Verifiable Ranked Search Over Dynamic Encrypted Data in Cloud Computing

Qin Liu[†‡*], Xiaohong Nie[†], Xuhui Liu[†], Tao Peng[§], and Jie Wu[¶]
[†] College of Computer Science and Electronic Engineering, Hunan University, P. R. China, 410082
[‡] State key Laboratory of Networking and Switching Technology,
Beijing University of Posts and Telecommunications, P. R. China, 100876
[§] School of Information Science and Engineering, Central South University, P. R. China, 410083
[¶] Department of Computer and Information Sciences, Temple University, Philadelphia, PA 19122, USA
[*]Correspondence to: gracelq628@hnu.edu.cn

***Abstract*—Big data has become a hot topic in many areas where the volume and growth rate of data require cloud-based platforms for processing and analysis. Due to open cloud environments with very limited user-side control, existing research suggests encrypting data before outsourcing and adopting Searchable Symmetric Encryption (SSE) to facilitate keyword-based searches on the ciphertexts. However, no prior SSE constructions can simultaneously achieve *sublinear search time*, *efficient update and verification*, and *on-demand file retrieval*, which are all essential to the development of big data. To address this, we propose a Verifiable Ranked Searchable Symmetric Encryption (VRSSE) scheme that allows a user to perform top-K searches on a *dynamic* file collection while efficiently verifying the correctness of the search results. VRSSE is constructed based on the *ranked inverted index*, which contains multiple inverted lists that link sets of file nodes relating a specific keyword. For verifiable ranked searches, file nodes are ordered according to their ranks for such a keyword, and information about a node's prior/following neighbor will be encoded with the *RSA accumulator*. Extensive experiments on real data sets demonstrate the efficiency and effectiveness of our proposed scheme.**

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

I. Introduction

Over the past few years, the continuous increase of data volume has produced overwhelming data flows in both structured and unstructured formats. Data creation is occurring at a record rate, and has emerged as a widely agreed-upon trend in both industry and academia. Due to big data's high volume, high velocity, and wide variety of characteristics, traditional platforms are incapable of analyzing and processing such significant amounts of dynamic data of varying modalities.

Cloud computing is a new paradigm that enables ubiquitous and on-demand access to a shared pool of configurable computing resources. Cloud-based platforms with overwhelming advantages over traditional platforms are increasingly utilized as potential hosts for big data. Since the cloud service provider (CSP) is outside the users' trusted domain, existing research suggests encrypting data before outsourcing [1]. In a typical cloud computing environment, a user will utilize Symmetric Key Encryption (SKE) and Searchable Symmetric Encryption (SSE) to encrypt file and keyword contents, respectively, and then upload the ciphertexts to the cloud. Later, she will generate a search token, TK_w, to retrieve all the files containing keyword w and perform decryption locally to recover the files' contents. In this process, files, keywords, and search tokens are encrypted under the user's private key. Thus, the CSP cannot know which keywords the user has searched for or what files have been returned.

As a seminal work in SSE, Curtmola et al. [2], provided a rigorous security definition and constructed schemes based on an inverted index for sublinear search. Recently, Kurosawa et al. [3], [4] constructed verifiable SSE schemes, in which the user can detect any cheating behavior of malicious servers. Kamara et. al. [5], [6] proposed dynamic SSE schemes, where the user can efficiently update encrypted data. As an attempt to optimize the search results, ranked SSE schemes [7], [8], [9], [10] have also been proposed to allow the users to retrieve the best-matched files. However, these works hardly meet the security needs of big data in cloud computing environments.

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 a set of files indexed by keyword *Bob*, an email received on April 2 will have a higher rank than an email received on April 1. To keep keyword and file contents secret, Alice stores them in encrypted forms to the cloud. Since there could be hundreds of files matching a specific keyword, the consumed communication will be extensive if the CSP returns all the matched files. Therefore, Alice may want to perform a top-K search to retrieve the most recent emails. Moreover, to save money, Alice may want to store only the emails received in the last three months.

In this application scenario, the adopted SSE scheme should meet the following requirements: (1) *Ranked search.* The user is allowed to perform a top-K search to retrieve the best-matched files. (2) *Dynamic.* The user is able to update (add and delete) files stored in the cloud. (3) *Verifiability*. The malicious CSP may delete encrypted files 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 files. Therefore, the user should have the ability to verify the correctness of the search results. (4) *Efficiency*. The user can efficiently perform searches, updates,

978-1-5386-2704-4/17/$31.00 ©2017 IEEE

and verifications on a set of encrypted files.

Unfortunately, existing SSE schemes only partially address these requirements. To simultaneously satisfy all these properties, this paper proposes a Verifiable Ranked Searchable Symmetric Encryption (VRSSE) 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 build a *ranked inverted index* $\mathcal{I}$ from a collection of files to facilitate top-K searches, while recording the rank information in a *verifiable matrix*, $\mathcal{V}$, for verifiable updates and searches. Specifically, $\mathcal{I}$ contains multiple inverted lists, each of which links to a set of file nodes that relates to a specific keyword. File nodes are ordered according to the rank of keywords. Information about a node's prior/following neighbor, recorded in $\mathcal{V}$, will be encoded with the RSA accumulator [11]. $\mathcal{I}$ is an extension of the inverted index [2] and allows for sublinear-time searches. Furthermore, $\mathcal{I}$ facilitates updates since the update of a file D only needs to modify the neighbors of the file node that corresponds to D. Our main contributions are as follows:

- To the best of our knowledge, this is the first attempt to devise a verifiable, ranked, and dynamic SSE scheme to preserve big data security in a cloud environment.
- Our scheme allows the user to efficiently update the file collection and verify the correctness of a top-K search while preserving user privacy from the CSP.
- We theoretically analyze the security and performance of our scheme and conduct extensive experiments on real data sets to validate its effectiveness and efficiency.

Paper organization. We provide the preliminaries in Section II and an overview of this work in Section III. After constructing our scheme in Section IV, we evaluate it in Section V. Finally, we conclude the paper in Section VI.

II. Preliminaries

A. System Model

The system consists of the cloud service provider (CSP) and the cloud user. The CSP maintains cloud infrastructures, pooling the bandwidth, storage space, and CPU power to provide data storage and query services. Cloud users, who pay the services residing on the cloud, can be further classified into *data owner* and *data user* according to data ownership.

The data owner first creates ciphertexts $\mathcal{C} = \{C_1, \ldots, C_n\}$ for a file collection $\mathcal{D} = \{D_1, \ldots, D_n\}$. It then builds an encrypted index $\mathcal{I}$ and a verifiable matrix $\mathcal{V}$ from $\mathcal{D}$ and the universal keywords $\mathcal{W} = \{w_1, \ldots, w_m\}$. After uploading all the above information to the cloud, she can perform updates (add/delete) on ciphertexts and retrieve the data of interest *on demand* in a *verifiable* way. The data owner can also delegate the search/verification ability to authorized data users. In this paper, we do not discriminate between the data owner and data user, and refer to them both as *user* for simplicity.

B. Adversary Model

The users are assumed to be fully trusted, and the CSP is the potential attacker who is assumed to be *honest but curious*. That is, the CSP will always correctly execute a given protocol, but may try to learn some additional information about the stored data and the received message.

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. As defined in [2], access pattern refers to the outcome of search results, i.e., which files have been returned; the search patterns refer to whether two searches were performed for the same keyword. Like existing SSE schemes, our scheme will reveal the access pattern and the search pattern to the CSP. Furthermore, in a top-K search, only K highest ranked files will be returned. Therefore, information about file ranks will also be leaked during the search phase.

C. RSA Accumulator

Let κ 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 $\mathcal{F} = \{f : \{0,1\}^{3\lambda} \to \{0,1\}^{\lambda}\}$ be a two-universal family of hash functions. Let $\mathbf{N} = pq$, 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:

(1) For each y_i, we choose a random prime x_i, denoted as $\mathcal{P}(y_i)$, such that $f(x_i) = y_i$. The accumulated value for E can be calculated as $Acc(E) = g^{\prod_{i=1}^{n} \mathcal{P}(y_i)} \mod \mathbf{N}$. (2) For any subset $E' \subseteq E$, a witness $\pi = g^{\prod_{y_i \in E - E'} \mathcal{P}(y_i)} \mod \mathbf{N}$ can be produced. (3) The subset test can be carried out by checking $Acc(E) = \pi^{\prod_{y_i \in E'} \mathcal{P}(y_i)} \mod \mathbf{N}$.

III. Scheme Overview

A. Notations

The set of all binary strings of length η is denoted as $\{0,1\}^{\eta}$ and the set of finite binary strings is denoted as $\{0,1\}^{*}$. Given a sequence of elements $\mathcal{S}$, we refer to its i-th element as $\mathcal{S}[i]$ and to its total number as $\#\mathcal{S}$. Given a matrix $\mathcal{M}$, the entry in its i-th and j-th column is denoted as $\mathcal{M}[i][j]$. If s is a string then $|s|$ refers to its bit length. The concatenation of c strings $s_1, \ldots, s_c$ is denoted by $\langle s_1, \ldots, s_c \rangle$. For quick reference, the most relevant notations used in our scheme are listed below:

- $\mathcal{D} = \{D_1, \ldots, D_n\}$: A collection of n files, where $j \in [1, n]$ is the identifier of file D_j. The ciphertext collection $\mathcal{C} = \{(1, C_1), \ldots, (n, C_n)\}$.
- $\mathcal{W} = \{w_1, \ldots, w_m\}$: A set of m keywords, where $i \in [1, m]$ is the identifier of keyword w_i.
- $\mathcal{ID}(w_i) = (id_1, \ldots, id_{\#w_i})$: A sequence of $\#w_i$ identifiers of files containing keyword w_i.
- $\mathcal{ID}(D_i) = (id_1, \ldots, id_{\#D_i})$: A sequence of $\#D_i$ identifiers of keywords contained in file D_j.
- $\mathcal{I} = \{T_s, A_s\}$: The ranked inverted index consists of a search table T_s and a search array A_s.
- $\mathcal{V}$: The verifiable matrix with $m \times n$ entries.
- $TK_w, TK_*(D)$: The search token generated for keyword w and the add/delete token generated for file D.
- $\mathcal{C}_{w,K}$: The results for the top-K search of keyword w.

B. Ranked Inverted Index

The ranked inverted index $\mathcal{I} = \{T_s, A_s\}$, where for each word $w \in \mathcal{W}$, a list $\mathcal{L}_w$ of $\#w$ nodes are randomly stored in the search array A_s and the pointer to the head of $\mathcal{L}_w$ is included in the search table T_s. Specifically, a search array A_s is an array that consists of $\#A_s$ cells with $A_s[i]$ denoting the value stored at location i. A search table T_s is a dictionary that stores $\#T_s$ 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, $T_s[k] = v$ denotes storing the value v under key k in T_s. In our construction, $\#T_s = m + 1$ where the first m entries correspond to keywords in $\mathcal{W}$ and the last entry points to an unused cell in A_s. Similar to [5], we set $\#A_s = |\mathcal{C}|/8 + z$, where $|\mathcal{C}|$ is the size of ciphertext collection and $z \in \mathbb{N}$ is the size of unused cells. Unlike the previous work, in which file identifers randomly appear in $\mathcal{ID}(w)$, our scheme requires $\mathcal{ID}(w)$ to have the following ranking property:

Ranking property. *Let $R_{i,j}$ denote the rank of file $D_j \in \mathcal{D}$ for keyword $w_i \in \mathcal{W}$, and let $L_{i,j}$ denote the order of j in $\mathcal{ID}(w_i)$. For $j, k \in \mathcal{ID}(w_i)$, we have that j is before k (denoted by $L_{i,j} < L_{i,k}$) if the rank of D_j is higher than that of D_k for w_i, (denoted by $R_{i,j} < R_{i,k}$).*

That is, given $\mathcal{ID}(w) = (id_1, \dots, id_{\#w})$, id_j is the identifier of the rank-j file for $j \in [1, \#w]$. Thus, for each keyword $w \in \mathcal{W}$, the ranked list $\mathcal{L}_w$ can be defined as follows:

Ranked list. *$\mathcal{L}_w$ is composed of $\#w$ nodes $(N_1, \dots, N_{\#w})$ and defined as $N_j = \langle id_j, addr_s(N_{j+1})\rangle$, where $id_j \in \mathcal{ID}(w)$ is the identifier of the rank-j file for keyword w and $addr_s(N_{j+1})$ is the address of node N_{j+1} in the search array A_s. In the special case, $N_{\#w} = \langle id_{\#w}, \mathbf{0}\rangle$.*

C. Verifiable Matrix

Let $H : \{0,1\}^* \to \{0,1\}^\kappa$ be a collision-free hash function where κ is a parameter for securing the system. Since a keyword appears in, at most, n files, the verifiable matrix $\mathcal{V}$ is an $m \times n$ matrix where row $i \in [1, m]$ corresponds to a keyword $w \in \mathcal{W}$ and column $j \in [1, n]$ corresponds to a rank $j \in [1, n]$. The relationship between the row i and the keyword w is determined by the key-value pairs of the search table T_s.

Let $\mathcal{V}[i][j]$ denote the entry in the i-th row and j-th column, and let $\mathcal{R}_{i,j}$ be a random string stored at $\mathcal{V}[i][j]$. We have:

$$\mathcal{V}[i][j] = \begin{cases} H(id_{j-1}, id_j, id_{j+1}), j \in [1, \#w] \\ \mathcal{R}_{i,j}, otherwise, \end{cases} \tag{1}$$

where $\mathcal{V}[i][j]$ records the information of rank-$(j-1)$, rank-j, and rank-$(j+1)$ files for keyword w. $\mathcal{V}[i][1] = H(\mathbf{0}, id_1, id_2)$, and $\mathcal{V}[i][\#w] = H(id_{\#w-1}, id_{\#w}, \mathbf{0})$. For $j \in [\#w+1, n]$, $\mathcal{V}[i][j]$ is filled with random strings.

IV. The Proposed VRSSE Scheme

A. Our Construction

Suppose that the whole system is secured under parameter κ. Let $SKE = (Gen, Enc, Dec)$ be a symmetric-key encryption scheme, where Gen is a key generation algorithm, Enc is an encryption algorithm, and Dec is a decryption algorithm. Let $\mathcal{F} = \{f : \{0,1\}^{3\kappa} \to \{0,1\}^\kappa\}$ be a two-universal family of functions, let $\mathbb{H}_1 : \{0,1\}^* \to \{0,1\}^*$ be random oracles, and let $H : \{0,1\}^* \to \{0,1\}^\kappa$ be a collision-free hash function. 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 pseudorandom functions (PRFs). Our VRSSE scheme is constructed as follows:

(Initial phase)

- $Setup(1^\kappa) \to (PK, SK)$: The user randomly chooses four κ-bit strings k_1, k_2, k_3, k_4 as keys of PRFs, runs $SKE.Gen(1^\kappa)$ to generate k_e, and generates $(\mathbf{N} = pq, g)$. Let $\mathcal{P}(y)$ be a random prime x such that $f(x) = y$. We have $PK = (\mathbf{N}, g, f)$ and $SK = (p, q, k_e, k_1, k_2, k_3, k_4)$.

(Store phase)

- $EncIndex(SK, \mathcal{D}, \mathcal{W}) \to \mathcal{I}$: Let A_s be an array of size $|\mathcal{C}|/8 + z$ and T_s be a dictionary of size $m + 1$. For each keyword $w \in \mathcal{W}$, the user performs following:

(1) She creates list $\mathcal{L}_w$ by choosing $\#w$ random locations in A_s. For $i \in [1, \#w]$, N_i is set to $\langle id_i, addr_s(N_{i+1})\rangle$ as defined in Section III-B, and will be encrypted with Eq. 2:

$$A_s[addr_s(N_i)] = (N_i \oplus \mathbb{H}_1(P_{k_3}(w), r_i), r_i), \tag{2}$$

where r_i is a κ-bit random string.

(2) She stores the address of the head of $\mathcal{L}_w$ in T_s by setting:

$$T_s[F_{k_1}(w)] = addr_s(N_1) \oplus G_{k_2}(w). \tag{3}$$

Let **free** denote a keyword not in $\mathcal{W}$. The user creates an unencrypted free list $\mathcal{L}_{\mathbf{free}}$ by randomly choosing $z \in \mathbb{N}$ unused cells in A_s. Let $\mathbf{F}_1, \dots, \mathbf{F}_z$ be the free nodes in A_s. For $1 \leq i \leq z$, she sets $A_s[addr_s(\mathbf{F}_i)] = \langle \mathbf{0}, addr_s(\mathbf{F}_{i-1})\rangle$, where $addr_s(\mathbf{F}_0) = NULL$. Then she sets $T_s[\mathbf{free}] = addr_s(\mathbf{F}_z)$. Finally, she outputs the encrypted index $\mathcal{I} = (T_s, A_s)$.

- $EncFile(\mathcal{D}, SK) \to \mathcal{C}$: For each file $D_i \in \mathcal{D}$, the user runs $SKE.Enc(k_e, D_i)$ to generate the ciphertext C_i. The ciphertext collection $\mathcal{C} = \{(1, C_1), \dots, (n, C_n)\}$.
- $AccGen(PK, SK, \mathcal{D}, \mathcal{W}) \to (\mathcal{V}, \mathbf{A})$: Let $\mathcal{V}$ be an $m \times n$ matrix as defined in Section III-C. For each keyword $w \in \mathcal{W}$, the user locates the row in $\mathcal{V}$ by computing $F_{k_1}(w)$, and she encrypts each non-random value by calculating $\mathcal{V}[F_{k_1}(w)][j] \oplus S_{k_4}(w)$. The RSA accumulator [11] is applied to verify the correctness of the search results. Then, she calculates $\mathbf{A} = (\mathbf{A}_C, \mathbf{A}_I)$ with Eqs. 4∼5 and stores them locally:

$$\mathbf{A}_C = g^{\prod_{i=1}^{n} \mathcal{P}(H(i, H(C_i)))} \mod \mathbf{N}, \tag{4}$$

$$\mathbf{A}_I = g^{\prod_{i=1}^{m} \prod_{j=1}^{n} \mathcal{P}(H(i, \mathcal{V}[i][j]))} \mod \mathbf{N}. \tag{5}$$

(Search phase)

- $SrcToken(w, SK) \to TK_w$: To retrieve top-K files containing keyword w, the user generates a search token $TK_w = (\tau_1, \tau_2, \tau_3) = (F_{k_1}(w), G_{k_2}(w), P_{k_3}(w))$.
- $Search(TK_w, K, \mathcal{I}) \to \mathcal{C}_{w,K}$: On receiving $TK_w = (\tau_1, \tau_2, \tau_3) = (F_{k_1}(w), G_{k_2}(w), P_{k_3}(w))$, the CSP locates $T_s[\tau_1]$ and returns $\perp$ if τ_1 is not in T_s. Otherwise, it computes

$\alpha_1 = T_s[\tau_1] \oplus \tau_2$ to recover the pointer to the head of $\mathcal{L}_w$. Suppose $A_s[\alpha_1] = (v_1, r_1)$; it recovers N_1 by computing:

$$(id_1, addr_s(N_2)) = v_1 \oplus \mathbb{H}_1(\tau_3, r_1). \quad (6)$$

For $2 \leq i \leq K+1$, node N_i will be recovered as above. Let $\mathcal{ID}(w,K) = \{id_1, \ldots, id_K\}$ be the identifers of the top-K files containing keyword w. The returned ciphertexts are set as $\mathcal{C}_{w,K} = (id_{K+1}, \{(i, id_i, C_{id_i})\}_{i \in [1,K] \wedge id_i \in \mathcal{ID}(w,K)})$. Note that a top-$K$ query will be extended to a top-$(K+K')$ query if there are K' files in $\mathcal{C}_{w,K}$ marked with **delete**.

• $GenProof(TK_w, PK, \mathcal{C}, \mathcal{V}) \rightarrow \Pi$: The CSP computes proofs $\Pi = \{\pi_C, \pi_{I,1}, \pi_{I,2}\}$ by computing:

$$\begin{aligned} \pi_C &= g^{\prod_{i \notin \mathcal{ID}(w,K)} \mathcal{P}(H(i, H(C_i)))} \mod \mathbf{N}, \\ \pi_{I,1} &= \prod_{id_j \notin \mathcal{ID}(w,K)} \mathcal{P}(H(\tau_1, \mathcal{V}[\tau_1][j])), \\ \pi_{I,2} &= g^{\prod_{i \neq \tau_1} \prod_{j=1}^{n} \mathcal{P}(H(i, \mathcal{V}[i][j]))} \mod \mathbf{N}. \end{aligned} \quad (7)$$

(Recovery phase)

• $Verify(PK, SK, \mathcal{C}_{w,K}, \Pi, \mathbf{A}) \rightarrow \{0,1\}$: For each ciphertext in $\mathcal{C}_{w,K}$, the user first computes $x_i = \mathcal{P}(H(id_i, H(C_{id_i})))$ and then checks if Eq. 8 holds:

$$\mathbf{A}_C = (\pi_C)^{\prod_{id_i \in \mathcal{ID}(w,K)} x_i} \mod \mathbf{N}. \quad (8)$$

The user then reconstructs $\mathcal{V}[F_{k_1}(w)][1], \ldots, \mathcal{V}[F_{k_1}(w)][K]$ from $\mathcal{C}_{w,K}$. Next, for $j = 1, \ldots, K$, she computes $z_j = \mathcal{P}(H(F_{k_1}(w), \mathcal{V}[F_{k_1}(w)][j]))$ and checks if Eq. 9 holds:

$$\mathbf{A}_I = (\pi_{I,2})^{\pi_{I,1} \cdot \prod_{j=1}^{K} z_j} \mod \mathbf{N}. \quad (9)$$

If so, the $Verify$ algorithm outputs 1, otherwise it outputs 0.

• $DecFile(\mathcal{C}_{w,K}, SK) \rightarrow \{D\}_{C \in \mathcal{C}_{w,K}}$: The user runs $SKE.Dec(k_e, C)$ to recover D for each ciphertext in $\mathcal{C}_{w,K}$.

(Update phase)

• $UpdToken(SK, D) \rightarrow TK_*(D)$: To add file D_{n+1}, for each keyword w_{id_i} with $id_i \in \mathcal{ID}(D_{n+1})$, the user:

(1) determines the rank of D_{n+1}, denoted as J, and then computes $\beta_i = (\beta_i[1], \beta_i[2], \beta_i[3], \beta_i[4])$:

$$\begin{aligned} \beta_i[1] = F_{k_1}(w_{id_i}), \beta_i[2] = \langle \mathrm{J}-1, \mathcal{V}'[\beta_i[1]][\mathrm{J}-1] \rangle, \\ \beta_i[3] = \langle \mathrm{J}, \mathcal{V}'[\beta_i[1]][\mathrm{J}] \rangle, \beta_i[4] = \langle \mathrm{J}+1, \mathcal{V}'[\beta_i[1]][\mathrm{J}+1] \rangle, \end{aligned}$$

where $\mathcal{V}'$ as the updated verifiable matrix can be computed:

$$\begin{aligned} \mathcal{V}'[\beta_i[1]][\mathrm{J}-1] &= H(\overline{id}_{\mathrm{J}-2}, \overline{id}_{\mathrm{J}-1}, n+1) \oplus S_{k_4}(w_{id_i}), \\ \mathcal{V}'[\beta_i[1]][\mathrm{J}] &= H(\overline{id}_{\mathrm{J}-1}, n+1, \overline{id}_{\mathrm{J}}) \oplus S_{k_4}(w_{id_i}), \\ \mathcal{V}'[\beta_i[1]][\mathrm{J}+1] &= H(n+1, \overline{id}_{\mathrm{J}}, \overline{id}_{\mathrm{J}+1}) \oplus S_{k_4}(w_{id_i}), \end{aligned}$$

where $\overline{id}_J$ is the identifier of the rank-J file for keyword w_{id_i} in the original verifiable matrixes $\mathcal{V}$, which can be obtained by performing a top-K search on the cloud.

(2) computes $\lambda_i = (G_{k_2}(w_{id_i}), P_{k_3}(w_{id_i}), \langle n+1, \mathbf{0} \rangle \oplus \mathbb{H}_1(P_{k_3}(w_{id}), r_i), r_i)$ where r_i is a random string.

Finally, the user sets the update token $TK_*(D) = TK_{\textbf{add}}(D) = \{(n+1, C_{n+1}), \mathfrak{C}, \tau_v, \tau_a\}$, where $\mathfrak{C} = (\mathcal{R}_{1,n+1}, \ldots, \mathcal{R}_{m,n+1})$ is a sequence of κ-bit random strings, $\tau_v = (\beta_1, \ldots, \beta_{\#D_{n+1}})$ and $\tau_a = (\lambda_1, \ldots, \lambda_{\#D_{n+1}})$.

To delete file D_i, the user sets the update token $TK_*(D) = TK_{\textbf{del}}(D) = (i, \textbf{delete})$ and then sends it to the CSP.

• $AccUpdate(PK, SK, TK_*(D), \mathbf{A}_C, \mathbf{A}_I) \rightarrow (\mathbf{A}'_C, \mathbf{A}'_I)$: If $TK_*(D) = TK_{\textbf{add}}(D)$, the user updates $\mathbf{A}_C$ by computing:

$$\mathbf{A}'_C = (\mathbf{A}_C)^{\mathcal{P}(H(n+1, H(C_{n+1})))} \mod \mathbf{N}. \quad (10)$$

For each keyword w_{id_i} with $id_i \in \mathcal{ID}(D)$, she computes:

$$z_i = \frac{\prod_{j \in [\mathrm{J}-1, \mathrm{J}+1]} \mathcal{P}(H(F_{k_1}(w_{id_i}), \mathcal{V}'[F_{k_1}(w_{id_i})][j]))}{\prod_{j \in [\mathrm{J}-1, \mathrm{J}+1]} \mathcal{P}(H(F_{k_1}(w_{id_i}), \mathcal{V}[F_{k_1}(w_{id_i})][j]))},$$

where J is the rank of file D_{n+1} for keyword w_{id_i}. Then she updates $\mathbf{A}_I$ by computing:

$$\mathbf{A'}_I = (\mathbf{A}_I)^{\prod_{w_{id_i} \notin \mathcal{ID}(D)} \mathcal{R}_{i^*, n+1} \prod_{w_{id_i} \in \mathcal{ID}(D)} z_i \mod \mathcal{P}} \mod \mathbf{N},$$

where $i^* = F_{k_1}(w_{id_i})$, $\mathcal{P} = (p-1)(q-1)$, and $\mathcal{R}_{i^*,n+1} \in \mathfrak{C}$ is a random string.

If $TK_*(D) = TK_{\textbf{del}}(D)$, the user first generates ciphertexts C' and C_i by running $SKE.Enc(k_e, \mathbf{delete})$ and $SKE.Enc(k_e, D_i)$, respectively. She then computes $x = \mathcal{P}(H(i, H(C_i)))$, $x' = \mathcal{P}(H(i, H(C')))$, and $d = x'/x \mod (p-1)(q-1)$. She then updates $\mathbf{A}_C$ to $\mathbf{A}'_C = (\mathbf{A}_C)^d$.

• $Update(\mathcal{I}, \mathcal{C}, \mathcal{V}, TK_*) \rightarrow (\mathcal{I}', \mathcal{C}', \mathcal{V}')$: If the update token $TK_*(D) = TK_{\textbf{del}}(D) = (i, \textbf{delete})$, the CSP replaces the ciphertext C_i with **delete**. Otherwise, given $TK_*(D) = TK_{\textbf{add}}(D) = \{(n+1, C_{n+1}), \mathfrak{C}, \tau_v, \tau_a\}$, the CSP first adds $\mathfrak{C}$ as the last column of the verifiable matrix and then updates $\mathcal{C}$ to $\mathcal{C}'$ by adding $(n+1, C_{n+1})$ to $\mathcal{C}$. For $1 \leq i \leq \#D_{n+1}$, the CSP performs updates as follows:

(1) It computes $\varphi \leftarrow T_s[\textbf{free}]$ to find the last free location φ in the search array A_s, and it calculates $(\mathbf{0}, \varphi_{-1}) \leftarrow A_s[\varphi]$ to find φ_{-1}, the address of next free node in A_s. Then, it updates the search table to point to φ_{-1} by setting $T_s[\textbf{free}] = \varphi_{-1}$.

(2) It recovers a pointer to the first node N_1 by computing $\alpha_1 \leftarrow T_s(\beta_i[1]) \oplus \lambda_i[1]$. Then, it recovers nodes $\{N_2, \ldots, N_{\mathrm{J}}\}$ in the search array by running the $Search$ algorithm, where J is the rank of file D_{n+1}.

(3) Let α_J denote the address $addr_s(N_J)$ of node J in the search array. It stores the new node at location φ and sets $A_s[\varphi] = (\lambda_i[3] \oplus \langle \mathbf{0}, \alpha_{\mathrm{J}} \rangle), \lambda_i[4])$. If $A_s[\alpha_{\mathrm{J}-1}] = (v, r)$, it sets $A_s[\alpha_{\mathrm{J}-1}] = (v \oplus \langle \mathbf{0}, \varphi \oplus \alpha_{\mathrm{J}} \rangle, r)$.

(4) It sets $T_s[\beta_i[1]] = \varphi \oplus \lambda_i[1]$ if $\mathrm{J} = 1$.

(5) It moves $\mathcal{V}[\beta_i[1]][j]$ backward one column for $j \in [\mathrm{J}+1, n]$. Then it replaces $\mathcal{V}[\beta_i[1]][\mathrm{J}-1]$, $\mathcal{V}[\beta_i[1]][\mathrm{J}]$, and $\mathcal{V}[\beta_i[1]][\mathrm{J}+1]$ with the updated ciphertexts in $\beta_i[2], \beta_i[3], \beta_i[4]$.

B. Security Sketch

The SSE scheme proposed in [5] has been proven to be CPK-2 secure. We will show that our scheme leaks only extra rank information compared with [5]. In the storage phase, the message from the user to the CSP is $\{\mathcal{C}, \mathcal{I}, \mathcal{V}\}$, where $\mathcal{C}$ is generated by the SKE scheme that has been proven to be CPA secure under the adaptive adversaries. Furthermore, $\mathcal{I}$ and $\mathcal{V}$ are encrypted by PRFs, which are secured by the secret keys $k_1, \ldots, k_4$. Therefore, our scheme is as secure as [5].

TABLE I
COMPARISON OF COMPUTATION COST

Server	Store	Search	Add	Delete
Kurosawa [4]	$O(1)$	$O(n \cdot m)$	$O(1)$	$O(1)$
Our VRSSE	$O(1)$	$O(n \cdot m)$	$O(1)$	$O(1)$
User	Store	Search	Add	Delete
Kurosawa [4]	$O(n \cdot m)$	$O(n)$	$O(m)$	$O(1)$
Our VRSSE	$O(n \cdot m)$	$O(K)$	$O(\#D)$	$O(1)$

TABLE II
COMPARISON OF COMMUNICATION COST

	Kamara [5]	Kurosawa [4]	Our VRSSE
Store	$O(\sum \#w + m)$	$n \cdot (m + 128)$	$O(\sum \#w + n \cdot m \cdot \kappa)$
Search	$O(\kappa)$	$n + O(\kappa)$	$O(\kappa)$
Add	$O(\#D \cdot \kappa)$	m	$O(m \cdot \kappa + \#D \cdot \kappa)$
Delete	$O(\kappa)$	$O(\kappa)$	$O(\kappa)$

In the update phase, the message from the user to the CSP is $\{TK_*\}$. If $TK_* = TK_{\textbf{add}}$, the CSP will know the identifier of the newly added file D. Then, it will update the encrypted ranked list $\mathcal{I}$ by adding a node associated with file D before the J-th node in $\mathcal{L}_w$ where J denotes the rank of D for keyword w. Furthermore, the CSP will update the $\mathrm{J}-1$-th, J-th, and $\mathrm{J}+1$-th columns of the verifiable matrix $\mathcal{V}$. Therefore, the CSP will know the rank of file D apart from the access pattern and search pattern. However, $TK_{\textbf{add}}$ is in the encrypted forms and the keyword/file contents are kept secret from the CSP. If $TK_* = TK_{\textbf{del}}$, no more information will be leaked except the identifier of the removed file.

In the search phase, the message from the user to the CSP is $\{TK_w\}$, the construction of which is the same as [5]. Given TK_w, the CSP will know the identifers of the K best match files and the rank of each returned file. However, TK_w is encrypted by the PRFs under secret keys. The CSP cannot deduce file/keyword contents without these keys because of the security of PRFs. Therefore, our VRSSE scheme leaks no more information than [5] except for the rank information. ∎

V. EVALUATION

A. Performance Analysis

We will compare the performance of our VRSSE scheme with the schemes proposed in [5] and [4] in terms of computational and communication complexities. For ease of comparison, the other schemes are denoted as Kamara [5] and Kurosawa [4], respectively. For computation cost, we only consider the most expensive operations, i.e., the calculation of the RSA accumulators. Note that Kamara [5] is not verifiable, and thus, its cost is 0; it will not be listed. Given a collection of n files and m keywords, the comparison results are shown in Table I, where K is the parameter for a top-K search and $\#D$ is the total number of keywords contained in D.

All three schemes encrypt files with SKE, and the sizes of the ciphertexts are the same. Therefore, we only consider the sizes of index/matrix/tokens. The comparison of communication costs is shown Table II, where κ is the security parameter, $\#w$ is the number of files containing keyword w, and other notations are the same as those in Table I.

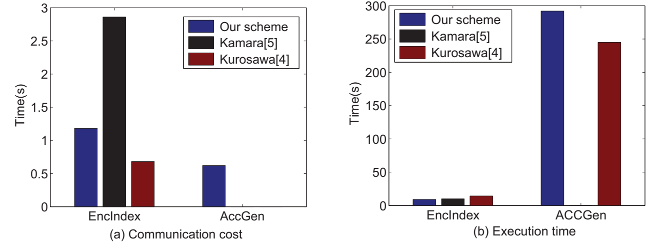

Fig. 1. Comparison in store phase.

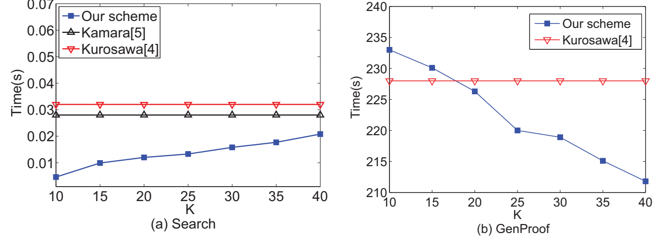

Fig. 2. Execution time in the search phase.

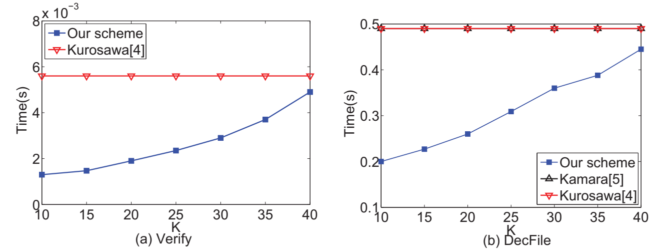

Fig. 3. Execution time in the recovery phase.

B. Parameter Setting

Experiments are conducted on a local machine running the Microsoft Windows 7 Ultimate operating system with an Inter Core i3 CPU running at 2.3GHz and a 4GB memory. The programs are implemented in Java and compiled using Eclipse 4.3.2. The cryptographic algorithms are implemented with JPBC library [12].

To validate the effectiveness and efficiency of our scheme in practice, we conduct a performance evaluation on a real data set, the Internet Request For Comments dataset (RFC) [1]. This data set has 6,870 plaintext files with a total size of about 349MB. The average size of each file is 52KB. We use the Hermetic Word Frequency Counter [2] to extract keywords from each RFC file, and we choose $[1, 5]$ keywords for each file after ranking them by frequency of occurrence. In the experiments, we select $n = 1,000$ files from the data set. The number of distinct keywords is $m = 884$, and each keyword appears in 1∼44 files. We execute each experiment multiple times to obtain the average execution time.

C. Experiment Results

We will compare the performance of our scheme with Kamara [5] and Kurosawa [4] in terms of the communication cost and execution time. Since Kamara [5] is unverifiable, it lacks algorithms *AccGen*, *GenProof*, *Verify*, and *AccUpdate*. In all three schemes, SKE is employed to encrypt file contents, and the computation and communication costs of generating a

[1] http://www.ietf.org/rfc.html
[2] http://www.hermetic.ch/wfc/wfc.htm

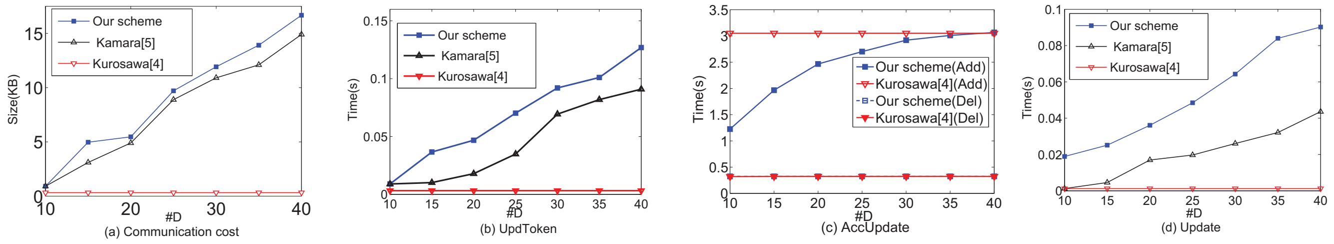

Fig. 4. Communication cost and execution time in the update phase.

search token are very small. Therefore, we omit comparison of the algorithms $EncFile$ and $SrcToken$ in our experiments.

Fig. 1-(a) shows the comparison results of communication costs in the store phases. For the $EncIndex$ algorithm, Kamara [5] generates a copy of index and thus, incurs the most cost. For the $AccGen$ algorithm, our scheme is the most expensive because of the transmission of the verifiable matrix. Fig. 1-(b) shows the comparison results of execution time in the store phase. To generate an encrypted index, our scheme needs to encrypt m entries in the search table and $\sum \#w$ cells of the search array. Kamara [5] needs to encrypt *dual* structures and Kurosawa [4] needs to encrypt n bits for each keyword. Therefore, our scheme incurs the minimal execution time for *EncIndex*. To generate accumulated values, our scheme needs to encrypt each non-random value in the verifiable matrix; therefore, it incurs more computation time than Kurosawa [4].

Fig. 2 shows the execution time in the search phase. In the *Search* algorithm, our scheme only needs to decrypt $K + 1$ cells to return the top-K files; thus, it incurs the minimal cost. Similarly, the communication cost can be reduced to $K/\#w$. To generate a proof, our *GenProof* algorithm generates Π, the computation cost of which is mainly impacted by K. Our execution time decreases from $233s$ to $211.74s$ as K increases from 10 to 40.

Fig. 3 shows the comparison results in the recovery phase. Since our scheme only needs to decrypt K best-match files, it incurs the minimal cost for *DecFile*. To verify the search results, our scheme needs to calculate x_i and z_i for each returned file. Therefore, the computation cost is impacted by K. The execution time for our *Verify* algorithm grows from 0.74ms to 4.9ms as K increases from 5 to 40.

Fig. 4 shows the comparison results in the update phase. In our scheme, given a newly added file D associated with $\#D$ keywords, the user calculates and transmits $\beta[i], \lambda[i]$ in the $UdpToken$ algorithm and z_i in the $AccUpdate$ algorithm for $i \in [1, \#D]$. Further, the CSP needs to update corresponding forwarding pointers for the nodes in $\mathcal{L}_w$. Therefore, both the communication and computation costs of adding a file are impacted by $\#D$. In terms of deleting a file, the costs of the $AccUpdate$ algorithm in both our scheme and Kurosawa [4] are constant. Moreover, the costs of the $Update$ algorithm are almost 0 in both schemes, in which means the CSP only needs to replace the ciphertext with **delete**.

VI. Conclusion

Due to the high volume and velocity of big data, storing big data on cloud-based platforms is wise. In such an environment, a key problem is how to perform a top-K search on a set of dynamic files securely, efficiently, and in a way that is verifiable. In this paper, we propose a VRSSE scheme to achieve verifiable updates and ranked searches on a set of encrypted files. Experiment results show that our scheme has a better search performance than existing SSE schemes, since the user can retrieve data of interest on demand. However, our VRSSE scheme supports only single-keyword searches. As part of our future work, we will try to design a multi-keyword VRSSE scheme to achieve conjunctive keyword searches.

Acknowledgment

This work was supported in part by NSFC grants 61632009 and 61402161; NSF grants CNS 1449860, CNS 1461932, CNS 1460971, CNS 1439672, CNS 1301774, and ECCS 1231461; the Hunan Provincial Natural Science Foundation of China (Grant 2015JJ3046); and the Open Foundation of State Key Laboratory of Networking and Switching Technology (Beijing University of Posts and Telecommunications) under Grant SKLNST-2016-2-20.

References

[1] X. Liu, Q. Liu, T. Peng, and J. Wu, "Dynamic access policy in cloud-based personal health record (PHR) systems," *Information Sciences*, 2017.

[2] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky, "Searchable symmetric encryption: improved definitions and efficient constructions," in *Proc. of CCS*, 2006.

[3] K. Kurosawa and Y. Ohtaki, "Uc-secure searchable symmetric encryption," in *Proc. of FC*, 2012.

[4] K. Kurosawa and Y. Ohtaki, "How to update documents verifiably in searchable symmetric encryption," in *Proc. of CNS*, 2013.

[5] S. Kamara, C. Papamanthou, and T. Roeder, "Dynamic searchable symmetric encryption," in *Proc. of ACM CCS*, 2012.

[6] S. Kamara and C. Papamanthou, "Parallel and dynamic searchable symmetric encryption," in *Proc. of FC*, 2013.

[7] N. Cao, C. Wang, M. Li, K. Ren, and W. Lou, "Privacy-preserving multi-keyword ranked search over encrypted cloud data," *IEEE Transactions on Parallel and Distributed Systems*, , 2014.

[8] C. Wang, N. Cao, K. Ren, and W. Lou, "Enabling secure and efficient ranked keyword search over outsourced cloud data," *IEEE Transactions on Parallel and Distributed Systems*, 2012.

[9] 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 Transactions on Parallel and Distributed Systems*, 2014.

[10] 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 Transactions on Computers*, 2016.

[11] J. Camenisch and A. Lysyanskaya, "Dynamic accumulators and application to efficient revocation of anonymous credentials," in *in Proc. of CRYPTO*, 20026.

[12] A. De Caro and V. Iovino, "jPBC: Java pairing based cryptography," in *Proc. of IEEE ISCC*, 2011.