# SEMFS: Secure and Efficient Multi-keyword Fuzzy Search for Cloud Storage

Sanjeet Kumar Nayak(✉) and Somanath Tripathy

Department of Computer Science and Engineering,
Indian Institute of Technology Patna, Patna, India
{sanjeet.pcs13,som}@iitp.ac.in

**Abstract.** Cloud computing has become a popular technology for outsourcing data and providing reliable data services. Encryption is essential to preserve privacy of the outsourced sensitive data. Keyword search over encrypted data would enhance the effective utilization of outsourced storage. In this work, we propose an efficient Searchable Symmetric Encryption (SSE) scheme called SEMFS (Secure & Efficient Multi-keyword Fuzzy Search Scheme) to allow the cloud to search over outsourced encrypted data. SEMFS uses quotient filter for efficient indexing and faster searching. The most attractive feature of this scheme is to allow update the entries of index file dynamically (to achieve better performance) preserving data privacy. Experimental analysis shows that SEMFS achieves higher throughput than the bloom filter based scheme, when implemented. Security of SEMFS has been analyzed against known ciphertext and known plaintext attack models.

**Keywords:** Cloud storage · Searchable symmetric encryption
Multi-keyword search · Quotient filter · Fuzzy search

## 1 Introduction

With the advancement of cloud computing, data outsourcing has become a major motive of several individuals and enterprises. But simply outsourcing the sensitive data like personal, financial, medical, etc., in plaintext format would lead to security risk. In addition to this, there could be risks from cloud storage server, as both data owner (DO) and cloud server (CS) are not in the same trusted domain [24]. This issue would be addressed if encrypted data is uploaded on CS. As DO outsources data in encryption format, it creates difficulty for the data users (DUs) to search a data file using keywords. Therefore, there is a call for some mechanisms to facilitate data users to search directly over the encrypted data.

Searchable encryption (SE) facilitates DU to securely search a file from encrypted files, using certain specific keywords without decrypting. Hence, SE is a widely adopted solution that helps DU to search over a vast collection of

© Springer International Publishing AG 2017
R. K. Shyamasundar et al. (Eds.): ICISS 2017, LNCS 10717, pp. 50–67, 2017.
https://doi.org/10.1007/978-3-319-72598-7_4

outsourced data. There are two kinds of SE schemes, namely, searchable symmetric encryption (SSE) and searchable public-key encryption (SPE). The inherent cryptographic techniques used by the SSE and SPE schemes are symmetric key cryptography and asymmetric key cryptography respectively. SSE schemes are more efficient and easier to implement as compared to SPE based schemes.

Song et al. [27] presented a searchable encryption which searches the entire document sequentially, so consumes more time. Subsequently, many works including [6,11,13] have been proposed to address this issue, using index based matching of keywords. But, unfortunately, all these schemes facilitate single keyword search over encrypted data. For enriching the search functionality, several schemes like [7,14,16] proposed multiple keyword search over encrypted data. However, these schemes only support exact keyword search. But, in practical scenario user searching behavior would lead to minor typing mistakes and format inconsistencies. Data users may not input exactly the same pre-set keyword due to some typos, inconsistencies in the representation of the word or due to lack of exact knowledge. Recently, few works [10,17,19,20] extended the search functionality to address this issue supporting user searching behavior (known as fuzzy search). Wang et al. [29] proposed an efficient multi-keyword fuzzy search over the encrypted data. They used bloom filter for efficient search algorithm.

In this paper, we propose an efficient multi-keyword fuzzy search technique for cloud storage named SEMFS. It preserves privacy of the index file, search query and documents without needing predefined dictionary for dynamic data operation. We use quotient filter for the first time to enable searching over encrypted data; as a result, SEMFS achieves efficient indexing and faster searching as compared to bloom filter based scheme. This scheme allows the data owners to directly update the secure index file present in the cloud server if the corresponding files are modified. The effectiveness and efficiency of the scheme is evaluated using experimental evaluation. Along with this, we analyze the security of the proposed scheme against known ciphertext and known plaintext attack models.

The rest of the paper is organized as follows. Section 2 discusses the related works in this area. Section 3 describes the system model and threat model. In Sect. 4, we introduce details of our proposed scheme. We describe the security analysis of the proposed scheme in Sect. 5. Section 6 provides some discussions on the proposed scheme along with performance analysis. Finally, the concluding remarks are provided in Sect. 7.

## 2 Related Work

Searchable encryption (SE) was first proposed by Song et al. [27]. The motive behind their proposal is to enable searching on encrypted data without leaking any information to the untrusted server. Their scheme is a symmetric key based proposal. Here, data user has to go through the entire document to search a particular keyword in a document. So searching overhead is linear in terms of length of the document. In [13], the author has developed a secure index per

file to reduce the searching overhead. A data user can query for a keyword using the trapdoor which is a function of keyword and secret key. Bloom filter [5], a space efficient data structure, is used as a per document index to track words in each document. Thus, searching overhead is reduced. The authors in [9] proposed a keyword index which associates each word with its corresponding file. Data owner uses pseudo random bits to mask the dictionary based keyword index for each file and sends it to the cloud server. Later the data user recovers the index. Curtmola et al. [11] proposed an inverted index based searchable symmetric encryption scheme which indexes per-keyword. A single encrypted hash table is built for the entire file collection. Here, each entry consists of a keyword trapdoor corresponding to the encrypted form of those file identifiers contain that keyword. The searching scheme proposed in [28] reduces the search time to logarithm order. It ensures the privacy of searched keywords. First public key encryption with keyword search (PKES) scheme is given by Boneh et al. [6]. Functionality of this scheme is quite similar to that of the Song et al. [27]. But, only the authorized data users can search with the corresponding private key. This scheme uses an adversary model similar to Goh et al.'s scheme [13], but requires the use of computationally intensive pairing operations [21]. All above works support single keyword search over the outsourced encrypted data.

As multiple keyword search has become a necessary functionality of SE schemes, many techniques try to incorporate this. Among these works, conjunctive keyword search schemes [14–16] return the list of documents that contain all the keywords while disjunctive keyword search schemes [7,33] return list of documents that contain a subset of query words. However, these multi-keyword search schemes do not provide ranking of the search result, which helps in retrieving most relevant files containing the keywords. Cao et al. [8] proposes a privacy preserving multi-keyword ranked search scheme over encrypted data.

A fuzzy keyword search algorithm proposed by Li et al. [19]. This is a wild card based scheme and facilitates the data users to search the desired file with minor typing errors and even if the format is inconsistent. Liu et al. [20] improves this scheme by reducing the index size. Chuah et al. [10] proposed a privacy preserving tree based fuzzy searching algorithm. All these schemes support single keyword based fuzzy search. Later Wang et al. [29] proposes a multi-keyword based fuzzy search using bloom filter. It uses locality sensitive hashing in place of wild cards to enhance the efficiency of fuzzy search. Along with this, it eliminates the need of a predefined dictionary for dynamic data updation.

# 3 Problem Formulation

## 3.1 System Model

A typical system model for secure search over encrypted data is as shown in Fig. 1. It consists of three entities, namely, data owner (DO), cloud server (CS) and data user (DU). DO outsources the data files (documents) to an untrusted cloud server. Each document contains a set of words referred as "keywords", through which the document can uniquely be identified. Keywords can

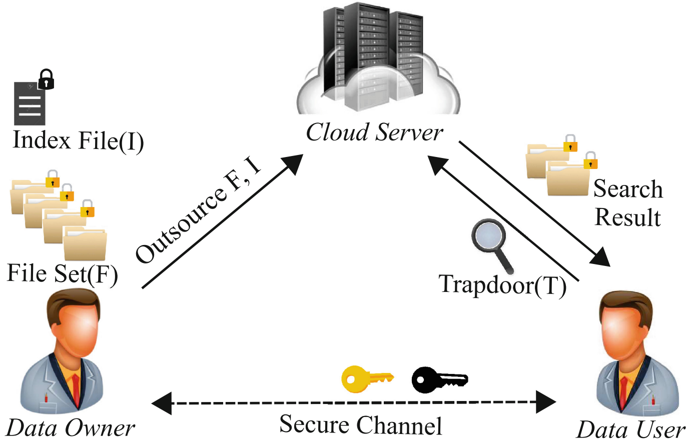



**Fig. 1.** Cloud data storage architecture for secure search over encrypted data. (Color figure online)

be extracted by analyzing the entire document either manually or through an automated procedure or more sophisticated text mining algorithms like multi purpose automatic topic indexing (Maui) [22] and rapid automatic keyword extraction (RAKE) [25]. DU submits a set of keywords as a query to CS. CS returns the corresponding files by searching those keywords in the index file. Let $F = \{f_1, f_2, f_3, ..., f_n\}$ denotes the set of files that DO wants to outsource. To protect the confidentiality, these files could be encrypted using symmetric encryption like advanced encryption standards (AES) [12].

To facilitate searching over encrypted files in efficient manner, DO creates a quotient filter based secure searchable index file ($\mathrm{QF}_i$) using a secret key ($K_{sec}$). $\mathrm{QF}_i$ is formed using a collection of distinct keywords $W = \{w_1, w_2, w_3, ..., w_m\}$ of the file $f_i$. Finally, data owner outsources both $F$ and $\mathrm{QF}_i$ to CS. Cloud server stores $F$ and $\mathrm{QF}_i$ in its data storage to enable data searching mechanism for DU. When DU sends a trapdoor ($T_{kw'}$) of a keyword $kw^{'}$ to CS, it checks for the same in $\mathrm{QF}_i$. It sends the corresponding collection of files. In addition to this necessary operations, DO sometimes update a file and corresponding $\mathrm{QF}_i$ and informs regarding the same to the cloud server.

Initially, data owner shares the symmetric encryption key ($K_{sym}$) and secret key ($K_{sec}$) to all authorized DUs. Using $K_{sec}$, DU generates the trapdoor ($T_{kw'}$) of a keyword $kw^{'}$ and sends it to CS. Cloud server searches for the corresponding trapdoor in the index file and returns the encrypted data files to DU. DU can decrypt those files using $K_{sym}$.

### 3.2 Threat Model

Cloud server is considered to be an "honest-but-curious." This signifies that CS executes secure searching over outsourced data honestly, at the same time it is

curious to derive the information regarding queried word [18,30]. However, data owner and data user are assumed to act honestly and trust each other. The focus of this work is mainly over confidentiality rather than availability. Distribution of the secret informations ($K_{sym}$, $K_{sec}$) are being performed via a secure channel, and such key distribution mechanisms are studied separately in [2,23].

Based on the information available to CS, we consider the following two threat models.

– **Known Ciphertext Model:** In this model, the cloud server has access to the collection of encrypted files $F$, secure index file $I$ (outsourced by DO) and trapdoor of the keyword $T_{kw'}$ (submitted by DU). Further, CS can have a collection of previously submitted trapdoors.
– **Known Plaintext Model:** This model is more stronger than the previous model. Here, the cloud server has some extra background knowledge including the information of the known ciphertext model. Background information includes statistical information that can be generated using a similar kind of data set as that data owner used. They can use the known index/trapdoor generation mechanism using frequency of words. This information can help them to find some private information of the data owner.

In both these models, target of CS is to derive exact keywords that DU searches. It will also try to find the content of the encrypted files which can leak privacy of the keywords.

### 3.3 Design Goals

To enable secure searching over encrypted data with the above system and threat model the following design goals are desired.

– **Multi-keyword Fuzzy Search:** This feature allows to search multiple keywords using logical connectives like "AND", "OR", etc. This scheme should allow fuzzy search so that minor typos and format inconsistencies will not disappoint the user searching experience. For example, files containing keyword "international football match" should be returned as a search result for the mis-typed word "international football mach" or "internasional football match."
– **Dynamic Update:** DO sometimes update a file and corresponding $\text{QF}_i$ stored in the cloud server has to be updated. So, scheme should be designed to provide dynamic data operation (insertion/deletion/modification) over the secure index file in an efficient manner.
– **Ranking of Search Result:** Scheme should rank the relevance of files in response to a given search query for the convenience of the data user.
– **No Predefined Dictionary:** Use of a predefined dictionary for the search operations leads to difficulty during dynamic update of index file. Schemes without predefined dictionary can update index file comfortably.
– **Confidentiality of Document, Privacy of Index File and Trapdoor:** The keywords stored in the index file as well as the search query (or trapdoor)

should not reveal any information to the cloud server regarding the searched keywords ($w_i$). As the outsourced documents are present in the CS, they have to be encrypted to preserve the confidentiality of the documents. Therefore, if any one of the three is not encrypted, then the privacy of the searched keyword may leak.

# 4 SEMFS: The Proposed Scheme

## 4.1 Basic Idea

SEMFS creates a per file quotient filter ($\mathrm{QF_i}$) which contains an index information of all the keywords present in file $f_i$. To enable multi-keyword search, we convert each keyword into a trigram set and use a modified quotienting function to insert the trigram set into $\mathrm{QF_i}$. Quotient filter and trigram set are precisely discussed as under.

**Quotient Filter:** Quotient filter (QF) has been introduced by Bender et al. in [4]. It is a time-efficient data structure for representing a set, to support membership queries. QF returns no to the membership query assures that the queried element is definitely not present in the set. Otherwise, the element is said to be probably present. Thus, quotient filter never returns false negative.

QF stores a $m$ bit hash value (known as fingerprint ($FP$)) of an element $\mathbb{E}$ as follows. This $m$ bit value is split into two parts as remainder ($FP_r$) and quotient ($FP_q$). The least significant $r$ bit constitute remainder and the most significant $q = m - r$ bit constitute quotient ($FP_q$). This is known as quotienting technique. $FP_q$ is used as an index to find the corresponding slot (or bucket) for $FP$ in the QF and the slot is filled with $FP_r$. Inserting a fingerprint in QF may encounter a soft collision when only $FP_q$ of two or more fingerprints are same. The canonical slot is the bucket in which a fingerprint's remainder ($FP_r$) would be stored in the absence of a collision. All remainders of fingerprints with the same quotient are stored contiguously, and known as run for the corresponding quotient. Hence, a run constitutes of all the slots that contain remainders with the same quotient. But, a cluster is a greater sequence of occupied slots whose first element is the only element stored in its canonical slot. One or more runs may present in a cluster.

To search a stored fingerprint within a cluster, each bucket contains additional three bits known as *is_occupied*, *is_continuation*, *is_shifted*. All these are initialized to 0 at the beginning. *is_occupied* bit of bucket $i$ is set when the bucket $i$ is the canonical slot ($FP_q = i$) for some fingerprint $FP$ which is stored somewhere in QF. *is_continuation* bit is set to 0 indicates the start of a run. *is_shifted* bit is set to 0 indicates start of a cluster. Algorithms 1 and 2 describe the searching and insertion procedure of a fingerprint $FP$ in the QF respectively [3].

A schematic diagram of a QF is shown in Fig. 2. This example considers $FP_q = \lfloor \frac{FP}{2^8} \rfloor$ and $FP_r = FP \mod 2^8$. This QF contains fingerprint values 258,

**Algorithm 1.** To search an element $\mathbb{E}$ in QF

**Input:** QF, $\mathbb{E}$
**Output:** Probably present/Definitely not present

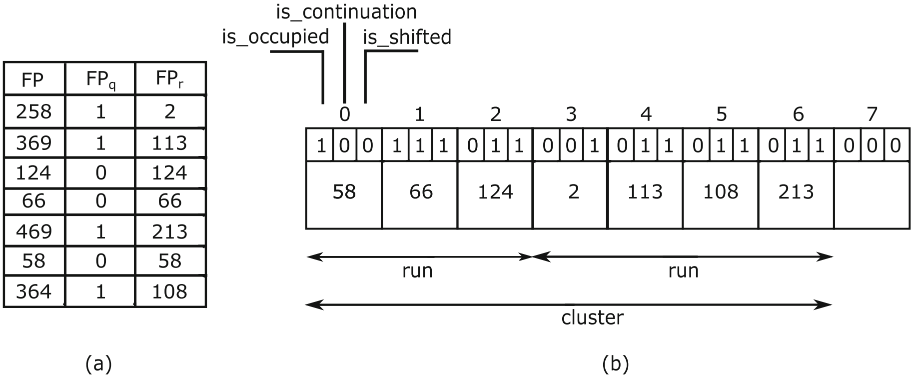
```
 1: Find fingerprint (FP) of E.
 2: Find quotient (FP_q) and remainder (FP_r) for FP.
 3: Set running_count ← 0.
 4: if ¬(is_occupied QF[FP_q] = 0) then                ▷ bucket FP_q is empty
 5:     FP is not present in the filter.
 6: else
 7:     repeat
 8:         Scan left from bucket FP_q
 9:         if is_occupied = 1 then
10:             running_count ++;
11:         end if
12:     until find a bucket with is_shifted = 0
13:     repeat
14:         Scan right from current bucket
15:         if is_continuation = 0 then
16:             running_count −−;
17:         end if
18:     until running_count = 0
19:     Compare the stored remainder in each bucket in the quotient's run with FP_r.
20:     if found then
21:         Element is in the filter (probably).
22:     else
23:         Element is not in the filter (definitely).
24:     end if
25: end if
```

**Algorithm 2.** To insert an element $\mathbb{E}$ in QF

**Input:** QF, $\mathbb{E}$
**Output:** QF with $\mathbb{E}$ inserted

```
1: Find fingerprint (FP) of E.
2: Find quotient (FP_q) and remainder (FP_r) for FP.
3: Proceed as Algorithm 1 till FP is definitely not in QF.
4: Choose a bucket in the current run by keeping the order sorted.
5: Insert FP_r and (set is_occupied bit).
6: Shift forward all remainders at or after the chosen bucket. Update the buckets'
   bits.
```

369, 124, 66, 469, 58 and 364. Figure 2 (a) presents the quotient, remainder pair for each fingerprint. Using these quotients and remainders, we inserted corresponding fingerprints using Algorithm 2 in the QF. At last, the contents of QF is shown in Fig. 2 (b). Here, remainders are stored in the corresponding bucket (quotient represents the bucket number) in the filter. Each bucket of QF contains three bits known as *is_occupied*, *is_continuation*, *is_shifted* (meta data) and the remainder. Here, fingerprints 124, 66, and 58 have the same quotient



| FP | $FP_q$ | $FP_r$ |
|---|---|---|
| 258 | 1 | 2 |
| 369 | 1 | 113 |
| 124 | 0 | 124 |
| 66 | 0 | 66 |
| 469 | 1 | 213 |
| 58 | 0 | 58 |
| 364 | 1 | 108 |



**Fig. 2.** An example of quotient filter

and constitute a run. Even though 258 could be placed in slot 1, but as run of 0 occupies 3 slots, it is shifted from its canonical slot. Fingerprints 258, 369, 469, and 364 constitute run of quotient 1. Fingerprints 258, 369, 124, 66, 469, 58 and 364 constitute a cluster.

**Trigram Set Construction from Keyword:** Each keyword of the corresponding file is transformed into a trigram set ($TS$). A trigram set contains all the contiguous three letters appeared in the keyword (considering circular way). For example, the trigram set of a keyword "relativity" is {rel, ela, lat, ati, tiv, ivi, vit, ity, tyr, yre}. Here, we assume that keywords $\in \{a, b, ..., z\}^+$.

Each different trigram element (of alphabet set) is enumerated to a unique decimal number using a function $\mathbb{F}$ which works as follows. Letters $[a \cdots z]$ are mapped to $[0 \cdots 25]$. The equivalent decimal representation of the trigram entry can be obtained using Eq. 1.

$$\mho_i = \mathbb{F}(TS_i) = (TS_i[0] * 26^2 + TS_i[1] * 26^1 + TS_i[2] * 26^0) \tag{1}$$

For example,

$$\begin{aligned}\mathbb{F}(rel) &= (r * 26^2 + e * 26^1 + l * 26^0)\\ &= (17 * 26^2 + 4 * 26^1 + 11 * 26^0)\\ &= (11492 + 104 + 11) = 11607\end{aligned}$$

Now, the modified representation of $TS$ (i.e. $\mho_i, \forall i \in TS$) contains equivalent decimal representations of a trigram.

**Inserting Trigrams into Quotient Filter:** At first, $\mho_i$ is divided into two parts ($\mho_{iq}$ and $\mho_{ir}$) as

$$\mho_{iq} = \left\lfloor \frac{\mho_i}{\beta} \right\rfloor \tag{2}$$

$$\mho_{ir} = \mho_i \mod \beta \tag{3}$$

Here, $\beta$ is a user defined parameter.[1] $\mho_{iq}$ is used as an index in QF and $\mho_{ir}$ is stored in the corresponding slot. Insertion is carried out as per Algorithm 2.

### 4.2 Operational Details of SEMFS

SEMFS consists of following four phases as depicted in Fig. 3 and discussed below.

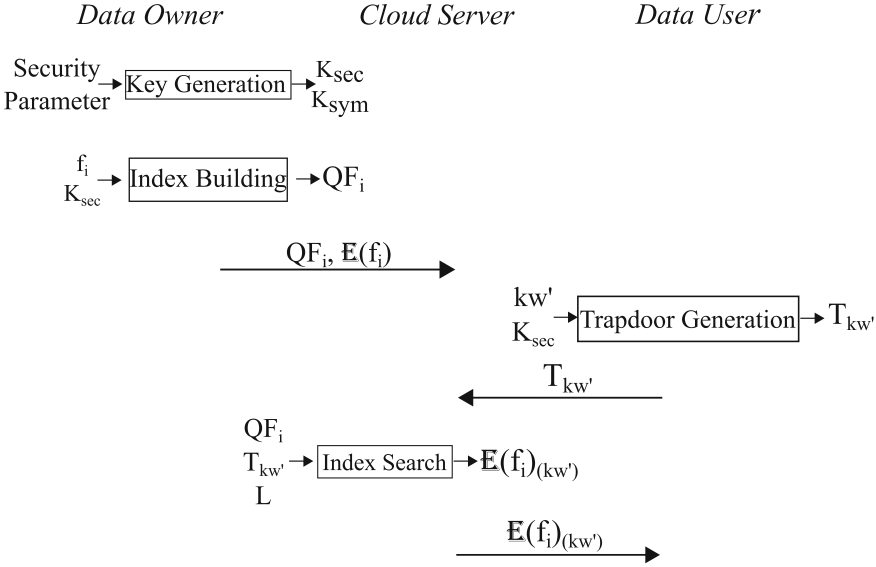



**Fig. 3.** Schematic diagram of proposed SEMFS.

1. **Key generation phase:** DO generates a secret key ($K_{sec}$) and a symmetric key ($K_{sym}$) to be shared with the DUs. It declares a secure hash function ($\mathbb{H}$) like SHA 2 [1] and symmetric encryption algorithm like advanced encryption standard (AES) [12]. In addition, data owner declares a system wide parameter $L$ which indicates the level of fuzziness. Level of fuzziness would be defined as the ability of the technique to support at most $L$ number of typing mistake or format inconsistencies. For example, if $L = 1$, then SEMFS should return the files indexed by keywords "Tom and Jane", when a data user submits query for the keywords "Tom a<u>m</u>d Jane."
2. **Index building phase:** In this phase, DO constructs a QF using Algorithm 3 with the input as keywords ($kw_j$) of the corresponding file.

[1] Let, $\alpha \times \beta = \gamma$ where $\alpha$ is the total number of slots in QF and $\beta$ is the total number of different values that can be stored in each slot.

A delimiter (like ",") may be placed at the end of each keyword as the terminating character. Then, a trigram set ($TS$) is constructed for each keyword. Each entry of $TS$ is represented in equivalent decimal form. Hash digest of the equivalent decimal form of trigram entry and $K_{sec}$ (secret key) is computed. Then, the digest is divided by $\beta$, results into quotient and remainder which would be inserted into $\text{QF}_i$ accordingly using Algorithm 2. Finally, this algorithm results in a secure index ($QF_i$) of the corresponding file $f_i$. $QF_i$ is treated as secure index of the file $f_i$ as CS does not know $K_{sec}$.

Now DO encrypts file ($f_i$) using AES with key $K_{sym}$ and uploads the encrypted file ($\mathscr{E}(f_i)$) with the corresponding secure index file $QF_i$ to the CS.

**Algorithm 3.** BuildIndex

**Input:** $f_i$, $kw_j$, $\mathbb{H}$, $K_{sec}$, $\gamma$
**Output:** $QF_i$

1: **for** $j = 1$ to $m$ **do** ▷ for each keyword $\{kw_1, ..., kw_m\}$ in $f_i$
2: Construct trigram set $TS_j = \{t_1, ..., t_p\}$.
3: **for** $k = 1$ to $p$ **do**
4: Find decimal representation of $t_k (\mho_k)$.
5: Find $\mathscr{C}_k = \mathbb{H}(\mho_k || K_{sec})$.
6: Find $\mathbb{C}_k = \mathscr{C}_k \mod \gamma$ ▷ $\gamma = \alpha \times \beta$
7: Find $\mathbb{C}_{k_q} = \left\lfloor \frac{\mathbb{C}_k}{\beta} \right\rfloor$ and $\mathbb{C}_{k_r} = \mathbb{C}_k \mod \beta$
8: Using $\mathbb{C}_{k_r}$ and $\mathbb{C}_{k_q}$ insert $\mathbb{C}_k$ in the $\text{QF}_i$.
9: **end for**
10: **end for**

3. **Trapdoor generation phase:** In this phase, DU builds trapdoors for each keyword (to be searched) using Algorithm 4. Then, a trigram set $TS^{'}$ is constructed for each keyword (after placing the same delimiter used during *index building phase*) to be searched. Each entry of $TS^{'}$ is represented in equivalent decimal form. Hash of the equivalent decimal form of trigram entry and $K_{sec}$ is computed. This digest value is inserted into the trapdoor set ($T_{kw'}$). Then, DU sends ($T_{kw'}$) to the CS.
4. **Searching phase:** Here, CS executes Algorithm 5 to get the list of corresponding encrypted files $\mathscr{E}(f_i)$ containing the keywords present $kw^{'}$. Algorithm 5 searches for each element of $T_{kw'}$ (i.e. $\mathbb{C}_j^{'}$) in each $QF_i$ using Algorithm 1 and remembers the number of matching found. As proposed scheme SEMFS converts all the keywords (either to insert in $QF_i$ or to search in $QF_i$) into trigram format, then, three trigrams will be affected if one letter is mismatched. As $L$ defines the permissible number of typing mistakes or format inconsistencies that will be relaxed to find the corresponding file that contains the keywords, we say if number of matching is $\geq 3L$, the corresponding file is returned by this algorithm. At last, a collection of all such files is sent back to the DU. Upon receiving the encrypted files, DU will decrypt the files using symmetric encryption key $K_{sym}$.

**Algorithm 4.** BuildTrapdoor

**Input:** $kw^{'}$, $\mathbb{H}$, $K_{sec}$, $\gamma$
**Output:** $T_{kw'}$

```
1: for j = 1 to q do          ▷ for each keyword to be searched kw' = {kw_1', ..., kw_q'}
2:     Construct trigram set TS_j' = {t_1', ..., t_r'}.
3:     for k = 1 to r do
4:         Find decimal representation of t_k' (Ʊ_k').
5:         Find 𝒞_k' = ℍ(Ʊ_k' || K_sec).
6:         Find ℂ_k' = 𝒞_k' mod γ
7:         Insert ℂ_k' into trapdoor set (T_kw').
8:     end for
9: end for
```

**Algorithm 5.** IndexSearch

**Input:** $T_{kw'}$, $L$
**Output:** $f_i$

```
1: count ← 0
2: for i = 1 to n do
3:     for j = 1 to ψ do                                   ▷ |T_kw'| = ψ
4:         if ℂ_j' ∈ QF_i then
5:             count ← count + 1
6:         end if
7:     end for
8:     if count ≥ (ψ − 3L) then
9:         return f_i
10:    end if
11: end for
```

## 5 Security Analysis

In this section, we analyze the security of SEMFS under two different kinds of security attack models, namely, known ciphertext model and known plaintext model.

### 5.1 Confidentiality of Files

In SEMFS, the outsourced files ($\mathscr{E}(f_i)$) are encrypted using a standard symmetric encryption algorithm AES [12]. In addition to this, DO sends $K_{sym}$ (used to encrypt/ decrypt the files) through a secure channel to the authorized DUs. Without this key, CS can not decrypt the files as AES encryption scheme is secure. So, confidentiality of the files can be achieved.

### 5.2 Privacy Protection of Index Files

In case of known ciphertext model, CS has only the following information. It has access to the quotient filter ($\mathrm{QF}_i$), encrypted files ($\mathscr{E}(f_i)$) and trapdoor set

$T_{kw'}$. To break the security of SEMFS under known ciphertext model, CS could try to guess the content of $\mathrm{QF}_i$ to find exact keywords of $f_i$. But, due to the following reasons, CS fails to guess the content of $\mathrm{QF}_i$ correctly.

In index building phase, DO inserts $\mathbb{C}_k$'s ($= \mathscr{C}_k \mod \gamma$) in quotient filter $\mathrm{QF}_i$, where $\mathscr{C}_k = \mathbb{H}(\mho_k || K_{sec})$. $\mathscr{C}_k$'s are computed using a secret key $K_{sec}$. As DO sends $K_{sec}$ through a secure channel to the authorized DUs, CS cannot know $K_{sec}$. In addition to this, it is computationally difficult for the CS to find $\mathscr{C}_k$ and $K_{sec}$, only by knowing the digest of secure hash function ($\mathbb{H}$) as it is a one-way hash function. Hence, the content of the index file $\mathrm{QF}_i$ is well protected from CS in SEMFS.

In case of known plaintext model, including the information regarding quotient filter, encrypted files and trapdoor set, CS has extra information regarding plaintext of some documents. It is computationally difficult for the CS to generate a correct index file (same as that of the DO) of the files using the known information regarding plaintext of some documents due to the following reason. As $K_{sec}$ is used to generate $\mathscr{C}_k$ and only DO and authorized DUs know $K_{sec}$, CS cannot generate correct $\mathscr{C}_k$. Also, CS cannot find $K_{sec}$ by knowing a pair of trigram $\mho_k$ and corresponding encrypted trigram $\mathscr{C}_k$ due to the one way property of hash function. Hence, the index file generated by the CS will be different from the original one. Thus, content of the index file $\mathrm{QF}_i$ is well protected from CS in SEMFS under known plaintext model.

### 5.3 Privacy Protection of Search Query

In case of known ciphertext model, CS has only the following information. It has access to the trapdoor set $T_{kw'}$, quotient filter ($\mathrm{QF}_i$) and encrypted files ($\mathscr{E}(f_i)$). It has access to the trigram set $T_{kw'}$. To break the security of SEMFS under known ciphertext model, CS could try to guess the content of trapdoor set $T_{kw'}$ to know exact keywords that DU wants to search. But, due to the following reasons, CS fails to guess the content of $T_{kw'}$ correctly.

In the trapdoor building phase, DU constructs the trapdoor set $T_{kw'}$ which contains $\mathbb{C}_k{}'$ ($= \mathscr{C}_k{}' \mod \gamma$, where $\mathscr{C}_k{}' = \mathbb{H}(\mho_k{}' || K_{sec})$). $\mathscr{C}_k{}'$'s are computed using a secret key $K_{sec}$. As DO sends $K_{sec}$ through a secure channel to the authorized DUs, CS cannot know $K_{sec}$. In addition to this, it is computationally difficult for the CS to find $\mathscr{C}_k{}'$ and $K_{sec}$, only by knowing the digest of secure hash function ($\mathbb{H}$) as it is a one-way hash function. Hence, the privacy of the trapdoor set $T_{kw'}$ is protected from CS in SEMFS.

In case of known plaintext model, including the information regarding quotient filter, encrypted files and trapdoor set, CS has extra information regarding plaintext of some documents. It is computationally difficult for the CS to generate a correct trapdoor set (same as that of the DU) using the known information regarding plaintext of some documents due to the following reason. As $K_{sec}$ is used to generate $\mathscr{C}_k{}'$ and only DO and authorized DUs know $K_{sec}$, CS cannot generate correct $\mathscr{C}_k{}'$. Also, CS cannot find $K_{sec}$ by knowing a pair of trigram $\mho_k{}'$ and corresponding encrypted trigram $\mathscr{C}_k{}'$ due to the one way property of

hash function. Hence, the index file generated by the CS will be different from the original one. Thus, privacy of the trapdoor set is preserved in SEMFS under known plaintext model.

# 6 Discussion

## 6.1 Choice of Trigrams

Trigram set would be the best alternative as unigram and bigram set of a keyword would be identical for those words with anagram. As an example, the unigram set of "cat" and "act" is identical ($\{c, a, t\}$). Similarly, the bigram set of the keywords "deeded" and "deed" are identical ($\{de, ee, ed\}$). Trigram set of these two keywords are different ($\{dee, eed, ede, ded\}$ and $\{dee, eed\}$ respectively). This kind of two meaningful words with same trigram set we could not find from English dictionary and therefore conjectured to be least probable. Therefore, we choose to represent keywords using a trigram set.

CS returns all the files correspond to the set of trapdoors, if the number of matching trigrams $\leq 3L$ which is used in Algorithm 5. SEMFS converts all the keywords into circular trigram format to ensure each letter is present in three trigrams (otherwise, first and last letter will be present in only one trigram). This ensures that exactly $3L$ numbers of trigrams mismatched between trigrams present in trapdoor and trigrams present in $\text{QF}_i$.

## 6.2 Efficient Index File Updation in SEMFS

After updating a document, DO needs update the corresponding index file also. A straight forward way of updating (inserting/deleting/modifying) the index file is to download it from the CS, update it and resend the updated index file to CS, which is generally considered in bloom filter based searching schemes like Wang et al. [29]. This method increases the I/O operation and file transfer cost in the network. SEMFS supports direct updation of index file without downloading them as quotient filter supports addition and deletion of elements. In SEMFS, the update on the index file is carried out by the CS when it receives a tuple of 3 values $\langle \mathbb{C}_k, \text{QF}_i, \text{UpdateType} \rangle$ from the DO (Here, UpdateType $\in$ $\{\text{Insert, Delete}\}$). CS insert/delete/modify $\mathbb{C}_k$ in $\text{QF}_i$ using similar procedure as Algorithm 2. Here, CS cannot learn anything about the keyword as $\mathbb{C}_k$ is generated using $K_{sec}$ and $\mathbb{H}$. Insertion or deletion of a whole document is also possible in SEMFS.

## 6.3 Ranking the Search Result

Ranking of the search result immensely enhances schemes usability by returning the matching files in a ranked order. SEMFS supports ranking of search results. For this CS has to do some additional work during the searching phase. It maintains a list containing name of the file and number of matching trigrams. This

list is sorted in ascending order the number of matching trigrams. Now, top-$k$ most relevant files corresponding to DU's interested keyword are sent to the DU.

Keyword's frequency can also be considered as a tool to find the relevance between the files and keywords which can help in ranking the search result. Relevance score of a keyword for a file is more if the frequency of the keyword is more in the document. A scoring technique is widely used in plaintext information retrieval [26]. It helps in choosing most relevant document containing a set of keywords when we find more than one documents containing the same set of keywords.

### 6.4 Performance Analysis

Search functionalities like single keyword search, multiple keyword search, fuzzy keyword search, ranked keyword search, requirement of index file, requirement of predefined dictionary, confidentiality of files, preserving privacy of index file and preserving privacy of trapdoor are the important features need to be considered for designing an efficient SSE scheme in cloud storage. Table 1 summarizes the comparison of said features for different existing schemes with SEMFS. [9,27] and [13] schemes do not support multiple keyword search, while [11,15,30–32] and [18] do not support fuzzy keyword search. However, Li et al.'s scheme [19] enables fuzzy search but suitable for a single keyword searching. Later on Wang et al. [29] proposed an SSE scheme based on Bloom filter that enables fuzzy multi-keyword search. It can be observed that, SEMFS supports all the search functionalities mentioned earlier. Along with these functionalities, SEMFS performs dynamic update of index files efficiently.

To verify the efficiency of SEMFS, we implemented both bloom filter and quotient filter on a PC equipped with Intel Core i5 processor at 3.2 GHz and 4 GB RAM. An introduction to bloom filter is provided in Appendix A. Figure 4 shows that the time consumed by quotient filter during insertion and searching process is less than bloom filter as the number of elements increases. It can be observed from the result that with the increase in number of elements the speedup of quotient filter increases as compared to bloom filter as only a single hash function is required in case of quotient filter. We implemented Wang et al.'s scheme [29] and SEMFS and the result is shown in Fig. 5. This result is obtained for a single file with gradually increasing the number of keywords. Index generation time and trapdoor generation time of the respective schemes are very close as algorithms for index file generation and trapdoor generation are identical. Index file generation time and trapdoor generation time in Wang et al's scheme is more as compared to SEMFS in both Fig. 5 (a) and (b) respectively due to the following reason. Wang et al's scheme is a bloom filter based scheme and contain extra matrix multiplications than SEMFS. Therefore, we assure that the SEMFS is efficient as compared to Wang et al.'s scheme [29] with the application of QF for searchable symmetric encryption.

**Table 1.** Comparison of SSE schemes in cloud storage

| Schemes | SKWS | MKWS | FKWS | RKWS | IF | PD | PP | CD |
|---|---|---|---|---|---|---|---|---|
| Song et al. [27] | Y | **N** | **N** | **N** | **N** | N | **N** | Y |
| Chang et al. [9] | Y | **N** | **N** | **N** | Y | **Y** | **N** | Y |
| Goh et al. [13] | Y | **N** | **N** | **N** | Y | N | Y | Y |
| Curtomola et al. [11] | Y | Y | N | **N** | Y | **Y** | Y | Y |
| Wang et al. [30] | Y | Y | **N** | Y | Y | N | Y | Y |
| Wang et al. [31] | Y | Y | **N** | Y | Y | N | Y | Y |
| Yu et al. [32] | Y | Y | **N** | Y | Y | **Y** | Y | Y |
| Li et al. [18] | Y | Y | **N** | Y | Y | **Y** | Y | Y |
| Li et al. [19] | Y | **N** | Y | **N** | Y | N | Y | Y |
| Wang et al. [29] | Y | Y | Y | **N** | Y | N | Y | Y |
| Hong et al. [15] | Y | Y | **N** | **N** | Y | N | Y | Y |
| SEMFS | Y | Y | Y | Y | Y | N | Y | Y |

SKWS: Single Keyword Search, MKWS: Multiple Keyword Search, FKWS: Fuzzy Keyword Search, RKWS: Ranked Keyword Search, IF: Requirement of Index File, PD: Requirement of Predefined Dictionary, PP: Privacy Preserving, CD: Confidentiality of Document

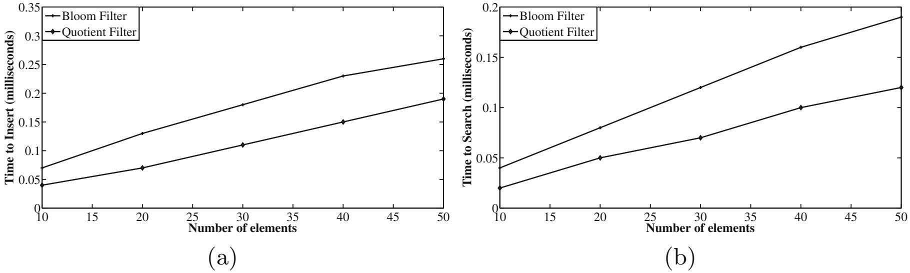



**Fig. 4.** Insertion and searching time comparison between bloom filter and quotient filter

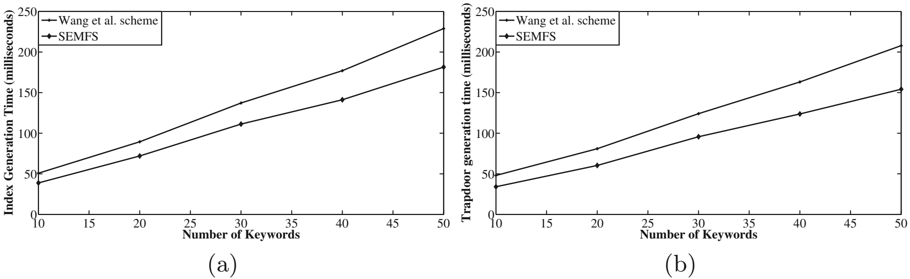



**Fig. 5.** Index generation and trapdoor generation time comparison between Wang et al.'s scheme [29] and SEMFS

## 7 Conclusion

Sensitive data are encrypted (to preserve privacy) before storing remotely, so achieving effective utilization of stored data in cloud has become challenging. This work proposed an efficient Searchable Symmetric Encryption (SSE) scheme called SEMFS which facilitates the data owner to outsource the encrypted data allowing a user to search the file using trapdoor (of keyword) without giving a clue to others (including CS) about the keyword. SEMFS uses quotient filter for efficient indexing and faster searching. Most appealing feature of the proposed scheme is to support dynamic updation of index file. Experimental analysis showed that SEMFS had higher throughput than the bloom filter based scheme, when implemented.

## Appendix A Bloom Filter

Bloom filter is a space-efficient data structure for representing a set in order to support membership queries [5]. A bloom filter for representing a set $S$ containing $n$ number of elements is described by an array of m bits size. All these m bits are initialized to 0. It uses $k$ independent hash functions defined as $\mathcal{H} = \{h_i | h_i : S \rightarrow [1, m], 1 \leq i \leq k\}$. To insert an element $s \in S$ into the bloom filter, all the $h_i(s)$-th position are set to 1 in the $m$ bit array. To search an element $q \in S$, first we find $k$ number of array positions using $h_i(q), 1 \leq i \leq k$. Then, we check if the corresponding bit of any of the $k$ position is 0, then $q$ is definitely not present in the set $S$. Otherwise, $q$ probably present in the set $S$. Consider the following example. Figure 6 shows an example of bloom filter with m = 10 and k = 3. Here, the filter represents the set $S = \{A, B\}$. Black colored arrow shows the corresponding positions of the bloom filter that each element of set $S$ is mapped to. Now if we will search for an element $D$, then we can observe from the figure that at least one position to which $D$ is mapped contains 0 (shown in red colored lines). So $D$ is definitely not present in the set. If we will search for an element $C$, then we can observe from the figure that all the positions to which $C$ is mapped contain 1. Hence, $C$ is probably present in the set.

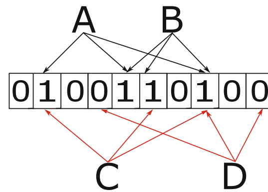



**Fig. 6.** An example of boolean filter (Color figure online)

## References


1. Secure Hash Algorithm 2. http://csrc.nist.gov/publications/fips/fips180-2/fips180-2withchangenotice.pdf
2. Adusumilli, P., Zou, X., Ramamurthy, B.: DGKD: distributed group key distribution with authentication capability. In: Proceedings of the Sixth Annual IEEE SMC Information Assurance Workshop, pp. 286–293. IEEE (2005)
3. Bender, M.A., Farach-Colton, M., Johnson, R., Kraner, R., Kuszmaul, B.C., Medjedovic, D., Montes, P., Shetty, P., Spillane, R.P., Zadok, E.: Don't thrash: how to cache your hash on flash. Proc. VLDB Endow. **5**(11), 1627–1637 (2012)
4. Bender, M.A., Farach-Colton, M., Johnson, R., Kuszmaul, B.C., Medjedovic, D., Montes, P., Shetty, P., Spillane, R.P., Zadok, E.: Don't thrash: how to cache your hash on flash. In: Proceedings of the 3rd USENIX Conference on Hot Topics in Storage and File Systems, p. 1 (2011)
5. Bloom, B.H.: Space/time trade-offs in hash coding with allowable errors. Commun. ACM **13**(7), 422–426 (1970)
6. Boneh, D., Di Crescenzo, G., Ostrovsky, R., Persiano, G.: Public key encryption with keyword search. In: Cachin, C., Camenisch, J.L. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 506–522. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-24676-3_30
7. Boneh, D., Waters, B.: Conjunctive, subset, and range queries on encrypted data. In: Vadhan, S.P. (ed.) TCC 2007. LNCS, vol. 4392, pp. 535–554. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-70936-7_29
8. Cao, N., Wang, C., Li, M., Ren, K., Lou, W.: Privacy-preserving multi-keyword ranked search over encrypted cloud data. In: Proceedings of the IEEE INFOCOM 2011, pp. 829–837. IEEE (2011)
9. Chang, Y.-C., Mitzenmacher, M.: Privacy preserving keyword searches on remote encrypted data. In: Ioannidis, J., Keromytis, A., Yung, M. (eds.) ACNS 2005. LNCS, vol. 3531, pp. 442–455. Springer, Heidelberg (2005). https://doi.org/10.1007/11496137_30
10. Chuah, M., Hu, W.: Privacy-aware bedtree based solution for fuzzy multi-keyword search over encrypted data. In: Proceedings of the 31st IEEE International Conference on Distributed Computing Systems Workshops (ICDCSW) 2011, pp. 273–281. IEEE (2011)
11. Curtmola, R., Garay, J., Kamara, S., Ostrovsky, R.: Searchable symmetric encryption: improved definitions and efficient constructions. In: Proceedings of the 13th ACM Conference on Computer and Communications Security, pp. 79–88. ACM (2006)
12. Daemen, J., Rijmen, V.: The Design of Rijndael: AES-The Advanced Encryption Standard. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-662-04722-4
13. Goh, E.J.: Secure indexes. Technical report 2003/216, IACR Cryptology ePrint Archive (2003)
14. Golle, P., Staddon, J., Waters, B.: Secure conjunctive keyword search over encrypted data. In: Jakobsson, M., Yung, M., Zhou, J. (eds.) ACNS 2004. LNCS, vol. 3089, pp. 31–45. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-24852-1_3
15. Hong, C., Li, Y., Zhang, M., Feng, D.: Fast multi-keywords search over encrypted cloud data. In: Cellary, W., Mokbel, M.F., Wang, J., Wang, H., Zhou, R., Zhang, Y. (eds.) WISE 2016. LNCS, vol. 10041, pp. 433–446. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-48740-3_32

16. Hwang, Y.H., Lee, P.J.: Public key encryption with conjunctive keyword search and its extension to a multi-user system. In: Takagi, T., Okamoto, T., Okamoto, E., Okamoto, T. (eds.) Pairing 2007. LNCS, vol. 4575, pp. 2–22. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-73489-5_2
17. Kuzu, M., Islam, M.S., Kantarcioglu, M.: Efficient similarity search over encrypted data. In: Proceedings of the IEEE 28th International Conference on Data Engineering (ICDE), pp. 1156–1167. IEEE (2012)
18. Li, H., Yang, Y., Luan, T.H., Liang, X., Zhou, L., Shen, X.S.: Enabling fine-grained multi-keyword search supporting classified sub-dictionaries over encrypted cloud data. IEEE Trans. Dependable Secure Comput. **13**(3), 312–325 (2016)
19. Li, J., Wang, Q., Wang, C., Cao, N., Ren, K., Lou, W.: Fuzzy keyword search over encrypted data in cloud computing. In: Proceedings of the IEEE INFOCOM 2010, pp. 1–5. IEEE (2010)
20. Liu, C., Zhu, L., Li, L., Tan, Y.: Fuzzy keyword search on encrypted cloud storage data with small index. In: Proceedings of the IEEE International Conference on Cloud Computing and Intelligence Systems (CCIS) 2011, pp. 269–273. IEEE (2011)
21. Lynn, B.: The pairing-based cryptography library (2006). https://crypto.stanford.edu/pbc/. Accessed 27 Mar 2013
22. Medelyan, O.: Human-competitive automatic topic indexing. Ph.D. thesis, The University of Waikato (2009)
23. Nabeel, M., Yoosuf, M., Bertino, E.: Attribute based group key management. In: Proceedings of the 14th ACM Symposium on Access Control Models and Technologies, pp. 115–124. ACM (2014)
24. Nayak, S.K., Tripathy, S.: Privacy preserving provable data possession for cloud based electronic health record system. In: Proceedings of the IEEE Trustcom/BigDataSE/ISPA, pp. 860–867. IEEE (2016)
25. Rose, S., Engel, D., Cramer, N., Cowley, W.: Automatic keyword extraction from individual documents. In: Text Mining: Applications and Theory, pp. 1–20. John Wiley & Sons (2010)
26. Singhal, A.: Modern information retrieval: a brief overview. IEEE Data Eng. Bull. **24**(4), 35–43 (2001)
27. Song, D.X., Wagner, D., Perrig, A.: Practical techniques for searches on encrypted data. In: Proceedings of the IEEE Symposium on Security and Privacy, pp. 44–55. IEEE (2000)
28. van Liesdonk, P., Sedghi, S., Doumen, J., Hartel, P., Jonker, W.: Computationally Efficient searchable symmetric encryption. In: Jonker, W., Petković, M. (eds.) SDM 2010. LNCS, vol. 6358, pp. 87–100. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-15546-8_7
29. Wang, B., Yu, S., Lou, W., Hou, Y.T.: Privacy-preserving multi-keyword fuzzy search over encrypted data in the cloud. In: Proceedings of the IEEE INFOCOM 2014, pp. 2112–2120. IEEE (2014)
30. Wang, C., Cao, N., Li, J., Ren, K., Lou, W.: Secure ranked keyword search over encrypted cloud data. In: Proceedings of the 30th International Conference Distributed Computing Systems (ICDCS), pp. 253–262. IEEE (2010)
31. Wang, C., Cao, N., Ren, K., Lou, W.: Enabling secure and efficient ranked keyword search over outsourced cloud data. IEEE Trans. Parallel Distrib. Syst. **23**(8), 1467–1479 (2012). IEEE
32. Yu, J., Lu, P., Zhu, Y., Xue, G., Li, M.: Toward secure multikeyword top-k retrieval over encrypted cloud data. IEEE Trans. Dependable Secure Comput. **10**(4), 239–250 (2013)
33. Zhang, B., Zhang, F.: An efficient public key encryption with conjunctive-subset keywords search. J. Netw. Comput. Appl. **34**(1), 262–267 (2011)