EnDAS: Efficient Encrypted Data Search as a Mobile Cloud Service

Ruhui Ma, Jian Li, Haibing Guan, Mingyuan Xia and Xue Liu

Abstract—Document storage in the cloud infrastructure is rapidly gaining popularity throughout the world. However, it poses risks to consumers unless the data is encrypted for security. Encrypted data should be effectively searchable and retrievable without any privacy leaks, particularly for the mobile client. Although recent research has solved many security issues, the architecture cannot be applied on mobile devices directly under the mobile cloud environment. This is due to the challenges imposed by wireless networks, such as latency sensitivity, poor connectivity, and low transmission rates. This leads to a long search time and extra network traffic costs when using traditional search schemes. This study addresses these issues by proposing an efficient Encrypted DAta Search (EnDAS) scheme as a mobile cloud service. This innovative scheme uses a lightweight trapdoor (encrypted keyword) compression method, which optimizes the data communication process by reducing the trapdoor's size for traffic efficiency. In this study, we also propose two optimization methods for document search, called the Trapdoor Mapping Table (TMT) module and Ranked Serial Binary Search (RSBS) algorithm, to speed the search time. Results show that EnDAS reduces search time by 34% to 47% as well as network traffic by 17% to 41%.

Index Terms—Mapping Table, Compression, Ranking Search, Encrypted Search, Mobile Cloud.

1 Introduction

Since cloud computing can support elastic services and provide an economical use of storage and computation resources, it is rapidly gaining popularity. With powerful cloud services, many data providers can populate their data in clouds instead of directly serving users. The cloud also allows providers to delegate important tasks such as document searches. To protect data security, the documents and their indexes are usually encrypted before outsourcing to the cloud for searches [1], [2]. When users need to query certain documents, they first send keywords to the original data provider. The provider then generates encrypted keywords (also called trapdoors) and returns the trapdoors to the user. The user then sends these trapdoors to the cloud. Upon receiving the trapdoors, the Cloud uses a special search algorithm to select a set of desired documents (encrypted) based on the encrypted indexes and given trapdoors. Finally, the user receives these encrypted search results and uses the private key from the provider to decrypt documents. This architecture, as depicted in Figure 1, protects data security while entitling the providers to use both the computation and storage power of the Cloud for document searches. Due to these advantages, this architecture has already been well-adopted in privacy-preserving search systems [2], [3], [4], [5], [6].

- R. Ma et al are with Shanghai Key Laboratory of Scalable Computing and Systems, Shanghai Jiao Tong University, Shanghai, China, 200240. E-mail: ruhuima, li-jian, hbguan@sjtu.edu.cn
- X. Liu and M. Xia are with the School of Computer Science, McGill University, Montreal, Canada, H3A 0E9. E-mail: xueliu@cs.mcgill.ca, mingyuan.xia@mail.mcgill.ca

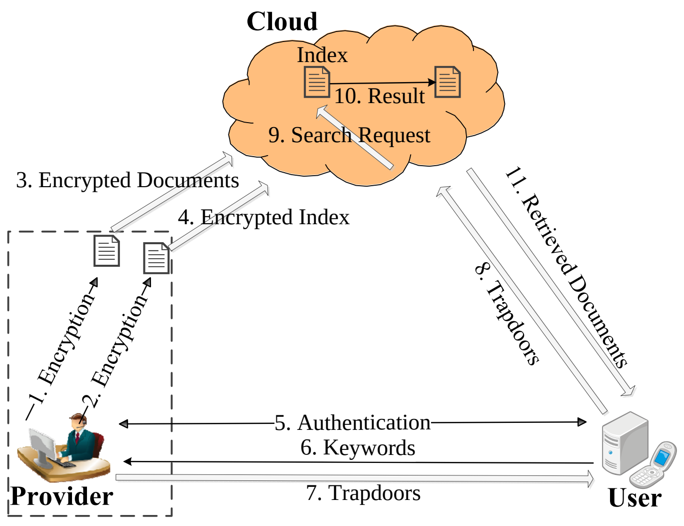


Figure 1. Traditional Encrypted Search System over Cloud

Mobile devices (e.g. smartphones and tablets) were estimated to surpass two billion growth (0.3 billions for PCs) in the year 2014, which dominates the overall shipment of consumer electronics devices [7]. Nowadays, users heavily utilize mobile devices to request document search services. In general, mobile devices connect to the Internet mainly via wireless networks (WiFi/3G/4G/LTE), which incurs some challenges as compared to traditional wired networks. These challenges include: **1) Latency sensitivity:** these wireless networks incur longer network latency, which can slow down a single search request if the search request requires many network round trips. For example, in the traditional design shown in Figure 1, a single search requires three round trips and results in notable latency for wireless communication. **2) Poor connectivity:** Mobile devices are normally incapable of maintaining a long-running connection with the Cloud, mostly for energy-saving purposes. Multiple search requests could incur numerous re-connection operations and extra authentication costs. **3) Low network transmission rate:** Mobile devices are normally equipped with low-power transmission components, bringing slower transmission rates.

For example, the traditional system shown in Figure 1 requires two network round trips between the user and the provider (for authentication and trapdoor generation) and one between the user and the cloud (for document retrieval). Three round trips simply impose notable search delay and excessive network traffic, which could be costly for a mobile device.

According to our measurement, a search request in the traditional system could produce trapdoors with a size up to 1.2MB [8]. When performing search requests, the trapdoor has to be sent twice (step 7 and 8). In such case, privacy-preserving searches could lead to longer search delay and more bandwidth consumption, which could not be affordable to mobile users.

This study focuses on traffic and search time inefficiency issues over the mobile cloud. We present an efficient Encrypted DAta Search (EnDAS) scheme as a mobile cloud service to tackle these problems. Our system supports multi-keyword privacy-preserving search and greatly reduces network traffic and search delays. For network traffic, EnDAS pre-computes trapdoors for common search keywords and thus avoids one network round trip for re-computing trapdoor per request. We further propose several mechanisms to compress trapdoors and demonstrate that our pre-computed trapdoor table has a size of 0.31MB and could be effectively stored and loaded in mobile device memory. In terms of search time, EnDAS retrofits the search algorithm in the cloud. Based on the binary tree principle, we present Ranked Serial Binary Search (RSBS) algorithm, which could reduce query time in the cloud. Our contributions can be summarized as follows:

1) We examined the traditional encrypted search architecture in terms of network traffic and search time. Results show that the conventional approach is not applicable in mobile-cloud environments.
2) We developed EnDAS to address these challenges. Our architecture includes a trapdoor compression method to reduce traffic costs, as well as a Trapdoor Mapping Table (TMT) module and RSBS algorithm to reduce search time.
3) We evaluated the efficiency of EnDAS in network traffic and search time. We demonstrated that with EnDAS architecture, we can reduce network traffic by 17% to 41% and search time by 34% to 47%.

The remainder of this article is organized as follows: Section 2 describes the traditional encrypted search system architecture and problems. Section 3 describes the detailed design of the EnDAS system, as well as analyze its network traffic and search time efficiency. Section 4 evaluates the systems, and related work is covered in Section 5. Section 6 provides conclusions and implications.

2 Problem Statement

In this section, we briefly introduce existing privacy-preserving search architectures and outline their shortcomings, both in terms of search delay and network traffic.

2.1 Traditional Encrypted Search System

As shown in Figure 1, the traditional encrypted search system over the cloud is composed of three different participants, Provider, Cloud and User, which are defined below.

The Provider possesses a set of documents and their indexes. It intends to outsource these to the cloud and let users contact the cloud for the search service. The **Cloud** is a commercial organization that provides computation and storage resources in the form of virtual machines, commonly known as "cloud" services. The **User** is someone who submits keywords to search documents that contain these keywords. In our scenario, users would use mobile device such as smartphones and tablets to submit search requests.

Figure 1 details the execution flow of a traditional encrypted search over the cloud, including three main flows: documents and indexes uploading process (steps 1 to 4), trapdoor generation process (steps 5 to 8) and document retrieval process (steps 9 to 11). The weight of lines indicates the amount of data being transferred.

Documents and indexes uploading process: First, the provider in charge of this flow stems all words in these documents to be stored in the cloud and retains these terms. Then each term is encrypted and considered as one index's keyword. The encryption algorithm can employ the classic symmetric-key cryptography algorithm such as the Advanced Encryption Standard [9], [10]. The frequency of each term in the document set is counted and then written into the corresponding entry of the document index. Finally, the provider encrypts this index and outsources it to the cloud with the encrypted documents. In essence, this index is a word frequency table encrypted by the computable encryption algorithm. Some studies have utilized the Fast Accumulated Hash (FAH) algorithm to achieve these purposes [11], [12], [13].

Trapdoor generation process: To perform a search request, the user first authenticates with the provider. During authentication, the provide would send its secret key to the user to decrypt the documents stored in cloud.

Once authenticated, the user would send the search keywords to the provider. The provider then computes trapdoors, commonly with FAH algorithms and replies back. In such case, two round trips are required (authentication and trapdoor generation) for a user to obtain the trapdoor for the search keywords.

Document retrieval process: In this process, the user sends the noised trapdoor to the cloud. The cloud then removes noise in the trapdoor and searches the indexes with a search algorithm. When documents are found, the cloud ranks them according to each document's score. Then the top-k relevant documents are chosen and sent to the user. Finally, they are decrypted and recovered by the user. In general, the Ranked Serial Search (RSS) algorithm [14], [15] is chosen as the search algorithm.

2.2 Network Traffic Inefficiency Problem

As noticed in the trapdoor generation process, the trapdoor is traditionally generated by the provider to provide data security. However, in such case, the trapdoors would need to be transmitted twice per request (between the provider and the user plus between the user and cloud). Figure 1 depicts the search flow with two network communication round trips for traditional systems, including trapdoor generation process and document retrieval process. Here we do not care for the authentication process as well as transmitting target documents from the cloud to the user. So the total network traffic of the traditional system depends on network traffic cost when generating trapdoors.

Then we analyze the network traffic cost of the traditional system with two network round trips, which is shown in Figure 2.

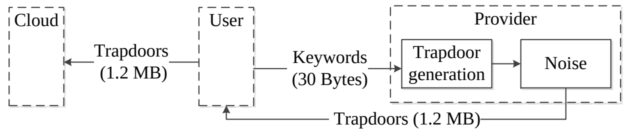


Figure 2. Trapdoor Generation in the Traditional System

From Figure 2, we calculate the network traffic cost by Equation (1).

$$\begin{aligned} N_{trr} &= 2 \times S_{trap} + S_{key} \\ &= 2 \times 1.2MB + 30Bytes \\ &\approx 2.4MB \end{aligned} \tag{1}$$

, where N_{trr} represents the total network traffic cost, S_{trap} is as the size of the trapdoor, and S_{key} is as the size of the keyword.

According to Equation (1), only one search request costs so high network traffic that this two-round-trip network communication is inefficient for users in the mobile cloud environment.

2.3 Search Time Inefficiency Problem

The search delay mainly composes the trapdoor generation time and document search time. Trapdoor generation time faces challenges in mobile wireless networks: high communication latency, poor connectivity and low network transmission rate. According to Figure 2, we could calculate the trapdoor generation time by Equation (2). The traditional system requires users to transmit the trapdoor twice (up to 1.2MB a time), which could easily reach 300ms.

$$T_{trr} = 2 \times T_{net} + T_{gen} + T_{noi} \tag{2}$$

,where T_{trr} represents the total time delay, T_{net} is as the time delay of one round trip, T_{gen} is as the time delay of trapdoor generation, and T_{noi} is as the time delay of adding noise in the trapdoor.

Also according to our measurement, the trapdoor generation (steps 5 to 8 in Figure 1) time accounts for 59.9% of the total search delay.

On the other hand, document retrieval time depends on the search algorithm in the cloud. The RSS algorithm is often used to retrieve documents in the cloud (steps 9 and 10 in Figure 1), which ranks the documents according to relevance scores. However, it must undergo a 3-level iteration to obtain related documents, and its time complexity is about $O(n^3)$. This search process is significantly inefficient, leading to a long retrieval time in the mobile cloud that is not feasible for the user. This work addresses these challenges with an innovative, efficient encrypted search scheme that can be used over mobile cloud, as described in Sections 3.

3 EnDAS Design

This section introduces the design of the EnDAS system and retrofitted trapdoor generation process in EnDAS. Compared the EnDAS system (Figure 3) with traditional system (Figure 1), the main difference is that (1) network traffic is reduced by a single round trip information exchange and the trapdoor compression method; and (2) the search time is reduced by the RSBS algorithm and the TMT module; and (3) the computing burden for generating trapdoors is also offloaded by the TMT module. Aforementioned performance benefits are enabled by a retrofitted trapdoor generation process (Section 3.2) and a retrofitted search algorithm (Section 3.3).

3.1 Architecture of the EnDAS System

Figure 3 shows the search flow in EnDAS system. The trapdoor generation process and the cloud search algorithm are retrofitted to reduce search delay and network traffic.

For trapdoor generation, EnDAS stores a pre-computed Trapdoor Mapping Table (TMT) in mobile devices, which maps common English words to corresponding trapdoors. When the mobile device initiates a search request, the trapdoor is looked up from the table instead of being requested from the provider. This optimization saves one network round trip for the trapdoor generation. Furthermore, EnDAS also provides

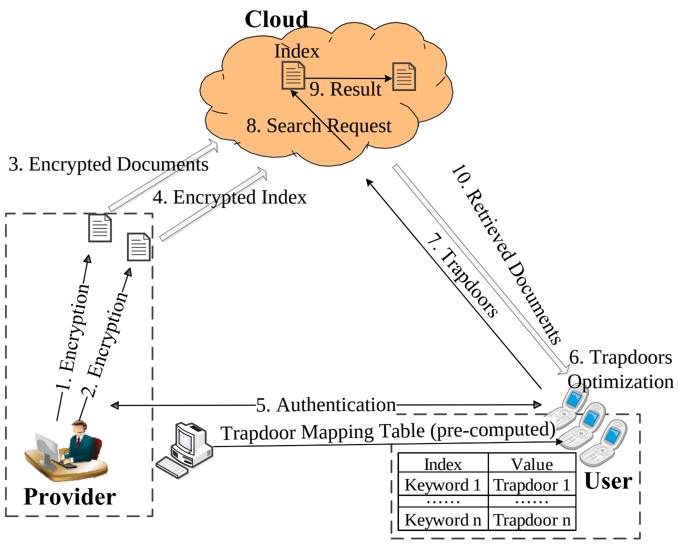


Figure 3. EnDAS system over mobile cloud

new algorithms to optimize and compress trapdoors to reduce network traffic to transmit trapdoors. Section 3.2 will elaborate the details of the EnDAS trapdoor generation process.

For the search algorithm, EnDAS proposes to leverage a binary tree structure to reduce the lookup costs and thus improve the search responsiveness. Section 3.3 would further explain the details.

3.2 Retrofitted Trapdoor Generation Process

The retrofitted trapdoor generation process is described in this subsection, as shown in Figure 5 and Algorithm 1. This process includes the trapdoor mapping table and the trapdoor compression algorithm.

3.2.1 Overview

With retrofitted trapdoor generation process, it is not necessary for an authenticated user calculate pure trapdoors (which will incur heavy computation). After a keyword is stemmed, an user can just query the trapdoor mapping table for the trapdoors, as shown in Algorithm 1. Since the trapdoor mapping table stores the information needed for mapping and search, the heavy computation for generating trapdoors is not needed to be conducted online. This not only avoids the recalculation if the term is found, but also reduces the number of necessary round trips from two to one.

However, it is inevitable that the trapdoors of some keywords have not been stored in the trapdoor mapping table in advance. In this case, the keyword is encrypted by the user(instead of the provider (line 1 to 6 in Algorithm 1). Then the newly retrieved or generated pure trapdoor is added with some noises from a noise set Θ, to prevent the cloud from examining the same trapdoors, shown in line 7 of Algorithm 1.

Note that the size of the trapdoor and the noise can be too large to bring too much burden for the transmission. To address this issue, as shown in line 8 in Algorithm 1, a lightweight trapdoor compression method is used to extract each trapdoors characteristic bits, record as well as accumulate location of each characteristic bit in order, and transmit the compressed trapdoor to the cloud. Since these characteristic bits only occupy a small proportion in this trapdoor, the compressed trapdoor will lead to additional reduced traffic cost for transmitting the trapdoors to the cloud.

We will elaborate the two aforementioned components in the following.

3.2.2 Trapdoor Mapping Table Module

We found that there was a long calculation time from building the trapdoor on the provider side. In a traditional system, the calculation of generating a trapdoor of a given keyword is constituted by term stemming, encryption and adding noise by the provider. Among these three steps, it is shown in Figure 4 that the time of encryption stands for a significant proportion of the the total trapdoor calculation time.

Figure 4 displays three columns, denoting the total calculation time for generating trapdoors for one keyword, two keywords and three keywords respectively. As shown in Figure 4, the encryption time occupies nearly 85% of the total calculation time. This is because that the encryption operation requires more computing resources than others, as it accumulates all terms together to generate a hash code.

To reduce trapdoor construction time, our method ships the encryption process from the online approach to offline. Furthermore, the trapdoor generation process utilizes a Trapdoor Mapping Table (TMT), which stores a

Algorithm 1 Trapdoor Generation Process

```
Input:
    Keyword: $K$
    Hash function in FAH algorithm: $H()$
    Mapping function in FAH algorithm: $G()$
    Noise set: $\Theta = \{\varepsilon_1, \varepsilon_2, \dots \varepsilon_p\}$
Output:
    Index: Compressed trapdoor $\bar{\tau}_t{}'$
1: Extract the term $t$ from $K$.
2: if the term $t$ hits in the TMT module then
3:     Obtain its pure trapdoor without any noise.
4: else
5:     Hash it by $H()$ and get its $l$-bit hash code $\tau_t = H(t)$;
       Map $\tau_t$ to $\bar{\tau}_t = \{0,1\}^r$ by $G()$, which contains $r$ bits
6: end if
7: Choose $q$ noises from the noise set $\Theta$ to build a subset
   $\varepsilon = \{\varepsilon_1, \varepsilon_2, \dots \varepsilon_q\}$, and accumulate it with $\bar{\tau}_t$ to get $\bar{\tau}_t \bigwedge \varepsilon$.
8: Calculate the location of each characteristic bit 0 in $\bar{\tau}_t \bigwedge \varepsilon$
   by utilizing an $m$-bit $\{0,1\}$ codes to record this location ($r = 2^m$), accumulate values of locations in order
   $\underbrace{\{0,1\}^m \bigwedge \{0,1\}^m \bigwedge \dots \{0,1\}^m}_{f}$ , get a compressed trapdoor $\bar{\tau}_t{}' = \{0,1\}^{f \times m}$ ($f$ as the number of characteristic bits).
9: return $\bar{\tau}_t{}'$ .
```

large amount of frequently-used trapdoors (since an English vocabulary of just 3,000 words provides coverage for around 95% of common texts [16], here we assume a proper size of keywords is about 3,000 words) calculated offline. The key for this trapdoor mapping table is a term from stemmed keywords, while its value corresponds to encrypted terms (a pure trapdoor without any noise).

Next we analyzed the availability of the TMT module. According to our measurement, we found that in 20,000 trapdoors, the size of more than 80% of trapdoors ranges from 20 to 60 bytes. That is, encrypted keywords have a small size. So we selected 5,893 different words (including 3,000 common words and 2,893 rare/uncommon words [17]) as keywords to be encrypted, and then stored them in TMT module. We achieve that the actual size of TMT was about 0.31 MB, according to our measurement. Although some rare words are not in the TMT module, users rarely search documents with them, and therefore we can fully ignore these words.

Figure 3 indicates that this TMT requires only one transfer to the user, while its size is smaller than the size of one noised trapdoor (0.4 MB) from the provider to the user (step 7 in Figure 1). That is, the TMT module saves not only traffic but also search time. TMT module does not require the provider to compute trapdoors through expensive communication between the provider and the user, while it only requires the user to look up trapdoors, avoiding re-computing trapdoors. It reduces network round trips for trapdoor generation from two to one, which is shown in Figure 3.

We analyze the performance of EnDAS in search time when generating trapdoors. Utilizing TMT module, EnDAS has only one network round trip used to search target documents, which is shown in Figure 3. The total search delay of EnDAS when generating can be calculated by Equation (3)

$$T_{srr} = 1 \times T_{net} + T_{look} + T_{noi} + T_{comp} \tag{3}$$

,where T_{srr} represents the total time delay, T_{net} is as the time delay of one round trip, T_{gen} is as the time delay of trapdoor lookup, T_{noi} is as the time delay of adding noise in the trapdoor, and T_{comp} is as the time delay of compressing noised trapdoor.

Comparing Equation (3) with Equation (2), we find that $T_{srr} < T_{trr}$. This is because EnDAS is only required to look up, noise and compress trapdoors, rather than encrypting trapdoors. In essence, the noise method is that accumulating noises chosen from the noise set after each trapdoor in order would consume a few time. On the other hand, the trapdoor compression method also causes a few time, and the reason will be elaborated in Section 3.2.3. Looking up trapdoors in the TMT module spends so few computing resources that we can fully ignore it. According to our measurement, encrypting trapdoors in the traditional system costs much longer calculation time (85% of total time for trapdoor generation) than other operations on trapdoors (e.g. noise). This is because the encryption operation requires more computing resources than others since it accumulates all terms together to achieve a hash code. And this conclusion is shown in Figure 4, which displays three columns, denoting a single keyword, two keywords and three keywords respectively.

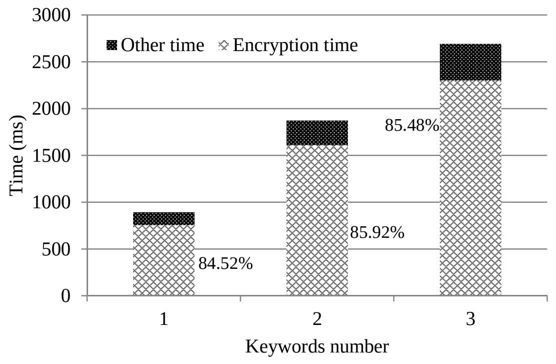


Figure 4. Trapdoor calculation time

3.2.3 *Trapdoor Compression*

We now introduce the lightweight trapdoor compression method. The key idea behind this trapdoor compression method is that we utilize the location of each trapdoor's characteristic bit to represent this trapdoor, since characteristic bit 0 can show all the features of the trapdoor and also occupy a much smaller proportion compared with non-characteristic bit 1. We first analyze the availability and then provide the detailed design for the compression method.

After the trapdoor is constructed, we get an r-bit trapdoor without any noise, in which the proportion of characteristic bits is small. The expectation of the characteristic bit 0 in one pure trapdoor is $\frac{r}{2^d}$, as described in Section 3.3.3.

Notes that we see the noises in set Θ consist of n various noises (Section 3.3.4), and each noise is also a r-bit hash code (pure trapdoor). However, each term is not yet part of the document set before encrypted, thus these noises differs from any trapdoor. Here we select q noises in the noise set Θ $(q \leq n)$ so that a sub-noise set can be achieved as $\varepsilon = \{\varepsilon_1, \varepsilon_2 \dots \varepsilon_q\}$. They are used to optimize each trapdoor, and each noise can be represented as $\varepsilon_i = \{0,1\}^r$, where r represents the number of bits in each noise.

In the worst case of $\varepsilon = \Theta$, we choose all the noises (p various noises) to noise each trapdoor. So for an r-bit pure trapdoor, we get a *(p+1)r*-bit nosied trapdoor. If each noise has no characteristic bit, the number of characteristic bits in this trapdoor are also $\frac{r}{2^d}$ in the best scenario. However, in the worst case scenario, if each noise has r characteristic bits, the number of characteristic bits in this trapdoor is $\frac{(p+1) \times r}{2^d}$. Furthermore, in the worst case, noises seriously impact each term's characteristic bit. Therefore, we estimate that the ratio of the numbers of 0 and 1 is almost $\frac{\frac{(p+1)\times r}{2^d}}{r-\frac{(p+1)\times r}{2^d}} = \frac{(p+1)}{2^d-p-1}$. From this result, we can deduce that the number of 0 will be much less than the number of 1. This is a very important attribute. This analysis demonstrates that adding noise into the trapdoor still ensures a small proportion for characteristic bit 0.

Accordingly, we present a lightweight trapdoor compression method by accumulating locations of term's characteristic bits to alleviate much of the burden of the transmission process, as shown in line 8 of Algorithm1. Here we elaborate the compression method in details. After each trapdoor is noised, it is defined as $\{0,1\}^{(k+1)r}, k \leq n$, where n is the number of noises in the noise set Θ. Now we define r as 2^m, so that the location of each characteristic bit in the trapdoor can be represented as an *m*-bit code, while all non-characteristic bits are ignored. We easily get each characteristic bit's location value and then accumulate them in order. This new accumulated code created by the locations of characteristic bits is considered as the compressed trapdoor, which is finally sent to the cloud.

In fact, the trapdoor compression method brings not only a few time delay but also much more network traffic reduction. This is because the trapdoor compression method is to extract all the characteristic bits 0, record their location values and accumulate these location values. Since all the characteristic bits only occupy a much smaller portion in the trapdoor, the compression process for characteristic bits 0 would cost less computing resources compared with encrypting the keyword. The process of encrypting the keyword is that all the characteristic bits 0 and non-characteristic bits 1 are required to be calculated twice, including dividing and mapping. And this keyword encryption process is the same as the process of term encryption when encrypting index in the provider side, as described in Section 3.3.3. So we easily achieve $T_{look} + T_{comp} < T_{gen}$.

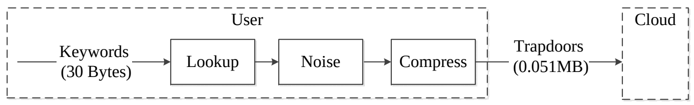


Figure 5. Trapdoor Generation in EnDAS System

Then we elaborate on the network traffic of EnDAS when generating trapdoors, as shown in Figure 5. In the EnDAS system, the network traffic is reduced by the trapdoor compression method and single network round trip, compared with the traditional system. We easily obtain that the total network traffic N_{srr} is only 0.051MB, which is much smaller than the total network traffic of the traditional system N_{trr} (2.4MB) calculated by Equation (1). This is because the compressed trapdoor is only composed by a few location values of characteristic bits 0, the size of which is smaller than the size of the whole trapdoor.

3.3 Efficient Search Algorithm

The efficient search algorithm proposed by EnDAS relies on a binary search tree structure to accelerate indexing. In the section, we will first introduce the conventional privacy-preserving index construction procedures, including index construction (Section 3.3.1), index slicing (Section 3.3.2) as well as index encryption (Section 3.3.3), and then elaborate our binary search tree construction (Section 3.3.4) to accelerate index matching. Finally we will present our RSBS algorithm (Section 3.3.5) which leverages this data structure to perform privacy-preserving searches more efficiently.

3.3.1 Document Index Construction

The cloud uses the indexes provided by the provider to quickly search documents. The provider is responsible for constructing document indexes and sends to the cloud. In general, two important matrices are commonly used [18] to generated the index of documents. The Term-Frequency (TF) matrix denotes the frequency of each term in documents. The Inverted Document Frequency (IDF) matrix depicts the significance of rare terms that are used to distinguish documents. The multiplication of these two matrices, which produces the score matrix A. The matrix A will be encrypted and outsourced to the cloud, rather than traditional TF matrix and IDF matrix. This avoids multiplication operation ($TF \times IDF$) when searching documents score in the cloud. Suppose we have N documents and T terms, matrix A is a N-by-T matrix. Each element $RS_{t,c}$ stands for the relevance score of term *t* in document *c*, for a particular document *c*, $c \in \{1, \ldots N\}$ and a term *t*, $t \in \{1, \ldots, T\}$. We use the column vectors I_c of matrix A as the index for a particular document. These notations have the following relationships:

$$A = (I_1, I_2, \ldots, I_N) = \begin{pmatrix} RS_{1,1} & RS_{1,2} & \ldots & RS_{1,N} \\ RS_{2,1} & RS_{2,2} & \ldots & RS_{2,N} \\ \vdots & \vdots & \ddots & \vdots \\ RS_{T,1} & RS_{T,2} & \ldots & RS_{T,N} \end{pmatrix} \tag{4}$$

We compute the score for a particular document *c* as the sum of all elements within its index I_c, which is

$$Score_c = \sum_{t=1}^{T} RS_{t,c} \tag{5}$$

3.3.2 Index Slicing

After the plain-text document indexes are produced, the provider then divides each index into s slices s ($s \leq T$) according to the score value. We elaborate this process as follows.

According to score value, we divide the index I_c into s slices, and each slice has a normalized score value. And terms in one slice, such as the slice $Slice_c$, are given a same score value as the normalized score of this slice.

According to this principle, we define the slice $Slice_c$ as the *j*th slice of the index I_c, so its score scope $Scope_{j,c} = [\frac{max(RS_{t,c})-min(RS_{t,c})}{s} \times (j-1), \frac{max(RS_{t,c})-min(RS_{t,c})}{s} \times j]$ ($t \in \{1,\dots,T\}, c \in \{1,\dots N\}, 1 \leq j \leq s$), where $max(RS_{t,c})$ denotes the maximum score of the matrix A, $min(RS_{t,c})$ denotes the minimum score of the matrix A.

Then we select a score value w ($w \in Scope_{1,c}$) as the normalized score of the first slice in the index I_c, so we define the score of the slice $Slice_c$ as $Score_{j,c}$ equals $wj, wj \in Scope_{j,c}$. Note that we do not care for the specific score value, and only focus on slice sort, such as $Score_{j-1,c} \leq Score_{j,c} \leq Score_{j+1,c}$. In fact, all terms in the slice $Slice_c$ are given the same score. That is, the score of each term depends on the location of corresponding slice. So we achieve the final score of the document c calculated by Equation (6).

$$Score_c = w \times j, w \in Scope_{1,c}, 1 \leq j \leq s \tag{6}$$

where w denotes a random score value, s denotes the number of slices in the index I_c, j denotes the location of the slice and $Scope_{1,c}$ denotes the score of the first slice in the index I_c.

3.3.3 Index Encryption

The provider then encrypts each index with a given FAH algorithm by encrypting each index's slices, before sending them to the cloud. We base our scheme on previous privacy-preserving searching systems [6], [11], [15].

Here the FAH encryption algorithm for document indexes is employed in previous literature [6]. Utilizing this FAH algorithm, we encrypt slices of each index. The detailed encryption process for one slice $Slice_c$ of the index I_c is that encrypting *l*-bit term t in $Slice_c$ is used by the hash function $H()$, and mapping *l*-bit encrypted term τ_t into *r*-bit optimized term $\bar{\tau}_t$ is by the mapping function *G()*, where $l = d \times r$; and then accumulating all the *r*-bit optimized terms together. Finally we get the encrypted slice $Slice_c^{'}$. In this way, we can encrypt the index I_c by accumulating all the slices (s slices), and obtain the encrypted index $I_c^{'}$ equals accumulating all the optimized terms in this document, shown as $I_c^{'} = \bar{\tau}_1\bar{\tau}_2\dots\bar{\tau}_T$.

Note that the number of each term's characteristic bit 0 is much less than that of non-characteristic bit 1, and the number ratio of value 0 and 1 is $\frac{\frac{1}{2^d}}{1-\frac{1}{2^d}} = \frac{1}{2^d-1}$. In the same way, we can deduce that the expectation of the number of 0 (characteristic bit) in $\bar{\tau}_t$ is $\frac{r}{2^d}$. From this encryption process, the characteristic bits in each term are preserved, and we can check the existence of the characteristic bits to determine whether a given keyword exists in the index stored in the cloud.

3.3.4 Binary Search Tree for Indexes

In the previous literature [6], searching a trapdoor among indexes is fairly inefficient. In this work, we propose to generate a binary search tree for indexes in order to accelerate the search time.

By utilizing the FAH algorithm, each document's index is processed as a hash code comprised by accumulated terms. With time, we can testify if a given trapdoor appears in this document. If so, we will further identify the slice within the document, which contains the given trapdoor. To accelerate the entire procedure, we construct a binary tree. In this data structure, the top level is a hash code comprised by all accumulated terms. On the second level, each descendant only contains the accumulated terms of half of the index. Further down, all descendants contain the accumulated terms of half of that from its parent. With such structure, the height of the tree is at most s (the number of slices in the index) and thus the search efficiency is $O(s)$. Table 1 depicts an example.

Table 1
The Binary Search Tree for Index $I_c^{'}$

$\bar{\tau}_1\bar{\tau}_2\dots\bar{\tau}_T$				
$\bar{\tau}_1\dots\bar{\tau}_{\lfloor\frac{T}{2}\rfloor}$			$\bar{\tau}_{\lfloor\frac{T}{2}+1\rfloor}\dots\bar{\tau}_T$	
$\dots$				
$\bar{\tau}_1$	$\bar{\tau}_2$	$\bar{\tau}_3$	$\dots$	$\bar{\tau}_T$

Before sending the binary search tree to the cloud, the provider will add noise to the index to prevent statistical privacy leakage. Here, the method proposed in [6] is used in the EnDAS system. We define a noise set Θ, where each noise is accumulated to an index.

3.3.5 RSBS Algorithm

Upon receiving a trapdoor (encrypted form of search keywords), the cloud would perform a privacy-preserving search from the indexes provided by the provider. Then it selects top-*k* documents that contain the given search keywords. This process is achieved by using the RSBS algorithm shown in Algorithm 2.

Algorithm 2 Ranked Serial Binary Search (RSBS) algorithm

```
Input:
    Noised trapdoors (one per search keyword): $\bar{\tau_1}^{'}$,...,$\bar{\tau_e}^{'}$
    Encrypted document indexes: $A = \bar{I}_1^{'} \cdots \bar{I}_N^{'}$
    The number of documents to return: $k$
Output:
    Top-$k$ documents that best match the search request:$D = \{D_1, D_2, \cdots, D_k\}$
1: $Scores = zeros(0, N)$ // create an array of N zeros
2: for $i := 1$ to $N$ do
3:     for $n := 1$ to $e$ do
4:         Score[i] ← Score[i] + bsearch($\bar{\tau_n}^{'}$,$\bar{I}_i^{'}$,1,$s_i$) // search if
           the keyword appears in any of the $s$ slices of the
           document
5:     end for
6: end for
7: sorted, indices = sort(Scores) // sort the score array and
   get the indices or old element in the sorted array.
8: $D \leftarrow indices[0 : k-1]$ // get the top-k documents
9: return $D$
```

The RSBS algorithm aims to find the top-*k* documents that best match the search keywords provided by the user. To this end, it maintains a score array for each document. The main idea is to compute accumulated scores for each document and then selects the top-*k* ones. Thus, RSBS has two layers of loops one line 2 and 3. The inner most part (line 4) calculates the score of a give keyword in a given document, with our binary search mechanism. The binary search will start from the binary tree we constructed and descend to a slice that contains the keyword or find that the keyword does not appear in the document. If the keyword appears in the document, then the score will be calculated by Equation (6) and updated to the $Scores$ array. Otherwise, a zero will be recorded.

Time complexity analysis. The RSBS algorithm traverses through all documents and all keywords in user's search request, which makes the inner-most body (line 4) iterated for eN times. Here e represents the number of keywords provided by the user, and N represents the number of documents. In each iteration, the binary search will be executed (line 4), and its time complexity is $O(log(s))$ (s slices in each index). Thus RSBS algorithm has a time complexity of $O(eNlog(s))$. Comparing with traditional systems with a time complexity of $O(eNs)$, RSBS can effectively reduce the search time by utilizing the binary search. In practice, RSBS algorithm can be further parallelized to compute eN binary searches concurrently, which could further reduce its actual execution time.

4 Evaluation

In this section, we analyze and evaluate the EnDAS system's performance in network traffic and search time. Next, we introduce the experimental environment and evaluate it in detail.

4.1 Experimental Environment

To evaluate the EnDAS system, we implemented our system on the private cloud with Openstack Essex [19] from our lab. We rented a virtual machine with 8G memory for the cloud. We also implemented the RSBS algorithm, written as a python program, to search and return the retrieved documents to the user. Here, the user utilized a mobile device utilized an Android tablet with a Cortex-A9 Quad 1.4GHz CPU, and 2GB memory. The tablet is connected to a mobile network with 72Mbps rate. The trapdoor mapping table is pre-computed on a PC and uploaded to the mobile device before experiments, which consumes 0.31MB of device storage.

The encrypted document set used here is the corpus of 2,386 VOA news [20] extracted from the web site covering subjects such as politics, education, economy, military, etc. The number of terms in each news item is fewer than 2^{11}. For simplicity, we choose $r = 2^{16}$ bits. In order to facilitate construction of the final long hash code, we let $d = 13$. That is, the number of terms to be accumulated is not more than 2^{13}. We can achieve the final $l = r \times d = 851,968$, which means that each term will generate a 851,968-bit hash code. The optimized hash code is 65,536-bit by extracting the characteristic bits. The 851,968-bit hash code is generated by using MD5, SHA1 and SHA2 algorithms, recursively. We also generated 50 noise keywords randomly. We will compare the search performance of EnDAS, the traditional encrypted search systems and the plain-text search systems with no encryption.

4.2 Search Time Evaluation

To reduce the search time and improve the calculation efficiency, we utilized the TMT module and the RSBS algorithm in the EnDAS system. In this part, we first evaluate the overall search time and its breakdown. Then we present the performance or the RSBS algorithm in terms of the search time.

4.2.1 *Performance of EnDAS in Search Time*

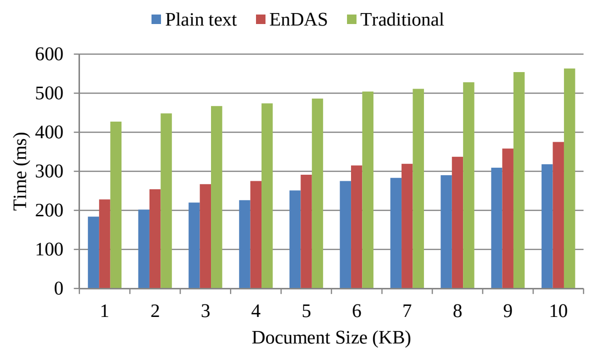


Figure 6. The Performance Comparison in Search Time

With the TMT module and the RSBS algorithm, we evaluate the EnDAS system's overall performance compared with Traditional and Plain text. We choose 10,000 keywords to search target documents and evaluate the search time for the three schemes with the document size ranging from 1KB to 10KB. Results are shown in Figure 6.

In Figure 6, we see that EnDAS saves about 47% of the time compared to Traditional text for 1 KB documents, and by 34% for 10KB documents. Notice that the search time of EnDAS is not much more than that of plain text. In fact, these results benefit from RSBS algorithm, but also the TMT module which reduces one network communication round trip.

To further analyze the overall search process for a search request, we evaluated the detailed search time, as shown in Table 2. In this table, we see that two network round trips in Traditional text cost around 250 ms to build a noised trapdoor, while single network round trip in EnDAS only spends 109.79 ms, and the time reduced for building a trapdoor mainly benefits from the TMT module. In addition, the transmission time for a compressed trapdoor with a small size (from the user to the cloud) is much less than that for an original trapdoor. We discuss the reduced search time for documents in the cloud in Section 4.2.2. In short, the EnDAS system outperforms the traditional system in terms of search time.

Table 2
The Comparison of Search Time Breakdown
(U as User, P as Provider, C as Cloud)

Overall Search Process	Traditional (ms)	EnDAS (ms)
Authentication	55.52	55.61
Transmitting a Keyword (U to P)	28.27	N/A
Building a Trapdoor	174.42	109.79
Transmitting a Trapdoor (P to U)	44.77	N/A
Transmitting a Trapdoor (U to C)	42.68	25.73
Searching Documents in C	69.44	17.21
Documents Retrieval	90.78	87.45
Total Search Time	505.88	295.79

4.2.2 Cloud Search Time with RSBS Algorithm

The RSBS algorithm features a binary search compared with the RSS algorithm. Now we emphasize the search time in the cloud with RSBS algorithm and compare it with RSS algorithm shown in Table 3.

In traditional systems, the index without binary optimization is only the TF-IDF index, while the optimized index A is used in EnDAS. In this study, we divided each document's index into 550 slices; that is, in EnDAS, each document's index has 550*2-1=1,099 columns after they are optimized with the binary tree principle. We conducted 10,000 queries with random chosen keywords for the single keyword search, the two keyword search and the three keyword search, respectively. Search time is shown in Table 3.

Table 3 indicates that the search time of the cloud with RSBS is about 60x shorter than that of RSS. This also indicates that $O(eNs) > O(eNlog(s))$. The binary search and the score calculation method shown in Equation (6), which only requires the additive operation, revealed these interesting results. In this way, the RSBS algorithm has a better search efficiency than the RSS algorithm.

To evaluate RSBS's query accuracy, we build indexes with different slice numbers and search top-k documents with two keywords ('government' and 'security'), and

Table 3
The Comparison of Search Time in Mobile Cloud

Keywords	RSS (ms)	RSBS (ms)
1	54.14	0.935
2	113.62	1.922
3	160.91	2.745

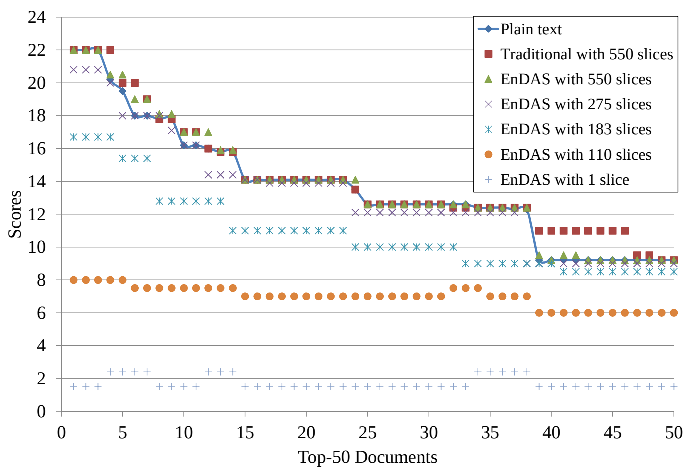


Figure 7. Scores with Different Slice Numbers

the result is as Figure 7. In this experiment, the blue line in the Figure 7 is the result calculated from Plain text, and we choose it as the standard rank. As for accuracy, we just need to care about the rank, rather than the detailed scores, and whether the rank is in accordance with the standard rank. In Figure 7, with the fine-grained sliced index, EnDAS and Traditional system are similar with Plain text, that is, fine grain partition for index can ensure RSBS's query accuracy.

4.3 Network Traffic Evaluation

In the EnDAS system, which benefits from the trapdoor compression method and the TMT module, we reduced network traffic significantly. Next we evaluate and analyze the overall system network traffic reduction and the performance of the trapdoor compression method.

4.3.1 Performance of EnDAS in Network Traffic

Assisted by the trapdoor compression method and the TMT module, EnDAS costs less network traffic than the traditional system. We evaluate this in the subsection. Figure 8 shows the throughput comparison of Plain text, EnDAS and the Traditional system.

We see that the transmission speed for the 1KB-size document is most effective, and the speed increases from 32 KB/s to 65 KB/s. Even if the document is 10 KB in size, the transmission speed is also effective (a 21% improvement). In addition, the throughput of EnDAS is almost similar as that of Plain text. In a word, the EnDAS system outperforms the traditional system in terms of network traffic costs.

4.3.2 Performance of Trapdoor Compression

In this subsection, we test the trapdoor compression's effectiveness on the noised trapdoor, and we randomly choose 10,000 keywords from the corpus and generate the corresponding trapdoor.

For the pure trapdoor without any noise, its length is $r = 2^{16}$ bits, so we need 16 bits to denote each

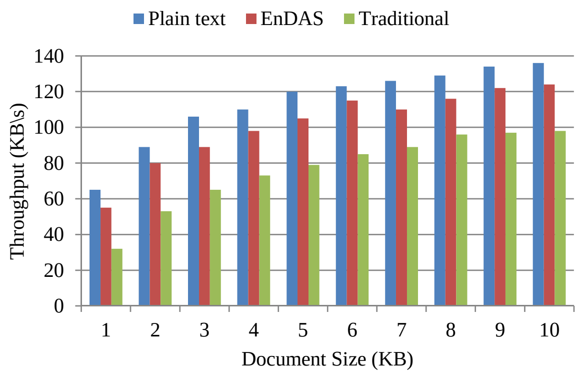


Figure 8. The Throughput Comparison

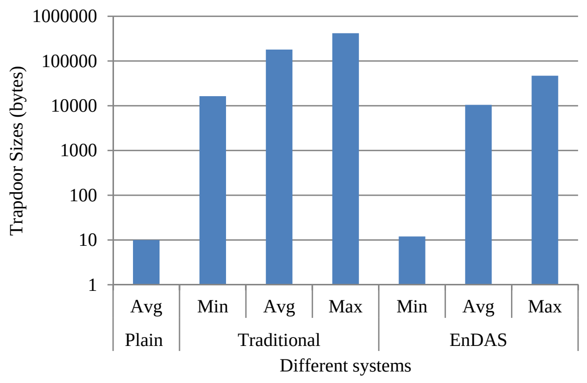


Figure 10. The Comparison of Trapdoor Sizes

characteristic bit's location. However, after adding some noises into the pure trapdoor, we cannot get the length of the noised trapdoor. Therefore, we choose different numbers of pure trapdoors to add noise, which are randomly selected from noise set Θ. The results are shown in Figure 9. We find that the average length of trapdoors with some noise is about $20 \times r$bits; that is, each pure trapdoor is optimized by 19 noises, as shown in Figure 9.

Then we compare the size of plain text with that of the corresponding trapdoor in the traditional system and the EnDAS system. The results are shown in Figure 10. The average length of the trapdoor in EnDAS is reduced by 95%, while even in the worst case scenario, the length of the trapdoor in EnDAS is reduced by 89%. Note that in the best case scenario, the length of the compressed trapdoor is equal to its corresponding plain text. This indicates that this simple compression method is highly effective, which could reduce network traffic costs.

5 Related Work

Recently, many studies have focused on encrypted search schemes to protect data security and improve search efficiency. For data security, we mainly introduce encryption algorithms and noise methods, while for performance efficiency, we mainly introduce search algorithms, including the Boolean keyword search algorithm and the Ranked keyword search algorithm.

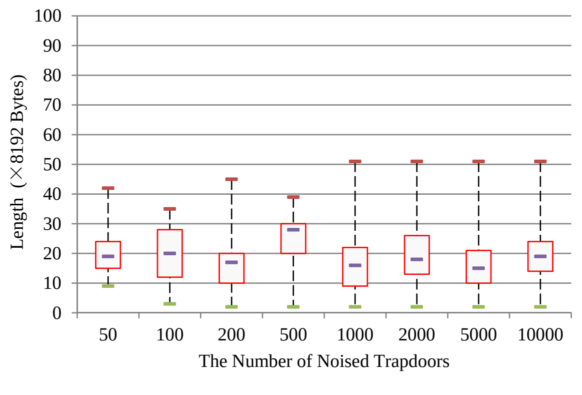


Figure 9. The Average Length of Each Noised Trapdoors

For data security, the previous encryption algorithms cannot directly apply to mobile cloud, because it is hard to achieve efficient network traffic and search time to address the important issues for mobile cloud. Agrawal et al. [21] proposed a one-to-one mapping order preserving encryption method; however, it leads to information leaks. Wang et al. [3] proposed a one-to-many mapping order preserving encryption method that requires a complex computation process, and therefore is not suitable for the mobile cloud. Wang et al. [4] and Swaminathan et al. [22] employed an order-preserving encryption [23] method to retrieve data from encrypted cloud data, which preserved security perfectly. However, this can only be applied in a single-keyword search that retrieves files in a coarse granularity. Some researchers solved this problem through fully homomorphic encryption [5], [24], [25], [26], to retain the security of the encrypted search scheme. In a word, these Order Preserving Encryption (OPE) algorithms [23], [21] and fully homomorphic encryption [5], [24], [25], [26] methods proved themselves secure and accurate enough for searching encrypted data purpose. However, they cost many computing resources. As network traffic and search time efficiency becoming important, a complicated algorithm is not suitable in mobile devices. So we choose an efficient encryption algorithm, fast accumulated hash (FAH)[11], [12], [13], to encrypt document's index and keywords in EnDAS.

Moreover, to protect trapdoor security, adding noise in the trapdoors is a popular method in traditional encryption search schemes. Chen et al. [27] proposed a distributed statistical query scheme with a new noise method. And this noise method was achieved by adding random "coin" value into statistical results. But denoise method is also required. Orencik et al. [6] presented an optimized noised method, which can be easily obtained through adding dummy trapdoor into target trapdoors. To ensure search time efficiency, we select this method in EnDAS and do not delete the noise when searching in the cloud.

For performance efficiency, previous researchers focused on search algorithm in the server side, which can be divided into the Boolean keyword search and the Ranked keyword search from information retrieval concerns. In Boolean keyword searches, documents are searched by the presence and absence of keywords in a document. In other words, it returns ”all-or-nothing,” like [28], [29], [30], [31], [32]. Due to this effect, the Boolean keyword search cannot accurately denote the relevance between files and the keywords, and all files containing the keywords will be returned to the data users. In addition, this method is high-traffic consumptive, which is undesirable and reduces the users' satisfaction.

The ranked keyword search will return documents according to the relevance score. Zerr et al. [33] proposed a novel technique that makes the server side carry out the search operation. However, it should send many unrelated documents back and let the user filter them. This is a waste of traffic, which is unsuitable for the mobile cloud. Bowers et al. [34] proposed a distributed cryptographic system that preserved the security of the document retrieval process and the high availability of the system, but this system suffers from two network round trips and lager calculation complexity for target documents. Wang et al.[3] proposed a single round trip encrypted search scheme, but their system is not secure enough, as it leaks the keyword and associated document information from multiple keyword searches. Li et al. [35] proposed a single-keyword encryption search scheme utilizing ranked keyword search, which optimizes network communication between the user and the cloud by transferring the computing burden from the user to the cloud. But it still suffers from inefficient trapdoor generation process and only supports single-keyword search. Cao et al. [2], [36] realized a multi-keyword search method, but when the field of the record becomes large, their index building procedure would be extremely time-consuming and their trapdoor vector would be very large. Wang et al. [37] and Orencik et al.[14], [6] proposed several novel multi-keyword search methods, but they still suffered from traffic and search time inefficiency due to two network round trips. To achieve search efficiency, we choose a ranked keyword search algorithm and optimize it with binary tree principle. In addition, to address inefficient two network round trips, we utilize a trapdoor mapping table to obtain singe network round trip.

6 Conclusion

In this work, we proposed a novel encrypted search system EnDAS over the mobile cloud, which improves network traffic and search time efficiency compared with the traditional system. We started with a thorough analysis of the traditional encrypted search system and analyzed its bottlenecks in the mobile cloud: network traffic and search time inefficiency. Then we developed an efficient architecture of EnDAS which is suitable for the mobile cloud to address these issues, where we utilized the TMT module and the RSBS algorithm to cope with the inefficient search time issue, while a trapdoor compression method was employed to reduce network traffic costs. Finally our evaluation study experimentally demonstrates the performance advantages of EnDAS.

Acknowledgements

This work was supported by NSFC (Grant No.61272100, 61202374), 863 Program of China (No. 2012AA010905), The Ministry of Education Major Project (No. 313035) and NRF Singapore under its CREATE Program. The corresponding author is Prof. Haibing Guan.

References

[1] D. Huang, “Mobile cloud computing,” *IEEE COMSOC Multimedia Commun. Tech. Committee (MMTC) E-Letter*, vol. 6, no. 10, pp. 27–31, 2011.

[2] N. Cao, C. Wang, M. Li, K. Ren, and W. Lou, “Privacy-preserving multi-keyword ranked search over encrypted cloud data,” in *Proc. Int. Conf. Comput. Commun. (INFOCOM)*, Apr. 2011, pp. 829–837.

[3] C. Wang, N. Cao, K. Ren, and W. Lou, “Enabling secure and efficient ranked keyword search over outsourced cloud data,” *IEEE Trans. Parallel Distrib. Systems*, vol. 23, no. 8, pp. 1467–1479, 2012.

[4] C. Wang, N. Cao, J. Li, K. Ren, and W. Lou, “Secure ranked keyword search over encrypted cloud data,” in *Proc. IEEE Int. Conf. Distrib. Comput. Syst. (ICDCS)*, Jun. 2010, pp. 253–262.

[5] C. Gentry and S. Halevi, “Implementing gentrys fully-homomorphic encryption scheme,” in *Advances in Cryptology–EUROCRYPT 2011*, 2011, pp. 129–148.

[6] C. Örencik and E. Savaş, “Efficient and secure ranked multi-keyword search on encrypted cloud data,” in *Proc. Joint EDBT/ICDT Workshops*, Mar. 2012, pp. 186–195.

[7] Gartner, “Worldwide traditional pc, tablet, ultramobile and mobile phone shipments on pace to grow 7.6 percent in 2014,” http://www.gartner.com/newsroom/id/2645115.

[8] Trellian, “Keywords number,” http://www.keyworddiscovery.com/keyword-stats.html?date=2014-03-01.

[9] V. Rijmen and J. Daemen, “Advanced encryption standard,” *Federal Information Processing Standard*, pp. 19–22, 2001.

[10] X. Lai, “On the design and security of block ciphers,” Ph.D. dissertation, Diss. Techn. Wiss ETH Zürich, Nr. 9752, 1992. Ref.: JL Massey; Korref.: H. Bühlmann, 1992.

[11] K. Nyberg, “Fast accumulated hashing,” in *Proc. Int. Workshop Fast Softw. Encryption (FSE)*, Feb. 1996, pp. 83–87.

[12] Nyberg and Kaisa, “Commutativity in cryptography,” in *Proc. Int. Workshop Funct. Anal.*, 1995.

[13] J. Benaloh and M. De Mare, “One-way accumulators: A decentralized alternative to digital signatures,” in *Advances in Cryptology-EUROCRYPT 1993*, 1994, pp. 274–285.

[14] C. Örencik and E. Savaş, “An efficient privacy-preserving multi-keyword search over encrypted cloud data with ranking,” *Distrib. Parallel Databases*, vol. 32, no. 1, pp. 119–160, Mar. 2014.

[15] P. Wang, H. Wang, and J. Pieprzyk, “An efficient scheme of common secure indices for conjunctive keyword-based retrieval on encrypted data,” pp. 145–159, 2009.

[16] S. Gendreau, “How many words do i need to know? the 95/5 rule in language learning, part 2/2,” http://www.lingholic.com/how-many-words-do-i-need-to-know-the-955-rule-in-language/-learning-part-2.

[17] Manythings.org, “English vocabulary,” http://www.manythings.org/vocabulary/lists/l/.

[18] J. S. Culpepper, G. Navarro, S. J. Puglisi, and A. Turpin, “Top-k ranked document search in general text databases,” in *Proc. Annu. Euro. Conf. Algorithms (ESA)*, Sep. 2010, pp. 194–205.

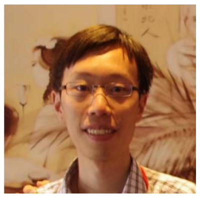
[19] R. cloud computing, “Openstack cloud software,” http://www.openstack.org.

[20] VOANEWS.COM, "Voice of american," http://www.voanews.com.
[21] R. Agrawal, J. Kiernan, R. Srikant, and Y. Xu, "Order preserving encryption for numeric data," in *Proc. ACM SIGMOD Int. Conf. Manag. Data (COMAD)*, Jun. 2004, pp. 563–574.
[22] A. Swaminathan, Y. Mao, G.-M. Su, H. Gou, A. L. Varna, S. He, M. Wu, and D. W. Oard, "Confidentiality-preserving rank-ordered search," in *Proc. ACM Workshop Storage Secur. Survivability (StorageSS)*, Oct. 2007, pp. 7–12.
[23] A. Boldyreva, N. Chenette, Y. Lee, and A. Oneill, "Order-preserving symmetric encryption," in *Advances in Cryptology-EUROCRYPT 2009*, 2009, pp. 224–241.
[24] C. Gentry, "A fully homomorphic encryption scheme," Ph.D. dissertation, Stanford University, 2009.
[25] M. Van Dijk, C. Gentry, S. Halevi, and V. Vaikuntanathan, "Fully homomorphic encryption over the integers," in *Advances in Cryptology-EUROCRYPT 2010*, 2010, pp. 24–43.
[26] D. Stehlé and R. Steinfeld, "Faster fully homomorphic encryption," in *Advances in Cryptology-ASIACRYPT 2010*, 2010, pp. 377–394.
[27] R. Chen, A. Reznichenko, P. Francis, and J. Gehrke, "Towards statistical queries over distributed private user data," in *USENIX Symp. Netw. Syst. Des. Implementation (NSDI)*, vol. 12, 2012, pp. 13–13.
[28] D. X. Song, D. Wagner, and A. Perrig, "Practical techniques for searches on encrypted data," in *Proc. IEEE Symp. Secur. Priv. (SSP)*, May. 2000, pp. 44–55.
[29] E.-J. Goh *et al.*, "Secure indexes," *IACR Cryptology ePrint Archive*, vol. 2003, p. 216, 2003.
[30] D. Boneh, G. Di Crescenzo, R. Ostrovsky, and G. Persiano, "Public key encryption with keyword search," in *Advances in Cryptology-Eurocrypt 2004*, 2004, pp. 506–522.
[31] Y.-C. Chang and M. Mitzenmacher, "Privacy preserving keyword searches on remote encrypted data," in *Proc. Int. Conf. Appl. Cryptogr. Netw. Secur. (ACNS)*, Jun. 2005, pp. 442–455.
[32] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky, "Searchable symmetric encryption: improved definitions and efficient constructions," in *Proc. ACM Conf. Comput. Commun. Secur. (CCS)*, Oct. 2006, pp. 79–88.
[33] S. Zerr, E. Demidova, D. Olmedilla, W. Nejdl, M. Winslett, and S. Mitra, "Zerber: r-confidential indexing for distributed documents," in *Proc. Int. Conf. Extending Database Technol. (EDBT)*, Mar. 2008, pp. 287–298.
[34] K. D. Bowers, A. Juels, and A. Oprea, "Hail: a high-availability and integrity layer for cloud storage," in *Proc. ACM Conf. Comput. Commun. Secur. (CCS)*, Dec. 2009, pp. 187–198.
[35] J. Li, R. Ma, and H. Guan, "Tees: An efficient search scheme over encrypted data on mobile cloud," *IEEE Trans. Cloud Comput.*, Feb. 2015.
[36] N. Cao, C. Wang, M. Li, K. Ren, and W. Lou, "Privacy-preserving multi-keyword ranked search over encrypted cloud data," *IEEE Trans. Parallel Distrib. Systems*, vol. 25, no. 1, pp. 222–233, Jan. 2014.
[37] B. Wang, S. Yu, W. Lou, and Y. T. Hou, "Privacy-preserving multi-keyword fuzzy search over encrypted data in the cloud," in *Proc. Int. Conf. Comput. Commun. (INFOCOM)*, Apr. 2014, pp. 2112–2120.

RuHui Ma received his Ph.D. degree in computer science from Shanghai Jiao Tong University (SJTU), China, in 2011. Now he is an Assistant Professor in SEIEE at SJTU. His main research interests are in virtual machines, computer architecture and compiling.

HaiBing Guan received his Ph.D. degree in computer science from the Tongji University (China), in 1999. He is currently a professor with the Faculty of Computer Science, Shanghai Jiao Tong University (SJTU), Shanghai, China. His current research interests include virtualization and hardware/software co-design.

Jian Li obtained his Ph.D. in Computer Science from the Institute National Polytechnique de Lorraine (INPL) - Nancy, France in 2007. He is an Associate Professor in the School of Software at SJTU. His research interests include real-time scheduling theory, network protocol design and embedded systems.



Mingyuan Xia is a PhD. student at McGill University. His research interests include mobile software systems, storage systems and big data stacks, etc. He received bachelor degree from Shanghai Jiao Tong University.

Xue Liu is an Associate Professor in the School of Computer Science at McGill University. He obtained his Ph.D. in Computer Science from the University of Illinois at Urbana-Champaign in 2006. His research interests include green IT, embedded systems and networking.