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 datasets. 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, 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\textsuperscript{+}, 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
first provide a basic construction, denoted by VDERS\textsuperscript{0}, which enables verifiable document insertion operations. Then, we provide an advanced construction, denoted by VDERS\textsuperscript{3}, 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 VSRSSE 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 using 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.
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 $\Psi$ and remote auxiliary information $\Phi$ for verifiable searches. After uploading $(c, I, \Phi)$ to the cloud, she can perform updates on ciphertexts with an update token $T_u$ and retrieve documents on demand with a search token $T_w$ in a verifiable way. On receiving the search results and a search proof $(R_w, II)$ 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 specified 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, ..., 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 except search pattern, access pattern, and document ranks.

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 $N = pq$ and $\varphi(N) = (p - 1)(q - 1)$, and let $G$ be a cyclic group of size $(p - 1)(q - 1)/4$ where $g$ is a generator of $G$. For a set of elements $E = \{y_1, \ldots, y_k\}$ 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 $P(y_i)$, such that $y_i = f(x_i)$. The accumulator is calculated as $Acc(E) = g^{\prod_{i=1}^k P(y_i)} \pmod N$.
- For any subset $E' \subseteq E$, a witness $\pi = g^{\prod_{y_i \in E' \setminus E} P(y_i)} \pmod N$ is produced.
- The subset test is carried out by checking $Acc(E) = \prod_{y_i \in E} P(y_i) \pmod N$.

**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 $N = pq$ and a random element $y \in \mathbb{Z}_N$, it is hard to find $x$ and $e > 1$ such that $y = x^e \pmod N$.

**Proposition 2.** Given $N, g, f$, and $E = \{y_1, \ldots, y_k\}$, it is hard to find $y \notin E$ and $\pi$ such that $\pi^{P(y)} = Acc(E) \pmod N$ 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 $0$ be a string of $0$s with length $\kappa$. For $t \in \mathbb{N}$, notation $|t|$ is used to denote the set of integers $\{1, \ldots, t\}$. The set of all
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 \( v \), its \( i \)th element is denoted by \( v[i] \) or \( V_i \) and its total number of elements by \( |v| \). Given a matrix \( M \), the element in its \( i \)th row and \( j \)th column is denoted by \( M[i][j] \). 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 \( (s_1 \ldots s_t) \).

The data set consists of a sequence of \( n \) documents \( d = (D_1, \ldots, D_n) \), where the \( j \)th document \( D_j \), associated with identifier \( j \), contains a sequence of keywords \( w_i \), for \( j \in [n] \). The ciphertext collection is denoted by \( c = (C_1, \ldots, C_n) \), where \( C_j \) is the ciphertext of document \( D_j \) for \( j \in [n] \). The universal distinct keywords extracted from \( d \) are denoted by \( w = w_1 \cup \cdots \cup w_n = (W_1, \ldots, W_m) \). The \( i \)th keyword \( W_i \in w \), associated with identifier \( i \), is contained in a sequence of documents \( d_{w[i]} \), 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 \( \xi \) be a function outputting the identifier of a document or a keyword. For document \( D \in d \) and keyword \( W \in w \), 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 \( W \in w \) are randomly stored in a search array \( A \), 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 \( O(|d_{w[i]}|) \), 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 \( (\log |A_s| + 1)\)-length string of 1s. For each keyword \( W \in w \), 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 \( |d_W| \) nodes \((N_1, \ldots, N_{|d_W|})\) and each node is defined as \( N_j = (id_j, \xi_j, addr_s(N_{j+1})) \), where \( id_j \) is identifier of the \( j \)th document in \( d \), \( \xi_j \) is the score of the \( j \)th document in \( d_W \), and \( addr_s(N_{j+1}) \) is the address of node \( N_{j+1} \) in a search array \( A_s \). In the special case, \( addr_s(N_{|d_W|+1}) = -1 \).

As for data structures, the search array \( A_s \) is an array with \( A_s[i] \) denoting the value stored at location \( i \) of \( A_s \) and \( A_s[i] \rightarrow v \) 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, \( T_s[k] \rightarrow v \) denotes storing the value \( v \) under key \( k \) in \( T_s \). Let \( \#A_s \) and \( \#T_s \) denote the size of \( A_s \) and \( T_s \), respectively. Similar to [13], we set \( \#T_s = m + 1 \), where the first \( m \) entries correspond to keywords in \( w \) and the last entry points to an unused entry in \( A_s \); and we set \( \#A_s = |c|/8 + z \) to protect statistical information about \( d \), where \( |c| \) 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 \( H : \{0,1\}^* \rightarrow \{0,1\}^m \) be a collision-free hash function, let \( \sigma \) be a pseudo-random permutation (PRP) on \( |m| \), and let \( S : \{0,1\}^k \times \{0,1\}^k \rightarrow \{0,1\}^k \) be a pseudo-random function (PRF) with key \( k_i \). The verifiable matrix \( V \) is an \( m \times n \) matrix defined as follows:

\[
V[i][j] = \begin{cases} 
H(ID_{j-1}, ID_j, ID_{j+1}) \oplus S_{k_i}(W), & j \in [|d_W|] \\
\gamma_{ij}, & \text{otherwise}
\end{cases}
\]

where \( i = \sigma(\xi(W)) \), \( ID_j \) denotes the identifier of the rank-\( j \) document for keyword \( W \), and \( \gamma_{ij} \) is a random string of length \( \kappa \) stored at \( V[i][j] \). Therefore, \( V[i][j] \) records the information of the rank-\((j-1)\), rank-\( j \), and rank-\((j+1) \) documents for keyword \( W \). In the special case, we have \( ID_0 = ID_{|d_W|+1} = 0 \).

### 4.4 Scheme Definition

The VDERS scheme consists of the following algorithms:

- \((\text{params}, SK) \rightarrow \text{GenKey}(1^n)\): It takes a security parameter \( \kappa \) as input and outputs public parameters \( \text{params} \) and a secret key \( SK \).
- \((I, c) \rightarrow \text{Encryption}(SK, d, w)\): It takes a secret key \( SK \), a sequence of documents \( d \), and a sequence of keywords \( w \) as inputs, and outputs a ranked inverted index \( I \) and a sequence of ciphertexts \( c \).
- \((\Phi, \Psi) \rightarrow \text{AccGen}(\text{params}, SK, d, w, c)\): It takes public parameters \( \text{params} \), a secret key \( SK \), a sequence of documents \( d \), a sequence of keywords \( w \), and a sequence of ciphertexts \( c \) as inputs. It outputs remote auxiliary information \( \Phi \) and a local evidence \( \Psi \).
- \(T_W \rightarrow \text{SrcToken}(SK, W)\): It takes a secret key \( SK \) and a keyword \( W \) as inputs to generate a search token \( T_W \).
- \(R_W \rightarrow \text{Search}(I, T_W)\): It takes a search token \( T_W \) and an index \( I \) as inputs to output search results \( R_W \).

1. For the single keyword search, we choose the term frequency (TF) as the relevance score, which is calculated by \( TF = \frac{d_w}{D} \), where \( D \) 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].
P • Π ← GenProof(\text{params, c, } \Phi, T_W, R_W)\): It takes public parameters \text{params}, a sequence of ciphertexts c, remote auxiliary information \Phi, a search token \(T_W\), and search results \(R_W\) as inputs to generate a search proof \(\Pi\).

• \{0, 1\} ← Verify(\text{params, SK, } \Psi, T_W, R_W, \Pi)\): It takes public parameters \text{params}, a secret key SK, a local evidence \(\Psi\), a search token \(T_W\), search results \(R_W\), and a search proof \(\Pi\) as inputs. It outputs 1 if verification is successful, and outputs 0 otherwise. If the output is 1, for each ciphertext in \(R_W\), it takes a secret key SK as input to recover plaintext.

• \(\langle T'_i, \Psi' \rangle ← \text{UpdToken}(\text{params, } D, \Psi)\): It takes public parameters \text{params}, a secret key SK, a document \(D\), and a local evidence \(\Psi\) as inputs to generate an update token \(T'_i\) and a new local evidence \(\Psi'\), where \(\Psi' = \{\text{add, del}\}\).

• \(\langle I', c', \Phi' \rangle ← \text{Update}(I, c, \Phi, T_i)\): It takes a ranked inverted index \(I\), a ciphertext collection \(c\), remote auxiliary information \(\Phi\), and an update token \(T_i\) as inputs. It generates a new ranked inverted index \(I'\), a new ciphertext collection \(c'\), and a new remote auxiliary information \(\Phi'\).

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 \(\langle I, c \rangle\) 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 \(L_1(d), L_2(d, W), L_3(d, D),\) and \(L_4(d, D)\) be stateful leakage algorithms. We consider the following probabilistic experiments.

• \textbf{Real}_{Adv}(\kappa)\): the challenger runs GenKey(\(1^\kappa\)) to generate a secret key SK. Adv outputs d and receives (I, c) from the challenger so that \(\langle I, c \rangle ← \text{Encryption}(SK, d, w)\). Adv makes a polynomial number of adaptive queries, and for each query, it receives from the challenger either a search token \(T_W\) such that \(T_W ← \text{SrcToken}(SK, W)\) or an update token \(T_i\), such that \(\langle T'_i, \Psi' \rangle ← \text{UpdToken}(\text{params, } D, \Psi)\). Finally, Adv returns a bit b that is output by the experiment.

• \textbf{Ideal}_{Adv, Sim}(\kappa)\): Adv outputs d. Given \(L_1(d), Sim\) generates and sends a pair (I, c) to Adv. Adv makes a polynomial number of adaptive queries. Given either \(L_2(d, W), L_3(d, D),\) or \(L_4(d, D)\), Sim returns an appropriate token and in the case of an add operation, a ciphertext 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 \(|\text{Pr}[\text{Real}_{Adv}(\kappa) = 1] - \text{Pr}[\text{Ideal}_{Adv, Sim}(\kappa) = 1]|\) 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 \(\langle d, w, 1 \rangle\) and search queries Adv can return \(\langle R_W, \Pi \rangle\) for some query such that \(R_W ≠ R_W\) and the user accepts the result.

Definition 4 (Verifiability of VDERS). VDERS is verifiable if for any PPT adversary Adv, \(\text{Pr}[\text{Adv wins}]\) is negligible for any \(\langle d, w, 1 \rangle\) 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 V to record the document ranking information for each keyword, and calculates the remote auxiliary information \(\Phi\) and a local evidence \(\Psi\) by encoding the elements of V into RSA accumulator.

5.2 Our Construction

Let \(\mathcal{F} = \{f : \{0, 1\}^n → \{0, 1\}^s\}\) be a family of two-universal hash functions and let \(\text{SKE} = (\text{Gen}, \text{Enc}, \text{Dec})\) be a symmetric-key encryption (SKE) scheme, where \(\text{Gen}\) is a key generation algorithm, \(\text{Enc}\) is an encryption algorithm, and \(\text{Dec}\) is a decryption algorithm. Let \(h : \{0, 1\}^n → \{0, 1\}^s\) be a random oracle and let \(H : \{0, 1\}^n → \{0, 1\}^s\) be a collision-free hash function. To compute \(H(x_1, ..., x_n)\), we first transform each input \(x_i\) into a bit string \(\overline{x_i}\) and then take the concatenation of \(\overline{x_1}, ..., \overline{x_n}\) as the inputs of \(H\). Let \(F : \{0, 1\}^{n(n+1)/2} → \{0, 1\}^{b} \) be PRFs.

VDERS is constructed as follows:

• \(\langle \text{params, ST} \rangle ← \text{GenKey}(1^n)\): Given \(\langle N = pq, g \rangle\) as shown in Section 3.3, the user randomly chooses a hash function \(f \in \mathcal{F}\) and four \(k\)-bit strings \(k_1, k_2, k_3, k_4\) as keys of PRFs. Then, she runs algorithm \(\text{SKE.Gen}\) to generate the key \(k_0\) of SKE, chooses a PRP \(\sigma\) on \([m]\), and sets the public parameters \(\text{params} = \langle N, g, f \rangle\) and the secret key as \(\text{SK} = \langle p, q, \sigma, k_1, k_2, k_3, k_4 \rangle\).

• \(\langle I, c \rangle ← \text{Encryption}(SK, d, w)\): Assume that \(-1\) denotes a \((\log \#A + 1)\)-length string of 1s and that

3. A prime \(x\) is chosen randomly from \(f^{-1}\). A secret key \(k_s \in \{0, 1\}^s\) shared between the user and the CSP is used as the randomness when computing \(x\), so that they obtain the same outputs from \(f^{-1}\).
Algorithm 1. VDERS\textsuperscript{S} Encryption

Input: Secret key $SK$, documents $d$, and keywords $w$

OutPut: Ranked inverted index $I$ and ciphertexts $c$

1: Initialize the search table $T_s$ with an empty dictionary of size $m + 1$ and initialize the search array $A_s$ with an empty array of size $|c|/8 + z$

2: for each keyword $W \in w$ do

3: Create a ranked linked list $L_W = \{N_1, \ldots, N_{d_W}\}$ by randomly choosing $|d_W|$ unused entries in $A_s$, where $N_i = (i, d, addr(s(N_{i+1}))$ is defined in Definition 1.

4: for $i \in [d_W]$ do

5: Choose a $k$-bit random string $r_i$ and set $A_s[addr(s(N_i))] \leftarrow (N_i \oplus \oplus(P_{3s}(W), r_i), r_i)$

6: Encrypt the address of the head node $N_1$ and set $T_s[F_s(W)] \leftarrow addr(s(N_1) \oplus G_{s2}(W))$

7: Create a free list $F_{free} = (F_1, \ldots, F_z)$ by randomly choosing $z$ unused entries in $A_s$, where $F_i = (0, 0, addr(s(F_{i-1})))$ and $addr(s(F_0)) = -1$

8: for $i = z$ to $1$ do

9: Set $A_s[addr(s(F_i))] \leftarrow (F_i, 0)$

10: Store the address of the tail node $F_z$ in $T_s$ by setting $T_s[free] \leftarrow addr(s(F_z))$

11: Fill the remaining entries of $A_s$ with random strings

12: Set $I = (T_s, A_s)$

13: for $j \in [n]$ do

14: Set $C_j = SKE(\text{Enc}(k_j, D_j))$ and $c = c \cup C_j$

15: return $(I, c)$

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 $W \in w$, 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 $F_{free}$ 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 hide statistical information about the document collection (Lines 11-12). Finally, she generates a sequence of ciphertexts $c$ and returns $(I, c)$ as the outputs of Algorithm 1 (Lines 13-15).

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

$$\psi_C = g^{\prod_{i=1}^{n}P(H(i,H(C_i)))} \mod N,$$

$$\psi_I = g^{\prod_{i=1}^{n}P(H(i,V[i]))} \mod N. \quad (2)$$

- $T_W \leftarrow \text{SrcToken}(SK, W)$: To retrieve the top-$K$ documents, the user generates a search token $T_W = (F_{h1}(W), G_{h2}(W), P_{b1}(W), K, \sigma(\xi(W)))$ for keyword $W \in w$. If the user wants to retrieve all documents containing keyword $W$, $K$ in $T_W$ is replaced by notation $\ast$.

- $R_W \leftarrow \text{Search}(I, T_W)$: For top-$K$ queries, the CSP runs Algorithm 2 to output search results $R_W = (\text{id}_{W}^{K+1}, s_{W}^{K+1}, c_{W}^{K+1})$, where $\text{id}_{W}^{j}$, $s_{W}^{j}$, and $c_{W}^{j}$ denote the first $j$ identifiers, scores, and ciphertexts in $\text{id}_{W}$, $s_{W}$, and $c_{W}$, respectively. For normal queries, search results $R_W$ are set as $(\text{id}_{W}, s_{W}, c_{W})$.

In Algorithm 2, the CSP parses $T_W$ as $(t_1, t_2, t_3, t_4, t_5)$, and returns $\emptyset$ if $t_1$ 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 $R_W$ (Lines 9-17).

Algorithm 2. VDERS\textsuperscript{S} Search

Input: Ranked inverted index $I$ and search token $T_W$

OutPut: Search results $R_W$

1: Parse $T_W$ as $(t_1, t_2, t_3, t_4, t_5)$

2: if $t_1$ is not in $T_s$ then

3: return $\emptyset$

4: Compute $T_s[t_1] \oplus t_2$ and recover $addr(s(N_1))$ the address of the head node $N_1$ of $L_W$ in $A_s$

5: repeat

6: Obtain the encrypted node $N_i$ stored at $A_s[addr(s(N_i))]$ and parse $N_i$ as $(w, r_i)$

7: Compute $v_i \oplus \oplus(t_3, r_i)$ and recover node $N_i = (i, d, addr(s(N_{i+1})))$

8: until $addr(s(N_{i+1})) = -1$

9: Sort documents according to their scores

10: Let $I_D$ and $e$ be the identifier and score of the rank-$j$ document for $W$, respectively

11: if $t_4 \neq \emptyset$ then

12: Set $\text{id}_{W}^{K+1} = (\text{id}_1, \ldots, \text{id}_{K+1}), s_{W}^{K+1} = (e_1, \ldots, e_{K+1})$, and $c_{W}^{K+1} = (\text{C}_{d_1}, \ldots, \text{C}_{d_{K+1}})$

13: Set $R_W \leftarrow (\text{id}_{W}^{K+1}, s_{W}^{K+1}, c_{W}^{K+1})$

14: else

15: Set $\text{id}_{W} = (\text{id}_1, \ldots, \text{id}_{d_W}), s_{W} = (e_1, \ldots, e_{d_W})$, and $c_{W} = (\text{C}_{d_1}, \ldots, \text{C}_{d_{d_W}})$

16: Set $R_W \leftarrow (\text{id}_{W}, s_{W}, c_{W})$

17: return $R_W$

- $\Pi \leftarrow \text{GenProof}(\text{params}, c, \Phi, T_W, R_W)$: For top-$K$ queries, the CSP parses $T_W$ as $(t_1, t_2, t_3, t_4, t_5)$, and sets search proof $\Pi = (\Pi_C, \Pi_I)$ with

$$\Pi_C = g^{\prod_{i=1}^{n}P(H(i,H(C_i)))} \mod N, \quad (4)$$

$$\Pi_I = g^{\prod_{i=1}^{n}P(H(i,V[i]))} \mod N, \quad (5)$$

where $\phi_i = \prod_{j=1}^{n}P(H(i, V[i][j]))$. For normal queries, parameter $K$ in both Eqs. (4) and (5) is replaced by $d_W$.

- $\{0, \text{!} \} \leftarrow \text{Verify}(\text{params}, SK, \Psi, T_W, R_W, \Pi)$: Let $I_D$ 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.

$$x_i = g^{\prod_{j=1}^{n}P(H(i, V[j][j]))} \mod N, \quad (6)$$

$$x_i = y \implies x_i \equiv y \mod N \implies 1 \implies x_i \equiv y \mod N. \quad (7)$$
where \( k \in [K] \), she computes \( y_j \) with Eq. (8) and checks whether Eq. (9) holds.

\[
y_j \leftarrow \mathcal{P}(H(t_5, V_{t_5}[j])).
\]

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 \(|d_W|\). If the output is 1, for each ciphertext \( C_j \in \mathcal{R}_W \), the user runs SKE.Dec to recover document \( D_j \).

- \( (T_*, \Psi' \leftarrow \text{UpdateToken}(params, SK, D, \Psi)) \): To delete document \( D \) with \( \xi(D) = j \), the user sets the update token as \( T_\text{del} = (j, \text{delete}, C') \), where \( C' \) is output of SKE.Enc \((k_v, 0)\). Then, she retrieves \( C_i \) from the CSP and generates a new local evidence as \( \Psi' = (\psi'_C, \psi'_J) \), where \( \psi'_J = \psi_J \) and \( \psi'_C \) is computed with

\[
\psi'_C = (\psi_C)^{\mathcal{P}(H(V_{t_5}[j])) \mod \mathcal{N}}} \mod \mathcal{N}.
\]

To add document \( D \) with \( \xi(D) = n + 1 \), the user sets the update token as \( T_\text{add} = ((n + 1, \text{add}, C_{n+1}), \zeta, \tau_0, \tau_2) \), where \( C_{n+1} \) is the output of SKE.Enc \((k_v, D_{n+1})\), \( \zeta = (\gamma_1, \ldots, \gamma_m) \) are a sequence of \( k \)-bit random strings, \( \tau_0 = (\beta_1, \ldots, \beta_{|w_{i+1}|}) \) and \( \tau_2 = (\lambda_1, \ldots, \lambda_{|w_{i+1}|}) \). For the \( i \)-th keyword \( W \in w_{i+1} \), she computes \( \beta_i \) and \( \lambda_i \) as follows:

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

\[
\lambda_i[1] = F_{k_5}(W), \quad \lambda_i[2] = G_{k_2}(W), \quad \lambda_i[3] = r_i, \\
\lambda_i[4] = (n + 1, \epsilon_i, 0) \oplus H(P_{k_2}(W), \lambda_i[3]), \\
\lambda_i[5] = \sigma(\xi(W)).
\]

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, \( \epsilon_i \), with those of documents in \( \mathcal{R}_W \), and sets \( \beta_j = (\beta_j[1], \beta_j[2], \beta_j[3], \beta_j[4], \beta_j[5], \beta_j[6]) \) as:

\[
\beta_i[1] = H(I\mathcal{D}_{j-2}, I\mathcal{D}_{j-1}, n + 1) \oplus S_{k_6}(W), \\
\beta_i[2] = H(I\mathcal{D}_{j-1}, n + 1, I\mathcal{D}_j) \oplus S_{k_4}(W), \\
\beta_i[3] = H(n + 1, I\mathcal{D}_j, I\mathcal{D}_{j+1}) \oplus S_{k_4}(W), \\
\beta_i[4] = H(I\mathcal{D}_{j-2}, I\mathcal{D}_{j-1}, I\mathcal{D}_j) \oplus S_{k_4}(W), \\
\beta_i[5] = H(I\mathcal{D}_{j-1}, I\mathcal{D}_j, I\mathcal{D}_{j+1}) \oplus S_{k_4}(W), \\
\beta_i[6] = J,
\]

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

\[
\psi'_C = (\psi_C)^{\mathcal{P}(H(n+1, H(C_{n+1})) \mod \mathcal{N}}} \mod \mathcal{N}.
\]

For \( i \in [|w_{n+1}|] \), she computes \( y_i \) with:

\[
y_i = \prod_{i=1}^{j} \mathcal{P}(H(\lambda_i[5], \beta_i[j]) \mod \mathcal{N}).
\]

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

\[
\psi'_J = (\psi_J)^{\prod_{i=1}^{j} \mathcal{P}(H(\lambda_i[5], \beta_i[j])) \mod \mathcal{N}},
\]

where \( \chi = [m]/\{\lambda_1[5], \ldots, \lambda_{|w_{i+1}|}[5]\} \) is a set of elements and \( \epsilon_i \in \xi \) is a \( k \)-bit random string.
For the addition operation, the CSP first generates $c'$ by adding $C_{n+1}$ into $c$ (Lines 4-6). To generate $I = (T', \lambda')$, it executes lines 7-13 of Algorithm 3 as follows: For each keyword $W \in w_{n+1}$, it stores the new node at the last free location $\varpi$ in $A_s$ and sets it as the first node of $L_W$, and then it updates $T_s[\lambda_i[1]]$ to point to $\varpi$ and updates $T_s[\text{free}]$ to point to the second last free location $\varpi_{-1}$ in $A_s$. Finally, it generates $\Phi' = V'$ and returns $(I', c', \Phi')$ (Lines 14-27).

Algorithm 3. VDERS$^0$ Update

**Input:** The original ranked inverted index $I$, the original ciphertext $c$, the original remote auxiliary information $\Phi$, and the update token $T_s$.

**Output:** The updated versions of ranked inverted index, ciphertexts, and remote auxiliary information $(I', c', \Phi')$.

1. Set $I' = I$, $c' = c$, and $\Phi' = \Phi$ (i.e., $V' = V$).
2. If $T_s = T_{\text{del}} = (\{, \text{delete}, C'\}$ then
   3. Mark $C_j$ as delete and replace $C_i \in c'$ with $C'_{\text{del}}$.
4. Else
   5. $T_s = T_{\text{add}} = (n + 1, \text{add, } C_{n+1}, \xi, \tau_s, \tau_a)$
   6. Set $c'$ and $\Phi'$.
   7. For the $i$-the keyword $W \in w_{n+1}$ do
      8. Compute $\varpi \leftarrow T_s[\text{free}]$ and obtain the last free location $\varpi$ in $A_s$.
      9. Compute $(0, 0, \varpi_{-1}, 0) \leftarrow A_s[\varpi]$ and obtain the second last free location $\varpi_{-1}$ in $A_s$.
      10. Update the free list to point to the second last free location $\varpi_{-1}$ in $A_s$ by setting $T_s[\text{free}] = \varpi_{-1}$.
      11. Recover the address of the first node $N_1$ of $L_W$ in $A_s$ by computing $a_1 = T_s[\lambda_i[1]] \odot \lambda_i[2]$.
      12. Store a new node at location $\varpi$ as the first node of $L_W$ by setting $A_s[\varpi] = (\lambda_i[4] \odot (0, 0, \varpi_{-1}, 0), A_s[3])$
      13. Store the point to the first node of $L_W$ in $T_s$ by setting $T_s[\lambda_i[1]] = (0, 0, \varpi_{-1}, 0) \odot \lambda_i[2]$
   14. Add $\xi$ as the last column of $V'$.
   15. For the $i$th keyword $W \in w_{n+1}$ do
      17. If $J < n$ then
         18. For $j = n$ to $J + 1$
      20. If $J = 1$ then
         21. Set $V'[\tau][1] = \beta_i[2]$ and $V'[\tau][J + 1] = \beta_i[3]$
      22. Else
         23. If $J = d_0[1] + 1$ then
            24. Set $V'[\tau][J - 1] = \beta_i[1]$ and $V'[\tau][J + 1] = \beta_i[2]$
      25. Else
         26. Set $V'[\tau][J - 1] = \beta_i[1], V'[\tau][J + 1] = \beta_i[2],$ and $V'[\tau][J + 1] = \beta_i[3]$
      27. Return $(I', c', \Phi')$

5.3 Illustration Example

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

![Fig. 3. The ranking information of documents.](image-url)

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

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

$$\psi_C = g^{\Pi_{i=1}^{n-1} P(H_i[H(C_i)])} \mod N,$$

$$\psi_I = g^{\Pi_{i=1}^{n-1} P(H(V[i][i]))} \mod N,$$

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

(Search phase) To retrieve the top-1 document containing keyword $W_2$, the user sends the search token $T_W = (F_2, W_2, G_2, W_2, P_2, W_2, 1, 3)$ to the CSP.

The CSP locates $T_s[F_2(W_2)] = T_s[3]$ and recovers the address of the first node in $A_s$ by computing $4 = T_s[3] \odot G_2(W_2)$. It then computes $A_4[4] \odot H(F_2(W_2), r_1)$ to recover the first node $N_1 = (4, 120, 3)$, 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 $N_2 = (3, 110, -1)$, where 3 is the identifier of the second document, 110 is file score, and -1 indicates $D_2$ is the last document containing keyword $W_2$. The CSP sorts documents according to their scores, and sets the search results as $R_W = (W_2, W_1)$.

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

$$\pi_C = g^{\Pi_{i=1}^{n-1} P(H_i[H(C_i)])} \mod N,$$

$$\pi_I = g^{\Pi_{i=1}^{n-1} P(H(V[i][i]))} \mod N,$$

where $\phi_I = \Pi_{i=1}^{n-1} P(H_i[I(V[i][i]])$. The messages returned to the user are set as $(R_W, \Pi)$.

Once the user receives the messages returned, she computes $x = P(H(4, H(C_1)))$ and checks if $\psi_C = \pi_C^x \mod N$. She then reconstructs $V[3][1] = H(0, 4, 3) \odot S_2(W_2)$, computes $y = P(H(3, V[3][1]))$, and checks if $\psi_I = \pi_I^y \mod N$. Note that, the following equations hold only when the search results are correct:

$$\pi_C^x \mod N = g^{\Pi_{i=1}^{n-1} P(H_i[H(C_i)])} = g^{\Pi_{i=1}^{n-1} P(H_i[H(C_i)])} = \psi_C.$$
If all these checks succeed, she can recover document \( D_2 \) from ciphertext \( C_2 \in R_{\mathcal{W}} \).

(Update Phase). As shown in Fig. 3b, document \( D_9 \), the rank-2 document for keyword \( W_2 \), is added. The user generates the update token \( T_s = T_{\text{add}} = ((5, add, C_5), \zeta, t_s, t_o) \) as follows: she first generates the ciphertext \( C_5 \) of document \( D_9 \) with SKE, and sets \( \zeta = (y_1, y_2, y_3) \), where \( y_1 \) is a random string of \( k \) bits. Then, she sets \( t_s = \beta \) where \( \beta[1] = H(0,4,5) \oplus S_k(W_2) \), \( \beta[2] = H(4,5,3) \oplus S_k(W_2) \), \( \beta[3] = H(5,3,0) \oplus S_k(W_2) \), \( \beta[4] = H(0,4,3) \oplus S_k(W_2) \), \( \beta[5] = H(4,3,0) \oplus S_k(W_2) \), and \( \beta[6] = 2 \). Furthermore, she chooses a random string \( r_3 \) of \( k \) bits and sets \( t_o = \lambda \) where \( \lambda[1] = 3, \lambda[2] = G_k(W_2), \lambda[3] = c_3, \lambda[4] = (5, 115, 0) \oplus H(P_{s_k}(W_2), \lambda[3]), \) and \( \lambda[5] = \sigma(2) = 3 \).

Next, she computes \( y \) as:
\[
\begin{align*}
  y &= \prod_{j=1}^{r_3} P(H(\lambda[5], j)) \mod \varphi(N) \\
  &= \prod_{j=1}^{r_3} P(H(\lambda[5], j)) \mod \varphi(N),
\end{align*}
\]

Finally, she generates the new local evidence \( \Psi' = (\psi'_C, \psi'_I) \) by setting \( \psi'_C = \Psi_C (P(H(5,H(C_5))) \mod N \), and \( \psi'_I = \psi_I (P(H(5,H(C_5))) \mod N \).

Given \( T_s \), the CSP generates the updated verifiable matrix \( V' \) and the updated ranked inverted indexes \( V \) 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.

- \( L_1(D) = (\#A_s, \{\xi(W)\}_{W \in \mathcal{W}}, \{\xi(D)\}_{D \in \mathcal{D}}, \{||D||\}_{D \in \mathcal{D}}) \).
- \( L_2(d, W) = (\xi(W), \text{ACCP}(W)) \). \( \text{ACCP}(W) \) is the access pattern defined as the sequence of document identifiers \( \text{id}_W = (ID_1, \ldots, ID_{d\text{max}}) \) where \( ID_j \) is the identifier of the rank-j document for keyword \( W \).
- \( L_3(d, D) = (\xi(D), \{\xi(W), \text{apprs}(W)\}_{W \in \mathcal{W}}, ||D||) \) where \( \text{apprs}(W) \) is a bit set to 1 if \( W \) appears in at least one document in \( d \) and to 0 otherwise.
- \( L_4(d, D) = (\xi(D)) \).

**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

\[ \text{satisfies confidentiality and verifiability under the strong RSA assumption in the random oracle model.} \]

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 \( V \) to the CSP for the generation of search proofs. The complexity of \( V \) is \( O(nm) \), 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 \( D_j \in d \) as follows:

**Definition 5 (Deletion table).** \( T_j \) is a dictionary of size \( |w_j| \). For each keyword \( W \in w_j, T_j[W] \) stores \( \text{addr}_s(N) \), where \( N \) is the node in the ranked linked list \( L_W \) corresponding to document \( D_j \), and \( \text{addr}_s(N) \) 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 \( W \in w_j \) by storing \( N \) as the last node of the free list \( L_{\text{free}} \) and modifying the pointer of node \( N_{-1} \) to point to node \( N_1 \). To reduce the communication cost, we encode the verifiable matrix \( V \) into accumulated values so that the CSP can generate search proofs correctly without \( V \).

**Algorithm 4. VDERS\textsuperscript{\texttt{s}}.Encryption**

**Input:** Secret key \( SK \), documents \( d \), and keywords \( w \)

**Output:** Ranked inverted index I and ciphertexts \( c' \)

1: Execute lines 1-12 of Algorithm 1 and generate I = (\( T_s, A_s \))
2: for each document \( D_j \in d \) \( \text{do} \)
3: Initialize \( T_j \) with an empty dictionary of size \( |w_j| \)
4: for each keyword \( W \in w_j \) \( \text{do} \)
5: \( \text{Set } T_j[W] \leftarrow \text{addr}_s(N) \)
6: \( \text{Set } S \leftarrow \text{SKE.Enc}(c, T_j) \)
7: \( \text{Set } C_{j} \leftarrow \text{SKE.Enc}(c, D_j) \) and \( c' = c' \cup (C_j, S) \)
8: \( \text{return } (I, c') \)

6.2 Our Construction

The main differences between VDERS\textsuperscript{0} and VDERS\textsuperscript{\texttt{s}} lie in the following algorithms.

- \( (I, c') \leftarrow \text{Encryption}(SK, d, w) \): As shown in Algorithm 4, the user first executes lines 1-12 of Algorithm 1 and generates the ranked inverted index I = (\( T_s, A_s \)). For each document \( D_j \in d \), 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 \( c' \) is set as \( (C_j, S, T_j)_{j=1} \).
(\Phi', \Psi') \leftarrow \text{AccGen}(\text{params}, SK, d, w, c') \rightarrow: The user constructs an $m \times n$ verifiable matrix $V$ as defined in Section 4.3. For each keyword $W \in w$, she calculates $i = \sigma(\xi(W))$ and $\phi_i = \prod_{j=1}^{m} \mathcal{P}(H(i, V[i][j])) \bmod N$, and sets remote auxiliary information as $\Phi' = (\phi_1, \ldots, \phi_m)$. Note that the primary role of $V$ is to record document ranking information. Since such information has been encoded into $\Phi'$, $V$ can be removed from local storage. Finally, she sets a local evidence as $\Psi' = (\psi'_c, \psi'_t)$ where $\psi'_c = \psi_C$ is calculated with Eq. (2) and $\psi'_t$ is calculated with

$$\psi'_t = g^{\prod_{i=1}^{n} \phi_i} \bmod N. \quad (16)$$

- $\mathcal{R}'_W \leftarrow \text{Search}(I, T_W)$: The CSP executes lines 1-10 of Algorithm 2. Let $c_k'$ be the first $K$ ciphertexts in $c_W$. For top-$K$ queries, it sets $\mathcal{R}'_W = (id_W, s_W, c_k')$. For normal queries, it sets $\mathcal{R}'_W = (id_W, s_W, c_W)$. For the special case, if $\mathcal{R}_W$ is the identifier of the rank-$j$ document for keyword $W$ before update, and $\mathcal{R}_W' = (id_W, s_W, c_k')$, she calculates $v_i = \phi_{\lambda[i]} \times x_i \bmod N$, where $x_i$ is calculated as follows:

$$x_i = \prod_{i=1}^{\mathcal{R}_W - 1} \mathcal{P}(H(\lambda[i], \lambda[i + 1])) \bmod N \quad (20)$$

Next, she generates the new local evidence as $\Psi' = (\psi'_c, \psi'_t)$ with Eqs. (21) and 22, respectively.

$$\psi'_c = (\psi'_c)^{\mathcal{R}_W - 1} \bmod N \quad (21)$$

$$\psi'_t = (\psi'_t)^{\mathcal{R}_W - 1} \bmod N \quad (22)$$

To add document $d_{n+1}$ with $\xi(D) = n + 1$, the user sets the update token as $T_s = T_{add} = ((n + 1), \text{add}, C_{n+1}, T_{n+1})$, $\tau_v, \tau_a$, where $C_{n+1}$ is output of $\text{SecEnc}(k_e, D_{n+1})$, $T_{n+1}$ is output of $\text{SecEnc}(k_e, T_{n+1})$, $\tau_v = (v_1, \ldots, v_{|w|})$ and $\tau_a = (a_1, \ldots, a_{|w|})$. For the $i$th keyword $W \in w_{n+1}$, she calculates $v_i$ and $\lambda_i$ as follows:

1) She computes $\lambda_i = (\lambda[1], \ldots, \lambda[i])$ with Eq. (11).

2) As deletion operations, she reconstructs $V[\lambda[i]][1], \ldots, V[\lambda[i]][|d_W|]$ with Eq. (1), and calculates $\phi_{\lambda[i]} = \prod_{j=1}^{n} \mathcal{P}(H(\lambda[i], V[\lambda[i]][j])) \bmod N$. Then, she calculates $x_i$ with Eq. (23) and sets $v_i = \phi_{\lambda[i]} \times x_i \bmod N$.

$$x_i = \prod_{i=1}^{\mathcal{R}_W - 1} \mathcal{P}(H(\lambda[i], \lambda[i + 1])) \bmod N \quad (23)$$

where $\beta_j = (\beta[1], \ldots, \beta[|w|])$ is calculated with Eq. (12). Next, she generates the new local evidence as $\Psi' = (\psi'_c', \psi'_t')$ with Eqs. (13) and 22, respectively. In Eq. (22), where $|w_j|$ is replaced by $|w_{n+1}|$.

$$(T', \Phi', \Psi') \leftarrow \text{Update}(I', c', \Phi', T_s):$ If the update token $T_s = T_{del} = ((j, \text{delete}, \tau_v, \tau_a)$, the CSP...
generates $c'$ by removing $(\bar{T}, C)$ from $c$. It generates $\Phi'$ by replacing $\phi_{\lambda[i]}$ with $v_i$ for $i \in \|w_j\|$. To generate $I' = (T'_i, A'_i)$, for the $i$th keyword $W \in w_j$, 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 $\pi \leftarrow T_{n}[\text{free}]$ and finds the last free location in $\lambda_n$.
2) It updates the free list to point to node $N$ by setting $T_{i}[\text{free}] \leftarrow \lambda_i[3]$.
3) It releases the location of $N$ by setting $A_i[\lambda_i[3]] \leftarrow (0, 0, 0, 0)$.
4) If $\lambda_i[2] \neq 0$, let $A_i[\lambda_i[2]] = (v-1, r-1)$. It updates $N$'s next pointer by setting $A_i[\lambda_i[2]] \leftarrow (0, 0, \lambda_i[3] \oplus \lambda_i[4] \oplus \lambda_i[5], v, r-1)$.
5) If $N$ is the first node of $W$, it sets $T_i[\lambda_i[1]] \leftarrow \lambda_i[4]$.

If $T_s = T_{add} = ((n + 1, add, C_{n+1}, T_{n+1}), r_v, r_a)$, it generates the updated ranked inverted index $I'$ in the same way as VDERS' Update. Next, it generates $c'$ by adding $(C_{n+1}, T_{n+1})$ into $c'$ and generates $\Phi'$ by replacing $\phi_{\lambda[i]}$ with $v_i$ for $i \in \|w_{n+1}\|$.

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 $T_{del} = (j, del)$ to the CSP, which will send back $T_c$ and remove $(C_j, T_j)$. Once the user searches any keyword $W \in \lambda_j$, she can obtain related information for generating $(\lambda_j, v_j, x_j)$. With this information, the CSP can update $(I, \Psi, \Phi)$ accordingly. After searching all keywords in $\lambda_j$, she completes all components of the deletion token. Similarly, to add document $A_{n+1}$ in VDERS', the user keeps the document identifier and related score locally and sends $T_{add} = ((n + 1, add, C_{n+1}, T_{n+1}), r_v, r_a)$ to the CSP, which updates $c'$ and $I$. Once searched any keyword in $\lambda_{n+1}$, she can obtain related information for modifying $\Psi$ and $\Phi$ accordingly. After searching all keywords in $\lambda_{n+1}$, 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 $L(d, D)$ as follows:

#### 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, $|d_W|$ is the number of documents containing keyword $W$, and $|w_j|$ 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 $\Xi = \sum |d_W|$ 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.
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 an 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 1~3 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 Encryption, 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\(^9\) and VDERS\(^8\) encrypt a ranked inverted index in the same way. Since VDERS\(^9\) needs to encrypt a deletion table \(T\), in addition, and thus it incurs more computation and communication costs compared with VDERS\(^8\). BASELINE2 encrypts an inverted index which includes a deletion table and a deletion array in addition, and thus its costs exceed VDERS\(^9\). For the AccGen algorithm, VDERS\(^9\) is the most expensive in both computation and communication costs because of the generation and transmission of a verifiable matrix \((m \times n \times k)\) bits in size; VDERS\(^8\) incurs the minimal computation cost since the number of RSA accumulator calculations is reduced to \(|d|\) 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\(^8\) 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; therefore, the execution time of (a) 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.

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\(^9\) 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 \(w\), and the execution time increases slightly as \(n\) increases. For example, the execution time in VDERS\(^9\) and VDERS\(^8\) 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\(^9\) 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

---

\(^5\) http://www.cs.cmu.edu/~enron/
constructions have better performance compared with BASELINE1. To verify the search results, the number of RSA accumulator calculations in BASELINE1, VDERS, and VDERS ̈ is mainly impacted by |d_u| + n_w, |w|, and |d_u| + |w|, 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 ̈ 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 ̈ 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 |w_j| increases. For example, as |w_j| increases from 10 to 40, the cost of BASELINE1 remains as 8 KB, but the costs of BASELINE2, VDERS, 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 |w_j|, but the cost of VDERS ̈ badly grows with |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 ̈ 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 |w_j|, but the time of VDERS ̈ grows with |w_j|. In Fig. 9d, both the time of VDERS ̈ and BASELINE2 grows as |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


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 in cloud computing. Now, she is an associate professor at the College of Computer Science and Cyberspace Security, Guizhou University, China. She has been a distinguished researcher at the University of Michigan at Ann Arbor since 2018. 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.

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.

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; a research professor at 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 Nanxing University, Nanchang, Jiangxi Province, China, in 2017. Now, he 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.

Authorized licensed use limited to: Temple University. Downloaded on August 17,2022 at 04:18:14 UTC from IEEE Xplore. Restrictions apply.