R-PEKS: RBAC enabled PEKS for secure access of cloud data

IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received August 29, 2019, accepted September 11, 2019, date of publication September 16, 2019,
date of current version September 26, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2941560

R-PEKS: RBAC Enabled PEKS for Secure Access of Cloud Data

K. RAJESH RAO[1], INDRANIL GHOSH RAY[2], WAQAR ASIF[2], ASHALATHA NAYAK[3], (Member, IEEE), AND MUTTUKRISHNAN RAJARAJAN[2], (Senior Member, IEEE)

[1]Department of Information and Communication Technology, Manipal Institute of Technology, Manipal 576104, India
[2]Department of Electrical and Computer Engineering, City, University of London, London EC1V 0HB, U.K.
[3]Department of Computer Science and Engineering, Manipal Institute of Technology, Manipal 576104, India

Corresponding authors: Indranil Ghosh Ray (indranil.ghosh-ray@city.ac.uk) and Ashalatha Nayak (asha.nayak@manipal.edu)

This work was supported by the European Union's Horizon 2020 Framework Programme for Research and Innovation under Grant 740690.

ABSTRACT In the recent past, few works have been done by combining attribute-based access control with multi-user PEKS, i.e., public key encryption with keyword search. Such attribute enabled searchable encryption is most suitable for applications where the changing of privileges is done once in a while. However, to date, no efficient and secure scheme is available in the literature that is suitable for these applications where changing privileges are done frequently. In this paper our contributions are twofold. Firstly, we propose a new PEKS scheme for string search, which, unlike the previous constructions, is free from bi-linear mapping and is efficient by 97% compared to PEKS for string search proposed by Ray et.al in TrustCom 2017. Secondly, we introduce role based access control (RBAC) to multi-user PEKS, where an arbitrary group of users can search and access the encrypted files depending upon roles. We termed this integrated scheme as R-PEKS. The efficiency of R-PEKS over the PEKS scheme is up to 90%. We provide formal security proofs for the different components of R-PEKS and validate these schemes using a commercial dataset.

INDEX TERMS Access control, cloud computing, MUSE, PEKS, RBAC, SUSE.

I. INTRODUCTION

In the cloud, encryption may be a suitable mechanism to protect the data at rest. However, encryption prevents searching within the data which is essential for better usability of the encrypted data. This gives rise to a new area of research, called searchable encryption (SE). One problem of using SE in a symmetric setting is the maintenance of index, especially for applications where the dataset undergoes a frequent update. A variant of SE, called *public key encryption with keyword search* or PEKS is the most popular encryption technique which was introduced in [1] and is free from any index generation. Following this, much research has been carried out on *single-user searchable encryption* (SUSE) with *access control* mechanisms [2]. However, *multi-user searchable encryption* (MUSE) is becoming more relevant for most of the commercial applications involving a large group of users with complex access structures. Some work has also been done on MUSE by delegating the permission of searching among multiple users in an access controlled environment. Most of these works involve attribute-based access structure.

SE for keyword search yields huge outputs for most of the commercial applications which deal with a large dataset. Most of these outputs are not intended, which gives rise to unnecessary network traffic. SE for *string search* is of special interest as this customizes the search. By string, we refer to a sentence, which is an ordered sequence of words. Instead of a word search or disjunctive word search, search for ordered word sequence customizes the search to better precision. In this paper, we treat any word that occurs in plain text as a keyword and make no difference between word and keyword. Thus a string is an ordered sequence of keywords. In [3], non-adaptively secure SE for string search was proposed, but the approach was in a symmetric key setting. In [3] authors pointed out why it is impossible to design adaptively secure SE in symmetric search. In [4], the authors introduced the idea of *string search* using PEKS which is adaptively secure. All existing PEKS schemes are based on pairing-based cryptography where the basic component is *bi-linear mapping*.

The associate editor coordinating the review of this manuscript and approving it for publication was Petros Nicopolitidis.

One problem with this technique, especially in the context of string search, is the huge computational cost which makes it less suitable for commercial applications [5]. In this paper, we introduce a new adaptively secure PEKS scheme, which is free from *bi-linear mapping* and compatible with role based access control (RBAC) mechanism for secure search and access of outsourced encrypted data.

Recently, most of the researchers applied SE with *attribute based access control* (ABAC) to provide restricted access based on the attributes for their outsourced personal health record (PHR). In PHR datasets, changing privileges (i.e., user-permission assignments) are minimal [6], [7]. Since ABAC is the most suitable approach for these cases, protecting PHR data in the cloud by the means of attribute based keyword search over encrypted data is the most preferred choice of researchers [5], [8]–[10].

In big organizations, a large number of employees access data under a complex access structure. MUSE finds its application in such cases. If the search needs to be customized using string search, PEKS is the best possible solution for that. Since the access structure is subjected to frequent modifications, the concept of RBAC is considered to be the most suitable access control mechanism [7]. For example, wireless networks are secured with RBAC in small and medium-sized businesses. Here without RBAC, the ad-hoc process of granting and revoking network privileges as well as access to users becomes extremely difficult to manage especially when the number of employees increases beyond a certain point. Thus to design access control in MUSE for frequently changing string search privileges, the combination of RBAC with PEKS is considered to be optimal. Additionally, RBAC has minimal overhead when a large number of employees enter and exit the organization. Here we describe an industrial application where our model can provide the most optimal solution, but due to the confidentiality agreement, the company's identity is not disclosed.

As per the PCI DSS (Payment Card Industry Data Security Standard), the credit/debit card information should not be stored in plain text. So companies use a hashing technique (SHA-256) to store the card information in the database. So during the transaction when the user enters his card details, based on the hash value of the card details the information such as name and address are auto-filled.

It may be noted that the use of hashing technique is also exposed to the risk of statistical attack. In our model, even if the adversary gets the trapdoor he cannot search unless he gets access to the search algorithm. Migrating from hashing to PEKS comes with the additional cost but to make the searching process efficient, we introduce role based PEKS, i.e., R-PEKS for such situations. To the best of our knowledge, no secure searching scheme is available using PEKS with RBAC.

In this paper, we consider the following three real-world scenarios where PEKS, integrated with RBAC, is a good choice for commercial applications. We coin the term **R-PEKS** for it. We validate and compare our model with existing schemes using the methodologies proposed in [11].

Scenario 1: An organization wants to outsource its large amount of data to the cloud service provider (CSP). Such data can be accessed by the set of employees who are privileged by their roles to search.

Scenario 2: When an employee P wants to get access to certain data of some employee Q who is working under him, then the privilege is given to P to get access to Q's data.

Scenario 3: When a group G is formed from a subset of employees who are having different roles, then the group level privilege is given to all members for accessing data pertaining to the group.

Most of the threat models assume that the data owner and data user are trusted. However, the cloud service provider (CSP) is considered as *honest-but-curious* [3], [4], [12]. Recently, in MUSE, the threat model was developed by considering that data users are colluding with the CSP where both are *honest-but-curious* [13]. We develop R-PEKS under this threat model.

Main contributions of this paper are as follows:

1) Our proposed model is at the intersection of RBAC and PEKS, which facilitates data owner to provide restricted data search and access based on the RBAC configuration. We implement access control in three different modes depending on the above scenarios, namely, *single-user* (Scenario 1), *multi-user peer to peer* (Scenario 2) and *multi-user group* (Scenario 3).
2) We introduce a new PEKS scheme which, unlike the previous schemes, is free from bi-linear mapping and is more practical and efficient compared to the earlier scheme of [4] by 97% but yet providing an equivalent level of security.
3) With normal PEKS, the search request of a user u is spread over the whole dataset of the organization, of which a limited portion is meant for the access of the user u. In such PEKS, the access control is handled manually after the search. In this context, R-PEKS enforces the access control during the search and the efficiency of R-PEKS over the PEKS scheme is up to 90% when the part of dataset pertaining to user u is 10% of the whole cloud data.

Rest of the paper is organized as follows: In Section II we discuss the related work. Section III highlights the architecture of our proposed model R-PEKS. Section IV defines the R-PEKS components and Section V describes the R-PEKS design. In Section VI we discuss the security analysis of different components of R-PEKS. In Section VII we provide experimental results. In Section VIII we focus on comparison with other schemes and Section IX concludes the paper and outlines areas for future work.

II. RELATED WORK

Sensitive data especially patient's electronic health records in cloud storage can be protected by combining traditional

public key encryption scheme with access control [14]. The drawback of traditional encryption techniques is that it cannot provide privacy-preserving keyword searches on encrypted data. Further, in [14], the mathematical function used for the encryption scheme is bi-linear mapping which has a huge computational cost and makes it inefficient for lightweight applications.

Access control mechanisms with the SUSE scheme provide restricted access to data based on *roles*, *keys*, and *attributes*. In role based access, i.e., role based encryption of [15], the security for cloud storage is enabled by the data owner where he encrypts data using some role and public parameters. In the case of key based access control of [16], the accessible file's decryption key is given to the users. Integrating such a model with SE increases the complexity of the key management when the user accesses a large number of files. Recently, in [17]–[20], an attribute based SE scheme was implemented which result in huge computational cost because of the security assumption, which is based on BDH (Bi-linear Diffie-Hellman) assumption due to the usage of bi-linear mapping. Further, in [2], [9], [10], *fine-grained access control* and *multi-field* search query were implemented in SE with ABAC during file access in the cloud. The file in the cloud is attached with an encrypted index to label the keywords and access policy. So the user can get access to the file if the keyword is matched and the access policy mechanism grants the permission. Further, it allows the user to locally derive the search capability. String search can be viewed upon as ordered multiple keyword searches. Few schemes were proposed for string search in symmetric searchable encryption (SSE) [3] and PEKS [4], but both these schemes are without access control mechanism.

Access control mechanisms with MUSE schemes are constructed using *broadcast encryption*, along with *attributes*, *coarse-grained* and *fine-grained* access control. The key problems identified in MUSE are the key management and access control. Broadcast encryption [21] was the first mechanism to introduce access control in a multi-user environment, where the message is encrypted by broadcaster and can be decrypted only by those users who are part of the broadcast channel. Further, based on the broadcast encryption technique, coarser-grained access control was introduced into MUSE in a cloud environment in [22]. In [23], Key-Aggregate Searchable Encryption (KASE) was introduced in cloud storage for decreasing the key management during data sharing where each document was encrypted with a different key. This was implemented by generating a single key called key-aggregate in MUSE. In [24], single or multi-keyword search based on attributes and access control were implemented, which is known as *fine-grained access control*. In [25] multi-keyword search along with authorization in the multiple user setting was studied, where the authorization was meant for a specified period of time. Work has also been done to improve search efficiency among multiple users using SSE for cloud applications [26]. In recent past, PEKS scheme was combined with ABAC to provide restricted access on *personal health record* (PHR). In [8], the authors designed PEKS integrated with ABAC to support multi-keyword search in multi-user settings. In [5], the expressive SE scheme in PEKS was studied and developed for out-sourced PHRs. In this work, the authors treated keyword search predicates as access policies and express them as a conjunction or any other boolean expression of keywords. Further, in [5], *bi-linear mapping* in prime-order group was used to make it somewhat secure compared to *bi-linear mapping* in the composite-order group. It may be noted that ABAC in PEKS is not a suitable mechanism when access policies are changing quite often. This is because, with every change in the policies, the data owner needs to download, decrypt and re-encrypt the data [6]. In [13], data users colluding with the CSP is considered as a mandatory requirement of MUSE. Moreover, in SE, most of the threat model focused only on privacy against honest-but-curious CSP and data users [13], [22].

In light of the above discussions, it is clear that an access control mechanism in SE is very much essential to maintain a large number of privileged access. Such privileged access in SE should be conformable for future analysis and change in an efficient way. In this paper, we propose R-PEKS based on RBAC and a new PEKS. Unlike the earlier ABAC based schemes, R-PEKS is more suitable for applications where the change of user-permission assignments is a frequent activity. Also, the underlying PEKS in R-PEKS is free from *bi-linear mapping* and thus make it lightweight and more efficient compared to earlier schemes. In Table 1, we provide a comparison of our scheme with the existing models. It may be noted that our PEKS security relies on CDH (Computational Diffie-Hellman) assumption whereas in [4], [5] the security assumption was based on the BDH assumption. CDH is a very popular hard problem and many state-of-the-art schemes are based on CDH assumptions. For example, authors in [27], [28] also used CDH as the basis of their security proof.

III. ARCHITECTURE OF OUR PROPOSED MODEL - R-PEKS

The design of single-user and multi-user R-PEKS settings for secure access of cloud data is given in Figure 1. The **data owner** of Figure 1 is the enterprise which can grant or deny a certain set of permissions to its users. Here *encryption* and *searching* are the two operations on a dataset which constitutes *permission*. Therefore to maintain the restricted access on such permissions, the data owner maintains the role manager. In Figure 1, the *searchable encryption mode* component enables SUSE and MUSE as two different modes for searching the string request based on the privileges in the PEKS domain. Key pairs are generated for each authorized user in *Key generation* component. A **data user** may be an individual or group with limited data scope who can request for string search. CSP in the **Public cloud** is responsible to store the encrypted data and to execute the search algorithm on the cipher using the trapdoor and the public key. For the security reason, the roles that are assigned to users (i.e., user-role

TABLE 1. Properties of different searchable encryption schemes.

Property	KSAC [9]	SSE [3]	PEKS [4]	Expressive keyword search [5]	This paper
Search	multi-keyword	string	string	multi-keyword	string and multi-keyword
SE scheme	SSE	SSE	PEKS	PEKS	PEKS
Mathematical tool	HPE	index using secure hash chain	bi-linear mapping in elliptic curve group	bi-linear mapping in prime order group	exponentiation on prime order multiplicative group
Security assumption	RDSP and IDSP	secured index based on AES and SHA-256	BDH	BDH	CDH
SE setting	single-user	single-user	single-user	multi-user	single and multi-user (peer to peer and group)
Access control	ABAC	no	no	ABAC	RBAC
Security	keyword privacy	search pattern privacy	keyword privacy	keyword privacy	keyword privacy and hosted data confidentiality

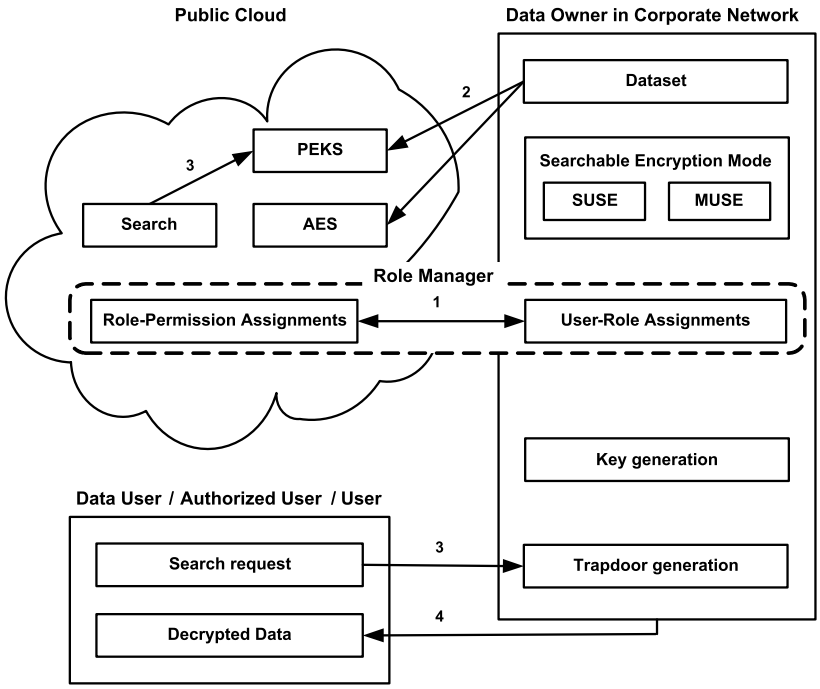

FIGURE 1. Design of the R-PEKS settings for secure access of cloud data.

assignments), as well as permissions (i.e., role-permission assignments), are maintained across the **corporate network** and **public cloud** respectively. Therefore, along with the data owner, the CSP should also authorize the users to access the defined permissions, i.e., The CSP performs a string search using PEKS based on the role-permission assignments. The detailed description of the activities are given below:

1) Initially, the data owner generates RBAC configuration, key pair for all the authorized users and enables two different modes of searchable encryption, i.e., SUSE and MUSE.
2) Data owner encrypts using PEKS and AES (see Figure 1) and stores the dataset using single-user and multi-user R-PEKS in public cloud.
3) The trapdoors are generated by the data owner on receiving the string search request and *mode* of R-PEKS from the data user. Such trapdoors are used by the CSP in the public cloud to perform a selective search based on RBAC configuration and identify the required files from the dataset.
4) Finally, the CSP returns the ID of the files that match the search to the data owner, which then

decrypts the corresponding AES-encrypted files using AES-decryption for the data user.

Remark 1: It may be noted that PEKS encryption is an one-way function as searchable ciphers produced by PEKS encryption cannot be decrypted. It is used only for searching using the trapdoor. In the R-PEKS, to retrieve the plain text file, we encrypt the plain text files using PEKS as well as AES and maintain a map between these two encrypted file pointers. Whenever some PEKS encrypted file pointer is detected by the search algorithm, the corresponding AES encrypted file is sent to the data owner for decryption and retrieval of the plain text file.

Remark 2: We maintain a map since AES encryption and decryption components of R-PEKS are straight forward, we skip this component while describing R-PEKS in the following sections.

IV. R-PEKS COMPONENTS

A. PEKS: DEFINITIONS AND PRELIMINARIES

1) 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 $\mathrm{D} \subseteq \mathbb{P}(\triangle)$ be the collection of n documents $\mathrm{D} = (D_1, D_2, \ldots, D_n)$. Let $id(D_i)$ be the unique identifier for the document D_i. We denote the list of all n document identifiers in D by $id(\mathrm{D})$, i.e., $id(\mathrm{D}) = \{id(D_1), \ldots, id(D_n)\}$. Furthermore, let $D(w_j)$ be the collection of all documents in D containing the word w_j. A string s of l words is an ordered tuple $(w_1, w_2, \ldots, w_l)$. Let $\mathrm{D}(s)$ denote a collection of documents in D that contains the string s. It is easy to check that $\mathrm{D}(s) \subseteq \bigcup_{i=1}^{l} D(w_i)$. We denote by $\delta(\mathrm{D})$, all the distinct keywords connected to the document collection D.

2) CRYPTOGRAPHIC PRIMITIVES

Here we define cryptographic primitives that are needed for our PEKS 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 $\lambda > a$, $negl(\lambda) < \frac{1}{p(\lambda)}$ [29]. We also use *pseudo prime number generator* [30], 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 $x \leftarrow S$.

3) 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_Enc(A_{pub}, w): For a public key A_{pub}, and a word w, this algorithm produces 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 $C = PEKS\_Enc(A_{pub}, w')$ and a trapdoor t_w, this algorithm outputs 'yes' if $w = w'$, otherwise 'no'.

Two cryptographic hash functions are used in our scheme Π_{PEKS}, namely H_1 and H_2 which are as follows: $H_1 : \{0,1\}^* \times \{0,1\}^* \mapsto \mathbb{Z}_p$, $H_2 : \{0,1\}^* \times \mathbb{Z}_p \mapsto \{0,1\}^\lambda$. For our implementation, we take G as $\mathbb{Z}_p$. Thus, for a word $w \in \{0,1\}^*$, $H_{1\langle k_2, k_1\rangle}(w) = H_{1\langle k_2\rangle}\left(H_{1\langle k_1\rangle}(w)\right) \in \mathbb{Z}_p$.

B. OUR PEKS SCHEME

In this section, we present our PEKS scheme Π_{PEKS} for string search.

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

1. KeyGen(1^λ) : KeyGen is a probabilistic key generation algorithm that is run by the data owner to setup the scheme. It takes a security parameter λ, and returns the setup. Since KeyGen is randomized, we write it as $(a, k_1, k_2) \leftarrow \mathrm{KeyGen}(1^\lambda)$. The public key is $pk = \{k_2\}$ and the secret key is $sk = \{a, k_1\}$ where $a \leftarrow \mathbb{Z}_p$.
2. PEKS_Enc(k_1, k_2, a, D_i) : PEKS_Enc is a probabilistic algorithm run by the data owner to generate the ciphertext C_i corresponding to document D_i. Since PEKS_Enc is deterministic, we write this as $C_i = PEKS\_Enc(k_1, k_2, a, D_i)$.
3. Trapdoor(k_1, k_2, a, s) : Trapdoor is a deterministic algorithm run by the data owner to generate trapdoors for a given string of words $s = (w_1, w_2, \ldots, w_l)$. It takes k_1, k_2, 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 $t = Trapdoor(k_1, k_2, a, s)$.
4. Search(k_2, C, t) : Search is run by the CSP 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 $\mathrm{D}(s)$, 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

Require: security parameter λ.
Ensure: a, k_1 and k_2.
$p \leftarrow PPNG(1^\lambda)$;
Set $a \leftarrow \mathbb{Z}_p$;
$(a, k_1, k_2) \leftarrow KeyGen(1^\lambda)$;

Remark 3: To encrypt the document D_i, we read the whole document as a stream of words and form an ordered sequence$(w_1, w_2, \ldots, w_l)$. A string is a subsequence of this sequence which is encrypted by applying PEKS_Enc on

Algorithm 2 PEKS_Enc

Require: a, k_1, k_2 and $\mathrm{D} = (D_1, \ldots, D_n)$.
Ensure: $\mathrm{C} = (c_1, \ldots, c_n)$.
Form a collection $\mathrm{W} = \{w_1, \ldots, w_d\}$ 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;
  $j \leftarrow 1$;
  $wp \leftarrow 0$;
  while (!EOF) do
    read j-th string in $s_j = (w_1, \ldots, w_m)$;
    $k \leftarrow 1$;
    while $k \leq m$ **do**
      $A = H_{1\langle k_2, k_1\rangle}(w_k)$;
      $B = H_{2\langle k_2\rangle}(A^{a*(k+wp)})$;
      Append B in C_i;
      $k \leftarrow k + 1$;
      $wp \leftarrow wp + 1$;
    end while
    $j \leftarrow j + 1$;
  end while
  $i \leftarrow i + 1$;
end while

every word in the string. If the same word occurs multiple times, for each instance the encrypted footprint will be different depending on the position (denoted by k in the algorithm) of the word in the string.

Algorithm 3 Trapdoor

Require: $s = (w_1, w_2, \ldots, w_l), k_1, k_2, a$.
Ensure: $t = (t_1, \ldots, t_l)$.
$j \leftarrow 1$;
while $j \leq l$ **do**
  $A_j = H_{1\langle k_2, k_1\rangle}(w_j)$;
  $t_j = A_j^a$;
  $j \leftarrow j + 1$;
end while

Remark 4: To search in a cipher, we first read all the ciphers of form $[B]$ in a list called *listB*. To find the match of the first trapdoor, i.e., t_1, we check it against each entry of *listB*. This is done in the second while loop. If a match is found, then the index of that block is stored in *start_pointer* and the rest of $l - 1$ trapdoors are checked against next $l - 1$ blocks in *listB* starting from *start_pointer* $+ 1$. If a match is found in all l successive steps, we add the file pointer in *encrypted_file_pointer* and go for the next file. If the match fails in any step we repeat the matching of t_1 for the remaining blocks in the same way until the file is exhausted.

Remark 5: In the PEKS, we assume that the data owner creates the data, encrypt using PEKS and uploads to the public cloud for future search. So the KeyGen and PEKS_Enc are run by the data owner. To search, the query is converted

Algorithm 4 Search

Require: $t = (t_1, \ldots, t_l), \{C_1 \ldots, C_n\}, k_2$.
Ensure: *encrypted_file_pointers*, a list of encrypted document pointers;
exists = *false*; *counter* $\leftarrow 0$;
$i \leftarrow 1$;
while $i \leq n$ **do**
  read all blocks of the form $[B]$ from C_i and arrange B's in *listB*;
  start_pointer $= -1$;
  $m \leftarrow 1$;
  while ($m \leq$ *listB.length*) **do**
    compute $chk = H_{2\langle k_2\rangle}(t_1^m)$);
    if ($chk ==$ *listB*$[m]$) **then**
      start_pointer $= m$;
      exists $=$ *true*;
      counter $= 1$;
      break;
    else
      $m = m + 1$;
    end if
    if (*counter* $= 1$) **then**
      $k \leftarrow 2$;
      while ($k \leq l$) **do**
        if $(H_{2\langle k_2\rangle}(t_k^{start\_pointer+counter})$ == *listB*[*start_pointer* + *counter*]) **then**
          counter $\leftarrow$ *counter* $+ 1$;
        else
          exists $=$ *false*;
          break;
        end if
        $k \leftarrow k + 1$;
      end while
    end if
    if (*exists* $=$ *true*) **then**
      add i to *encrypted_file_pointers*
    end if
  end while
end while

to trapdoor by the data owner and is given to the cloud. CSP runs the search using these trapdoors along with the public key of the data user. In R-PEKS, the data owner is the data administrator who is responsible for creating employee accounts and provide employees role based access to search on the encrypted data stored in the cloud. So, in R-PEKS, the data owner runs KeyGen for each employee/user and provides them with their keys. Further, the data owner fetches the keys of the corresponding employee and encrypt the files based on the assigned roles. To search, employees request data owner for the legitimate trapdoor for a given search query. Search is run by the CSP only on the privileged files using the trapdoor and the public key of the corresponding data. The search results are sent to the data owner who further decrypts and redirects the files to the employee.

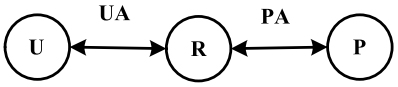

FIGURE 2. NIST RBAC [31].

Remark 6: Unlike traditional public key encryption scheme, in PEKS the data owner uses both the secret key and public key for encryption and trapdoor generation. Therefore data owner does not allow anybody to take the ownership of encryption as well as trapdoor generation. The data owner delegates only the search process, where the third party can search by using the data owner's public key without having any dependency on the secret key.

In the next lemma, we study the correctness of the search algorithm.

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 $C_i \leftarrow PEKS\_Enc(k_1, k_2, a, D_i)$. Let $[B_1], \ldots, [B_l]$ are the encrypted blocks in C_i for the string s taken in order. Then Search(k_2, C, t) 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., $id(D_i) \in Search(k_2, C, t)$.

Proof: Let $A_1 = H_{1\langle k_2,k_1\rangle}(w_1)$. for some j. Note that, then $B_1 = H_{2\langle k_2\rangle}(A_1^{am})$ for some integer m depending on the position of the word w_1 in the file. Now,

$$\begin{aligned} H_{2\langle k_2\rangle}(t_1^m)) &= H_{2\langle k_2\rangle}(H_{1\langle k_2,k_1\rangle}(w_1)^a)^m) \\ &= H_{2\langle k_2\rangle}(A_1^{am}) \\ &= B_1 \end{aligned}$$

This detects $[B_1]$ for t_1. Two consecutive blocks B_j and B_{j+1} are detected for two consecutive words w_j and w_{j+1} if there are two consecutive numbers, say k and $k+1$, such that $B_{j+1} = H_{2\langle k_2\rangle}(A_{j+1}^{a(k+1)})$, and $B_j = H_{2\langle k_2\rangle}(A_j^{a(k)})$, where $A_{j+1} = H_{1\langle k_2,k_1\rangle}(w_{j+1})$ and $A_j = H_{1\langle k_2,k_1\rangle}(w_j)$. Hence the result follows. ■

Remark 7: It may be noted that the second while loop of string search algorithm (i.e., Algorithm 4) is responsible for maintaining the ordering of keywords which is essential to detect string. We have also achieved a multi-keyword search by relaxing this condition for ordering. For the experimental results, we have used the variant which is dealing with string search only, i.e., Algorithm 4. Therefore our PEKS scheme Π_{PEKS} can search string and multi-keyword based on the user's request. In order to perform a selective search, we integrated our PEKS scheme Π_{PEKS} with the RBAC model (i.e., R-PEKS). The detailed explanation about RBAC components and functions which are crucial for integrating with our PEKS scheme Π_{PEKS} are given in Section IV-C.

C. RBAC MODEL

The NIST RBAC reference model defines different RBAC elements, RBAC assignments and mapping functions [31]. The pictorial representation of the NIST RBAC model is shown in Figure 2. Let us define some of the elements, assignments and functions which are crucial for R-PEKS. Let U denote a set of users. Let R denote a set of roles where each role is a job assigned to some users in an organization. Let P denote a set of permissions where each permission is an approval to perform an operation on an object. Formally we express this by $P = 2^{OP\times OB}$, where OP is the set of operations and OB is the set of objects. Let S_u denote a set of users who authorize the user u to search and access data pertaining to them. Let G denote a set of authorized users, i.e., a group. Let **UA** denote a many-to-many mapping which is user-to-role assignment relation, i.e., **UA** $\subseteq U \times R$. Let **PA** denote a many-to-many mapping which is a permission-to-role assignment relation, i.e., **PA** $\subseteq P \times R$. Also from [31], **assigned_permissions:** $R \rightarrow 2^P$ is a mapping function from role to a set of permissions. So, for some $r \in R$, **assigned_permissions(r)** = { p ∈ P | (p, r) ∈ **PA** }.

RBAC Configuration. Let **UP** denote a many-to-many mapping of user-to-permission assignment relation i.e., **UP** $\subseteq U \times P$. Also let us consider the function **RoleMining** which on input **UP**, outputs **UA** and **PA**, i.e., **RoleMining**: $U \times P \rightarrow$ { **UA**, **PA** }. The access control on RBAC can be defined as a *CheckAccess* function which is responsible for the authorization process. The function is defined as *CheckAccess*: $U \times P \rightarrow P$. *CheckAccess* takes a user u and the set of all permissions and returns the legitimate set of permissions pertaining to user u through roles. Formally, $CheckAccess(u, P) = \{p : p \in P \wedge \forall r, (u, r) \in \mathbf{UA} \wedge (p, r) \in \mathbf{PA}\}$. Similarly, we define *CheckGroupAccess*: $G \times P \rightarrow P$, where G is a group of users who may have different roles. Through *CheckGroupAccess* the permission is granted for all members of the group depending on the role assigned to the group.

V. R-PEKS DESIGN

In this section, we present our R-PEKS for single-user and multi-user settings. *Those components of Π_{PEKS}, that are reused in R-PEKS schemes as it is, are not described here. For these components of Π_{PEKS}, which are modified with access control functionality, we use the metacharacter "*" for the same set of parameters used in the corresponding component in Π_{PEKS}. Also, we modify the name of such components by prefixing the original name used in Π_{PEKS} by R.*

A. R-PEKS SCHEME

Single-user R-PEKS scheme is a SUSE in R-PEKS, where user u is authorized to search in the encrypted domain restricted by the assigned role. Further, the construction is given below.

Scheme 2: A **single-user R-PEKS scheme**, Π_{suse}, is a collection of five polynomial-time algorithms (KeyGen, R-PEKS_Enc, Trapdoor, R-Search, *CheckAccess*) such that:

1. R-PEKS_Enc(∗, *CheckAccess*(u, .)): R-PEKS_Enc is a deterministic algorithm run by the data owner to generate the ciphertext C_i for D_i, if the permission set P_u, obtained from *CheckAccess*, allows access to the D_i.

2. R-Search($*$, *CheckAccess*(u, .)): Search is run by the CSP in order to search only for the privileged documents in D which are accessible under the permissions obtained by *CheckAccess*.

Remark 8: In a single-user R-PEKS scheme, the data owner performs only the selective encryption and CSP performs only the selective search based on the roles assigned to the user using *CheckAccess*.

Multi-user R-PEKS: peer to peer scheme is a MUSE in R-PEKS. Here user u is authorized to search in the encrypted domain of users in S_u. The construction is given below.

Scheme 3: A **multi-user R-PEKS: peer to peer scheme** Π_{musePP} is a collection of seven polynomial-time algorithms (KeyGen, R-PEKS_Enc, AddUser, RevokeUser, Trapdoor, R-Search, *CheckAccess*) such that:

1. AddUser(u, U): AddUser is a deterministic algorithm run by the data owner to add a new user. It is assigned with a new user $u' \in U$ and updates S_u.
2. RevokeUser(u, U): RevokeUser is a deterministic algorithm run by the data owner to remove an existing user. It is revoked with the existing user $u' \in U$ and updates S_u.
3. R-Search($*$, *CheckAccess*(u, .)): Search is run by the CSP in order to search only for the privileged documents in D which are accessible under the permissions obtained by *CheckAccess* over S_u.

Remark 9: In multi-user R-PEKS: peer to peer scheme, the requisite permission is given by the data owner to search on others data is managed by AddUser and RevokeUser to add and revoke the user's privilege respectively. The algorithms for AddUser and RevokeUser simply denote many-to-many mapping which is user-to-user assignment relation, i.e., $\mathbf{UU} \subseteq U \times U$. RevokeUser just updates such assignments for user $u \in U$ by revoking the assignment pertaining to the existing user $u' \in U$. Criteria for revoking user-to-user assignment using RevokeUser is done if there is any change in the existing privilege of users. The data owner performs the selective encryption for users based on the assigned roles using the keys generated for the individual users. Finally, CSP performs the selective search for the authorized user who has the requisite permission to search on others data.

Multi-user R-PEKS: group scheme is a MUSE in R-PEKS. Here group G is authorized to search in the encrypted domain of some users (i.e., a subset of U) depending on the role assigned to G.

Scheme 4: An **multi-user R-PEKS: group scheme** Π_{museG} is a collection of seven polynomial-time algorithms (R-KeyGen, R-PEKS_Enc, Group_AddUser, Group_RevokeUser, Trapdoor, R-Search, *CheckGroupAccess*) such that:

1. R-KeyGen(1^{λ}) : R-KeyGen is a probabilistic key generation algorithm that is run by the data owner to setup the scheme for the group G.
2. R-PEKS_Enc($*$, *CheckGroupAccess*(G, .)): R-PEKS_Enc is a deterministic algorithm run by the data owner to generate the ciphertext C_i for D_i, if the permission set P_G, obtained from *CheckGroupAccess* allows access to the D_i.
3. Group_AddUser and Group_RevokeUser are the deterministic algorithms run by the data owner to add and remove a user from the group respectively. Since these functions are very much similar to AddUser and RevokeUser, it is not shown explicitly.
4. R-Search($*$, *CheckGroupAccess*(G, .)): Search is run by the CSP in order to search only for the privileged documents in D which are accessible under the permissions obtained by *CheckGroupAccess* over group G.

Remark 10: In multi-user R-PEKS: group scheme, the group which is a subset of users are created by the data owner. Key generation, group privileges and managing the group is done by the data owner. Further, the data owner performs the selective encryption based on the assigned roles for the group using the generated keys. Finally, based on the request the CSP performs the selective search based on the assigned roles for the group.

Remark 11: We observe that R-PEKS is independent of the underlying PEKS, i.e., Π_{PEKS} in a sense that any PEKS system can be used to install R-PEKS. It may be noted that in R-PEKS for single-user, i.e., Π_{suse}, the encryption is done using underlying PEKS encryption whenever there is a permission which is derived from role based structure given by *CheckAccess(.)* (See Scheme 2). Once the permission is given, the encryption is done using encryption algorithm of Π_{PEKS}, i.e. *PEKS_Enc*(.). A similar analysis holds for Search(). Thus the idea of R-PEKS can be instantiated with respect to any arbitrary PEKS system. For example, in subsection VII-B, for the analysis purpose, RBAC is instantiated with PEKS proposed in [4] under the name *r-PEKS*. To the best of our knowledge, prior to this work, the PEKS scheme of [4] was the only adaptively secure PEKS scheme for string search and so we select this scheme for comparison.

VI. SECURITY ANALYSIS

Threat model. In the R-PEKS scheme, we consider both CSP and data users as *honest-but-curious*. Under this assumption, CSP can infer additional privacy information, i.e., role-permission assignments and other information related to the *search pattern* and *access pattern* of the data [21]. Further, the malicious data users may collude with CSP to access unauthorized files by tweaking the roles. Unlike [24], the key and role management is handled by the data owner. So in our model, CSP cannot collude with any revoked malicious users in accessing the unauthorized privileges.

In the next two subsections, we discuss the security of R-PEKS and its components. In Subsection VI-A, we show that our PEKS is adaptively secure under the CDH assumption. In Subsection VI-B, we show that our R-PEKS system ensures data confidentiality by secure access.

A. SECURITY OF PEKS

In this section, we show that Π_{PEKS} is secure against adaptive string attack. The basic idea behind an adaptive string attack is that the adversary $\mathcal{A}$ is allowed to ask for PEKS encryptions of multiple strings chosen adaptively. The definition of security requires that $\mathcal{A}$ should not be able to distinguish the PEKS encryption of two arbitrary strings of same length, even when $\mathcal{A}$ is given access to *PEKS_Enc*() and *Trapdoor*() oracle. 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 2 (Game_Adaptive_Generic$_{\mathcal{A},\pi}(1^k)$*):*

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*(.) 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*(.) 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_Generic*$_{\mathcal{A},\pi}(1^k) = 1$, we say that $\mathcal{A}$ succeeded.

Since our scheme deals with two cryptographically strong hash functions, namely, H_1 and H_2, we would like to extend Definition 2 by giving oracle access of these two hash functions to an adversary, which leads to the modified security definition as given below.

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.

It is easy to observe that if a scheme is secure under Definition 3, then it is also secure under Definition 2. This is because from the point of view of an adversary, if oracle access of these two hash functions is withdrawn from Definition 3, then this becomes Definition 2. Also reducing these two oracle access amounts to reduce the degree of freedom of an adversary in terms of fetching information from oracle and thus adversary cannot get more information than what he could with Definition 3. We record this trivial fact in the following theorem without proof.

Theorem 1: If a PEKS scheme is secure under Definition 3, then it is also secure under Definition 2.

So it suffices to show that our scheme is secure under Definition 3. Consider the following definition which is based on Definition 3.

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_{PEKS} is adaptively secure. The proof relies on the hardness of *Computational Diffie-Hellman Problem* (CDH). Now we provide a variant of CDH problem for a multiplicative group which is suitable for our scheme.

1) COMPUTATIONAL DIFFIE-HELLMAN PROBLEM (CDH)

Let $(\mathbb{G}, \circ)$ be a multiplicative abelian group. Also let g be the generator of $\mathbb{G}$ and let $x, y \in \mathbb{Z}$ such that $a = g^x$, $b = g^y$ and $c = g^z$. The CDH problem is as follows:

given a, b, g as input, compute g^{xyz}. CDH is said to be intractable if all polynomial time algorithms have a negligible advantage in solving CDH.

Suppose $\mathcal{A}$ is a polynomial size adversary that has advantage $negl(\lambda)$ in breaking Π_{PEKS}, i.e., $Pr[Game\_Adaptive_{\mathcal{A},\Pi_{PEKS}}(1^k) = 1] \leq \frac{1}{2} + negl(\lambda)$. 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. Here we construct a simulator $\mathcal{S}$ and in Theorem 2 we show that $\mathcal{A}$ solves the CDH problem with probability at least $\epsilon' = \frac{negl(\lambda)}{e \times n_T \times n_{H_2}}$, where e is the base of the natural logarithm. The construction of simulator and the proof technique of Theorem 2 is similar to that of [1] but in a different setting on *strings* instead of *keywords* and also the proof is done against CDH assumption.

2) THE SIMULATOR $\mathcal{S}$

Let g be the generator of $\mathbb{G}$. Let the simulator $\mathcal{S}$ is given g, $p_1 = g^a$, $p_2 = g^b$, $p_3 = g^c$, $p_4 = g^{ac}$. The simulator $\mathcal{S}$ simulates the challenger and interacts with the adversary $\mathcal{A}$ as follows:

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

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 $H_1^*(w_i) = h_i \in \mathbb{G}$.
   - 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 $h_i = p_2 \times g^{a_i} \in \mathbb{G}$, otherwise $h_i = g^{a_i} \in \mathbb{G}$.
   - 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 $H_1^*(w_i) = h_i \in \mathbb{G}$.
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 $t_i = p_1^{a_i}$.

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 $0 \le i \le 1$, $1 \le j \le m$. 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 $1 \le k \le m$.

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

III $\mathcal{S}$ responds to the challenge $\{[B_1], [B_2], \ldots, [B_m]\}$ where $\mathcal{S}$ choses B_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.

Before going into the Theorem 2, here we will prove some inequalities which are crucial for the theorem.

Lemma 2: Let us consider the following events :

ε_1 : The event denoting $\mathcal{S}$ does not abort while $\mathcal{A}$ is making trapdoor queries.

ε_2 : The event denoting $\mathcal{S}$ does not abort while $\mathcal{A}$ is making challenge.

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

Then, (1) $P[\varepsilon_1] \ge \frac{1}{e}$, (2) $P[\varepsilon_1] \ge \frac{1}{n_T}$ and (3) $P[\varepsilon_3] \ge 2 \times negl(\lambda)$.

Proof:

1. Since in $list_1$, the distribution of c_i's are independent of the distribution of h_i's, we have $P[onetrapdoorquery\ triggeringabort] = \frac{1}{(n_T+1)}$. So,
$P[\varepsilon_1] = (1 - P[\text{one trapdoor query triggering abort}])^{n_T}$
$= \left(1 - \frac{1}{(n_T+1)}\right)^{n_T} \ge \frac{1}{e}$.
2. $P[\mathcal{S}$ will abort during challenge$] = P[c_{i,k} = 1 : i \in \{0, 1\}, k \in \{1, \ldots, m\}] = \left(1 - \frac{1}{n_T+1}\right)^{2m} \le 1 - \frac{1}{n_T}$.
So, $P[\varepsilon_2] \ge 1 - p[\mathcal{S}$ will abort during challenge$] = \frac{1}{n_T}$.
3. Let ε_3 be the event denoting $\mathcal{A}$ queried against at least one of the strings s_0 and s_1. $P[\ \mathcal{A}$ breaks the scheme$] = P[b = b']$
$= P[(b = b')|\varepsilon_3]P[\varepsilon_3] + P[(b = b')|\varepsilon_3']P[\varepsilon_3']$
$\le \frac{1}{2}P[\varepsilon_3] + \frac{1}{2}$.
From Definition 4, $negl(\lambda) \le |P[b = b'] - \frac{1}{2}| = \frac{1}{2}P[\varepsilon_3]$.
Thus $P[\varepsilon_3] \ge 2\epsilon$.

■

Theorem 2: Π_{PEKS} *is adaptively secure against chosen keyword attack in the random oracle model assuming CDH is intractable.*

Proof: It may be noted that the challenge implicitly defines B_1 as $H_2^*\left(H_1^*(w_{b,1})^{ac}\right)$. Now,

$$B_1 = H_2^*\left(H_1^*(w_{b,1})^{ac}\right) = H_2^*\left(g^{(b+a_b)ac}\right)$$

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 $\frac{t}{(p_4)^{a_b}}$ as its guess for g^{abc}, 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 $t = g^{ac(b+a_b)}$. 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 $\frac{t}{(p_4)^{a_b}} = \frac{g^{ac(b+a_b)}}{g^{a_b ac}} = g^{abc}$. It is easy to check that $P[\mathcal{S}$ solves CDH $] = P[\varepsilon_0] \times P[\varepsilon_1] \times P[\varepsilon_2] \times P[\varepsilon_3]$. So, from Lemma 2, $P[\mathcal{S}$ solves CDH $] \ge \frac{negl(\lambda)}{e \times n_T \times n_{H_2}}$. ■

3) DISCUSSION ON ADAPTIVE SECURITY OF Π_{PEKS}

The basic idea behind the security proof of Theorem 2 is that the adversary $\mathcal{A}$ is allowed to ask for PEKS encryptions and trapdoors of multiple keywords and strings chosen adaptively. This is formalized by allowing $\mathcal{A}$ to interact freely with an encryption and trapdoor oracle. So from $\mathcal{A}$'s point of view, encryption and trapdoor functionality comes as a *black-box* that encrypts keywords and strings of $\mathcal{A}$'s choice using the secret key which is unknown to $\mathcal{A}$. Since keywords are mapped into finite field elements using the hash function H_1 and the final output of PEKS encryption maps the encrypted keyword to $\{1, 0\}^*$ using the hash function H_2, $\mathcal{A}$ is also provided with the oracle access of these two hash functions. When $\mathcal{A}$ queries its oracle by providing it with a keyword as input, the PEKS encryption oracle returns a ciphertext as the reply. Since PEKS encryption is randomized, the oracle uses fresh random coins each time it responds to a query. The definition of security requires that $\mathcal{A}$ should not be able to distinguish the encryption of two arbitrary keywords or strings of same lengths, even when $\mathcal{A}$ is given access to *PEKS_Enc*(.), *Trapdoor*(.), H_1(.) and H_2(.). The security of our scheme relies on the CDH assumption, i.e., so long as the CDH assumption holds, we need to show that $\mathcal{A}$ cannot win the game defined in Definition 3 with a probability much more than $\frac{1}{2}$. In the proof, we constructed a simulator $\mathcal{S}$, who simulates the challenger in such a way so as to solve CDH. We have shown that if adversary $\mathcal{A}$ wins the game defined in Definition 3, then $\mathcal{S}$ solves the CDH in non-negligible probability. Thus if CDH is believed to be insolvable, reasonably we may infer that $\mathcal{A}$ cannot win the game of Definition 3.

B. SECURITY OF R-PEKS: DATA CONFIDENTIALITY BY SECURE ACCESS

In this section, we show that R-PEKS ensures hosted data confidentiality, i.e., protection of access privilege during string search.

Definition 5: R-PEKS is said to have secure access under the injection of faulty roles if R-PEKS access only the files within the scope of the user's privilege during string search.

In the next theorem, we prove that our R-PEKS access only the files based on the roles during string search which ensures hosted data confidentiality by secure access.

Theorem 3: Even when the data user colludes with a CSP by the injection of faulty roles, they are unable to access the hosted data by string search over any file that is not in the scope of the data user's assigned roles/privileges

Proof: In R-PEKS, let P be the set of all permissions. Let $r_i \in R_u$ and $r_j \notin R_u$. Also let user u access the permission set $P(r_i)$ through the assigned role set R_u , i.e. $CheckAccess(u, P) = P(r_i)$. So, $P(r_i) = \{p : p \in P, (u, r_i) \in \mathbf{UA} \wedge (p, r_i) \in \mathbf{PA}\}$.

From the role mining algorithm, there is a threshold δ, such that $|\mathbf{assigned\_permissions(r}_i) \cap \mathbf{assigned\_permissions(r}_j)| \leq \delta$. Therefore the similarity rate ranges from 0 to δ. According to [32], we carry out the security mutation analysis by focusing on fault injection of roles, done by CSP while colluding with the data user, as given below :

Original search request of user u is served by the data owner by presenting this request as R-Search $(*, CheckAccess(u, P))$. Since $r_i \in R_u$, R-Search $(*, CheckAccess(u, P))$ should be transformed into R-Search$(*, P(r_i))$.

Under the fault injection assumption, let r_i be replaced by CSP to a faulty role, say r_j. Thus original search request, i.e., R-Search$(*, CheckAccess(u, P))$ transforms in to R-Search$(*, P(r_j))$, where $P(r_j) = \{p : p \in P, (u, r_j) \in \mathbf{UA} \wedge (p, r_j) \in \mathbf{PA}\}$. There are two possibilities:

Case 1: $|\mathbf{assigned\_permissions(r}_i) \cap \mathbf{assigned\_permissions(r}_j)| > 0$

Case 2: $|\mathbf{assigned\_permissions(r}_i) \cap \mathbf{assigned\_permissions(r}_j)| = 0$

In Case 1, string search is performed on files common to r_i and r_j. In Case 2, no operation is performed. Suppose the actual number of files identified for a string search on a user without faulty injection is $\mathbb{F}$. Also let us assume that the same number of files are identified for successive *k-1* searches on the same string. Finally, let the number of identified files for the same string search caused by the fault injection of roles on the k^{th} iteration of search is δ. If files returned after search are the files that are accessible according to role r_i, we call it a *success*. If $\mathbb{T}$ is the expected number of files satisfying a particular search string after k iterations, then we define *score* S, on *success* as $S = \frac{(k-1)*\mathbb{F}+\delta}{\mathbb{T}}$. It is easy to check that $S \leq 1$. This is because searches for user u are performed using trapdoors generated from u's private key on data encrypted by u's public key. So from the PEKS privacy, under no situations, the files corresponding to the permissions [$\mathbf{assigned\_permissions}(r_j) - \mathbf{assigned\_permissions}(r_i)$] will respond to the search, even if they contain the query string as these files are not encrypted using u's public key. Thus these files never contribute in S.

If $S < 1$, then the system is under attack by the injection of a faulty role that affected the string search. If $S = 1$, then $\delta = \mathbb{F}$ and it is a fuzzy state. In such cases, the string search is not affected even though the system is under attack. Therefore in all the above cases, the R-PEKS scheme is secure under Definition 5, which ensures a high level of hosted data confidentiality when data user collude with the CSP by injecting the faulty role during the string search. ■

VII. EXPERIMENTAL RESULTS

In this section, we present an experimental set-up by generating the RBAC model on the PEKS environment. We also provide the performance analysis of our proposed model on the TIMIT dataset [33]. Finally, we validate the correctness of our model, i.e. the data confidentiality using *test automation framework* [34]. The implementation is done on Intel Core(TM) *i5−7500* with 16 GB RAM using Java in Windows platform. For the cryptographic primitives, 'Jpair' library is used.

A. CREATION OF RBAC CONFIGURATION USING RMINER

The creation of RBAC configuration using RMiner [35] is achieved in two phases. Firstly, we generate **UP** (see Subsection IV-C) and secondly, we generate roles and its assignments to users as well as permissions, based on the **UP**. The required objects to generate a **UP** are taken from the TIMIT speech corpus [33], which is a phonemically and lexically transcribed speech of American English speakers of different dialects. This dataset is comprised of 42 distinct phonetic symbols giving rise to an average of approximately 200 words in 4607 files in a document of size 19 MB. For the experimentation purpose, we considered file-level access, i.e., each file is treated as an object and each object is assigned permission, which is either 1 (grant) or 0 (deny) under two operations, namely encryption and search. It is assumed that if a file has permission for encryption then it also has permission for a search. Further, we have incorporated 500 users in the implementation. The users and files are given as an input to RMiner [35], which is a role mining tool used for the generation of **UP** and for the creation of RBAC configuration, i.e., **UA** and **PA** (see Subsection IV-C). The generated **UP** for the given number of users and permissions is listed in Table 2. Also, in Table 2, the parameters presented in fourth, fifth, sixth and seventh rows are given as input and the rest is generated by the RMiner tool.

We use HPRoleMinimization algorithm of RMiner, which provides the most compact and optimal solution in the creation of RBAC configuration compared to other algorithms in the RMiner tool. The total time taken by RMiner algorithm is 46.43 seconds.

Remark 12: R-PEKS system is independent of the underlying dataset as long as the language of the dataset has a finite number of alphabets and can be recorded and stored in the cloud. It may be noted that the image dataset can be stored in the cloud although its atomic data blocks are pixels, such datasets are not conformable to R-PEKS. R-PEKS trivially works on plain English text files. In this paper, we choose the TIMIT dataset to test and verify that R-PEKS can also work on a nontrivial dataset. Another reason for selecting such data lies in the fact that there is a growing trend in the voice data

TABLE 2. Statistics of the generated RBAC components for TIMIT speech dataset [33] using RMiner.

Input/Output	Values
Total number of users	500
Total number of files	4607
Generated |UP|	11,26,308
Mean of permissions in role	25
Variance of permissions in role	10
Mean of roles in user	10
Variance of roles in user	5
Generated |UA|	313
Generated |PA|	27,857
Generated |R|	285

processing industries. In these type of organizations, they gather and store voice data over a public cloud for various commercial applications and research, where access to voice data (i.e., privilege) can be subject to frequent modifications. Such applications lack in privacy-preserving access to voice data, especially in a multi-user environment without compromising data confidentiality, data integrity, and anonymity of users.

B. PERFORMANCE ANALYSIS ON PEKS AND R-PEKS

All performances are analyzed based on the average time taken by 5 different users to perform encryption and search. These are illustrated on the speech dataset by comparing with the existing models. For the analysis purpose, RBAC is instantiated with PEKS proposed in [4] and further, it is referred to as *r-PEKS*.

1) PERFORMANCE OF PEKS

In the next lemma, we provide a complexity analysis of our scheme.

Lemma 3: Using Π_{PEKS}, the number of group operations for searching a query of l-word string in one ciphertext document C is $O\left(\left(\frac{|C|}{\lambda}\right)+(l-1)\right)$.

Proof: It is easy to observe that each block of the form $[B]$ is of size λ bits. Thus in the encrypted document C, the number of blocks is $\frac{|C|}{\lambda}$. To detect the first block requires $O\left(\frac{|C|}{\lambda}\right)$ group operations. Since the blocks are not shuffled, to detect rest of the $l-1$ blocks, $l-1$ group operations are needed. Thus the number of group operations is $O\left(\left(\frac{|C|}{\lambda}\right)+(l-1)\right)$. ■

Figure 3 compares the average PEKS encryption time for our PEKS scheme against PEKS of [4]. We plotted the graph by considering the average time in seconds along Y-axis against the number of files needed to encrypt along X-axis. For both schemes, the graph reflects a linear growth with an increase in the number of files. However, the average time taken by our PEKS scheme is 1 second to 5 seconds to encrypt 50 files (on an average of size little over $200kB$) to

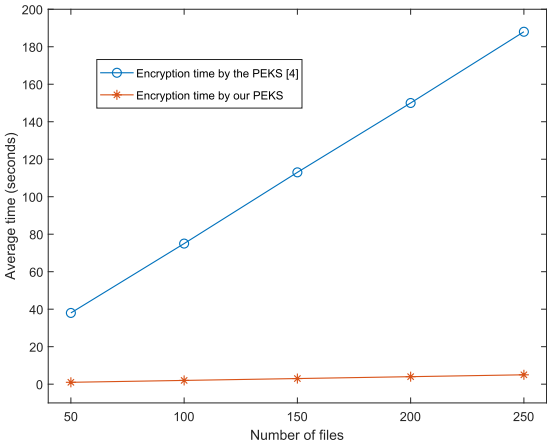

FIGURE 3. Comparison of encryption time for our PEKS and PEKS [4].

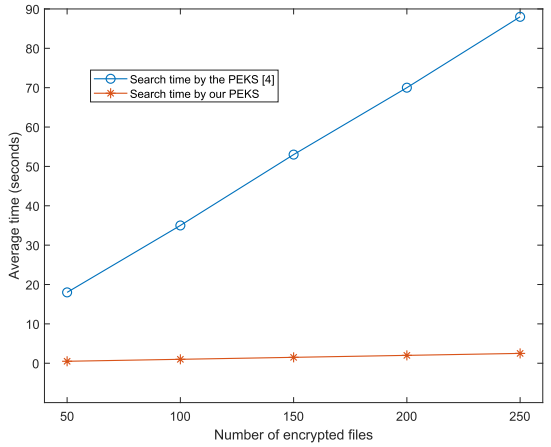

FIGURE 4. Comparison of search time for our PEKS and PEKS [4].

250 files (on an average of size little over $1MB$) which is very much less compared to the existing PEKS [4] where it took 38 seconds to 188 seconds for the same operation.

Figure 4 represents the performance of searching over encrypted files. We plotted the average time of search (in seconds along Y-axis) against the number of encrypted files (along X-axis). The graph shows a linear growth with an increase in the number of files. The average time taken by our PEKS scheme to search over 50 to 250 encrypted documents is 0.5 seconds to 2.5 seconds which is 97% efficient compared to the performance of PEKS of [4], where they took 18 seconds to 88 seconds for the same search operation.

2) PERFORMANCE OF R-PEKS AND PEKS

We note that both the PEKS schemes can have a better performance when integrated with RBAC. Now, we discuss the enhancements of R-PEKS over our PEKS. In Figure 5, we provide a comparison of R-PEKS and our PEKS search performance.

We plotted the graph by considering the average time of search (in seconds along Y-axis) against the available number of files (along X-axis) using R-PEKS and our PEKS scheme.

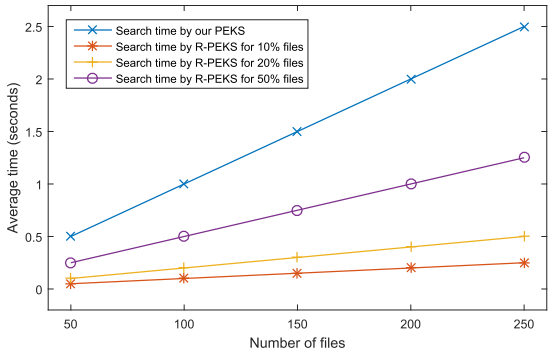

FIGURE 5. Comparison of search time for R-PEKS and PEKS.

In all the cases, the graph shows a linear growth with an increase in the number of files, however, the rate of growth reflects the efficiency of R-PEKS against the PEKS scheme. In PEKS, since the search is not refined by privileges (i.e., by roles), the search is performed among all the existing files. So the average search time using our PEKS scheme for any percentage of matching files is 2.5 seconds on 250 files. Since R-PEKS is a selective search, the string search request is performed only on the privileged files. So the average search time using R-PEKS is 0.25 seconds, 0.5 seconds and 1.25 seconds when the assigned roles contain 10%, 20% and 50% privileged files over 250 files respectively. So effectively from the user's point of view, R-PEKS will be faster compare to PEKS. This explains the efficiency of R-PEKS over our PEKS by 90% when the required data is present only in 10% files over 250 files.

3) PERFORMANCE OF DIFFERENT PEKS SCHEMES WITH RBAC

In [11], the authors proposed the idea of generating synthetic datasets to evaluate the performance of role mining algorithms. Towards this, we have generated the two synthetic datasets presented in Table 3 and Table 4 to compare the performance of R-PEKS. Both the synthetic datasets consist of five parameters, namely, number of users (#U), number of roles (#R), number of permissions for each user (#P), maximum number of permissions for a role and scope of privileges (range of files). In the first synthetic dataset, the user's scope of privileges is a varying parameter and other parameters are constant as shown in Table 3 and the corresponding comparison study is presented in Figure 6. In the second synthetic dataset, the user's permission is a varying parameter and other parameters are constant as shown in Table 4 and the corresponding comparison study is presented in Figure 7.

In Figure 6 and Figure 7 we plot the average time (along Y-axis) needed by the users to encrypt the documents and to access the data by searching (along X-axis) based on the values tabulated in Table 3 and Table 4 respectively. Figure 6 reflects a linear growth of the average time of encryption against the scope of privileges (range of files). Figure 6 shows that for R-PEKS, average time varies from 0.5 seconds to 1.7 seconds, whereas for r-PEKS, the average time varies from 8 seconds to 9.2 seconds over the scope of privileges varying from 50 to 250 for a constant 10 number of permissions. In the case of searching, R-PEKS takes constant average time of 0.1 seconds whereas r-PEKS takes 3.6 seconds respectively under the same scope of privileges and permissions as mentioned in Table 3. Therefore, the above comparison reveals that the efficiency of R-PEKS over r-PEKS during encryption and search are 81% and 97% respectively.

Figure 7 reflects a linear growth with the increase in the number of permissions for a constant scope of privileges. Further, the average time needed for encrypting 50 files in a scope of 80 privileges using R-PEKS and r-PEKS is around

TABLE 3. Users with the varying scope of privileges for a constant number of permissions.

#U	#R	#P	Maximum number of permissions for a role	Scope of privileges
500	100	25	5	50
500	100	25	5	100
500	100	25	5	150
500	100	25	5	200
500	100	25	5	250

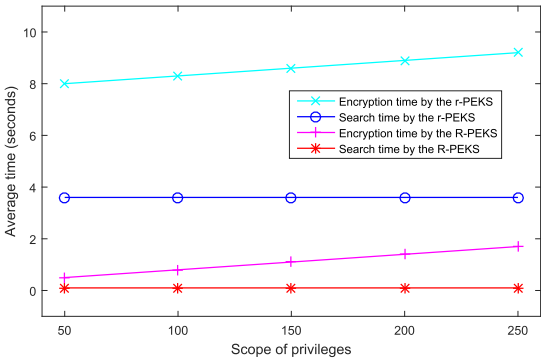

FIGURE 6. Comparison of encryption and search time for r-PEKS and R-PEKS on the varying scope of privileges for a constant number of permissions.

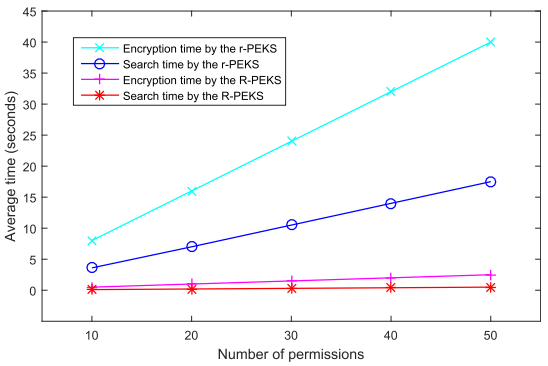

FIGURE 7. Comparison of encryption and search time for r-PEKS and R-PEKS on a varying number of permissions for a constant scope of privileges.

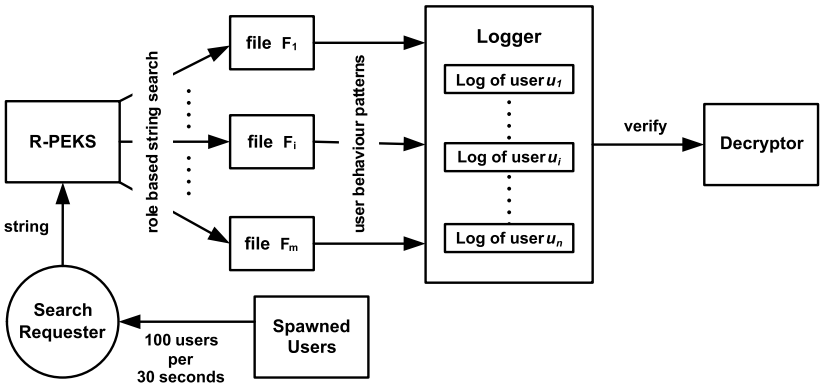

FIGURE 8. Test automation framework to determine the correctness of role based file access for string search using R-PEKS.

TABLE 4. Users with varying number of permissions for a constant scope of privileges.

#U	#R	#P	Maximum number of permissions for a role	Scope of privileges
500	100	10	5	80
500	100	20	5	80
500	100	30	5	80
500	100	40	5	80
500	100	50	5	80

2.5 seconds and 40 seconds respectively. The average time needed to search the data over the 50 encrypted files using R-PEKS and r-PEKS is 0.5 seconds and 17.5 seconds respectively. Above comparison reveals that the efficiency of R-PEKS over r-PEKS during encryption and search are 93% and 97% respectively.

Remark 13: The basic difference between the encryption scheme of [14] and R-PEKS is that our scheme is SE where one can directly search in encrypted data, whereas [14] uses traditional encryption where such searching is not possible. Since comparing performances of traditional encryption with SE won't reveal much, we did the comparisons between our scheme and another BDH based PEKS scheme.

4) AUTOMATION OF SECURE ACCESS BY R-PEKS

In this section, we determine the functional correctness of our proposed model on accessing only permitted files using R-PEKS in a single-user and multi-user environment. To validate this experimentally, we have carried out the testing by developing a test automation framework, which can spawn 100 parallel users having different roles in every 30 seconds as shown in Figure 8. In the *logger* block of Figure 8, the user behavior patterns are saved for analysis.

To determine the correctness, 10 different users are retrieved randomly from the log and all the user behavior patterns are monitored. After one hour of the experiment, the data that are saved in the logger are studied which reveals not a single instance of unauthorized access.

VIII. COMPARISON WITH OTHER SCHEMES

A. COMPARISON WITH SCHEME IN [36]

In [36], the authors proposed a MUSE scheme with efficient access control for cloud storage, where the keyword index and trapdoor can be generated with the help of a proxy server. To achieve this authors constructed a new MUSE scheme, where the keyword index and trapdoor can be generated with the help of an additional proxy server. It may be noted that in our scheme, we need not generate any such index and thus is free from the additional index management and is also free from the risk of leakage from the index.

Also in [36], the core mathematical construct for the searchable encryption is bi-linear mapping, which is a computationally intensive module. It may be noted that the scheme proposed in this paper is the PEKS scheme which is free from such bi-linear mapping. To the best of our knowledge, this is the only such PEKS scheme which is free from bi-linear mapping and is much more efficient and easy to implement.

The search complexity of the scheme proposed in this paper is the number of finite field operation which is linear in $|C|$ for a cipher file of size $|C|$. Although the complexity analysis of the search is not explicitly mentioned, but from Algorithm 4 of [36], it is clear that search complexity is $O(n)$ where n is the number of keywords in a cipher file. Since we are doing sentence search where each word is of length λ, n in [36] is equivalent to $\frac{|C|}{\lambda}$, where $|C|$ is the size of the cipher file. So, when expressed in our notation, the search complexity of [36] is also linear in $|C|$ but operations are in the elliptic curve which is costly compared to finite field operations.

B. COMPARISON WITH SCHEME IN [37]

In [37] authors proposed a template of adaptively secure searchable encryption scheme with access control without any practical instantiation and so the comparison seems difficult. However, we observed that their scheme is based on the SSE of [21]. One problem in SSE is that adaptive security and string search cannot be achieved simultaneously.

In this paper, we emphasis on a customized search under three application scenarios by enforcing a string search without compromising the security. So the scheme of [37] cannot be adapted for the application scenarios which are considered in this paper.

IX. CONCLUSION AND FUTURE WORK

RBAC enabled PEKS is the most suitable and efficient solution for secure search applications in a multi-user setting where user-permissions assignments are updated frequently. Towards this, we have designed R-PEKS. We have designed a new PEKS, called Π_{PEKS} for this. In the threat model, we have considered honest-but-curious CSP as well as data users and have analyzed the security requirements. Finally, we have shown R-PEKS to be secure under the definition of adaptive security and also provides hosted data confidentiality by secure access. The experiment is conducted on the TIMIT speech dataset [33] to evaluate the performance of our proposed model. We have shown that with respect to the PEKS of [4] for string identification, our PEKS scheme, i.e., Π_{PEKS} is efficient by 97%. We have also shown that using R-PEKS, a user can gain up to 90% efficiency on searching when the share of data pertaining to him is 10% of the whole cloud data. For the comparison study, we also have implemented PEKS of [4] with RBAC and named it r-PEKS. With the generated synthetic dataset, we have shown that the efficiency of R-PEKS against r-PEKS is up to 87% for encryption and 97% for searching. We have developed the *test automation framework* and have determined the functional correctness of R-PEKS by it. It might be of interest to explore how PEKS can be integrated with dynamic user-permission assignments using the least privilege user-role assignment problem for better performance and security.

REFERENCES

[1] D. Boneh, G. Di Crescenzo, R. Ostrovsky, and G. Persiano, "Public key encryption with keyword search," in *Proc. 23rd Int. Conf. Theory Appl. Cryptograph. Techn.*, Cham, Switzerland, 2004, pp. 506–522.
[2] H. Yin, J. Zhang, Y. Xiong, L. Ou, F. Lil, S. Liao, and K. Li, "CP-ABSE: A ciphertext-policy attribute-based searchable encryption scheme," *IEEE Access*, vol. 7, pp. 5682–5694, 2019.
[3] I. G. Ray, Y. Rahulamathava, and M. Rajarajan, "A new lightweight symmetric searchable encryption scheme for string identification," *IEEE Trans. Cloud Comput.*, to be published. doi: 10.1109/TCC.2018.2820014.
[4] I. G. Ray and M. Rajarajan, "A public key encryption scheme for string identification," in *Proc. 16th IEEE Trustcom/BigDataSE/ICESS*, Sydney, NSW, Australia, Aug. 2017, pp. 104–111.
[5] H. Cui, Z. Wan, R. H. Deng, G. Wang, and Y. Li, "Efficient and expressive keyword search over encrypted data in cloud," *IEEE Trans. Depend. Sec. Comput.*, vol. 15, no. 3, pp. 409–422, May/Jun. 2018.
[6] L. Y. Zhang, Y. Zheng, J. Weng, C. Wang, Z. Shan, and K. Ren, "You can access but you cannot leak: Defending against illegal content redistribution in encrypted cloud media center," *IEEE Trans. Depend. Sec. Comput.*, to be published. doi: 10.1109/TDSC.2018.2864748.
[7] D. R. Kuhn, E. J. Coyne, and T. R. Weil, "Adding attributes to role-based access control," *IEEE Comput.*, vol. 43, no. 6, pp. 79–81, Jun. 2010.
[8] K. He, J. Guo, J. Weng, J. Weng, J. K. Liu, and X. Yi, "Attribute-based hybrid Boolean keyword search over outsourced encrypted data," *IEEE Trans. Depend. Sec. Comput.*, to be published. doi: 10.1109/TDSC.2018.2864186.
[9] Z. Shen, J. Shu, and W. Xue, "Keyword search with access control over encrypted cloud data," *IEEE Sensors J.*, vol. 17, no. 3, pp. 858–868, Feb. 2017.
[10] J. Benaloh, M. Chase, E. Horvitz, and K. Lauter, "Patient controlled encryption: Ensuring privacy of electronic medical records," in *Proc. 16th ACM Workshop Cloud Comput. Secur.*, Chicago, IL, USA, 2009, pp. 103–114.
[11] J. Vaidya, V. Atluri, J. Warner, and Q. Guo, "Role engineering via prioritized subset enumeration," *IEEE Trans. Depend. Sec. Comput.*, vol. 7, no. 3, pp. 300–314, Jul./Sep. 2010.
[12] Q. Wang, M. He, M. Du, S. S. M. Chow, R. W. F. Lai, and Q. Zou, "Searchable encryption over feature-rich data," *IEEE Trans. Depend. Sec. Comput.*, vol. 15, no. 3, pp. 496–510, May/Jun. 2018.
[13] C. Van Rompay, R. Molva, and M. Önen, "Secure and scalable multi-user searchable encryption," in *Proc. 6th Int. Workshop Secur. Cloud Comput.*, Incheon, South Korea, 2018, pp. 15–25.
[14] K. Riad, R. Hamza, and H. Yani, "Sensitive and energetic IoT access control for managing cloud electronic health records," *IEEE Access*, vol. 7, pp. 86384–86393, 2019.
[15] L. Zhou, V. Varadharajan, and H. Hitchens, "Achieving secure role-based access control on encrypted data in cloud storage," *IEEE Trans. Inf. Forensics Security*, vol. 8, no. 12, pp. 1947–1960, Dec. 2013.
[16] E.-J. Goh, H. Shacham, N. Modadugu, and D. Boneh, "SiRiUS: Securing remote untrusted storage," in *Proc. 10th Annu. Netw. Distrib. Syst. Secur. Symp.*, San Diego, CA, USA, 2003, pp. 1–25.
[17] S. Wang, S. Jia, and Y. Zhang, "Verifiable and multi-keyword searchable attribute-based encryption scheme for cloud storage," *IEEE Access*, vol. 7, pp. 50136–50147, 2019.
[18] S. Wang, J. Ye, and Y. Zhang, "A keyword searchable attribute-based encryption scheme with attribute update for cloud storage," *PLoS ONE*, vol. 13, no. 5, 2018, Art. no. e0197318.
[19] S. Wang, Y. Zhang, and Y. Zhang, "A blockchain-based framework for data sharing with fine-grained access control in decentralized storage systems," *IEEE Access*, vol. 6, pp. 38437–38450, Jun. 2018.
[20] S. Wang, D. Zhang, Y. Zhang, and L. Liu, "Efficiently revocable and searchable attribute-based encryption scheme for mobile cloud storage," *IEEE Access*, vol. 6, pp. 30444–30457, 2018.
[21] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky, "Searchable symmetric encryption: Improved definitions and efficient constructions," *J. Comput. Secur.*, vol. 19, no. 5, pp. 895–934, Jan. 2011.
[22] Z. Liu, Z. Wang, X. Cheng, C. Jia, and K. Yuan, "Multi-user searchable encryption with coarser-grained access control in hybrid cloud," in *Proc. 4th Int. Conf. Emerg. Intell. Data Web Technol.*, Xi'an, China, Sep. 2013, pp. 249–255.
[23] B. Cui, Z. Liu, and L. Wang, "Key-aggregate searchable encryption (KASE) for group data sharing via cloud storage," *IEEE Trans. Comput.*, vol. 65, no. 8, pp. 2374–2385, Aug. 2016.
[24] W. Sun, S. Yu, W. Lou, Y. T. Hou, and H. Li, "Protecting your right: Attribute-based keyword search with fine-grained owner-enforced search authorization in the cloud," in *Proc. 33rd Annu. IEEE Conf. Comput. Commun.*, Toronto, ON, Canada, Apr./May 2014, pp. 226–234.
[25] Y. Yang, X. Liu, and R. Deng, "Multi-user multi-keyword rank search over encrypted data in arbitrary language," *IEEE Trans. Depend. Sec. Comput.*, to be published. doi: 10.1109/TDSC.2017.2787588.
[26] G. Wang, C. Liu, Y. Dong, P. Han, H. Pan, and B. Fang, "IDCrypt: A multi-user searchable symmetric encryption scheme for cloud applications," *IEEE Access*, vol. 6, pp. 2908–2921, 2017.
[27] W. Shen, J. Qin, J. Yu, R. Hao, and J. Hu, "Enabling identity-based integrity auditing and data sharing with sensitive information hiding for secure cloud storage," *IEEE Trans. Inf. Forensics Security*, vol. 14, no. 2, pp. 331–346, Feb. 2019.
[28] M. Liu, Y. Zhao, and S. Chen, "eCK-security authenticated key agreement protocol based on CDH assumption," in *Proc. 2nd IEEE Int. Conf. Comput. Commun.*, Chengdu, China, Oct. 2016, pp. 213–216.
[29] J. Katz and Y. Lindell, *Introduction to Modern Cryptography*. Boca Raton, FL, USA: CRC Press, 2014.
[30] D. R. Stinson, *Cryptography: Theory and Practice*. Boca Raton, FL, USA: CRC Press, 2005.
[31] R. Sandhu, D. F. Ferraiolo, and D. R. Kuhn, "The NIST model for role-based access control: Towards a unified standard," in *Proc. 5th ACM Workshop Role-Based Access Control*, Berlin, Germany, 2000.
[32] D. Xu, M. Tu, M. Sanford, L. Thomas, D. Woodraska, and W. Xu, "Automated security test generation with formal threat models," *IEEE Trans. Depend. Sec. Comput.*, vol. 9, no. 4, pp. 526–540, Jul./Aug. 2012.
[33] *TIMIT Acoustic-Phonetic Continuous Speech Corpus*. Accessed: 2007. [Online]. Available: http://www.fon.hum.uva.nl/david/ma_ssp/2007/timit/train/dr5/fsdc0/

[34] L. Wybouw-Cognard, "Test automation framework," U.S. Patent 9 925 872, Aug. 22 2002.
[35] R. Li, H. Li, W. Wang, X. Ma, and X. Gu, "RMiner: A tool set for role mining," in *Proc. 18th ACM Symp. Access Control Models Technol.*, Amsterdam, The Netherlands, 2013, pp. 193–196.
[36] Z. Lv, M. Zhang, and D. Feng, "Multi-user searchable encryption with efficient access control for cloud storage," in *Proc. 6th Int. Conf. Availability, Rel. Secur.*, Singapore, Dec. 2014, pp. 366–373.
[37] N. Löken, "Searchable encryption with access control," in *Proc. 12th Int. Conf. Availability, Rel. Secur.*, Reggio Calabria, Italy, 2017, Art. no. 24.

K. RAJESH RAO received the B.E. degree in computer science and engineering from Visvesvaraya Technological University, Belgaum, in 2008, and the M.Tech. degree in computer science and information security from the Manipal Institute of Technology, Manipal, India, in 2012. He is currently pursuing the Ph.D. degree. He was a Visiting Researcher with the Information Security Group, City, University of London, U.K., from 2018 to 2019. He is currently an Assistant Professor-Senior with the Department of Information and Communication Technology, Manipal Institute of Technology. His research interests include but is not limited to cloud security, access control models, and applied cryptography.

INDRANIL GHOSH RAY received the B.Sc. degree (Hons.) in mathematics from Calcutta University, in 2000, the MCA (Master of Computer Applications) degree from the University of Kalyani, in 2003, and the Ph.D. degree in computer science from the Indian Statistical Institute, in 2016. He worked in software industry for six years as a Software Engineer and a Senior Software Engineer. He has been a Research Associate with the Information Security Group, School of Engineering and Mathematical Sciences, City, University of London, U.K., since June 2016. His research interest includes but is not limited to homomorphic encryption and its application in privacy preserving, cloud computing, searchable symmetric encryption, public key encryption with keyword search, MDS codes and its applications in lightweight cryptography, and algebraic immunity of S-Boxes based on power mappings.

WAQAR ASIF received the B.Eng. and M.S. degrees from well-reputed institutions in Pakistan, in 2009 and 2012, respectively, and the Ph.D. degree from the Department of Electrical and Computer Engineering, City, University of London, in 2016, where he is currently a Research Fellow. Before moving to U.K., he was a Lecturer with Bahria University, Pakistan, for a year. He secured multiple merit-based scholarships, which include an Erasmus Mundus STrongTies Scholarship for one-and-half years for Cyprus, where he was a Researcher with Frederick University. His research interests include but is not limited to graph theory, sensor networks, network performance metrics, blockchain, and network privacy.

ASHALATHA NAYAK received the B.Tech. and M.Tech. degrees in computer science and engineering from Mangalore University, Karnataka, India, and the Ph.D. degree from the School of Information Technology, IIT Kharagpur, in the area of model-based testing. She is currently a Professor and the Head of the Department of Computer Science and Engineering, Manipal Institute of Technology, Mahe, Manipal, India. Her research interests include semantic web, software testing, intelligent agents, and cloud security.

MUTTUKRISHNAN RAJARAJAN is currently a Professor of security engineering with the City, University of London, U.K., where he currently leads the Information Security Group. He is a Visiting Researcher with the British Telecommunication's Security Research and Innovation Laboratory. His research interests include privacy-preserving data analytics, cloud computing, the Internet of Things security, and wireless networks. He has published well over 300 articles and continues to be involved in the editorial boards and technical programme committees of several international security and privacy conferences and journals. He is an Advisory Board Member of the Institute of Information Security Professionals, U.K., and acts as an advisor to the U.K. Government's Identity Assurance Programme (Verify U.K.).