A New Lightweight Symmetric Searchable Encryption Scheme for String Identification

Indranil Ghosh Ray, Yogachandran Rahulamathavan and Muttukrishnan Rajarajan, Senior Member, IEEE

***Abstract*—In this paper, we provide an efficient and easy-to-implement symmetric searchable encryption scheme (SSE) for string search, which takes one round of communication, $O(n)$ times of computations over n documents. Unlike previous schemes, we use hash-chaining instead of chain of encryption operations for index generation, which makes it suitable for lightweight applications. Unlike the previous SSE schemes for string search, with our scheme, server learns nothing about the frequency and the relative positions of the words being searched except what it can learn from the *history*. We are the first to propose probabilistic trapdoors in SSE for *string search*. We provide concrete proof of non-adaptive security of our scheme against *honest-but-curious* server based on the definitions of [12]. We also introduce a new notion of *search pattern privacy*, which gives a measure of security against the leakage from trapdoor. We have shown that our scheme is secure under search pattern indistinguishability definition. We show why SSE scheme for string search cannot attain adaptive indistinguishability criteria as mentioned in [12]. We also propose modifications of our scheme so that the scheme can be used against *active* adversaries at the cost of more rounds of communications and memory space. We validate our scheme against two different commercial datasets (see [1], [2]).**

***Index Terms*—Cloud storage, Symmetric key, Searchable encryption, hash-chain, lightweight cryptography.**

I. Introduction

The cloud is designed to hold a large number of encrypted documents. With the advent of cloud computing, growing number of clients and leading organizations have started adapting to the private storage outsourcing. This allows resource constrained clients to privately store large amounts of encrypted data in cloud at low cost. However, this prevents one from searching. This gives rise to a newly emerging field of research, called *searchable encryption* (SE). SE can be classified into *symmetric searchable encryption*s (SSE) and *asymmetric searchable encryption*s (ASE). In this paper, we study the SSE for *string search*. In the SSE, the client encrypts the data and stores it on the cloud. It may be noted that client can organize the data in an arbitrary manner and can maintain additional data structures to achieve desired data efficiently. In this process, the initial client side computation is thus as large as the data, but subsequent computations to access data is less for both client and the cloud server.

Since huge volumes of documents are stored in a cloud server, searching against a *keyword* may result into large number of documents, most of which are not intended, causing unnecessary network traffic. This motivates the idea of searching against a *string*, which allows the search to be more specific. Searching for *string* is a *multi keyword* search where the ordering of keywords is preserved. So in addition to the presence of all these keywords in a document, their *ordering* and *adjacency* are to be taken care off while searching. So the *index table* needs to be prepared in such a way that the adjacency information of the words can be preserved.

Although few works are available in the literature involving string search (e.g. [15], [18], [22], [23], [25]), but most of them lack formal security proof against the revised definitions of [12] and also expose lots of informations to the server following the search (see Table I of Section II). In the SSE scheme, the server is expected to learn nothing about the search queries and data collections. SSE achieves this by using symmetric cryptographic primitives instead of heavy computations of public key encryption at the cost of small leakage of information [12]. Here we take an example which will be extended throughout the paper to illustrate our algorithms and data structures.

Example 1: Let us consider the text file ToyExample.txt in the following figure, which is to be encrypted and uploaded to a public cloud for future search.

ToyExample.txt

This is a
demonstration text
for showing how line
breaking works.

Let us consider the set of keywords as follows :
$keywords = \{this, is, a, demonstration, text, for, showing, how, line, works, breaking\}$.
The client encrypts and outsources this file to the server and later he wants to search a string *"this is a demonstration"*.

Remark 1. *While generating the index, client converts all letters into lower case and generate the index and trapdoors accordingly.*

It may be noted that in traditional SE schemes, this is treated as multi-keyword search for keywords : *this, is, a*, and *demonstration*. The drawback of this approach is that the adjacency of the words is not considered. However, the proposed string search not only looks for those keywords, but also consider the order. We will continue using this example in the subsections of Section IV to explain different phases of our proposed scheme.

Our Contributions : In [12], authors proposed the first efficient SSE construction, achieving sublinear search time and introduced the notion of *non-adaptive* and *adaptive* indistinguishability definitions of security for SSE. In the same work, authors introduced the idea of *history* connected to a finite number of consecutive keyword searches. We extended that definition for string search as is necessary for security proof, and call it *history-of-strings*. Using this new definition, we carryout the necessary changes in the definition of *non-adaptive* indistinguishability [12] for SSE performing string search. Finally we prove that our proposed scheme is secure under the *non-adaptive* indistinguishability definition of SSE security against *honest-but-curious* server. Although indistinguishability definition of SSE security takes care of the the security of keyword from index, however it does not provide security against the leakage from trapdoor. Towards this we have introduced the notion of *search pattern security* and have shown our scheme to be secure under *search pattern indistinguishability* definition. The novelty of our scheme is

Indranil Ghosh Ray is with the Department of Electrical and Computer Engineering, City, University of London. Email: Indranil.Ghosh-Ray@city.ac.uk
Yogachandran Rahulamathavan is with the Institute for Digital Technologies, Loughborough University London. E-mail: y.rahulamathavan@lboro.ac.uk
Muttukrishnan Rajarajan is with the Department of Electrical and Computer Engineering, City, University of London. Email: r.muttukrishnan@city.ac.uk

that although the index is generated by the client at the beginning, and remains same for the same dataset through out the process and thus static in nature, however the trapdoors are dynamic in nature, making it more difficult for the eavesdroppers to understand the search patterns and thus is more secure against attacks like replay attacks, frequency analysis based attacks and many more.

Our scheme achieves searches in one communication round and requires $O(n)$ times computations for searching a string in n documents, which is optimal. Also the scheme requires no storage on the client side and $O(n)$ storage on the server side for the n-document collection. Lastly, the scheme guarantees minimal leakage in a sense that server directly knows nothing about the frequency of the words being searched and their relative positions in the documents except what it can learn from the *history* of search. Unlike the index generation techniques (sequence of encryptions of a key) used in [12], [15], we use the *hash-chain* technique, which is faster, and is thus suitable for *lightweight* applications.

For the first time we address the problem of string search using symmetric searchable encryption against the *active adversary*, who by trick can place a document of his choice in the document collections. We propose a modification of our scheme to deal with *active adversary* securely at the cost of maintaining a list of keywords at the client's end and two rounds of communications.

We also implement the scheme against two different commercial datasets, namely, a 20 MB DNA dataset [1] and a 19 MB TIMIT speech data [2] and successfully achieve string search functionality in encrypted domain.

Rest of the paper is organized as follows: In Section II we discuss the related works in SSE. In Section III, we provide definitions and preliminaries. In Section IV, we discuss our proposed scheme in detail. In Section V, we analyze the security of our proposed scheme. In Section VI, we discuss the time and space complexity of our scheme and provide experimental results of our scheme against commercial datasets (see [1], [2]). We conclude the paper in Section VII.

II. Related Works

For the last ten years, searchable encryption has been the focus for many leading research groups and several results were proposed [3]–[10], [12], [16], [17], [19], [20], [24]. In [3], authors defined computational and statistical relaxations of the existing notion of perfect consistency and provided a new scheme that was statistically consistent. They also proposed a transformation of an anonymous *identity based encryption scheme* (IBE) to a secure *public key encryption with keyword search scheme* (PEKS) that guarantees consistency. In [4] authors presented as-strong-as-possible definitions of privacy and some constructions for public-key base encryption schemes where the encryption algorithm is deterministic. In the same work, new methods were proposed for database encryption that permit fast (i.e. sub-linear, and in fact logarithmic, time) search while provably providing privacy that is as strong as possible subject to this fast search constraint. The work in [4] also generalizes their methods to obtain a notion of efficiently searchable encryption schemes which permit more flexible privacy to search-time trade-offs via a technique called *bucketization*. In [5], authors studied the problem of searching on data that is encrypted using a public key system which they referred as PEKS and provided several constructions. In [6], authors show how to create a public-key encryption scheme that allows PIR (private information retrieval) searching over encrypted documents. Their solution was the first to reveal no partial information regarding the users search (including the access pattern) in the public-key setting and with small communication complexity. In [7], authors defined and solved the problem of privacy-preserving *multi-keyword ranked search* over encrypted data in cloud computing (MRSE). They established a set of strict privacy requirements for such a secure cloud data utilization system. Among various multi-keyword semantics, they choose the efficient similarity measure of ”coordinate matching,” i.e., as many matches as possible, to capture the relevance of data documents to the search query. They also used ”inner product similarity” to quantitatively evaluate such similarity measure. They provide two MRSE schemes to achieve various stringent privacy requirements in two different threat models. In [24], authors proposed an efficient searchable encryption scheme for auction (SESA) in emerging smart grid marketing, which is based on a public key encryption with keyword search technique to enable the energy sellers to inquire suitable bids while preserving the privacy of the energy buyers. In [8] authors provided a systematic study of various attack models against SSE based schemes.

Dynamic SSE was first considered by Song et al. [19], but no solution with sublinear search time existed before the work of Kamara et al. [13]. Recently, two new dynamic SSE schemes have been proposed. The first one, by Cash et al. [9], which is an extension of [10]. They showed that SSE is feasible on very large databases. In [9], authors designed and implemented dynamic symmetric searchable encryption schemes that efficiently and privately search server-held encrypted databases with tens of billions of record-keyword pairs. Their basic theoretical construction was for single-keyword searches and which offers asymptotically optimal server index size, fully parallel searching, and minimal leakage. In [10], authors presented another efficient SSE scheme which supports complex queries involving multiple keywords. Similar scheme may be found in [17]. In [11], authors studied the trade-off between locality and server storage size of SSE schemes.

In [12], authors introduced the idea of SSE with improved security definitions. They introduced the two most important security definitions, namely *non-adaptive indistinguishability* and *adaptive indistinguishability*. They also proposed SSE schemes for keyword search which they proved to be secure under these new security definitions.

In [16] authors studied the security provided by various encrypted databases and presented a series of attacks that recover the plaintext from encrypted database columns using only the encrypted column and publicly-available auxiliary information. In [17], authors studied efficient sub linear search techniques for arbitrary Boolean queries. They considered scalable DBMS with provable security for all parties, including protection of the data from both server (who stores encrypted data) and client (who searches it), as well as protection of the query, and access control for the query.

In [20], Stefanov et al. designed scheme for the first time to address *forward secrecy*. However, the problem of malicious servers has not been studied, except in [20], but, as we will see later, their proposition is flawed. In [19], the search complexity is linear in the number of documents stored in the database.

In [15], [18], [22], [23], [25], SSE schemes are developed for string search. In [22], authors designed the first SSE scheme for phrase search. This scheme works in two phases, each taking one round of communication. In the first phase these documents are identified which contains all words occurring in the phrase. In the second round the candidate documents are checked to confirm the existence of the phrase. In [15], authors proposed a scheme for string search in *non-adaptive* setting where they used some additional data structures and techniques (list, lookup tables, pseudo random functions and hash-chains for word sequencing) to keep track of position informations. In [15] the index generation technique is similar to the index generation of [12] and requires a sequence of encryption operation while forming index. In this paper, we achieve the same non-adaptive security by using a sequence of hashing instead of encryption operations which is faster and suitable for lightweight applications. In [23] schemes are proposed that enables efficient searching for an arbitrary string that may not be extracted as keywords at the cost of leaking some information for the sake of efficiency. In [25], authors introduced a SSE scheme that allows both encrypted phrase searches and proximity ranked multi-keyword searches to encrypted datasets on untrusted cloud. In [18], authors propose a faster way of secure string search based on bloom filters. It may be noted that our approach is based on index based scheme and we prove it to be non adaptively secure according to the definition introduced in [12].

TABLE I
PROPERTIES AND PERFORMANCES OF DIFFERENT SSE SCHEMES. SEARCH TIME IS PER KEYWORD, WHERE n IS THE NUMBER OF DOCUMENTS.

Property	SSE [12]	PSS [22]	[25]	LPSSE [15]	this paper
String search	no	yes	yes	yes	yes
non-adaptive security	yes	yes	no	yes	yes
adaptive security	yes	no	no	no	no
security against active adversary	no	no	no	no	yes
client storage	no	dictionary	trusted server	no	no
no of rounds	1	2	2	1	1
storage cost	$O(n)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$
no. of encryptions per keyword	$O(n)$	$O(n)$	$O(n)$	$O(n)$	1

Here we summarize our contributions:

1. We propose a *non-adaptively* secure SSE scheme for *string search* which takes one round of communication, $O(n)$ times of computation over n documents, $O(n)$ additional memory in the server side and no memory in the client side.
2. We provide a formal and proof to show that the scheme is *non-adaptively* secure.
3. Although schemes of [12] is secure against non-adaptive indistinguishability definition of security, but they does not guarantee security against the leakage from trapdoor. Unlike the previous SSE schemes, this is the first SSE scheme for string search which generates *probabilistic* trapdoors which allows *search pattern privacy*. We formally introduce this new notion of security in SSE and show that our scheme is secure under *search pattern indistinguishability* definition (see Definition 9).
4. In the previous SSE schemes for string search, the index tables are generated by creating linked-lists corresponding to every keywords, where informations related to occurrence of the keyword in i-th document is stored in i-th node along with the key, say k_{i+1}, and is encrypted with a key, say, k_{i-1}. Which leads to a sequence of encryption functions for generating the index and a sequence of decryption functions while searching. We achieve the same non-adaptive security by applying sequence of masked hashing, i.e., hash-chain, which is faster and suitable for lightweight applications.
5. We propose modifications of our scheme so that it can be used against *active adversary*.

III. Notations and Definitions

Document collections and Data Structures: Let $\triangle = \{w_1, w_2, \ldots, w_d\}$ be a dictionary of d words and $\mathbb{P}(\triangle)$ be the set of all possible documents which are collections of words. Let $\boldsymbol{D} \subseteq \mathbb{P}(\triangle)$ be the collection of n documents $\boldsymbol{D} = (D_1, D_2, \ldots, D_n)$. Let $id(D_i)$ be the unique identifier for the document D_i. We denote the list of all n document identifiers in $\boldsymbol{D}$ by $id(\boldsymbol{D})$, i.e., $id(\boldsymbol{D}) = \{id(D_1), \ldots, id(D_n)\}$. Furthermore, let $\boldsymbol{D}(w_j)$ be the collection of all documents in $\boldsymbol{D}$ containing the word w_j. A string s of l words is an ordered tuple $(w_1, w_2, \ldots, w_l)$. Let $\boldsymbol{D}(s)$ denotes a collection of documents in $\boldsymbol{D}$ that contains the string s. It is easy to check that $\boldsymbol{D}(s) \subseteq \bigcup_{i=1}^{l} \boldsymbol{D}(w_i)$. We denote by $\delta(\boldsymbol{D})$, all the distinct keywords connected to the document collection $\boldsymbol{D}$. We denote by $f_{i,j}$ the frequency of occurrence of the keyword w_j in D_i. Let $f_j = max\{f_{i,j} | 1 \le i \le n\}$. Also let $f = max\{f_j | 1 \le j \le \delta(\boldsymbol{D})\}$.

Since the index is meant for server to perform search, we call it server side index and denote it by ***SI***. The generation of ***SI*** involves generation of *index table* $\boldsymbol{I}$ along with the vectors $\boldsymbol{I}_r$ and $\boldsymbol{I}_c$. $\boldsymbol{I}$ is a $n \times |\delta(\boldsymbol{D})|$ array where i-th row corresponds to i-th document D_i and j-th column corresponds to the word w_j and (i, j)-th element is denoted by $\boldsymbol{I}[i][j]$. $\boldsymbol{I}[i][j]$ is a set of f number of λ-bit strings treated as element of $\mathbb{Z}_p$, where p is a (λ)-bit prime, λ being the security parameter.

Cryptographic Primitives: Here we define cryptographic primitives that are needed for our SSE scheme for string search.

Definition 1. *A symmetric key encryption scheme is a probabilistic polynomial-time algorithms (Gen, Enc, Dec) such that:*

1. *The key-generation algorithm Gen takes as input the security parameter 1^λ and outputs a key k; we write this as $k \leftarrow Gen(1^\lambda)$ (thus emphasizing the fact that Gen is a randomized algorithm). We will assume without loss of generality that any key k output by $Gen(1^\lambda)$ satisfies $|k| > \lambda$.*
2. *The encryption algorithm Enc takes as input a key k and a plaintext message $m \in \{0, 1\}^*$, and outputs a ciphertext c. Since Enc is randomized, we write this as $c \leftarrow Enc_k(m)$.*
3. *The decryption algorithm Dec takes as input a key k and a ciphertext c, and outputs a message m . We assume that Dec is deterministic, and so write this as $m = Dec_k(c)$. It is required that for every λ, every key k output by $Gen(1^\lambda)$, and every $m \in \{0, 1\}^*$, it holds that $Dec_k(Enc_k(m)) = m$ [14].*

We typically denote an arbitrary negligible function by $negl$ such that for any arbitrary polynomial $p(.)$, there exists an integer a such that for all $\lambda > a$, $negl(\lambda) < \frac{1}{p(\lambda)}$ [14].

Definition 2. *private-key encryption scheme $\pi = (Gen, Enc, Dec)$ has indistinguishable encryptions under a chosen-plaintext attack (or is IND-CPA secure) if for all probabilistic polynomial-time adversaries $\mathcal{A}$ there exists a negligible function $negl$ such that $Pr[PrivK^{cpa}_{\mathcal{A},\pi}(\lambda) = 1] \le \frac{1}{2} + negl(\lambda)$, where the probability is taken over the random coins used by $\mathcal{A}$, as well as the random coins used in the experiment (for choosing the key, the random bit b, and any random coins used in the encryption process) [14].*

Throughout the paper we use the encryption and decryption functions ($Enc_k()$ and $Dec_k()$) which are from IND-CPA symmetric encryption scheme π. We also use *pseudo prime number generator* [21], denoted by $PPNG(1^\lambda)$ which outputs a λ-bit probabilistic prime number. In addition, we use *message authentication code*, $MAC_k(.)$ [14] which outputs in λ bits. We treat these outputs as elements of $\mathbb{Z}_p$, where p is a λ bit prime. We write them simply as $MAC_{k_m}(.)$ and $Enc_{k_m}(.)$. Also we use $\oplus$, $\ominus$ and $\otimes$ as the addition, subtraction and multiplication operators in $\mathbb{Z}_p$. Also for $a \in \mathbb{Z}_p$, a^{-1} denotes the multiplicative inverse in $\mathbb{Z}_p$.

Before closing this section we provide in Table II the list of notations discussed in this section.

TABLE II
SUMMARY OF NOTATIONS.

.

D	document collection.
D_i	i-th document.
n	number of documents.
$id(D_i)$	the unique identifier for the document D_i.
$id(D)$	list of document identifiers in the document collection.
$D(w_j)$	collection of all documents in D containing the word w_j.
$D(s)$	collection of all documents in D containing all words in the string s.
$\delta(D)$	all the distinct keywords connected to the document collection D.
$f_{i,j}$	the frequency of occurrence of the keyword w_j in D_i.
f_j	$max\{f_{i,j} \vert 1 \leq i \leq n\}$.
f	$max\{f_j \vert 1 \leq j \leq \delta(D)\}$.
I_r	I_r contains list of inverted indexes for the list of keywords.
I_c	I_r contains list of encrypted document identifiers.
SI	server side index which is a triplet (I, I_r, I_c).

IV. OUR SCHEME

In this section we present our SSE scheme Π_{ss} for string search, which is composed of four algorithms KeyGen, BuildIndex, Trapdoor and Search. First we formally define the scheme. In the subsequent subsections, we will discuss these algorithms in detail with illustrations.

Scheme 1 (Π_{ss})**.** *The scheme Π_{ss} is a collection of four polynomial time algorithms (KeyGen, BuildIndex, Trapdoor, Search) such that:*

1. *KeyGen(1^λ) : KeyGen is a probabilistic key generation algorithm that is run by the client to setup the scheme (see Algorithm 1). It takes a security parameter λ, and returns a secret master key k_m and a mask-key k' which are to be kept privately at client's end and a session key k_s which is to be shared between client and the server. Client also shares a λ-bit prime p with the server. The length of k_m, k' and k_s are polynomially bounded in λ.*
2. *BuildIndex(k_m, k', k_s, p) : BuildIndex is a probabilistic algorithm run by the client to generate* **SI** *$= (I, I_r, I_c)$. It takes k_m, k', k_s, p and returns* **SI***. Since BuildIndex is randomized, we write this as* **SI** *$\leftarrow BuildIndex_{(k_m,k',k_s,p)}(s)$ (see Algorithm 2).*
3. *Trapdoor(k_m, k_s, p, s) :Trapdoor is a probabilistic algorithm run by the client to generate a trapdoor for a given string of words $s = (w_1, w_2, \ldots, w_l)$. It takes k_m, k', k_s, p and s as input and outputs $t = (t_1, t_2, \ldots, t_l)$, where t_i is the trapdoor corresponding to the word w_i. Since trapdoor is randomized, we write this as $t \leftarrow Trapdoor_{(k_m,k',k_s,p)}(s)$ (see Algorithm 3).*
4. *Search(***SI***$, t)$: Search is run by the server in order to search for the documents in D that contain the string s. It takes k_s,* **SI** *and trapdoor t of the string s as inputs, and returns D(s), the set of identifiers of documents containing the string s. Since this algorithm is deterministic, we write it as D$(s) = Search_{k_s}(SI, t)$ (see Algorithm 4).*

Structure of index: In the array I, informations are stored related to positions of a word in a document. Recall that $I[i][j]$ is a set of f number of λ-bit strings. Now we describe how these bit strings are obtained. To keep track of ordering of words in a document, instead of position pointers, we use *hash-chain*. For a l-word sentence $(w_1, \ldots, w_l)$ in a document, say D_i, we first take a random integer $r \in \mathbb{Z}_p$. Then we form a *hash-chain* $(r_1, \ldots, r_l)$ such that $r_1 = r$ and for $2 \leq j \leq l$, $r_j = MAC_{k_s}(r_{j-1})$. Then we put $r_i + msk_j$ in $I[i][j]$, where $msk_j = MAC_{k'}(w_j)$. It may be noted that since k_m is privately kept at client's end, server cannot compute the mask. The masks are provided to the server through the trapdoors. So for each time the word w_i occurs in D_j some integer from $\mathbb{Z}_p$ is generated as mentioned above and sorted in $I[i][j]$. Since $|I[i][j]|$ can then reveal the frequency of the word w_j in D_i, we insert additional $(f - |I[i][j]|)$ number of random numbers from $\mathbb{Z}_p$ in each $I[i][j]$ so that $|I[i][j]|$ becomes f for all i's and j's. We call it cell-padding. This stops the leakage informations related to frequency and relative positions completely. We call $MAC_{k_m}(w_j)$ and index or client index of the word w_j. $I_r[j]$ contains inverted index of the word w_j, which is a λ-bit string. $I_c[i]$ is encrypted identifier of D_i, where the key of encryption is k_m.

Secure update of index: Our index table is dynamic in a sense that for each update of document collection the index can be updated without disturbing the existing index just by adding a new row corresponding to the new document where the entries in index are computed as described above.

In the following subsections, we provide the algorithms for KeyGen, BuildIndex, Trapdoor and Search in detail with illustrations.

A. Key Generation

Algorithm 1 Keygen

```
Input security parameter $\lambda$.
Output $k_m$, $k'$, $k_s$ and $p$.
  $k_m, k', k_s \leftarrow Gen(1^\lambda)$;
  $p \leftarrow PPNG(1^\lambda)$;
```

B. Index Generation

Algorithm 2 BuildIndex

```
Input $k_m$, $k'$, $k_s$, $p$, $D = (D_1, \ldots, D_n)$.
Output SI $= (I, I_r, I_c)$.
  Form a collection W $= \{w_1, \ldots, w_d\}$ of all distinct words occurring in D;
  $j \leftarrow 1$;
  while $j \leq d$ do
    $ci_j = MAC_{k_m}(w_j)$;
    $I_r[j] = ci_j^{-1}$;
    $j \leftarrow j + 1$;
  end while
  $i \leftarrow 1$;
  while $i \leq n$ do
    $I_c[i] \leftarrow Enc_{k_m}(id(D_i))$;
    For each sentence $s = (w_{s_1}, \ldots, w_{s_l})$ in $D_i$, chose $r \in \mathbb{Z}_p$ randomly and form $(r_1, \ldots, r_l)$ such that $r_1 = r$ and for $2 \leq j \leq l$, $r_j = MAC_{k_s}(r_{j-1})$. Associate $r_j$ with the word $w_{s_j}$.
    $j \leftarrow 1$;
    while $j \leq d$ do
      set $I[i][j]$ as all integers in $\mathbb{Z}_p$ that are associated with the word $w_j$ and add the mask $m_j = MAC_{k'}(w_j)$ with all of them in modulo $p$, i.e., in $\mathbb{Z}_n$;
      if $(|I[i][j]| < f)$ then
        Inject $(f - |I[i][j]|)$ number of random elements from $\mathbb{Z}_p^*$ in $I[i][j]$;
      end if
      $j \leftarrow j + 1$;
    end while
    $i \leftarrow i + 1$;
  end while
```

Example 1 (continued): (Illustration of BuildIndex)
Let the prime number be 31, i.e. $p = 31$. Also let us assume that $MAC_{k_m}(this) = 18$, $MAC_{k_m}(demonstration) = 9$, $MAC_{k_m}(a) = 13$, $MAC_{k_m}(text) = 15$, $MAC_{k_m}(for) = 10$ and $MAC_{k_m}(is) = 6$. It may be noted that the first row of server index table (see Table III), i.e., I_r represents inverses modulo 31 of client index values. For example $MAC_{k_m}(this)^{-1} \bmod 31 = 18^{-1} \bmod 31 = 19$. Thus the column corresponding to 19 is for the keyword "this". Similarly other values in the first rows are computed corresponding to other keywords. Let the search query be "*This is a demonstration*". Now we demonstrate how BuildIndex algorithm generates a portion of server side index corresponding to this search query occurring in the document ToyExample.txt. Let the encrypted file identifier corresponding to ToyExample.txt be 1221. For the word 'this', first we select $r_1 = 10 \leftarrow \mathbb{Z}_p$. So the hash chain corresponding to the search query "*This is a demonstration*" is $\left(10, MAC_{k_s}(10), MAC^2_{k_s}(10), MAC^3_{k_s}(10)\right)$ and let this gives the sequence $(10, 21, 5, 16)$. So when masking is applied to this sequence, we get $(10 + MAC_{k'}(this), 21 + MAC_{k'}(is), 5 + MAC_{k'}(a),$ $16 + MAC_{k'}(demonstration))$ and let this gives the sequence $(2, 21, 30, 4)$. Table III presents the server side index without cell-padding (see the paragraph "Structure of index" in Section IV).

TABLE III
SERVER INDEX TABLE.

.

	19	26	12	7	29
1221	2	21	30	4	7

Remark 2. *In [15], authors have proposed a SSE scheme for string search, which is similar to the non-adaptive SSE scheme of [12] for keyword search with some additional data structures and techniques (list, lookup tables, pseudo random functions and hash-chains for word sequencing) being used to keep track of position informations. However, with this approach, server learns the word frequency and relative positions of the underlying document. The BuildIndex algorithm (see Algorithm 2) is based on a new approach of inverted index generation in modulo prime field. As opposed to the unmasked hash-chains used in [15] for all words in a document, we use masked hash-chain and cell-padding which stops leakage of informations related to the relative positions of sentences and the frequency of words. As opposed to the idea of chain of encryptions in [12], we introduce the idea of masking for the security of index which is faster. In earlier schemes, to search for a word in n documents, n decryption operations were needed. In our scheme, all we need is unmasking which is a subtraction operation in $\mathbb{Z}_p$ for all entries of the corresponding column.*

C. Trapdoor Generation

Example 1 (continued): (Illustration of Trapdoor)
Let the search query be "*This is a demonstration*". Let us consider the computation of $Trapdoor(this) = (t_1, t_2, t_3)$ as follows:
let $Enc_{k_m}(this) = e = 20$ and $MAC_{k'}(this) = msk = 24$; We know $MAC_{k_m}(this) = ci = 18$. Now, Let $t_1 = MAC_{k_s}(e \oplus msk \oplus ci) = MAC_{k_s}(62) = 14$. Also let $t_2 = e \otimes ci = 20 \times 18 = 19$. Lastly, let $t_3 = e \otimes msk = 20 \times 24 = 15$. Thus $Trapdoor(this) = (t_1, t_2, t_3) = (14, 19, 15)$. Similarly we compute trapdoors for the key words 'is', 'a' and 'demonstration'.

Algorithm 3 Trapdoor

```
Input $w = (w_{c_1}, w_{c_2}, \ldots, w_{c_l})$, $k_m$, $k'$ $k_s$, $p$, $MAC_k(.)$.
Output $t = (t_1, \ldots, t_l)$.
  $j \leftarrow 1$;
  while $j \leq l$ do
    $e = Enc_{k_m}(w_{c_j})$;
    $msk = MAC_{k'}(w_{c_j})$;
    $ci = MAC_{k_m}(w_{c_j})$;
    $t_{j_1} = MAC_{k_s}(e \oplus msk \oplus ci)$;
    $t_{j_2} = e \otimes ci$;
    $t_{j_3} = e \otimes msk$;
    $t_j = (t_{j_1}, t_{j_2}, t_{j_3})$;
    $j \leftarrow j + 1$;
  end while
```

Remark 3. *Although the index table is static, i.e, created once at the beginning, but for every instance of search we compute probabilistic trapdoors and using these, we end up searching successfully from the index table. It may be noted that the probabilistic algorithm used in the BuildIndex is only for obtaining encrypted file pointers. Once the keys are fixed, $I_r[j]$'s and the entries in $I[i][j]$'s before cell-padding are deterministic. Let $t = (t_{j_1}, t_{j_2}, t_{j_3})$ be the trapdoor corresponding to the word w_j and $I_r[j]$ contains the corresponding inverted client index. So $I_r[j] = ci^{-1}$ in $\mathbb{Z}_p$, where $ci = MAC_{k_m}(w_j)$. Note that $t_{j_2} = e \times ci$, where e is the random part which is obtained only when t_{j_2} is multiplied with $I_r[j]$. Once e is obtained, msk can be obtained as $msk = t_{j_3} \times e^{-1}$. The correctness is asserted from the check $(MAC_{k_s}(e \oplus msk \oplus ci) == t_{j_1})$.*

D. Searching

Algorithm 4 Search

```
Input $t = (t_1, \ldots, t_l)$, SI, $k_s$, $MAC_k(.)$.
Output $encrypted\_file\_pointers$, a list of encrypted document pointers;
  the list $column$ and $column\_msk$ are set empty;
  $i \leftarrow 1$;
  while $i \leq l$ do
    $j \leftarrow 1$;
    while $j \leq d$ do
      $e = (t_{i_2} \otimes I_r[j])$;
      $m = t_{i_3} \otimes e^{-1}$;
      if $(MAC_{k_s}(e \oplus m \oplus I_r[j]^{-1}) == t_{i_1})$ then
        set mask of $j$ th column as $msk = m$, add $j$ to $column$ and $msk$ to $column\_msk$;
      end if
      $j \leftarrow j + 1$;
    end while
    $i \leftarrow i + 1$;
  end while
  the list $encrypted\_file\_pointers$ is set empty;
  $i \leftarrow 1$;
  while $i \leq n$ do
    if (there exists $l$ integers $p_1, \ldots, p_l$ such that $(p_j \oplus column\_msk[j]) \in I[i][column[j]]$, $1 \leq j \leq l$ and $MAC_{k_s}(p_j) == p_{j+1}$ for $1 \leq j \leq l - 1$) then
      add $I_c[i]$ to $encrypted\_file\_pointers$.
    end if
    $i \leftarrow i + 1$;
  end while
```

Example 1 (continued): (Illustration of Searching)
After receiving the trapdoor for the word 'this', server identifies the column in the server index table corresponding to the word 'this' in the following way:
It may be noted that when the column index is 1, we have $(t_2 \otimes I_r[1]) = 19 \otimes 19 = 20 = e$. Also $t_{i_3} \otimes e^{-1} = 24 = msk$. Thus $MAC_{k_s}\left(e \oplus msk \oplus I_r[j]^{-1}\right) = MAC_{k_s}(62) = 14 = t_1$. Thus the column corresponding to index 1 is identified for the trapdoor $(14, 19, 15)$ which was sent for the keyword 'this'. Similarly the column indexes are identified for the other keywords which are 2, 3 and 4 for the words 'is', 'a' and 'demonstration' respectively. Note that server can retrieve mask corresponding to keyword 'this' which is $msk = 24 = t_{i_3} \otimes e^{-1}$. Similarly server can compute masks for other words as well. So from the sequence obtained from the server index table, i.e., $(2, 21, 30, 4)$, server can compute the sequence $(10, 21, 5, 16)$ by subtracting corresponding masks. Now server forms the hash chain $(10, 21, 5, 16)$ = $\left(10, Mac_{k_s}(10), Mac^2_{k_s}(10), Mac^3_{k_s}(10)\right)$ which tells that this string exists in the document whose encrypted identifier is 1221. Server returns this to client which on decryption yield ToyExample.txt, i.e. the filename containing the queried string.

Remark 4. *In Algorithm 4, the input $(t_1, \ldots, t_l)$ corresponds to the l-word search string. In the index table, there are $|\delta(D)|$ number of columns, out of which the l columns corresponding to $t_1, \ldots, t_l$ are to be identified and also the corresponding column masks are to be computed for searching. This is done in the first while loop of the search algorithm (Algorithm 4). It may be noted that the column indexes corresponding to $t_1, \ldots, t_l$ are stored in a list variable, named, $column$ and the corresponding masks are stored in a list variable, named, $column\_msk$. To check for the existence of the string in a file corresponding to, say, i-th row of the index table, the second while loop of the algorithm unmasks the entries corresponding to $I[i][column[1]], \ldots, I[i][column[l]]$ and checks if there exists a matching pair (p_1, p_2), where $p_1 \in I[i][column[1]]$ and $p_2 \in I[i][column[2]]$ such that $p_2 = MAC_{k_s}(p_1)$. This process is repeated for $(l-1)$ times for all $(column[i], column[i+1])$ pairs $(1 \leq i \leq l-1)$ provided such matching pairs (p_1, p_{i+1}) are obtained at each of the steps such that $p_{i+1} = MAC_{k_s}(p_i)$. If $(l-1)$ steps are completed successfully, then that confirms the existence of the string in the file and the corresponding file identifier is added in the list named $encrypted\_file\_pointers$.*

In this section, we described the algorithms for key generation, index generation, trapdoor generation and searching for our scheme Π_{ss}. For better security, probabilistic trapdoor is desirable. Since the index is static, searching using probabilistic trapdoor and static index is difficult. Unlike previous schemes, our scheme produces probabilistic trapdoors and search algorithm accurately detects the string despite having static index. The correctness of this computation is explained in Remark 3. In the next section we provide the security of our scheme Π_{ss}.

V. Security Analysis

Although few SSE schemes are available for string search, but most of them lack formal security proof against the revised definitions of [12] and also leak lots of informations beyond what is leaked from *history* [12]. In Subsection V-A, we provide a formal proof of non-adaptive security of our scheme against *honest-but-curious server*. In Subsection V-B, we propose modifications of our scheme to protect against *active* adversaries at the cost of more rounds of communications and memory space.

A. Honest but curious server

An *honest-but-curious server* follows the protocol and takes no actions beyond those of an honest server, and attempts to learn about the plaintext of documents or terms that were queried. The idea of *non-adaptive* security for SSE scheme for an *honest-but-curious* server was first introduced in [12]. In order to explain the non-adaptive security, we first provide the definition of *history* and *trace*.

Definition 3. *(history) [12] Let $\triangle$ be a dictionary and $D \subseteq \mathbb{P}(\triangle)$ be a document collection over $\triangle$. A q-query history over D is a tuple $H = (D, w)$ that includes the document collection D and a vector of q keywords $w = (w_1, w_2, \ldots, w_q)$.*

The access pattern induced by a q-query *history* $H = (D, w)$ is given by $\alpha(H) = (D(w_1), \ldots, D(w_q))$ [12]. The search pattern corresponding to q-query *history* H is a $q \times q$ binary matrix $\sigma(H) = (h_{i,j})$ such that $h_{i,j} = 1$ if $w_i = w_j$ [12].

Definition 4. *(trace) [12] Let $\triangle$ be a dictionary and $D \subseteq \mathbb{P}(\triangle)$ be a document collection over $\triangle$. The trace induced by a q-query history $H = (D, w)$ is a sequence $\tau(H) = (|D_1|, |D_2|, \ldots, |D_n|, \alpha(H), \sigma(H))$ comprised of the lengths of the documents in D.*

We introduce the notion of *history-of-string* by extending the definition of *history* for string which is crucial for our security proof.

Definition 5. *(history-of-string) Let $\triangle$ be a dictionary and $D \subseteq \mathbb{P}(\triangle)$ be a document collection over $\triangle$. A q-query history over D is a tuple $\hat{H} = (D, s)$ that includes the document collection D and a vector of q strings $s = (s_1, s_2, \ldots, s_q)$.*

Let for the vector of q strings $s = (s_1, \ldots, s_q)$, there are only q' distinct words, i.e., $|\delta(s)| = q'$ and let these distinct words be $w_1, \ldots, w_{q'}$. The access pattern induced by a q-query *history-of-string* $\hat{H} = (D, s)$ is given by $\alpha(\hat{H}) = (D(w_1), \ldots, D(w_{q'}))$ [12]. The search pattern corresponding to such a q-query *history-of-string* $\hat{H}$ is a $q' \times q'$ binary matrix $\sigma(\hat{H}) = (h_{i,j})$ such that $h_{i,j} = 1$ if $w_i = w_j$ [12]. Similar to *trace*, we define *trace-of-history-of-strings* as follows:

Definition 6. *(trace-of-history-of-strings) Let $\triangle$ be a dictionary and $D \subseteq \mathbb{P}(\triangle)$ be a document collection over $\triangle$. The trace induced by a q-query history-of-strings $\hat{H} = (D, s)$ is a sequence $\tau(\hat{H}) = \Big(|D_1|, |D_2|, \ldots, |D_n|, \alpha(\hat{H}), \sigma(\hat{H})\Big)$ comprised of the lengths of the documents in D.*

Lemma 1. *Let for the vector of q strings $s = (s_{0,1}, \ldots, s_{0,q})$, there are only q' distinct words, i.e., $|\delta(s)| = q'$ and let these distinct words be $w_{0,1}, \ldots, w_{0,q'}$. Let $\hat{H}_0 = (D_0, s_{0,1}, \ldots, s_{0,q})$ and also let $H_0 = (D_0, w_{0,1}, \ldots, w_{0,q'})$. Similarly define $\hat{H}_1 = (D_1, s_{1,1}, \ldots, s_{1,q})$ and $H_1 = (D_1, w_{1,1}, \ldots, w_{1,q'})$. It is easy to check that $\tau(\hat{H}_0) = \tau(\hat{H}_1)$ implies $\tau(H_0) = \tau(H_1)$.*

Now we provide the definition of *non-adaptive indistinguishability security* and *non-adaptive semantic security* for SSE from [12] with slight modifications for strings.

Definition 7. *(Non-Adaptive Indistinguishability Security for SSE) Let $SSE = (Gen, Enc, Trpdr, Search, Dec)$ be an index based SSE scheme over dictionary $\triangle$, λ being the security parameter, and $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ be a non-uniform adversary. Consider the* *probabilistic experiment*

```
$Ind_{\mathcal{A},SSE}(\lambda)$ :
    $K \leftarrow Gen(1^{\lambda})$
    $(\mathcal{ST_A}, \hat{H}_0, \hat{H}_1) \leftarrow \mathcal{A}_1(1^{\lambda})$
    $b \leftarrow \{0,1\}$
    $parse\ \hat{H}_b\ as\ (D_b, s_b)$
    $(SI_b, c_b) \leftarrow Enc_K(D_b)$
    $for\ 2 \leq i \leq q$
        $t_{b,i} \leftarrow Trpdr_{k_s}(s_{b,i})$
    $let\ t_b = (t_{b,1}, \ldots, t_{b,q})$
    $b' = \mathcal{A}_{q+1}(\mathcal{ST_A}, SI_b, c_b, t_b)$
    $output\ 1\ if\ b' = b\ else\ output\ 0$
```

with the restriction that $\tau(D_0, s_{0,1}, \ldots, s_{0,q}) = \tau(D_1, s_{1,1}, \ldots, s_{1,q})$, $\mathcal{ST_A}$ *being a string that captures* $\mathcal{A}_1$*'s state.* SSE *is said to be secure in the sense of non-adaptive indistinguishability if for all polynomial size adversaries* $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$, $Pr[Ind_{SSE,\mathcal{A}}(\lambda) = 1] \leq \frac{1}{2} + negl(\lambda)$, *where probability is taken over the choice of* b *and coins of Gen and Enc.*

Remark 5. *Let for the vector of* q *strings* $s = (s_{0,1}, \ldots, s_{0,q})$ *in the definition of non-adaptive indistinguishability, there are only* q' *distinct words, i.e.,* $|\delta(s)| = q'$ *and let these distinct words be* $w_{0,1}, \ldots, w_{0,q'}$. *Let* $\hat{H}_0 = (D_0, s_{0,1}, \ldots, s_{0,q})$ *and also let* $H_0 = (D_0, w_{0,1}, \ldots, w_{0,q'})$. *Similarly define* $\hat{H}_1 = (D_1, s_{1,1}, \ldots, s_{1,q})$ *and* $H_1 = (D_1, w_{1,1}, \ldots, w_{1,q'})$. *From Lemma 1,* $\tau(\hat{H}_0) = \tau(\hat{H}_1)$ *implies* $\tau(H_0) = \tau(H_1)$. *Thus the non-adaptive indistinguishability definition of SSE for string search, i.e., Definition 7 implies non-adaptive indistinguishability of SSE for keyword search introduced by [12].*

Definition 8. *(Non-Adaptive Semantic Security for SSE) Let* $SSE = (Gen, Enc, Trpdr, Search, Dec)$ *be an indexed based SSE scheme over dictionary* $\triangle$, λ *being the security parameter, and* $\mathcal{A}$ *be an adversary,* $\mathcal{S}$ *being the simulator. Consider the probabilistic experiments*

```
$Real_{SSE,\mathcal{A}}(\lambda)$ :
    $K \leftarrow Gen(1^{\lambda})$
    $(\mathcal{ST_A}, \hat{H}) \leftarrow \mathcal{A}_1(1^{\lambda})$
    $parse\ \hat{H}\ as\ (D, s)$
    $(SI, c) \leftarrow Enc_K(D)$
    $for\ 1 \leq i \leq q$
        $t_i \leftarrow Trpdr_{k_s}(s_i)$
    $let\ t = (t_1, \ldots, t_q)$
    $output\ v = (SI, c, t)\ and\ \mathcal{ST_A}$
```

```
$Sim_{SSE,\mathcal{A},\mathcal{S}}(\lambda)$ :
    $(\hat{H}, \mathcal{ST_A}) \leftarrow \mathcal{A}(1^{\lambda})$
    $v \leftarrow \mathcal{S}(\tau(\hat{H}))$
    $output\ v = (SI, c, t)\ and\ \mathcal{ST_A}$
```

SSE *is said to be semantically secure if for all polynomial size adversaries* $\mathcal{A}$, *there exists a polynomial size simulator* $\mathcal{S}$, *such that for all polynomial size distinguishers* $\mathcal{D}$, $Pr[\mathcal{D}(v, \mathcal{ST_A}) = 1 : (v, \mathcal{ST_A}) \leftarrow Real_{SSE,\mathcal{A}}(\lambda)] - Pr[D(v, \mathcal{ST_A}) = 1 : (v, \mathcal{ST_A}) \leftarrow Sim_{SSE,\mathcal{A},\mathcal{S}}(\lambda)] \leq negl(\lambda)$, *where probability is taken over the choice of* b *and coins of Gen and Enc.*

Here we recall one result from [12] which we use in proving non-adaptive security of our scheme.

Theorem 1. *Non-adaptive semantic security of SSE implies non-adaptive indistinguishability of SSE [12].*

Remark 6. *Let* max *be the number of times the smallest keyword can be fitted into the largest of all* n *encrypted documents. We observe that* max *is an upper bound of maximum number of times a particular keyword may occur in a document i.e.* f *(see Section III). We use this* max *to simulate* f.

Theorem 2. Π_{ss} *is non-adaptively secure SSE scheme.*

Proof: To prove non-adaptive indistinguishability of Π_{ss}, from Theorem 1, it is sufficient to show that Π_{ss} is non-adaptively semantically secure. To show that we first describe a polynomial size simulator S such that for all polynomial size adversaries $\mathcal{A}$, the output of $Real_{SSE,A}(\lambda)$ and $Sim_{SSE,A,S}(\lambda)$ are indistinguishable. Let the simulator S on receipt of $\tau(\hat{H})$ generates $v^* = (SI^*, c^*, t^*) = ((I^*, I_r^*, I_c^*), (t_1^*, \ldots, t_q^*), (c_1^*, \ldots, c_n^*))$ as follows:

1. (simulating SI^*)
   a. If $q > 0$, set $|\delta(D)| = q$. Choose uniformly at random the keys k_s^* and k_m^*. From the trace, find all distinct words that occur in q strings. Let this collection be $\{w_1, \ldots, w_{q'}\}$. Generate a $n \times q'$ array I^*. Assign $I_r^*[i] = MAC_{k_m^*}(w_i)^{-1}$ mod p. Also compute $I_c^*[j] \leftarrow Enc_{k_m^*}(id(D_j))$ for $j = 1$ to n.
   b. For each string $s_j = (w_{j,1}, \ldots, w_{j,l})$ and document D_i containing the string, find the masked *hash-chain* $(r_1, \ldots, r_l)$ as mentioned in Section IV.
   c. For each $I^*[i][j]$ such that $|I^*[i][j]| < max$, run $MAC_{k_s^*}()$ $(max - |I^*[i][j]|)$ many times on different random values from $\mathbb{Z}_p$ and store the outputs at $I^*[i][j]$.
   d. If $q = 0$, we allocate $n \times max$ (see Remark 6) array for I and fill up the corresponding cells with max number of different random values from $\mathbb{Z}_p$.
2. (simulating $t^* = (t_1^*, \ldots, t_q^*)$) Compute t_i^* corresponding to the string $s_i = (w_{i,1}, \ldots, w_{i,l_i})$ as $t_i^* = (t_{i,1}^*, \ldots, t_{i,l_i}^*)$, where $t_{i,j}^* = (t_{i,j,1}^*, t_{i,j,2}^*, t_{i,j,3}^*)$ corresponding to the word say w_j, is computed as follows:
$$\begin{aligned} &e = Enc_{k_m^*}(w_j), msk = MAC_{k'^*}(w_j), \\ &ci = MAC_{k_m^*}(w_j), \\ &t_{i,j,1}^* = MAC_{k_s^*}(e \oplus msk \oplus ci), \\ &t_{i,j,2}^* = e \otimes ci, \quad t_{i,j,3}^* = e \otimes msk. \end{aligned}$$
3. (simulating $c^* = (c_1^*, \ldots, c_n^*)$) Set c_i^* to a $|D_i|$ bit string chosen uniformly at random.

Now we show that each element of v^* is computationally indistinguishable from its corresponding element of v to a distinguisher $\mathcal{D}$ that is given $\mathcal{ST_A}$.

1. (**SI** and **SI***) $I_r^*[i] = MAC_{k_m^*}(w_i)^{-1}$ mod p. Since $MAC_{k_m^*}(w_i)$ and $MAC_{k_m}(w_i)$ are indistinguishable, so is I_r^* and I_r. As with all but negligible probability, $\mathcal{ST_A}$ does not include the key k_m and π, the encryption scheme, is IND-CPA secure, this guarantees that each element of I_c is indistinguishable from its counter part in I_c^*. Due to the pseudo randomness of $MAC()$ and all but negligible probability, $\mathcal{ST_A}$ does not include the key k_m, we argue that each of the max number

of entries in $I[i][j]$ is indistinguishable from its counterpart in $I^*[i][j]$.

2. (t and t^*) Since computation of t involves computation using $MAC()$ and $Enc()$ and with all but negligible probability, $\mathcal{ST}_{\mathcal{A}}$ does not include the key (k_m, k', k_s) so from the pseudo-randomness of $MAC()$ and IND-CPA security of π, t and t^* are indistinguishable.
3. (c and c^*) As with all but negligible probability, $\mathcal{ST}_{\mathcal{A}}$ does not include the key (k_s, k_m) and π is IND-CPA secure, this guarantees that each element of c is indistinguishable from its counter part in c^*.

■

The non-adaptive indistinguishability guarantees the privacy of the keyword from the index. However, it does not guarantee any security against leakage of keyword from trapdoor. Towards this we introduce the notion of *search pattern privacy*.

Definition 9. *(Search Pattern Indistinguishability) Let* $SSE = (Gen, Enc, Trpdr, Search, Dec)$ *be an index based SSE scheme over dictionary* $\triangle$, λ *being the security parameter, and* $\mathcal{A}$ *be a non-uniform adversary. Consider the probabilistic experiment:*

```
$SPI_{SSE,\mathcal{A}}(\lambda)$ :
        $K \leftarrow Gen(1^\lambda); (\mathcal{ST}_{\mathcal{A}}, H) \leftarrow \mathcal{A}(1^\lambda)$
        $parse\ H\ as\ (D, s)$
        $(\mathbf{SI}, c) \leftarrow Enc_K(\mathbf{D})$
        $let\ w = (w_{b_1}, \ldots, w_{b_q})$
        $for\ 1 \leq i \leq q$
            $t_{b_i} \leftarrow Trpdr(w_{b_i})$
        $let\ t = (t_{b_1}, \ldots, t_{b_q})$
        $w_0, w_1 \leftarrow \triangle\ such\ that\ |w_0| = |w_1|$
        $b \leftarrow \{0, 1\}$
        $t_b \leftarrow Trpdr(w_b)$
        $b' \leftarrow \mathcal{A}^{Trpdr(K;.)}(1^\lambda, K, t_b, w, t, H, \mathcal{ST}_{\mathcal{A}})$
        $if\ b = b'\ then\ return\ 1\ else\ return\ 0$
```

The advantage of $\mathcal{A}$ *in the above experiment is defined as* $Adv^{SPP}_{\mathcal{A},SSE}(\lambda) = |Pr[SPI_{SSE,\mathcal{A}}(\lambda) = 1] - \frac{1}{2}|$. *SSE is said to be secure against search pattern indistinguishability if for all polynomial size adversaries* $\mathcal{A}$, $Adv^{SPP}_{\mathcal{A},SSE}(\lambda) \leq negl(\lambda)$.

Theorem 3. Π_{ss} *is search pattern secure scheme against chosen trapdoor attack in random oracle model.*

Proof: We construct a simulator $\mathcal{S}^{Enc_k(\cdot)}$, that has the encryption oracle for underlying private key encryption which is assumed to be IND-CPA secure under the Definition 2. The simulator $\mathcal{S}$ simulates the challenger and interacts with the adversary $\mathcal{A}$ as follows:

1. (simulating msk^* and ci^*) : Whenever $\mathcal{A}$ queries for trapdoor corresponding to some word, say w_i, $\mathcal{S}$ maintains two lists, namely $list_1 = \langle w_i, h_i^1 \rangle$ and $list_2 = \langle w_i, h_i^2 \rangle$ for msk and ci respectively which are initially empty. When $\mathcal{A}$ queries for trapdoor of w , $\mathcal{S}$ responds as follows:
   if w_i is in $list_1$ then it sets $msk = h_i^1$, else it choses $h_i^1 \leftarrow \{0,1\}^\lambda$. It makes an entry $\langle w_i, h_i^1 \rangle$ in $list_1$ and sets $msk^* = h_i^1$. Similarly it simulates $ci^* = h_i^2$.
2. (simulating $t_{i_1}, t_{i_1}, t_{i_1}$) : $\mathcal{S}$ asks the actual challenger to compute $e = Enc_{k_m}(w)$ (to $\mathcal{S}$ this is the encryption oracle). It then performs the following computations:
   $t_{i_2} = e \otimes ci^*$; $t_{i_3} = e \otimes msk^*$; $t_j = (t_{j_1}, t_{j_2}, t_{j_3})$; It sets $t_{i_1} = h$ where h is an entry corresponding to $e \oplus msk^* \oplus ci^*$ in $list_1$.

challenge phase : $\mathcal{A}$ produces a pair of challenge words w_0 and w_1.

I. $\mathcal{S}$ computes msk_i^* and ci_i^* for $i = 0, 1$.
II. $\mathcal{S}$ randomly selects $b \leftarrow \{0, 1\}$.
III. $\mathcal{S}$ responds to the challenge $t = (t_1, t_2, t_3)$ where $e_b = Enc_{k_m}(w_b)$ (using actual challenger) $t_2 = e_b \otimes ci_b^*$; $t_3 = e \otimes msk_b^*$; t_1, msk_b^* are taken from $list_1$ and ci_b^* from $list_2$ as mentioned earlier.
IV. A can continue to issue trapdoor queries for words.

Output phase : Eventually $\mathcal{A}$ outputs the guess $b'\{0, 1\}$. Then $\mathcal{S}$ outputs b'.

So $\mathcal{A}$ and $\mathcal{S}$'s point of view to the event $[b = b']$ is same, i.e., $Pr[b = b'] = Pr[SPI_{SSE,\mathcal{A}}(\lambda) = 1] = Pr[PrivK^{cpa}_{\mathcal{S},\pi}(\lambda) = 1]$.

But from the assumption, $Pr[PrivK^{cpa}_{\mathcal{S},\pi}(\lambda) = 1] \leq \frac{1}{2} + negl(\lambda)$. So $Pr[SPI_{SSE,\mathcal{A}}(\lambda) = 1] \leq \frac{1}{2} + negl(\lambda)$. Thus $Adv^{SPP}_{\mathcal{A},SSE}(\lambda) = |Pr[SPI_{SSE,\mathcal{A}}(\lambda) = 1] - \frac{1}{2}| \leq negl(\lambda)$. ■

Remark 7. *It may be noted that if adversary is given the search oracle, it can distinguish the two trapdoors after the search. This is just because adversary can include* w_0 *and* w_1 *in the previous queries to get legitimate trapdoors and may detect the columns of index table corresponding to these words by performing search. Finally, by searching with challenge trapdoor* t_b*, adversary can know to which column and therefore to which word this trapdoor is associated with. So long as we are dealing with honest-but-curious adversary, we need not to assume that adversary may search. Server can always search and maintain a history, which is unavoidable. In the next section, we deal with adversaries who can make server search.*

Remark 8. *It may be noted that the trapdoor function introduced in [12] was deterministic. To be more particular, for a keyword, say* w*, their trapdoor function is* $T(w) = (\pi_z(w), f_y(w))$*, where* π *is a pseudo random permutations and* f *is a pseudo random function. So, from the properties of* π *and* f*, given* $T(w)$*, it is computationally hard to get back* w*. But, since* π *and* f *are deterministic,* $T(w)$ *is strictly connected to* w*, i.e., each time the trapdoor function will yield same footprint for the input* w*, which stops it from being search pattern secure (see Definition 9).*

Remark 9. *While dealing with string search, designing a SSE scheme satisfying* adaptive-indistinguishability-security *definition of [12] seems impossible. This is because to generate an index in advance which is consistent with future search, one unavoidable assumption needed is the presence of all possible strings in each document. This can be done by considering all permutations of keywords for every document. More specifically, we observe that the consistent index generation for string search can be done by allowing* $(max)!$ *number of entries of each keyword and each document, where* max *is the estimated number of distinct keywords. But simulating such an index* **SI*** *using* Sim *comes with a cost of exponential size in* max *compared to the index that may be generated using* $Real$ *and thus* **SI*** *can be easily distinguished from* **SI**.

B. Active Adversary

Injection attacks refer to a broad class of attack vectors that allow an attacker to supply untrusted input to a program. In [8], authors refer to these attackers as *active adversary*. More precisely, an *active adversary* is one who can carryout a chosen-document attack in which it tricks the client into including a chosen document in the document set. We call it a spam document and denote it by D_s. Since server knows the contents of the spam document, whenever server detects the access pattern that involves the access of D_s, server may guess with significant probability $(\frac{1}{\delta(D_s)})$ the words corresponding to these columns of I that were being selected during the search by maintaining a book-keeping of columns being selected after every search.

There can be two cases. Firstly, server may manage to insert D_s into clients document collections $\boldsymbol{D}$ before the client's processing of $\boldsymbol{D}$ to generate $(\boldsymbol{SI}, c)$. To address such cases of spam detection is beyond the scope of this work and may be dealt by enabling spam filter. In the second case the client prepares and uploads $(\boldsymbol{SI}, c)$ to a server depending on $\boldsymbol{D}$ which does not contain spam document. Client updates the document collections $\boldsymbol{D}$ and corresponding $(\boldsymbol{SI}, c)$ whenever new documents are added or deleted. Server, after some time, manages to inject the spam document D_s without the knowledge of client. In this case we propose how Π_{ss} can be modified to resist such chosen document attack but at the cost of increased rounds of search.

Modification of I : Let initially there were n documents in $\boldsymbol{D}$ and $\boldsymbol{D} = \{D_1, \ldots, D_n\}$ and also let $\delta(\boldsymbol{D}) = \{w_1, w_2, \ldots, w_d\}$. Let after a modification of $\boldsymbol{D}$ now there are $n' = n + 1$ number of documents and the newly added document is a spam document. We denote this modified document collection by $\boldsymbol{D'}$. Let $W = \delta(\boldsymbol{D'}) \cap \delta(\boldsymbol{D})$ be the set of common words. It can be observed that if the search query string consists words that are from $\delta(\boldsymbol{D}) \setminus W$, then access pattern will not involve newly added document. Also if the search string involves words that are only from $\boldsymbol{D'} \setminus W$, access pattern does not involve previous documents. An access pattern involving new and old documents is possible if and only if all the words in the search string are from W. Note that client can always compute W if it maintains list of words say ***CI***. For each of the words in $w \in W$ client treats these words as $w||1$ to distinguish them from w which already exists in ***CI***. Next, client adds all words of the form $w||1$ and words from $\delta(\boldsymbol{D})' \setminus W$ in ***CI*** and adds new columns in I for all newly added words. Client adds a new row in I for the newly added document and computes entries in each cell of the newly added row same as before.

Two round communication for search : To search for a string $(w_1, \ldots, w_l)$, such that all w_i's are from W, client prepares another search string namely $(w_1||1, \ldots, w_l||1)$ and performs two searches for $(w_1, \ldots, w_l)$ and $(w_1||1, \ldots, w_l||1)$, one by one. With this approach the access pattern does not mix old documents with the newly added one.

VI. Performance Analysis

In this section we provide performance results of our proposed scheme Π_{ss}. First we study the theoretical search time and index size needed in Lemma 2 and Lemma 3 respectively.

Lemma 2. *Using Π_{ss}, the search time for a query of l-word string is $O(|\delta(\boldsymbol{D})| \times l \times n)$ operations in $\mathbb{Z}_p$, where the underlying document collection is $\boldsymbol{D}$ having n documents.*

Proof: Recall that in the index table there are $|\delta(\boldsymbol{D})|$ number of columns corresponding to $|\delta(\boldsymbol{D})|$ distinct keywords. Since Π_{ss} uses non-deterministic trapdoor, searching for column corresponding to each keyword takes $O(|\delta(\boldsymbol{D})|)$ number of operations in $\mathbb{Z}_p$. After l columns are found, for each document, $MAC()$ may be run to check for the adjacency of words l times, each of which takes constant time $O(1)$. Hence the result follows. ■

Lemma 3. *The index size needed for implementing Π_{ss} is $O(n \times |\delta(\boldsymbol{D})| \times f \times \lambda)$.*

Proof: Index is a $n \times |\delta(\boldsymbol{D})|$ array where each cell contains f number of entries from $\mathbb{Z}_p$, each being λ bit string as p is a λ bit prime. ■

Will parallelization improve the performance? We note that, by the construction of index, a row of the index, which corresponds to a document in the document collection, is independent of other rows and thus processing with one row has no overlap with that involving other rows. This crucial property enables the searching algorithm (Algorithm 4) conformable for parallelization. Currently the searching algorithm is linear in n, the number of documents (see Lemma 2). On parallelization of searching into t number of threads, each assigned with $n' = \frac{n}{t}$ rows, the searching will be linear in n' and since each of these threads will run in parallel, from Lemma 2 the complexity will be $O(|\delta(\boldsymbol{D})| \times l \times n')$ resulting into huge performance gain, specially in high load environments..

Now in the following subsections, we provide comparison of our scheme with schemes in [18] and [23] in terms of storage and sever side computation in the setup of string search. *It may be noted that unlike these schemes, we don't need any storage of data owner. Moreover, unlike our scheme, none of these schemes are proven to be secure under the newly introduced definitions of non-adaptive security of [12].*

A. Comparison with bloom-filter based scheme of [18]

In [18], authors proposed phrase search using *bloom-filters*. We start with converting their results related to complexity analysis in terms of our parameters for making it conformable for the comparison. In [18], authors used N, x, q, b_2, b_3 and k to denote number of documents, number of distinct keywords, number of words in the query, number of keyword pairs, number of keyword triplets and number of bloom filter hash functions respectively. Converting into our notation, we have $N \approx n$, $x \approx |\delta(D)|$ and $q \approx l$. Also $b_2 = O(|\delta(D)|^2)$ and $b_3 = O(|\delta(D)|^3)$.

Cloud side computation. In [18], authors showed that the computational cost of cloud server is $\#c = Mod(k(q-2)N) + And(Nb_3)$. Also from [18], to insert $|\delta(D)|$ keywords in a m-bit bloom-filter, the minimum *false-positive* probability p is achieved when $k = \frac{m}{|\delta(D)|}ln(2)$ and $m = -\frac{|\delta(D)|.ln(p)}{(ln(2))^2}$. Unlike the scheme of [18], in our scheme the probability of false positive rate is zero. So even when the scheme of [18] achieves minimum false positive rate, the cost of computation in asymptotic notation becomes

$$\begin{aligned}\#c &= Mod(k(q-2)N) + And(Nb_3)\\ &= O\left(\frac{m}{|\delta(D)|} \times l \times n\right) + O\left(n \times |\delta(D)|^3\right)\\ &= O\left(n \times |\delta(D)|^3\right)\end{aligned}$$

Since $|\delta(D)|^2 \gg l$, the scheme in [18] is more expensive in terms of cloud server computation.

Cloud side storage. In [18], authors showed that the storage at cloud server is $\#s$, where,

$$\begin{aligned}\#s &= x \times (log_2(x) + N) + N((b_2 + b_3)\\ &= O\left(|\delta(D)| \times (log_2(|\delta(D)|) + n) + n((|\delta(D)|^2 + |\delta(D)|^3)\right)\\ &= O\left(|\delta(D)| \times log_2(|\delta(D)|)\right) + O\left(n \times |\delta(D)|^3\right)\\ &= O\left(n \times |\delta(D)|^3\right)\end{aligned}$$

We observe that in the setup of string search where we consider all words as key words, $\lambda \times f \ll |\delta(D)|^2$. So the scheme in [18] incurs more storage in cloud compared to our scheme.

B. Comparison with TSet based scheme of [23]

In the TSet based approach of string search of [23], the computational cost of server side searching is exponential in $|t| \times (m-1)$

number of multiplication operations in $\mathbb{Z}_q$(see [23]), where $|t|$ is the number of outputs of TSet and m is the length of query and $log_2(q) = \lambda$ is the security parameter. Also from the TSet construction, which incorporates the combinations of keywords, $|t|$ itself is exponential on the distinct keywords, which, in our notation, is $|\delta(D)|$. Thus the cost of sever side searching is exponential in $|\delta(D)|$ which is linear in our case (see Lemma 2).

C. Experiment on Speech Data

From Lemma 2, it is clear that search time is linear in number of documents. Now we validate our scheme against two different commercial datasets ([1], [2]). We build a symmetric search engine based on the scheme Π_{ss} using Java in linux platform. The implementation is done on Asus A Series Core i3 laptop ((4 GB/ 1 TB HDD) 90NB0652-M32310 XX2064D). We use Java based 'Jpair' library for the cryptographic primitives.

The TIMIT speech corpus [TIMIT] of speech was collected in 1993 as a speech data resource for acoustic phonetic studies and has been used extensively for the development and evaluation of ASR studies. TIMIT contains broadband recordings of 630 speakers of eight major dialects of American English, each reading ten phonetically rich sentences. The corpus includes time-aligned orthographic, phonetic and word transcriptions as well as a 16-bit 16 kHz speech waveform file for each utterance. TIMIT was designed to further acoustic-phonetic knowledge and automatic speech recognition (ASR) systems. It was commissioned by DARPA and worked on by many sites, including Texas Instruments (TI) and Massachusetts Institute of Technology (MIT), hence the corpus' name.

Phone recognition in TIMIT has more than two decades of intense research behind it and its performance has naturally improved with time so that it can be useful for researchers, professionals and engineers specialized in speech processing when considering future research directions. TIMIT is the most accurately transcribed speech corpus in existence as it contains not only transcriptions of the text but also contains accurate timing of phones calls. This is impressive given that the average English speaker utters 14-15 phonic sounds (equivalent to phone symbols) a second. A more detailed description of the data may be obtained at [2].

D. Performance against Speech Dataset

The performances of string search on the speech data [2] are compared for two modes of implementations. The implementation of the scheme using masked *hash-chain* and padding using random values from $\mathbb{Z}_p$, f number of times as mentioned in Section IV is referred to as 'no leakage'. In another implementation we use masking over integers denoting word positions without padding. Here the frequency as well as the positions of the searched words in the documents are leaked, but after the search is done. So we refer to this as 'partial leakage'.

Figure 1 shows a comparison of index generation time for these two methods. Figure 2 shows a comparison of search time for these two methods. Search time mainly depends on the number of documents through which the search is to be performed. So far we have performed search on a string of length 5. So Figure 2 shows how search time varies with respect to the number of documents. Unlike [15], in our implementation, search time depends not only on the number of documents returned, but also on the number of documents in which the keywords to be searched are present. This is due to the fact that for efficient implementation, we first check if all keywords are present in a document. If all key words are present in a document then and only then we check for the adjacency. It may be noted that the major share of search time is taken by checking adjacency. So search time may also increase in these cases where all keywords are present in many documents, but not in a consecutive fashion, yielding zero number of documents to be returned.

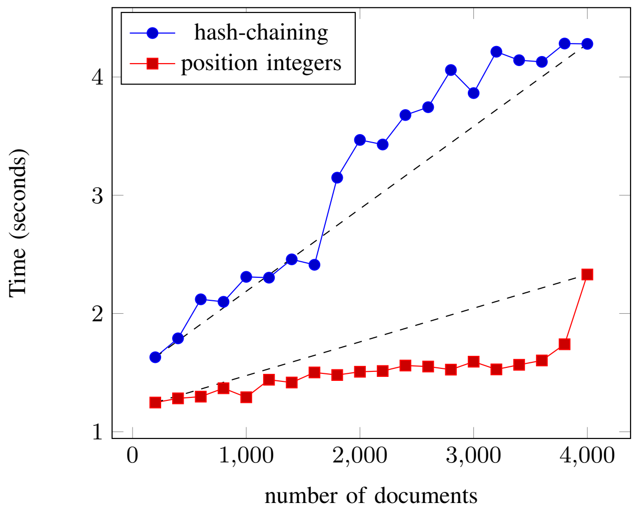

Fig. 1. Index generation graph comparing index generation time using padded and masked hash chain technique with index generation time using position integers

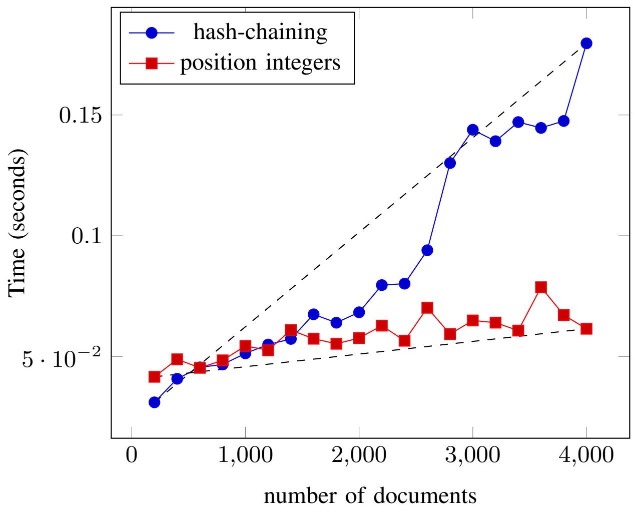

Fig. 2. Search time graph comparing the search time using hash chain technique with the search time using position integers

E. Experiment on DNA Data

We have implemented our algorithm for searching sequences of SNP's (Single Nucleotide Polymorphism) in large genome databases which is available at [1]. SNP represents a difference in a nucleotide which is a single DNA building block. In the data set there are 10000 binary DNA sequences, each of which is 2185-bit long. We assume that the bit pattern to be searched is of length at least 8 and encode the binary sequences by using a set, $\mathcal{K}$, of 2^8 i.e. 256 distinct symbols corresponding to every 8-bit of the data starting from the beginning. We treat each of these encoded sequences over $\mathcal{K}$ as different strings over $\mathcal{K}$. Search time in a document depends on the depth of the strings and the number of sequences. Since these sequences are very long compared to the speech data of Subsection VI-C, time taken for searching over this dataset is more compared to the time taken for searching over speech data. We apply our scheme π_{pss} on this data to search a string of length 3 in the encrypted domain, which is a same as searching 24-bit binary string in the binary sequence data. Due to huge size of the DNA data, in the implementation, we use directly integer pointers without masking and hash-chaining as otherwise the size of index is becoming larger than the available memory.

F. Performance against DNA Dataset

Figure 3 illustrates the time needed to generate index table for the DNA sequence, where we plot the time (in seconds along Y-axis) needed against number of encrypted DNA's (along X-axis). This graph also reflects a linear growth with the increase in number of DNA sequences. Figure 4 compares the performance of searching patterns over encrypted DNA sequences of length four over the set $\mathcal{K}$. So in binary it is a string of length 24. Here also we plotted time of search in seconds along Y-axis against the number of DNA sequences to be matched along X-axis. This graph also shows roughly a linear growth with the increase of number of encrypted sequences.

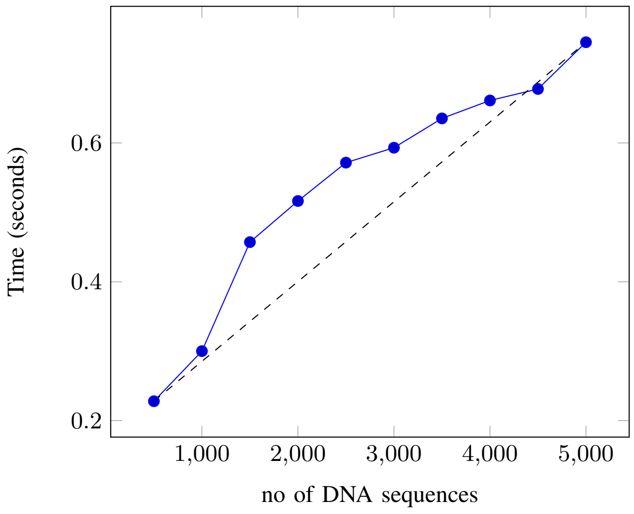

Fig. 3. Index generation graph for DNA sequences comparing index generation time using padded and masked hash chain technique with index generation time using position integers

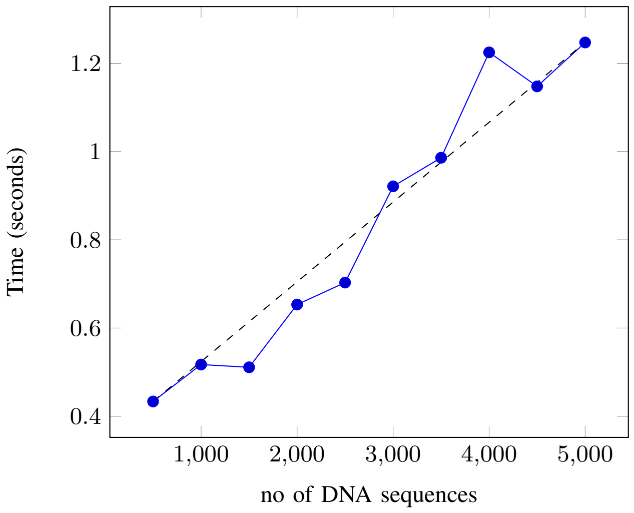

Fig. 4. Search time graph for encrypted DNA sequences comparing the search time using hash chain technique with the search time using position integers

VII. Conclusion

With the increasing number of documents stored in cloud, searching for the desired document can be a difficult and resource intensive task. One solution may be to use symmetric searchable encryption (SSE) which allows one party to outsource the storage of its data to another party (a cloud) privately while enabling to search selectively over it. In this paper we revisited the security definitions of [12] and proposed a new lightweight SSE scheme $\Pi_{s,s}$ for string search. We have shown that our scheme is secure under the *non-adaptive* indistinguishability definition [12]. For active adversary, we propose modification of the scheme $\Pi_{s,s}$ at the additional cost of memory at client's end and two rounds of communications for one modification of document collection. Towards this direction, future research can be performed to design efficient SSE scheme ideally with one round of communication. With our scheme, server does not learn the information related to word frequency and word positions except what it can learn from the history.

We, for the first time, introduce new security notion in SSE, named, *search pattern indistinguishability*. It may be observed that with non adaptive indistinguishability security, although the keywords are guaranteed to be secure from the possible leakage from index, however it does not guarantee the security from the possible leakage from trapdoor. Towards this, we for the first time introduce probabilistic trapdoor and prove that our scheme is secure under such criterion. We have implemented our scheme for the first time to search over phone symbols and validated it using the TIMIT dataset. We have also implemented our scheme over DNA data of [1] and successfully achieve pattern matching functionality over encrypted domain.

While dealing with string search, designing a SSE scheme satisfying *adaptive-indistinguishability-security* definition of [12] seems intuitively impossible. This is because to generate an index in advance which is consistent with future search, one unavoidable assumption needed is the presence of all possible strings in each document. This can be done by considering all permutations of keywords for every document which makes the index size exponential in n for n-document collection. According to the definition of [12], index size is linear in n which is essential from efficiency point of view. From the angle of this intuition, future research can be carried out to give a formal proof in support of non-existence of adaptively secure SSE scheme for string search. In this paper we have considered *honest-but-curious* adversaries and *active* adversaries. Also, designing SSE scheme for string search with *adaptive-indistinguishability-security* against some newly defined adversary can be a future research direction.

Acknowledgment

This research was partly supported by the Engineering and Physical Sciences Research (EPSRC) UK project Trusted Things & Communities: Understanding Enabling A Trusted ioT Ecosystem and European Commissions Horizon 2020 research and innovation project RED - Alert under grant agreement No 740688.

References

[1] https://github.com/iskana/pbwt-sec/tree/master/sample_dat.
[2] http://www.fon.hum.uva.nl/david/ma_ssp/2007/timit/train/dr5/fsdc0/.
[3] Michel Abdalla, Mihir Bellare, Dario Catalano, Eike Kiltz, Tadayoshi Kohno, Tanja Lange, John Malone-Lee, Gregory Neven, Pascal Paillier, and Haixia Shi. Searchable Encryption Revisited: Consistency Properties, Relation to Anonymous IBE, and Extensions. volume 21, pages 350–391. Springer, 2008.
[4] Mihir Bellare, Alexandra Boldyreva, and Adam ONeill. Deterministic and Efficiently Searchable Encryption. In *Annual International Cryptology Conference*, pages 535–552. Springer, 2007.

[5] Dan Boneh, Giovanni Di Crescenzo, Rafail Ostrovsky, and Giuseppe Persiano. Public Key Encryption With Keyword Search. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 506–522. Springer, 2004.

[6] Dan Boneh, Eyal Kushilevitz, Rafail Ostrovsky, and William E Skeith III. Public Key Encryption That Allows PIR Queries. In *Annual International Cryptology Conference*, pages 50–67. Springer, 2007.

[7] Ning Cao, Cong Wang, Ming Li, Kui Ren, and Wenjing Lou. Privacy-Preserving Multi-Keyword Ranked Search Over Encrypted Cloud Data. volume 25, pages 222–233. IEEE, 2014.

[8] David Cash, Paul Grubbs, Jason Perry, and Thomas Ristenpart. Leakage-Abuse Attacks Against Searchable Encryption. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 668–679. ACM, 2015.

[9] David Cash, Joseph Jaeger, Stanislaw Jarecki, Charanjit S Jutla, Hugo Krawczyk, Marcel-Catalin Rosu, and Michael Steiner. Dynamic Searchable Encryption in Very-Large Databases: Data Structures and Implementation. volume 2014, page 853. Citeseer, 2014.

[10] David Cash, Stanislaw Jarecki, Charanjit Jutla, Hugo Krawczyk, Marcel-Cătălin Roşu, and Michael Steiner. Highly-Scalable Searchable Symmetric Encryption With Support for Boolean Queries. In *Advances in Cryptology–CRYPTO 2013*, pages 353–373. Springer, 2013.

[11] David Cash and Stefano Tessaro. The Locality of Searchable Symmetric Encryption. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 351–368. Springer, 2014.

[12] Reza Curtmola, Juan Garay, Seny Kamara, and Rafail Ostrovsky. Searchable Symmetric Encryption: Improved Definitions and Efficient Constructions. volume 19, pages 895–934. IOS Press, 2011.

[13] Seny Kamara, Charalampos Papamanthou, and Tom Roeder. Dynamic Searchable Symmetric Encryption. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 965–976. ACM, 2012.

[14] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography*. CRC press, 2014.

[15] Mingchu Li, Wei Jia, Cheng Guo, Weifeng Sun, and Xing Tan. LPSSE: Lightweight Phrase Search With Symmetric Searchable Encryption in Cloud Storage. In *Information Technology-New Generations (ITNG), 2015 12th International Conference on*, pages 174–178. IEEE, 2015.

[16] Muhammad Naveed, Seny Kamara, and Charles V Wright. Inference Attacks on Property-Preserving Encrypted Databases. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 644–655. ACM, 2015.

[17] Vasilis Pappas, Fernando Krell, Binh Vo, Vladimir Kolesnikov, Tal Malkin, Seung Geol Choi, Wesley George, Angelos Keromytis, and Steve Bellovin. Blind Seer: A Scalable Private DBMS. In *2014 IEEE Symposium on Security and Privacy*, pages 359–374. IEEE, 2014.

[18] Hoi Ting Poon and Ali Miri. Fast phrase search for encrypted cloud storage. *IEEE Transactions on Cloud Computing*, 2017.

[19] Dawn Xiaoding Song, David Wagner, and Adrian Perrig. Practical Techniques for Searches on Encrypted Data. In *Security and Privacy, 2000. S&P 2000. Proceedings. 2000 IEEE Symposium on*, pages 44–55. IEEE, 2000.

[20] Emil Stefanov, Charalampos Papamanthou, and Elaine Shi. Practical Dynamic Searchable Encryption With Small Leakage. In *NDSS*, volume 14, pages 23–26, 2014.

[21] Douglas R Stinson. *Cryptography: Theory and Practice*. CRC press, 2005.

[22] Yinqi Tang, Dawu Gu, Ning Ding, and Haining Lu. Phrase Search Over Encrypted Data With Symmetric Encryption Scheme. In *2012 32nd International Conference on Distributed Computing Systems Workshops*, pages 471–480. IEEE, 2012.

[23] Yoshinao Uchide and Noboru Kunihiro. Searchable symmetric encryption capable of searching for an arbitrary string. Wiley Online Library, 2016.

[24] Mi Wen, Rongxing Lu, Jingsheng Lei, Hongwei Li, Xiaoghui Liang, and Xuemin Sherman Shen. SESA: An efficient searchable encryption scheme for auction in emerging smart grid marketing. *Security and Communication Networks*, 7(1):234–244, 2014.

[25] Steven Zittrower and Cliff C Zou. Encrypted Phrase Searching In The Cloud. In *Global Communications Conference (GLOBECOM), 2012 IEEE*, pages 764–770. IEEE, 2012.

Indranil Ghosh Ray Indranil Ghosh Ray received a B.Sc. degree (Mathematics Honors) from the Calcutta University, in 2000, and a MCA (Master of Computer Applications) degree in 2003 from University of Kalyani. He worked in software industry for six years then as software engineer and senior software engineer. He joined Indian Statistical Institute, Kolkata, India as Junior Research Fellow in 2010 and was promoted to Senior Research Fellow in 2013. He received a Ph.D degree in computer science from Indian Statistical Institute in 2016. He is a Research Associate with the Information Security Group, School of Engineering and Mathematical Sciences, City University of London, UK since June 2016. His research interests include Homomorphic Encryption and its application in Privacy preserving Cloud Computing, Searchable Symmetric Encryption (SSE), Public Key Encryption with keyword search, MDS codes and its applications in Lightweight Cryptography, algebraic immunity of S-Boxes based on Power Mappings but are not limited to these only.

Yogachandran Rahulamathavan Y. Rahulamathavan received a B.Sc. degree (first-class honors) in electronic and telecommunication engineering from the University of Moratuwa, Sri Lanka, in 2008, and a Ph.D. degree in signal processing from Loughborough University, UK in 2011. From April 2008 to September 2008, he was an Engineer at Sri Lanka Telecom, Sri Lanka and from November 2011 to March 2012, he was a Research Assistant with the Advanced Signal Processing Group, School of Electronic, Electrical and Systems Engineering, Loughborough University, UK. He has worked as a Research Fellow with the Information Security Group, School of Engineering and Mathematical Sciences, City University London, UK. Moreover, Dr. Rahulamathavan received a scholarship from Loughborough University to pursue his Ph.D. degree. He is currently working as a Faculty member with Loughborough University, UK. His research interests include signal processing, machine learning and information security and privacy. http://www.drrahul.uk/

Muttukrishnan Rajarajan Muttukrishnan Rajarajan (Raj) is a Professor of Security Engineering at City, University of London, United Kingdom. He currently leads the Information Security Group at City and his research interests are in the areas of privacy preserving data analytics, Cloud Computing, Internet of Things Security and Wireless Networks. He has published well over 300 papers and continues to be involved in the editorial boards and technical programme committees of several international security and privacy conferences and journals. Raj is a visiting Researcher at the British Telecommunications Security Research and Innovation laboratory and is an advisory board member of the Institute of Information Security Professionals (IISP), UK and acts as an advisor to the UK Governments Verify.UK programme. He is a Senior Member of IEEE.