A Parallelized Disjunctive Query based Searchable Encryption Scheme for Big Data

Shahzaib Tahir[a,b,*], Liutauras Steponkus[a], Sushmita Ruj[c], Muttukrishnan Rajarajan[a], Ali Sajjad[d]

[a] *School of Mathematics, Computer Science and Engineering, City, University of London, UK.*
[b] *Department of Information Security, National University of Sciences and Technology, Islamabad, Pakistan.*
[c] *Indian Statistical Institute, 203 B.T. Road, Kolkata 700108, India.*
[d] *Research and Innovation, British Telecommunications, Adastral Park, Ipswich, UK.*

Abstract

Searchable Encryption (SE) allows a client to search over large amounts of encrypted data outsourced to the Cloud. Although, this helps to maintain the confidentiality of the outsourced data but achieving privacy is a difficult and resource intensive task. With the increase in the query effectiveness, *i.e.*, by shifting from single keyword SE to multi-keyword SE there is a notable drop in the efficiency. This motivates to make use of the advances in the multi-core architectures and multiple threads where the search can be delegated across different threads to perform search in a parallel fashion. The proposed scheme is based on probabilistic trapdoors that are formed by making use of the property of modular inverses. The use of probabilistic trapdoors helps resist distinguishability attacks. The rigorous security analysis helps us to appreciate the advantage of having a probabilistic trapdoor. Furthermore, to validate the performance of the proposed scheme, it is implemented and deployed onto the British Telecommunication's Public Cloud offering and tested over a real speech corpus. The implementation is also extended to anticipate the performance gain by using the multi-core architecture that helps to maintain the lightweight property of the scheme.

Keywords: Searchable Encryption-as-a-Service (SEaaS), Application Encryption Service, Probabilistic Trapdoor, Privacy Preservation, Inverted Index, Multi-threading

1. Introduction

Over the recent years Could Computing has emerged as a platform that allows on demand resource sharing of the computer processing resources and data to the remote device/clients. Previously the cloud was well known to offer services such as SaaS, PaaS

*Corresponding author
Email addresses: shahzaib.tahir@city.ac.uk; shahzaib.tahir@mcs.edu.pk (Shahzaib Tahir), Liutauras.Steponkus.1@city.ac.uk (Liutauras Steponkus), sush@isical.ac.in (Sushmita Ruj), r.muttukrishnan@city.ac.uk (Muttukrishnan Rajarajan), ali.sajjad@bt.com (Ali Sajjad)

and IaaS [1], but, recently the Cloud has emerged as Resource-as-a-Service (RaaS) [2]. Cloud enables ease of access, high availability of resources and reduced infrastructure cost. Considering these advantages, day-by-day people are becoming more and more reliant onto the Cloud and the resources it has to offer. This resources sharing has also given rise to the Cloud as Database-as-a-Service (DBaaS) [3][4][5] due to which massive amount of data is being outsourced to the Cloud. The advancement in the field of computer security has made people aware of the privacy concerns associated with the outsourced data. If the outsourced data is not encrypted prior to being outsourced to the Cloud, it can be prone to confidentiality and privacy concerns. In recent times people would trade-off the security in order to take advantage of the Cloud but now the documents can be encrypted and then outsourced to the Cloud that requires a mechanism to search over the encrypted documents. This gives rise to the need of having a SE scheme. Therefore, the Cloud needs to be coupled with Searchable Encryption as a Service (SEaaS) so that the clients can confidently upload their documents to the Cloud while equally maintaining security and privacy.

There are three challenges related to SE as highlighted in [6], *i.e.*, (a) security and privacy (b) efficiency and (c) query effectiveness. Keeping in view these challenges, the primary aim of designing a SE scheme is to maintain a balance between the security and privacy, query effectiveness and efficiency. The proposed SE scheme enhances the query-effectiveness by allowing multi-keyword disjunctive queries. Unlike a conjunctive query, a disjunctive query means that the keywords to be searched are independent and not inter-related. For example, a conjunctive query would be "A brown fox jumps", whereas a disjunctive query is "Sydney, Hello, Elsevier, IEEE, TrustCom". Considering our use-case it is highly feasible to explore the advances in multi-core architecture to delegate the search for disjunctive queries across multiples threads. This will help enhance the efficiency of the scheme by reducing the return time of the results to the client.

1.1. Our Contributions

In this paper the following contributions are made by extending the SE scheme presented in [7]:

- We enhance the efficiency of the Ranked Multi-keyword Searchable Encryption (RMSE) scheme proposed in [7] by tuning the scheme to enable multi-threaded searching. The proposed scheme is a lightweight ranked multi-keyword SE scheme that supports disjunctive queries. The probabilistic trapdoors resist distinguishability attacks.

- To appreciate the advantage gained from the probabilistic trapdoors we present formal proofs for the security definitions proposed in [8] by mapping them to our proposed construction and later on validate the proposed scheme accordingly.

- We measure the performance two-fold following which for the first time we propose Searchable Encryption-as-a-Service (SEaaS) for British Telecom's Cloud Service. Firstly we implement our proof of concept prototype and deploy it onto the British Telecom's Alpha Cloud offering. Secondly, we tune the scheme to enable multi-threading to achieve parallelization in a multi-core setup. Both variants are tested over an encrypted Telephone Speech corpus (real dataset).

1.2. Organization

Section 2 gives a high level view of our problem by highlighting the system model. This helps to draw attention to the design goals and associated security and privacy concerns. Section 3, discusses the literature by emphasizing their advantages and shortcomings. Section 4 revisits the existing security definitions. Finally, in Section 5 the proposed RMSE allowing multi-threading is presented. The security analysis done in Section 6 validates the security of the proposed scheme by presenting the formal security proofs. The performance analysis is done in Section 7 that includes the analysis of the storage overhead and the computational time. The performance of the scheme is measured by deploying it onto the BT Cloud Server and measuring the performance over multiple threads. The conclusion is drawn towards the end of the paper in Section 8.

2. Problem formulation

This section highlights the problem of Searchable Encryption by presenting the system model. This discussion leads to formally stating the design goals and defining the phases involved in the proposed SE scheme thereafter.

2.1. The System Model

As illustrated in Figure 1, our ranked multi-keyword searchable encryption (RMSE) scheme mainly comprises of two entities: Bob (client/ data owner) and the Cloud Server (CS). Bob possesses a corpus containing N documents $\mathcal{D} = \{D_1, D_2, \cdots, D_N\}$. Bob wants to encrypt and outsource his documents to the CS while being able to search for keyword(s) over the encrypted corpus. Instead of single keyword search, Bob is interested in performing disjunctive multi-keyword search over the encrypted corpus while preserving the privacy of the documents and the search. Bob identifies a set of unique keywords $\mathcal{W} = \{w_1, w_2, \ldots, w_M\}$ from the $\mathcal{D}$ documents and forms a dictionary. There is a high probability that Bob's trapdoor (search query) may be disjunctively mapped to multiple documents, therefore, he wishes to retrieve documents based on some ranking mechanism. The ranking comes with an increase in the computational time as discussed in the Section 7.

In [9], a formula (Equation 1) has been presented that is commonly used for the relevance frequency calculation by researchers [10][11][7][8] for designing rank based SE schemes.

$$RF(W, D) = \sum_{T \in W} \frac{1}{|D|} \cdot (1 + \ln f_{(T,D)}) \cdot \ln(1 + \frac{N}{f_T}) \tag{1}$$

where W denotes the keyword; D denotes the document; $|D|$ denotes length of the document obtained by counting the words appeared in the document D; $f_{(T,D)}$ denotes number of times a keyword W appears within a particular document D; f_T denotes the number of documents in the dataset that contain the keyword W and N denotes the total number of documents in the dataset.

Apart from this ranking function other scoring functions such as Lucene or Juru, after modification may be used. All the scoring functions vary in computational time and quality of outcome. In [12] a comparison of Lucene and Juru has been performed. The proposed construction uses Equation 1 for the ease of comparison with similar existing schemes.

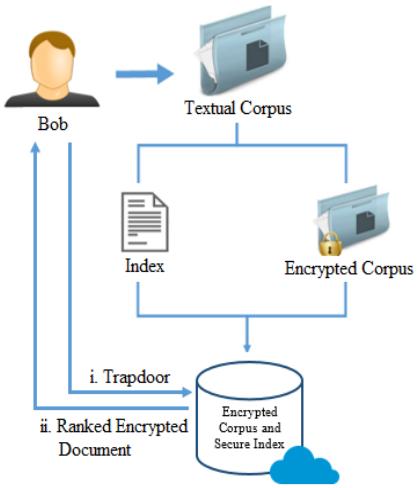

Figure 1: The System Architecture.

Using the master key K, Bob generates a secure ranked index table (I) and outsources it to the CS along with the encrypted documents $\mathcal{D}$. The CS is assumed to be "trusted but curious" (*i.e.*, semi-honest), in other words the CS provides reliable services but it is also interested in learning private information that can be extracted from the outsourced documents, index table or trapdoor. In order to perform a disjunctive keyword search, Bob using his private key generates a valid probabilistic trapdoor and sends it to the CS. The CS splits the trapdoor over different threads that will search in parallel over the secure index table (I) on Bob's behalf and reduce the obtained result to return the encrypted document identifiers in the ranked order, as shown in Figure 2.

The benefits of multi-threading are already highlighted in [13]. Multi-threading coupled with SE offers the following advantages:

- Effective resource sharing especially the resources that will be idle otherwise.
- Accelerated efficiency and reduced search time.
- Utilization of advances made in the multi-core and multi-processor architectures.
- Increase in the responsiveness of the system by performing parallel searches.

From here on, Bob will be represented as a client throughout the rest of the paper.

2.2. Design Goals

The proposed construction bears the following security and performance goals:

- *Trapdoor Unlinkability:* The primary goal is that the trapdoor should be probabilistic, therefore for the same keyword searched again a new trapdoor should be generated. This helps resist distinguishability attacks.
- *Privacy Guarantee:* Apart from the outcome of the search, the CS should not deduce any keyword related information from the secure index and trapdoors.

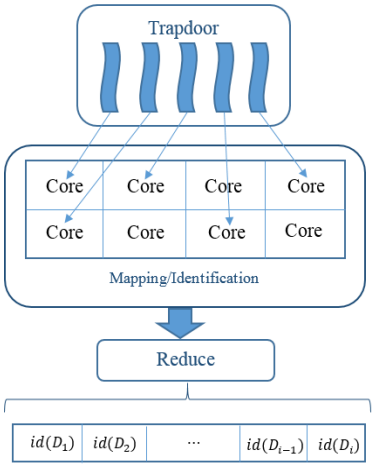

Figure 2: Trapdoor: MapReduce Framework.

- *Lightweight:* The scheme should be lightweight by default. To accelerate the performance of the scheme it should possibly support parallel search.

Now that the goals have been mentioned, we formally define our RMSE Scheme.

Definition (Ranked Multi-Keyword Searchable Encryption Scheme (RMSE)): The proposed RMSE comprises of five polynomial time algorithms Π = (KeyGen, Build_Index, Build_Trap, Search_Outcome, Dec) such that:

$(K, k_s) \leftarrow \text{KeyGen}(1^\lambda)$: is a probabilistic key generation algorithm that takes a security parameter λ as the input. It outputs a master key K and a session key k_s. This algorithm is run by the client.

$(I) \leftarrow \text{Build\_Index}(K, \mathcal{D})$: is a deterministic algorithm that takes the master key K and collection of documents $\mathcal{D}$ as the input. The algorithm returns a secure index I. This algorithm is run by the client.

$T_w \leftarrow \text{Build\_Trap}(K, k_s, w, num)$: is a probabilistic algorithm that takes the master key K, a session key k_s, set of disjunctive keywords $w \in P(\mathcal{W})$, the number (num) of documents $\mathcal{D}$ required as the input. The algorithm returns a trapdoor T_w. The algorithm is run by the client.

$X \leftarrow \text{Search\_Outcome}(k_s, I, T_w)$: is a deterministic algorithm run by the CS. The algorithm takes the session key k_s, index table I and the trapdoor (T_w) as the input and returns X, a set of desired document identifiers encrypted $Enc_K(id(D_i))$ containing the set of disjunctive keywords w in ranked order.

$D_i \leftarrow Dec(K, X)$: is a deterministic algorithm. The algorithm requires the client's master key K and encrypted set of document identifiers $Enc_K(id(D_i))$ to decrypt and recover the document id's. This algorithm is executed by the client.

3. Related Work

As discussed in Section 1, the previous researches fail as they were not able to maintain a balance between security and privacy, query effectiveness and efficiency. This section discusses some of the significant work that has been carried out in the field of SE:

3.1. Single-keyword SE

SE has been an important area of research for over a decade now. Song *et al.*, in [14], were the first to propose practical ways of searching over encrypted documents but there were not any formal definitions associated to SE back then. In [15], the authors had proposed new and improved definitions related to Searchable Encryption. They proposed two efficient constructions that were secure under the newly proposed definitions. Similar to our construction, their scheme was secure against an adaptive adversary. However, their scheme was based on deterministic trapdoors and did not resist distinguishability attacks. Their work was extended by authors in [10][11] to perform ranked based SE. This was the first time the concept of ranking was introduced in SE. The authors presented two schemes for single keyword ranked search over encrypted text. There was an advantage of their later scheme as it supported dynamic inverted index, *i.e.*, whenever a new file was uploaded to the CS the re-ranking of the entire index table was not required. To reduce the leakage associated to the frequency of occurrence of keywords to the server the authors used Order Preserving Symmetric Encryption (OPSE). Although the authors claimed that the proposed scheme is privacy preserving, in [16], the authors demonstrated a successful differential attack on their scheme and hence the index table could not be termed secure anymore.

In [17], authors introduced a construction that did not require a complex data structure such as an inverted index and could do the processing in parallel. Their scheme also supported update of the data. However, the scheme was again dependent upon deterministic trapdoors. They utilized a new tree-based multi-map data structure to index documents. Although their scheme supports modification of data on run time, however, it was again based on deterministic trapdoors.

3.2. Multi-keyword SE

In [18], authors introduced the public-key systems that support conjunctive, subset and range queries over encrypted data. Although the proposed scheme used bilinear maps and may be termed better than the trivial constructions but it was prone to distinguishability attacks as the trapdoors were probabilistic. Similarly, in [19], authors used bilinear maps for performing search but their scheme also lacked in providing indistinguishability.

The concept of "coordinate matching" while performing multi-keyword search was introduced in [20] by Cao *et al.*. The proposed scheme allowed "as many matches as possible" and enabled ranked searching. To generate trapdoors they made use of a vector similar to a bloom filter, where each entry of the bloom filter represents the presence or absence of a keyword. It is also noted that the authors introduced dummy keywords to the documents to hide the leakage associated to the index table. Although this helped to reduce the leakage associated to the index table but the dummy keywords also reduced the accuracy of the results which also effected the ranking functionality.

In [21], authors introduced a conjunctive query enabled ranked SE scheme. As mentioned in the introduction that there is a difference between conjunctive and disjunctive

queries and the scheme needs to be designed accordingly. Although the authors aimed to facilitate "conjunctive keywords" but the search was performed similar to disjunctive keywords. The relevance score generation formula is quite similar Equation (1). The authors used inverted index table for the blind storage. Similar to [20] the authors used a vector/bloom filter to represent a trapdoor, where each element represents the presence or absence of a keyword. They also introduced dummy integers to the vector to increase the privacy of the trapdoor and reducing the leakage. However, the authors did not measure the accuracy of the proposed scheme and the effect that the dummy keywords had on the precision. In [22], authors propose a multi-keyword SE encryption scheme and use homomorphic encryption for the index generation and the trapdoor generation. Although, due to the inefficiency of homomorphic encryption the scheme cannot be deployed onto a real world but they have implemented their scheme using multi-threading. We further refer readers to [23][6] that present a survey of the existing SE schemes.

Unlike the existing schemes, the proposed RMSE scheme is based on probabilistic trapdoors that resist distinguishability attacks. The scheme is based on a secure inverted index table. The inverted index table uses a masking function that prevents from the statistical analysis attacks. Although the trapdoor is probabilistic but it leads to the exact matching and identification of the keyword, hence the scheme does not have any false errors. The scheme allows disjunctive queries and makes use of the advancements in the multi-core processors to perform parallel searching.

4. Security Definitions

In [8], authors discuss the limitations of the previous definitions proposed in [15] for SE and present new definitions that can help appreciate the advantage of having probabilistic trapdoors. To validate the security of the scheme we revisit the security definitions proposed in [8],[7] and extend them to facilitate multi-keyword SE.

4.1. Keyword-Trapdoor Indistinguishability for RMSE

Keyword-Trapdoor Indistinguishability is the ability of a SE scheme to hide and dissipate the redundancy in the statistics of the keywords into the trapdoor. This helps to achieve adaptive security, *i.e.*, for the same keywords being searched multiple times a new and unique trapdoor will be generated. Hence even if the adversary is maintaining a history of keywords and associated trapdoors, it cannot guess the future searches. Therefore, to guess the keywords or the content of the documents a large amount of data needs to be intercepted in polynomial time.

Definition 4.1(Keyword-Trapdoor Indistinguishability): *Let* RMSE=(KeyGen, Build_Index, Build_Trap, Search_Outcome, Dec) *be a Ranked Multi-keyword Searchable Encryption Scheme over a dictionary of keywords* $\mathcal{W} = \{w_1, w_2, \cdots, w_M\}$*, set of documents* $\mathcal{D} = \{D_1, D_2, \cdots, D_N\}$*,* λ *be the security parameter,* $P(\mathcal{W})$ *represent a power set containing* 2^M *possible disjunctive members and* $A_{j;1\leq j\leq M+1}$ *be a non-uniform adversary. Consider the following probabilistic experiment* $Key\_Trap_{RMSE,\mathcal{A}}(\lambda)$:

$Key\_Trap_{RMSE,\mathcal{A}}(\lambda)$
$(K, k_s) \leftarrow KeyGen(1^\lambda)$
$(I) \leftarrow Build\_Index(K, \mathcal{D})$

$for\, 1 \leq i \leq M; M = |P(\mathcal{W})|$
$(st_{\mathcal{A}}, w_i) \leftarrow \mathcal{A}_i(st_{\mathcal{A}}, T_{w_1}, \cdots, T_{w_i})$
$T_{w_i} \leftarrow Build\_Trap_K(w_i)$
$(st_{\mathcal{A}}, w_0, w_1) \leftarrow \mathcal{A}_0(1^\lambda);\ (w_0, w_1) \in P(\mathcal{W})$
$b \xleftarrow{\$} \{0,1\}$
$(T_{w_b}) \leftarrow Build\_Trap(K, k_s, w_b, num)$
$b' \leftarrow \mathcal{A}_{M+1}(st_{\mathcal{A}}, T_{w_b})$
$T'_w \leftarrow Build\_Trap_{k_s}(w_j); j \in M$
$if\, b' = b, output\, 1$
$otherwise\, output\, 0$

where $st_{\mathcal{A}}$ represents a string that captures $\mathcal{A}$'s state. The keyword-trapdoor indistinguishability holds for all the polynomial-size adversaries $(\mathcal{A}_0, \mathcal{A}_1, \cdots, \mathcal{A}_{M+1})$ such that $N = poly(\lambda)$,

$$Pr[Key\_Trap_{RMSE,\mathcal{A}(\lambda)} = 1] \leq \frac{1}{2} + negl(\lambda)$$

4.2. Trapdoor-Index Indistinguishability for RMSE

Trapdoor-Index Indistinguishability refers to the complexity offered by a SE scheme that disables an adversary to identify the index table entries and documents corresponding to the search query prior to the search. This should hold true even if the adversary maintains a history of keywords, trapdoor and outcome of the searches. Such a property can only be achieved if the trapdoors are probabilistic. Therefore, to guess the index table entry corresponding to a trapdoor and associated keywords, a large amount of data needs to be intercepted in polynomial time.

Definition 4.2(Trapdoor-Index Indistinguishability): *Let* RMSE=(KeyGen, Build_Index, Build_Trap, Search_Outcome, Dec) *be a Ranked Multi-keyword Searchable Encryption Scheme over a dictionary of keywords* $\mathcal{W} = \{w_1, w_2, \cdots, w_M\}$*, set of documents* $\mathcal{D} = \{D_1, D_2, \cdots, D_N\}$*,* λ *be the security parameter,* $P(\mathcal{W})$ *represent a power set containing* 2^M *possible disjunctive members and* $\mathcal{A} = (A_0, A_1)$ *be a non-uniform adversary. Consider the following probabilistic experiment* $Key\_Trap_{RMSE,\mathcal{A}}(\lambda)$:

$Trap\_Index_{RMSE,\mathcal{A}}(\lambda)$
$(K, k_s) \leftarrow KeyGen(1^\lambda)$
$(I) \leftarrow Build\_Index(K, \mathcal{D})$
$for\, 1 \leq i \leq M$
$let\, I' = I[0][i] = H_K^{-1}(w_i)$
$T_{w_i} \leftarrow Build\_Trap(K, k_s, w_i, num)$
where $(w_0, w_1, \cdots, w_i) \in P(\mathcal{W})$
$b \xleftarrow{\$} \{0,1\}$
$(st_{\mathcal{A}}, w_0, w_1) \leftarrow \mathcal{A}_0(st_{\mathcal{A}}, 1^\lambda, \mathcal{W}, I', T_{\mathcal{W}})$; where $(w_0, w_1) \in P(\mathcal{W})$
$(T_{w_b}) \leftarrow Build\_Trap(K, k_s, w_b, num)$
$b' \leftarrow \mathcal{A}_1(st_{\mathcal{A}}, I_{w_b})$
$if\, b' = b, output\, 1$
$otherwise\, output\, 0$

where $st_{\mathcal{A}}$ represents a string that captures $\mathcal{A}$'s state. The trapdoor-index indistinguishability holds if for the polynomial-size adversaries $(\mathcal{A}_0, \mathcal{A}_1)$,

$$Pr[Trap\_Index_{RMSE,\mathcal{A}(\lambda)} = 1] \leq \frac{1}{2} + negl(\lambda)$$

where probability is over the choice of b.

5. Proposed RMSE Scheme

In this section we present our RMSE scheme. As mentioned in the Section 2, the proposed RMSE scheme comprises of 5 polynomial-time algorithms. The proposed scheme is an extension of the scheme proposed in [7] that is tuned to facilitate multi-threading. To ease the understanding and the flow of events that take place during the life cycle, we take a modular approach, so Table 1 represents the algorithms that run on the client's side, whereas, Table 2 shows the algorithms running at the Server's side.
A brief description of the client side and the server side tasks are given as follows:

5.1. KeyGen Phase:

The client triggers the KeyGen algorithm. Having the security parameter λ, the client generates two cryptographic keys, *i.e.*, a master key K and a shared key k_s. The master key is kept secret whereas the shared key is shared with the CS.

5.2. Build_Index Phase

This phase is run by the client. The client initializes a prime number P of the order $\lambda + 1$bits. Using the master key, the client computes the hashes of all the keywords in the dictionary as $H_K(\mathcal{W})$ and stores their inverses in the first row of the index table I. The encrypted document identifiers $E_K(id(\mathcal{D}))$ are placed along the first column of the index table I. Using the Equation (1), the client calculates the relevance frequencies that represent the frequency of the occurrence of a keyword within a document and the entire dataset thereafter. The RFs are calculated and placed at the respective location within the index table I. In order to enhance the security of the index table and to prevent statistic analysis attacks the RF's are masked in such a way that the correlation between the RF's remains same but deters the possibility of an attack. Upon the successful generation of the index table I, the secure index table is outsourced to the CS.

5.3. Build_Trap Phase

In order to search for a keyword, the client runs the Build_Trap phase. This phase requires a probabilistic encryption algorithm such as AES-CBC to produce probabilistic trapdoors. The algorithm also makes use of the Hash function used in the Build_Index phase by taking the master key and the keyword as the input. It is to be noted that the inverse is already present in the index table. So even though the trapdoor is probabilistic but it can easily be mapped to the entry where $a * b * a^{-1} = b$. Now the server only needs to compute the Hash using the shared key k_s. The Build_Trap algorithm is designed in such a way that it can facilitate single keyword search or multi-keyword searching.The trapdoor is generated and sent to the CS.

Table 1: Proposed RMSE Scheme-Client Side

1) **KeyGen Phase:**

Given a security parameter λ, the Client generates the cryptographic keys $K, k_s \leftarrow \{0,1\}^{\lambda}$; where K, k_s is the master key and session key respectively.

2) **Build_Index Phase:**

Given a set of documents $\mathcal{D}$, dictionary of keywords $\mathcal{W}$, a master key K, Hash functions $H(\cdot)$, prime number p of the size $\lambda+1$ bits, random number $R \leftarrow CSPRNG(1^{\lambda})$, the index table (I) is generated by the client and sent to the Server. The index table has the dimensions $(N+1) \times (M+1)$, where, M represents the total number of keywords and N represents the total number of documents. The index table is generated as follows:

- for $1 \leq t \leq M$:
  - let $a \leftarrow (H_K(W_t))$
  - Compute $I[1][t] = a^{-1}$;
- for $1 \leq u \leq N$:
  - Compute $I[u][1] = E_K(id(D_u))$;
  - Calculate the RF for W_t occurring in D_u using Equation (1) and store as $I[u][t]$;
- $Mask(RF)$:
  - for $1 \leq m \leq M$:
    - for $1 \leq n \leq N$:

      Choose R in Z_p;

      $I[n+1][m+1] = I[n+1][m+1] * R$;

3) **Build_Trap Phase:**

Given the master key (K), the session key (k_s), a Hash function $H(\cdot)$, desired number of documents (num), the trapdoor T_w is generated by the client as follows:

- let $b \leftarrow (Enc_K(w))$.
- for $1 \leq u \leq w$:
  - let $a \leftarrow (H_K(w))$;
  - let $c \leftarrow a * b$;
  - $B[u] = c$;
- let $d \leftarrow H_{k_s}(b)$;
- $T_W \leftarrow (d, B, num)$;

4) **Dec Phase:**

Given the master key (K) and a set X of encrypted document identifiers stored in ranked order, the decryption is achieved using the session key (k_s).

Table 2: Proposed RMSE Scheme-Server Side

1) **Search_Outcome Phase:**

Given a trapdoor T_w transmitted by the client, a session key k_s, a Hash functions $H(\cdot)$ (same as Build_Trap phase) and the index table I, the search is done by the Server and the ranked encrypted documents are returned to the client. The search is done as follows:

- Split trapdoor into $q = w$ parts, assign to an individual thread.
- For each thread do the following:
  - for $1 \leq l \leq sizeofI$:
    - if $(d == H_{k_s}(B[q] * a^{-1}))$:
    - $Y[q] = l$
    - Sort the RFs in descending order.
  - $X_w[\,] \leftarrow Enc_K(id(D_i))$
- Add all the X_w into another array Y.
- Input the arrays Y into the MapReduce framework to obtain the documents containing the keywords.

5.4. Search_Outcome Phase

This algorithm is run by the CS. Depending upon the number of keywords for which the trapdoor T_w is generated, the CS splits the trapdoor into i parts among different threads. Since the search can be performed in parallel, the encrypted document identifiers are identified for each thread. Then the results from the individual thread are fed as the input to another thread that reduces to identify encrypted document identifiers in ranked order. The results are shared with the client.

5.5. Dec Phase

The client using his master key K decrypts the results to uncover the underlying document identifiers.

6. Security Analysis

This section presents the theorems and lemmas to validate the security claims of the scheme. It is important to analyze the leakage profile of the proposed RMSE before validating the security of the scheme.

6.1. Leakage Profile

In order to validate the security of a scheme it is important to analyze the leakage profile of the proposed RMSE scheme. This helps to examine the effects of the scheme on the security definitions by demonstrating whether the scheme is in line with the security definitions proposed in the Section 4. This analysis includes all the artifacts that evolve during the lifetime of the system and we analyze them individually. The leakages L_1, L_2, L_3 are associated with the Index Table (I), Trapdoor (T_w) and the Search Outcome (SO) respectively. The leakages are explained below:

- *Leakage* L_1: This leakage is linked with the Index Table I and highlights the information revealed by the index table. The index table is generated by the data owner (client) and outsourced to the CS. This leakage is defined as:

$$L_1(I) = \{(H_K(W_M))^{-1}, Enc_K(id(D_N), Mask(RF), (H_K(W_M))\}$$

- *Leakage* L_2: This leakage is related to the trapdoor generated for a set of keywords (w) and represented by T_w. This leakage is defined as follows:

$$L_2(T_w) = \{(i, B[i] \leftarrow Enc_K(W_i) * (H_K(W_i))), (d \leftarrow H_{k_s}(W_i)), num\}$$

- *Leakage* L_3: This leakage incurs due to the outcome of the search when the CS searches for the disjunctive keywords (w_i) against which trapdoor is formed. The leakage is defined as follows:

$$L_3(SO) = \{OC(w), Enc_K(id(D_i))_{\forall T_{W_i} \in D_i}\}$$

where OC represents the outcome of the search against the keyword set.

6.2. Security Validation

In order to validate the security of the proposed RMSE scheme, we present a theorem that confirms whether the proposed scheme provides Keyword-Trapdoor and Trapdoor-Index indistinguishability. This leads to a lemma that helps validate the conformance of the proposed RMSE scheme to the privacy preservation property by taking the leakage profile into account.

Theorem 1: *RMSE provides Keyword-Trapdoor and Trapdoor-Index Indistinguishability.*

Proof: We now prove that the scheme presented in the Section 5 provides indistinguishability. The proof is two fold; firstly we prove that the scheme provides Keyword-Trapdoor Indistinguishability and then we prove that it is Trapdoor-Index indistinguishable. We present the game based proofs as follows:

A. Keyword-Trapdoor Indistinguishability

The term Keyword-Trapdoor Indistinguishability states that an adversary should not be able to distinguish between two trapdoors generated for different keywords. Therefore, a unique trapdoor should be generated for the same keywords being searched again. The game proceeds as follows:

Let RMSE be a ranked multi-keyword SE scheme. Suppose an index table (I) is generated over a dictionary of keywords $\mathcal{W} = \{w_1, w_2, \cdots, w_M\}$ extracted from a set of documents $\mathcal{D} = \{D_1, D_2, \cdots, D_N\}$ where $M, N \in \mathbb{N}$. The game is played between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$ and comprises of three phases:

- **Setup:** The adversary $\mathcal{A}$ sends a keyword to the challenger $\mathcal{C}$. The challenger runs Build_Trap algorithm and generates a trapdoor corresponding to the keyword and sends it to the adversary $\mathcal{A}$. This may continue until the adversary has not queried all the possible keywords $\mathcal{W}$ and received the associated trapdoors. Hence now the adversary has formed a dictionary against all the previous search queries.

- **Challenge:** The adversary $\mathcal{A}$ outputs two set of disjunctive keywords w_1', w_2' such that $w_1' \neq w_2'$. The challenger $\mathcal{C}$ tosses a fair coin $\beta \leftarrow \{0,1\}$ and runs Build_Trap phase to generate a trapdoor $T_{w'_\beta}$. The trapdoor $T_{w'_\beta}$ is sent to the adversary $\mathcal{A}$. The adversary $\mathcal{A}$ is allowed to run the setup phase again and query the same keywords again if interested.

- **Guess:** The adversary $\mathcal{A}$ returns a guess $\beta' \in \{0,1\}$ of β

The advantage of the adversary $\mathcal{A}$ in winning the game is defined as:

$$\text{KI}Adv_{\mathcal{A}} = |Pr[\beta' = \beta] - 1/2|$$

B. Trapdoor-Index Indistinguishability

The term Trapdoor-Index Indistinguishability states that an adversary should not be able to distinguish between two index table entries corresponding to the trapdoors prior to the search. Therefore, a unique trapdoor should be generated for the same keywords being searched again in such a way that the corresponding index entries should not be revealed. The game proceeds as follows:

Let RMSE be a ranked multi-keyword SE scheme. Suppose an index table (I) is generated over a dictionary of keywords $\mathcal{W} = \{w_1, w_2, \cdots, w_M\}$ extracted from a set of documents $\mathcal{D} = \{D_1, D_2, \cdots, D_N\}$ where $M, N \in \mathbb{N}$. The game is played between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$. The game proceeds as follows:

- **Setup:** The adversary $\mathcal{A}$ sends a keyword to the challenger $\mathcal{C}$. The challenger runs Build_Trap algorithm and generates a trapdoor corresponding to the keyword. The challenger also identifies the index table entries corresponding to the trapdoor and sends the keywords, trapdoor and index table entries to the adversary $\mathcal{A}$. This may continue until the adversary has not queried all the possible keywords $\mathcal{W}$ and received the associated trapdoors and index table $I_{\mathcal{W}}$ entries. Hence now the adversary has formed a dictionary against all the previous search queries.

- **Challenge:** The adversary $\mathcal{A}$ outputs two keywords w_1', w_2' such that $w_1' \neq w_2'$. The challenger $\mathcal{C}$ tosses a fair coin $\beta \leftarrow \{0,1\}$ and runs Build_Trap phase to generate a trapdoor $T_{w'_\beta}$. The trapdoor $T_{w'_\beta}$ is sent to the adversary $\mathcal{A}$. The adversary $\mathcal{A}$ is allowed to run the setup phase again and query the same keywords again if interested.

- **Guess:** The adversary $\mathcal{A}$ returns a guess index entry $I'_{w'_{\beta'}}$ such that $\beta' \in \{0,1\}$ of β.

The advantage of the adversary $\mathcal{A}$ in winning the game is defined as:

$$\text{TI}Adv_{\mathcal{A}} = |Pr[\beta' = \beta] - 1/2|$$

Hence, the proposed RMSE scheme provides Keyword-Trapdoor and Trapdoor-Index Indistinguishability.

Lemma 1: *The proposed RMSE is a Privacy Preserving SE scheme as it provides* $(L_1, L_2, L_3)-$*security and is according to the definitions 4.1 and 4.2.*

Proof Sketch: Theorem 1 already proves that the proposed scheme conforms to the Definitions 4.1, 4.2 and provides keyword-trapdoor and trapdoor-index indistinguishability. We now have to examine the effects of the leakages onto the definition's conformity. Referring to the leakages (L_1, L_2, L_3) defined above, it is observed that most of the leakages are either encrypted or hashed and do not lead to any privacy or security concerns. We also make use of a masking function that helps to enhance the security and privacy of the scheme. The masking function helps to deter statistical analysis attacks. Our proposed masking function can also be replaced with Order Preserving Encryption (OPE). The proposed trapdoors are also probabilistic that help to prevent the distinguishability attacks if eavesdropping takes place. Therefore, the proposed RMSE is privacy preserving.

7. Performance Analysis

The comparative complexity analysis of the proposed scheme is already performed in [7]. Before proceeding towards discussing the performance of the proposed RMSE scheme, we discuss the storage overhead that gives an insight of the memory resources consumed. Later-on in this section we discuss the British Telecom's Cloud Service (BTCS) architectural details and the computational time respectively.

7.1. Storage Overhead

Storage overhead is an important metrics that helps to analyze the memory acquired by the proposed scheme. The memory consumption is highly dependent upon the underlying data structure. Referring to the Section 5 it is observed that two keys are stored at the client side, *i.e.*, K and k_s. Having the security parameter λ, the keys are of the size 128 bits. Hence the total storage at the client side is $128 * 2/8 = 32$ bytes.

Now we analyze the storage overhead of the Server. It is observed that the Server has to store the session key k_s, the index table I and the encrypted documents $\mathcal{D}$. The session key k_s is similar to the client's, *i.e.*, 128 bits. Given N documents and M keywords, the storage overhead of the index table I is 8(m*n). The storage incurred as a result of the stored encrypted documents is $n * D_{avg}$. Hence the storage required at the CS is $8(m * n) + n * D_{avg}$.

As already discussed in the Section 2, multithreading is dependent upon the number of queried keywords. Each thread has to search over the entire index table I so the space complexity in the worst-case scenario would be $O(mn)$, where, m and n represent the number of keywords and the number of documents respectively.

7.2. British Telecommunications Cloud Service

We now explain the architectural details of the British Telecommunication's Cloud Service (BTCS). This is important because it will effect the deployment of the proposed scheme and the computational time thereafter.

7.2.1. Application Encryption Service

The Application Encryption (AE) service features a set of SDKs and APIs coupled with the Key Management Service (KMS) to perform cryptographic and key management operations securely, efficiently and effectively. This helps to increase the client's trust onto the Cloud while allowing and simplifying the process of including encryption to the applications. This also helps to lower the complexity by making the deployment simpler and reducing the costs associated to the development and implementation of the cryptographic primitives. AE is widely being used by clients to integrate application level confidentiality to the existing infrastructure. Similarly, the BTCS is also reliant on a trusted third party AE service. The AE server is FIPS-140-2 certified [24] that approves the cryptographic modules by scaling the security of the system into four levels depending upon the compliance and robustness of the system.

The AE service used by the BTCS clients is totally independent of the BTCS and does not communicate with the BTCS. Hence all the key-management and cryptographic operations are controlled by the client independently, resulting in complete trust of the client onto the AE service. The KMS component is centralized and can be hosted either on the client's own workstation for complete trust or onto any Cloud platform. The responsibilities of a KMS include the generation, storage and management of cryptographic keys, governance of the access policies and administrator profiles. It also provides libraries that implement encryption/decryption without having any hidden back-doors in place. Figures 3 and 4 show the activity diagrams that illustrates the flow of events during the life cycle of the RMSE scheme when deployed onto the BT Cloud Service.

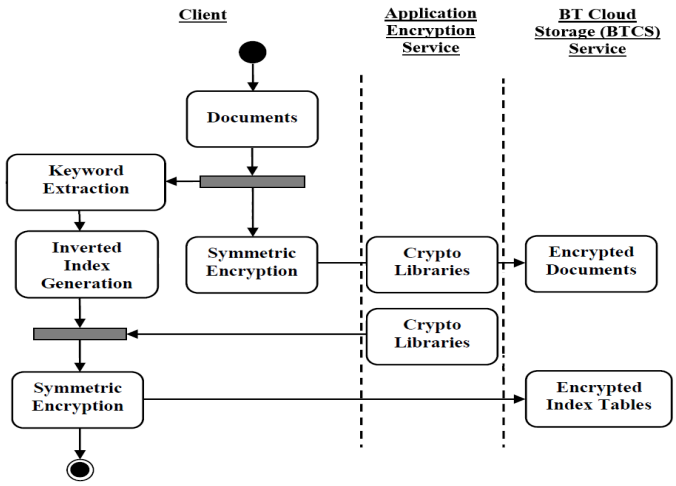

Figure 3: Activity Diagram: Setup

7.3. Dataset Description

The dataset used for evaluating the implementation is the Switchboard-1 Telephone Speech Corpus (LDC 97S62) [25]. The dataset was originally collected by Texas Instruments in 1990-1, under DARPA sponsorship. The Switchboard-1 speech database

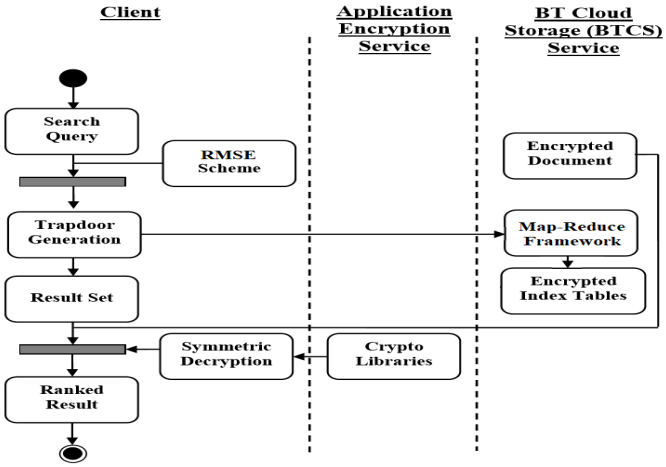

Figure 4: Activity Diagram: Searching

[26] is a corpus of spontaneous conversations which addresses the growing need for large multi-speaker databases of telephone bandwidth speech. The corpus contains 2430 conversations averaging 6 minutes in length; in other words, over 240 hours of recorded speech, and about 3 million words of text, spoken by over 500 speakers of both genders from every major dialect of American English. The dataset comprises of more than 120,000 distinct keywords.

7.4. System Specification

The proposed RMSE scheme is implemented in Java and the results have been generated in Matlab R2016a. The client-server architecture has been implemented on separate machines. The client-side is our workstation and Server-side is the BTCS. The communication takes place through sockets.

- Client side: The confidentiality is achieved by implementing 128-bit AES-CBC and the keyed cryptographic hash function used is SHA-256. The specification of the workstation is 2.7 GHz Intel Core i5 processor, 8GB RAM, running at 1867 MHz DDR3.
- Server side: The searching is performed at the BTCS. The resources allocated at the BTCS include a Dual Core Intel (R) Xeon (R) CPU E5-2660 v3 running at 2.60 GHz and 8GB of RAM.

7.5. Performance Estimation

To analyze the performance of the proposed RMSE scheme in a multi-threaded environment, we analyze each of the phases (already discussed in Section 5) separately. Since the KeyGen phase and the Dec phase are identical to other schemes, we do not evaluate them. The performance estimation for the remaining phases is discussed below:

7.5.1. Build_Index Phase

Figure 5, graphically represents the computational time of the proposed RMSE scheme for the index generation. Although with the addition of ranking to the scheme the query effectiveness increases but the performance also decreases. Therefore, we measure the computational time for the ranked and unranked index generation separately. We start by generating the index for 100 documents and gradually scaling them to 2000 documents while keeping the number of keywords fixed to 120,000. We experience several peak values in the graphs due to the inconsistency among the documents, *i.e.*, due to the different size of documents. The number of documents are represented along the x-axis whereas the time in seconds is along the y-axis. It is observed that although in general our construction shows a linear growth, however, unranked index generation is efficient as compared to the ranked index generation. The ranked scheme for 2000 documents takes 20.3 seconds for the index generation whereas the unranked index generation takes 11 seconds. Hence, in scenarios where ranking is not required the proposed construction modified as unranked can ensure much better results. It is to be noted that the index generation is a one time process and does not need to be generated for every query.

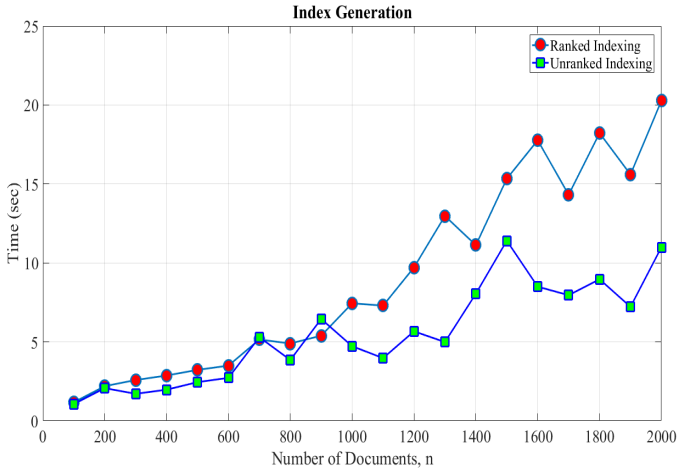

Figure 5: Computational time for the index generation.

7.5.2. Build_Trap Phase

Figure 6, illustrates the computational time that the proposed RMSE scheme requires for trapdoor generation. We start with 1 keyword and scale it to 5 keywords. The time in seconds is along the y-axis, whereas, the number of keywords are along the x-axis. It can be seen that the proposed RMSE scheme on applying graph normalization shows a linear growth with the increase in the number of disjunctive keywords. Hence for generating a trapdoor containing 5 keywords, we require 5.21 seconds.

7.5.3. Search_Outcome Phase

Search_Outcome phase refers to the computational time required for performing disjunctive search and obtaining ranked documents. We delegate the search across multiple

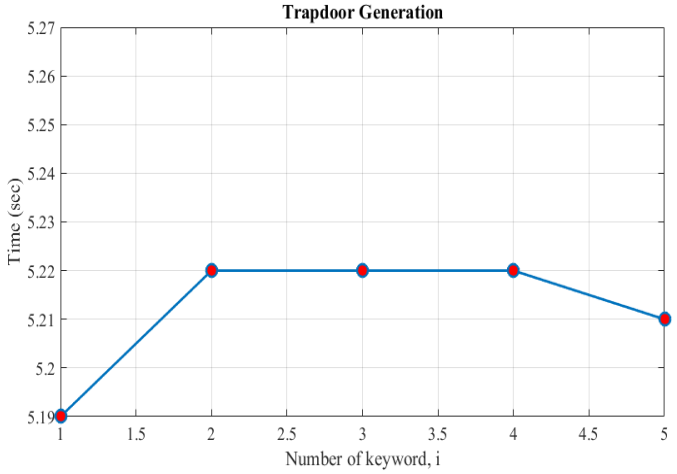

Figure 6: Computational time for trapdoor generation.

threads. The threads search for the individual keywords and reduce the result from all the threads to an outcome. The total number of threads used are equal to the number of keywords in our search query.

Figure 7 shows the computational time while searching for disjunctive keywords. In the graph, ST represents Single-thread and MT represents Multi-threads. Firstly we search for two keywords "about time" using a single thread. Later on we perform search over multiple threads. It can be observed that the efficiency increases by extending the search over multiple threads as compared to the single-threaded search. The increase in the efficiency is due to the parallel and concurrent execution of the search algorithm across multiple threads. Since the designed system supports multi-threading architecture, the computational time of the algorithm can be interpreted as a function of the number of threads. For multiple threads the keywords used are {about, time, and, is, or}. Each thread processes a particular keyword to be searched across the database. The number of documents are represented along the x-axis and the time in seconds is along the y-axis. We observe the peak values across the results because of the varying size of the documents, *i.e.*, with the increase in the size of the documents the number of keywords increases, resulting in an increase in the search space. For 5 keywords the multithreads takes a maximum of 0.04 seconds, whereas, for 2 keywords the single threaded system takes 0.052 seconds. All the searches have been performed for 2000 documents.

It is evident that by using multiple threads the performance increases significantly. We now deploy the RMSE scheme onto the BTCS. To do a comparison, we deploy the RMSE(Multi-threading) scheme and RMSE(Single-threading) scheme. The communication between the client and the BTCS takes place through sockets, therefore, the results also include the network latency incurred during the communication. The network latency also includes the communication with the Application Encryption Service.

Figure 8 shows a comparison among the single-thread RMSE scheme and the multi-thread RMSE scheme. We again validate our claim of the multi-thread RMSE scheme being more efficient then the single-thread RMSE scheme. The number of documents

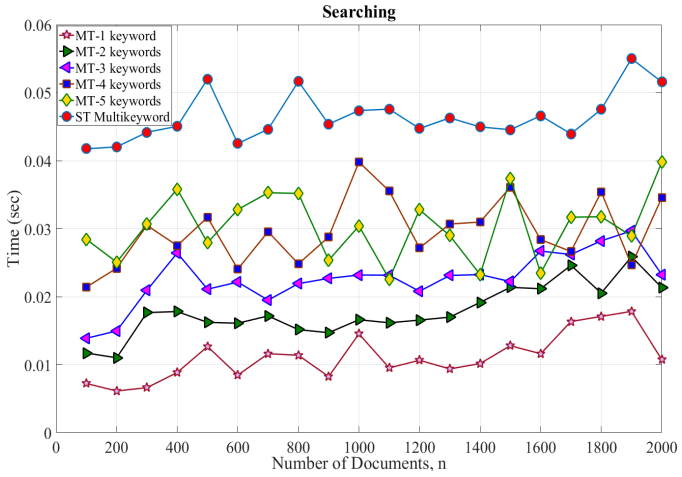

Figure 7: Computational time for multi-threaded search.

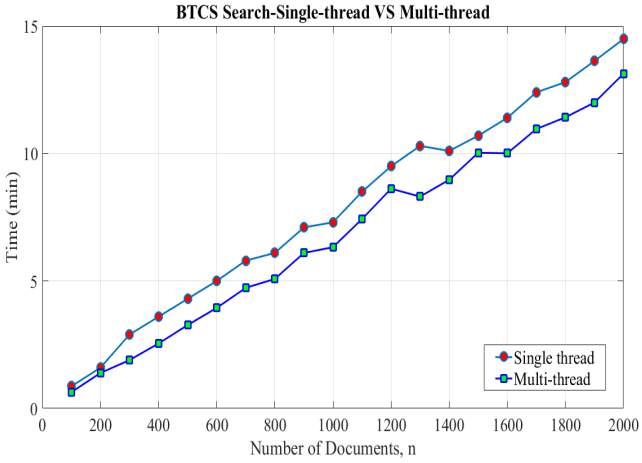

Figure 8: Computational time for search on BTCS with network latency.

are presented along the x-axis and the time in minutes is along the y-axis. For the query "about time" the single-thread search takes 13.49 minutes, whereas, the multi-thread search takes 13.1 minute. With the increase in the number of queried keywords in the multi-threading setting, we believe that we will not see an exponential growth in the computational time of the scheme when deployed onto the BTCS. Therefore, by extending the proposed scheme over multiple threads we are able to enhance the efficiency of our proposed RMSE scheme.

8. Conclusion

Searchable Encryption is a postmodern approach that allows to search for keyword(s) over the encrypted set of documents. With the reliance onto the Cloud, users are outsourcing enormous amount of data to the Cloud. This gives rise to the term "Big data". Prior SE scheme fail to maintain efficiency when extended over the Big Data. This paper aimed to enhance the efficiency of the SE scheme presented in [7]. The scheme was extended to support the map-reduce framework and implemented over multiple threads. The extension may have led to security and privacy concerns, but formal security analysis yields that the scheme still maintains the property of probabilistic trapdoors. The proposed scheme was deployed onto the BT Cloud Server and the performance gain between the primary scheme and the multi-threaded scheme was measured. The multi-threaded scheme outperformed the existing scheme (proposed in [7]) in terms of efficiency.

Acknowledgements

We are thankful to British Telecom Plc. for their support and guidance during the deployment phase. We are also thankful to Intelligent Voice Ltd. for providing the dataset used for extending this work.

References

[1] A. Huth, J. Cebula, The basics of cloud computing, United States Computer Emergency Readiness Team (2011) 1–4.
[2] O. Agmon Ben-Yehuda, M. Ben-Yehuda, A. Schuster, D. Tsafrir, The resource-as-a-service (raas) cloud, Proceedings of the 4th USENIX conference on Hot Topics in Cloud Computing (2012) 12–12.
[3] C. Curino, E. P. Jones, R. A. Popa, N. Malviya, E. Wu, S. Madden, H. Balakrishnan, N. Zeldovich, Relational cloud: A database-as-a-service for the cloud, 5th Biennial Conference on Innovative Data Systems Research (2011) 235–241.
[4] H. Hacigumus, B. Iyer, S. Mehrotra, Providing database as a service (2002) 29–38.
[5] M. Abourezq, A. Idrissi, Database-as-a-service for big data: An overview, International Journal of Advanced Computer Science and Applications (IJACSA) 7 (1) (2016) 157–177.
[6] C. Bösch, P. Hartel, W. Jonker, A. Peter, A survey of provably secure searchable encryption, ACM Computing Surveys (CSUR) 47 (2) (2015) 18.
[7] S. Tahir, M. Rajarajan, S. Ruj, An efficient disjunctive query enabled ranked searchable encryption scheme, Proceedings of the 16th IEEE International Conference on Trust, Security And Privacy in Computing and Communications (2017) 425–432.
[8] S. Tahir, S. Ruj, Y. Rahulamathavan, M. Rajarajan, C. Glackin, A new secure and lightweight searchable encryption scheme over encrypted cloud data, IEEE Transactions on Emerging Topics in Computing 99 (99) (2017) 1–1.
[9] I. H. Witten, A. Moffat, T. C. Bell, Managing gigabytes: compressing and indexing documents and images, Morgan Kaufmann, 1999.
[10] C. Wang, N. Cao, J. Li, K. Ren, W. Lou, Secure ranked keyword search over encrypted cloud data, 30th International IEEE Conference on Distributed Computing Systems (ICDCS) (2010) 253–262.
[11] C. Wang, N. Cao, K. Ren, W. Lou, Enabling secure and efficient ranked keyword search over outsourced cloud data, IEEE Transactions on parallel and distributed systems 23 (8) (2012) 1467–1479.
[12] D. Cohen, E. Amitay, D. Carmel, Lucene and juru at trec 2007: 1-million queries track.
[13] A. Silberschatz, P. B. Galvin, G. Gagne, Operating system concepts essentials, John Wiley & Sons, Inc., 2014.
[14] D. X. Song, D. Wagner, A. Perrig, Practical techniques for searches on encrypted data, IEEE Symposium on Security and Privacy (2000) 44–55.

[15] R. Curtmola, J. Garay, S. Kamara, R. Ostrovsky, Searchable symmetric encryption: improved definitions and efficient constructions, Journal of Computer Security 19 (5) (2011) 895–934.
[16] K. Li, W. Zhang, C. Yang, N. Yu, Security analysis on one-to-many order preserving encryption-based cloud data search, IEEE Transactions on Information Forensics and Security 10 (9) (2015) 1918–1926.
[17] S. Kamara, C. Papamanthou, Parallel and dynamic searchable symmetric encryption, International Conference on Financial Cryptography and Data Security (2013) 258–274.
[18] D. Boneh, B. Waters, Conjunctive, subset, and range queries on encrypted data, Theory of cryptography (2007) 535–554.
[19] R. A. Popa, N. Zeldovich, Multi-key searchable encryption., IACR Cryptology ePrint Archive 2013 (2013) 508.
[20] N. Cao, C. Wang, M. Li, K. Ren, W. Lou, Privacy-preserving multi-keyword ranked search over encrypted cloud data, IEEE Transactions on parallel and distributed systems 25 (1) (2014) 222–233.
[21] H. Li, Y. Yang, T. H. Luan, X. Liang, L. Zhou, X. S. Shen, Enabling fine-grained multi-keyword search supporting classified sub-dictionaries over encrypted cloud data, IEEE Transactions on Dependable and Secure Computing 13 (3) (2016) 312–325.
[22] M. Strizhov, I. Ray, Secure multi-keyword similarity search over encrypted cloud data supporting efficient multi-user setup., Transactions on Data Privacy 9 (2) (2016) 131–159.
[23] F. Han, J. Qin, J. Hu, Secure searches in the cloud: a survey, Future Generation Computer Systems 62 (2016) 66–75.
[24] F. PUB, Security requirements for cryptographic modules, FIPS PUB 140.
[25] J. Godfrey, E. Holliman, Switchboard-1 Release 2 - Linguistic Data Consortium.
URL `https://catalog.ldc.upenn.edu/LDC97S62`
[26] J. Godfrey, E. Holliman, J. McDaniel, SWITCHBOARD: telephone speech corpus for research and development (1992) 517–520doi:10.1109/ICASSP.1992.225858.