A Public Key Encryption Scheme for String Identification

Indranil Ghosh Ray and Muttukrishnan Rajarajan
Department of Electrical and Computer Engineering
City, University of London.
London, United Kingdom
Email: Indranil.Ghosh-Ray@city.ac.uk, r.muttukrishnan@city.ac.uk

***Abstract*—One of the major limitations of index based encrypted string search on big dataset is the inherent problem of big index generation, maintenance and update which stops it from being dynamic in a sense that one could not modify data or add or remove keywords. Also for a resource constraint client, to generate an index linear in the size of big dataset is difficult. In this paper, we provide an efficient easy-to-implement public key based searchable encryption scheme for string search which is adaptively secure and does not need any index. We provide concrete proof of the adaptive security of our scheme against honest-but-curious server. We validate our scheme against three different publicly available datasets.**

***Keywords*-Cloud storage; Searchable encryption; PEKS; Public key; Homomorphic encryption; Elliptic Curve;**

I. Introduction

In recent past, *searchable encryption* (SE) has unfolded as a crucial research domain at the intersection of cryptography and secure cloud computing. SE allows one party to outsource the storage of its data to another party (a cloud) privately while enabling to search selectively over it. 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.

Although few works are available in the literature involving string search (e.g. [1]–[4]), but all of them are based on index based searching using SSE and none of them are adaptively secure and expose lots of informations to the server following the search.

Our Contributions :

1. In [5], authors proposed PEKS scheme for *keyword search*. We extend their construction and propose an *adaptively* secure PEKS scheme for *string search* which takes one round of communication and $O(n)$ times of computation over n documents.
2. Our scheme is at the intersection of *partially homomorphic public key encryption* and PEKS. The partial homomorphic property is used to detect adjacency of words which is crucial for string identification (see Remark 1). We observe that unlike fully homomorphic system, a partially homomorphic system will suffice and also gives reasonable performance.
3. We provide a formal proof to show that the scheme is *adaptively* secure. To the best of our knowledge, this is the first SE scheme for string search which is adaptively secure.
4. We validate our scheme using three different datasets (the speech data, English text data and DNS records).
5. With SSE based search, additional computation is needed for index maintenance with subsequent modification of data. However, with this new scheme of string search, no such additional computation is needed.

II. Related Works

For the last ten years, searchable encryption has been the focus for many leading research groups and several results were proposed [5]–[17]. In [13], authors defined computational and statistical relaxations of the existing notion of perfect consistency and provided a new scheme that was statistically consistent. In [12] authors presented as-strong-as-possible definitions of privacy and some constructions for public-key based encryption schemes where the encryption algorithm is deterministic. The work in [12] 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 to as PEKS and provided several constructions. In [15], authors show how to create a public-key encryption scheme that allows PIR (private information retrieval) searching over encrypted documents. In [14], authors defined and solved the problem of privacy-preserving *multi-keyword ranked search* over encrypted data in cloud computing (MRSE).

Dynamic SSE was first considered by Song et al. [7], but no solution with sublinear search time existed before the work in [18]. Recently, two new dynamic SSE schemes have been proposed. The first one, by Cash et al. [10], which is an extension of [8]. In [8], authors presented another efficient SSE scheme which supports complex queries involving multiple keywords. Similar scheme may be found in [9]. In [19], authors studied the trade-off between locality and server storage size of SSE schemes. In [6], authors introduced

the idea of symmetric searchable encryption (SSE) with improved 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 [9], authors studied efficient sub linear search techniques for arbitrary Boolean queries. In [11], Stefanov et al. designed a scheme for the first time to address *forward secrecy*. In [7], the search complexity is linear in the number of documents stored in the database.

In [1]–[4], SSE schemes are developed for string search. With these schemes searching is done based on some secure index which is generated by client and stored at the cloud. The size of the index is as large as the dataset.

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

Property	SSE [6]	PSS [2]	[1]	LPSSE [3]	this paper
String search	no	yes	yes	yes	yes
adaptive security	yes	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($max\{\|D_i\| : i \in [1,n]\}$)

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 $D(w_j)$ 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 $\boldsymbol{D}(s) \subseteq \bigcup_{i=1}^{l} D(w_i)$. We denote by $\delta(\boldsymbol{D})$, all the distinct keywords connected to the document collection $\boldsymbol{D}$.

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

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 $\lambda > a$, $negl(\lambda) < \frac{1}{p(\lambda)}$ [20].

In addition, we use *massage authentication code*, $MAC_k(.)$ [20] which outputs in λ bits. We treat these outputs as elements of $\mathbb{Z}_p$ where p is a λ bit prime. We also use *pseudo prime number generator* [21], denoted by $PPNG(1^\lambda)$ which outputs a λ-bit probabilistic prime number. For a finite set S, we denote the operation of picking an element uniformly at random from S by $x \leftarrow S$.

Public Key Encryption with Searching:

Definition 1. *A non-interactive public key encryption with keyword search scheme consists of the following polynomial time randomized algorithms:*

1. *KeyGen(λ): This algorithm takes security parameter, λ, and generates a public/private key pair A_{pub}, A_{priv}.*
2. *PEKS(A_{pub}, w): For a public key A_{pub}, and a word w, this algorithm produces a searchable encryption of w.*
3. *Trapdoor($A_{priv,w}$): Given a private key A_{priv} and a word w, this algorithm produces a trapdoor t_w.*
4. *Search(A_{pub}, C, t_w): Given the public key A_{pub} and searchable encryption $C = PEKS(A_{pub}, w')$ and a trapdoor t_w, this algorithm outputs 'yes' if $w = w'$, otherwise 'no'.*

Bilinear map: Our construction is based on bilinear map $\mathbf{e} : G_1 \times G_2 \to G_3$, where G_1, G_2 and G_3 are groups of same order p, where p is λ-bit prime. Here the group G_1 is the set of points of the elliptic curve $Y^2 = X^3 - 3X + B$ over $\mathbb{Z}_p$. G_2 corresponds to the points of twisted curve $Y^2 = X^3 - 3X - B$ over $\mathbb{Z}_p$. For any $g \in G_3$, we write g_b to express the λ-bit representation of g. We now formally define bilinear map using G_1, G_2 and G_3 which will be helpful to formalize the security proof of our scheme.

Definition 2. *Let G_1, G_2 and G_3 are three groups of prime order, say p, such that there exists a map $e : G_1 \times G_2 \to G_3$. The map is a bilinear map if the following properties hold:*

1. *Computable: for any $g_1 \in G_1$ and $g_2 \in G_2$, there is a polynomial time algorithm to compute $\mathbf{e}(g_1, g_2) \in G_3$.*
2. *Bilinear: for any $a, b \in \mathbb{Z}_p$, $\mathbf{e}(a.g_1, b.g_2) = ab.\mathbf{e}(g_1, g_2)$.*
3. *If g_1 and g_2 are generators of G_1 and G_2 respectively, then $\mathbf{e}(g_1, g_2)$ is a generator of G_3.*

Two cryptographic hash functions are used in out scheme π_{pss}, namely H_1 and H_2 which are as follows: $H_1 : \{0,1\}^* \times \{0,1\}^* \mapsto G_2$, $H_2 : \{0,1\}^* \times G_3 \mapsto \{0,1\}^\lambda$. H_1 and H_2 are realized using $MAC()$ as follows:

$$H_1(k, w) = MAC_k(w) \times g_2.$$

For some $g \in G_3$,

$$H_2(k, g) = MAC_k(g_b), \text{ where } k \text{ is the key.}$$

In our scheme we denote the key (session key) as k_s and for simplicity of expression we write $H_1(k_s, w)$ and $H_2(k_s, g)$ as $H_1(w)$ and $H_2(g)$ respectively.

IV. Our Scheme

In this section we present our SSE scheme Π_{ss} for string search.

Scheme 1 (Π_{pss})**.** *The scheme Π_{pss} is a collection of four polynomial time algorithms (KeyGen, Enc, 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. It takes a security parameter λ, and returns the setup. Since KeyGen is randomized, we write it as $(p, G_1, G_2, G_3, g_1, h, g_2, a, k_s) \leftarrow$ KeyGen(1^λ). The public key is $pk = (g_1, h, g_2)$ and the secret key is $sk = a \leftarrow \mathbb{Z}_p$. The session key is k_s and p is a random prime number whose length is polynomially bounded in λ. pk, k_s and p are shared with server.*
2. *PEKS_Enc(k_s, g_1, h, g_2, D_i) : PEKS_Enc is a probabilistic algorithm run by the client to generate the cipher text C_i corresponding to document D_i. It takes k_s, pk and p, and returns C_i. Since PEKS_Enc is probabilistic, we write this as $C_i \leftarrow PEKS\_Enc(k_s, g_1, h, g_2, D_i)$.*
3. *Trapdoor(a, s) : Trapdoor is a deterministic 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_s, a 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 deterministic, we write this as $t = Trapdoor(a, s)$.*
4. *Search$(\mathbf{C}, t, k_s, g_1, h, g_2)$: Search is run by the server in order to search for the documents in D that contain the string s. It takes a ciphertext collection C corresponding to D and the trapdoor t corresponding to s and returns D(s), the set of identifiers of documents containing the string s.*

The Construction: Now we provide the actual algorithms for key generation (Algorithm 1), PEKS encryption (Algorithm 2), trapdoor generation (Algorithm 3) and searching (Algorithm 4).

Algorithm 1 Keygen

Input security parameter λ.
Output G_1, G_2, g_1, $h = a.g_1$, g_2, a, k_s and p.
$(p, G_1, G_2, g_1, h, g_2, a, k_s) \leftarrow KeyGen(1^\lambda)$;
$p \leftarrow PPNG(1^\lambda)$;

Remark 1. *In Algorithm 2, to encrypt j-th sentence $s_j = (w_1, \ldots, w_m)$, an element r_j is chosen uniformly at random from $\mathbb{Z}_p$. The encryption of w_i, i.e., $[A_i, B_i]$ is computed as follows: $A_i = (i + r_j)g_1$ and $B_i = H_2(\mathbf{e}((i + r_j).h, H_1(w_i)))$, $i \in \{1, \ldots, m\}$. It may be noted that for encrypted blocks, say, $[A_c, B_c]$ and $[A_d, B_d]$, of two adjacent words, $A_d = A_c + g_1$. This additive homomorphism of elliptic curve group is utilized for string identification.*

Algorithm 2 PEKS_Enc

Input k_s, g_1, g_2, h p, $D = (D_1, \ldots, D_n)$.
Output $C = (c_1, \ldots, c_n)$.
Form a collection $W = \{w_1, \ldots, w_d\}$ of all distinct words occurring in D;
$i \leftarrow 1$;
while $i \leq n$ **do**
  open D_i in read mode and C_i in write mode;
  $j \leftarrow 1$;
  while (!EOF) do
    read j-th string in $s_j = (w_1, \ldots, w_m)$;
    $r_j \leftarrow \mathbb{Z}_p$;
    $k \leftarrow 1$;
    while $k \leq m$ **do**
      compute $t_{w_k} = \mathbf{e}((k + r_j).h, H_1(w_k))$;
      Compute $A = (k + r_j).g_1, B = H_2(t_{w_k})$;
      Append $[A, B]$ in C_i;
      $i \leftarrow k + 1$;
    end while
    $j \leftarrow j + 1$;
  end while
  $i \leftarrow i + 1$;
end while
shuffle all the blocks of the form $[A, B]$ uniformly throughout the cipher;

Algorithm 3 Trapdoor

Input $w = (w_1, w_2, \ldots, w_l)$, k_s, a, p.
Output $t = (t_1, \ldots, t_l)$.
$j \leftarrow 1$;
while $j \leq l$ **do**
  $t_j = a.H_1(w_j)$;
  $j \leftarrow j + 1$;
end while

Correctness of the PEKS encryption of our scheme is easy to realize from the following lemma.

Lemma 1 (Correctness)**.** *Let $s = (w_1, \ldots, w_l)$ be a string such that s is in the document D_i in the document collection D and $C_i \leftarrow PEKS\_Enc(k_s, g_1, h, g_2, D_i)$. Let $[A_1, B_1], \ldots, [A_l, B_l]$ are the encrypted blocks in C_i for the string s taken in order. Then Search$(\mathbf{C}, t, k_s, g_1, h, g_2)$ will point out the identifier corresponding to D_i where $t = (t_1 \ldots, t_l)$ is the trapdoor corresponding to s taken in order, i.e., $id(D_i) \in Search(\mathbf{C}, t, k_s, g_1, h, g_2)$.*

Proof: Let $A_1 = (r + 1)g_1$, where $r \in \mathbb{Z}_p$. Note that, then

Algorithm 4 Search

Input $t = (t_1, \ldots, t_l)$, $\{C_1 \ldots, C_n\}$, k_s, g_1, h, g_2.
Output $encrypted\_file\_pointers$, a list of encrypted document pointers;

```
exists = false;
counter ← 0;
i ← 1;
while i ≤ n do
  read all blocks of the form [A, B] from C_i and arrange
  A's in listA and B's in listB;
  j ← 1;
  while (j ≤ listA.length) do
    compute chk = H_2(e(listA[j], t_1));
    if (chk = listB[j]) then
      A = listA[j];
      exists = true;
      counter = 1;
      break;
    end if
  end while
  if (counter = 1) then
    k ← 2;
    while (k ≤ l) do
      j ← 1;
      while (j ≤ listA.length) do
        if [(listA[j] = A × g_1) &&
        (H_2(e(listA[j], t_j)) = listB[j])] then
          A = A × g_1;
          counter ← counter + 1;
          break;
        end if
        j ← j + 1;
      end while
      if (counter ≠ k) then
        exists = false;
        break;
      end if
      k ← k + 1;
    end while
  end if
  if (exists = true) then
    add i to encrypted_file_pointers
  end if
end while
```

$B_1 = H_2(\mathbf{e}((r_1+1)h, H_1(w_1)))$. Now,

$$\begin{aligned}\mathbf{e}(A_1, t_1) &= \mathbf{e}\left(A_1, a.H_1(w_1)\right)\\ &= \mathbf{e}\left((r+1)g_1, a.H_1(w_1)\right)\\ &= a.\mathbf{e}\left((r+1)g_1, H_1(w_1)\right)\\ &= \mathbf{e}\left((r+1)a.g_1, H_1(w_1)\right)\\ &= \mathbf{e}\left((r+1)h, H_1(w_1)\right)\end{aligned}$$

Thus $H_2(\mathbf{e}(A_1, t_1)) = H_2(\mathbf{e}((r+1)h, H_1(w_1))) = B_1$. This detects $[A_1, B_1]$ for t_1. Rest A_j's are detected using the property $A_{j+1} = (g_1 + A_j)$ and corresponding B_j's are computed similar to B_1. Hence the result follows. ■

V. Security Analysis

In this section we show that our scheme is secure against adaptive string attack. We first define an experiment for any PEKS scheme $\pi = (KeyGen, PEKS\_Enc, Trapdoor, Search)$, any adversary $\mathcal{A}$, and any value k of the security parameter.

Definition 3. *[$Game\_Adaptive_{\mathcal{A},\pi}(1^k)$]*

1. *A key (pk, sk) is generated by running $KeyGen(1^k)$.*
2. *The adversary $\mathcal{A}$ is given input 1^k and oracle access to $PEKS\_Enc(.)$, $H_1(.)$, $H_2(.)$ and $Trapdoor(.)$ and outputs a pair of strings s_0, s_1 of the same length, say m.*
3. *$b \leftarrow \{0,1\}$ and then a ciphertext $c \leftarrow PEKS\_Enc(s_b)$ is computed and given to $\mathcal{A}$. We call c the challenge ciphertext.*
4. *The adversary $\mathcal{A}$ continues to have oracle access to $PEKS\_Enc(.)$, $H_1(.)$, $H_2(.)$ and $Trapdoor(.)$ and outputs a bit b'.*
5. *The output of the experiment is defined to be 1 if $b' = b$, and 0 otherwise. In case $Game\_Adaptive_{\mathcal{A},\pi}(1^k) = 1$, we say that $\mathcal{A}$ succeeded.*

Definition 4. *A PEKS scheme, denoted by $\pi = (KeyGen, PEKS\_Enc, Trapdoor, Search)$, is said to be adaptively secure under chosen plain text attack if for all probabilistic polynomial time adversaries $\mathcal{A}$, there exists a negligible function $negl$ such that $Pr[Game\_Adaptive_{\mathcal{A},\pi}(1^k) = 1] \leq \frac{1}{2} + negl(k)$, where the probability is taken over the random coins used by $\mathcal{A}$, as well as the random coins used in the game.*

In the next theorem, we prove that our scheme π_{pss} is adaptively secure.

Bilinear Diffie-Hellman problem(BDH): Let g_1 and g_2 be the generators of G_1 and G_2 respectively and $a, b, c \in \mathbb{Z}_p$ such that $a \neq bc$, $b \neq ac$ and $c \neq ab$. The BDH problem is as follows:
given $g_1, g_2, a.g_1, a.g_2, b.g_1, b.g_2, c.g_1, c.g_2$ as input, compute $(abc).\mathbf{e}(g_1, g_2) \in G_3$. BDH is said to be intractable if

all polynomial time algorithms have a negligible advantage in solving BDH.

Theorem 1. *π_{pss} is adaptively secure against chosen keyword attack in the random oracle model assuming BDH is intractable.*

Proof: Suppose $\mathcal{A}$ is a polynomial size adversary that has advantage $negl(\lambda)$ in breaking π_{pss}, i,e., $Pr[Game\_Adaptive_{\mathcal{A},\pi_{pss}}(1^k) = 1] \leq \frac{1}{2} + negl(\lambda)$. Suppose $\mathcal{A}$ makes at most n_{H_2} hash function queries to H_2 and at most and at most n_T trapdoor queries. We construct a simulator $\mathcal{S}$, that solves the BDH problem with probability at least $\epsilon' = \frac{negl(\lambda)}{e \times n_T \times n_{H_2}}$, where e is the base of natural logarithm.

Let g_1 and g_2 be the generators of G_1 and G_2 respectively. Let the simulator $\mathcal{S}$ is given g_1, g_2, $p_1 = a.g_1$, $q_1 = a.g_2$, $p_2 = b.g_1$, $q_2 = b.g_2$, $p_3 = c.g_1$, $q_3 = c.g_2$. The simulator $\mathcal{S}$ simulates the challenger and interacts with the adversary $\mathcal{A}$ as follows:

$\mathcal{S}$ sets $A_{pub} = [g_1, g_2, p_1]$.

1. (simulating H_1^*): Whenever $\mathcal{A}$ queries for H_1, $\mathcal{S}$ maintains a list $\langle w_j, h_j, a_j, c_j \rangle$ called $list_1$ which is initially empty. When $\mathcal{A}$ queries for w_i, $\mathcal{S}$ responds as follows:
   I. If w_i already appears in $list_1$, say $\langle w_i, h_i, a_i, c_i \rangle$, then algorithm $\mathcal{S}$ responds with $H_1^*(w_i) = h_i \in G_2$.
   II. Otherwise $\mathcal{S}$ generates a random coin $c_i\{0,1\}$ such that $Pr[c_i = 0] = \frac{1}{n_T+1}$.
   III. $\mathcal{S}$ picks a random $a_i \in \mathbb{Z}_p$. If $c_i = 0$, $\mathcal{S}$ sets $h_i = q_2 + a_i g_2 \in G_2$, otherwise $h_i = a_i g_2 \in G_2$.
   IV. $\mathcal{S}$ adds $\langle w_i, h_i, a_i, c_i \rangle$ to the list and responds to $\mathcal{A}$'s query by setting $H_1^*(w_i) = h_i \in G_2$.
2. (simulating H_2^*): Whenever $\mathcal{A}$ queries for $H_2(t)$ for a new t, $\mathcal{S}$ picks a random value v from $\{0,1\}^\lambda$ and sets $H_2^*(t) = v$ and adds $\langle t, v \rangle$ to $list_2$. If t is already in the list, then $\mathcal{S}$ just sets $H_2^*(t) = v$.
3. (simulating trapdoor) : To get the simulated trapdoor corresponding to w_i, $\mathcal{S}$ sets $h_i = H_1^*(w_i)$. If $c_i = 0$, it stops. Otherwise $\mathcal{S}$ sets the trapdoor $t_i = a_i.q_1$.

challenge phase :

$\mathcal{A}$ produces a pair of challenge strings $s_0 = (w_{0,1}, \ldots, w_{0,m})$ and $s_1 = (w_{1,1}, \ldots, w_{1,m})$ which are of same length, say, m.

I $\mathcal{S}$ computes $H_1^*(w_{i,j}) = h_{i,j}$ for $0 \leq i \leq 1$, $1 \leq j \leq m$. For $i = 0$ and 1, let the corresponding entries in $list_1$ are $\langle w_{i,k}, h_{i,k}, a_{i,k}, c_{i,k} \rangle$, where $1 \leq k \leq m$.

II $\mathcal{S}$ randomly selects $b \leftarrow \{0,1\}$.

III $\mathcal{S}$ responds to the challenge $\{[u_3, J_1], [u_3 * g_1, J_2], \ldots, [u_3 * g_1^{m-1}, J_m]\}$ where $\mathcal{S}$ choses J_i's randomly from $\{0,1\}^\lambda$.

IV $\mathcal{A}$ can continue to issue trapdoor queries for strings other than s_0 and s_1.

It may be noted that the challenge implicitly defines J_1 as $H_2^*\left(e\left(cp_1, H_1(w_{b,1})\right)\right)$. Now,

$$\begin{aligned} J_1 &= H_2^*\left(e\left(cp_1, H_1(w_{b,1})\right)\right) \\ &= H_2^*\left(e\left(cag_1, (bg_2 + a_b g_2)\right)\right) \\ &= H_2^*\left(e\left(acg_1, (b + a_b) g_2\right)\right) \\ &= H_2^*\left(ac(b + a_b) e\left(g_1, g_2\right)\right) \end{aligned}$$

So similar computation for $w_{b,k}$, $k = 2, \ldots, m$ indicates that this is a valid PEKS for the string s_b.

Output phase : Eventually $\mathcal{A}$ outputs the guess $b' \in \{0,1\}$. Then $\mathcal{S}$ picks a random pair (t, v) from $list_2$ and outputs $t - a_b \mathbf{e}(p_1, q_3)$ as its guess for $abc.\mathbf{e}(g_1, g_2)$, where a_b is the value used in the challenge phase. $\mathcal{A}$ must have issued the query for either s_0 or s_1 as otherwise $\mathcal{A}$'s view on the PEKS will be independent of s_0 or s_1 and thus $\mathcal{A}$ cannot have the advantage of ϵ in breaking the scheme. Therefore with probability $\frac{1}{2}$, $list_2$ contains an entry (t, v) such that $t = ac(b + a_b).\mathbf{e}\left(g_1, g_2\right)$. Let ε_0 be the event denoting that $\mathcal{S}$ selects this pair (t, v). Then $P[\varepsilon_0] = \frac{1}{n_{H_2}}$, and then $\mathcal{S}$ outputs

$$\begin{aligned} t - a_b \mathbf{e}(p_1, q_3) &= ac(b + a_b).\mathbf{e}\left(g_1, g_2\right) - a_b ac.\mathbf{e}(g_1, g_2) \\ &= abc.\mathbf{e}(g_1, g_2). \end{aligned}$$

Let ε_1 be the event denoting that $\mathcal{S}$ does not abort while $\mathcal{A}$ is making trapdoor queries. Then, $P[\varepsilon_1] = 1 - \frac{1}{(n_T+1)^{n_T}} \geq \frac{1}{e}$ (see APPENDIX A).

Let ε_2 be the event denoting $\mathcal{S}$ does not abort while $\mathcal{A}$ is making challenge. Then, $P[\varepsilon_2] \geq \frac{1}{n_T}$ (see APPENDIX B).

Let ε_3 be the event denoting $\mathcal{A}$ queried against at least one of the strings s_0 and s_1. Then $P[\varepsilon_3] \geq 2.negl(\lambda)$ (see APPENDIX C).

So, $P[\mathcal{S} \text{ solves BDH }] = P[\varepsilon_0] \times P[\varepsilon_1] \times P[\varepsilon_2] \times P[\varepsilon_3] \geq \frac{negl(\lambda)}{e \times n_T \times n_{H_2}}$. ∎

VI. Experimental Results

Now we provide complexity analysis of our scheme in next two lemmas.

Lemma 2. *Using Π_{ss}, the number of group operations for searching a query of l-word string in one ciphertext document C is $O\left(\left(\frac{|C|}{2\lambda}\right) + (l-1)\right)$.*

Proof: It is easy to observe that each block of the form $[A, B]$ is of size 2λ bits. Thus in the encrypted document C, the number of blocks is $\frac{|C|}{2\lambda}$. Since the blocks are shuffled, to detect the first block requires $O\left(\frac{|C|}{2\lambda}\right)$ group operations. If the string exists, then each of the rest $l-1$ steps needs $O(\frac{|C|}{2\lambda})$ searching to detect corresponding A_i's followed by a group operation to detect B_i. Thus the number of group operations is $O\left(\left(\frac{|C|}{2\lambda}\right) + (l-1)\right)$. ∎

Now we provide performance results of our proposed scheme Π_{ss} based on three different datasets.The comparison of performances and storages of various schemes with

that of Π_{ss} can be found in Table I. We build a 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, namely 'Elliptic Curve'.

A. Speech Data

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 automatic speech recognition systems (ASR).

Figure 1 illustrates the time needed to generate PEKS ciphers for the speech data. In Figure1 we plot the time (in seconds along Y-axis) needed to encrypt documents (along X-axis). It may be noted that the graph reflects a linear growth with the increase in number of documents. Also the time needed for encrypting 200 documents is 150 seconds. Figure 2 represents the performance of searching over encrypted data. Here also we plotted time of search in seconds along Y-axis against the number of encrypted documents which are plotted along X-axis. This graph also shows a linear growth with the increase of number of encrypted documents with a maximum of 70 seconds needed to search over 200 encrypted documents.

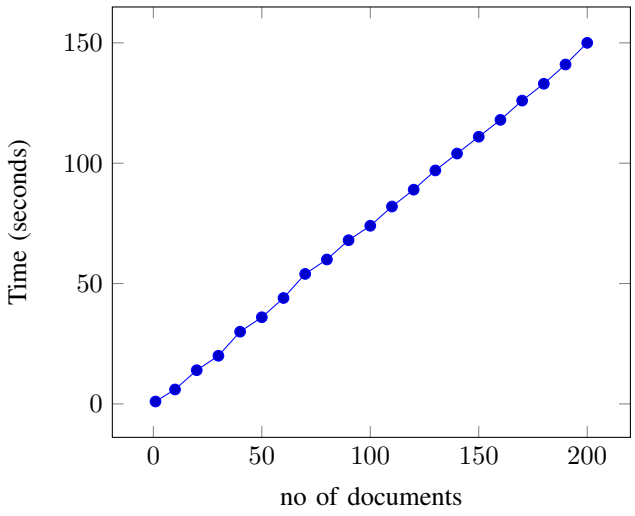

Figure 1. PEKS encryption graph of TIMIT speech data

B. English Text Data

In this section we provide results based on a dataset which is a Switchboard-1 Telephone Speech Corpus (LDC97S62) [22] originally collected by Texas Instruments in 1990-1, sponsored by DARPA.

Figure 3 illustrates the time needed to generate PEKS ciphers for English text data where we plot the time (in seconds along Y-axis) needed against number of encrypted

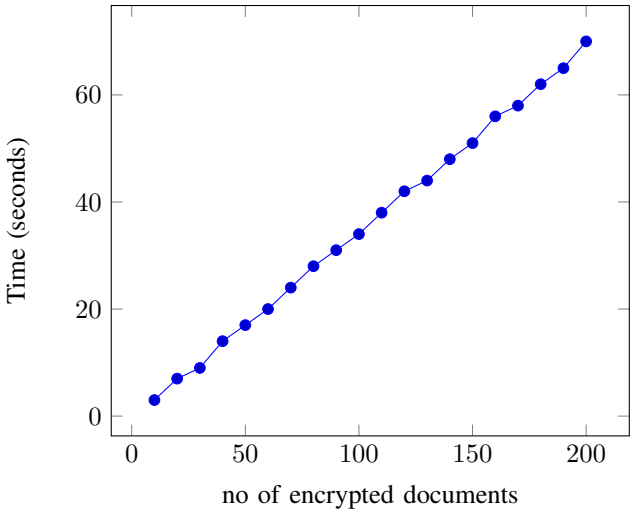

Figure 2. Search time graph of TIMIT speech data

documents (along X-axis). This graph also reflects a linear growth with the increase in number of documents. The time needed for encrypting 80 documents is 1586 seconds. Figure 4 compares the performance of searching over encrypted data for search strings of two different lengths, 1 and 3. So this graph compares the keyword search against string search.

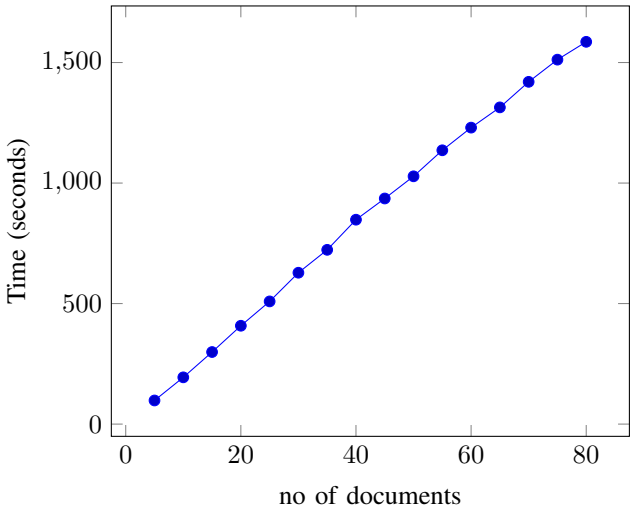

Figure 3. PEKS encryption graph for Switchboard-1 speech database

C. DNA Data

We have implemented our algorithm for searching sequences of SNP's (Single Nucleotide Polymorphism) in large genome databases which is available at [23]. SNP represents a difference in a nucleotide which is a single DNA building block.

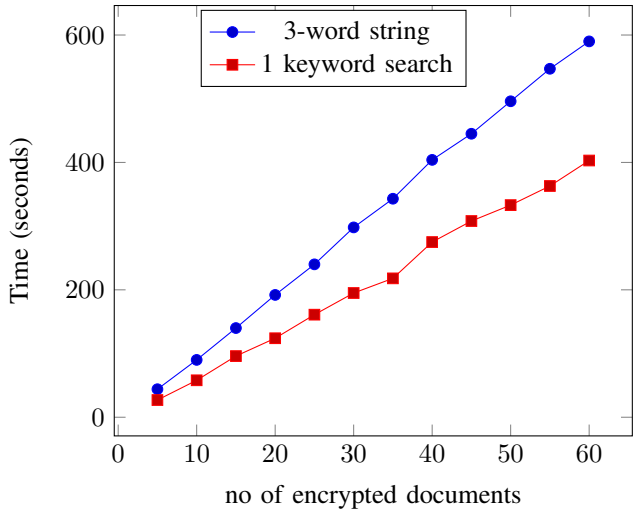

Figure 4. Search time graph for Switchboard-1 speech database

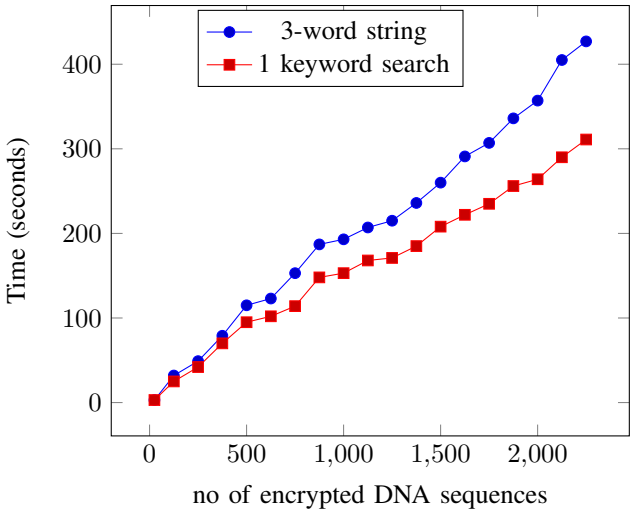

Figure 6. Search time graph over DNA data comparing the search time using one keyword against a string of three keywords.

Figure 5 illustrates the time needed to generate encrypted 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. Also the time needed for encrypting 750 DNA sequences is 1005 seconds. Figure 6 compares the performance of searching patterns over encrypted DNA sequences for two different lengths, 1 and 3 over the set K. So in binary, both are strings of length 8 and 24 respectively.

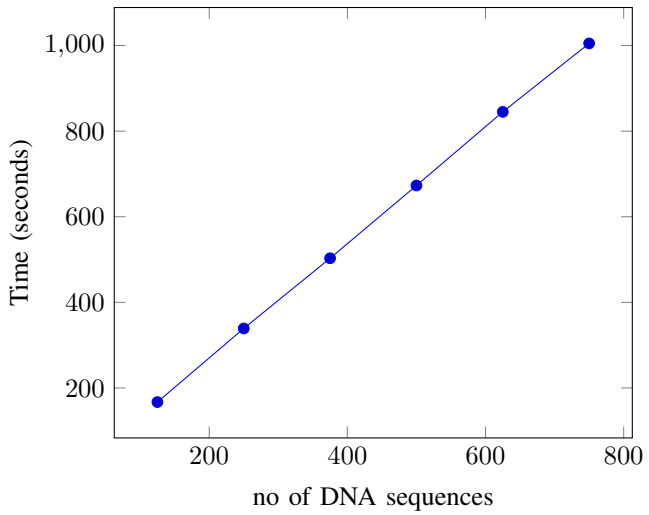

Figure 5. PEKS encryption graph of DNA data

VII. Conclusion

With the increasing number of documents stored in cloud, searching for the desired document can be a difficult and resource intensive task. With SE, one can store a large collection of encrypted documents with a server for future search. In this paper we revisited the PEKS construction of [5] and proposed a new PEKS scheme Π_{pss} for string search. We have shown that our scheme is secure under the *adaptive* indistinguishability definition. We provide performance results of our proposed scheme Π_{ss} based on three different commercial datasets [22]–[24]. We provide comparison of performances and storages of various schemes with that of Π_{ss} in Table I.

Acknowledgment

The authors would like to thank Intelligent Voice for their various help in this work.

References

[1] S. Zittrower and C. C. Zou, "Encrypted Phrase Searching In The Cloud," in *Global Communications Conference (GLOBECOM), 2012 IEEE*. IEEE, 2012, pp. 764–770.

[2] Y. Tang, D. Gu, N. Ding, and H. Lu, "Phrase Search Over Encrypted Data With Symmetric Encryption Scheme," in *2012 32nd International Conference on Distributed Computing Systems Workshops*. IEEE, 2012, pp. 471–480.

[3] M. Li, W. Jia, C. Guo, W. Sun, and X. Tan, "LPSSE: Lightweight Phrase Search With Symmetric Searchable Encryption in Cloud Storage," in *Information Technology-New Generations (ITNG), 2015 12th International Conference on*. IEEE, 2015, pp. 174–178.

[4] Y. Uchide and N. Kunihiro, "Searchable Symmetric Encryption Capable of Searching for an Arbitrary String." Wiley Online Library, 2016.

[5] D. Boneh, G. Di Crescenzo, R. Ostrovsky, and G. Persiano, "Public Key Encryption With Keyword Search," in *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2004, pp. 506–522.

[6] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky, "Searchable Symmetric Encryption: Improved Definitions and Efficient Constructions," vol. 19, no. 5. IOS Press, 2011, pp. 895–934.

[7] D. X. Song, D. Wagner, and A. Perrig, "Practical Techniques for Searches on Encrypted Data," in *Security and Privacy, 2000. S&P 2000. Proceedings. 2000 IEEE Symposium on*. IEEE, 2000, pp. 44–55.

[8] D. Cash, S. Jarecki, C. Jutla, H. Krawczyk, M.-C. Roşu, and M. Steiner, "Highly-Scalable Searchable Symmetric Encryption With Support for Boolean Queries," in *Advances in Cryptology–CRYPTO 2013*. Springer, 2013, pp. 353–373.

[9] V. Pappas, F. Krell, B. Vo, V. Kolesnikov, T. Malkin, S. G. Choi, W. George, A. Keromytis, and S. Bellovin, "Blind Seer: A Scalable Private DBMS," in *2014 IEEE Symposium on Security and Privacy*. IEEE, 2014, pp. 359–374.

[10] D. Cash, J. Jaeger, S. Jarecki, C. S. Jutla, H. Krawczyk, M.-C. Rosu, and M. Steiner, "Dynamic Searchable Encryption in Very-Large Databases: Data Structures and Implementation." vol. 2014. Citeseer, 2014, p. 853.

[11] E. Stefanov, C. Papamanthou, and E. Shi, "Practical Dynamic Searchable Encryption With Small Leakage." in *NDSS*, vol. 14, 2014, pp. 23–26.

[12] M. Bellare, A. Boldyreva, and A. ONeill, "Deterministic and Efficiently Searchable Encryption," in *Annual International Cryptology Conference*. Springer, 2007, pp. 535–552.

[13] M. Abdalla, M. Bellare, D. Catalano, E. Kiltz, T. Kohno, T. Lange, J. Malone-Lee, G. Neven, P. Paillier, and H. Shi, "Searchable Encryption Revisited: Consistency Properties, Relation to Anonymous IBE, and Extensions," vol. 21, no. 3. Springer, 2008, pp. 350–391.

[14] N. Cao, C. Wang, M. Li, K. Ren, and W. Lou, "Privacy-Preserving Multi-Keyword Ranked Search Over Encrypted Cloud Data," vol. 25, no. 1. IEEE, 2014, pp. 222–233.

[15] D. Boneh, E. Kushilevitz, R. Ostrovsky, and W. E. Skeith III, "Public Key Encryption That Allows PIR Queries," in *Annual International Cryptology Conference*. Springer, 2007, pp. 50–67.

[16] M. Naveed, S. Kamara, and C. V. Wright, "Inference Attacks on Property-Preserving Encrypted Databases," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2015, pp. 644–655.

[17] D. Cash, P. Grubbs, J. Perry, and T. Ristenpart, "Leakage-Abuse Attacks Against Searchable Encryption," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2015, pp. 668–679.

[18] S. Kamara, C. Papamanthou, and T. Roeder, "Dynamic Searchable Symmetric Encryption," in *Proceedings of the 2012 ACM conference on Computer and communications security*. ACM, 2012, pp. 965–976.

[19] D. Cash and S. Tessaro, "The Locality of Searchable Symmetric Encryption," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2014, pp. 351–368.

[20] J. Katz and Y. Lindell, *Introduction to Modern Cryptography*. CRC press, 2014.

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

[22] [Online]. Available: https://catalog.ldc.upenn.edu/

[23] [Online]. Available: https://github.com/iskana/PBWT-sec/tree/master/sample_dat

[24] [Online]. Available: http://www.fon.hum.uva.nl/david/ma_ssp/2007/TIMIT/train/dr5/fsdc0/

APPENDIX A

Let ε_1 be the event denoting $\mathcal{S}$ does not abort while $\mathcal{A}$ is making trapdoor queries. Since in $list_1$, the distribution of c_i's are independent of the distribution of h_i's, so $P[\text{one trapdoor query triggering abort}] = \frac{1}{(n_T+1)}$. So,

$$\begin{aligned} P[\varepsilon_1] &= (1 - P[\text{one trapdoor query triggering abort}])^{n_T} \\ &= \left(1 - \frac{1}{(n_T+1)}\right)^{n_T} \geq \frac{1}{e}. \end{aligned}$$

APPENDIX B

Let ε_2 be the event denoting $\mathcal{S}$ does not abort while $\mathcal{A}$ is making challenge. Now,

$$\begin{aligned} &P[\mathcal{S} \text{ will abort during challenge}] \\ =&P[c_{i,k} = 1 : i \in \{0,1\}, k \in \{1,\ldots,m\}] \\ =&\left(1 - \frac{1}{n_T+1}\right)^{2m} \\ \leq&1 - \frac{1}{n_T}. \end{aligned}$$

So,

$$\begin{aligned} P[\varepsilon_2] &\geq 1 - p[\mathcal{S} \text{ will abort during challenge}] \\ &= \frac{1}{n_T}. \end{aligned}$$

APPENDIX C

Let ε_3 be the event denoting $\mathcal{A}$ queried against at least one of the strings s_0 and s_1.

$$\begin{aligned} &P[\ \mathcal{A} \text{ breaks the scheme}] \\ &= P[b = b'] \\ &= P[(b = b')|\varepsilon_3]P[\varepsilon_3] + P[(b = b')|\varepsilon_3']P[\varepsilon_3'] \\ &\leq \frac{1}{2}P[\varepsilon_3] + \frac{1}{2}. \end{aligned}$$

From the assumption on $negl(\lambda)$ (see Definition 4),

$$\begin{aligned} negl(\lambda) &\leq |P[b = b'] - \frac{1}{2}| \\ &= \frac{1}{2}P[\varepsilon_3]. \end{aligned}$$

Thus $P[\varepsilon_3] \geq 2\epsilon$.