, SKE is employed to encrypt document contents in all schemes, and the computation and communication costs of generating a search token are very small. Therefore, we left out the time of encrypting documents in algorithm Encrption, and omitted the comparison of algorithm SrcToken.

Fig. 5 illustrates the comparison results in the initial phase. For the Encryption algorithm, BASELINE1 encrypts a string of length n for every keyword, and thus incurs the most computation and communication costs. VDERS^0 and VDERS* encrypt a ranked inverted index in the same way. Since VDERS* needs to encrypt a deletion table T_i in addition, and thus it incurs more computation and communication costs compared with VDERS^0. BASELINE2 encrypts an inverted index which includes a deletion table and a deletion array in addition, and thus its costs exceed VDERS^0. For the AccGen algorithm, VDERS^0 is the most expensive in both computation and communication costs because of the generation and transmission of a verifiable matrix ($m \times n \times \kappa$ bits in size); VDERS* incurs the minimal computation cost since the number of RSA accumulator calculations is reduced to $|\mathbf{d}_W|$ for each keyword; BASELINE1 incurs the minimal communication cost since there is no need to transmit auxiliary information to the CSP. Furthermore, the communication cost of VDERS* in the AccGen algorithm is also largely reduced since it only needs to transmit m integers to the CSP.

Fig. 6 shows the execution time at the server side in the search phase. In Fig. 6a, as the number of documents n grows, BASELINE1 needs to decrypt longer strings and return more documents, rendering the execution time to grow linearly; The search time in our VDERS constructions and BASELINE2 is impacted by the number of documents matching a keyword, and thus increases smoothly. In Fig. 6b, the execution time of algorithm GenProof in both VDERS^0 and BASELINE is mainly impacted by the number of documents n. However, in our advanced GenProof algorithm, the computation of Eq. (17) is based on Φ^*, and the execution time increases slightly as n increases. For example, the execution time in VDERS^0 and VDERS* increases from 68s to 582s, and from 4s to 18s, respectively, as n increases from 5,000 to 30,000. From Figs. 6c and 6d, we know that the execution time of algorithm Search is slightly impacted by parameter K, but the execution time of algorithm GenProof decreases smoothly as K increases. For example, as K increases from 10 to 50, the execution time of algorithm Search in VDERS^0 fluctuates around 0.08 ms, and the execution time of algorithm GenProof in VDERS* decreases from 20s to 12s.

Fig. 7 shows the execution time at the user side in the search phase. From this figure, we know that our VDERS

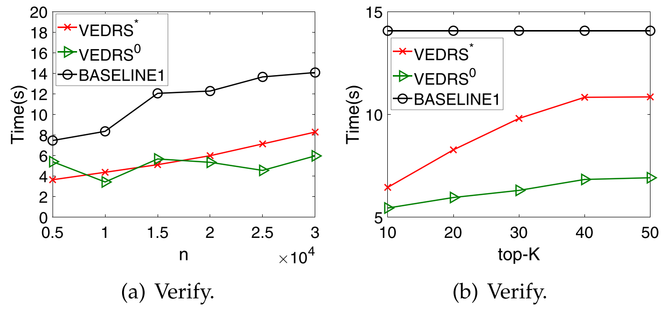


Fig. 7. Execution time in the search phase (user side). Given fixed $K = 20$, the time of (a) changes over the number of documents n; Given fixed $n = 30,000$, the time of (b) changes over parameter K.

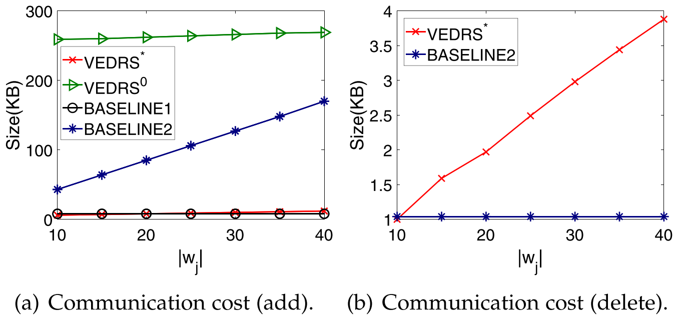
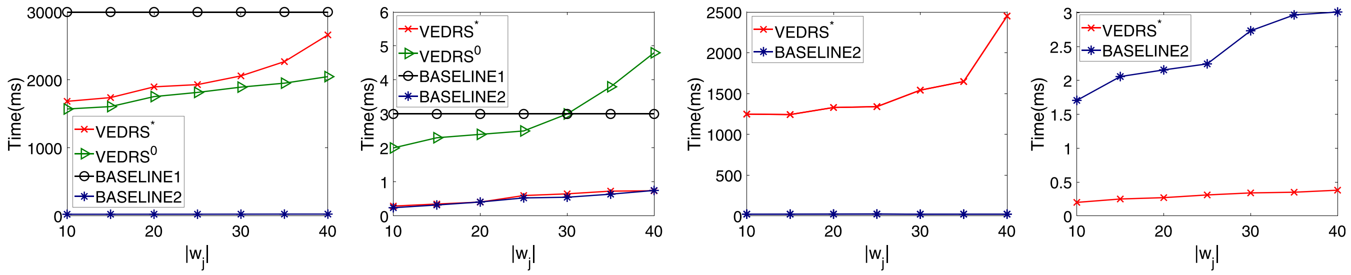


Fig. 8. Communication cost in the update phase. Given fixed $n = 30,000$, the communication costs of (a) and (b) change over the number of keywords in a document $|\mathbf{w}_j|$.

5. http://www.cs.cmu.edu/∼./enron/



(a) UpdToken (add). (b) Update (add). (c) UpdToken (delete). (d) Update (delete).

Fig. 9. Execution time in the update phase. Given fixed $n = 30,000$, the execution time of (a)-(d) changes over the number of keywords in a document $|\mathbf{w}_j|$. (a) and (c) show the computation time at the user side, and (b) and (d) show the computation time at the server side.

constructions have better performance compared with BASELINE1. To verify the search results, the number of RSA accumulator calculations in BASELINE1, VDERS^0, and VDERS* is mainly impacted by $|\mathbf{d}_W| + n$, K, and $|\mathbf{d}_W| + K$, respectively. Therefore, the execution time of BASELINE1 in Fig. 7a grows linearly as n increases, but it remains unchanged in Fig. 7b as K increases. On the contrary, the execution time of VDERS* in Fig. 7a changes slightly and VDERS^0 just fluctuates around a fixed value as n increases, but they both grow linearly in Fig. 7b as K increases.

Fig. 8 shows the communication costs in the update phase. In terms of adding a document, VDERS^0 incurs the largest communication cost because it needs to upload m extra random numbers to the CSP. BASELINE1 incurs the minimal cost because it only needs to upload m bits. From Fig. 8a, we know that the cost of BASELINE2 grows linearly, the costs of our VDERS constructions grow slightly, but the cost of BASELINE1 remains unchanged, as $|\mathbf{w}_j|$ increases. For example, as $|\mathbf{w}_j|$ increases from 10 to 40, the cost of BASELINE1 remains as 8 KB, but the costs of BASELINE2, VDERS^0, and VDERS*, increase from 43 KB to 170 KB, from 259 KB to 269 KB, and from 6 KB to 12 KB, respectively. In terms of deleting a document, the communication costs of BASELINE1 and VDERS* are negligibly small, and we only consider VDERS* and BASELINE2. From Fig. 8b, we know that the cost of BASELINE2 is independent of $|\mathbf{w}_j|$, but the cost of VDERS* badly grows with $|\mathbf{w}_j|$.

Fig. 9 shows the computation time in the update phase. In Fig. 9a, BASELINE1 needs to compute m times of RSA accumulators, and thus it incurs the maximum execution time. In Fig. 9b, VDERS^0 incurs the maximum execution time since it needs to update both the ranked inverted index and the verifiable matrix. In terms of deleting a document, we still only consider VDERS* and BASELINE2. In Fig. 9c, the time of BASELINE2 is independent of $|\mathbf{w}_j|$, but the time of VDERS* grows with $|\mathbf{w}_j|$. In Fig. 9d, both the time of VDERS* and BASELINE2 grows as $|\mathbf{w}_j|$ increases. Furthermore, BASELINE2 with a more complicated index structure incurs more execution time than VDERS*.

8 Conclusion

In this paper, we design a VDERS scheme to simultaneously support sublinear search time, efficient update and verification, and on-demand information retrieval in cloud computing environment. Experiment results demonstrate that our scheme is effective for verifying the correctness of top-K searches on a dynamic document collection. As part of our future work, we will try to design a forward secure VDERS scheme, where the update phase does not leak keyword information about a newly added document.

Acknowledgments

This work was supported in part by NSFC grants 61632009, 61872133, 61872130, and 61572181; NSF grants CNS 1824440, CNS 1828363, CNS 1757533, CNS 1629746, CNS 1651947, and CNS 1564128; the Guangdong Provincial Natural Science Foundation (No. 2017A030308006); and the High Level Talents Program of Higher Education in Guangdong Province (No. 2016ZJ01).

References

[1] Q. Liu, Y. Guo, J. Wu and, G. Wang, “Effective query grouping strategy in clouds,” *J. Comput. Sci. Technol.*, vol. 32, pp. 1231–1249, 2017.
[2] M. Xiao, J. Wu, L. Huang, R. Cheng and Y. Wang, “Online task assignment for crowdsensing in predictable mobile social networks,” *IEEE Trans. Mobile Comput.*, vol. 16, no. 8, pp. 2306–2320, Aug. 2017.
[3] Q. Liu, G. Wang, F. Li, S. Yang and J. Wu, “Preserving privacy with probabilistic indistinguishability in weighted social networks,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 5, pp. 1417–1429, May 2017.
[4] S. Zhang, Q. Liu, and Y. Lin, “Anonymizing popularity in online social network with full utility,” *Future Generation Comput. Syst.*, vol. 72, pp. 227–238, 2017.
[5] K. Xie, X. Li, X. Wang, J. Cao, G. Xie, J. Wen, D. Zhang and Z. Qin, “On-line anomaly detection with high accuracy,” *IEEE/ACM Trans. Netw.*, vol. 26, no. 3, pp. 1222–1235, Jun. 2018.
[6] X. Liu, Q. Liu, T. Peng and J. Wu, “Dynamic access policy in cloud-based personal health record (PHR) systems,” *Inf. Sci.*, vol. 379, pp. 62–81, 2017.
[7] G. S. Poh, J. -J. Chin, W. -C. Yau, K. -K. R. Choo, and M. S. Mohamad, “Searchable symmetric encryption: Designs and challenges,” *ACM Comput. Surveys*, vol. 50, 2017, Art. no. 40.
[8] Q. Liu, X. Nie, X. Liu, T. Peng, and J. Wu, “Verifiable ranked search over dynamic encrypted data in cloud computing,” *Proc. IEEE/ACM 25th Int. Symp. Quality Service*, 2017, pp. 1–6.
[9] T. Peng, Q. Liu, B. Hu, J. Liu, and J. Zhu, “Dynamic keyword search with hierarchical attributes in cloud computing,” *IEEE Access*, vol. 6, pp. 68948–68960, 2018.
[10] J. Camenisch and A. Lysyanskaya, “Dynamic accumulators and application to efficient revocation of anonymous credentials,” *Proc. Annu. Int. Cryptology Conf.*, 2002, pp. 61–76.
[11] D. Song, D. Wagner and A. Perrig, “Practical techniques for searches on encrypted data,” in *Proc. IEEE Symp. Security Privacy*, 2000, pp. 44–55.
[12] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky, “Searchable symmetric encryption: Improved definitions and efficient constructions,” in *Proc. 13th ACM Conf. Comput. Commun. Security*, 2006, pp. 79–88.
[13] S. Kamara, C. Papamanthou, and T. Roeder, “Dynamic searchable symmetric encryption,” in *Proc. ACM Conf. Comput. Commun. Security*, 2012, pp. 965–976.

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

[15] D. Cash, J. Jaeger, S. Jarecki, C. Jutla, H. Krawczyk, M. Rosu, and M. Steiner, "Dynamic searchable encryption in very-large databases: Data structures and implementation," in *Proc. Annu. Netw. Distrib. Syst. Security Symp.*, 2014, pp. 23–26.

[16] M. Naveed, M. Prabhakaran, and C. A. Gunter, "Dynamic searchable encryption via blind storage," in *Proc. IEEE Symp. Security Privacy*, 2014, pp. 639–654.

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

[18] E. Stefanov, C. Papamanthou, and E. Shi, "Practical dynamic searchable encryption with small leakage," in *Proc. Annu. Netw. Distrib. Syst. Security Symp.*, 2014, pp. 72–75.

[19] R. Bost, "$\Sigma o\phi o\varsigma$: Forward secure searchable encryption," in *Proc. Conf. Comput. Commun. Security*, 2016, pp. 1143–1154.

[20] X. Song, C. Dong, D. Yuan, Q. Xu, and M. Zhao, "Forward private searchable symmetric encryption with optimized I/O efficiency," *IEEE Trans. Dependable Secure Comput.*, Apr. 2018. (Early Access), doi:10.1109/TDSC.2018.2822294.

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

[22] Soleimanian, Azam and S. Khazaei, "Publicly verifiable searchable symmetric encryption based on efficient cryptographic components," *Designs Codes Cryptography*, vol. 87, pp. 123–147, 2019.

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

[24] W. Sun, X. Liu, W. Lou, Y. T. Hou, and H. Li, "Catch you if you lie to me: Efficient verifiable conjunctive keyword search over large dynamic encrypted cloud data," in *Proc. IEEE Conf. Comput. Commun.*, 2015, pp. 2110–2118.

[25] S. Jiang, X. Zhu, L. Guo, and J. Liu, "Publicly verifiable boolean query over outsourced encrypted data," *IEEE Trans. Cloud Comput.*, Mar. 2017. (Early Access), doi:10.1109/TCC.2017.2684811.

[26] J. Zhu, Q. Li, C. Wang, X. Yuan, Q. Wang and K. Ren, "Enabling generic, verifiable, and secure data search in cloud services," *IEEE Trans. Parallel Distrib. Syst.*, vol. 29, no. 8, pp. 1721–1735, Aug. 2018.

[27] N. Cao, C. Wang, M. Li, K. Ren and W. Lou, "Privacy-preserving multi-keyword ranked search over encrypted cloud data," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 1, pp. 222–233, Jan. 2014.

[28] W. K. Wong, D. W-l. Cheung, B. Kao and N. Mamoulis, "Secure kNN computation on encrypted databases," in *Proc ACM SIGMOD Int. Conf. Manag. Data*, 2009, pp. 139–152.

[29] W. Sun, B. Wang, N. Cao, M. Li, W. Lou, Y. T. Hou, and H. Li, "Verifiable privacy-preserving multi-keyword text search in the cloud supporting similarity-based ranking," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 11, pp. 3025–3035, Nov. 2014.

[30] C. Chen, X. Zhu, P. Shen, J. Hu, S. Guo, Z. Tari and A. Y. Zomaya, "An efficient privacy-preserving ranked keyword search method," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 4, pp. 951–963, Apr. 2016.

[31] W. Zhang, Y. Lin, S. Xiao, J. Wu and S. Zhou, "Privacy preserving ranked multi-keyword search for multiple data owners in cloud computing," *IEEE Trans. Comput.*, vol. 65, no. 5, pp. 1566–1577, May 2016.

[32] Z. Fu, K. Ren, J. Shu, X. Sun and F. Hua,"Enabling personalized search over encrypted outsourced data with efficiency improvement," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 9, pp. 2546–2559, Sep. 2016.

[33] C. Guo, X. Chen, Y. Jie, Z. Fu, M. Li, and B. Feng, "Dynamic multi-phrase ranked search over encrypted data with symmetric searchable encryption," *IEEE Trans. Serv. Comput.*, Oct. 2017. (Early Access), doi:10.1109/TSC.2017.2768045.

[34] Q. Liu, G. Wang, X. Liu, T. Peng, and J. Wu, "Achieving reliable and secure services in cloud computing environments," *Comput. Elect. Eng.*, vol. 59, pp. 153–164, 2017.

[35] F. Kerschbaum, "Frequency-hiding order-preserving encryption," in *Proc. 22nd ACM SIGSAC Conf. Comput. Commun. Security*, 2015, pp. 656–667.

Qin Liu received the BSc degree in computer science from Hunan Normal University, China, in 2004, the MSc and PhD degrees in computer science from Central South University, China, in 2007 and 2012. She has been a visiting student at Temple University, Philadelphia, PA. Her research interests include security and privacy issues in cloud computing. Now, she is an associate professor with the College of Computer Science and Electronic Engineering at Hunan University, China. She is a member of the IEEE.

Yue Tian received the BSc degree in E-Commerce from Nanchang University, Nanchang, Jiangxi Province, China, in 2017. Now, she is working toward the postgraduate degree majoring in computer science with the College of Computer Science and Electronic Engineering, Hunan University, Changsha, Hunan Province, China. Her research interests include security and privacy issues in cloud computing. She is a student member of the IEEE.

Jie Wu is the director of the Center for Networked Computing and Laura H. Carnell professor at Temple University. He also serves as the director of International Affairs at College of Science and Technology. He served as chair of Department of Computer and Information Sciences from the summer of 2009 to the summer of 2016 and associate vice provost for International Affairs from the fall of 2015 to the summer of 2017. Prior to joining Temple University, he was a program director at the National Science Foundation and was a distinguished professor at Florida Atlantic University. His current research interests include mobile computing and wireless networks, cloud and green computing, and network trust and security. He serves on several editorial boards, including the *IEEE Transactions on Mobile Computing*, the *IEEE Transactions on Service Computing*, the *Journal of Parallel and Distributed Computing*, and the *Journal of Computer Science and Technology*. He was an IEEE Computer Society distinguished visitor, ACM Distinguished Speaker, and chair for the IEEE Technical Committee on Distributed Processing. He is a fellow of the AAAS and a fellow of the IEEE. He is the recipient of the 2011 China Computer Federation Overseas Outstanding Achievement Award.

Tao Peng received the BSc degree in computer science from Xiangtan University China, in 2004, the MSc degree in circuits and systems from Hunan Normal University, China, in 2007, and the PhD degree in computer science from Central South University, China, in 2017. Now, she is an assistant professor of School of Computer Science and Cyber Engineering, Guangzhou University, China. Her research interests include network and information security issues. She is a member of the IEEE.

Guojun Wang received the BS degree in geophysics, and the MS and PhD degrees in computer science from Central South University, China, in 1992, 1996, and 2002. He is the Pearl River Scholar Professor of Guangdong Province at School of Computer Science and Cyber Engineering, Guangzhou University, China. He has been an adjunct professor at Temple University, Philadelphia, PA; a visiting scholar at Florida Atlantic University, Boca Raton, FL; a visiting researcher at the University of Aizu, Japan; and a research fellow at the Hong Kong Polytechnic University. His research interests include network and information security, and cloud computing. He is a distinguished member of CCF, and a member of the IEEE, ACM, and IEICE.