Catch You If You Lie to Me: Efficient Verifiable Conjunctive Keyword Search over Large Dynamic Encrypted Cloud Data

Wenhai Sun*†, Xuefeng Liu*, Wenjing Lou†, Y. Thomas Hou† and Hui Li*
*The State Key Laboratory of Integrated Services Networks, Xidian University, Xi'an, Shaanxi, China
†Virginia Polytechnic Institute and State University, Blacksburg, VA, USA

***Abstract*—Encrypted data search allows cloud to offer fundamental information retrieval service to its users in a privacy-preserving way. In most existing schemes, search result is returned by a *semi-trusted* server and usually considered authentic. However, in practice, the server may malfunction or even be *malicious* itself. Therefore, users need a result verification mechanism to detect the potential misbehavior in this computation outsourcing model and rebuild their confidence in the whole search process. On the other hand, cloud typically hosts large outsourced data of users in its storage. The verification cost should be efficient enough for practical use, i.e., it only depends on the corresponding search operation, regardless of the file collection size. In this paper, we are among the first to investigate the efficient search result verification problem and propose an encrypted data search scheme that enables users to conduct secure conjunctive keyword search, update the outsourced file collection and verify the authenticity of the search result efficiently. The proposed verification mechanism is efficient and flexible, which can be either delegated to a *public* trusted authority (TA) or be executed *privately* by data users. We formally prove the *universally composable* (UC) security of our scheme. Experimental result shows its practical efficiency even with a large dataset.**

I. Introduction

While cloud computing provides unparalleled benefits to its users in a "pay-as-you-go" manner, such as on-demand computing resource configuration, ubiquitous and flexible access, considerable capital expenditure savings, etc., security concern is still the major inhibitor of cloud adoption for many large companies, organizations and individuals [1]. Encrypting sensitive data before uploading them to cloud storage, e.g., Google Drive, Dropbox, etc., can avoid user privacy breach, but the obfuscated data thwart the cloud to quickly sort out intended information as per user-selected keywords of interest.

In the literature, encrypted data search is proposed to address the above challenges, but the majority of these schemes [2], [3], [4], [5], [6], [7], [8] assume that the cloud server is *semi-trusted*. In other words, the server will not deviate from the designated protocol and return erroneous search result. This assumption is usually insufficient in the real world due to the underlying software/hardware malfunctions, financial incentives (cloud server may intentionally save computational resources and return false result), or even the existence of a *malicious* server controlled by an outside attacker, etc. Therefore, cloud users may desire a more trustworthy secure search system beyond the *semi-trusted* model, i.e., they can be assured of the authenticity of returned search result in a more challenging scenario where a fully *malicious* cloud server exists. Furthermore, the result verification cost should be minimal and affordable to users irrespective of the outsourced large data collection. Otherwise it will not be of practical value considering the dramatically increasing number of resource-constrained mobile devices.

On the other hand, a preferred *verifiable* search scheme should be constructed without sacrificing other critical search functionalities. One of these is *conjunctive keyword search* [3], [6], [7], [8], i.e., it allows the cloud server to produce search result containing all the queried keywords within one search operation. This multi-keyword search capability not only boosts search efficiency, but also improves the overall user experience. Moreover, a practical scheme should also work for *dynamic data* [4], [5], [9], [10], i.e., search can be conducted even after *inserting*, *deleting*, or *modifying* a file, which is specially appealing to users who would like to update their files while retaining the encrypted data search functionality without rebuilding the whole system from scratch.

In this paper, aiming to provide all the above search functionalities in a challenging *malicious* model while preserving search privacy, we propose an efficient verifiable conjunctive keyword search scheme (VCKS) over large dynamic encrypted cloud data. We use the *inverted index* structure [2], [11], [10] to build our secure index and allow data user to delegate her search task to a cloud server. We exploit the *bilinear-map accumulator* [12], [13] technique to construct an authenticated data structure. As such the user can verify the returned search result either *privately* by herself or with the assistance of a *public* TA. The proposed VCKS scheme also supports file collection update, i.e., *insertion*, *deletion* and *modification*. Finally, the extensive experimental evaluation shows the efficiency and practicality of our scheme. Our contributions can be summarized as follows:

1) To the best of our knowledge, our proposed VCKS scheme is the first secure search solution, which supports *conjunctive keyword search*, *dynamic data update* and *search result verification* simultaneously.

2) We evaluate the performance of the scheme with large real-world dataset and show that it is efficient enough for practical use. The verification cost merely depends on the corresponding search operation, irrespective of the size of the

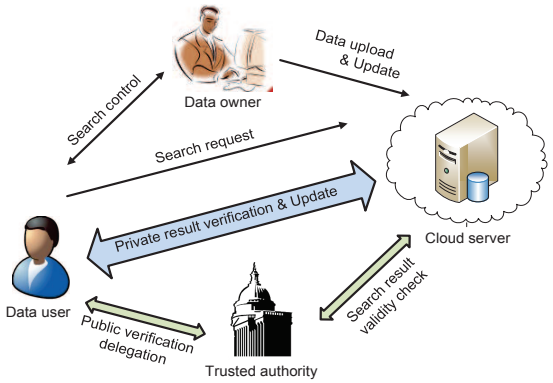


Fig. 1. System model.

searched data collection. Furthermore, the verification mechanism is flexible in the sense that it can be either delegated to a *public* TA or be executed *privately* by a data user.

3)We formally prove that our proposed VCKS scheme is *UC-secure* against a *malicious* adversary.

II. Related Work

Secure search technique has been achieved in both symmetric [2], [3], [4], [5], [6], [7], [8] and asymmetric [14], [15], [18] settings with a variety of search functionalities investigated in the literature.

Static Search. In the symmetric setting, Curtmola et al. [2] proposed an efficient secure single-keyword search scheme, and gave a formal security notion, i.e., *security against chosen-keyword attack* (CKA1) and a stronger notion of *adaptive security against chosen-keyword attack* (CKA2). To enrich the search functionality, secure multi-keyword search was realized in [3], [6] (conjunctive keyword search) and [7], [8] (conjunctive and disjunctive keyword search). Furthermore, Sun et al. [8] improved the search efficiency and accuracy using a tree-based index structure and the *cosine similarity measure* in the *vector space model*. In the public key scenario, Boneh et al. [14] presented the first public key encryption with keyword search scheme constructed from identity-based encryption. Recently, Sun et al. [15] proposed the first attribute-based keyword search scheme to realize fine-grained owner-enforced search authorization. Note that the above schemes only support *static data*, and are secure against a *semi-trusted* server.

Dynamic Search. Goh [16] proposed a dynamic secure search scheme but the bloom filter based index may introduce false positive into the final search result. Chang et al. [17] also presented a dynamic search solution with linear search time. Kamara et al. [4] proposed a dynamic version of [2], supporting data insertion and deletion on the outsourced dataset, and proved it CKA2-secure. Later they accelerated the search process by using parallelization technique [5]. However, these works *will not* be secure against a *malicious* adversary, and users cannot verify the authenticity of returned search result.

Verifiable Search. Wang et al. [19] use the hash chain to verify the single keyword search result. In [20], a verifiable logarithmic-time search scheme was presented to support range queries. Kurosawa et al. proposed the first UC-secure verifiable search scheme with single keyword [11] and extended it to a dynamic version [10] later. For static data, Sun et al. proposed the first verifiable multi-keyword (conjunctive and disjunctive) search with hash and signature techniques in [21] and later presented a verifiable attribute-based keyword search in [22]. Stefanov et al. [9] recently gave a dynamic encrypted data search scheme with small search privacy leakage, which enables result verification for single keyword search. It is worth noting that most of these search verification mechanisms are heuristic constructions without evaluating the practical performance, especially for large-scale dataset stored in the cloud. In addition, no scheme can achieve *conjunctive*, *dynamic*, and *publicly/privately verifiable* search at the same time as shown in Tab. I.

TABLE I
Comparison of verifiable search solutions

Scheme	Query type	Dynamism	Verifiability	PPE[1]
[19]	single	static	private	no
[20]	range	dynamic	private	no
[11]	single	static	private	no
[10]	single	dynamic	public/private	no
[21]	conjunctive[2]	static	private	no
[22]	conjunctive	static	private	no
[9]	single	dynamic	private	no
This paper	conjunctive	dynamic	public/private	yes

[1] PPE = Practical performance evaluation.
[2] This work also supports disjunctive keyword search.

III. Problem Formulation

Our proposed VCKS scheme consists of three main entities: *data owner*, *data user*, and *cloud server*, as shown in Fig. 1. Data owner first prepares ciphertexts $C = \{c_1, c_2, ..., c_n\}$ for the file collection $F = \{f_1, f_2, ..., f_n\}$ of size n by using any secure encryption algorithm, such as AES. She also generates an encrypted index with a pre-defined dictionary $\mathcal{W} = \{w_1, w_2, ...w_m\}$ containing m keywords and verification related data for these files. Then data owner uploads all the above information to cloud server. Later she can update the server-hosted file collection arbitrarily, i.e., file *insertion*, *deletion* or *modification*. Authorized data users are able to obtain a search token from data owner for multiple keywords of interest and other auxiliary information via the search control mechanism [2], which is outside the scope of this paper. On receiving the search token from data user, server performs the *conjunctive keyword search* over the secure index of C. Our scheme supports both *private* and *public* search result verification as shown in Fig. 1. For the latter, data user can offload the computational burden of verification to a *public* TA. In this case, server returns the result and its proof to the TA. The TA will send the result to the user if it is valid. Otherwise, it notifies the user of its rejection.

A. Definition of VCKS

We give the definition of our scheme in the following.

Definition 1: (Verifiable conjunctive keyword search). A verifiable conjunctive keyword search scheme for dynamic data is a tuple (Setup, Enc, GenTree, GenToken, Search, GenProof, UpdToken, Update, Verify, Dec) of ten polynomial-time algorithms such that:

- $(K, s, pub) \leftarrow \mathsf{Setup}(1^\lambda)$: On input a security parameter λ, this probabilistic algorithm outputs secret keys K, s, and other public parameters pub.
- $(\gamma, C) \leftarrow \mathsf{Enc}(K, \delta, F)$: On input a secret key K, an index δ and a set of files F, this probabilistic algorithm outputs an encrypted index γ, and a set of ciphertexts C.
- $T \leftarrow \mathsf{GenTree}(s, \delta, C)$: On input a secret key s, an index δ and ciphertexts C, this deterministic algorithm outputs an accumulation tree T (c.f. Sect. IV).
- $\tau_Q \leftarrow \mathsf{GenToken}(K, Q)$: On input K and an intended keyword set $Q = \{w_{j_1}, w_{j_2}, ..., w_{j_t}\} \subseteq \mathcal{W}$, this (possibly probabilistic) algorithm outputs a search token τ_Q.
- $C(Q) \leftarrow \mathsf{Search}(\gamma, \tau_Q, C)$: On input an encrypted index γ, a search token τ_Q and ciphertexts C, this deterministic algorithm outputs the search result $C(Q)$ for the file list L_Q, where each ciphertext c_i contains all the intended keywords in Q and its identifier i is included in L_Q.
- $\Pi \leftarrow \mathsf{GenProof}(C(Q), T, pub)$: On input a search result $C(Q)$, an accumulation tree T for the file storage and public parameters pub, this deterministic algorithm outputs a proof Π.
- $\tau_u \leftarrow \mathsf{UpdToken}(u, f_i, d(v))$: On input an update operation $u \in \{$*modify, insert, delete*$\}$, a file f_i and corresponding digests $d(v)$ for nodes v in T, this (possibly probabilistic) algorithm outputs an update token τ_u.
- $(\gamma', C', T') \leftarrow \mathsf{Update}(\gamma, C, T, \tau_u)$: On input an encrypted index γ, a set of ciphertexts C, an accumulation tree T and an update token τ_u, this deterministic algorithm outputs new γ', C', and T'.
- $(accept, reject) \leftarrow \mathsf{Verify}(C(Q), \Pi, d(r), pub)$: On input a search result $C(Q)$, a proof Π, a root digest $d(r)$ from data owner, and parameters pub (also the secret key s in the case of private verification), this deterministic algorithm outputs $accept$ if the search result is valid; else, it outputs $reject$.
- $f \leftarrow \mathsf{Dec}(K, c)$: On input K and a file ciphertext c, this deterministic algorithm outputs a plaintext file f.

B. Security Definition

*1) **Privacy**:* Almost all the existing secure search schemes [2], [3], [4], [5], [7], [8], [9], [11], [10] leak *search pattern*, i.e., whether the same keyword was used for search in the past or not, and *access pattern*, i.e., after searching keywords in Q, the file list L_Q is disclosed. In practice, these privacy information cannot be preserved efficiently. Thus, we in this work do not aim to protect them. Similar to [5], we define two stateful leakage functions $\mathcal{L}_1$ and $\mathcal{L}_2$ to precisely capture what is being revealed by ciphertext and the tokens: 1) $\mathcal{L}_1(\delta, F)$. On input the index δ and the file collection F, this function outputs the dictionary size $|\mathcal{W}|$, the file collection size $|F|$, the file identifiers i and its size $|i|$, and the size of each file $|f_i|$. For update, this function also reveals the identifiers and/or the size of the corresponding files; 2) $\mathcal{L}_2(\delta, F, Q)$. Given the index δ, the file collection F, and the keyword set Q searched in the past, this leakage function reveals *search* and *access patterns*.

We adapt the *privacy* definition in [10] to a dynamic conjunctive keyword search setting, where a *semi-trusted* server is considered. Note that this security definition is slightly stronger than CKA2 security defined in [2], [4], [5].

Definition 2: (Privacy). For a dynamic conjunctive keyword search scheme as given in Def. 1, we consider the following experiments, where $\mathcal{A}$ is a stateful adversary, $\mathcal{S}$ is a stateful simulator, and $\mathcal{L}_1$ and $\mathcal{L}_2$ are stateful leakage functions.

$\mathbf{Real}_{\mathcal{A}}(\lambda)$: The challenger runs $\mathsf{Gen}(1^\lambda)$ to generate a key K. $\mathcal{A}$ sends a tuple (F, δ) to the challenger and receives $(\gamma, C) \leftarrow \mathsf{Enc}(K, \delta, F)$. The adversary makes a polynomial number of queries by picking $q \in \{Q, (u, i)\}$. If $q = Q$ is a search query then the adversary receives a search token $\tau_Q \leftarrow \mathsf{GenToken}(K, Q)$ from the challenger. If $q = (u, i)$ the adversary receives from the challenger an update token $\tau_u \leftarrow \mathsf{UpdToken}(u, i)$. Finally, $\mathcal{A}$ outputs a bit b.

$\mathbf{Ideal}_{\mathcal{A},\mathcal{S}}(\lambda)$: $\mathcal{A}$ chooses a tuple (F, δ). Given $\mathcal{L}_1(\delta, F)$, simulator $\mathcal{S}$ outputs a tuple (γ, C) and returns it to $\mathcal{A}$. In the search phase, the adversary makes a polynomial number of queries by picking $q \in \{Q, (u, i)\}$. If $q = Q$ is a search query, reveal $\mathcal{L}_2(\delta, F, Q)$ to $\mathcal{S}$ and return τ_Q generated by $\mathcal{S}$ to $\mathcal{A}$. If $q = (u, i)$, $\mathcal{S}$ is given the updated output of $\mathcal{L}_2(\delta, F, \{w_j\})$ and sends τ_u to $\mathcal{A}$. Finally, $\mathcal{A}$ outputs a bit b in this experiment.

We say that our dynamic conjunctive keyword search scheme in Def. 1 satisfies *privacy* if there exists a probabilistic polynomial-time (PPT) simulator $\mathcal{S}$ such that for any PPT adversary $\mathcal{A}$, $|Pr[\mathbf{Real}_{\mathcal{A}}(\lambda) = 1] - Pr[\mathbf{Ideal}_{\mathcal{A},\mathcal{S}}(\lambda) = 1]|$ is negligible.

*2) **Verifiability**:* Due to possible data corruption, software/hardware malfunctions, and even the existence of a *malicious* server in the system, search result returned to the user may be false or contain errors. The data user should be able to detect such misbehavior to guarantee the validity of the search operation. Specifically, given a valid search result $C(Q)$ and its proof Π for a search token τ_Q , the adversary $\mathcal{A}$ wins if she can forge invalid $C^*(Q)$ and Π^* that will pass the Verify algorithm. We have the following definition.

Definition 3: (Verifiability). A verifiable and dynamic conjunctive keyword search scheme in Def. 1 satisfies *verifiability* if for any PPT adversary $\mathcal{A}$, the probability of successfully forging search result and its proof is negligible for any fixed $(F, \mathcal{W}, \gamma)$ and search tokens τ_Q.

*3) **UC-Security**:* The security of a protocol proven in a stand-alone setting is preserved under composition if it is secure in the universally composable security framework [23]. In the UC framework, an environment $\mathcal{Z}$ exists to produce all the input and read all the output in the system, and arbitrarily interacts with an adversary $\mathcal{A}$. We say a protocol securely realizes a given functionality $\mathcal{F}$ if for any adversary $\mathcal{A}$, there exists an ideal-world adversary $\mathcal{S}$ such that no $\mathcal{Z}$ can tell wether it is interacting with $\mathcal{A}$ and parties running the protocol, or with $\mathcal{S}$ and parties that interact with $\mathcal{F}$ in the ideal world.

We define the ideal functionality $\mathcal{F}$ of our proposed VCKS scheme in what follows.

Definition 4: (Ideal functionality $\mathcal{F}$). The adversary $\mathcal{S}$ is only given $\mathcal{L}_1(\delta, F)$ and $\mathcal{L}_2(\delta, F, Q)$ in this ideal world. The ideal functionality $\mathcal{F}$ interacts with user (data owner or data user) P_1, server P_2 and adversary $\mathcal{S}$, and runs as below:

- On receiving (F, δ) from P_1, verify that it is the first upload input from P_1. If so, store (F, δ), and reveal $\mathcal{L}_1(\delta, F)$ to $\mathcal{S}$. Otherwise discard this input.
- On receiving search token τ_Q from P_1, reveal $\mathcal{L}_2(\delta, F, Q)$ to $\mathcal{S}$. If $\mathcal{S}$ returns "accept", send search result to P_1; else, send "reject" to P_1.
- On receiving update token τ_{mod} from P_1, replace corresponding file in F and reveal $\mathcal{L}_1(\delta, F)$ to $\mathcal{S}$.
- On receiving update token τ_{del} from P_1, delete corresponding file in F and reveal $\mathcal{L}_1(\delta, F)$ to $\mathcal{S}$.
- On receiving update token τ_{in} from P_1, insert corresponding file to F and reveal $\mathcal{L}_1(\delta, F)$ to $\mathcal{S}$.

IV. Preliminaries

Bilinear-map Accumulator. Bilinear-map accumulator [12], [13] is an efficient data authentication mechanism that 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. Bilinear-map accumulator can be realized using either *symmetric* or *asymmetric* pairing. For ease of illustration, we adopt the symmetric version in this paper.

Let $\mathbb{G}$ and $\mathbb{G}_T$ be two cyclic multiplicative group with the same prime order p. g is a generator of $\mathbb{G}$. Thus, a bilinear pairing is defined as $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$ with the properties of *bilinearity*, *computability* and *non-degeneracy*. To construct a bilinear-map accumulator, we generate an accumulation value $acc(L) = g^{\prod_{a_i \in L}(a_i+s)}$ in $\mathbb{G}$ for a set L of n elements $\{a_1, a_2, ..., a_n\}$ in $\mathbb{Z}_p^*$, where $s \in \mathbb{Z}_p^*$ is a randomly chosen value and $\prod_{a_i \in L}(a_i + s)$ is a *characteristic polynomial* for the set L. For any subset $L' \subseteq L$, a witness $Wit_{L',L} = g^{\prod_{a_i \in L - L'}(a_i+s)}$ can be produced. Subsequently, the subset test can be carried out by checking

$$e(g^{\prod_{a_i \in L'}(a_i+s)}, Wit_{L',L}) \stackrel{?}{=} e(acc(L), g). \tag{1}$$

Note that only given corresponding elements a and $\{g^{s^i} : 0 \leq i \leq q\}$ where q is an upper bound on n, $g^{\prod(a+s)}$ can be constructed with polynomial interpolation [24]. The security of the bilinear-map accumulator is derived from the q-strong bilinear Diffie-Hellman (q-SBDH) assumption [25].

This data structure can also support update operation. For example, to insert a new element a_{n+1} into the set L, we can obtain a new set accumulation value $acc'(L) = acc(L)^{(a_{n+1}+s)}$, and $acc'(L) = acc(L)^{(a_i+s)^{-1}}$ is an updated accumulation value after deleting some element a_i from L.

Accumulation Tree. To support efficient integrity check over multiple sets in one data structure, we extend bilinear-map accumulator to a *collision-resistant* accumulation tree [25]. Specifically, suppose there are m sets $\{L_1, ..., L_j, ..., L_m\}$, for each of which $acc(L_j)$ is computed. By choosing a constant $0 \leq \epsilon \leq 1$, a tree T can be generated with $l = \lceil 1/\epsilon \rceil$ levels and m leaves. Each leaf node v represents a particular set L_j in the set collection. It stores the accumulation value $acc(L_j)$ and $d(v) = acc(L_j)^{(s+j)}$ (this proves that L_j refers to $acc(L_j)$). Each internal node v of this constant-height tree T has degree $O(m^\epsilon)$ and contains the hash $d(v)$ of a set of its children $N(v)$, where $d(v) = g^{\prod_{u \in N(v)}(s+h(d(u)))}$ and $h : \mathbb{G} \rightarrow \mathbb{Z}_p^*$ is a collision-resistant hash function. Hence, the integrity of the set is protected by its accumulation value and the accumulation tree protects the integrity of all the accumulation values stored in the leaves. For instance, Fig. 2 shows a 2-degree accumulation tree of 2 levels for sets L_1, L_2, L_3 and L_4 by selecting $\epsilon = 0.5$.

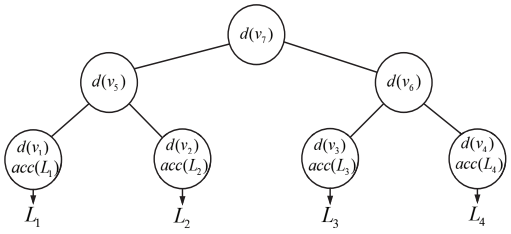


Fig. 2. Example of an accumulation tree with $\epsilon = 0.5$.

Not only does an accumulation tree support update operation on dynamic data collection, which is inherent from bilinear-map accumulator, it also can be used to verify set operations, such as set intersection. More precisely, given t queried sets $\{L_{j_1}, L_{j_2}, ..., L_{j_t}\}$, the intersection set $I = L_{j_1} \cap L_{j_2} \cap ... \cap L_{j_t}$ should satisfy the following two conditions.

- **Subset**: $I \subseteq L_{j_1} \cap I \subseteq L_{j_2} \cap ... \cap I \subseteq L_{j_t}$;
- **Completeness**: $(L_{j_1} - I) \cap (L_{j_2} - I) \cap ... \cap (L_{j_t} - I) = \emptyset$.

To meet the first requirement, the verifier only needs to check Eq.1. As for completeness condition, suppose $A_{j_b}(s)$ is the characteristic polynomial of set $L_{j_b} - I$ for $1 \leq b \leq t$. We need find another t polynomials $P_{j_b}(s)$ such that $\sum_{b=1}^{t} P_{j_b}(s) A_{j_b}(s) = 1$, which can be computed efficiently by Euclidean algorithm. Thus we obtain the completeness witnesses $Cwit_{I,L_{j_b}} = g^{P_{j_b}(s)}$ accordingly. Given the subset witnesses $Wit_{I,L_{j_b}} = g^{A_{j_b}(s)}$, we say the completeness condition is satisfied if the following equation holds:

$$\prod_{b=1}^{t} e(Cwit_{I,L_{j_b}}, Wit_{I,L_{j_b}}) \stackrel{?}{=} e(g, g). \tag{2}$$

V. Our Construction

By indexing the dataset using inverted index structure [10], we design our VCKS scheme with an efficient result verification mechanism that can be realized in both *public* and *private* settings. The index $\delta = \{a_{j,i}\}$ in our scheme is an $m \times n$ matrix as shown in Fig. 3 such that if f_i contains the keyword w_j, then $a_{j,i} = 1$; otherwise set $a_{j,i} = 0$. We denote δ_j as the j^{th} row of δ. In what follows, we begin to describe our proposed VCKS scheme in terms of system level operations,

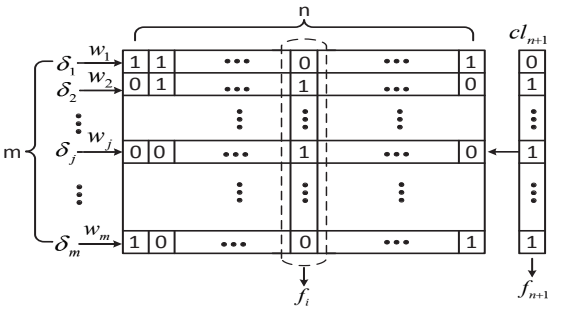


Fig. 3. Illustration for matrix index δ and insertion operation for f_{n+1}.

i.e., **Data Upload**, **Search**, **Data Download**, **Update**, where each operation may contain one or more algorithms in Def. 1.

A. Data Upload

In this initial operation, the data owner generates a secret key set $K = (k_1, k_2, k_3)$ by calling the algorithm Setup, where k_1 and k_2 are keys of a pseudorandom function prf_k, and the key k_3 is for the secure symmetric-key encryption algorithm Enc and decryption algorithm Dec shared with data user. The data owner uses the algorithm Enc to encrypt the file collection F into a ciphertext set C. She then prepares the secure index $\gamma = \{(\pi_{\varphi(j)}, \delta'_{\varphi(j)})_{1 \le j \le m}\}$ as follows:

1) For each keyword $w_j \in \mathcal{W}$, compute $\pi_j = prf_{k_1}(w_j)$;
2) Set δ'_j equal to the first n bits of $\delta_j \oplus prf_{k_2}(w_j)$;
3) Apply a random permutation φ on $\{1, ..., m\}$.

The Setup algorithm also outputs a secret key s, and public parameters $pub = \{p, \mathbb{G}, \mathbb{G}_T, g, e, h, g^s, g^{s^2}, ..., g^{s^q}\}$, where $q = max\{m, n\}$. Subsequently, the data owner generates an accumulation tree T for index δ using the algorithm GenTree. For a leaf node v of T pointing to a ciphertext set $C(i)_j = \{c_i | a_{j,i} = 1\}$ associated with δ_j, compute digest $d(v) = acc(C(i)_j)^{(\pi_j+s)} = g^{\prod_{c_i \in C(i)_j}(h(i,c_i)+s)(\pi_j+s)}$ and store it in this leaf node. Otherwise, let $d(v) = g^{\prod_{u \in N(v)}(h(d(u))+s)}$. $d(r)$ is the digest on the root node r of T.

The data owner retains the secret key s and all the node digests $d(v)$ for $v \in T$. Then she uploads the file ciphertexts C and the secure index γ along with the accumulation tree T to the cloud server.

B. Search

For a set Q of t intended keywords $\{w_{j_1}, ..., w_{j_t}\}$ from data user, the data owner calls the algorithm GenToken to obtain the search token $\tau_Q = \{(\alpha_{j_b} = prf_{k_1}(w_{j_b}), \beta_{j_b} = [prf_{k_2}(w_{j_b})]_{1...n})\}$ for $1 \le b \le t$, where β_{j_b} is the first n bits of $prf_{k_2}(w_{j_b})$, and returns it to the user.

After receiving the search token from the data user, the Search algorithm is invoked by the cloud server. More precisely, it identifies each tuple $(\pi_{\varphi(j_b)}, \delta'_{\varphi(j_b)})$ in the secure index γ if $\pi_{\varphi(j_b)} = \alpha_{j_b}$. Next, the cloud server is able to recover $\delta_{j_b} = \delta'_{\varphi(j_b)} \oplus \beta_{j_b}$ for $1 \le b \le t$. Finally, the search result $C(Q)$ can be derived by performing intersection operation on sets $\{\delta_{j_1}, \delta_{j_2}, ..., \delta_{j_t}\}$, where $c_i \in C(Q)$ contains all t keywords of interest in Q.

The server also prepares the proof Π for (public) result verification with the GenProof algorithm as below:

- Accumulation value $acc(C(i)_{j_b})$ and Π_{j_b} for each index row δ_{j_b}. Let $v_0, v_1, ..., v_l$ be the path in T from the leaf node v_0 associated with $acc(C(i)_{j_b})$ to the root node $v_l = r$. Set $\psi_z = g^{\prod_{u \in N(v_z) \setminus v_{z-1}}(h(d(u))+s)}$ for $z = 1, 2, ..., l$. As such, Π_{j_b} is defined as $\{(d(v_0), \psi_1), (d(v_1), \psi_2), ..., (d(v_{l-1}), \psi_l)\}$;
- Subset witness $Wit_{C(Q),C(i)_{j_b}}$ and completeness witness $CWit_{C(Q),C(i)_{j_b}}$, for $b = 1, 2, ..., t$;
- Coefficients $\sigma_0, \sigma_1, ..., \sigma_\rho$ of the characteristic polynomial for $\{h(i, c_i)\}$[1], where $c_i \in C(Q)$ and ρ is the size of the search result;
- The root node digest $d(r)$.

The cloud server returns all the encrypted files $C(Q)$ identified in L_Q and the proof Π.

C. Data Download

The data user first verify the search result in either *public* or *private* setting.

Algorithm 1: Public Search Result Verification

Input: Search result $C(Q)$, proof Π, root node digest $d(r)$ from data user and system parameters pub.
Output: "accept" or "reject".

```
Check d(r) in Π with that from data user, and the correctness of coefficients σ_0, σ_1, ..., σ_ρ. If any one fails output "reject", otherwise continue;
for b = 1 → t do
    Check e(d(v_0), g) ?= e(acc(C(i)_{j_b}), g^{π_{j_b}} g^s) (3);
    for z = 1 → l − 1 do
        check e(d(v_z), g) ?= e(ψ_z, g^{h(d(v_{z−1}))} g^s) (4);
    end
    Check e(d(r), g) ?= e(ψ_l, g^{h(d(v_{l−1}))} g^s) (5);
    If any one of the equations 3, 4 and 5 fails output "reject", otherwise continue;
end
Check subset condition by Eq. 6. If it fails output "reject", otherwise continue;
Check completeness condition by Eq. 2. If it fails output "reject", otherwise continue;
If none of the above fails, output "accept";
```

1) Public verifiability: The data user delegates the verification task to a *public* TA. In this scenario, the cloud server returns the search result and its proof to the TA. With only access to public parameters, i.e., g^{s^i}, the TA calls the algorithm Verify to verify the search result as illustrated in Algorithm 1. Note that the user also needs to send the latest root node digest $d(r)$ acquired from the data owner to the TA in order to

[1]Given the roots of the polynomial, we can compute its coefficients efficiently by polynomial interpolation [24]. Accumulation value, subset witness and completeness witness can also be constructed by using the corresponding coefficients and public parameters $g, g^s, ..., g^{s^q}$.

facilitate the result verification (line 1)[2]. Given search result $C(Q)$, the coefficients can be verified efficiently (line 1) [25]. By checking equations 3, 4 and 5 (line 3, 5 and 7 respectively), we can guarantee that the index row δ_{j_b} is associated with the ${j_b}^{th}$ leaf node of T. To check the subset condition for all the corresponding $C(i)_{j_b}$ in a batch manner (line 10), we make use of the equation below

$$e(\prod_{\iota=0}^{\rho}(g^{s^\iota})^{\sigma_\iota}, \prod_{b=1}^{t} Wit_{C(Q),C(i)_{j_b}}) \stackrel{?}{=} e(\prod_{b=1}^{t} acc(C(i)_{j_b}), g), \quad (6)$$

where g^{s^i} is from the public parameters pub. If the algorithm outputs "accept", $C(Q)$ is indeed the search result with respect to the queried keyword set Q. the TA will send it to the data user. The user then decrypts c_i to f_i by calling the algorithm Dec. Otherwise, the TA notifies the user of the rejection.

Algorithm 2: Private Search Result Verification

Input: Search result $C(Q)$, proof Π (exclusive of the coefficients $\sigma_0, \sigma_1, ..., \sigma_\rho$), root node digest $d(r)$ from data owner and system parameters pub.

Output: "accept" or "reject".

1 Check $d(r)$ in Π with that from data owner. If it fails output "reject", otherwise continue;
2 **for** $b = 1 \rightarrow t$ **do**
3 Check $e(d(v_0), g) \stackrel{?}{=} e(acc(C(i)_{j_b}), g^{(\pi_{j_b}+s)})$ (7);
4 **for** $z = 1 \rightarrow l-1$ **do**
5 check $e(d(v_z), g) \stackrel{?}{=} e(\psi_z, g^{(h(d(v_{z-1}))+s)})$ (8);
6 **end**
7 Check $e(d(r), g) \stackrel{?}{=} e(\psi_l, g^{(h(d(v_{l-1}))+s)})$ (9);
8 If any one of the equations 7, 8 and 9 fails output "reject", otherwise continue;
9 **end**
10 Check subset condition by Eq. 10. If it fails output "reject", otherwise continue;
11 Check completeness condition by Eq. 2. If it fails output "reject", otherwise continue;
12 If none of the above fails, output "accept";

2) Private verifiability: In case the TA is unreachable or does not even exist, we are able to achieve more computationally efficient *private* search verification by giving the secret key s to legitimate users as shown in Algorithm 2. The cloud server directly returns the result and its proof to the user. The proof Π does not include the coefficients $\sigma_0, \sigma_1, ..., \sigma_\rho$. Note that with secret key s, equations 7, 8, 9 and 10 can be computed more efficiently than their counterparts in the *public* Verify algorithm. The subset condition can be verified by the following equation:

$$e(g^{\prod_{c_i \in C(Q)}(h(i,c_i)+s)}, \prod_{b=1}^{t} Wit_{C(Q),C(i)_{j_b}})$$
$$\stackrel{?}{=} e(\prod_{b=1}^{t} acc(C(i)_{j_b}), g). \quad (10)$$

D. Update

In a dynamic cloud storage, the data owner is able to *modify*, *insert* or *delete* files arbitrarily.

1) Modify: This update operation only results in a modified version f_i' of the original file f_i and has the file identifier i unchanged. Suppose that f_i' has the same keywords with f_i. Hence, the data owner does not need to update the secure index. By the algorithm UpdToken, the data owner acquires the update token $\tau_{mod} = (i, c_i', \{d'(v)\})$. c_i' is the ciphertext of f_i' and $\{d'(v)\}$ is the modified digest set computed as follows. For the path $v_0, v_1, ..., v_l$ from a leaf node v_0 containing c_i' to root node $v_l = r$, update $d'(v_0) = d(v_0)^{(h(i,c_i)+s)^{-1}(h(i,c_i')+s)}$ and set $d'(v_z) = d(v_z)^{(h(d(v_{z-1}))+s)^{-1}(h(d'(v_{z-1}))+s)}$ for $z = 1, ..., l$. On receiving this update request from the owner, the cloud server updates the ciphertext set and accumulation tree.

2) Delete: To delete a file f_i from the storage, the data owner adopts the UpdToken algorithm to generate an update token $\tau_{del} = (i, \{d'(v)\})$. The deletion operation is analogue to file modification except that for each leaf node containing c_i, set corresponding $d'(v_0) = d(v_0)^{(h(i,c_i)+s)^{-1}}$. Finally the server uses the Update algorithm to delete the original ciphertext c_i, and produce a new accumulation tree T'.

3) Insert: To insert a new file f_{n+1} into current file collection, the algorithm UpdToken generates a new $(n+1)^{th}$ column cl_{n+1} for the matrix index δ as shown in Fig. 3. For $1 \leq j \leq m$, $a_{j,n+1} = 1$ if the file contains keyword w_j; let $a_{j,n+1} = 0$ otherwise. Then cl_{n+1} is obfuscated to cl'_{n+1} by $a_{j,n+1} \oplus [prf_{k_2}(w_j)]_{n+1}$ and apply the random permutation φ to cl'_{n+1}, where $[prf_{k_2}(w_j)]_{n+1}$ is the $(n+1)^{th}$ bit of $prf_{k_2}(w_j)$. The owner encrypts f_{n+1} to c_{n+1} by Enc. With the related new leaf node digests $d'(v_0) = d(v_0)^{(h(n+1,c_{n+1})+s)}$, an updated digest set $\{d'(v)\}$ can be computed. The corresponding update token τ_{in} is a tuple $(n+1, c_{n+1}, cl'_{n+1}, \{d'(v)\})$, which allows the cloud server to update the file collection, the secure index and accumulation tree by calling the Update algorithm.

Remark. Data owner may keep a set of succinct file stubs $h(i, c_i)$ after **Data Upload** operation for update efficiency. The storage overhead is negligible compared with the size of F. Otherwise, she need sign them and interact with server every time the Update algorithm is triggered. In the *private* setting, data user with secret key s and file digest information from data owner can also update the file collection. This is a desirable feature in the case that the outsourced dataset is allowed to be written by multiple group users.

VI. Security Analysis

In this section, we analyze the security properties of our proposed scheme and show that it achieves the defined security

[2]The data owner can also sign this digest with a time stamp to guarantee the freshness of the search result, but the TA (or user in the private setting) needs additional cryptographic operations to verify this signature.

goals. We first prove that our VCKS scheme satisfies *privacy* in Def. 2 with a *semi-trusted* server (adversary). After incorporating the *verifiability* in Def. 3, our final scheme achieves the stronger notion of security, namely *UC-security* against a *malicious* adversary (c.f. Sect. III.B).

Theorem 1: The VCKS scheme satisfies *privacy* in Def. 2.

Proof: Let $\mathcal{A}$ and $\mathcal{S}$ be an adversary and a simulator in $\mathbf{Ideal}_{\mathcal{A},\mathcal{S}}(\lambda)$ in Def. 2, respectively. Given the leakage function $\mathcal{L}_1(\delta, F)$, $\mathcal{S}$ outputs (γ', C') as follows. It simulates the encrypted file $c_i = \mathsf{Enc}_{k_1}(0^{|f_i|})$ for $i = 1, ..., n$, where k_1 is randomly selected for the CPA-secure encryption algorithm Enc, and $|f_i|$ is revealed by $\mathcal{L}_1$. To simulate the secure index $\gamma = \{(\pi_{\varphi(j)}, \delta'_{\varphi(j)})\}$ for $j = 1, ..., m$, $\mathcal{S}$ sets π_j as a random number and chooses $\delta_j \in \{0,1\}^n$ at random. $\mathcal{S}$ then applies a random permutation φ on $\{1, ..., m\}$ and sends (C', γ') to $\mathcal{A}$.

Adversary $\mathcal{A}$ can make a polynomial number of queries by picking $q \in \{Q, (u, i)\}$. If q is a search query for a keyword set Q of t conjunctive keywords $\{w_{j_b}\}_{b=1,...,t}$, the leakage function $\mathcal{L}_2(\delta, F, Q)$ reveals L_Q to $\mathcal{S}$. Given this, for $b = 1, ..., t$ $\mathcal{S}$ can generate $a_{j_b,i} = 1$ if $i \in L_Q$; otherwise, set $a_{j_b,i} = 0$. Then $\mathcal{S}$ sets $\alpha_{j_b} = \pi_{\varphi(j_b)}$ and computes $\beta_{j_b} = \delta'_{\varphi(j_b)} \oplus (a_{j_b,1}, ..., a_{j_b,n})$. She returns $\tau'_Q = \{(\alpha_{j_b}, \beta_{j_b})\}$ to $\mathcal{A}$. If $q = (u, i)$ is an update query: 1) $u = modify$. Given $|f'_i|$ from leakage function, $\mathcal{S}$ simulates $c'_i = Enc_{k_1}(0^{|f'_i|})$. Then $\mathcal{S}$ sends $\tau'_{mod} = (i, c'_i)$ to $\mathcal{A}$; 2) $u = delete$. $\mathcal{S}$ returns $\tau'_{del} = i$ to $\mathcal{A}$; 3) $u = insert$. With $|f_{n+1}|$ from $\mathcal{L}_1(\delta, F)$, $\mathcal{S}$ computes $c_{n+1} = Enc_{k_1}(0^{|f_{n+1}|})$. Choose $cl'_{n+1} \in \{0,1\}^m$ and apply the random permutation φ on it. Then $\mathcal{S}$ sends $\tau'_{in} = (n+1, c_{n+1}, cl'_{n+1})$ to $\mathcal{A}$.

The adversary $\mathcal{A}$ cannot distinguish C' from C in experiment $\mathbf{Real}_{\mathcal{A}}(\lambda)$ since Enc is CPA-secure. Due to the pseudorandom function prf used in $\mathbf{Real}_{\mathcal{A}}(\lambda)$ $\mathcal{A}$ cannot distinguish γ' from γ either. Likewise, $\mathcal{A}$ cannot tell the differences between $\{\tau'_Q, \tau'_{mod}, \tau'_{del}, \tau'_{in}\}$ in $\mathbf{Ideal}_{\mathcal{A},\mathcal{S}}(\lambda)$ and $\{\tau_Q, \tau_{mod}, \tau_{del}, \tau_{in}\}$ in $\mathbf{Real}_{\mathcal{A}}(\lambda)$ because of the CPA-secure Enc, pseudorandom function prf and random permutation φ. Thus, $\mathcal{A}$ cannot distinguish $\mathbf{Real}_{\mathcal{A}}(\lambda)$ and $\mathbf{Ideal}_{\mathcal{A},\mathcal{S}}(\lambda)$. ■

Next, we prove the VCKS scheme secure against a *malicious* adversary.

Theorem 2: The proposed VCKS scheme satisfies *privacy* in Def. 2 and *verifiability* in Def. 3.

Proof: (Sketch) Similar to the proof of Theorem 1, we can prove *privacy* property. Therefore we only prove *verifiability*.

Suppose the adversary $\mathcal{A}$ can break *verifiability* with non-negligible probability for any fixed $(F, \mathcal{W}, \gamma)$ and search tokens τ_Q. $\mathcal{A}$ can produce $(C^*(Q), \Pi^*) \neq (C(Q), \Pi)$ but the *public* Verify algorithm outputs *accept*. $C(Q)$ and Π are valid search result and its proof respectively.

We will show that the probability of the above situation is negligible given the collision-resistant hash function h and the security of the accumulation tree proved in [25].

1) $C^*(Q) = C(Q)$ and $\Pi^* \neq \Pi$. If only $d^*(r) \neq d(r)$, the Verify algorithm will definitely output *reject*. If the coefficients in Π^* are computed incorrectly, the coefficient validity check will fail with high probability. If either the accumulation value or the related Π_j is incorrect, one of the equations 3, 4 and 5 will not hold with non-negligible probability. Otherwise, the subset condition by Eq. 6 or the completeness condition by Eq. 2 will fail.

2) $C^*(Q) \neq C(Q)$ and $\Pi^* = \Pi$. In this case, the probability that Verify outputs *accept* is negligible because given the coefficients in Π^* equal to those in Π and different search results, the coefficient validity check will succeed only with negligible probability. Even if it passes this check, it will not satisfy the subset condition by Eq. 6 or the completeness condition by Eq. 2.

3) $C^*(Q) \neq C(Q)$ and $\Pi^* \neq \Pi$. If $d^*(r) \neq d(r)$, the Verify algorithm will output *reject*. If the integrity of the accumulation tree and the corresponding accumulation values is verified, and the coefficients in Π^* are computed correctly, $C^*(Q)$ will not satisfy the subset condition by Eq. 6 or the completeness condition by Eq. 2.

For the *private* Verify algorithm, we are also able to prove the *verifiability* property analogue to the above proof. ■

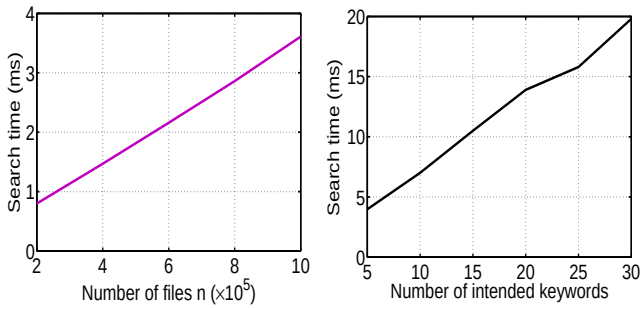
In what follows, we prove the UC-security of our scheme.

Theorem 3: The VCKS scheme is UC-secure if it satisfies *privacy* in Def. 2 and *verifiability* in Def. 3.

Proof: (Sketch) 1) If no parties are compromised by the adversary $\mathcal{A}$ in our protocol, for each keyword set Q, the user (P_1) outputs the correct search result $C(Q)$. Thus the environment $\mathcal{Z}$ cannot distinguish the real world from the ideal world since it only interacts with P_1.

2) If P_1 is corrupted by $\mathcal{A}$, then $\mathcal{A}$ can send the communication pattern of P_1 to $\mathcal{Z}$. In the ideal world, an adversary $\mathcal{S}$ can run $\mathcal{A}$ internally by playing the role of server (P_2). All messages between $\mathcal{Z}$ and $\mathcal{A}$ are forwarded by $\mathcal{S}$. $\mathcal{Z}$ cannot distinguish the real world from the ideal world since it will not interact with P_2 and $\mathcal{S}$ can play the role of P_2 faithfully.

3) If $\mathcal{A}$ corrupts P_2 and P_2 breaks *verifiability* in Def. 3 with negligible probability, the ideal world adversary $\mathcal{S}$ can run $\mathcal{A}$ internally by playing the role of P_1. All messages between $\mathcal{Z}$ and $\mathcal{A}$ are forwarded by $\mathcal{S}$. As such, $\mathcal{S}$ acts as same as the simulator in the Def. 2 and the ideal functionality $\mathcal{F}$ will reveal $\mathcal{L}_1$ and $\mathcal{L}_2$ to $\mathcal{S}$. From the proof of the Theorem 1, we can see that the inputs to $\mathcal{A}$ are distinguishable from those in the real world. In other words, $\mathcal{A}$ in the ideal world behaves as same as in the real world. On the other hand, $\mathcal{Z}$ cannot distinguish the outputs of the user in the real world and in the ideal world from the proof of the Theorem 2. For a search query, if $\mathcal{A}$ returns the valid ciphertext of the search result and its proof, the user will output correct plaintext of the search result in the real world, and in the deal world, $\mathcal{S}$ will return "accept" to $\mathcal{F}$ and $\mathcal{F}$ will send correct plaintext of the search result to P_1; otherwise, the user will output "reject", and $\mathcal{Z}$ will receive "reject" in the real world, and in the ideal world, $\mathcal{S}$ will return "reject" to $\mathcal{F}$, $\mathcal{F}$ will send "reject" to P_1 and $\mathcal{Z}$ will receive "reject" from P_1. For all the update queries, i.e., $modify$, $delete$ and $insert$, the user receives nothing from $\mathcal{A}$. Therefore, she can always update the corresponding authentication information correctly and outputs nothing. Thus, $\mathcal{Z}$ cannot distinguish the real world from the ideal world. ■



(a) (b)

Fig. 4. Search efficiency. (a) For the different size of file collection with the number of queried keywords $t = 2$. (b) For the different number of queried keywords with the number of files $n = 2 \times 10^5$.

VII. Performance Evaluation

In this section, we evaluate the performance of our proposed VCKS scheme with real-world dataset, i.e., the Enron Email Dataset [26], which consists of about half million files. To demonstrate the efficiency and effectiveness of the VCKS, we extend the size of this dataset to one million by inserting duplicates but with different file identifiers. For simplicity, we set a two-level accumulation tree with $\epsilon = 0.5$. We conduct the experiment using C and the Pairing-Based Cryptography (PBC) Library [27] on a Linux server with Intel Core i7 Processor 2.4GHz. We adopt the type A elliptic curve of 160-bit group order to realize the symmetric version of our proposed scheme, which provides 1024-bit discrete log security equivalently. Our scheme can also be implemented by any other secure asymmetric pairing technique. The experimental result is an average of 10 trials.

A. Storage Overhead

Server side. The cloud server only stores the file ciphertexts C, the secure index γ and the accumulation tree T after the Setup operation by data owner. The storage overhead of C varies a lot for different file encryption method. Thus, we do not consider it here. For the size of γ, it is mainly determined by file collection size n and dictionary size m. As shown in Tab. II, if there are one million files in the collection, the size of γ is linear to m. On the other hand, if m is fixed, the size of index is proportional to n as shown in Tab. III. Our search verification scheme is storage-efficient, because Tab. IV shows that the storage overhead of T with the fixed number of levels is only up to m, regardless of the dataset size n, and a minimal storage space suffices to host this tree structure.

Client side. The data owner and data users are all clients of the secure search system. In the *public* scenario, apart from the secret key s, data owner keeps the root node digest $d(r)$ after the accumulation tree generation phase and later sends it to users for search verification propose. She also retains the hash values $h(i, c_i)$ for all the files in C to efficiently update the file collection. In the *private* setting, data owner sends users s, $d(r)$ and $h(i, c_i)$ to enable efficient Update and *private* verification operations. The main storage overhead on the client side is to host all the hash values $h(i, c_i)$. We implement the hash function with SHA-256. Thus, for one million files, the size of their hash values are merely 32 MB.

TABLE II
Size of encrypted index with $n = 1,000,000$

m	1,000	2,000	3,000	4,000	5,000
Size of γ (MB)	125.03	250.06	375.10	500.13	625.16

TABLE III
Size of encrypted index with $m = 2,000$

n ($\times 10^5$)	2	4	6	8	10
Size of γ (MB)	50.06	100.06	150.06	200.06	250.06

TABLE IV
Size of accumulation tree with two levels

m	1,000	2,000	3,000	4,000	5,000
Size of T (KB)	66.1	103.9	195.6	260.1	324.6

B. Search Efficiency

The main computational cost for search process is to do the set intersection on t binary index vectors of size n with complexity $O(tn)$. We apply the bitwise AND operation to the queried keyword vectors $\{\delta_{j_b}\}$ for $b = 1, ..., t$. As shown in Fig. 4(a), given the same two queried keywords, time cost for search is linear to file collection size n. In Fig. 4(b), it shows that search is more time-consuming with the increased number of intended keywords. Experiment shows that our proposed VCKS scheme enables very fast conjunctive keyword search even with considerably large file collection. With a more powerful cloud server in practice, we expect that the search operation can be more efficient.

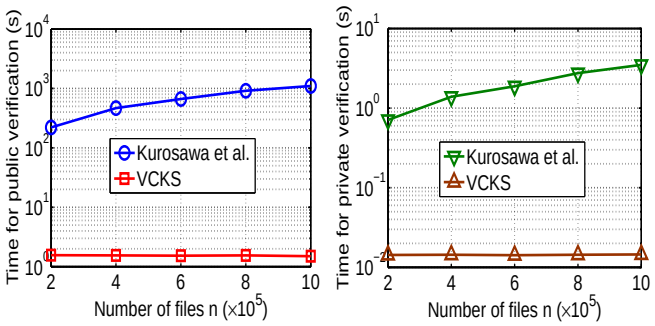
TABLE V
Time of generating an accumulation tree with $n = 1 \times 10^6$

m	1,000	2,000	3,000	4,000	5,000
Time (s)	618.95	1,237.89	1,856.84	2,475.78	3,094.73

C. Verification Efficiency

We evaluate the performance of the proposed verification mechanism in terms of the accumulation tree generation time, and result verification time in the *public* and *private* scenario.

1) Accumulation tree generation: Constructing the accumulation tree involves sum, multiplication and exponentiation operations in group G, and hash operation. Tab. V shows that with the dataset containing one million files, tree generation time is proportional to the dictionary size m. Notice that this computational burden on the data owner is a one-time cost. After the accumulation tree along with the encrypted index and dataset ciphertext is outsourced to the cloud server, the following operations, i.e., update and search verification, can be executed efficiently. Thus, the overall efficiency is totally acceptable in practice.



(a) (b)

Fig. 5. Verification efficiency with $t = 2$. (a) Public verification. (b) Private verification.

2) Search result verification: Kurosawa et al. [10] designed a verification scheme for *single* keyword search by using RSA accumulator [28], [29], which also supports *public* and *private* verifiability. However, the verification complexity there is linear in the problem size $O(tn)$. To compared with our work, we adapt their scheme to be capable of conjunctive keyword search verification, i.e., after verifying each intended *single* keyword search, user will conduct conjunctive keyword search locally by index vector intersection. As shown in Fig. 5(a) and Fig. 5(b), our scheme can be orders of magnitude faster than theirs in both *public* and *private* settings. Moreover, the verification time is almost constant irrespective of n. In fact, the verification complexity of our scheme is $O(t+\rho)$, only decided by the related search operation, where ρ is number of files in the final search result. Therefore, our scheme is more suitable and practical for conjunctive keyword search over a large number of files stored in the cloud.

VIII. Conclusion

In a more challenging *malicious* model, we propose an efficient verifiable conjunctive keyword search scheme over large dynamic encrypted cloud data. Our scheme allows file update, i.e., users can insert, delete or modify a file without affecting the effective *conjunctive keyword search* operation. Furthermore, the verification cost is only contingent on the related search operation, regardless of the file collection size. Experimental result with a large real-world dataset shows the efficiency and practicality of our scheme. We also prove that the proposed scheme is *UC-secure* against a *malicious* adversary.

Acknowledgments

This work was supported in part by the NSFC grants U1401251 and 61402352, the U.S. NSF grant CNS-1217889, and IBM GSUR.

References

[1] J. Vijayan, "Cloud security concerns are overblown, experts say", http://www.computerworld.com/s/article/9246632/Cloud_security_concerns_are_overblown_experts_say.

[2] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky, "Searchable symmetric encryption: improved definitions and efficient constructions," in *Proc. of ACM CCS*, pp. 79-88, 2006.

[3] P. Golle, J. Staddon and B. R. Waters, "Secure conjunctive keyword search over encrypted data," in *Proc. of ACNS*, pp. 31-45, 2004.

[4] S. Kamara, C. Papamanthou and T. Roeder, "Dynamic searchable symmetric encryption," in *Proc. of ACM CCS*, pp. 965-976, 2012.

[5] S. Kamara and C. Papamanthou, "Parallel and dynamic searchable symmetric encryption," in *Proc. of FC*, pp. 258-274, 2013.

[6] D. Cash, S. Jarecki, C. Jutla, H. Krawczyk, M. Rosu, and M. Steiner, "Highly-scalable searchable symmetric encryption with support for boolean queries," in *Proc. of CRYPTO*, pp. 353-373, 2013.

[7] N. Cao, C. Wang, M. Li, K. Ren, and W. Lou, "Privacy-preserving multi-keyword ranked search over encrypted cloud data," in *Proc. of IEEE INFOCOM*, pp. 829-837, 2011.

[8] W. Sun, B. Wang, N. Cao, M. Li, W. Lou, Y. T. Hou, and H. Li, "Privacy-preserving multi-keyword text search in the cloud supporting similarity-based ranking," in *Proc. of ACM ASIACCS*, pp. 71-82, 2013.

[9] E. Stefanov, C. Papamanthou, and E. Shi, "Practical dynamic searchable encryption with small leakage," in *Proc. of NDSS*, 2014, accepted.

[10] K. Kurosawa and Y. Ohtaki, "How to update documents verifiably in searchable symmetric encryption," in *Proc. of CANS*, pp. 309-328, 2013.

[11] K. Kurosawa and Y. Ohtaki, "UC-secure searchable symmetric encryption," in *Proc. of FC*, pp. 285-298, 2012.

[12] M. H. Au, P. P. Tsang, W. Susilo, and Y. Mu, "Dynamic universal accumulators for DDH groups and their application to attribute-based anonymous credential systems," in *CT-RSA*, pp. 295-308, 2009.

[13] I. Damgård and N. Triandopoulos, "Supporting non-membership proofs with bilinear-map accumulators," *Cryptology ePrint Archive*, Report 2008/538, 2008, [Online]. Available: http://eprint.iacr.org/2008/538.

[14] D. Boneh, G. D. Crescenzo, R. Ostrovsky, and G. Persiano, "Public key encryption with keyword search," In *Proc. of EUROCRYPT*, pp. 506-522, 2004.

[15] W. Sun, S. Yu, W. Lou, Y. T. Hou, and H. Li, "Protecting your right: Attribute-based keyword search with fine-grained owner-enforced search authorization in the cloud," in *Proc. of IEEE INFOCOM*, pp. 226-234, 2014.

[16] E.-J. Goh, "Secure indexes," *Cryptology ePrint Archive*, Report 2003/216, 2003, [Online]. Available: http://eprint.iacr.org/2003/216.

[17] Y.-C. Chang and M. Mitzenmacher, "Privacy preserving keyword searches on remote encrypted data," in *Proc. of ACNS*, pp. 442-445, 2005.

[18] B. Wang, Y. Hou, M. Li, and H. Li, "Maple: scalable multi-dimensional range search over encrypted cloud data with tree-based index," in *Proc. of ACM ASIACCS*, pp. 111-122, 2014.

[19] C. Wang, N. Cao, K. Ren and W. Lou, "Enabling secure and efficient ranked keyword search over outsourced cloud data," *IEEE TPDS*, vol. 23, no. 8, pp. 1467-1479, 2012.

[20] Y. Lu, "Privacy-preserving logarithmic-time search on encrypted data in cloud," in *Proc. of NDSS*, 2012.

[21] 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 TPDS*, vol. 25, no. 11, pp. 3025-3035, 2014.

[22] W. Sun, S. Yu, W. Lou, Y. T. Hou, and H. Li, "Protecting your right: Verifiable Attribute-based keyword search with fine-grained owner-enforced search authorization in the cloud," *IEEE TPDS*, PrePrint, 2014.

[23] R. Canetti. "Universally composable security: A new paradigm for cryptographic protocols," [Online]. Available: http://eprint.iacr.org/2000/067.

[24] F. P. Preparata and D. V. Sarwate, "Computational complexity of Fourier transforms over finite fields," *Mathematics of Computation*, vol. 31, no. 139, pp. 740-751, 1977.

[25] C. Papamanthou, R. Tamassia, and N. Triandopoulos, "Optimal verification of operations on dynamic sets," in *Proc. of CRYPTO*, pp. 91-110, 2011.

[26] W. W. Cohen, "Enron Email Dataset." [Online]. Available: https://www.cs.cmu.edu/~enron/

[27] "Pairing-based cryptography libray." [Online]. Available: http://crypto.stanford.edu/pbc/

[28] J. Benaloh and M. De Mare, "One-way accumulators: A decentralized alternative to digital signatures," in *Proc. of EUROCRYPT'93*, pp. 274-285, 1994.

[29] J. Camenisch and A. Lysyanskaya, "Dynamic accumulators and application to efficient revocation of anonymous credentials," in *Proc. of CRYPTO*, pp. 61-76, 2002.