Sādhanā (2020) 45:9
https://doi.org/10.1007/s12046-019-1227-5

Verifiable top-*k* searchable encryption for cloud data

B LYDIA ELIZABETH[1] and A JOHN PRAKASH[2,*]

[1]Department of Information Technology, Anna University, Chennai, India
[2]Ramanujan Computing Centre, Anna University, Chennai, India
e-mail: lydiajohn@annauniv.edu; johnprakash@annauniv.edu

MS received 19 February 2018; accepted 1 October 2019; published online 20 December 2019

Abstract. With the proliferation of data and the appealing features of cloud computing, the data owners are motivated to outsource their data to the public cloud. Privacy and security, especially for sensitive data, are still a concern, as the data owners have no physical control over the outsourced data. To ensure confidentiality, sensitive data is encrypted before outsourcing to the cloud, which obsoletes the data utilization using traditional keyword-based search. To address this issue, a verifiable top-*k* searchable encryption for cloud data (VSED) is proposed with provisions for dynamic update operations like addition and deletion of documents. Specifically, an encrypted inverted index is constructed using a secret orthogonal vector and partial homomorphic encryption. To support the ranked search, the widely used term frequency and inverse document frequency rule is used to find the top-*k* documents. To verify the query results returned by the cloud server, this scheme provides a verifiable search using keyed hashes. Security analysis demonstrates that the proposed scheme is semantically secure, with correctness and privacy guarantees proved in the standard security simulation model. Simulations performed on real-world dataset demonstrate that the proposed scheme is efficient and practical.

Keywords. Secure cloud storage; searchable encryption; dynamic search; verifiable search; top-*k* search..

1. Introduction

Cloud computing is an internet-based service model that consists of group of remote servers integrated to provide ubiquitous on demand service that can be provisioned rapidly from a shared pool of configurable resources at reduced cost [1]. Due to increase in need for computing resources, individuals and businesses outsource their computing and storage needs to the cloud [2]. The business benefit of cloud storage is undeniable and enterprises achieve effective functionality at a reduced price while improving business agility. Also, the public cloud storage option enables the Small and Medium Enterprises (SMEs) to fully focus on their core business rather than IT management [3]. However, security and privacy are the major barriers for adopting the cloud for both the large enterprises and SMEs [4]. According to a survey [5], an overwhelming majority of 91% of organizations are concerned about public cloud security. This is because the data owner has no physical control over the outsourced data.

The security of stored data is of concern, especially when outsourced data is sensitive data like health records, critical business data and credit card information to name a few [2]. The cloud server (CS) or cloud service provider (CSP) and data owners (DOs) are not in the same trusted domain, which may put the outsourced unencrypted data at risk. CSPs may leak information regarding the outsourced data to unauthorized entities or even be hacked or data could be destroyed with malicious intents. Therefore, sensitive data has to be encrypted by the DO prior to outsourcing to combat unsolicited access and to preserve privacy. Additionally, to speed up retrieval, indices are sent to the CSP, who can provide various functionalities on behalf of the DO. The commonly used indices are index per document (forward index) or index per keyword (inverted index). Inverted index is a popular data structure [6] used to speed up the search. However, the inverted index is inherently sequential. The index has to be rebuilt in order to update (addition/deletion) keywords and documents. Even if the updates are handled using a separate data structure like a delete array, the construction is complex and allows only minimal updates [7]. Second, updates leak information since the update information requires rebuilding inverted index at the CSP-owned resources.

Most of the traditional searchable encryption schemes [8] do not capture the relevance of the retrieved documents. When applied on large data such as big data or collaborative data, there are a few drawbacks. First, without foreknowledge of the encrypted data stored at the CSP, the authorized data users (DUs) have to go through every document to find the ones that match their interest. Second, especially in the pay-per-view cost model, downloading all

*For correspondence

the documents that match the search query incurs unnecessary traffic and cost. Ranked retrieval conserves bandwidth and computations, besides improving user productivity, as only the most relevant documents are retrieved. Hence, top-*k* is an essential information retrieval technique that makes economic usage of resources like bandwidth and computational resources. The working principle of public cloud storage as a service therefore relies in the exhibition of the CSPs' competence in protecting the outsourced data by providing security mechanisms. Outsourcing of data to the CSP raises new security concerns. First, since the CSP and the DO are not in the same trusted domain, the secrecy of the outsourced data is a challenge. It is required that the CSP must not learn any information from the stored data, or the way it is accessed [9]. Second, the completeness of the search, i.e. the retrieved result must include all the documents that match the search query. The search results must be verifiable, since a malicious CSP can return only fragments of the search result in order to save computational resources.

Most of the existing literature do not provide practical and fully functional searchable encryption construction. For searchable encryption to be practical, the constructions should have properties like privacy preserving, efficient ranked search and verifiable search with provisions to dynamically update the encrypted documents without index reconstruction. As an effort to address these issues, verifiable top-*k* ranked searchable encryption over encrypted cloud data (VSED) is proposed. The contributions are summarized as follows:

1. Construction of a top-*k* searchable encryption scheme based on inverted index and secret orthogonal vector to retrieve documents based on their relevance.
2. The construction supports dynamic update on encrypted index flexibly without reconstructing the entire index.
3. Verifiable search in order to verify the completeness of the search result returned by the CS.
4. Security analysis to demonstrate that VSED is semantically secure with provable trapdoor unlinkability, correctness and privacy guarantees.
5. Experimental evaluations on real-world dataset is used to show the efficiency of the proposed scheme.

1.1 *Organization*

The paper is organized as follows. Section 2 describes the related works. Section 3 gives the problem formulation, which includes the system model, system design and notations. Section 4 presents the construction of the proposed work VSED. The security model and the security analysis are given in sections 5 and 6, respectively. The performance analysis of VSED is described in section 7. Finally, section 8 concludes the paper with suggestions for future work.

2. Related works

Searchable encryption is a cryptographic primitive that has gained considerable attention recently. A number of traditional searchable encryption techniques use symmetric key setting with the focus on improvement of efficiency and formalization of security definitions. Song *et al* [10] were the first to put forward a practical solution for searchable encryption with a linear search complexity. Goh [11] proposed a secure per document index construction based on bloom filter and pseudo-random. The search complexity is proportional to the number of documents in the document set and the construction with bloom filter results in false positives. Chang and Mitzenmacher [12] presented a similar construction with an index per document with linear search time. Curtmola *et al* [13] constructed an inverted index or per keyword index and pointed a link between the index and the trapdoor. The search complexity realized is a sub-linear search. Boneh and Waters [14] were the first to present an asymmetric searchable encryption scheme. Their construction has a drawback of low search efficiency and large computation cost. The aforementioned constructions support single-keyword and Boolean search without capturing the relevance of the documents.

Multi-keyword top-k search: The common pattern used by data users to search the outsourced data is to use multiple query keywords at a time. Cao *et al* [15] introduced multi-keyword ranked search using coordinate matching and vector space model. The search complexity is a linear search and the scheme uses a deterministic trapdoor; thus, it is prone to distinguishability attack. Since then, there are numerous schemes and multi-keyword searchable encryption schemes have gained attention [16–18]. Sun *et al* [19] realized a secure multi-keyword search using a searchable index-tree based on vector space model and cosine similarity measures along with the TF–IDF to rank relevant documents. Jiang *et al* [20] present a multi-keyword ranked search using an inverted index, and a special data structure QSet to mask the correspondence between the keyword and the document set that contains the keyword. TF–IDF values are used to find the relevance scores.

Verifiable search: The search results returned by the CS may not contain complete result or can contain errors. This is possible due to malicious CS intending to save computational resource or due to software/hardware malfunction. Therefore, a mechanism to verify the completeness of the search result is desired. A number of verifiable search is based on Merkle hash tree [21–26], but suffers from huge storage overhead. Other schemes include the cryptographic signature schemes [27, 28]. Wang *et al* [29] propose a verifiable auditing scheme based on bloom filters. The scheme suffers from computational and communication overload.

Dynamic updates: In literature, very few schemes support addition or deletion of keywords in the document.

Song *et al* [10] present a dynamic scheme where the document is divided into fixed-size words and each word is individually indexed. Updating is straightforward but with varying trade-offs between security and efficiency. The construction of Goh [11] using bloom filters updates the document but suffers from false positives and linear search time. Kamara *et al* [7] presented a construction with inverted index but with complex implementation. Kamara and Papamanthou [30] proposed an index based on keyword red black tree for Boolean and single-keyword search. Cash *et al* [31] proposed a dynamic searchable encryption scheme using “T-Sets”—a data structure for the keyword/identity tuple. A separate database is used to add tuples, and deleted tuples are stored in a revocation list. The construction supports only single-keyword search. Naveed *et al* [32] proposed a dynamic searchable encryption via a blind storage that allows DOs to store dynamic collection of documents with the CS. However, the scheme leaks information on the updates. Moreover, updating the existing document by addition or deletion of keywords is not supported. Xia *et al* [33] proposed an encryption scheme based on keyword balanced binary tree and greedy depth-first search. The cost of the search and the time complexity of trapdoor are high.

3. Problem formulation

3.1 *System model*

The system model involves four entities, namely the data owner or document owner (DO), CS or the CSP, secure co-processor (SCP) and the data user (DU), as illustrated in figure 1. The SCP [34] (like the IBM PCIe or the Freescale C29x) is assumed to reside at the CSPs’ isolated execution environment. It is assumed that the CS and the SCP do not collude. The DO has a collection of document set $\mathcal{D} = \{d_1, \ldots, d_n\}$ to be outsourced to the CS. Using a secure encryption algorithm, the DO generates the encrypted document set $\mathcal{C} = \{C_1, \ldots, C_n\}$ from the document set $\mathcal{D}$. To enable searching through the document set, an encrypted searchable index per keyword (inverted index) is constructed from the document set $\mathcal{D}$. Both the encrypted document set $\mathcal{C}$ and the encrypted index $I_{\mathcal{D}}$ are outsourced to the CS. The DO computes the term frequency and inverse document frequency $\mathcal{T}$, and sends the encrypted score index $\mathcal{S}$ to the SCP. To search for the documents containing the keyword of interest, an authorised DU acquires a trapdoor corresponding to the search keyword from the DO. On receiving the trapdoor, the CSP executes search over the encrypted index $I_{\mathcal{D}}$ and retrieves all the documents that match the search query. The CSP sends the trapdoor and an optional k obtained from the DU to the SCP to compute the top-k documents. The SCP verifies and computes the scores for the query keyword using the encrypted score index $\mathcal{S}$, ranks according to its relevance to the query and returns the top-k document identifiers along with verifiable parameters to the CSP. The CSP returns the top-k documents along with the verifiable parameter to the DU.

3.2 *Design goals*

To enable dynamic, efficient and secure top-k ranked search over encrypted cloud data, VSED’s construction aims at the following design goals.

1. *Top-k multi-keyword ranked search*: The design of the scheme must accept multiple input queries from the DU

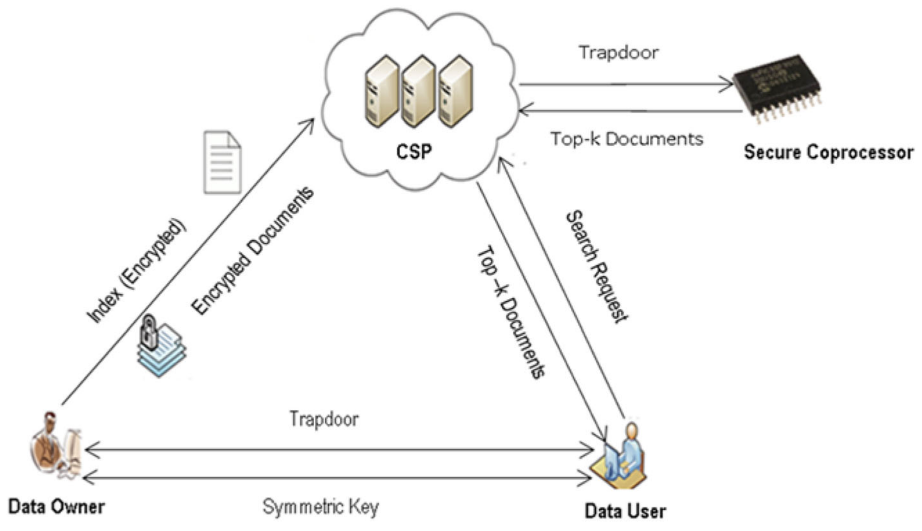

Figure 1. Architecture of VSED.

and return the top-k relevant results that match the search query.

2. *Dynamic index*: The scheme should have provisions for dynamic update, i.e. both addition and deletion of documents, without reconstructing the index.
3. *Search and update efficiency*: The scheme should be accomplished with sub-linear search and update time.
4. *Verifiable search*: The scheme should support the verifiability of the completeness of the results returned from the CS.
5. *Privacy preserving*: The construction should not leak any information (access pattern, history, search pattern, trace) to the CS beyond what is allowed.

4. Construction of VSED protocol (π_V)

The construction of the VSED scheme aims to securely and publicly perform computations with encrypted data or to modify the encrypted data using special functions in such a way that they are computed while preserving the privacy. Such encryption mechanisms are called homomorphic cryptosystems. Cryptosystems like RSA, Paillier and ElGammal exhibit partial homomorphic properties. VSED makes use of the Paillier cryptosystem's homomorphic addition property [35]. The preliminary set-up and related algorithms required for construction of VSED are described in this section. The construction of the VSED scheme consists of the following algorithms.

- *KeyGen*: A probabilistic polynomial-time algorithm that generates secret parameters for encryption and index construction. The KeyGen algorithm generates secret parameter for standard symmetric encryption and homomorphic Paillier cryptosystem.
- *Enc*(*pk*, *m*, *r*): A probabilistic polynomial-time algorithm that takes a secret parameter, message and a random number as input to produce a cipher text.
- *Dec*(*sk*, *c*): A probabilistic polynomial-time algorithm that takes a secret parameter and cipher text as input to output the original plain text.
- $BuildScoreIndex(\mathcal{D}, \mathcal{W}, pk, \mathcal{V})$: This algorithm is executed by the DO, which takes the data item $w \in \mathcal{W}$, public key pk and secret vector set $\mathcal{V}$ as inputs, and outputs a score index $\mathcal{S}$.
- $BuildIndex(\mathcal{D}, \mathcal{W}, pk, \mathcal{V})$: This algorithm is executed by the DO, which takes the keyword $w \in \mathcal{W}$, the public key pk and secret vector set $\mathcal{V}$ as inputs, and outputs an encrypted index $I_{\mathcal{D}}$.
- $TrapDoor(w_q, pk, \mathcal{V})$: This algorithm takes a keyword $w_i \in \mathcal{W}$, public key pk and secret vector set $\mathcal{V}$, and generates the trapdoor T_{w_q} to be used for searching a keyword in the encrypted index.
- $Search(I_D, T_{w_q}, k)$: This algorithm takes an index $I_{\mathcal{D}}$, a trapdoor T_{w_q} and k, the number of top matching documents, to retrieve and outputs the set of top-k file identifiers denoted by $\mathcal{F}_{qk} = \{f_i\}_{\forall f_i \in w_q}$.
- $topk(k, \mathcal{E}_{\mathcal{F}q}, sk, T_{w_q}, \mathcal{S})$: This algorithm takes a value k, encrypted binary index vector $windex_q$ (computed by the search algorithm and derived from $\mathcal{E}_{\mathcal{F}q}$), secret parameter sk, encrypted score index $\mathcal{S}$ and a trapdoor T_{w_q}, generated by the DO, and outputs the top-k document identifiers that match the query keyword.
- $Update(pk, I_D, \mathcal{S}, f_u, \mathcal{V})$: This algorithm takes an index $I_{\mathcal{D}}$, trapdoor to update Y_{w_q}, the public key pk, secret vector set $\mathcal{V}$ and document or keyword to be updated as input, and outputs an updated index and score index. $AddKeyword(pk, I_D, Y_{w_u}, \mathcal{S}, f_u, \mathcal{V})$, for adding new keywords, and $DeleteKeyword(pk, I_D, Y_{w_u}, \mathcal{S}, f_u, \mathcal{V})$, for deleting keywords from the encrypted index, are the two algorithms that constitute update.

4.1 *Set-up*

Let $\mathcal{D} = d_1, \ldots, d_n$ where $\mathcal{D}$ denotes the set of documents and d_i denotes the i^{th} document in the set of documents. Let $\mathcal{C} = \{C_1, ..., C_n\}$ denote the set of encrypted documents such that $C_i = Enc(sk_{STD}, d_i)$. Any standard symmetric key encryption algorithm proven to be a pseudo-random function like AES is assumed for encrypting the files or documents. The operation of encrypting the files is denoted as $Enc(sk_{STD}, d_j)$, where sk_{STD} denotes the secret key of such encryption algorithm and d_j denotes the document to be encrypted. Let $\mathcal{F} = f_1, \ldots, f_n$ denote the file identifiers of the encrypted documents and each f_i is given by $f_i = id(C_i)$. Let $\mathcal{W} = \{w_1, .., w_{|\mathcal{W}|}\}_{\neq}$ be the set of distinct keywords of the document collection $\mathcal{D}$.

4.2 *Key generation*

Key generation is a probabilistic polynomial-time algorithm that generates the secret parameters used for encryption and building index. The VSED algorithm uses Paillier cryptosystem for generation of trapdoor and index. Hence, its key generation set-up is assumed.

Let $n = pq$ for primes p and q; set $\lambda = lcm(p-1, q-1)$, and choose $g \in Z^*_{n^2}$ such that $\gcd(L(g^{\lambda} \bmod n^2), n) = 1$, where $L(x) = (x-1)/n$. Then $P = R = Z_n, C = Z^*_n$ and $K = (n, g, p, q, \lambda)$, where n, g, p, q and λ are defined earlier. Given security parameter ε, the algorithm chooses two distinct $\varepsilon/2-$bit primes p and q, sets $n = pq$ and $\lambda = lcm(p-1, q-1)$, and chooses $g \in Z^*_{n^2}$ such that $\gcd(L(g^{\lambda} \bmod n^2), n) = 1$. The tuple($n$, g) is the public key denoted by pk, and the tuple (p, q, λ) is a private key denoted by sk.

The Principle of Orthogonality states that two vectors $X, Y \in R$ are orthogonal or perpendicular if $X.Y = 0$. Moreover, $X_1, .., X_p \in R^n$ are mutually orthogonal if

$X_i.X_j = 0$ whenever $i \neq j$. A set of mutually orthogonal vectors is called an orthogonal set. Mutually orthogonal unit vectors $v_1, ..., v_p \in R^n$ are said to be orthonormal. Let $\mathcal{V}$ be a set of mutually orthogonal vectors given by $\mathcal{V} = v_1, v_2, \ldots, v_{|\mathcal{W}|}$ where v_i denotes the i^{th} row vector and $|\mathcal{W}|$ denotes the number of keywords in the keyword set $\mathcal{W}$. The row vectors in the set $\mathcal{V}$ exhibit mutual orthogonal properties such that $\{v_i.v_i = 1 : v_i = v_i, v_i \in \mathcal{V}\}$ and $\{v_i.v_j = 0 : v_i \neq v_j, v_i, v_j \in \mathcal{V}\}$.

Alternatively, $v_1, v_2, \ldots, v_p$ is called an orthonormal set. A square $n \times n$ matrix H with elements ± 1 that satisfies $H \times H^T = nI_n$ is called a Hadamard matrix of order n [36]. The Hadamard matrices exhibit good orthogonal properties. The Hadamard matrix can be generated by choosing p Hadamard arrays $HA_1, HA_2, .., HA_p$ each of size, say, $e_i \times e_i$ for $1 \leq i \leq p$ where each e_i is either 2, 4 or 8. Later, construct $e_1 e_2 \ldots e_p$-sized matrix HA_M by the tensor product of these p matrices given as

$$\mathcal{V} = H_M = H_1 \otimes H_2 \otimes \ldots \otimes H_p. \tag{1}$$

4.3 *Encryption*

Given a message $m \in P$ and a public key $pk = (n, g), Enc(pk, m, r)$ chooses a random $r \in Z_n^*$ such that $\gcd(r, n) = 1$ and returns the cipher text given by

$$c = g^m r^n \bmod n^2. \tag{2}$$

4.4 *Decryption*

Given a ciphertext $c \in \mathcal{C}$ and a private key $sk = (p, q, \lambda)$, decryption $Dec(sk,\ c)$ returns the message

$$m = \frac{(L(c^\lambda \bmod n^2))}{L(g^\lambda \bmod n^2) \bmod n}. \tag{3}$$

4.5 *BuildScoreIndex*

The document score is computed based on term frequency (*tf*) and inverse term frequency (*idf*). The product of *tf* and *idf* is denoted by $\mathcal{T}$.

Term frequency is the frequency of a keyword w_i for the document d_j and is given by

$$tf(w_i, d_j) = \text{frequency of } w_i. \tag{4}$$

The inverse document frequency of a keyword w_i for the document collection $\mathcal{D}$ is given by

$$idf(w_i, \mathcal{D}) = \log\left(\frac{|\mathcal{D}|}{|d_j \in \mathcal{D} : w_i \in d_j|}\right). \tag{5}$$

Inverse document frequency of a keyword w_i over the document collection $\mathcal{D}$, denoted by $idf(w_i, \mathcal{D})$, is the log of the ratio of the total number of documents in the document collection, represented as $|\mathcal{D}|$, to the number of documents in the document collection containing the keyword w_i, represented as $|d_j \in \mathcal{D} : w_i \in d_j|$. The term-inverse document score of a keyword w_i for the document d_j is given by

$$\mathcal{T}(w_i, d_j) = tf(w_i, d_j) idf(w_i, \mathcal{D}). \tag{6}$$

4.5a *Algorithm description for* $\mathcal{S} \leftarrow$*BuildScore Index*$(\mathcal{D}, \mathcal{W}, pk, \mathcal{V})$: The algorithm description to compute the term frequency is as follows:

1. Initialize the document term frequency $tf(w_i, d_j)$ and inverted document frequency $idf(w_i, d_j)$ for all keywords and documents to be 0. Also, initialize all term-inverse document score $\mathcal{T}(w_i, d_j)$ to 0.
2. For each document $d_j \in \mathcal{D}$, increment the document term frequency for the corresponding keyword $w_i \in d_j$ as $tf(w_i, d_j) = tf(w_i, d_j) + 1$.
3. For each keyword $w_i \in \mathcal{W}$, compute the inverse document term frequency as $idf(w_i, \mathcal{D}) = \log\left(\frac{|\mathcal{D}|}{|d_j \in \mathcal{D} : w_i \in d_j|}\right)$.
4. For each keyword $w_i \in \mathcal{W}$ and document $d_j \in \mathcal{D}$, calculate $\mathcal{T}(w_i, d_j) = tf(w_i, d_j) idf(w_i, \mathcal{D})$.
5. Later, the encrypted per document score index denoted by $\mathcal{S}(d_j)$ is computed for all documents $d_j \in \mathcal{D}$ as follows:

$$\mathcal{S}(d_j) = \sum_{i=1}^{|w_i \in d_j|} v_i M_i \mathcal{T}(w_i, d_j) + v_r r' \tag{7}$$

where $v_i = \mathcal{V}(w_i)$, $M_i = Enc(pk, w_i, r_{w_i})$, $v_r = \mathcal{V}(r)$ and $(r_{w_i}, r') \in Z_n^*$ such that $\gcd(r, n) = 1$.

The diagrammatic representation of the score index is shown in figure 2.

4.6 *BuildIndex*

The index of VSED scheme denoted by I_D is computed as follows and diagrammatically shown in figure 3:

$$I_D = \sum_{i=1}^{|\mathcal{W}|} (v_i M_i S_i) + v_r r'. \tag{8}$$

4.6a *Algorithm description for* $I_D \leftarrow$ *BuildIndex* $(\mathcal{D}, \mathcal{W}, pk, \mathcal{V})$: The step by step procedure to build encrypted index is described as follows:

1. Calculate $v_i = \mathcal{V}(w_i)$ the corresponding secret vector assigned for the keyword w_i.
2. Calculate $M_i = Enc(pk, w_i, r_{w_i})$ where pk is the public key of the Paillier cryptosystem, w_i is the respective keyword and a random $r_{w_i} \in Z_n^*$ such that $\gcd(r, n) = 1$.

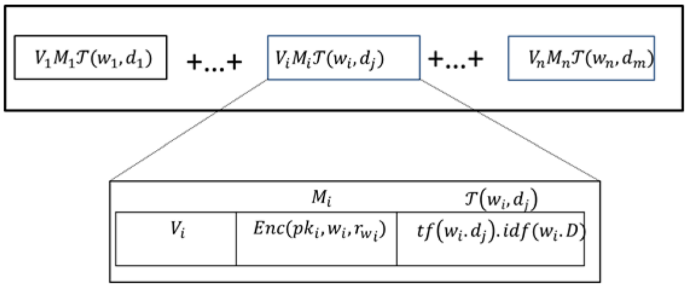

Figure 2. Score index.

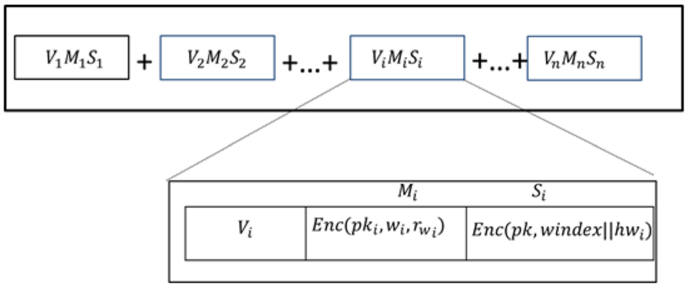

Figure 3. Index.

3. Calculate $S_i = Enc(pk, \mathcal{L}(w_i))$ where
   (a) pk is the public key of the Paillier cryptosystem,
   (b) $\mathcal{L}(w_i) = windex_i \| h_{w_i}$ where
      i. $windex_i$ is the binary vector index for the keyword w_i. The binary vector index denoted as $windex_i$ is a data vector in which a value of 1 in a position indicates the presence of a keyword in that document. For example, consider a binary vector $windex_i = 1011$ for a keyword w_i in document collection $\mathcal{D} = d_1, d_2, d_3, d_4$; then documents d_1, d_3, d_4 are said to contain the keyword w_i.
      ii. $h_{w_i} = \mathcal{H}(sk, windex_i)$ is the keyed hash of $windex_i$ for the keyword w_i.
4. Calculate $v_r = \mathcal{V}(r)$, a random vector, and r' is a random number added for randomness.
5. Repeat steps 1–4 for all keywords $w_i \in \mathcal{W}$ and cumulatively add them to obtain the index $I_\mathcal{D}$.

4.7 *Trapdoor*

The trapdoor generation of VSED can be represented as

$$T_{w_q} = v_q M_{w_q}^{-1} + v_r r' \tag{9}$$

4.7a *Algorithm description for* $T_{w_q} \leftarrow TrapDoor(w_q, pk, \mathcal{V})$: The step by step procedure to generate trapdoor is given as follows:

1. Calculate $v_{w_q} = \mathcal{V}(w_q)$, the corresponding vector for the keyword to be queried w_q.
2. Calculate $M_{w_q}^{-1} = Enc(pk, -w_q, r_q)$ where pk is the public key of the Paillier cryptosystems, $-w_q$ is the negation of the keyword to be queried, r_q is a random number chosen such that $r_q \in Z_n^*$ and $\gcd(r, n) = 1$.
3. Calculate $v_r = \mathcal{V}(r)$, the random vector, and r' is a random number added for randomness
4. Calculate the trapdoor T_{w_q} for the keyword to be queried w_q as $T_{w_q} = v_{w_q} M_{w_q}^{-1} + v_r r$'.

4.8 *Search*

The search mechanism involves multiplying the received trapdoor with the index, represented as

$$\mathcal{E}_{\mathcal{F}_q} = I_\mathcal{D} T_{w_q}. \tag{10}$$

4.8a *Algorithm description for* $\mathcal{F}_{qk} \leftarrow Search(I_\mathcal{D}, T_{w_q}, k)$: The step by step description of the search mechanism is given as follows:

1. Calculate $\mathcal{E}_{\mathcal{F}_q} = I_\mathcal{D} T_{w_q}$ and find the respective document file identifiers:
   (a) If $\mathcal{E}_{\mathcal{F}_q}$equals 0, then return null indicated by $\perp$. This happens when either the query keyword is not found in the document set (i.e., $|d_j \in D : w_i \in d_j| = 0$) or when a invalid trapdoor is submitted.

(b) If $\mathcal{E}_{\mathcal{F}_q} \neq 0$ then send $(k, \mathcal{E}_{\mathcal{F}_q}, T_{w_q})$ to the SCP invoking the top-k algorithm. The SCP returns the top-k document identifiers that match the query keyword.

2. Later, the CSP returns the corresponding documents to the user who had sent the trapdoor along with $\mathcal{F}_{qk} = \mathcal{F}'_{qk} \| h_{\mathcal{F}'_{qk}}$ where $\mathcal{F}'_{qk}$ is the set of top-k document identifiers that match the query keyword and $h_{\mathcal{F}'_{qk}} = \mathcal{H}(sk, \mathcal{F}'_{qk})$ is the keyed hash of those top-k document identifiers that match the query keyword.

Hence, it can be observed that $\mathcal{F}_{qk}$ can be retrieved without revealing the keyword and $h_{\mathcal{F}'_{qk}}$ can be used by the user to verify whether the documents returned by the CSP are correct. The correctness, verifiability and security are discussed in the security analysis subsection.

4.9 *Top-k*

The top-k ranking is computed by the SCP, ensuring that neither the CSP nor the SCP learns anything about the data involved other than the allowed information leakage described in the security model. The SCP will have the encrypted $\mathcal{S}$ score index and may have the knowledge of the number of documents in the document set that match the search query, but does not know to which keyword the scores are associated. The CSP has the encrypted index and the top-k documents only. CSP does not have the knowledge of either the keywords being searched or the information of all the documents that contain the query keyword as only top-k documents are returned to it by SCP.

4.9a *Algorithm description for* $\mathcal{F}_{qk} \leftarrow topk(k, \mathcal{E}_{\mathcal{F}q}, sk, T_{w_q}, \mathcal{S})$: The step by step process for computing the top-k documents at the SCP is as follows:

1. SCP decrypts $\mathcal{E}_{\mathcal{F}_q}$ using the private key sk to obtain $windex_i \| h_{w_i}$.
2. Calculate the hash for $windex_i$ as $h'_{w_i} = \mathcal{H}(sk, windex_i)$.
3. Verify whether the received hash h_{w_i} and the computed hash h'_{w_i} are the same.
4. If there is a mismatch $(h'_{w_i} \neq h_{w_i})$, SCP returns null represented by $\perp$. This prevents any modification or fabrication of $\mathcal{E}_{\mathcal{F}_q}$by the CSP.
5. If $h'_{w_i} = h_{w_i}$, then calculate the following:

   (a) Obtain $windex_i$ from $\mathcal{E}_{\mathcal{F}_q}$by decrypting it with sk.

   (b) Calculate the score $\mathcal{Q}(d_x)$ for all x, if $windex_i[x] = 1$, by computing $\mathcal{Q}(d_x) = \mathcal{S}(d_x)T_{w_q}$.

   (c) Sort the scores $(sort(\mathcal{Q}, k))$; obtain the first k document identifiers denoted by $\mathcal{F}'_{qk}$ and given by $\mathcal{F}'_{qk} = sort(\mathcal{Q}, k)$.
6. Calculate the hash as $h_{\mathcal{F}'_{qk}} = \mathcal{H}(sk, \mathcal{F}'_{qk})$ where $h_{\mathcal{F}'_{qk}}$ is the hash of top-k document identifiers for verification.
7. Calculate $\mathcal{F}_{qk} = \mathcal{F}'_{qk} \| h_{\mathcal{F}'_{qk}}$ where $\mathcal{F}_{qk}$ is the concatenation of the set of top-k document identifiers and the hash.
8. Send $\mathcal{F}_{qk}$ to the calling search algorithm.

4.10 *Update*

The dynamic keyword update process involves two algorithms, namely *AddKeyword* and *DeleteKeyword*.

4.10a *Algorithm description for adding keyword* $(I_{\mathcal{D}}, \mathcal{S}) \leftarrow AddKeyword(pk, I_{\mathcal{D}}, Y_{w_u}, \mathcal{S}, f_u, \mathcal{V})$: The step by step procedure to add a keyword dynamically is as follows:

1. For adding a keyword that already belongs to the set $\mathcal{W}$, (i.e., $w_u \in \mathcal{W}$) the old binary vector index has to be removed and the new binary vector index has to be added to the index:

   (a) Calculate $Y_{w_u} = v_u M_u S_u$ where w_u is the keyword whose binary vector index needs to be updated, $v_u = \mathcal{V}(w_u)$, $M_u = Enc(pk, w_u, r_u)$ and $S_u = Enc(pk, \mathcal{L}(w_u))$.

   (b) Calculate $Y'_{w_u} = v_u M_u S'_u$ where w_u is the keyword whose binary vector index needs to be updated, $v_u = \mathcal{V}(w_u)$, $M_u = Enc(pk, w_u, r_u)$ and $S'_u = Enc(pk, \mathcal{L}(w'_u))$ is the newly generated binary vector index in which the respective bit for the document is set as given by $windex'_u[f_u] = 1$ for a given keyword w_u.
2. For adding a new keyword or a keyword that does not belong to the set $\mathcal{W}$, (i.e., $w_u \notin \mathcal{W}$) the new binary vector index has to be added to the index:

   (a) Set $Y_{w_u} = 0$ and calculate $Y'_{w_u} = v_u M_u S'_u$ where w_u is the new keyword whose binary index vector needs to be added, $M_u = Enc(pk, w_u, r_u)$, $v_u = \mathcal{V}(w_u)$ and $S'_u = Enc(pk, \mathcal{L}(w'_u))$ is the newly generated binary index vector in which the respective bit for the document is set as given by $windex'_u[f_u] = 1$ for a given keyword w_u.
3. The DO sends Y_{w_u}, Y'_{w_u} to the CSP and the CSP computes $I_{\mathcal{D}} = I_{\mathcal{D}} - Y_{w_u} + Y'_{w_u}$, where $I_{\mathcal{D}}$ is the index.
4. Compute $\mathcal{S}'(d_{f_u}) \leftarrow BuildScoreIndex(\mathcal{V}, d_{f_u}, w_u)$, the new score index and send it to SCP.

4.10b *Algorithm description for deleting keyword* $(I_{\mathcal{D}}, \mathcal{S}) \leftarrow DeleteKeyword(pk, I_D, Y_{w_u}, \mathcal{S}, f_u, \mathcal{V})$: The step by step procedure to delete a keyword dynamically is as follows:

1. For deleting a keyword, the old binary index vector has to be removed from the index:

   (a) Calculate $Y_{w_u} = v_u M_u S_u$ where w_u is the keyword whose binary index vector needs to be updated, $v_u = \mathcal{V}(w_u)$, $M_u = Enc(pk, w_u, r_u)$ and $S_u = Enc(pk, \mathcal{L}(w_u))$, and set $Y'_{w_u} = 0$.

2. The DO sends $\{Y_{w_u}, Y'_{w_u}\}$ to the CSP and the CSP computes $I_{\mathcal{D}} = I_{\mathcal{D}} - Y_{w_u} + Y'_{w_u}$, where $I_{\mathcal{D}}$ is the index.
3. Compute $\mathcal{S}'(d_{f_u}) \leftarrow BuildScoreIndex(\mathcal{V}, d_{f_u}, w_u)$, the new score index, and send the same to the SCP.

4.11 *Protocol description* (π_V)

Let π_V denote VSED protocol and the description of the working of the protocol is given as follows:

1. Let P_1 be the CSP, P_2 be the DO and P_3 be the SCP. P_2 computes the collection of encrypted files $\mathcal{C}$ given by

$$\mathcal{C} \leftarrow Enc(sk_{STD}, d_1), \ldots, Enc(sk_{STD}, d_n).$$

2. P_2 invokes the *BuildIndex* algorithm to build the index for all the keywords given by

$$I_D \leftarrow BuildIndex(\mathcal{D}, \mathcal{W}, pk, \mathcal{V}).$$

3. P_2 invokes the *BuildScoreIndex* algorithm to build the document scores for the keywords given by

$$\mathcal{S} \leftarrow BuildScoreIndex(\mathcal{D}, \mathcal{W}, pk, \mathcal{V}).$$

4. P_2 sends the index, score index and the encrypted files to P_1 and document score index to P_3 given by

$$P_1 \leftarrow P_2 : (I_{\mathcal{D}}, \mathcal{C});\ P_3 \leftarrow P_2 : (\mathcal{S}).$$

5. If an user wants to search for a keyword over the secure cloud storage then P_2 generates a trapdoor and sends it to user, which then sends it to P_1, or simply it can be considered as P_2 sending it to P_1 given by

$$P_1 \leftarrow P_2 : T_{w_q} \leftarrow TrapDoor(w_q, pk, \mathcal{V}).$$

6. P_1 receives the trapdoor T_{w_q}, invokes $Search(I_{\mathcal{D}}, T_{w_q}, k)$ algorithm and computes $\mathcal{E}_{\mathcal{F}q}$.

   (a) P_1 sends $\mathcal{E}_{\mathcal{F}q}$ to P_3, which then executes the algorithm $topk(k, \mathcal{E}_{\mathcal{F}q}, T_{w_q}, \mathcal{S})$ and returns to P_1 the top-k document identifiers given by

$$P_3 \leftarrow P_1 : (\mathcal{E}_{\mathcal{F}q});\ P_1 \leftarrow P_3 : \mathcal{F}_{qk} \leftarrow topk(k, \mathcal{E}_{\mathcal{F}q}, sk, T_{w_q}, \mathcal{S}).$$

   (b) P_1 returns $\mathcal{F}_{qk}$, a set of file identifiers with the hash returned by Search to P_2, which is given by

$$P_2 \leftarrow P_1 : \mathcal{F}_{qk} \leftarrow Search(I_{\mathcal{D}}, T_{w_q}, k).$$

   (c) P_1 receives the file identifiers $\mathcal{F}_{qk}$ from P_3 and returns the top-k documents to P_2.

7. For keyword addition, P_2 generates $Y_{w_u} f_u, \mathcal{V}$, computes $AddKeyword(pk, I_{\mathcal{D}}, Y_{w_u}, \mathcal{S}, f_u, \mathcal{V})$ and then sends (Y_{w_u}, Y'_{w_u}) to P_1 to update $I_{\mathcal{D}}$. Then it computes $\mathcal{S}' \leftarrow BuildScoreIndex(d_{f_u})$ and sends to P_3, which updates $\mathcal{S}$.
8. For keyword deletion, P_2 generates $Y_{w_u} f_u, \mathcal{V}$, computes $DeleteKeyword(pk, I_{\mathcal{D}}, Y_{w_u}, \mathcal{S}, f_u, \mathcal{V})$ and then sends (Y_{w_u}, Y'_{w_u}) to P_1 to update $I_{\mathcal{D}}$. Then computes $\mathcal{S}' \leftarrow BuildScoreIndex(d_{f_u})$ and sends to P_3, which updates $\mathcal{S}$.

5. Security analysis

The construction of the security model in VSED assumes an adversarial entity denoted by $\mathcal{A}$ that controls the CSP and can attack the execution of the protocol denoted by π_V. The parties under the control of the adversary are said to be corrupted and follow the adversary's instructions. A secure protocol is said to withstand any adversarial attack, assuming the adversary is computationally bounded. Therefore, to formally claim and prove that a protocol is secure, an accurate definition of secure protocol execution is required.

5.1 *Model*

The problem of cloud storage is similar to a two-party protocol problem where the two parties are CSP and DO. A two-party problem is defined by specifying a random process that maps pairs of inputs to pairs of outputs, one from each party. Generally, such a process is referred to as a functionality and denoted as $\mathrm{f} : \{0,1\}^* \times \{0,1\}^* \rightarrow \{0,1\}^* \times \{0,1\}^*$, where $\mathrm{f} : \{0,1\}^* \rightarrow (\mathrm{f}_1, \mathrm{f}_2)$. For every pair of inputs (x, y), the output is a vector of random variables $(\mathrm{f}_{P_1}(x,y), \mathrm{f}_{P_2}(x,y))$ where $\mathrm{f}(P_1)$ and $\mathrm{f}(P_2)$ are received by entities P_1 and P_2, respectively. The same process is also sometimes represented as $(x,y) \leftarrow (\mathrm{f}_{P_1}(x,y), \mathrm{f}_{P_2}(x,y))$ where the entity P_1 is the CSP and P_2 is the DO. Unlike the general two-party problem defined earlier, in the secure cloud storage problem considered here, only P_2 receives the specified output from P_1 such that P_1 learns nothing about the input. P_2 does not output anything back to P_1. This is formalized as one-sided simulation by [36], and P_1 is assumed to be corrupted.

The standard security simulation model formalizes security in a general way as ideal world and real world. In an ideal world, it is assumed that an external trusted [38] party helps the parties to carry out their computation securely. In an ideal world, the parties involved in the two-party computation just send their inputs to the trusted party, which computes the desired function and securely passes the correct prescribed output to each party, and each party receives output. Hence, the adversary can force only the corrupted parties to send their chosen inputs and cannot do anything else. Two important security properties hold in the ideal world.

1. *Privacy:* No party should learn anything more than its prescribed outputs. In particular, the only information that should be learned about other parties' inputs is what can be derived from the output itself securely. For example, while searching for documents matching a keyword over a secure cloud storage, the only information revealed to CSP should be the document identifiers that match the search query criteria without giving any information about keywords contained in the documents or in the search query. Privacy holds in the ideal world, because the only message ever received by a party is its output and so it cannot learn anything else from what it did not receive.
2. *Correctness:* Each party is guaranteed that the output that it receives is correct. For example, while searching for documents matching a keyword over a secure cloud storage, the output should be the correct document identifiers that match the desired keywords, which should be guaranteed. In ideal world, correctness holds since the trusted party cannot be corrupted and thus it will always compute the function correctly.

The adversary considered in this security model is static malicious polynomial time such that honest parties remain honest throughout, while the corrupted parties remain corrupted. The adversary may arbitrarily deviate from the protocol specification. A detailed description of the execution in ideal and real model can be found in [38–40].

Let $\texttt{Real}_{\pi,A(z),i}(x,y,k)$ denote the output of the honest party and the adversary $\mathcal{A}$ (controlling P_i) after a real execution of protocol π where P_1 has input x, P_2 has input y, $\mathcal{A}$ has an auxiliary input z and the security parameter is k.

Let $\texttt{Ideal}_{f,S(z),i)}(x,y,k)$ be the analogous distribution in an ideal execution with a trusted party who computes f for the parties. Also, let $\texttt{View}^{\mathcal{A}}_{\pi,A(z),i)}(x,y,k)$ denote the view of the adversary after a real protocol execution of π_V as before. Then, the following definition describes the security model [26].

5.1a *Computational indistinguishability:* Let the security parameter be denoted as k. A function $\mu(.)$ is negligible if for every polynomial $p(.)$ there exists a value N such that for all $k > N$, $\mu(k) < 1/p(k)$ holds. Let $X = \{X(a,k)\}_{k\in N, a\in 0,1^*}$ and be the distribution ensembles. Then, it can be said that X and Y are computationally indistinguishable, denoted $X\overset{c}{\equiv}Y$, if for every non-uniform distinguisher D there exists a negligible function $\mu(.)$ such that for every $a \in \{0,1\}^*$

$$|Pr[D(X(a,k)) = 1] - Pr[D(Y(a,k)) = 1]| < \mu(k). \quad (11)$$

5.1b *Privacy:* Let f be a functionality where only P_2 receives output. A protocol π_V is said to securely compute f with one-sided simulation if the following holds:

1. For every non-uniform probabilistic polynomial-time adversary $\mathcal{A}$ controlling P_2 in the real model, there exists a non-uniform probabilistic expected polynomial-time adversary S for the ideal model, such that

$$\{\texttt{Real}_{\pi,\mathcal{A}(z),2}(x,y,k)\}\overset{c}{\equiv}\{\texttt{Ideal}_{f,S(z),2}(x,y,k)\} \quad (12)$$

where $|x| = |y|$, $x,y,z \in \{0,1\}^*, k \in N$.
2. For every non-uniform probabilistic polynomial-time adversary $\mathcal{A}$ controlling P_1

$$\{\texttt{View}^{\mathcal{A}}_{\pi_V,\mathcal{A}(z),1}(x,y,k)\}\overset{c}{\equiv}\{\texttt{View}^{\mathcal{A}}_{\pi_V,\mathcal{A}(z),1}(x,y',k)\} \quad (13)$$

where $|x| = |y'|$, $x,y,z \in \{0,1\}^*, k \in N$.

Since it is a one-sided simulation and P_1 is the only party assumed to be corrupted, proving the second part of Definition 2 is enough to guarantee privacy, meaning that P_1 learns nothing whatsoever about P_2's input.

5.1c *Search pattern* (s_p): Let s_p denote the frequency of the keyword queries searched, which is found by checking the equality between two queries. Formally, let $Q_1, Q_2, \ldots, Q_n$ be a set of n consecutive queries, and search pattern s_p is an $n \times n$ binary matrix where $s_p(i,j) = 1$ *iff* $Q_i = Q_j$.

5.1d *Access pattern* (a_p): Let a_p denote the access pattern and is a collection of tuples $a_i = (\mathcal{F}_i, Q_i)$ where $\mathcal{F}_i$ is a set of file identifiers returned for the query Q_i.

5.1e *History* (t_n): Let t_n denote the history given by $t_n(\mathcal{D}, q)$ where q, given by$Q_1, Q_2, \ldots, Q_n$, and $\mathcal{D}$, given by $d_1, d_2 \ldots, d_n$, are the set of search queries and corresponding document identifiers returned, respectively.

5.1f *Trace* $\gamma(t_n)$: The trace is the allowed information leaked to an adversary.

Let $\gamma(t_n) = \{(f_1,\ldots,f_n), (|C_1|,\ldots,|C_n|), t_n, I_{\mathcal{D}}, T_{w_q}, Y_{w_u}, \mathcal{S}\}$ where $C_i = Enc(pk, d_i)$ is the encryption of document $d_i, f_i = id(C_i)$ is the file identifier of C_i, $|C_i|$ is the size of f_i, t_n is the history, $I_{\mathcal{D}}$ is the index, $T(w_q)$ is the trapdoor(s), $Y(w_u)$ is the update trapdoor(s) and $\mathcal{L}(w_i)$ is the hash map data structure holding the per document index.

A secure protocol should guarantee that Definition 2 holds and no more information than the trace is leaked to an adversary.

5.2 *Analysis*

Theorems 7–9 prove independently the correctness of the protocol, and subsequently the privacy and correctness are proved in Theorem 10 with respect to the security model.

The scheme is said to return top-k documents correctly for a given encrypted binary data vector $\mathcal{E}_{\mathcal{F}_q}$, trapdoor T_{w_q} for a keyword $w_q \in \mathcal{W}$ and score index $\mathcal{S}$ such that

$$Pr[\mathcal{F}_{qk} \leftarrow topk(k, sk, \mathcal{E}_{\mathcal{F}_q}, T_{w_q}, \mathcal{S}) : \mathcal{F}_{qk} = f_{i_{d_j \in \mathcal{D}: w_q \in d_j}}^{i=1,..k}] = 1. \tag{14}$$

The top-k algorithm described in section 4.9 returns the top-k document denoted by $\mathcal{F}_{qk}$. The search algorithm inputs the binary vector index or binary vector such that the binary vector for an query keyword w_q contains $windex_q[d_j] = 1$ iff $w_q \in d_j$, where $j = 1, \ldots, |\mathcal{D}|$, while the top-$k$ returns $\mathcal{F}_{qk} = f_{i_{d_j \in \mathcal{D}: w_q \in d_j}}^{i=1,..k}$, the top-$k$ document file identifiers f_i such that $d_j \in \mathcal{D} : w_q \in d_j$ and the value of $i = 1, \ldots, k$. The encrypted binary vector whose bits are set satisfies the condition $d_j \in \mathcal{D} : w_q \in d_j$ and its retrieval is given by

$$\mathcal{E}_{\mathcal{F}_q} T_{w_q} = S_q = Enc(pk, \mathcal{L}(w_q)) = Enc(pk, windex_q \| h_{w_q}). \tag{15}$$

The vector $windex_q$ contains the binary data vector as described in section 4.6, such that $windex_q[x] = 1$ if and only if $d_j \in D : w_q \in d_j$. Therefore, $\forall x$, $windex_q[x] = 1$, then the algorithm described in section 4.9 computes $\mathcal{Q}(d_x) = \mathcal{S}(d_x) T_{w_q}$ where $\mathcal{Q}(d_x)$ is the document score, and from section 4.5 it can be written as

$$\mathcal{S}(d_x) = \sum_{i=1}^{|w_i \in d_j|} (v_i M_i \mathcal{T}(w_i, d_j)) + v_r r'. \tag{16}$$

From section 4.7, $T_{w_q} = v_{w_q} M_{w_q}^{-1} + v_r r'$ where $M_{w_q}^{-1} = Enc(pk, -w_q, r_q)$. Therefore, on substituting for T_{w_q}, $\mathcal{S}$, applying the principle of orthogonality and expanding:

$$\begin{aligned}
\mathcal{Q}(d_x) =& \mathcal{S}(d_x) T_{w_q} \\
=& \left(\sum_{i=1}^{|w_i \in d_x|} (v_i M_i \mathcal{T}(w_i, d_x)) + v_r r')(v_{w_q} M_{w_q}^{-1} + v_r r' \right) \\
=& (v_i M_i \mathcal{T}(w_i, d_x))(v_{w_q} M_{w_q}^{-1} + v_r r') + 0 \\
=& v_q v_q M_{w_q} M_{w_q}^{-1} \mathcal{T}(w_q, d_x) \\
=& g^{w_q} g^{-w_q} r_q^n (r_q^n)^{-1} \mathcal{T}(w_q, d_x) \\
=& \mathcal{T}(w_q, d_x).
\end{aligned} \tag{17}$$

Hence, the score of the document d_x can be retrieved correctly. Similarly, for all other documents, $\forall x$, $windex_q[x] = 1$, the top-k algorithm computes the score, sorts the score and returns the top-k scored document to the CSP. Hence, it can be said that the algorithm correctly returns the top-k documents, formally given by

$$Pr[\mathcal{F}_{qk} \leftarrow topk(k, sk, \mathcal{E}_{\mathcal{F}_q}, T_{w_q}, \mathcal{S}) : \mathcal{F}_{qk} = f_{i_{d_j \in \mathcal{D}: w_q \in d_j}}^{i=1,..k}] = 1. \tag{18}$$

The scheme is said to provide correct verifiable search for a given trapdoor T_{w_q} for a keyword $w_q \in \mathcal{W}$:

$$Pr[\mathcal{F}_{qk} \leftarrow Search(k, I_{\mathcal{D}}, T_{w_q}) : \mathcal{F}_{qk} = \perp_{w_q \notin \mathcal{W}} or f_{i_{w_q \in \mathcal{W}}}] = 1. \tag{19}$$

The search algorithm operates under two cases: $w_q \in \mathcal{W}$ and $w_q \notin \mathcal{W}$.

5.2a *Case* $w_q \notin \mathcal{W}$: The search algorithm can be written as $\mathcal{E}_{\mathcal{F}_q} = I_{\mathcal{D}} T_{w_q}$ where w_q is the keyword searched with the trapdoor T_{w_q} while $\mathcal{E}_{\mathcal{F}_q}$ is the resultant encrypted binary vector, $windex_{w_q}[d_j] = 1 : d_j \in \mathcal{D} : w_q \in d_j$. The index and the trapdoor can be written as

$$I_{\mathcal{D}} = \sum_{i=1}^{|\mathcal{W}|} (v_i M_i S_i) + v_r r'; \; T_{w_q} = v_q M_{w_q}^{-1} + v_r r' \tag{20}$$

where

$$\begin{aligned}
M_{w_q} =& Enc(pk, w_q, r_q); \; M_{w_q}^{-1} = Enc(pk, -w_q, r_q); \\
S_q =& Enc(pk, windex_q \| h_{w_q}).
\end{aligned}$$

Substituting for $I_{\mathcal{D}}$, T_{w_q} and expanding, the equation can be written as

$$\begin{aligned}
\mathcal{E}_{\mathcal{F}_q} =& I_{\mathcal{D}} T_{w_q} \\
=& \left(\sum_{i=1}^{|\mathcal{W}|} (v_i M_i S_i) + v_r r')(v_q M_{w_q}^{-1} + v_r r' \right) \\
=& 0 \\
=& \perp.
\end{aligned}$$

Since $v_i.v_q = 1$ iff $v_i = v_q$, by the principle of orthogonality, all vectors $v_i \in \mathcal{V}$ are mutually orthogonal. Therefore, when a trapdoor T_{w_q} is generated for a keyword w_q such that $w_q \notin \mathcal{W}$, then search algorithm outputs correctly null represented by $\perp$.

5.2b *Case* $w_q \in \mathcal{W}$: Similar to the previous case, the search algorithm for $w_q \in \mathcal{W}$ is given as follows:

$$\begin{aligned}
\mathcal{E}_{\mathcal{F}_q} =& I_{\mathcal{D}} T_{w_q} \\
=& \left(\sum_{i=1}^{|\mathcal{W}|} (v_i M_i S_i) + v_r r')(v_q M_{w_q}^{-1} + v_r r' \right) \\
=& (v_1 M_1 S_1 + \cdots + v_{|\mathcal{W}|} M_{|\mathcal{W}|} S_{|\mathcal{W}|})(v_q M_{w_q}^{-1} + v_r r') \\
=& (v_q M_q S_q)(v_q M_{w_q}^{-1} + v_r r') \\
=& (v_q v_q M_q M_{w_q}^{-1} S_q) + (v_q v_r M_q M_{w_q}^{-1} S_q r') \\
=& (v_q v_q M_q M_{w_q}^{-1} S_q) \\
=& g^{w_q} g^{-w_q} r_q r_q^{-1} S_q \\
=& S_q \\
=& Enc(pk, windex_q \| h_{w_q}).
\end{aligned}$$

Thus, the encrypted binary vector $\mathcal{E}_{\mathcal{F}_q}$ is retrieved correctly for the keyword w_q searched when $w_q \in \mathcal{W}$. The search algorithm then sends $\mathcal{E}_{\mathcal{F}_q}$ to the SCP for retrieval of the top-k document identifiers. From Theorem 7, it was proved that given $\mathcal{E}_{\mathcal{F}_q}$, the top-$k$ algorithm returns the top-k documents correctly. Also, the hash of the binary vector for every keyword $h_{w_q} = \mathcal{H}(sk, windex_q)$ being concatenated with the binary vector itself, the CSP cannot forge S_q sent to the SCP. Moreover, from the top-k algorithm it can be seen that $\mathcal{F}_{qk} = \mathcal{F}'_{qk}||h_{\mathcal{F}_{qk}}$, and the set of top-$k$ document identifiers also contains the hash of the top-k document identifiers sent to the user. Thus, S_q computed by CSP and $\mathcal{F}_{qk}$ computed by the SCP and returned to the user by CSP are verifiable. Therefore, it is proved that the VSED's search algorithm is said to provide correct verifiable search for a given trapdoor T_{w_q} for a keyword given by

$$Pr[\mathcal{F}_{qk} \leftarrow Search(k, I_{\mathcal{D}}, T_{w_q}) : \mathcal{F}_{qk} = \perp_{w_q \notin \mathcal{W}} or f_{i_{w_q \in \mathcal{W}}}] = 1. \tag{21}$$

The scheme is said to provide correct dynamic index update for a given update trapdoor Y_{w_q}.

The index update uses an update trapdoor to make dynamic updates. The index and the score index are updated during addition of new keyword to a document or deletion of a keyword from a document. For both cases, update trapdoor is generated. The process of adding a keyword to the encrypted index uses simple addition and deletion of vectors represented as $I_{\mathcal{D}} = I_{\mathcal{D}} - Y_{w_u} + Y'_{w_u}$, where the old sub-index is denoted by $Y_{w_u} = v_u M_u S_u$ and the new sub-index that needs to be updated in the index is denoted by $Y'_{w_u} = v_u M_u S'_u$. Therefore, it can be written as

$$\begin{aligned} I_{\mathcal{D}} =& I_{\mathcal{D}} - Y_{w_u} + Y'_{w_u} \\ =& \left(\sum_{i=1}^{|\mathcal{W}|} (v_i M_i S_i) + v_r r\right) - v_u M_u S_u + v_u M_u S'_u \\ =& (v_1 M_1 S_1 + \cdots + v_{|\mathcal{W}|} M_{|\mathcal{W}|} S_{|\mathcal{W}|}) - Y_{w_u} + Y'_{w_u} \\ =& (v_1 M_1 S_1 + \cdots + v_u M_u S'_u + \cdots + v_{|\mathcal{W}|} M_{|\mathcal{W}|} S_{|\mathcal{W}|}). \end{aligned}$$

From these equations, it can be seen that the new sub-index vector is updated in the index $I_{\mathcal{D}}$ correctly. Similar reasoning can also be applied to the case when a keyword is to be deleted. Hence, it can be said that the scheme is said to provide correct dynamic index update for a given update trapdoor Y_{w_q}.

The Paillier encryption function is a homomorphic and semantically secure cryptosystem and its security analysis can be found in [35].

Assume that the Paillier cryptosystem is semantically secure and then the protocol π_V computes securely in the presence of non-colluding static malicious adversaries in accordance with Definition 2.

Definition 2 states that a protocol securely computes in the presence of non-colluding static malicious adversaries if the view of the adversary when given y and view of the adversary when given y' are computationally indistinguishable. The definition can thus be written as

$$\{\texttt{View}^{\mathcal{A}}_{\pi_{V,\mathcal{A}(z)},1}(x, y, k)\} \overset{c}{\equiv} \{\texttt{View}^{\mathcal{A}}_{\pi_{V,\mathcal{A}(z)},1}(x, y', k)\} \tag{22}$$

where $|x| = |y| = |y'|$ and $x, y, z \epsilon \{0,1\}^*, k \in N$.

5.2c *Game*: Assume the existence of an encryption oracle as defined by [41]. Assume a challenger who generates a key pair (pk, sk) in accordance with and based on the Paillier cryptosystem key set-up instructions. The challenger publishes the key pk but keeps sk private. Adversary $\mathcal{A}$ is given access to an encryption oracle (encryption service), namely $\mathcal{E}$. The oracle can perform a polynomial bounded number of encryptions or other operations. $\mathcal{A}$ then chooses two plaintexts (or keywords) $m_0 = y$ and $m_1 = y'$ and submits to the challenger. The challenger selects a bit $a \in 0, 1$ uniformly at random, encrypts m_a as $C = Enc(pk, m_a)$ and sends C to $\mathcal{A}$. The adversary may then perform any number of additional operations or encryptions using the encryption oracle. $\mathcal{A}$ then outputs its guess for the value of a. From Definition 1, for computational indistinguishability

$$|Pr[D(X(a, k)) = 1] - Pr[D(Y(a, k)) = 1]| < \mu(k). \tag{23}$$

From [44], it can be said that Paillier encryption mechanism is semantically secure. Therefore, the advantage of $\mathcal{A}$ in playing *Game* is negligible or utmost $\mu(k)$, which means $\mathcal{A}$ can distinguish between $Enc(pk, y)$ and $Enc(pk, y')$ though $\mathcal{A}$ knows the plaintext y and y' with only negligible probability $\mu(k)$. Hence, message encrypted with $Enc(., .)$ is computationally indistinguishable. The parameters defined such as trapdoor T_{w_q}, search pattern s_p, access pattern a_p, history t_n, trace $\gamma(t_n)$ including sub-index $\mathcal{L}$ and document score index $\mathcal{S}$ are encrypted using $Enc(., .)$, making them computationally indistinguishable too.

Generally, in a real world, the parties run some protocol among themselves without the help of the trusted party, unlike the ideal world. The real world protocol executed by the parties without the trusted party is said to be secure if no adversary can do more harm in a real world protocol execution than in an execution that takes place in the ideal world. In an ideal/real simulation paradigm for any adversary carrying out a successful attack in the real world, there exists an adversary that successfully carries out the same attack in the ideal world. However, this is contradictory since successful adversarial attacks cannot be carried out in the ideal world. Therefore, it is concluded that all adversarial attacks on protocol execution in the real world must also fail. The security of the real world protocol or real protocol is established by comparing the output of a real protocol execution to the output of an ideal computation. A real protocol execution is said to emulate ideal protocol execution if the input/output distributions of the adversary and the participating parties in the real and ideal executions are essentially the same.

Hence, the parameters that constitute the view of the adversary in Definition 2 are computationally indistinguishable as given here:

$$\{\texttt{View}^{\mathcal{A}}_{\pi_{V,\mathcal{A}(z),1}}(x,y,k)\} \overset{c}{\equiv} \{\texttt{View}^{\mathcal{A}}_{\pi_{V,\mathcal{A}(z),1}}(x,y',k)\}. \qquad (24)$$

Also, the malicious adversaries are assumed to be non-colluding since P_1 and P_3 compute the top-k documents among themselves. P_3 is by design embedded with the private key sk to decrypt the encrypted binary vector $windex_x$. P_3 leaks the tuple (document, score) but not the respective keyword as the keyword is encrypted within the trapdoor, which is computationally indistinguishable. Moreover, it has to be noted that P_3 does not have in its possession the encrypted documents. Thus, P_1 knows the set of documents or top-k documents returned for a query like any other mechanisms leaking access pattern and does not learn anything more than that.

Hence, as long as the views of $\mathcal{A}$ for y and y' are computationally indistinguishable, the protocol π_V is said to securely compute in the presence of non-colluding malicious adversaries. This guarantees privacy and correctness of the protocol as per Definition 2.

6. Performance analysis

VSED is implemented usng Java in a Windows server with Intel Xeon Processor (2.30 GHz) using the real-world Enron [42] e-mail dataset. The performance of VSED is compared to that of multi-keyword ranked searchable encryption (MRSE) proposed by Cao *et al* [43]. VSED is evaluated for the efficiency of index construction, trapdoor generation, search and update of index. The efficiency comparison is given in table 1 and feature comparison is given in table 2.

6.1 *Index construction*

The index construction, a one-time operation performed by the DO, is described in sections 4.5 and 4.6. The time cost for encrypting the binary vector $\mathcal{L}(w_i)$ and the keyword M_i is dependent on the size of the dictionary $|\mathcal{W}|$. The time cost for building the index $I_{\mathcal{D}}$ is equal to the number of documents $|\mathcal{D}|$ in the document set and the number of keywords in the dictionary $|\mathcal{W}|$. In MRSE, the major computation of the index involves the splitting process and multiplication between two $(n+2)\times(n+2)$ matrix and a $(n+2)$ dimension vector where $n=|\mathcal{W}|$, the number of keywords in the document set. The size of the sub-index is linear to the size of the data vector. Therefore, the time complexity of MRSE is $O(|\mathcal{D}||\mathcal{W}|^2)$ and storage complexity is $O(|\mathcal{D}||\mathcal{W}|)$. The time complexity of VSED is $O(|\mathcal{D}||\mathcal{W}|)$ and the storage complexity is $O(|\mathcal{D}|+|\mathcal{W}|)$ ($|\mathcal{D}|$) towards score index and $|\mathcal{W}|$ for the index). Figure 4 shows the time cost for building the index $I_{\mathcal{D}}$ by varying the documents and size of the dictionary $|\mathcal{W}|=4000$. The time cost is linear with the size of the document set $|\mathcal{D}|$. Figure 5 shows the dependence of the index $I_{\mathcal{D}}$ on the dictionary. The number of keywords in the dictionary has comparatively less influence than the number of documents in the document set.

Table 1. Comparison of efficiency with other schemes.

Schemes	Index	Storage	Trapdoor	Search	Update
Cao *et al* [43]	$O(\|\mathcal{D}\|\|\mathcal{W}\|^2)$	$O(\|\mathcal{D}\|\|\mathcal{W}\|)$	$O(\|\mathcal{W}\|^2)$	$O(\|\mathcal{D}\|\|\mathcal{W}\|)$	NA
Xia *et al* [33]	$O(\|\mathcal{D}\|\|\mathcal{W}\|^2)$	$O(\|\mathcal{D}\|\|\mathcal{W}\|)$	$O(\|\mathcal{W}\|^2)$	$O(\theta\|\mathcal{W}\|\log\|N\|)$	$O(\|\mathcal{W}\|^2\log\|\mathcal{D}\|)$
VSED	$O(\|\mathcal{D}\|\|\mathcal{W}\|)$	$O(\|\mathcal{D}\|+\|\mathcal{W}\|)$	$O(1)$	$O(\mathcal{F}_{q_k}(1+\log\mathcal{F}_{qk}))$	$O(\|\mathcal{D}\|\|\mathcal{W}\|)$

$|\mathcal{D}|$: # of documents, $|\mathcal{W}|$: # of keywords, θ: no. of leaf nodes that contain one or more keywords in the query, $\mathcal{F}_{qk}$: top-k file identifiers.

Table 2. Features comparison.

Schemes	Privacy	Multi-keyword	Top-*k*	Verifiable	Dynamic
Kamara and Papamanthou [30]	Yes	No	No	No	Yes
Cao *et al* [43]	Yes	Yes	Yes	No	No
Sun *et al* [44]	No	Yes	No	Yes	No
Wang *et al* [45]	Yes	Yes	No	No	No
Ding *et al* [46]	Yes	Yes	Yes	No	No
Jiang *et al* [47]	Yes	Yes	Yes	Yes	No
VSED	*Yes*	*Yes*	*Yes*	*Yes*	*Yes*

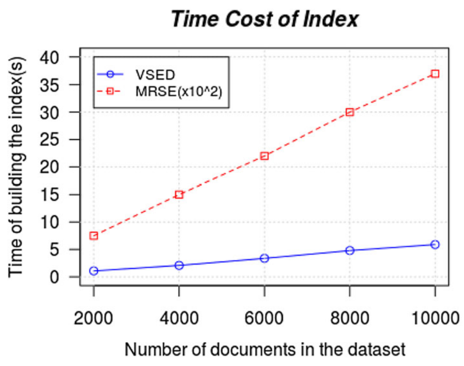

Figure 4. *BuildIndex* time vs #documents.

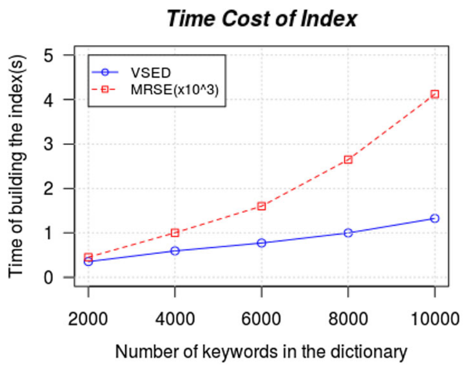

Figure 5. *BuildIndex* time vs #keywords.

6.2 *Trapdoor generation*

The trapdoor T_{w_q} for VSED is generated by the DO using the keyword of interest or the query keyword w_q. The computation involves a multiplication operation of the secret vector with $M_{w_q}^{-1}$, where $M_{w_q}^{-1} = Enc(pk, -w_q, r_q)$, and addition with a random secret vector, which is a very trivial operation. Figure 6 shows the time cost for trapdoor for $|\mathcal{D}| = 1000$ and $Q = 10$. The time cost for generation of trapdoors does not depend on the document set $|\mathcal{D}|$ or on the keyword set $|\mathcal{W}|$. However, in MRSE, the size of the dictionary influences the time cost for trapdoor as shown in figure 6. The number of query keywords w_q influences the time cost of the trapdoor. Figure 7 shows time cost of building the trapdoor with $|\mathcal{D}| = 1000$ and $|\mathcal{W}| = 4000$ by varying the query keyword. The number of query keywords w_q is varied from 5 to 25 keywords. The trapdoor computation time for VSED is significantly less when compared with MRSE. Similar to index construction, MRSE incurs a splitting vector and two matrix multiplications. The time cost for trapdoor is O(1) for VSED and O($|\mathcal{W}|^2$) for MRSE.

6.3 *Search efficiency*

The search operation is performed by the CS. The computation is a multiplication of the index $I_{\mathcal{D}}$ that is with the

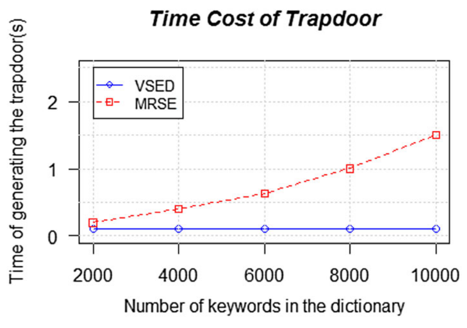

Figure 6. Trapdoor time vs #keywords.

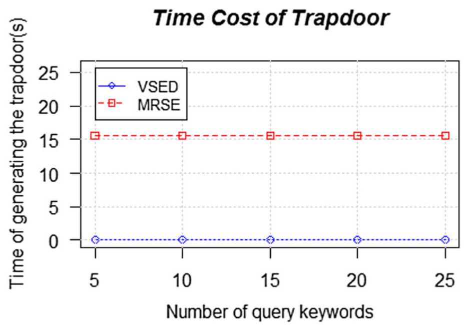

Figure 7. Trapdoor time vs # query keywords.

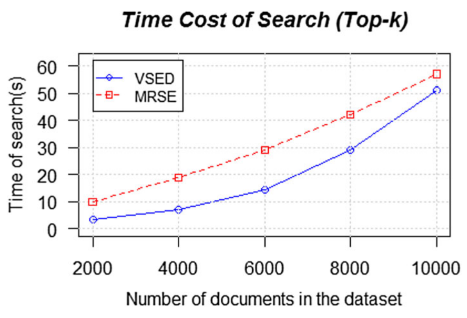

Figure 8. Search time vs #documents.

CSP with the trapdoor T_{w_q}. The computation carried by the CSP is the product of the index $I_{\mathcal{D}}$ and the trapdoor T_w. For top-k retrieval, at the SCP, the time cost for search includes the time to decrypt $\mathcal{E}_{\mathcal{F}_q}$, the file identifiers returned from the CSP, and ranking the retrieved document identifiers from the binary vector. The time complexity of search operation in VSED is constant time, i.e. O(1) without top-k and O($\mathcal{F}_{q_k}(1 + \log \mathcal{F}_{q_k})$) with top-$k$. The complexity of MRSE is O($|\mathcal{D}||\mathcal{W}|$), as similarity scores for all the documents are computed. It can be inferred that for all practical cases, $\mathcal{F}_{q_k} \ll |\mathcal{D}|$ and $\mathcal{F}_{q_k} \ll |\mathcal{W}|$. The time cost for search by varying the document and varying the query for top-k search is shown, respectively, in figures 8 and 9. In

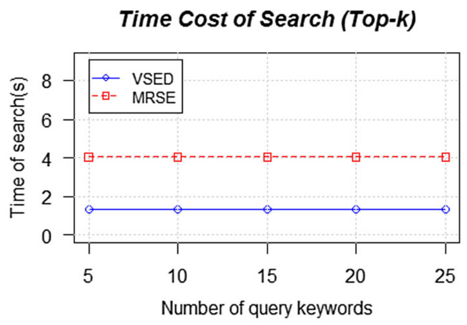

Figure 9. Search time vs # query keywords.

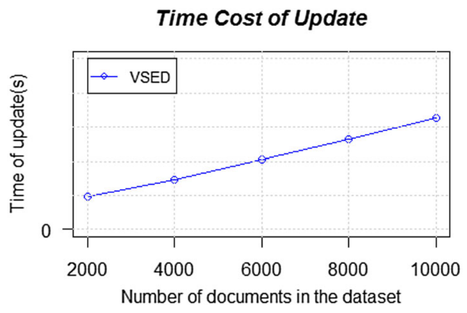

Figure 10. Time cost of update.

figure 8, it can be seen that the number of documents have less influence on search as the operation $\mathcal{E}_{\mathcal{F}_q} = I_{\mathcal{D}} T_{w_q}$ is a simple multiplication operation. However, the time for search also includes the time for scoring and sorting the documents identifiers returned by the CSP at the SCP. Figure 9 shows the time cost by keeping the documents constant $|\mathcal{D}| = 1000$, dictionary $|\mathcal{W}| = 4000$ constant and varying the query keywords. The number of query keywords has less influence on VSED and has no influence in MRSE. In MRSE, the search algorithm consists of finding the similarity scores ranking for all the documents. Thus the query time is dependent on the number of documents and the number of keywords has less influence. In VSED, if the user wishes for top-k, the k value along with the file identifier is sent to the SCP to compute the ranking. Therefore, VSED cannot be compromised by timing-based side channel attacks that try to differentiate queries based on query time.

6.4 *Update efficiency*

In order to update the document, the DO modifies the binary vector of the keyword that requires update. The trapdoor for update is sent to the CS, which is similar to the trapdoor function. The CS updates the index by removing the old binary index vector and adds the new binary vector; therefore, the time complexity is $O(|\mathcal{D}||\mathcal{W}|)$. Figure 10 shows the time cost of update with fixed number of keywords in the dictionary $|\mathcal{W}| = 4000$ for query $|Q| = 10$ and varying the documents.

7. Conclusion

In this paper we have presented a secure and verifiable top-k ranked search over encrypted cloud data with support for dynamic addition and deletion of documents. The proposed scheme uses an inverted index and a secret orthogonal vector to achieve better search and update efficiency. The security analysis demonstrates that VSED is semantically secure with completeness and correctness guarantees. The experimental evaluation indicates that the proposed the construction is efficient in terms of time and storage complexity. In the future, we will modify the proposed scheme to work efficiently and securely even without the presence of an SCP.

References

[1] Ren K, Wang C and Wang Q 2012 Security challenges for the public cloud. *IEEE Internet Comput.* 16(1): 69–73

[2] Kamara S and Lauter K 2010 Cryptographic cloud storage. In: *Proceedings of the 14th International Conference on Financial Cryptograpy and Data Security*, pp. 136–149

[3] Motahari-Nezhad H R, Stephenson B and Singhal S 2009 Outsourcing business to cloud computing services: opportunities and challenges. *IEEE Internet Comput.* (special issue)

[4] Armbrust M, Fox A, Griffith R, Joseph A D, Katz R H, Konwinski A, Lee G, Patterson D A, Rabkin A, Stoica I and Zaharia M 2009 *Above the clouds: a Berkeley view of cloud computing*. Rep. UCB/EECS 28(13), Department of Electrical Engineering and Computer Sciences, University of California, Berkeley

[5] Schulze H 2016 *Cloud security survey*, https://pages.cloudpassage.com/cloud-security-survey-report-2016.html (accessed May 2016)

[6] Bösch C, Hartel P, Jonker W and Peter A 2015 A survey of provably secure searchable encryption. *ACM Comput. Surv.* 47(2): 18

[7] Kamara S, Papamanthou C and Roeder T 2012 Dynamic searchable symmetric encryption. In: *Proceedings of CCS 2012*, pp. 965–976

[8] Wang C, Ren K, Lou W and Li J 2010 Toward publicly auditable secure cloud data storage services. *IEEE Netw.* 24(4)

[9] Wang J, Chen X, Li J, Zhao J and Shen J 2017 Towards achieving flexible and verifiable search for outsourced database in cloud computing. *Future Gener. Comput. Syst.* 67: 266–275

[10] Song D X, Wagner D and Perrig A 2000 Practical techniques for searches on encrypted data. In: *Proceedings of the IEEE Symposium on Security and Privacy*, pp. 44–55

[11] Goh E J 2003 Secure indexes. *IACR Cryptol. ePrint Archive* 216

[12] Chang Y and Mitzenmacher M 2005 Privacy preserving keyword searches on remote encrypted data. In: *Proceedings of the Third International Conference on Applied Cryptography and Network Security*, pp. 442–455

[13] Curtmola R, Garay J, Kamara S and Ostrovsky R 2011 Searchable symmetric encryption: improved definitions and efficient constructions. *J. Comput. Secur.* 19(5): 895–934

[14] Boneh D and Waters B 2007 Conjunctive, subset, and range queries on encrypted data. In: Vadhan S P (Ed.) *TCC 2007 Proceedings*. LNCS, vol. 4392, pp. 535–554. Heidelberg: Springer

[15] Cao N, Wang C, Li M, Ren K and Lou W 2011 Privacy-preserving multi-keyword ranked search over encrypted cloud data. In: *Proc. IEEE INFOCOM* 25: 829–837

[16] Swaminathan A, Mao Y, Su G M, Gou H, Varna A L, He S, Wu M and Oard D W 2007 Confidentiality-preserving rank-ordered search. In: *Proceedings of the ACM Workshop on Storage Security and Survivability*, pp. 7–12

[17] Zerr S, Olmedilla D, Nejdl W and Siberski W 2009 Zerber+R: top-*k* retrieval from a confidential index. In: *Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology*, ACM, pp. 439–449

[18] Yu J, Lu P, Zhu Y, Xue G and Li M 2013 Toward secure multikeyword top-*k* retrieval over encrypted cloud data. *IEEE Trans. Depend. Secure.* 10(4): 239–250

[19] Sun X, Zhu Y, Xia Z, Wang J and Chen L 2013 Secure keyword-based ranked semantic search over encrypted cloud data. *Adv. Sci. Technol. Lett.* 31: 271–283

[20] Jiang T, Chen X, Wu Q, Ma J, Susilo W and Lou W 2017 Secure and efficient cloud data deduplication with randomized tag. *IEEE Trans. Inf. Foren. Sec.* 12(3): 532–543

[21] Devanbu P, Gertz M, Martel C and Stubblebine S G 2002 Authentic third-party data publication. In: *Data and Application Security*. Boston, MA: Springer, pp. 101–112

[22] Kak A 2015 Elliptic curve cryptography and digital rights management. In: *Lecture Notes on Computer and Network Security*

[23] Cao N, Wang C, Li M, Ren K and Lou W 2014 Privacy-preserving multi-keyword ranked search over encrypted cloud data. *IEEE Trans. Parall. Distr.* 25(1):222-233

[24] Wang C, Cao N, Ren K and Lou W 2012 Enabling secure and efficient ranked keyword search over outsourced cloud data. *IEEE T. Parall. Distr.* 23(8): 1467–1479

[25] Basu A, Corena J, Kiyomoto S and Marsh S 2014 Privacy preserving trusted social feedback. In: *Proceedings of the 29th Annual ACM Symposium on Applied Computing*, pp. 1706–1711

[26] Zhang W, Lin Y, Xiao S, Wu J and Zhou S 2016 Privacy preserving ranked multi-keyword search for multiple data owners in cloud computing. *IEEE Trans. Comput.* 65(5): 1566–1577

[27] Yuan X, Wang X, Wang C, Squicciarini A C and Ren K 2016 Towards privacy-preserving and practical image-centric social discovery. *IEEE Trans. Depend. Secure.* 15(5): 868–882

[28] Pang H and Mouratidis K 2008 Authenticating the query results of text search engines. In: *Proceedings of the VLDB Endowment*, vol. 1(1), pp. 126–137

[29] Wang J, Chen X, Huang X, You I and Xiang Y 2015 Verifiable auditing for outsourced database in cloud computing. *IEEE Trans. Comput.* 64(11) :3293-3303

[30] Kamara S and Papamanthou C 2013 Parallel and dynamic searchable symmetric encryption. In: *Proceedings of the International Conference on Financial Cryptography and Data Security*. Berlin, Heidelberg: Springer, pp. 258–274

[31] Cash D, Jaeger J, Jarecki S, Jutla C S, Krawczyk H, Rosu M C and Steiner M 2014 Dynamic Searchable Encryption in Very-Large Databases: Data Structures and Implementation. In: *Proceedings of NDSS'14*, 23–26 February

[32] Naveed M, Prabhakaran M and Gunter C A 2014 Dynamic searchable encryption via blind storage. In: *Proceedings of the IEEE Symposium on Security and Privacy (SP)*, pp. 639–654

[33] Xia Z, Wang X, Sun X and Wang Q 2016 A secure and dynamic multi-keyword ranked search scheme over encrypted cloud data. *IEEE Trans. Parall. Distr.* 27(2): 340–352

[34] Baldimtsi F and Ohrimenko O 2015 Sorting and searching behind the curtain. In: *Proceedings of the International Conference on Financial Cryptography and Data Security*. Berlin, Heidelberg: Springer, pp. 127–146

[35] Paillier P 1999 Public-key cryptosystems based on composite degree residuosity classes. In: *Proceedings of the International Conference on the Theory and Applications of Cryptographic Techniques*. Berlin, Heidelberg: Springer, pp. 223–238

[36] Agaian S S 1985 Construction of generalized Hadamard matrices. In: *Hadamard Matrices and Their Applications*. Lecture Notes in Mathematics 1168. Berlin, Heidelberg: Springer, pp. 103–133.

[37] Hazay C and Lindell Y 2008 Constructions of truly practical secure protocols using standardsmartcards. In: *Proceedings of the 15th ACM Conference on Computer and Communications Security*, ACM, pp. 491–500

[38] Lindell Y and Pinkas B 2015 An efficient protocol for secure two-party computation in the presence of malicious adversaries. *J. Cryptol.* 28(2): 312–350

[39] Goldreich O and Lindell Y 2001 Session-key generation using human passwords only. In: *Proceedings of the Annual International Cryptology Conference*. Berlin, Heidelberg: Springer, pp. 408–432

[40] Canetti R 2001 Universally composable security: a new paradigm for cryptographic protocols. In: *Proceedings of the 42nd IEEE Symposium on Foundations of Computer Science*, pp. 136–145

[41] Bellare M and Rogaway P 1993 Random oracles are practical: a paradigm for designing efficient protocols. In: *Proceedings of the 1st ACM Conference on Computer and Communications Security*, pp. 62–73

[42] Cohen W W 2009 *Enron email dataset*, https://www.cs.cmu.edu/~./enron/ (accessed May 2008)

[43] Cao N, Wang C, Li M, Ren K and Lou W 2014 Privacy-preserving multi-keyword ranked search over encrypted cloud data. *IEEE Trans. Parall. Distr.* 25(1): 222–233

[44] Sun W, Liu X, Lou W, Hou Y T and Li H 2015 Catch you if you lie to me: efficient verifiable conjunctive keyword search over large dynamic encrypted cloud data. In: *Proceedings of the IEEE Conference on Computer Communications (INFOCOM)*, pp. 2110–2118

[45] Wang C, Cao N, Li J, Ren K and Lou W, 2010 Secure ranked keyword search over encrypted cloud data. In: *Proceedings of the IEEE 30th IEEE International Conference on Distributed Computing Systems*, pp. 253–262

[46] Ding X, Liu P and Jin H 2017 Privacy-preserving multi-keyword top-*k* similarity search over encrypted data. *IEEE Trans. Depend. Secure.* 5971: 114

[47] Jiang X, Yu J, Yan J and Hao R 2017 Enabling efficient and verifiable multi-keyword ranked search over encrypted cloud data. *Inform. Sci.* 403: 22–41