IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2017.DOI

Novel Multi-Keyword Search on Encrypted Data in the Cloud

YUNYUN WU[1], JINGYU HOU[2],JING LIU[3],Wanlei Zhou[4],(Senior member, IEEE), and SHAOWEN YAO[3]

[1]School of Information Science and Engineering, Yunnan University, Kunming, 650091, China
[2]School of Information Technology, Deakin University, Victoria 3125, Australia
[3]School of Software, Yunnan University, Kunming, 650091, China
[4]School of Software, University of Technology Sydney, Ultimo NSW 2007, Australia

Corresponding author: Shaowen Yao (e-mail: yaosw@ynu.edu.cn) and Jingyu Hou(e-mail: jingyu.hou@deakin.edu.au).

This work partially supported by the Natural Science Foundation of China (NSFC) under grant 61363084 and 61863036, and the China Scholarship Council (CSC) program.

ABSTRACT Searching on encrypted data has become a very important technique in cloud computing. Such searches enable the data owner to search on the encrypted data stored on the cloud without leaking any information. To obtain a better search experience, researchers have proposed many schemes which mainly focus on conjunctive and disjunctive keyword searches. However, a conjunction of all the keywords may result in very few results, whereas a disjunction will return too many results. With the current schemes, customizing the relevancy of the keywords to obtain the desired results is difficult. To solve these problems, we propose a novel scheme that supports the search with the user specified number of keywords contained in the search result. This number n can be used to customize the keyword relevancy. As a result, the data owner could obtain the desired search results containing any n keywords of a keyword set. The proposed scheme also supports the traditional disjunctive and conjunctive keyword searches when n equals 1 or the size of the keyword set, respectively. The keyword could be positive or negative. We first formally define its security, then prove that the proposed scheme is secure against the adaptive chosen keyword attack in the standard model and can defend against the offline keyword guessing attack to some extent. Furthermore, we present a theoretical performance comparison with other schemes, as well as the experimental performance evaluations on our implemented scheme.

INDEX TERMS Searchable Encryption, Multi-keyword Search, Cloud Security, Privacy Protection

I. INTRODUCTION

As cloud technology continuously develops, more companies are preferring to provide their services in the cloud. Therefore, it is very common for users to submit their private information to cloud servers for services. However, if the cloud server that hosts the private data from multiple users is untrusted, the data will be completely exposed to attackers. A common approach to solve this problem is to encrypt the data. Once the data have been encrypted, the server cannot reveal the data. Therefore, it is necessary to search the encrypted data without breaking the confidentiality of the data.

Public-key encryption with keyword search (PEKS) was first proposed by Dan Boneh [1] which enables to search on the encrypted data. In PEKS, a data sender generates a searchable ciphertext by using the intended data receiver's public key and stores it on the cloud. The data receiver could output a search token which is related to a keyword by means of his/her private key and sends a search request to the cloud server. The cloud server searches on the ciphertext with the search token and sends the relevant results back to the data receiver.

Early PEKS schemes [2] [3] focused almost exclusively on a single keyword search. To enhance the user's search experience, more and more expressive schemes that support conjunctive [4] [5] [6], disjunctive [7] [8] [9] [10], negative [11] keywords, range (such as greater-than) [12] and subset queries [13] [14] were also proposed. However, the existing PEKS schemes cannot well apply in such a scenario. Suppose a user wants to search for some information with a keyword set $K = \{A, B, C, D, E\}$. A conjunction of all the keywords may result in very few results, whereas a disjunction of these

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2019.2898751, IEEE Access

keywords may return too many results. In order to get the desired results, the user may search with a relevance number n to find the results which contain any n keywords of the keyword set K. Assume n=3, by just using conjunctions and disjunctions, the search expression would be:

$$(A \wedge B \wedge C) \vee (A \wedge C \wedge D) \vee ... \vee (C \wedge D \wedge E).$$

There are nine disjunctions and twenty conjunctions. This is a very laborious task. Therefore, it is worth computing the equation by one operation.

Contributions We solve the above problem by proposing a novel multi-keyword search scheme which is based on public-key encryption; thus, it is called the public-key encryption with expressive multi-keyword search (PEMKS). We denote a threshold n as the relevancy. By customizing the relevancy, the data owner can search for the results that contain any n keywords of a keyword set. The disjunctive and conjunctive keyword searches can be viewed as two special cases of the searches when n is equal to one or the size of the keyword set, respectively.

The proposed scheme borrows the idea of the attribute-based encryption [15] by representing the attributes as the keywords and the access policy as the search expression to enhance the search. In order to apply the idea in the proposed scheme, we first solve the problem of the keyword security (responding to the attribute security), and then improve the method to obtain the search results which contain any n keywords of a keyword set, rather than to get the boolean search results that the current schemes could support. In addition, the proposed scheme allows the keywords to be positive or negative, which enhances the search flexibility. A positive keyword means it is contained in the document, while a negative keyword means it is not contained in the document. At last, we define a security model based on PERKS [16] that can defend not only against the adaptive chosen keyword attack but also against the offline keyword guessing attack. Most of the existing PEKS schemes are under the offline keyword guessing attack. Since the adversary can generate the encrypted index for the keywords, it may determine the relationship between keywords and the search token received. Under the proposed security model, the adversary can only learn the structure of the expression tree rather than the information about keywords.

The main contributions of this work are summarized as follows:

1) An innovative encryption scheme that supports the user to customize the relevancy of the keywords and the server to obtain the search results with the relevancy by taking one operation.
2) A new definition of the semantic security model that defends against adaptive chosen keyword attacks. Under the defined security model, the proposed scheme can also defend against the offline keyword guessing attack.
3) A performance analysis that contains comparisons with other schemes, the implementation of the proposed scheme and the evaluation of its computational overhead and the storage overhead.

Organization The remainder of this paper is organized as follows. Section II provides a review of the related works. Section III presents some cryptographic notations and definitions. In section IV, we introduce the system model and security definitions. The details of the construction and security proof are presented in Section V. Section VI presents the performance analysis. Section VII concludes the paper.

II. RELATED WORK

Searchable encryption (SE) is a cryptographic primitive which enables the data owner to search on encrypted data without leaking any information about the data. SE is divided into two categories: the first one is symmetric searchable encryption (SSE), and the second one is public-key encryption with keyword search (PEKS). In SSE, the searchable ciphertext and search token are encrypted with the same key. Therefore, only the key holders could search on the encrypted data. In PEKS, any senders who have the receiver's public key could generate the searchable ciphertext, and the valid search token could only be computed by the specific receiver.

Song et al. [17] proposed the first SSE scheme in 2000. It was a full-text search scheme and suffered from the statistical attack. The subsequent schemes focus on the secure index-based method to improve the efficiency. The existing SSE schemes turn the attention to three points: security, efficiency and expressiveness. For security, Eu-Jin Goh [18] first define the security of indexes as the semantic security against the adaptive chosen keyword attack. However, it only keeps the index secure. To enhance the security of SSE, Curtmola et al. [19] comprehensively defined the security of the index, trapdoor, search pattern and access pattern. Many SSE schemes [20] [21] [22] are now based on this security model. For efficiency, Cash et al. [20] first improved the efficiency from linear to sub-linear on conjunctive keywords search. Kamara et al. [23] proposed a worst-case sub-linear complexity searchable symmetric encryption by using the improved method 2Lev arose by Cash [20]. It could remain the sub-linear time complexity for any keyword query. For expressiveness, most current schemes supported boolean [24] [23] and range [25] [26] queries.

The first PEKS scheme was proposed by Boneh et al. [1]. Subsequently, most following schemes make a tradeoff between security, efficiency and versatility of search criteria. For security, almost all the schemes can keep the index secure. However, the salient problem of PEKS is the offline keyword guessing attack. To solve this problem, the schemes largely fall into three classes. The first one [2] only allows the server to search the index with a secret key that is hosted by the server, and the adversary cannot be the server. The second one [16] [27] uses an authenticated keyword to limit the ability to generate the index with any keywords chosen by the adversary. The third one [8] constructs a fuzzy search

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2019.2898751, IEEE Access

trapdoor to defend against the attack under the condition that the keyword space is not the polynomial level. For efficiency, most existing schemes are linear complexity. P. Xu et al. [28] proposed a new index structure to support a parallel search with sublinear complexity. For versatility of search criteria, Early schemes [4] [5] could search with a conjunction of several keywords. In 2007, Boneh et al. [13] proposed a scheme that can implement conjunctive, subset, and range queries on encrypted data based on hidden vector encryption. Y. Zhang et al. [9] proposed a conjunctive and disjunctive multi-keyword scheme by using inner product. This approach is very simple and efficient, but the size of the vector must be as large as the size of the entire unique keyword set of all the data.

In recent years, attribute-based encryption (ABE) is widely applied in searchable encryption. Sahai and Waters [29] first defined the concept of attribute-based encryption as an extension of identity-based encryption. When a data user's attributes satisfy the access policy formulated by the data owner, the user could decrypt the ciphertext. Zheng et al. [30] arose the notion of attribute-based keyword search (ABKS) to enable the data sender to set the access policy of the data receivers for searching on his/her encrypted data. Following Zheng's work, Ameri et al. [31] proposed a scheme that ensures only the authorized users could search on the encrypted data which is generated at a certain time. Cui et al. [32] introduced a keyword search with an efficient revocation scheme by using ABE. It could apply in a multi-sender/multi-receiver scenario. ABE doesn't keep the attributes secure, while these ABKS schemes [30] [31] [32] [33] aim to achieve a fine-grained access control by using ABE. Therefore, the above ABKS schemes just defined the security for the keywords rather than the attributes. There are some other schemes [7] [8] [11] [34] implementing a boolean multi-keyword search by using ABE. They regarded the attributes and access policy as the keywords and search criteria, respectively. These schemes have a stronger secure definition for the attributes. In order to get a stronger security, these schemes used two fields to represent a keyword, where one is the keyword field and the other one is the keyword value. The keyword field is public, and the keyword value is secret. This method has a disadvantage that each field only could have a keyword value. Meanwhile, if the structure of the keywords is irregular, it is hard to set the keyword field.

In general, it is hard for the existing schemes to obtain the search results which contain any n keywords of a keyword set by one operation.

III. PRELIMINARIES

We begin by introducing some cryptographic notations and definitions that will be used in the construction.

A. COLLISION-RESISTANT HASH, BILINEAR MAPS AND COMPLEXITY ASSUMPTIONS

Collision-Resistant Hash [35] A hash function H is collision-resistant if for any polynomial-time adversary $\mathcal{A}$, there is a negligible function ε such that

$$\Pr[Hash - coll_{\mathcal{A},H}(n) = 1] \leq \varepsilon(n)$$

, where n is the value of the security parameter. For simplicity, denote H as a collision-resistant hash function.

Bilinear Maps [36] Let G_1 and G_2 be two groups of prime order p. A bilinear map $e : G_1 \times G_1 \rightarrow G_2$ between them satisfies the following properties :

1) Computable: given $g, h \in G_1$, there is a polynomial-time algorithm to compute $e(g, h) \in G_2$.
2) Bilinear: for any integer $x, y \in [1, p]$, we have $e(g^x, g^y) = e(g, g)^{xy}$.
3) Nondegenerate: if g is a generator of G_1, then $e(g, g)$ is a generator of G_2.

Decisional Bilinear Diffie-Hellman (DBDH) Assumption [36] Let $a, b, c, z \in \mathbb{Z}_p$ be chosen at random, and let g be a generator of G_1. The DBDH assumption is that no probabilistic polynomial-time algorithm β can distinguish the tuple $(g, A = g^a, B = g^b, C = g^c, e(g, g)^{abc})$ from the tuple $(g, A = g^a, B = g^b, C = g^c, e(g, g)^z)$ with more than a negligible advantage. The advantage of β is:

$$|\Pr[\beta(g, A, B, C, e(g, g)^{abc}) = 1] - \Pr[\beta(g, A, B, C, e(g, g)^z) = 1]|$$

, where the probability is taken over the random choice of the generator g, the random choice of $a, b, c, z \in \mathbb{Z}_p$, and the random bits consumed by β.

Multi-Decisional Bilinear Diffie-Hellman (MDBDH) Assumption [37] Let $a, b, c_1, ..., c_m, z_1, ..., z_m \in \mathbb{Z}_p$ be chosen at random and g be a generator of G_1. The MDBDH assumption is that no probabilistic polynomial-time algorithm β can distinguish the tuple $(g, A = g^a, B = g^b, C_1 = g^{c_1}, ..., C_m = g^{c_m}, e(g, g)^{abc_1}, ..., e(g, g)^{abc_m})$ from the tuple $(g, A = g^a, B = g^b, C_1 = g^{c_1}, ..., C_m = g^{c_m}, e(g, g)^{z_1}, ..., e(g, g)^{z_m})$ with more than a negligible advantage. The advantage of β is:

$$|\Pr[\beta(g, A, B, C_1, ..., C_m, e(g, g)^{abc_1}, ..., e(g, g)^{abc_m}) = 1] - \Pr[\beta(g, A, B, C_1, ..., C_m, e(g, g)^{z_1}, ..., e(g, g)^{z_m}) = 1]|$$

, where the probability is taken over the random choice of the generator g, the random choice of $a, b, c_1, ...c_m, z_1, ..., z_m \in \mathbb{Z}_p$, and the random bits consumed by β.

The MDBDH assumption has been proven to be equivalent to the DBDH assumption in [37].

B. EXPRESSION TREE

Before describing the expression tree, we present some notations of the parameters that will be used.

Notations Let $D = \{D_1, ..., D_n\}$ be an $n-$document collection. The identifier of document D_i is represented as id_i, $0 \leq i \leq n$, and the identifier could be any string that uniquely identifies the document. Let a keyword w contained in the document be a positive keyword, and a keyword which is not contained in the document be a negative keyword, denoted as w'. Let $W = \{w_1, w_2, ..., w_m\}$ be a set of distinct keywords

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2019.2898751, IEEE Access

that exist in the document collection D and $|W|$ be the size of the keyword set. Let W_i be the keyword set in a document D_i, and let $W_i \subseteq W$. The difference of two keyword sets A and B is defined as $A \bowtie B = (A - B) \cup (B - A)$.

Each record in an encrypted database consists of two parts: 1) an index I_i that is associated with a document D_i that contains the keyword information W_i, and 2) an encrypted document $C_i = Enc(D_i)$. The encrypted database is defined as:

$$EDB = \{(C_1, I_1), ..., (C_n, I_n)\}$$

, or $EBD = (C_i, I_i)_{i=1}^{n}$ for short.

Expression Tree Construction Let searching the encrypted data that contain any k keywords of a keyword set be an operation R_k, where k is the threshold value chosen by the data receiver. Let $ET(W_{ET}, O)$ be an expression tree converted from the search expression, where $W_{ET} = \{w_1, ..w_d\}$ is a set of keywords and O is a set of operations R_k on W_{ET} in the search expression. In an expression tree, each leaf node is associated with a positive keyword or a negative keyword, and each internal node x is associated with an operation R_{k_x}. Define num_x as the number of the children of an internal node x, k_x is an integer number, and $0 < k_x \leq num_x$. For calculation simplicity, the leaf node is also assigned with a threshold value $k_x = 1$. Let the parent of the node x in the tree be $parent(x)$, and let the identifier of node x in the tree be $index(x)$. Let ET be a structure of the expression tree without the information of keywords and threshold values. For example, if there is a search expression such as:

$$R_2(R_3(w_1, w_2, w_3), R_1(w_4, w_5), R_2(w_6, w_7, w_8, w_9')),$$

where $W_{ET} = \{w_1, w_2, w_3, w_4, w_5, w_6, w_7, w_8, w_9'\}$ and $O = \{R_2, R_3, R_1, R_2\}$, the expression tree $ET(W_{ET}, O)$ is as shown in FIGURE 1.

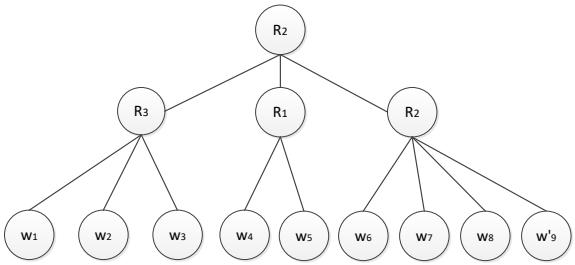

FIGURE 1: An Expression Tree

IV. SYSTEM MODEL AND SECURITY DEFINITIONS

In this section, we first introduce the system model, which contains the communication among the server, the data receiver and the data senders, as well as the definition of the proposed scheme, and then define the security of the proposed scheme.

A. SYSTEM MODEL

The following scenario shows how a correct PEMKS works. Suppose that there is a data receiver Alice, a data sender Bob, and an honest-but-curious server. When Bob wants to send some data to Alice, Bob should first obtain the pre-tag of the keyword from Alice in a secure way. Such communication channels between the senders and receiver are private in this phase. Then, Bob encrypts the data, generates indexes with pre-tags and sends them together to the server. When Alice wants to search for the required encrypted data, she could generate a trapdoor with an expression and send the trapdoor to the server. Once the server receives the search query with the trapdoor, it searches on the indexes to obtain matched encrypted data and sends them to Alice. All people, including the data receiver and data senders, could send the encrypted data and indexes to the server, but only the receiver can generate the trapdoor. This is a multi-sender/one-receiver system. Note that Alice does not search with a single keyword. She could search with any n keywords of a keyword set. At the same time, the server cannot learn anything about the keywords because the proposed scheme only sends the structure of the expression tree together with the trapdoor to the server. The procedure is shown in FIGURE 2.

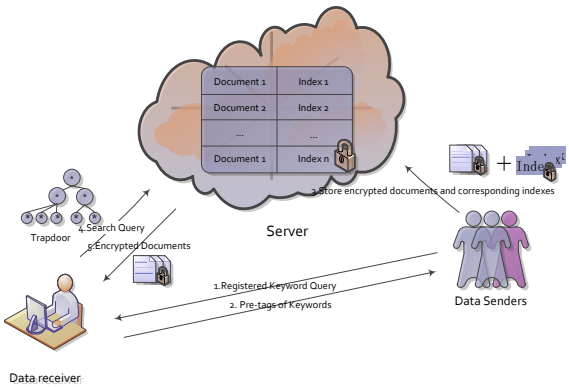

FIGURE 2: The Communication of PEMKS

The formal definition of the proposed PEMKS scheme is given as follows.

Definition 4.1 PEMKS on encrypted database EDB consists of five algorithms $PEMKS = (Gen, KeywordRg, BuildIndex, Trapdoor, Search)$ such that:

- $Gen(s)$: run by the receiver; it takes a security parameter s as the input, and it generates a public/private key pair (K_{pub}, K_{pri}).
- $KeywordRg(K_{pri}, w)$: run by the receiver; it takes the private key K_{pri} and a keyword w as the input, and it generates a pre-tag s_w.
- $BuildIndex(K_{pub}, S_W)$: a probabilistic algorithm and is run by the sender. It takes the public key K_{pub} and a pre-tag set S_W as input, and then generates an index I.
- $Trapdoor(K_{pri}, ET(W_{ET}, O))$: run by the receiver. It takes the private key K_{pri} and the expression tree $ET(W_{ET}, O)$ as input, and it generates the trapdoor T

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2019.2898751, IEEE Access

for searching.

- $Search(K_{pub}, T, I)$: a deterministic algorithm and is run by the server. It takes the public key K_{pub}, trapdoor T and index I as input. If I satisfies T, the server returns true and sends the corresponding encrypted data to the receiver; otherwise, it returns false.

In the traditional communication of Alice, Bob and the server, the data are encrypted by Alice's public key [1]. Alice's device may have limited resources for decrypting the encrypted data, such as a mobile phone. To solve this problem, we can encrypt the data with a symmetric key for efficiency. Then, Bob sends the symmetric key encrypted by Alice's public key together with the encrypted data to the server.

The correctness of the proposed scheme is introduced as follows which is based on [13].

Correctness For all indexes I and expression trees $ET(W_{ET}, O)$, P is a function that can check whether I satisfies $ET(W_{ET}, O)$ and is denoted as $P : I, ET(W_{ET}, O) \rightarrow (0, 1)$.
Let

$$(K_{pub}, K_{pri}) \xleftarrow{R} Gen(s), s_w \leftarrow KeywordRg(K_{pri}, w),$$
$$I \xleftarrow{R} BuildIndex(K_{pub}, S_W),$$
$$T \leftarrow Trapdoor(K_{pri}, ET(W_{ET}, O)).$$

There are two situations, as follows:

- If $P_{ET(W_{ET},O)}(I) = 1$, then $Search(K_{pub}, T, I) = true$.
- If $P_{ET(W_{ET},O)}(I) = 0$, then $\Pr[Search(K_{pub}, T, I) = false] > 1 - \varepsilon(s)$, where $\varepsilon(s)$ is a negligible function.

B. SECURITY DEFINITIONS

Inspired by the security models introduced in [15] [16] [13], the security is defined for the proposed scheme to ensure that it does not reveal any information about the keywords in the indexes. Formally, we define the security mainly against the chosen keyword attack and the offline keyword guessing attack:

- Chosen keyword attack [1]: The adversary $\mathcal{A}$ is allowed to query on a keyword he/she chooses, gets the responding trapdoor except the challenge keyword, and tests them. With these keywords and responding trapdoors, $\mathcal{A}$ could get some information of the keyword embedded in indexes.
- Offline keyword guessing attack [38]: Due to the fact that the keywords are chosen from a small space (polynomial size), the adversary $\mathcal{A}$ could generate the trapdoor with any keywords, and test it. With the relationship between keywords and the trapdoors, $\mathcal{A}$ can determine the keywords embedded in indexes.

In the above attacks, there are three types of adversaries in the proposed scheme: 1) a curious sender, who is able to potentially acquire pre-tags and indexes of any keyword he/she chooses; 2) an honest-but-curious server, who is able to potentially acquire all the trapdoors and indexes in an oblivious way; and 3) the server who is also a sender at the same time, is able to potentially acquire the pre-tags, indexes, and trapdoors.

The security of the proposed scheme is described as follows:

Setup: The challenger $\mathcal{B}$ takes a security parameter s as input and runs the Gen algorithm. $\mathcal{B}$ provides the adversary $\mathcal{A}$ with the public key K_{pub} while keeping the private key K_{pri} to itself. Then, $\mathcal{B}$ chooses a keyword set W that contains all the keywords that will be used in the game.

Query phase 1: The adversary adaptively chooses a finite number of queries in the following types of oracles.

- $KeywordRg$ oracle: the adversary adaptively chooses a finite number $q_{1,k}$ of keywords $W_{q_{1,k}} = \{w_1, ...w_{q_{1,k}}\} \subseteq W$ and sends them to $\mathcal{B}$. $\mathcal{B}$ runs the $KeywordRg(K_{pri}, w_i), 1 \leq i \leq q_{1,k}$ and sends pre-tags $\{s_{w_1}, ..., s_{w_{q_{1,k}}}\}$ to the adversary.
- $BuildIndex$ oracle: the challenger randomly selects a polynomial number of subsets of W and runs the $I_b \xleftarrow{R} BuildIndex(K_{pub}, KeywordRg(K_{pri}, W_b))$ for each subset W_b. Then, $\mathcal{B}$ sends the index set to the adversary. This algorithm is probabilistic. Even when the s_w is the same, it provides different (but valid) indexes.
- $Trapdoor$ oracle: the adversary chooses a finite number $q_{1,t}$ of expression tree $\{ET(W_{ET,1}, O_1), ..., ET(W_{ET,q_{1,t}}, O_{q_{1,t}})\}$. The challenger responds with the corresponding trapdoors $T_i \xleftarrow{R} Trapdoor(K_{pri}, ET(W_{ET,i} O_i))$ to the adversary, where $1 \leq i \leq q_{1,t}$ and $W_{ET,i} \subseteq W$.

At a certain point, the adversary sends the challenger two same-size keyword sets $\{W_0, W_1\}$ that are wished to be challenged. One restriction is that at most one of the following items could have occurred:

1) The $KeywordRg$ oracle has been queried with any keyword in W_0 or W_1.
2) The $Trapdoor$ oracle has returned $Trapdoor(K_{pri}, ET(W_{ET,i}, O_i))$, where $W_{ET,i}$ should not contain any keywords in $W_0 \bowtie W_1$, and $P_{ET_i(W_{ET,i},O_i)}(W_0) = P_{ET_i(W_{ET,i},O_i)}(W_1)$, for all $i = 1, 2, ..., q_{1,t}$.

Challenge phase: The challenger chooses a coin $\nu \in \{0, 1\}$ and gives $I_* \xleftarrow{R} BuildIndex(K_{pub}, W_*)$ to the adversary.

Query phase 2: The adversary continues to adaptively query the above types of oracles with the same restriction in **Query Phase 1**.

Guess: The adversary returns a guess $\nu' \in \{0, 1\}$ of ν and wins the game when $\nu' = \nu$.

We define the advantage of the adversary $\mathcal{A}$ in breaking the proposed scheme PEMKS as:

$$Adv_{\mathcal{A}} = \left| \Pr[\nu' = \nu] - \frac{1}{2} \right|.$$

Definition 4.2 PEMKS is secure against the adaptive cho-

sen keyword attack if for any polynomial-time adversary $\mathcal{A}$, the $Adv_{\mathcal{A}}$ is a negligible function.

The server has no duty to keep the index private, and the communication channel between servers and senders is public. According to the above game, the $BuildIndex$ oracle indicates that the adversary can obtain all the valid indexes. In the aforementioned security game, our scheme can also defend against the offline keyword guessing attack based on the following analysis:

1) When the adversary $\mathcal{A}$ is a sender, $\mathcal{A}$ could challenge the $KeywordRg$ and $BuildIndex$ oracles with restriction 1) in **Query Phase 1** and **Query Phase 2**. $\mathcal{A}$ cannot distinguish the challenged indexes by keeping the pre-tags and given indexes.
2) When the adversary $\mathcal{A}$ is a server, $\mathcal{A}$ could challenge two oracles $Trapdoor$ and $BuildIndex$ with restriction 2) in **Query Phase 1** and **Query Phase 2**. $\mathcal{A}$ cannot distinguish the challenged indexes by keeping the trapdoors and given indexes.
3) When the adversary $\mathcal{A}$ is simultaneously both a server and a sender, with the restrictions in **Query Phase 1** and **Query Phase 2**, $\mathcal{A}$ can obtain the pre-tags, trapdoors, and given indexes at the same time, except for the corresponding trapdoors and pre-tags of the challenged keyword sets. Therefore, $\mathcal{A}$ still cannot distinguish the challenged indexes.

A notable difference between traditional PEKS schemes is the $KeywordRg$ algorithm. We need to obtain the pre-tags before constructing an index. Therefore, it allows us to limit the adversary's ability to generate the challenged keywords' trapdoors and indexes at the same time, which is the key point for defending against the offline keyword guessing attack.

In fact, a pre-tag is only calculated once for each sender, and some pre-tags of special keywords can be published to reduce the communication overhead between a receiver and senders.

V. OUR CONSTRUCTION

A. CONSTRUCTION

Let G_1, G_2 be two bilinear groups of prime order p, and let $e : G_1 \times G_1 \to G_2$ be a bilinear map that satisfies the properties described in **Bilinear Maps**. The Lagrange coefficient is defined as $\Delta_{i,S}(x) = \prod_{j \in S, j \neq i} \frac{x-j}{i-j}$, where $i \in \mathbb{Z}_p$ and S is a set of elements in $\mathbb{Z}_p$. Each keyword is associated with a unique element in $\mathbb{Z}_p$, and $\mathbb{Z}_p = \{0, ..., p-1\}$ is a group modulo p.

- $Gen(s)$
  It takes a security parameter s as input and then randomly chooses a prime order p of groups G_1, G_2 and a hash function $H : \{0,1\}^* \to \mathbb{Z}_p$. Then, $\alpha, \beta, \chi, \delta$ are uniformly chosen at random from $\mathbb{Z}_p$. The public key is

$$K_{pub} = (g, s_* = g^{\beta}, g^{\delta}, Y = e(g,g)^{\alpha}, H)$$

  . The private key is $K_{pri} = (\alpha, \beta, \chi, \delta)$.

- $KeywordRg(K_{pri}, w)$
  It takes the private key K_{pri} and a keyword w as the input, and it calculates t_w as follows:

$$t_w = H(w||\chi).$$

  t_w is kept as a secret of the keyword. Then, it returns the pre-tag to the data sender:

$$s_w = g^{t_w}.$$

- $BuildIndex(K_{pub}, S_{W_i})$
  Let $W_i = \{w_1, ..., w_h\}$ be the set of keywords in a document D_i, and the set of pre-tags is $S_{W_i} = \{s_{w_1}, ...s_{w_h}\}$. Two random numbers $s, r \in \mathbb{Z}_p$ are chosen, and the index is computed as follows:

$$I = (E = Y^s, \{E^{(1)}_{w_j} = {s_{w_j}}^s\}_{w_j \in W_i}, E^{(2)} = {s_*}^s,$$
$$E^{(3)} = g^r, \{E^{(4)}_{w_j} = e(g^{\delta}, s_{w_j})^r\}_{w_j \in W_i}).$$

- $Trapdoor(K_{pri}, ET(W_{ET}, O))$
  For each node (including the leaves) in the expression tree, it first chooses a polynomial q_x following a top-down manner as described below. Let x denote the node in the expression tree, and let the highest degree d_x of the polynomial q_x be one less than k_x of the node, denoted as $d_x = k_x - 1$. Then, it sets $q_r(0) = \alpha$ for the root node r and randomly chooses other d_r points of the polynomial q_r. For any other node x, it sets $q_x(0) = q_{parent(x)}(index(x))$ and randomly chooses other d_x points. Each leaf node x is associated with a keyword, which could be a positive keyword w_i or a negative keyword w'_i. If the leaf node x is associated with a positive keyword w_i, we generate the trapdoor T_x in the following way:

$$T_x = (T^{(1)}_x = g^{\frac{q_x(0)}{t_{w_i}}}, T^{(2)}_x = {s_{w_i}}^{\delta}).$$

  If the leaf node x is associated with a negative keyword w'_i, T_x is calculated in the following way:

$$T_x = (T^{(3)}_x = g^{\frac{q_x(0)}{\beta}}, T^{(2)}_x = {s_{w_i}}^{\delta}).$$

  In the end, the trapdoor is

$$T = \{T_1, ..., T_j, ET\},$$

  where j is the number of leaf nodes, ET is the structure of the expression tree, and each node contains its identifier and the positive or negative attribute of the keyword.
- $Search(I, T)$:
  We first define a recursive algorithm $DecrytNode(T, I, x)$, which returns the F_x as the output, where I is an index, T is the trapdoor, and x is the node in the expression tree. The algorithm $DecrytNode(T, I, x)$ proceeds as follows:

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2019.2898751, IEEE Access

If the keyword is positive and $E_{w_i}^{(4)} = e(E^{(3)}, T_x^{(2)})$, we can calculate the F_x as follows:

$$\begin{aligned} F_x &= e(E_{w_i}^{(1)}, T_x^{(1)}) \\ &= e({s_{w_i}}^s, g^{\frac{q_x(0)}{t_x}}) \\ &= e(g,g)^{q_x(0)s}. \end{aligned}$$

If the keyword is negative and $E_{w_i}^{(4)} \neq e(E^{(3)}, T_x^{(2)})$, we can calculate the F_x as follows:

$$\begin{aligned} F_x &= e(E^{(2)}, T_x^{(3)}) \\ &= e({s_*}^s, g^{\frac{q_x(0)}{\beta}}) \\ &= e(g,g)^{q_x(0)s}. \end{aligned}$$

If there is no leaf node satisfying the above description, $search()$ returns false. Otherwise, $search()$ continues to compute the F_N for the internal node N.
For the node N with all the child nodes x, F_N is computed in the following way:

$$\begin{aligned} F_N &= \prod_{x \in S_N} F_x^{\Delta_{i,S'_N}(0)} \; where \; \begin{matrix} i = index(x) \\ S'_N = \{index(x) : x \in S_N\} \end{matrix} \\ &= \prod_{x \in S_N} (e(g,g)^{sq_x(0)})^{\Delta_{i,S'_N}(0)} \\ &= \prod_{x \in S_N} (e(g,g)^{sq_{parent(x)}(index(x))})^{\Delta_{i,S'_N}(0)} \\ &= e(g,g)^{sq_N(0)}. \end{aligned}$$

Finally, F_r can be calculated, where r is the tree's root.

- If $F_r = Y^s$, $Search()$ will return true.
- If $F_r \neq Y^s$, $Search()$ will return false.

We now present the proof of its correctness.

Correctness Let I be an index and $ET(W_{ET}, O)$ be the expression tree. Let $P \rightarrow (0,1)$ be the function that can check whether I satisfies $ET(W_{ET}, O)$:

Let

$$(K_{pub}, K_{pri}) \xleftarrow{R} Gen(s), s_w \leftarrow KeywordRg(K_{pri}, w),$$
$$I \xleftarrow{R} BuildIndex(K_{pub}, S_W),$$
$$T \leftarrow Trapdoor(K_{pri}, ET(W_{ET}, O)).$$

- If $P_{ET(W_{ET},O)}(I) = 1$, it simply calculates the Y^s as the above description.
- If $P_{ET(W_{ET},O)}(I) = 0$, the following lemma shows that if H is collision-resistant, $\Pr[Search(I,T) \neq false]$ is negligible.

Lemma 5.1 Whenever $P_{ET(W_{ET},O)}(I) = 0$, if H is collision-resistant, $\Pr[Search(I,T) \neq false]$ is negligible.

Proof: Let w be the keyword in the index, and let w_x be the keyword that is associated with the leaf node x in the expression tree. The $search()$ algorithm contains two steps: 1) If the node x is a leaf node and only w is matched with w_x, we could calculate the valid F_x for the node x. When w_x is a positive keyword, the word "match" means that $w = w_x$; otherwise, it means that $w \neq w_x$. 2) If the node x is an internal node and only x's children satisfy the operation, we could calculate the valid F_x and then obtain the correct F_r for the root.

According to the $search()$ algorithm, the proof is divided into two parts:

- If node x is a leaf node and w is not matched with w_x, then the probability of calculating a valid F_x is negligible. When w_x is a negative keyword, the probability is zero as $E_{w_i}^{(4)} = e(E^{(3)}, T_x^{(2)})$. When w_x is a positive keyword and H is collision-resistant, in the following equation:

$$\begin{aligned} & E_{w_i}^{(4)} = e(E^{(3)}, T_x^{(2)}) \\ \Rightarrow & e(g^\delta, s_{w_i})^r = e(g^r, {s_x}^\delta) \\ \Rightarrow & e(g, s_{w_i})^{r\delta} = e(g, s_x)^{r\delta}, \end{aligned}$$

  the probability of $s_x = s_{w_i}$ is negligible when $w_i \neq x$. Therefore, the probability of calculating a valid F_x is still negligible.

- If node x is an internal node and its children do not satisfy the operation, the probability of calculating a valid F_x is negligible. By using the interpolation theorem, it is very similar to the proof of privacy in the secret sharing scheme in [39]. It will not be proved again.

Therefore, whenever $P_{ET(W_{ET},O)}(I) = 0$, if H is collision-resistant, $\Pr[Search(I,T) \neq false]$ is negligible.

B. SECURITY ANALYSIS

Under the security model mentioned above, the security proof of the proposed scheme is given.

Theorem 5.1 Under the DBDH and MDBDH assumptions, the proposed PEMKS is secure against the adaptive chosen keyword attack.

Proof: Three types of adversaries are introduced in **Section IV**. The third adversary, who is the server and the sender at the same time, actually has the abilities of the first two adversaries. Therefore, the security proof is presented when the adversary is the third one.

In terms of the proof, the convenient way is to construct a sequence of games, where game G_0 is the same as G_1, G_1 is the same as G_2,..., and the last game is the same as the former one except that the index might be generated in a different way. Suppose that there is a adversary $\mathcal{A}$ can distinguish the sequence games, then our scheme can build a simulator $\mathcal{B}$ that solves the DBDH and MDBDH assumptions.

First, set groups G_1 and G_2 with a bilinear map e. Then, let $[E, \{E_{w_i}^{(1)}, E_{w_i}^{(4)}\}_{i\in[1,m]}, E^{(2)}, E^{(3)}]$ denote the challenge index that will be given to the adversary in the real attack. Let $R, \{R_i^{(4)}\}_{i\in[1,m]}$ be the random elements of G_2. We define the hybrid games, which differ in terms of what challenge index is given by the simulator $\mathcal{B}$ to the adversary $\mathcal{A}$:

$Game_0$: $I_0 = [E, \{E_{w_i}^{(1)}, E_{w_i}^{(4)}\}_{i\in[1,m]}, E^{(2)}, E^{(3)}]$

$Game_1$: $I_1 = [R, \{E_{w_i}^{(1)}, E_{w_i}^{(4)}\}_{i\in[1,m]}, E^{(2)}, E^{(3)}]$

$Game_2$: $I_2 = [R, \{E_{w_1}^{(1)}, R_1^{(4)}\}, \{E_{w_i}^{(1)}, E_{w_i}^{(4)}\}_{i\in[2,m]}, E^{(2)}, E^{(3)}]$

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2019.2898751, IEEE Access

.....

$Game_{m+1}$: $I_{m+1} = [R, \{E_i^{(1)}, R_i^{(4)}\}_{i\in[1,m]}, E^{(2)}, E^{(3)}]$

The proposed scheme shows that the challenge index in $Game_{m+1}$ leaks no information about the keywords since it is composed of five random group elements and illustrates that the transitions from $Game_0$ to $Game_1$ to $Game_2$ to .....to $Game_{m+1}$ are all computationally indistinguishable.

Lemma 5.2 Under the (t, ε)-DBDH assumption, in time t, there is no running adversary that can distinguish between $Game_0$ and $Game_1$ with an advantage greater than ε.

Proof: Suppose that there is an adversary $\mathcal{A}$ that can distinguish between $Game_0$ and $Game_1$ with advantage ε. We build a simulator $\mathcal{B}$ that plays the DBDH game with advantage ε.

$\mathcal{B}$ receives a DBDH challenge $[g, A = g^a, B = g^b, C = g^c, Z]$, where Z is either $e(g,g)^{abc}$ or a random element of G_2 with equal probability. The game is as follows:

Setup: The simulator $\mathcal{B}$ randomly chooses β, δ from $\mathbb{Z}_p$, retains the generator g, and then outputs $g_1 = g^\delta, g_2 = g^\beta$. Then, $\mathcal{B}$ selects a random oracle H and assigns $Y = e(A,B) = e(g,g)^{ab}$. The public key is as follows:

$$g, g_1 = g^\delta, g_2 = g^\beta, Y = e(g,g)^{ab}, H.$$

Finally, $\mathcal{B}$ selects a set W of strings from $\{0,1\}^*$ uniformly at random and sends W to $\mathcal{A}$.

Query Phase 1:

- $KeywordRg$ oracle: $\mathcal{A}$ can query H for the pre-tags, and $\mathcal{B}$ will maintain an $H-list$ to respond to $\mathcal{A}$. The $H-list$ consists of a list of the tuples $< w_i, c_i >$, which is initially empty. When $\mathcal{A}$ queries $w_i \in W$ to the $KeywordRg$ oracle, $\mathcal{B}$ will return the pre-tag to the adversary as follows:
  1) If the query of w_i is already in the $H-list$, $\mathcal{B}$ returns c_i to $\mathcal{A}$.
  2) Otherwise, $\mathcal{B}$ selects a random $a_i \in \mathbb{Z}_p$ and responds with $c_i = g^{a_i}$. Then, $\mathcal{B}$ adds the tuple $< w_i, c_i >$ to $H-list$.
- $BuildIndex$ oracle: $\mathcal{B}$ randomly selects a polynomial number of subsets of W. This collection of subsets is denoted as W^*. For each subset $W_i \subseteq W^*$, $\mathcal{B}$ runs the $BuildIndex$ and sends the set of indexes I^* to the $\mathcal{A}$.
- $Trapdoor$ oracle: $\mathcal{A}$ adaptively chooses a finite number of queries $q_{1,t}$ of expression tree $ET(W_{ET}, O)$ to challenge the $Trapdoor$ oracle. $\mathcal{B}$ sets up the polynomial for the nodes of the expression tree. First, $\mathcal{B}$ defines an algorithm as:

  $$Poly(ET_x(W_{ET}, O), \lambda_x).$$

  This is a recursion algorithm, which takes an expression tree $ET_x(W_{ET}, O)$ with a root x and an integer $\lambda_x \in \mathbb{Z}_p$ as input and outputs a polynomial. For simplicity, ET_x is short for $ET_x(W_{ET}, O)$. This algorithm starts by setting up a polynomial q_x with the highest degree d_x for the node x and $q_x(0)$=λ_x. The remaining points are chosen randomly from $\mathbb{Z}_p$. Then, the algorithm constructs a polynomial for each child node x by calling the algorithm $Poly(ET_x, q_{parent(x)}(index(x)))$. Note that $q_x(0) = q_{parent(x)}(index(x))$ for each child node x. Now, $\mathcal{B}$ runs $Poly(ET_x, a)$ to define polynomials for the nodes in the expression tree. Finally, $\mathcal{B}$ defines the final polynomial $Q_x(*) = bq_x(*)$ for each node x of the expression tree, and $Q_r(0) = ab$, where r is the root of the tree.

  $\mathcal{B}$ calculates T_x with the polynomial of the leaf node x as follows:

  1) If the leaf node is associated with a negative keyword w_i', $\mathcal{B}$ chooses the c_i from the $H-list$ where the keyword is w_i and returns T_x as follows:

  $$T_x = (T_x^{(3)} = g^{\frac{bq_x(0)}{\beta}} = B^{\frac{q_x(0)}{\beta}}, T_x^{(2)} = {c_i}^\delta)$$

  2) If the leaf node is associated with a positive keyword w_i, $\mathcal{B}$ chooses the c_i from the $H-list$ where the keyword is w_i and returns T_x as follows:

  $$T_x = (T_x^{(1)} = g^{\frac{bq_x(0)}{c_i}} = B^{\frac{q_x(0)}{c_i}}, T_x^{(2)} = {c_i}^\delta)$$

  Finally, $\mathcal{B}$ sends $T = \{T_1, ..., T_j\}$ to $\mathcal{A}$, where j is the number of leaf nodes.

Challenge: $\mathcal{A}$ chooses two keyword sets $W_0 = \{w_{0,1}, ..., w_{0,m}\}$, $W_1 = \{w_{1,1}, ...w_{1,m}\}$ with the same number of keywords under the constraint that at most one of the following items could have occurred.

1) The $KeywordRg$ oracle has been queried with any keyword in W_0 or W_1.
2) The $Trapdoor$ oracle has returned $Trapdoor(K_{pri}, ET(W_{ET,i}, O_i))$, where $W_{ET,i}$ should not contain any keyword in $W_0 \bowtie W_1$, $P_{ET(W_{ET,i},O_i)}(W_0) = P_{ET(W_{ET,i},O_i)}(W_1)$ for all $i = 1, 2, ..., q_{1,t}$, and the keywords cannot all be negative keywords.

Then, $\mathcal{B}$ flips a fair binary coin ν, chooses two random elements $c, r \in \mathbb{Z}_p$, and returns an index I as follows:

$$I = (E = Z, \{E_{w_{\nu,i}}^{(1)} = C^{a_i}\}_{w_{\nu,i}\in W_\nu},$$
$$E^{(2)} = C^\beta, E^{(3)} = g^r, \{E_{w_{\nu,i}}^{(4)} = e(g_1, c_i)^r\}_{w_{\nu,i}\in W_\nu})$$

Phase 2: $\mathcal{A}$ continues querying as in **Phase 1** under the constraint described above.

Guess: $\mathcal{A}$ eventually outputs a bit ν', representing its guess for ν.

If $Z = e(g,g)^{abc}$, then in $\mathcal{A}$'s view, this game is identical to the original game G_0. If Z is a random element in G_2, the advantage of $\mathcal{A}$ is negligible. Therefore, if $\mathcal{A}$ can distinguish game $Game_0$ from game $Game_1$ with a nonnegligible probability, $\mathcal{B}$ has a nonnegligible probability in breaking the DBDH assumption.

Lemma 5.3 Under the (t, ε)-MDBDH assumption, in time t, there is no running adversary that can distinguish between $Game_m$ and $Game_{m+1}$ with an advantage greater than ε for $m \in [1, m]$.

Proof: Suppose that there is an adversary $\mathcal{A}$ that can distinguish between $Game_m$ and $Game_{m+1}$ with advantage

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2019.2898751, IEEE Access

ε. We build a simulator $\mathcal{B}$ that plays the MDBDH game with advantage ε.

$\mathcal{B}$ receives an MDBDH challenge $[g, Z_{1,1} = g^{z_{1,1}}, ..., Z_{1,m} = g^{z_{1,m}}, Z_2 = g^{z_2}, Z_3 = g^{z_3}, R_1, ..., R_m]$, where R_i is either $e(g,g)^{z_{1,i}z_2z_3}$ or a random element of G_2 with equal probability. The game proceeds as follows:

Setup: The simulator $\mathcal{B}$ randomly chooses z_*, z_2 from $\mathbb{Z}_p$, retains the generator g, and then outputs $g_1 = g^{z_2}, g_2 = g^{z_*}$. Then, $\mathcal{B}$ selects a random oracle H and assigns $Y = e(A,B) = e(g,g)^{ab}$. The public key is as follows:

$$g, g_1 = g^{z_2}, g_2 = g^{z_*}, Y = e(g,g)^{ab}, H$$

Finally, $\mathcal{B}$ selects a set W of strings from $\{0,1\}^*$ uniformly at random and sends W to $\mathcal{A}$.

Query Phase 1:

- $KeywordRg$ oracle: $\mathcal{A}$ can query H for the pre-tags, and $\mathcal{B}$ will maintain an $H-list$ to respond to $\mathcal{A}$. The $H-list$ consists of a list of the tuples $< w_i, t_i >$. The list is initially empty. When $\mathcal{A}$ queries $w_i \in W$ to the $KeywordRg$ oracle, $\mathcal{B}$ returns the pre-tag to the adversary as follows:
  1) If the query of w_i is already in the $H-list$, $\mathcal{B}$ returns t_i to $\mathcal{A}$.
  2) Otherwise, $\mathcal{B}$ selects a random $z_{1,i} \in \mathbb{Z}_p$ and responds with $t_i = g^{z_{1,i}} = Z_{1,i}$. Then, $\mathcal{B}$ adds the tuple $< w_i, t_i >$ to $H-list$.
- The $BuildIndex$ oracle is the same as the $BuildIndex$ oracle in **Lemma 5.2**.
- $Trapdoor$ queries: $\mathcal{A}$ adaptively chooses a finite number of queries $q_{1,t}$ of expression tree $ET(W_{ET}, O)$ to challenge the $Trapdoor$ oracle. $\mathcal{B}$ generates the $T_x^{(1)}$ and $T_x^{(3)}$, which is the same as **Lemma 5.2**. The $T_x^{(2)}$ is as follows:

$$T_x^{(2)} = g^{z_2 z_i} = Z_2^{z_i}.$$

Challenge: $\mathcal{A}$ chooses two keyword sets $W_0 = \{w_{0,1}, ..., w_{0,m}\}$, $W_1 = \{w_{1,1}, ...w_{1,m}\}$ with the same constraint as in **Lemma 5.2**. Then, $\mathcal{B}$ flips a fair binary coin ν, chooses $c, z_3 \in \mathbb{Z}_p$, and returns an index I as follows:

For $i = m$, $\mathcal{B}$ outputs:

$$I_m = (E = (e(g,g)^{ab})^c, E_m^{(1)} = (g^{z_{1,m}})^c, E^{(2)} = (g^\beta)^c, E^{(3)} = g^{z_3} = Z_3, E_m^{(4)} = e(Z_{1,m}, Z_2)^{z_3} = R_m).$$

For $i \in [1, m-1]$, $\mathcal{B}$ outputs:

$$E_i^{(1)} = (g^{z_{1,i}})^c, E_i^{(4)} = e(Z_{1,i}, Z_2)^{z_3} = R_i.$$

Phase 2: $\mathcal{A}$ continues querying as in **Phase 1** under the constraint described above.

Guess: $\mathcal{A}$ eventually outputs a bit ν', representing its guess for ν.

If $R_i = e(g,g)^{z_{1,i}z_2z_3}$, then $\mathcal{A}$'s view of this game is identical to the original game. If R_i is a random element in G_2, then the advantage of $\mathcal{A}$ is negligible. Therefore, if $\mathcal{A}$ can distinguish game $Game_m$ from game $Game_{m+1}$ with a nonnegligible probability, then $\mathcal{B}$ has a nonnegligible probability in breaking the MDBDH assumption.

Under the above games, the proposed scheme also defend against the offline keyword guessing attack which is mentioned in section III.

The following is the discussion of the other two types of adversaries that are not considered above: a sender colluding with other senders and senders colluding with a server.

Sender Colluding with Other Senders: If a sender colludes with other senders, this means they can obtain the pre-tags they should not know, while they still cannot obtain the corresponding trapdoor. In fact, our scheme can defend against this type of adversary. The $BuildIndex$ algorithm is probabilistic. Different senders cannot generate the same index with the same keywords. Therefore, a sender colluding with other senders is equivalent to the adversary being a sender, as we have proven above.

Senders Colluding with A Server: If the senders collude with a server, this means that this type of adversary can obtain all the pre-tags, trapdoors and indexes without limitations. Suppose that Bob is a sender and Alice is a sender who also has root privileges of the server; there are two different situations:

- If Alice steals the pre-tags from Bob, this means Alice only acquires the pre-tags without the corresponding keywords. We can encrypt the keyword first and then generate the pre-tag (this method is proposed in [16]). Finally, Alice still cannot distinguish the indexes with the pre-tags.
- If Bob gives the pre-tags and corresponding keywords together to Alice, we can take some measures in practice to keep the proposed scheme secure. The data receiver keeps some special sets of keywords secure by only giving these keywords to some trusted senders.

Negative Keywords In the proposed scheme, we use a wild keyword to represent all the negative keywords that are not in the index. However, if the adversary queries the trapdoor with an expression tree that only contains negative keywords, we obtain the following result:

$$\begin{aligned}
&\prod_{x \in S_N} T_x^{(3)^{\Delta_{i,S'_N}(0)}} \text{where } \begin{matrix} i = index(x) \\ S'_N = \{index(x) : x \in S_N\} \end{matrix} \\
&= \prod_{x \in S_N} (g^{\frac{q_x(0)}{\beta}})^{\Delta_{i,S'_N}(0)} \\
&= \prod_{x \in S_N} (g^{\frac{q_{parent(x)}(index(x))}{\beta}})^{\Delta_{i,S'_N}(0)} \\
&= g^{\frac{q_N(0)}{\beta}}.
\end{aligned}$$

Finally, we can obtain $g^{\frac{\alpha}{\beta}}$ and then compute:

$$\begin{aligned}
&e(g^{\frac{\alpha}{\beta}}, s_*^s) \\
=&e(g^{\frac{\alpha}{\beta}}, g^{s\beta}) \\
=&e(g,g)^{s\alpha}.
\end{aligned}$$

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2019.2898751, IEEE Access

If all the keywords in the trapdoor are negative keywords, the trapdoor will match with all indexes, and the adversary could obtain all the encrypted data. Actually, the adversary cannot decrypt the encrypted data and learn any information about the keyword because s_* is just a wild keyword without a real meaning. In the real world, a person typically will not search with a set of negative keywords.

VI. PERFORMANCE ANALYSIS

A. EFFICIENCY

The scheme's efficiency includes the storage overhead of the public key, private key, index and trapdoor, as well as the computational overhead of algorithms $Gen()$, $KeywordRg()$, $BuildIndex()$, $Trapdoor()$ and $Search()$.

Let n, l, o, and m be the sizes of the distinct keyword set in encrypted data, keyword set in trapdoor, operation set and keyword set in an index, respectively. Let l_p, l_n be the numbers of positive keywords and negative keywords in the trapdoor, where $l = l_p + l_n$. The time consumed on the group is the main part of the overall computational overhead. Therefore, using the operations on the group to analyze the computational overhead in theory. Let E denote an exponentiation operation, and let P denote a pairing operation.

In $Gen()$, the size of the public key is a constant 4, and a part of the size of the private key is 4. The computational overhead is $3 * E$.

In $KeywordRg()$, the proposed scheme generates a pair of secret and pre-tag for each keyword, gives the pre-tag to the sender, and keeps the secret to ourselves. Therefore, the size of the private key depends on the number of secrets of pre-tags, which is $4 + n$. The computational overhead of generating all the pre-tags is $n * E$.

In $BuildIndex()$, the storage overhead of an index, represented as $|index|$, is determined by the number m of keywords in the index; thus, $|index| = 2m + 3$. The computational overhead of each index is $(m+3)*E+m*P$.

In $Trapdoor()$, each keyword in the trapdoor contains two group elements. Therefore, the storage overhead of the trapdoor $|Trapdoor|$ is $2 * l$. The computational overhead is $2l * E$.

Let $min()$ be a function to obtain the minimum number. In $Search()$, the valid F_x for the leaf node x should be computed firstly, which is $m * logl$ at most, and then computes F_x for internal nodes x in the expression tree. Therefore, the computational overhead of $search()$ is less than $(m * logl + min(m, l)) * P + (o + l) * E$.

Comparison We compare the performance of the proposed scheme with the other four schemes [7] [8] [11] [34]. The reason of choosing these four schemes is that they are all representative schemes published in the past 5 years. They support conjunction and disjunction at least, and are based on the basic idea of ABE. So these schemes are similar to our scheme. The proposed scheme aims to extend the search criteria rather than improving the efficiency. Therefore, our scheme is feasible when the storage overhead and computational overhead are similar to the other four schemes.

The storage overheads of these schemes are shown in TABLE 1. We publish the public key to everyone and keep the private key to ourselves. The server stores the indexes, and the storage overhead of the trapdoor is the communication overhead. There is little difference in terms of storage overhead between the proposed scheme and the other four schemes. The costs of the public key and trapdoor in our scheme are the least. The costs of the private key and index in our scheme are between the highest and lowest costs.

The computational overheads of these schemes are shown in TABLE 2. The $BuildIndex()$ algorithm is run by the data senders, the data receiver runs the $Trapdoor()$ algorithm, and the server runs the $Search()$ algorithm.Since the four schemes [7] [8] [11] [34] are based on the linear secret sharing scheme (LSSS) [40], v_1 is defined as the number of elements in the set of minimum subsets that satisfy the access structure and v_2 is defined as the number of elements in all the subsets mentioned in [11]. Since these two parameters are not used in our scheme, we present a simple example to explain v_1 and v_2. Given a search expression $(Z_1 = A\ AND\ (Z_2 = B\ OR\ Z_3 = C))$, the minimum subsets that satisfy the search expression are $\{Z_1, Z_2\}$ and $\{Z_1, Z_3\}$, and $v_1 = 2, v_2 = 4$.

As shown in TABLE 2, the computational overhead of $Search()$ in our scheme is not much less than that of the others. Actually, there is no need to compute for all nodes. The number of nodes that need to be computed depends on the operation R_k. If k is 1, then it means that we have constructed a constant polynomial, and only need to compute the overhead of one child node. In the experiments, once F_r of the root is generated, the algorithm can be terminated. In the real world, people generally search on short criteria. Therefore, the practical computational overhead is similar to the other schemes.

B. EXPERIMENT

We implement our scheme in the PBC library, which is a free lightweight C library providing the mathematical details of a bilinear pairing on a cyclic group. There are three groups $G1$, $G2$, and GT of prime order r and a bilinear map e to generate an element of GT that takes an element from $G1$ and an element from $G2$ as inputs. Our experiments run on an all-in-one desktop computer with an Intel Core i7-6650U CPU (4 core 2.20GHz) and 8 GB of RAM running 64-bit Windows 10 Pro.

At the beginning of this section, we analyzed the performance theoretically and mentioned two operations: exponentiation operation and pairing operation. Some simple experimental results about the computational overhead for each operation are shown in FIGURE 3, where the x-coordinate represents the different operations: $e(G1, G1)$, $e(G2, G2)$, $e(G1, G2)$, $G1^a$, $G2^a$ and GT^a. FIGURE 3 shows that the computational overheads of the pairing operations on different types of group elements are the same. $G1^a$ and $G2^a$ have the same computational overheads because they have the same data structures in the PBC library. GT^a has the minimum computational overhead.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2019.2898751, IEEE Access

TABLE 1: Storage overhead comparison with other schemes

Scheme	Public Key	Private Key	Trapdoor	Index
[7]	9	5	$6l$	$5m$
[8]	$n+5$	$n+4$	$2l$	$m+2$
[11]	$n+4$	$m+2$	$3l$	$m+2$
[34]	10	8	$(4n+2)l$	7
Our scheme	4	$n+4$	$2l$	$2m+3$

TABLE 2: Computational overhead comparison with other schemes

Scheme	$BuildIndex()$	$Trapdoor()$	$Search()$
[7]	$(7m+2)*E$	$(16l+1)*E$	$\leq (1+v_2)*E+(6v_2+1)*P$
[8]	$2(m+1)*E$	$4l*E$	$\leq v_2E+2v_2*P$
[11]	$(m+2)*E+P$	$4l*E$	$\leq v_2E+2(l_p+v_2)*P$
[34]	$(n+6)*E+P$	$(15l+1)*E$	$\leq 7P+(v_2+1)*E$
Our scheme	$(m+3)*E+m*P$	$2l*E$	$\leq (m*logl+min(m,l))*P+(o+l)*E$

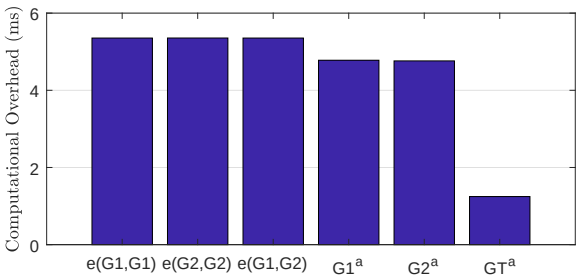

FIGURE 3: The computational overhead of operations

In FIGURE 4, we present the storage overhead of the proposed scheme, where Z_n means a prime order representing a secret key, and the others representing the standard group elements, public parameters, master key, pre-tag, index that contains two keywords, and trapdoor that contains three keywords, respectively.

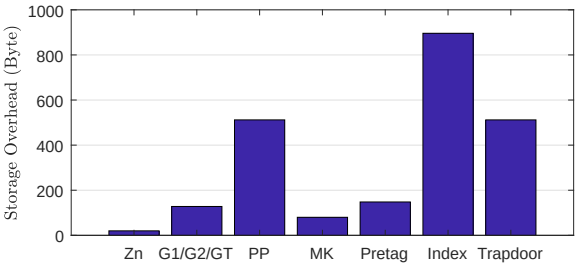

FIGURE 4: The storage overhead of PEMKS

In FIGURE 5, we present the computational overheads of the algorithms $KeywordRg()$, $BuildIndex()$, $Trapdoor()$, and $Search()$.

The first one shows the time for generating pre-tags containing 100 to 1000 keywords. The computational time is almost linear with the number of keywords.

The second one presents the computational overhead of generating indexes containing 10 to 50 documents. In the proposed scheme, the indexes are generated by multiple senders. Actually, each sender would not generate a large number of indexes. The computational time is linear with the number of documents. This figure also shows four other different cases, where each document contains 8 keywords, 10 keywords and 12 keywords. We observe that an increase in the number of keywords in each index implies an increase in the computational time for each index.

The third one shows the computational overhead for generating a trapdoor that contains 4 to 14 keywords. The time is also linear with the number of keywords in the trapdoor.

The final one is the computational overhead for searching. There are many factors that influence the search time, such as the numbers of keywords in the trapdoor and index, the number of matched indexes, the search criteria and so on. To obtain relatively realistic results, we generate indexes containing 10 to 18 keywords, and each keyword is chosen from 1000 keywords randomly. Different lines mean different numbers of keywords in the trapdoor. The computational time of the same number of keywords in the trapdoor is linear with the number of keywords per index, and the search time increases with the increase in the number of keywords in the trapdoor.

VII. CONCLUSION

In this paper, we propose a novel multi-keyword search scheme, called PEMKS, which solves the problem of how to customize the keywords' relevancy. The proposed scheme allows the data receiver to search encrypted data that contain any n keywords of a keyword set. At the same time, it can also support negation. We present the details of the scheme, formally define its security, and prove that the proposed scheme is secure against the adaptive chosen keyword attack and the offline keyword guessing attack. We present the performance analysis compared with other schemes in theory, implement the proposed scheme and conduct some experiments to evaluate the proposed scheme's performance.

REFERENCES

[1] 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.

[2] P. Xu, H. Jin, Q. Wu, and W. Wang, "Public-key encryption with fuzzy keyword search: A provably secure scheme under keyword guessing

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2019.2898751, IEEE Access

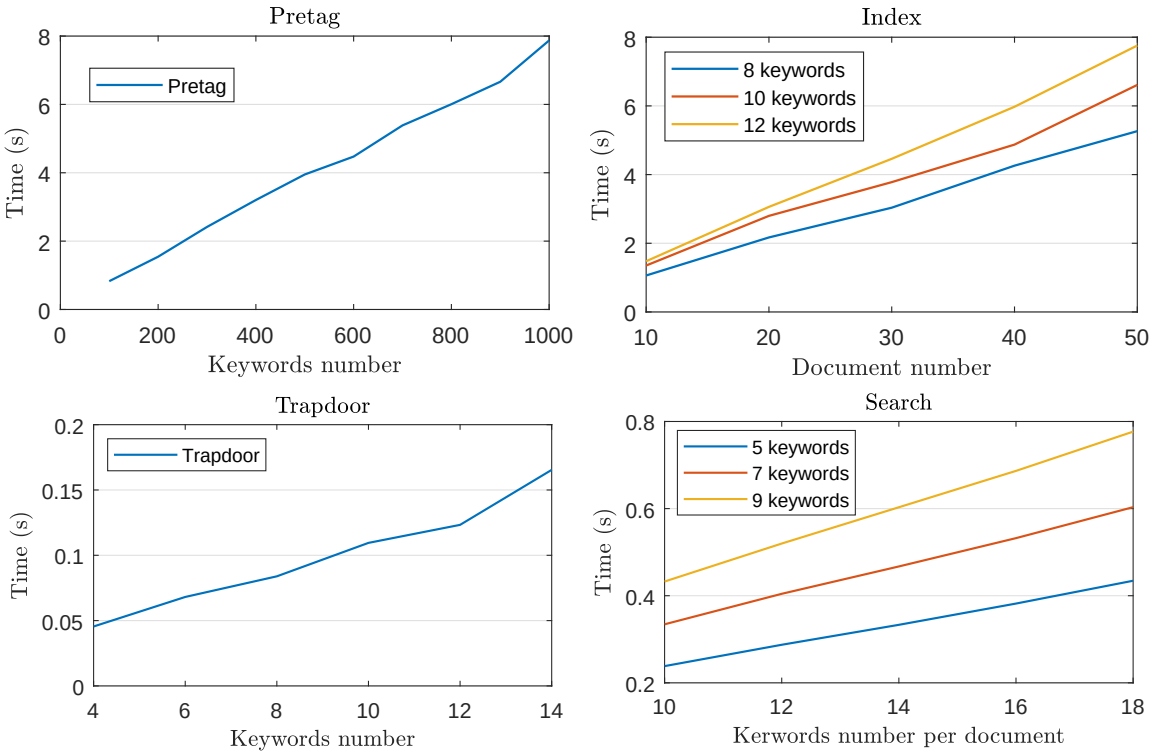

FIGURE 5: The computational overhead of PEMKS

attack," IEEE Transactions on computers, vol. 62, no. 11, pp. 2266–2277, 2013.

[3] J. Baek, R. Safavi-Naini, and W. Susilo, "Public key encryption with keyword search revisited," in International conference on Computational Science and Its Applications. Springer, 2008, pp. 1249–1259.

[4] Y. H. Hwang and P. J. Lee, "Public key encryption with conjunctive keyword search and its extension to a multi-user system," in International Conference on Pairing-Based Cryptography. Springer, 2007, pp. 2–22.

[5] D. J. Park, K. Kim, and P. J. Lee, "Public key encryption with conjunctive field keyword search," in International Workshop on Information Security Applications. Springer, 2004, pp. 73–86.

[6] S. Jiang, X. Zhu, L. Guo et al., "Publicly verifiable boolean query over outsourced encrypted data," IEEE Transactions on Cloud Computing, 2017.

[7] H. Cui, Z. Wan, R. Deng, G. Wang, and Y. Li, "Efficient and expressive keyword search over encrypted data in the cloud," IEEE Transactions on Dependable and Secure Computing, 2016.

[8] J. Lai, X. Zhou, R. H. Deng, Y. Li, and K. Chen, "Expressive search on encrypted data," in Proceedings of the 8th ACM SIGSAC symposium on Information, computer and communications security. ACM, 2013, pp. 243–252.

[9] Y. Zhang and S. Lu, "Poster: Efficient method for disjunctive and conjunctive keyword search over encrypted data," in Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security. ACM, 2014, pp. 1535–1537.

[10] J. Han, Y. Yang, J. K. Liu, J. Li, K. Liang, and J. Shen, "Expressive attribute-based keyword search with constant-size ciphertext," Soft Computing, no. 4, pp. 1–15, 2017.

[11] Z. Lv, C. Hong, M. Zhang, and D. Feng, "Expressive and secure searchable encryption in the public key setting," in International Conference on Information Security. Springer, 2014, pp. 364–376.

[12] Y. Miao, J. Ma, X. Liu, X. Li, Z. Liu, and H. Li, "Practical attribute-based multi-keyword search scheme in mobile crowdsourcing," IEEE Internet of Things Journal, vol. PP, no. 99, pp. 1–1, 2017.

[13] D. Boneh and B. Waters, "Conjunctive, subset, and range queries on encrypted data," in Theory of Cryptography Conference. Springer, 2007, pp. 535–554.

[14] S. Faber, S. Jarecki, H. Krawczyk, Q. Nguyen, M. Rosu, and M. Steiner, "Rich queries on encrypted data: Beyond exact matches," in European Symposium on Research in Computer Security. Springer, 2015, pp. 123–145.

[15] V. Goyal, O. Pandey, A. Sahai, and B. Waters, "Attribute-based encryption for fine-grained access control of encrypted data," in Proceedings of the 13th ACM conference on Computer and communications security. Acm, 2006, pp. 89–98.

[16] Q. Tang and L. Chen, "Public-key encryption with registered keyword search," in European Public Key Infrastructure Workshop. Springer, 2009, pp. 163–178.

[17] 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.

[18] E.-J. Goh et al., "Secure indexes." IACR Cryptology ePrint Archive, vol. 2003, p. 216, 2003.

[19] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky, "Searchable symmetric encryption: improved definitions and efficient constructions," Journal of Computer Security, vol. 19, no. 5, pp. 895–934, 2011.

[20] 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.

[21] Z. Fu, X. Wu, C. Guan, X. Sun, and K. Ren, "Toward efficient multi-keyword fuzzy search over encrypted outsourced data with accuracy improvement," IEEE Transactions on Information Forensics and Security, vol. 11, no. 12, pp. 2706–2716, 2017.

[22] J. Yao, Y. Zheng, C. Wang, and X. Gui, "Enabling search over encrypted cloud data with concealed search pattern," IEEE Access, 2018.

[23] S. Kamara and T. Moataz, "Boolean searchable symmetric encryption with worst-case sub-linear complexity," in International Conference on the Theory and Applications of Cryptographic Techniques, 2017, pp. 94–124.

[24] X. Yuan, X. Yuan, B. Li, and C. Wang, "Secure multi-client data access with boolean queries in distributed key-value stores," in Communications and Network Security, 2017, pp. 1–9.

[25] B. Wang, Y. Hou, M. Li, H. Wang, H. Li, and F. Li, Tree-Based Multi-dimensional Range Search on Encrypted Data with Enhanced Privacy. Springer International Publishing, 2014.

[26] N. S. Jho, K. Y. Chang, D. Hong, and C. Seo, "Symmetric searchable

encryption with efficient range query using multi-layered linked chains," Journal of Supercomputing, vol. 72, no. 11, pp. 4233–4246, 2016.

[27] Q. Huang and H. Li, "An efficient public-key searchable encryption scheme secure against inside keyword guessing attacks," Information Sciences, vol. s 403ÍC404, pp. 1–14, 2017.

[28] P. Xu, X. Tang, W. Wang, H. Jin, and L. T. Yang, "Fast and parallel keyword search over public-key ciphertexts for cloud-assisted iot," IEEE Access, vol. 5, pp. 24 775–24 784, 2017.

[29] A. Sahai and B. Waters, "Fuzzy identity-based encryption," in Annual International Conference on the Theory and Applications of Cryptographic Techniques. Springer, 2005, pp. 457–473.

[30] Q. Zheng, S. Xu, and G. Ateniese, "Vabks: Verifiable attribute-based keyword search over outsourced encrypted data," in IEEE INFOCOM, 2015, pp. 522–530.

[31] M. H. Ameri, M. Delavar, J. Mohajeri, and M. Salmasizadeh, "A key-policy attribute-based temporary keyword search scheme for secure cloud storage," IEEE Transactions on Cloud Computing, vol. PP, no. 99, pp. 1–1, 2018.

[32] J. Cui, H. Zhou, H. Zhong, and Y. Xu, "Akser: Attribute-based keyword search with efficient revocation in cloud computing," Information Sciences, vol. 423, 2017.

[33] S. Wang, D. Zhang, Y. Zhang, and L. Liu, "Efficiently revocable and searchable attribute-based encryption scheme for mobile cloud storage," IEEE Access, 2018.

[34] M. Ru, Y. Zhou, J. Ning, K. Liang, J. Han, and W. Susilo, "An efficient key-policy attribute-based searchable encryption in prime-order groups," 2017.

[35] M. Bellare, R. Canetti, and H. Krawczyk, "Keying hash functions for message authentication," in Annual International Cryptology Conference. Springer, 1996, pp. 1–15.

[36] D. Boneh and M. Franklin, "Identity-based encryption from the weil pairing," in Annual international cryptology conference. Springer, 2001, pp. 213–229.

[37] J. W. Byun, D. H. Lee, and J. Lim, "Efficient conjunctive keyword search on encrypted data storage system," in European Public Key Infrastructure Workshop. Springer, 2006, pp. 184–196.

[38] J. W. Byun, H. S. Rhee, H.-A. Park, and D. H. Lee, "Off-line keyword guessing attacks on recent keyword search schemes over encrypted data," Lecture Notes in Computer Science, vol. 4165, pp. 75–83, 2006.

[39] A. Shamir, "How to share a secret," Communications of the ACM, vol. 22, no. 11, pp. 612–613, 1979.

[40] P. Golle, J. Staddon, and B. Waters, "Secure conjunctive keyword search over encrypted data," in International Conference on Applied Cryptography and Network Security. Springer, 2004, pp. 31–45.