Fuzzy Keywords enabled Ranked Searchable Encryption Scheme for a Public Cloud Environment

Shahzaib Tahir, Student Member, IEEE, Sushmita Ruj, Senior Member, IEEE, Ali Sajjad, and Muttukrishnan Rajarajan, Senior Member, IEEE

Abstract—Searchable Encryption allows a user or organization to outsource their encrypted documents to a Cloud-based storage service, while maintaining the ability to perform keyword searches over the encrypted text. However, most of the existing search schemes do not take the almost certain presence of typographical errors in the documents under consideration, when trying to obtain meaningful and accurate results. This paper presents a novel ranked searchable encryption scheme that addresses this issue by supporting fuzzy keywords. The proposed construction is based on probabilistic trapdoors that help resist distinguishability attacks. This paper for the first time proposes Searchable Encryption as a Service (SEaaS). The proposed construction is deployed on the British Telecommunication’s public Cloud architecture and evaluated over a real-life speech corpus. Our security analysis yields that the construction satisfies strong security guarantees and is also quiet lightweight, by analyzing its performance over the speech corpus.

Index Terms—Searchable Encryption as a Service (SEaaS), Privacy by Design, Probabilistic Trapdoors, Inverted Index, Indistinguishability, Min hashing, Euclidean Norm, Relevance Frequency, Jaccard Similarity.

I. INTRODUCTION

Cloud is an environment that enables on-demand, ubiquitous resource sharing and data access to the Clients efficiently and effectively while minimizing the up-front infrastructure costs. Apart from the monumental advantages and benefits of relying upon the Cloud, it also poses a serious threat to the privacy and security of the Client and the data that is outsourced. One solution is to encrypt the documents prior to outsourcing them to the Cloud that may lead to huge network latency incurred due to the downloading of all the documents and decrypting them to search for a particular keyword. Therefore, a Searchable Encryption (SE) scheme is required that delegates the search to the Cloud in such a way that the confidentiality of the outsourced documents remains intact, while equally preserving the privacy of the keywords searched. This also helps to thwart effective eavesdropping of data transmitted between the Client and the Cloud Server.

Attempts to design Searchable Encryption schemes for securely storing data on the Cloud while enabling keyword search have generally considered three factors: 1) Security and privacy; 2) Efficiency; and 3) Query Effectiveness. Although all these factors are of equal importance but most of the existing schemes are unable to maintain a balance between them. Hence, the schemes lack in usability and can not be deployed onto a real Cloud infrastructure.

The aim of the proposed SE scheme is to preserve the privacy of the client’s data and the tasks that he performs on his data. This requires to have privacy preservation inherently and not as a pluggable feature. In [1], Cavoukian discusses the fundamental principles to achieve privacy by design. The aim of this research is to design a SE scheme that assures privacy preservation as a default feature.

The significance of SE is apparent from its vast areas of application where a lot of work is under progress. In [2], SE is studied in conjunction with e-Healthcare systems over the Cloud. SE is also being explored in relation to the IoT devices and smart meters as proposed in [3]. SE can also have a profound impact on secure transactions when coupled with blockchain technology, as discussed in [4]. With the emergence of Homomorphic Encryption, the use of Searchable Encryption is also being explored in genome analysis to securely analyse and search human DNA sequences [5].

In our fuzzy searching process, the Cloud service takes human typographical errors into account while performing the search, such that the search query does not have to exactly match the desired information. Typically, a single-keyword or a multi-keyword SE scheme allows only the exact matching of the keyword(s) and in-case a keyword is wrongly spelt, a null set is returned as the result. On the contrary, a fuzzy keyword enabled SE scheme uses a similarity based search mechanism that performs the search based on the edit distance to identify a possible set of keywords that could be desired by the Client. Until now a few fuzzy keywords enabled searchable encryption schemes have been proposed but they are not practical enough to be deployed on the Cloud. Some of the fuzzy schemes do not facilitate ranking and/or are based on deterministic trapdoors e.g. [6] [7] [8] [9] [10]. Therefore, the aim of this research is to explore the problem of fuzzy searching and propose novel, practical and efficient solutions to perform fuzzy Searchable Encryption while preserving the privacy. Furthermore, the proposed scheme should be based on probabilistic trapdoors and allow ranked searching.
A. Contributions

The following contributions to the field of SE are being made through this research:

- We present a novel fuzzy keywords enabled ranked searchable encryption scheme. The scheme is privacy preserving by design as it is based on probabilistic trapdoors that helps resist distinguishability attacks and preserve the privacy of the outsourced documents and search queries.
- We introduce Searchable Encryption as a Service (SEaaS). We design and develop a proof of concept prototype and test it over a encrypted real-world corpus of speech transcriptions outsourced to the British Telecom’s public Cloud storage service.

B. Organization

Section II discusses the proposed architecture with the help of a scenario that helps to formally present the design goals. Section III discusses the major relevant existing schemes by concisely highlighting their pros and cons. In Section IV, the preliminaries associated to the Fuzzy Ranked Searchable Encryption scheme (FRSE) are discussed that helps to conceptualize the proposed scheme. Section V revisits the existing security definitions by tuning them according to the proposed architecture. Finally, in Section VI the FRSE scheme is proposed. The Security analysis is performed in Section VII and the scheme is validated. The Section VIII presents an algorithmic comparative analysis against existing schemes. The scheme is deployed on the BT Cloud service and its performance and storage overhead is analyzed in Section IX. The details related to British Telecom’s public Cloud infrastructure are also presented in Section IX. The conclusions are drawn towards the end of the paper in Section X, followed by the acknowledgments.

II. Fuzzy Keywords enabled Ranked Searchable Encryption Scheme Architecture

In this section we explain the architectural details involved in the proposed scheme. This discussion leads to the identification of the design goals.

A. Proposed Architecture

In this section we discuss the proposed FRSE architecture, using the Cloud Server (CS) as an honest-but-curious component of the system. We explain this with the help of the following scenario:

Bob outsources his encrypted documents \( D = \{D_1, D_2, \cdots, D_n\} \) to the CS. He wants to perform fuzzy search over the encrypted set of documents. Typically, a searchable encryption algorithm does not take the user’s typographical errors into account. Therefore, for the wrongly spelt keyword (e.g., “aboot” instead of “about”), the search will return null. On the contrary, a fuzzy searchable encryption algorithm will still successfully infer a meaningful keyword from the misspelt keyword. So the search result will include the documents containing the keywords “about”, “abort” etc.

To perform efficient search, Bob will extract a dictionary of keywords \( W = \{W_1, W_2, \cdots, W_m\} \) from \( D \) and form an inverted index table and fuzzy index table respectively. The inverted index table makes use of relevance score generator for the ranking of the documents whereas the fuzzy table enables the fuzzy matching of the keywords. For the secure index generation, the cryptographic keys are generated. Upon the successful generation of the index tables \( (FI, I) \), Bob outsources the index tables along with the encrypted set of documents to the CS. This is a one time process.

As mentioned above, we assume that the CS is honest-but-curious. Here honesty means that the CS performs all its operations properly and correctly, but it is also willing and curious to learn any information that it can infer from the outsourced documents it is storing on behalf of the client, or from the search queries it receives from the client. In future, to search for a keyword over the encrypted set of documents, Bob using his private key will generate a meaningful probabilistic trapdoor and send it to the CS. The probabilistic trapdoor is unique for the same keyword being searched repeatedly, hence, it resists distinguishability attacks. Now the CS using the trapdoor performs the search over the index tables and returns the results i.e. the encrypted document identifiers in a ranked order.

Figure 1 illustrates the flow of events that take place during the entire life cycle of the search.

![Fig. 1. The System Architecture](image)

B. Design Goals

The proposed Fuzzy Ranked Searchable Encryption Scheme (FRSE) has the following design objectives:

- **Trapdoor Unlinkability**: The trapdoors should resist distinguishability attacks i.e. for the same keyword being searched again a new, unique and totally randomized trapdoor should be generated.
- **Ranked Searching**: The proposed scheme should facilitate ranked searching by presenting the search results based on some ranking functionality.
- **Modularized Fuzzy Component**: The fuzzy component should be designed in such a way that it can be
inter-orable with an existing SE schemes (single-keyword/multi-keyword) with minimal modifications.

- Cloud Compatibility: The designed scheme should be compatible and easy to integrate with an existing Public Cloud offering.
- Privacy-by-design: The designed scheme should be privacy preserving by default.

Definition (Fuzzy Ranked Searchable Encryption Scheme (FRSE)): The proposed FRSE comprises of five polynomial time algorithms \( \Pi = (\text{KeyGen}, \text{Build_Index}, \text{Build_Trap}, \text{Search_Outcome}, \text{Dec}) \) such that:

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

\[(I, F I) \leftarrow \text{Build_Index}(S', D): \text{is a probabilistic algorithm that takes a randomly permuted shingle vector } S' \text{ and collection of documents } D \text{ as the input. The algorithm returns an inverted index table } I \text{ and a fuzzy index table } F I. \text{ This algorithm is run by the client.}
\]

\[T_W \leftarrow \text{Build_Trap}(S', k_s, W, \text{num}): \text{is a probabilistic algorithm that takes a randomly permuted shingle vector } S', \text{ session key } k_s, \text{ keyword(s) } W, \text{ number (num) of required documents as the input. The algorithm returns a trapdoor, } T_W. \text{ The algorithm is run by the client.}
\]

\[X \leftarrow \text{Search_Outcome}(k_s, I, F I, T_W): \text{is a deterministic algorithm run by the CS. The algorithm takes the session key } (k_s), \text{ index table } (I), \text{ fuzzy index table } (F I), \text{ the trapdoor } (T_W) \text{ as the input and returns } (X), \text{ a set of required encrypted document identifiers } Enc_K(id(D_i)) \text{ in ranked order.}
\]

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

III. RELATED WORK

Focusing mainly on the query effectiveness, the literature can be summarized into two main categories as follows:

A. Single Keyword and Multi-KeyWord Searchable Encryption

The problem of SE dates back to the first scheme introduced by Song et al. in [11]. The proposed scheme did not require a secure index. Curtmola et al. in [12] introduced formal security definitions of SE and presented an index-based single keyword SE scheme. The scheme was based on deterministic trapdoors. In [13] authors for the first time shed light on the problem of ranked search and introduce a ranked single keyword SE scheme. However, in [14] authors are able to launch a successful differential attack on the scheme. In [15] authors introduced an index-based ranked SE scheme based on probabilistic trapdoors and presented improved security definitions for SE. These schemes only facilitated single keyword searching.

Until now several SE schemes have been proposed that support conjunctive queries in the sense that interrelated keyword queries can be conducted such as [16] [17]. Sun et al. in [18] introduced cosine similarity measurement for supporting privacy-preserving multi-keyword SE. In [19], authors for the first time introduced coordinate matching for ranked multi-keyword searching. Authors for the first time design a SE scheme that enables geometric range search on the spatial data in [20]. It is also observed that the scheme is generalized and allows different type of shape queries. Authors in [21] introduce a ranked SE scheme that facilitates disjunctive queries. All these aforementioned schemes provide multi-keyword SE but do not support fuzzy SE.

B. Fuzzy Searchable Encryption

In [22], authors for the first time introduce a privacy preserving construction that enables fuzzy keyword searching. Although the proposed scheme takes the typographical errors into account and performs fuzzy search, it requires to construct a fuzzy keyword set and takes all the possible erroneous keywords into consideration. Therefore, for the keyword CASTLE the substitution operation on the first character of the keyword produces the following set \{AASTLE, BASTLE, DASTLE, \cdots, YASTLE, ZASTLE\}. This may not be feasible for large datasets containing hundred thousand keywords because the size of the index may grow exponentially. Since this task is performed by the data owner having low memory and computational resources, it may lead to the entire memory consumption and wastage of resources.

Wang et al. in [8] for the first time introduced the concept of utilizing Locality Sensitive Hashing (LSH) into Searchable Encryption. Their scheme definitely brought a new perspective to design fuzzy SE schemes because a predefined dictionary spanning over the entire possible set of keywords (correct and erroneous) as discussed in [22] was not required. The authors presented two schemes that were able to resist attacks in the known ciphertext model and the known background model respectively. The schemes required per document bloom filter or vector to perform the fuzzy search which is not feasible in real deployment as it may consume too much storage on the cloud. Furthermore, due to the use of independent bloom filters the ranking of the documents cannot be achieved.

In [23] authors for the first time introduce uni-gram vectors for the fuzzy keyword search. They claim to have improved the accuracy as compared to the scheme proposed in [8]. They also propose a stemming algorithm that can query the keywords with the same root and the ranking of the results is based on it. Although their scheme is novel and enhances the query effectiveness, the scheme does not resist distinguishability attacks as the trapdoors are not indeterministic. Therefore, it cannot be termed as the ultimate fuzzy searchable encryption scheme.

Further details about the SE schemes can be found on [24], [25].

IV. FUZZY RANKED SEARCHABLE ENCRYPTION SCHEME

A. Preliminaries

The design of the proposed scheme is primarily based on several important design primitives i.e. Relevance score, Shingling, Min Hashing, Jaccard Similarity and Euclidean Norm. A brief introduction is given below:
1) Relevance Frequency: The relevance frequency (RF) helps to rank the documents. The RF formula [26] presented here is widely accepted and already used in SE [7] [13] [27]. So, given a keyword $W$, and a document $D$, the relevance frequency ($RF$), is calculated as:

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

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

2) Shingled Keywords: Given a set $W$ of keywords $\{W_1, W_2, \ldots, W_m\}$. Let $l$ be a constant that represents the sequence of $l$ characters to appear within the keywords $W$. The choice of $l$ is made such that the probability of any shingle appearing within a keyword should be low. The keyword is firstly transformed into a shingle set $S$ consisting of contiguous $l$ letters appeared in the keyword. The shingle set is converted to a vector $S$ that represents the presence or absence of a particular shingle. Hence for each shingled keyword, a vector $S$ of length $26l$-bit is required i.e., the entries of the vector $S$ represent the occurrence of shingles within a keyword.

3) Min Hashing: Given $q$ number of random hash functions ($i.e.$ random permutations), represented as $f_q : S \rightarrow R$ reduces the shingle vector $S$ and assigns a real number $R$ to form a signature vector ($SV$). Let $S_a$ and $S_b$ represent two shingle vectors for two different keywords. The random hash function permutations should satisfy the condition: $f_q(S_a) \neq f_q(S_b)$. Hence the permutations are independent. The min hash value of any vector is the number of first bit, in the permuted order, in which the vector has a 1.

4) Min hashing and Jaccard similarity: With reference to [28], Min hashing and Jaccard similarity ($JS$) are closely related as follows:

- Given two shingled vectors $S_a$ and $S_b$. The probability of $JS(f_q(S_a), f_q(S_b))$ is equal to the $JS(S_a, S_b)$.

The Jaccard similarity between two sets $X$ and $Y$ is calculated by:

$$JS(X, Y) = \frac{|X \cap Y|}{|X \cup Y|}$$

**Corollary 1:** The Jaccard Similarity between two sets $X$ and $Y$ is 0 if and only if $X \cap Y = \emptyset$

5) Euclidean Norm: Given two vectors $A = [a_1, a_2, \ldots, a_l]$ and $B = [b_1, b_2, \ldots, b_i]$, the Euclidean Norm $d$ represents the distance between the vectors $(A, B)$. The distance between the two $i$-dimensional vectors $A$ and $B$ is represented as:

$$d(A, B) = \sqrt{\sum_{i=1}^{n} (a_i - b_i)^2}$$

**B. Scheme Conceptualization**

Prior to presenting the proposed scheme, this section introduces the crux that help to conceptualize the scheme. This section is explained keeping in view the Public Cloud infrastructure. Before going into the explanation a toy example is presented in Figure 2 to help illustrate the significant SE phases.

![Fig. 2. Toy Example: i) Transform a keyword into a shingle vector. ii) Permutate a random vector from the shingle vector. iii) Form a Fuzzy Index table comprising of the signature vectors after applying $Q$ permutations for the Min Hashing. iv) Calculate the relevance frequencies and form an inverted index by masking the relevance scores. v) To generate a trapdoor, form a shingle vector and a randomized vector. Using the $Q$ permutations calculate the Min hashes and compute the Euclidean Norm between the two vectors. vi) Compute the Jaccard Similarity between the Fuzzy index entries and the trapdoor. vii) Obtain documents in ranked order.](image-url)
Algorithm (PermS): Let $P$ denote a set of $n$ possible permutations of the vector $S$ represented by $S_n$. Initialize a vector $S'$. A random permutation $V : S \rightarrow S'$ such that $S' \in S_n$ is generated as:

1. Let $S = [e_0, e_1, \cdots, e_{n-1}]$
2. For $k = 0, 1, \cdots, n-1$, do:
   - Select at random an element $e$ from $S$ such that $e \not\in S'$.
   - Set $S'[k] = e$.

Therefore, the first element $e$ is picked at random from all the $n$ elements contained in $S$, the second element is picked from $n-1$ elements and so on. There are $n!$ different ways to form a randomly permuted vector $S'$.

The advantage behind randomizing the vector $S$ is that the keywords cannot be guessed in polynomial time. This is highlighted in more detail in the security proofs presented in the Section VII. After the selection of a particular instance of the random permutation, the entire dictionary of the keywords is represented in the form of individual randomly permuted shingled vectors. As the vectors are mostly sparse, the technique of min hashing is applied to all the vectors so that the search is based on the signatures vectors in the Fuzzy Index $FI$.

3) Trapdoor Generation: Let $W$ denote the keyword to be searched for. Given a vector $T$, the keyword is shingled to populate the randomly permuted vector $S'$ accordingly such that $T = S'$. Since $T$ is mostly sparse, calculating the Jaccard Similarity $JS$ and Euclidean Norm of $T$ against all the signature vectors in the Fuzzy Index $FI$ is a resource intensive task. Therefore, min hashing is applied to $T$. Until now, the trapdoor was deterministic. To make the trapdoor probabilistic a random vector $T'$ is selected, the selection is done as follows:

Algorithm (ProbT): Let $T = [t_1, t_2, \cdots, t_n]$ denote a vector containing $n$ min hashed values associated to a keyword $W$. Initialize a vector $T'$. A random permutation $Q : T \rightarrow T'$ is generated as:

For $j = 1, 2, \cdots, n$, do:
- Let $U$ be a uniform random variable in the range $[1, 2^b]$ and $U \notin T, T'$.
- Set $T'[j] = U$.

Therefore, the vectors do not have any elements in common.

Upon the successful generation of random vector $T'$ the Euclidean Norm $dis = d(T, T')$ is computed. The trapdoor is $Enc_{k_s}(dis, T')$. The trapdoor is transmitted to the Cloud Server (CS). The next step is to perform search and identify the documents that contain the queried keyword.

4) Searching: Upon receiving the trapdoor, the Cloud Server using the session key $k_s$ decrypts the trapdoor to uncover the underlying content. The search is based on the corollary 1. Since $JS$ is applicable to sets only, so the CS calculates the $JS(T', SV_W)$ that requires the representation of vectors into sets. The conversion is done as follows:

Algorithm (ExtendedVector): Let $T = [t_1, t_2, \cdots, t_n]$ denote a vector containing $n$ entries. Initialize an empty set $L$. The ExtendedVector $E : T \rightarrow L$ is generated as:

For $j = 1, 2, \cdots, n$ do:
- Add $t_j$ to $L$ such that $t_j \notin L$.

The trapdoor is formed in such a way that the CS looks for the entries where the $JS$ is 0. On identifying the relevant entries, the CS computes the Euclidean Norm $d(T', SV_W)$, where $SV_W \in FI$ and $T' \in$ Trapdoor. If the searched keyword is contained in the dictionary, calculated Euclidean Norm will be equal to $dis$. Otherwise, if the keyword is not contained in the dictionary, the most relevant keyword will be having a Euclidean Norm $\approx dis$ varying by $\epsilon$. $\epsilon$ is the variance from the existing keywords and represents the threshold that can be controlled by the Client.

Upon the identification of the corresponding masked keyword identifier $mask(id(W))$, the CS refers to the Inverted Index Table, $I$. After identifying the relevant column, the CS returns ranked encrypted document identifiers to the client.

Note: The sole purpose of using session key $k_s$ between the Client and the CS is to avoid any passive attacks that may be carried out by an outsider. Session key may not be required if the channel is secure.

V. Security Definitions

In this section we revisit the existing definitions proposed in [15] that are related to probabilistic trapdoors. We extend the definitions to fit our construction.

A. Keyword-Trapdoor Indistinguishability for FRSE

Keyword-Trapdoor indistinguishability allows the adversary to select a keyword adaptively based on the history. More precisely, the adversary is given tuples $(W, T_W)$. Since the adversary now has all the possible keywords and associated trapdoors, it has to now submit two distinct keywords $(w_i, w_j)$. The challenger tosses a fair coin $b$ and sends the trapdoor corresponding to the keyword $w_b$ to the adversary. This process continues until the adversary has submitted polynomially-many queries and is then challenged to output the bit $b$.

Definition 5.1 Let $FRSE = \langle KeyGen, Build_Index, Build_Trap, Search_Outcome, Dec \rangle$ be a fuzzy-based ranked searchable encryption scheme over a dictionary $\Delta$, $\lambda$ be the security parameter, and $A_{i:0 \leq i \leq N+1}$ be a non-uniform adversary. Consider the following probabilistic experiment $Key_{TrapFRSE,A}(\lambda)$:

Key_{TrapFRSE,A}(\lambda)\left(\right):

$\left(K, k_s, r\right) \leftarrow KeyGen(\lambda)$

$\left(1, FI\right) \leftarrow Build_Index(S', D)$

for $1 \leq i \leq N$\:

$(st_A, w_i) \leftarrow A_1(st_A, T_{w_1}, \cdots, T_{w_i})$

$T_{w_i} \leftarrow Build_Trap_{k_s}(w_i)$

$(st_A, w_0, w_i) \leftarrow A_0(\lambda)$

$b \leftarrow \{0, 1\}$

$(T_{w_0}^b) \leftarrow Build_Trap(S', k_s, w_b, num)$

$b' \leftarrow A_{N+1}(st_A, T_{w_b})$
The keyword-trapdoor indistinguishability holds for all the polynomial-size adversaries \((A_0, A_1, \ldots, A_{N+1})\) such that \(N = \text{poly}(\lambda)\),

\[
Pr[\text{Key}_{\text{Trap}}_{\text{FRSE}}(A(\lambda)) = 1] \leq \frac{1}{2} + \text{negl}(\lambda)
\]

where probability is over the choice of \(b\).

B. Trapdoor-Index Indistinguishability for FRSE

Trapdoor-Index indistinguishability refers to the complexity offered by a fuzzy ranked searchable encryption scheme. More precisely, the adversary is given tuples \((W, T_W)\), \(FI\). Since the adversary now has all the possible keywords, associated trapdoors and fuzzy index table, he submits two distinct words \(w_0, w_1\). The challenger now tosses a fair coin \(b\) and submits the trapdoor, fuzzy index table entries corresponding to the keyword \(w_b\) to the adversary. This process continues until the adversary has submitted polynomially-many queries and is then challenged to output the bit \(b\).

**Definition 5.2** Let FRSE\(= (\text{KeyGen}, \text{Build}_{\text{Index}}, \text{Build}_{\text{Trap}}, \text{Search}_{\text{Outcome}}, \text{Dec})\) be a fuzzy ranked searchable encryption scheme over a dictionary \(\Delta\), \(\lambda\) be the security parameter, and \(A = (A_0, A_1)\) be non-uniform adversaries. Consider the following probabilistic experiment \(\text{Trap}_{\text{Index}}_{\text{FRSE},A}(\lambda)\):

\[
\text{Trap}_{\text{Index}}_{\text{FRSE},A}(\lambda) = \begin{cases} 
(K, k_s) \leftarrow \text{KeyGen}(\lambda) \\
(I, FI) \leftarrow \text{Build}_{\text{Index}}(S', D) \\
\text{for } 1 \leq i \leq n \\
\text{let } I' = FI[0][i] = \text{Enc}_K(W_i) \\
T_{w_i} \leftarrow \text{Build}_{\text{Trap}}_{k_s}(w_i) \\
b \leftarrow \{0, 1\} \\
(st_A, w_0, w_1) \leftarrow A_0(\lambda^4) \\
(T_{w_b}) \leftarrow \text{Build}_{\text{Trap}}(S', k_s, w_b, \text{num}) \\
b' \leftarrow A_1(st_A, FI_{w_b}) \\
\text{if } b' = b, \text{output } 1 \\
\text{otherwise output } 0
\end{cases}
\]

where \(st_A\) represents a string that captures \(A\)'s state. The trapdoor-index indistinguishability holds if for the polynomial-size adversaries \((A_0, A_1)\),

\[
Pr[\text{Trap}_{\text{Index}}_{\text{FRSE},A}(\lambda) = 1] \leq \frac{1}{2} + \text{negl}(\lambda)
\]

where probability is over the choice of \(b\).

In SE Privacy Preservation refers to getting valid search results without letting the server learn the underlying data. This also includes anything that can be inferred from the trapdoors, index tables and the encrypted documents outsourced to the Cloud. This leads to the following corollary:

**Theorem 1:** The proposed FRSE scheme provides Keyword-Trapdoor Indistinguishability and Trapdoor-Index Indistinguishability if the Inverted Index table \(I\) and Fuzzy index table \(FI\) are secure and the trapdoors are probabilistic.

VI. PROPOSED FRSE SCHEME

This section formally presents the detailed description of the proposed scheme. As mentioned earlier, the proposed scheme comprises of the following five polynomial-time algorithms:

- **Phase 1:** \(\text{KeyGen}(\lambda)\): Given a security parameter \(\lambda\), generate the keys \(K, k_s \leftarrow \{0, 1\}^\lambda\) and a random number \(r \leftarrow \text{CSPRNG}(\lambda^4)\).

- **Phase 2:** \(\text{Build}_{\text{Index}}(S', D)\):
  - Inverted Index\((I)\) Generation: Construct a matrix \((I)\) of the order \((n + 1) \times (m + 1)\) as follows:
    1. Extract a keyword set \(W = \{W_1, W_2, \ldots, W_m\}\) from a set of documents \(D = \{D_1, D_2, \ldots, D_n\}\).
    2. Set entries at \((i + 1, 1) = \text{Enc}_K(id(D))\); 1 \(\leq i \leq n\).
    3. Set entries at \((1, j + 1) = \text{Enc}_K(W_j)\); 1 \(\leq j \leq m\).
    4. Calculate the RF using equation 1 and populate the remaining entries \((i + 1, j + 1) = R(F(W_j, D_i))\).
    5. Mask the RFs as:
      - For \(a = 1, 2, \ldots, m\)
      - For \(b = 1, 2, \ldots, n\)
      - \((a, b) = (a, b) \times r\)
    6. Output the matrix \((I)\) that represents the inverted index table \(I\).
  - Fuzzy Index\((FI)\) Generation: Construct a matrix \(FI\) of the order \((Q + 1) \times (m + 1)\) where \(Q \leq q\) represents the random permutations used for min hashing represented as \(f_q : S \rightarrow R\) and \(m\) are the total number keywords. Construct \(FI\) as follows:
    1. Form a shingle vector \(S\) of the order \(26^2\) and randomize it to form \(S'\) using the algorithm \(\text{PermS}\) (presented previously in the Section IV).
    2. For each keyword \(W_m\) form a shingle set and permute according to \(S'\).
    3. For each \(S'\) compute the min hashes and form the corresponding Signature Vectors \(SV\) having \(Q\)-bit length.
    4. Set entries of \(FI\) at \((1, i) = \text{Enc}_K(W_i); 1 \leq i \leq m\).
    5. Set entries of \(FI\) at \((i, j) = SV[Q]; 1 \leq j \leq q\).

- **Phase 3:** \(\text{Build}_{\text{Trap}}(S', k_s, W, \text{num})\): Generate a probabilistic trapdoor vector \(T\) as follows:
  1. Represent the search keyword\((s)\) in the form of a shingle set\((s)\).
  2. Permute the same random vector \(S'\) according to the shingle set.
  3. Using the same \(Q\) permutations form a signature vector\((s)\) represented as \(T\).
4) According to the algorithm (ProbT) (presented previously in the Section IV) randomize the vector T to obtain T′.
5) Compute the Euclidean Norm d(T, T′)
The Trapdoor Tw = (d(T, T′), num); where num represents the total number of required documents. Compute EncK(Tw) and send it to the CS.

**Phase 4- Search Outcome** (k, I, FI, EncK(Tw)): Identify the documents Di ∈ D as the outcome of the search as follows:
1) Using the session key ks decrypt EncK(Tw).
2) Using algorithm (ExtendedVector) (presented in the Section IV) Convert T′ to a set L.
3) For each signature vector SVm stored in FI, convert to a set L′ using the algorithm (ExtendedVector). Calculate the Jaccard Similarity JS(L, L′).
   - If JS is 0 do:
     - Calculate the Euclidean Norm d′(T′, SVm)
4) The required keyword will have d ≈ d′ varying by ϵ (threshold).
5) Identify the corresponding masked keyword identifier mask(id(W)) in I and return the encrypted document identifiers X = EncK(Dnum) in ranked order.

Return X to the client.

**Phase 5-Dec** (K, X): Given X a set of encrypted document identifiers, decrypt X using the master key K to uncover the outcome of the search.

VII. Security Analysis

It is not possible to present a SE construction that does not leak information to the adversary. The proposed scheme limits the leakage by generating probabilistic trapdoors. Although the proposed scheme leaks very less information as compared to prior existing schemes, it is important to analyze the leakage profile. This leakage profile includes all leakages that may be important or unimportant and encrypted or unencrypted. The leakage profile is formed over involved artifacts including the index table I, fuzzy index FI, trapdoor Tw generated for a particular keyword and the outcome of the search. The leakages are as follows:

1) Leakage L1: This leakage highlights the information revealed by the index table. The index table I is generated by the Client and outsourced to the CS. This leakage is defined as follows:

\[ L_1(I) = \{\text{Total number of keywords}\} \]

2) Leakage L2: This leakage highlights the information revealed by the fuzzy index table FI. FI is generated by the Client and outsourced to the CS. This leakage is defined as follows:

\[ L_2(FI) = \{\text{Total number of keywords}, SV_W[Q]\} \]

3) Leakage L3: This leakage is associated to the information revealed by the trapdoor T generated for a particular keyword W. The probabilistic trapdoor T is generated by the Client and sent to the CS. Using the trapdoor, the CS searches on the client’s behalf. The leakage is defined as follows:

\[ L_3(T_w) = \{\text{Probabilistic Trapdoor T′}, \text{Euclidean Norm}(d(T, T′)), \text{Number of required documents}\} \]

4) Leakage L4: This leakage is bound to the search outcome (SO) of the trapdoor generated for a particular keyword represented as Tw. This leakage is induced as a result of the search carried out by the CS. This leakage is defined as follows:

\[ L_4(SO) = \{\text{OC}(W), \text{EncK(id(Di))} \forall T_w \in D\} \]

where OC represents the relevant outcome corresponding to the searched keyword.

**Discussion on Leakage:** In [30] the possible attacks are studied on the SE schemes that require a relational database based on Order Preserving Encryption (OPE). It may be observed that the proposed scheme does not require a relational database, therefore, the masking function can be strengthened by using OPE.

Referring to the leakage associated to the inverted index table (I), it may be observed that the index may only leak the presence or absence of an encrypted keyword within a document. The leakage related to fuzzy index table (FI) leaks the total number of keywords that form a fuzzy index table (FI) but the keywords itself can never be uncovered. The trapdoor is unlinkable as it is probabilistic, therefore, it does not reveal the accessing pattern prior to the search. The outcome of the search is only revealed. We now discuss the impact of these leakages on the security of the system as done in [15].

**Lemma 1.** The Fuzzy Ranked Searchable Encryption Scheme (FRSE) presented in the Section VI is “Privacy Preserving” according to Theorem 1, as it is L1, L2, L3, L4-secure and according to Definition (5.1,5.2) where L1 is associated with the index table (I) and leaks the total number of keywords. L2 is related to the fuzzy index table (FI) and leaks the total number of keywords, signature vectors (SV). L3 leaks a probabilistic vector T′, Euclidean norm and number of required documents. L4 leaks the outcome of a trapdoor and the encrypted file identifiers containing the searched keywords.

**Proof Sketch.** The security proof of the proposed scheme is achieved in two-fold; firstly it is proved that the scheme provides keyword-trapdoor and trapdoor-index indistinguishability resulting in the privacy preserving system. Secondly, the leakage profiles are analyzed to validate the conformance to the privacy preserving property.

Referring to the scheme proposed in Section VI, the algorithm comprises of five phases. The KeyGen phase generates two keys and a random number r, (K, ks, r) ← KeyGen(1^l). The Build Index phase generates an inverted index table and a fuzzy index.
The inverted index table \((I, FI) \leftarrow Build\_Index(S', D)\). The \(T_W \leftarrow Build\_Trap(S', k_s, W, num)\) generates a trapdoor corresponding to the keyword \((W)\) to be searched. The \(X \leftarrow Search\_Outcome(k_s, I, FI, T_W)\) represents the outcome of the search and \(D_i \leftarrow Dec(K, X)\) represents the decrypted document identifiers. In order to validate the security of the proposed scheme against this lemma, we first prove that our scheme satisfies the security definitions 5.1 and 5.2. The following claim is made that leads to prove that the proposed FRSE scheme satisfies the definitions 5.1 and 5.2.

**Claim:** If a FRSE scheme is Keyword-Trapdoor indistinguishable then Trapdoor-Index indistinguishability also holds true.

Firstly, we show that if there exists a polynomial-size adversary \(A\) that succeeds in the experiment \(key\_TrapFRSE,A(\lambda)\) with non-negligible probability over \(1/2\), then there exists a polynomial-size adversary \(B\) and a polynomial-size distinguisher \(D\) that distinguishes between the outputs of the experiment \(Trap\_IndexFRSE,B(\lambda)\).

The adversary \(B\) computes \((st_A, w_i) \leftarrow A_i(st_A, T_{w_i}, \ldots, T_{w_j})\); samples \(b \overset{\$}{\leftarrow} \{0, 1\}\). As a result \(B\) outputs the \(st_B = (st_A, b)\). The distinguisher \(D\) proceeds as follows:

1. Parses \((st_A, w_i) \leftarrow A_i(st_A, T_{w_i}, \ldots, T_{w_j})\) where \(1 \leq i \leq N\);
2. Computes \(b' \leftarrow A_{N+1}(st_A, T_{w_k})\);
3. Outputs \(1\) if \(b' = b\) and \(0\) otherwise;

Since \(A_i\) are polynomial size adversaries, therefore \(B\) and \(D\) are also polynomial size. The next phase would be to guess the success rate of \(D\). It may be noted that \(D\) will only correctly guess \(b\) and output 1 when the views of adversaries \(A_i\) may be different. It is to be noted that the trapdoor is randomized even for the same keyword searched twice, therefore it is independent of \(b\). An adversary will guess \(b\) with the maximum probability of \(1/2\), which is according to the definition 5.1.

Therefore, the initial assumption of having such an adversary that succeeds in the experiment \(key\_TrapFRSE,A(\lambda)\) with a non-negligible probability over \(1/2\) is wrong. Hence the distinguisher \(D\) cannot distinguish between the output induced by the experiment \(Trap\_IndexFRSE,A(\lambda)\) with non-negligible probability over \(1/2\). This also validates the definition 5.2. Hence the claim is correct.

So it remains to prove that an FRSE is “Privacy Preserving”. As discussed earlier, the FRSE scheme is based on probabilistic trapdoors that maintains unlinkability and indistinguishability. Since the proposed scheme is based on probabilistic trapdoors therefore a unique trapdoor is generated for the same keyword searched twice. The proposed trapdoor is primarily based on a randomly permuted vector of the order \(2^6!\) combinations that cannot be solved in polynomial time and leads to the NP-Hard problem. Therefore it is not possible for an adversary to form a relationship between the keyword, trapdoor or index table entries within polynomial time and prior to the search. This is also valid for an adaptive adversary maintaining a history of the search and the outcome. Therefore, the proposed scheme is in accordance with the proposed definitions. Therefore, this leads to the privacy preserving FRSE scheme on the whole and leads to the property of “Privacy-by-design”.

The next step is to prove that the aforementioned leakages do not affect the security of the scheme. It is observed that the proposed FRSE scheme is based on the NP-hard problem as there is no polynomial adversary to enumerate \(2^6!\) combinations of permutations. The assumption here is that the master key \((K)\) is kept secure. Therefore, the proposed scheme is \((L_1, L_2, L_3, L_4)\)-secure against adaptive/non-adaptive indistinguishability attacks and provides Keyword-Trapdoor Indistinguishability and Trapdoor-Index indistinguishability.

Therefore, considering the definitions (5.1,5.2), theorem 1, the leakages \(L_1, L_2, L_3, L_4\) and the associated proofs the proposed FRSE scheme is Privacy Preserving.

**VIII. PERFORMANCE METRICS**

**A. Algorithm Analysis**

In order to analyze the computational complexity it is important to perform the asymptotic analysis of the proposed scheme against similar existing schemes. The asymptotic analysis helps to measure the upper bound time complexity of the schemes. The complexity associated to a set of documents is represented by \(n\) and for the keywords by \(m\). As most of the schemes comprise of the 5 phases i.e., KeyGen, Build_Index, Build_Trap, Search_Outcome and the Dec phase, therefore we analyze these phases separately to perform the asymptotic comparative analysis. It may also be observed that since the KeyGen and Dec phases are identical to any other scheme, we do not take their complexities into consideration. Our analysis also considers ranking and non-ranking separately because the well known existing fuzzy scheme presented in [8] does not rank the documents. This will help readers to easily relate the complexities of the proposed scheme against similar existing schemes. Table I shows the comparative algorithmic analysis against similar existing schemes.

From the complexity analysis it can be observed that the proposed FRSE scheme, in the build_index phase, builds two indexes i.e. the inverted index table \((I)\) and the fuzzy index table \((FI)\). The inverted index table \(I\) enabling ranking requires \(\Theta(2mn + n)\) and unranked inverted index generation requires \(\Theta(mn + n)\). The fuzzy index table \(FI\) is not affected by the ranked or unranked searching, therefore, the complexity induced is \(\Theta(Qm)\), where \(Q\) represents the random permutations used for min hashing. As a result the total complexity of the ranked Build_Index phase is \(\Theta(2mn + n + Qm)\). The Build_Trap phase is asymptotically bound by \(\Theta(Q + 1)\). As mentioned in Section VI, the Search_Outcome is performed at the CS side. This phase is asymptotically bound by \(\Theta(2Qm + mn)\) in terms of ranked searching and \((2Qm + n + 1)\) for unranked searching. Apart from this, \(S\) is the complexity associated to [22] and represents the wildcard-based fuzzy set construction.
TABLE I
ALGORITHMIC COMPARATIVE ANALYSIS

<table>
<thead>
<tr>
<th>SCHEMES</th>
<th>Build_Index</th>
<th>Build_Traps</th>
<th>Search_OC</th>
</tr>
</thead>
<tbody>
<tr>
<td>[22]</td>
<td>(\Theta(mS))</td>
<td>(\Theta(mS + 1))</td>
<td>(\Theta(m^2))</td>
</tr>
<tr>
<td>[8] and [23]</td>
<td>(\Theta(mn + Qm))</td>
<td>(\Theta(Qm + 1))</td>
<td>(\Theta(2m^2))</td>
</tr>
<tr>
<td>This paper (ranked)</td>
<td>(\Theta(2mn + n + Qm))</td>
<td>(\Theta(Q + 1))</td>
<td>(\Theta(2Qm + mn))</td>
</tr>
<tr>
<td>This paper (unranked)</td>
<td>(\Theta(mn + n + Qm))</td>
<td>(\Theta(Q + 1))</td>
<td>(\Theta(2Qm + n + 1))</td>
</tr>
</tbody>
</table>

[8], [22] and [23] are not ranked SE schemes.

B. Storage Overhead

Another important metrics for measuring the performance of the scheme is the storage overhead analysis. This is dependent upon the data structure and helps analyse the amount of data stored either on the client’s side or the server side. We analyse both of the storages (i.e. the client and the server) separately. Referring to the proposed scheme presented in the Section VI it may be observed that the client stores the Master key \(K\) and the session key \(k_s\). Having the security parameter \(\lambda\), the size of the keys are 128bit. The client also stores a randomized shingle vector \(S’\) having the size \(26^2 = 676\) bits. Furthermore, \(Q\) permutations are also stored that are used to form a signature vector \(SV\). As mentioned earlier the variable \(Q\) increases/decreases the accuracy of the results. Considering \(Q = 10\)bit, the total storage required at the client’s side is \(128 + 128 + 676 + 10 = 942\) bits\(= 942/8\) bytes. This means the client requires only 117.75 bytes in terms of storage overhead.

Referring to the storage overhead of the CS, firstly the encrypted documents need to be stored. Having \(n\) documents and \(avg\) representing the average size of each document, this storage can be represented as \(n \times D_{avg}\). The fuzzy index table \(FI\) requires a storage space of \((m^*Qm)\), where \(m\) are the total number of keywords. The inverted index table \(I\) incurs a storage overhead of \(8(mn)\). The session key \(k_s\) is also stored at the CS. The total storage overhead at the CS is \(n \times D_{avg} + m \times Q + 8(mn) + 128\) bits.

We now discuss the storage overhead of the scheme presented in [8]. For clear and concise comparison we use similar abbreviations such as in our scheme. Having the security parameter \(\lambda\) the scheme requires to store the secret key \(K\) which is a combination of \((M_1, M_2, S)\), where \(M_1, M_2\) are matrices of the order \(\lambda \times \lambda\) and \(S\) is a vector of the order \(\lambda\). So having \(\lambda = 128\) bits, the CS requires \(128 \times 128 = 2048\) bits. The CS stores the secure index that is similar to a per document bloom filter i.e., \(n \times (n+m \times 128)\) bit. Whereas, the outsourced encrypted documents require the same storage as our proposed scheme.

It can be observed that our proposed scheme outperforms the scheme presented in [8] in terms of Client and CS storage overhead. We also refer readers to [31] that discusses the storage overhead induced by existing multi-keyword schemes.

IX. COMPUTATIONAL ANALYSIS

Before discussing the computational overhead, it is important to discuss the BTCS architectural details, the dataset related information and the system specifications that effect the computational results. This also helps to give more insight on the performance of the proposed scheme.

A. Application Encryption Service

The Application Encryption (AE) service is available as a part of the British Telecommunications Cloud service that implements and offers cryptographic services for its clients. These services include core cryptographic operations like encryption/decryption based on symmetric ciphers and cryptographic-hash based integrity checking, as well as supporting operations like key management, key storage and key retrieval etc. through a Key Management Service (KMS). The KMS component, in addition to the storage and management of the cryptographic keys, also enforces policy-based access control over the client’s keys. The AE service can also be hosted inside the clients’ premises for their complete control and trust in the cryptographic operations, or the clients can even construct their own version of the AE service from scratch as it is based on open standards and technologies. However, in our deployment scenario the AE service offered by BT is FIPS-140-1 certified and is treated as a trusted third party.

The AE service provides the following features to the clients:

- Centralized control and management of application-layer encryption services through XACML policies.
- NIST standard implementation of Advanced Encryption Standard (AES-256), RSA, SHA-256 and other cryptographic primitives.
- Provides a library that implements the OASIS PKCS#11 APIs, which the clients can integrate in their applications.

Figures 3 and 4 illustrate the activity diagrams that depict the flow of events of the proposed scheme when deployed onto the BT Cloud Storage Service and when the AE service is used.
B. Dataset Description

A realistic dataset of telephone speech is used for testing the performance of the proposed scheme. This dataset has been previously used in [15], [27] to study the performance of SE schemes. The dataset used is a Switchboard-1 Telephone Speech Corpus (LDC97S62) originally collected by Texas Instruments in 1990-1, under DARPA sponsorship. The Switchboard-1 speech database [32] 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. In total 120,000 distinct keywords are identified from the dataset.

C. Implementation Details

The proposed FRSE scheme has been implemented in Java and the results have been presented in the form of graphs using Excel 2013. The server side has been deployed onto a public Cloud platform (British Telecom's Cloud Storage). Whereas, the client-side is a local workstation. The client interacts and imports the required cryptographic primitives from the Application Encryption (AE) server. The communication between the client-side and the server-side is done through sockets. Therefore the results also include the network latency.

- **Client Side**: Entire algorithm (other than the Search_Outcome) is implemented at the client's system. The workstation used runs with an Intel Core i5 CPU running at 3.00GHz and 8GB of RAM.
- **Server Side**: Only the Search_Outcome is implemented at the BTCS. In other words, the entire search algorithm is delegated to the BTCS. The allocated resources include a Dual Core Intel (R) Xeon (R) CPU E5-2660 v3 running at 2.60GHz and 8GB of RAM.

D. Computation Overhead

In this section the computational time for the different phases of the proposed scheme is analyzed. As discussed in Section VI, the proposed scheme comprises of five polynomial time algorithms. We skip the computational time analysis of the KeyGen phase and the Dec phase. It is to be mentioned here that these graphs are generated directly from actual results obtained from the experiments and no data normalization techniques have been applied on the data.

Starting with the Build_Index phase, two main tasks are performed in this phase, i.e., the inverted index generation and the fuzzy index generation. It is again emphasized that the Build_Index is a one-time process. The computational cost for the inverted index generation mainly comes from the relevance score generator. Figure 5 highlights the computational cost of the algorithm. The inverted index is generated for fixed number of keywords, 120,000, and variable amount of documents. The experiment starts with 100 documents that are incremented by 100 on every iteration to a maximum of 1600 documents. The number of documents are presented along the x-axis and the time in seconds is presented along the y-axis. The graph shows a linear growth with the increase in the number of documents that can be realized upon normalizing the graph. For 1600 documents, inverted index generation takes 14.68 seconds.

In the next step we analyze the computational time for the fuzzy index generation. This computation is mainly incurred due to the shingaling of keywords and applying min hashing. The fuzzy index is not effected by the number of documents, therefore, only the number of keywords are varied. The results are presented in the form of a graph in Figure 6. The experiment starts with 10,000 keywords that are scaled to a maximum of 120,000 by gradually adding 10,000 on every iteration. The number of keywords are along the x-axis and the time in seconds is along the y-axis. It is observed that the fuzzy index shows a linear growth with the increase in the number of keywords. For 120,000 keywords, the fuzzy index generation requires approximately 2.84 seconds.
only facilitates single keyword search queries. In the experiment, a trapdoor is generated for the wrongly spelt keyword “aboot”. The same keyword is used to analyze the computational time of proceeding phases. The proposed scheme takes an average time of 0.09 seconds for generating the trapdoor.

The next phase is the Search_Outcome phase and performed on the BTCS. The computational time is analyzed in threefold. Firstly, the effect of varying number of keywords on the search results is analyzed. Secondly, the efficiency of search is analyzed by varying the number of documents. Lastly, the result is analyzed that also includes the network latency while retrieving the results directly from the BTCS.

It is observed that the Build_Index is highly effected by the change in the number of keywords and documents. This also means that with the change in the size of the index table the computational time of the Search_Outcome phase is also effected. Firstly we analyze the effects of varying number of keywords. Referring to the Figure 7, the number of keywords are changed ranging from 10,000 to 120,000. The number of keywords are presented along the x-axis and the time in seconds is along the y-axis. It is observed that for 100 documents and with the increase in the number keywords the efficiency decreases while searching for the keyword “aboot”, however, the graph shows a linear growth. Hence, for scenarios that require a few set of keywords, the search is very efficient. The search time for 10,000 and 120,000 keywords are 0.17 and 2.56 seconds respectively.

Next we analyze the effect of altering the number of documents. With increase in the number of documents the performance decreases. The reason for this decrease is due to the increase in the search space where the ranking is to be applied. The ranking is achieved through the sorting of the relevance scores which effects the computational cost. While searching for the keyword “aboot” takes around 2.17 seconds for 100 documents and 43.1 seconds for 1600 documents as shown in Figure 8.

Once the effects of the search time with varying amounts of keywords and documents has been analyzed, we search for the keyword “aboot” where we have an inverted index table generated for 1600 documents and fuzzy index table comprising of 120,000 keywords. We only vary the number of required documents starting from 100 and incrementing in steps of 100 to attain a maximum of 1600. The results are illustrated in Figure 9. The results also include the network latency incurred due to the communication between the BTCS and the Client through sockets. While searching for the keyword “aboot” the fuzzy search result includes the documents containing the keywords “about, abort, abouts”. It is observed that the scheme shows a linear growth. The number of required documents are presented along the x-axis and the time in minutes along the y-axis. The search time for 1600 documents is around 11.1 minutes.

1) Result Accuracy: To discuss the accuracy of the scheme, we analyze the precision and recall as described in [33]. The precision is represented as $\frac{t_p}{t_p + f_p}$ whereas the recall is defined as $\frac{t_p}{t_p + f_n}$. Before proceeding further, it is important to define false positive $f_p$ and false negatives $f_n$. In the proposed construction, false positives are those keywords that are not required but appear in the search. Similarly, a false negative represents the set of keywords expected to appear in the search.

For the experiment we randomly pick 100 keywords from the possible 120,000 keywords and we generate the results for the keyword “aboot”. Figure 10 shows the performance of the proposed scheme while varying the threshold $\epsilon$. If the keyword
is correctly spelt then the exact matching takes place and that would lead to 100% accuracy, however, in fuzzy search we use min hashing and apply Euclidean Norm and Jaccard similarity to compute the similarity between the trapdoor and the fuzzy index. By varying the $\epsilon$ value the accuracy of the results changes. We observe that with the increase in the threshold value the precision increases. For the threshold value of 0.1, the precision is 25% that increases to 67% when $\epsilon$ is 0.6 and to a maximum of 100% for $\epsilon$ value of 0.9. We also notice a slight fluctuation in the precision at different intervals due to the use of probabilistic trapdoor. During our experiment the fluctuation took place when the value of $\epsilon$ was 0.8. It is also observed that unlike [8] with the increase in $\epsilon$, the recall also increases. This is due to the reason that we are seeing an increase in the true positives with the increase in the $\epsilon$ value. The proposed scheme attains a maximum of 67% recall value when the value of $\epsilon$ approaches 0.9. It is also observed that the accuracy of [8] is better as compared to the proposed scheme, the reason being the small dataset that the authors have used for the experiment comprising of only 20 keywords. The accuracy can also be increased by reducing the number of keywords in the fuzzy index table.

**REFERENCES**


