Prime Inner Product Encoding for Effective Wildcard-Based Multi-Keyword Fuzzy Search

Qin Liu, *Member, IEEE*, Yu Peng, *Student Member, IEEE*, Shuyu Pei, *Student Member, IEEE*, Jie Wu, *Fellow, IEEE*, Tao Peng, *Member, IEEE*, and Guojun Wang, *Member, IEEE*

Abstract—With the prevalence of cloud computing, a growing number of users are delegating clouds to host their sensitive data. To preserve user privacy, it is suggested that data is encrypted before outsourcing. However, data encryption makes keyword-based searches over ciphertexts extremely difficult. This is even challenging for *fuzzy search* that allows uncertainties or misspellings of keywords in a query. In this article, we propose a prime inner product encoding (PIPE) scheme, which makes use of the *indecomposable* property of prime numbers to provide efficient, highly accurate, and flexible multi-keyword fuzzy search. Our main idea is to encode either a query keyword or an index keyword into a vector filled with primes or reciprocals of primes, such that the result of vectors' inner product is an integer only when two keywords are similar. Specifically, we first construct PIPE_0 that is secure in the known ciphertext model. Unlike existing works that have difficulty supporting AND and OR semantics simultaneously, PIPE_0 gives users the flexibility to specify different search semantics in their queries. Then, we construct PIPE_S that subtly adds random noises to a query vector to resist linear analyses. Both theoretical analyses and experiment results demonstrate the effectiveness of our scheme.

Index Terms—Data outsourcing, fuzzy search, primes, searchable encryption, wildcard

1 Introduction

Cloud computing, as a new service-oriented computing paradigm, centralizes massive computing and storage resources while delivering resources as services in a *pay-as-you-use* fashion. In other words, everything is a service (XaaS) in cloud computing, where customers enjoy various services anytime and anywhere, using any kinds of devices connecting to the Internet. Although cloud computing provides overwhelming benefits to consumers, such as elasticity and scalability, the application of cloud computing is still far from expected. The main reason is that customers worry that their sensitive data may be deliberately or unintentionally leaked by the cloud service provider (CSP). State-of-the-art CSPs experience noteworthy outages and security breaches from time to time. For example, recent news about Apple iCloud leaking out celebrities' sensitive photos and Amazon S3 leaking personal medical data. Therefore, achieving secure cloud services plays an important role in the development of cloud computing.

To protect user privacy from the CSP, existing research suggests encrypting data before outsourcing. This makes traditional data services like keyword-based searches very challenging. The simple solution of downloading all the encrypted data and decrypting them locally is extremely expensive. Therefore, investigating an efficient search service over ciphertexts becomes a paramount urgency.

Searchable encryption (SE) [1], [2], [3], [4], [5], [6], [7], [8], [9], [10], [11], [12], [13] has been proposed to perform secure searches over encrypted data. In SE, a user first builds searchable indexes for a collection of files and then uploads files and indexes in encrypted forms to a cloud. Later, the user generates an encrypted query (referred to as a trapdoor) to efficiently retrieve files containing specific keywords while keeping files and keywords secret. Although SE schemes allow the user to retrieve data of interest in a privacy-preserving way, almost all of them handle exact keyword matching. That is, the misspelling of a keyword will render an unsatisfactory query failure. In many cases, the user is unsure of the exact spelling of a keyword, but still wants to retrieve files as accurately as possible. Therefore, the feature of supporting approximate keyword matching (also known as *fuzzy search*) is especially important for cloud services when the user has limited knowledge about the underlying data she is searching for.

To improve the user experience of query services, Li *et al.* [14], [15] proposed the fuzzy search schemes that exploited edit distance to quantify keyword similarity. The main drawback of their schemes is the requirement of a predefined dictionary that covers possible keyword misspellings, making searches inefficient. Moreover, it permits only a single keyword in a query, requiring many rounds to support multi-keyword search. From the perspective of privacy protection, indexes and queries are encrypted in a deterministic way that disables *index indistinguishability* and *trapdoor unlinkability*. That is, given a set of encrypted indexes (or a

- *Qin Liu, Yu Peng, and Shuyu Pei are with the College of Computer Science and Electronic Engineering, Hunan University, Changsha, Hunan Province 410082, P.R. China. E-mail: {gracelq628, pengyu411, peishuyu}@hnu.edu.cn.*
- *Jie Wu is with the Department of Computer and Information Sciences, Temple University, Philadelphia, PA 19122 USA. E-mail: jiewu@temple.edu.*
- *Tao Peng and Guojun Wang are with the School of Computer Science and Cyber Engineering, Guangzhou University, Guangzhou, Guangdong Province 510006, P.R. China. E-mail: pengtao_work@163.com, csgjwang@gzhu.edu.cn.*

Manuscript received 25 July 2019; revised 13 April 2020; accepted 26 August 2020. Date of publication 1 September 2020; date of current version 8 August 2022. (Corresponding author: Qin Liu.)
Digital Object Identifier no. 10.1109/TSC.2020.3020688

set of trapdoors), their scheme is able to link one encryption to another if they are for the same keywords. Since then, a few works [16], [17], [18], [19], [20], [21], [22], [23], [24], [25], [26], [27], [28], [29], [30] have been conducted to improve search efficiency and user privacy or to support multi-keyword fuzzy searches. However, the research in this field is still in its infancy, and enabling effective fuzzy search in cloud computing remains a challenging problem.

In this paper, we propose a prime inner product encoding (PIPE) scheme, which allows a user to effectively search multiple *unsure* keywords over encrypted data. In our PIPE scheme, a user replaces an uncertain character in a keyword with a wildcard and the similarity between two keywords is measured by their edit distance excluding the number of wildcards. For simplicity, notation "*" is used as a wildcard throughout this paper to denote an unsure letter. For example, a doctor can enter "arterioscler****" instead of "arteriosclerosis" to retrieve the medical records containing the keyword "arteriosclerosis" if she is unsure about the last four letters of this keyword. Differing from previous wildcard-based fuzzy search schemes [14], [15] that require a predefined dictionary, our PIPE scheme exploits the *indecomposable property* of primes to provide an efficient fuzzy search. Our main idea is to encode either a query keyword or an index keyword as a vector, the elements of which are set to primes or the reciprocals of primes. Because a prime number has no positive divisors other than 1 and itself, the result of vectors' inner product is an integer only when two keywords are similar. To support multi-keyword fuzzy search, keyword vectors are organized into a matrix and matrix multiplication is exploited to support AND/OR queries.

Moreover, as the work in [21], [22], [23], [24], [25], [26], vectors are protected with the secure k-nearest neighbor scheme (KNN for short) [31]. KNN with splitting and randomization mechanisms enables index indistinguishability and trapdoor unlinkability, but has failed to resist linear analyses [32]. Therefore, we first provide a basic PIPE construction, denoted by PIPE_0, which is secure in the known ciphertext model. Then, we provide an advanced construction, denoted by PIPE_S, by subtly adding random noises to a query vector, to achieve enhanced privacy in the known background model. Compared with existing secure fuzzy search schemes, our PIPE scheme has the following merits: 1) *Efficiency*. It eliminates requirements for a predefined dictionary and thus, enables efficient searches. 2) *High accuracy*. It reduces false negatives and false positives caused by specific data structures (e.g., Bloom filters [33] and the locality-sensitive hashing (LSH) [34] utilized in [21], [22], [23]) and thus, allows a user to retrieve files as accurately as possible. 3) *Flexibility*. Unlike existing works that have difficulty supporting AND and OR semantics simultaneously, our scheme gives the user the flexibility to specify different search semantics in their queries. 4) *Enhanced privacy*. Our scheme is resistant to linear analyses but maintains index indistinguishability and trapdoor unlinkability. The main contributions of this paper are summarized as follows:

- To the best of our knowledge, this is the first attempt to devise a secure and effective fuzzy search scheme that simultaneously supports AND and OR search semantics.
- Two constructions are provided to achieve enhanced privacy in different security models.
- We evaluate the performance of our PIPE scheme on a real dataset. Experiment results demonstrate its efficiency and accuracy.

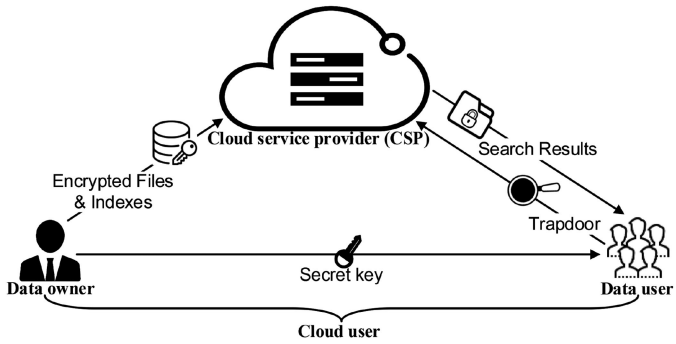


Fig. 1. System model.

The rest of the paper is organized as follows. We provide the preliminaries in Section 2, before the overview of this work in Section 3. We construct the basic and advanced schemes in Sections 4 and 5, respectively. Then, we evaluate our scheme in Section 6, before introducing related work in Section 7. Finally, we conclude this paper in Section 8.

2 Preliminaries

This section first introduces our models and goals, and then describes the cryptographic preliminaries.

2.1 System Model

Fig. 1 shows our system model, which consists of three types of entities: the data owner, the data user, and the cloud service provider (CSP). The data owner and the data user are collectively referred to as the cloud user. The CSP as the owner of cloud platforms provides data storage and query services to cloud users.

Given a collection of files $\mathcal{D}$ and the universal keyword set $\mathcal{W}$ extracted from $\mathcal{D}$, the data owner first builds a set of searchable indexes and then uploads both the encrypted files and the encrypted indexes, $\{\mathcal{C}, \mathcal{I}\}$, to the cloud. To retrieve files matching query Q, the data user first requests secret key SK from the data owner and then sends a trapdoor, T_Q, to the CSP. Upon receiving the search request, the CSP evaluates T_Q on $\mathcal{I}$ and returns the search results, $\mathcal{C}_Q$, to the data user who will locally recover file contents.

2.2 Adversary Model

The cloud user is assumed to be fully trusted. Once mutual authentication has finished, the secret keys will be transmitted through secure channels that are secured under existing security protocols such as SSL and SSH. The CSP is the potential adversary and is assumed to be *honest but curious* [35]. That means, the CSP will correctly execute a predefined protocol, but may try to learn some additional information outside its permission. In terms of the information available to the cloud server, we mainly consider the following models used in existing work [21], [22], [23], [24].

Known Ciphertext Model. The cloud server only knows the encrypted data set, the secure indexes, the submitted trapdoors, and the returned search results.

Known Background Model. The cloud server knows additional background information besides what is known in the known ciphertext model. The background refers to the statistical information about the files and queries (e.g., file distribution and keyword frequency), which can be used to infer certain (plaintext, ciphertext) pairs.

While data privacy can be preserved through standard symmetric key encryption (SKE), e.g., AES, we are committed to protect cloud users' privacy in the following aspects:

- *Keyword secrecy*. The CSP cannot deduce the contents of keywords from the encrypted indexes and trapdoors.
- *Index indistinguishability and trapdoor unlinkability*. Given a set of encrypted indexes (or a set of trapdoors), the CSP cannot decide whether they are generated for the same keywords or not.

2.3 Design Goals

Our design goal is to offer a secure and effective search service in clouds by simultaneously achieving the following:

- *Efficiency*. The user can perform multi-keyword fuzzy searches over encrypted data efficiently.
- *High accuracy*. The user can retrieve files of interest as accurately as possible.
- *Flexibility*. The user can specify different semantics in fuzzy-keyword searches.
- *Enhanced privacy*. Keyword secrecy, index indistinguishability, and trapdoor unlinkability are preserved against the CSP under linear analyses.

2.4 Secure k-Nearest Neighbor Scheme

KNN [31] allows efficient computation of the k-nearest neighbors over an encrypted data set and will be applied to encrypt both our indexes and queries. Let $\kappa \in \mathbb{N}$ denote the security parameter, which should be long enough to prevent brute force attacks (e.g., 128 bits). Given a vector $\mathbf{v}$, its ith element is denoted by $\mathbf{v}[i]$. Let notations "$\star$" and "$\cdot$" denote matrix multiplication and vector inner product, and let $\mathbf{M}^{-1}$ and $\mathbf{M}^T$ denote the inverse and transposed matrices of $\mathbf{M}$, respectively. As in the work in [21], a vector is expanded to $d \geq \kappa$ dimensions before encryption. Therefore, the process of adding artificial dimensions in [31] can be omitted. KNN tailored for our PIPE scheme is as follows:

- $Key(1^\kappa) \rightarrow sk$: It takes a security parameter $\kappa \in \mathbb{N}$ as input and generates the secret key $sk = (\mathbf{M}_1, \mathbf{M}_2, \mathbf{s})$, where $\mathbf{M}_1, \mathbf{M}_2 \in \mathbb{R}^{d \times d}$ are invertible matrices and $\mathbf{s}$ is a d-dimensional binary vector. Note that the number of 0 s should be approximately equal to the number of 1 s in the split vector $\mathbf{s}$ to maximize the randomness.
- $EncI(\mathbf{p}, sk) \rightarrow \mathbf{p}'$: It splits an index vector $\mathbf{p} \in \mathbb{R}^d$ into two vectors, $(\mathbf{p}_a, \mathbf{p}_b)$, as follows: for $i = 1$ to d, if $\mathbf{s}[i] = 1$, $\mathbf{p}[i]$ is randomly split into $\mathbf{p}_a[i], \mathbf{p}_b[i]$ such that $\mathbf{p}_a[i] + \mathbf{p}_b[i] = \mathbf{p}[i]$; if $\mathbf{s}[i] = 0$, both $\mathbf{p}_a[i]$ and $\mathbf{p}_b[i]$ are set to $\mathbf{p}[i]$. It encrypts $(\mathbf{p}_a, \mathbf{p}_b)$ and outputs $\mathbf{p}' = (\mathbf{p}'_a, \mathbf{p}'_b)$ where $(\mathbf{p}'_a = \mathbf{M}_1^T \star \mathbf{p}_a, \mathbf{p}'_b = \mathbf{M}_2^T \star \mathbf{p}_b)$.

TABLE 1
Summary of Notations

$\mathcal{D}$	A set of n files $\{D_1, \ldots, D_n\}$
$\mathcal{W}$	A set of m keywords $\{w_1, \ldots, w_m\}$
$\mathcal{C}$	A set of n ciphertexts $\{C_1, \ldots, C_n\}$
L	Max length of universal keywords
$\mathcal{W}_i$	A set of m_i keywords $\{w_{i,1}, \ldots, w_{i,m_i}\}$ in file D_i
I_i	An encrypted index built for D_i
$\widetilde{\mathcal{W}}_j$	A set of n_j keywords $\{\widetilde{w}_{j,1}, \ldots, \widetilde{w}_{j,n_j}\}$ in query Q_j
T_{Q_j}	A trapdoor of query Q_j
$\mathbb{A}$	A dictionary of 26 English characters ("a",...,"z")
$\mathbb{S}$	A dictionary of L dummy characters disjoint with $\mathbb{A}$
$\mathcal{P}$	A sequence of L primes $(p_1, \ldots, p_L)$

- $EncQ(\mathbf{q}, sk) \rightarrow \mathbf{q}'$: It splits a query vector $\mathbf{q} \in \mathbb{R}^d$ into two vectors, $(\mathbf{q}_a, \mathbf{q}_b)$, as follows: for $i = 1$ to d, if $\mathbf{s}[i] = 1$, both $\mathbf{q}_a[i]$ and $\mathbf{q}_b[i]$ are set to $\mathbf{q}[i]$; if $\mathbf{s}[i] = 0$, $\mathbf{q}[i]$ is randomly split into $\mathbf{q}_a[i]$ and $\mathbf{q}_b[i]$ such that $\mathbf{q}[i] = \mathbf{q}_a[i] + \mathbf{q}_b[i]$. It encrypts $(\mathbf{q}_a, \mathbf{q}_b)$ and outputs $\mathbf{q}' = (\mathbf{q}'_a, \mathbf{q}'_b)$ where $(\mathbf{q}'_a = \mathbf{M}_1^{-1} \star \mathbf{q}_a, \mathbf{q}'_b = \mathbf{M}_2^{-1} \star \mathbf{q}_b)$.
- $Search(\mathbf{p}', \mathbf{q}') \rightarrow v$: It calculates $v = \mathbf{p'}_a \cdot \mathbf{q}'_a + \mathbf{p'}_b \cdot \mathbf{q}'_b$ as the search result. Note that

$$\begin{aligned} &\mathbf{p'}_a \cdot \mathbf{q}'_a + \mathbf{p'}_b \cdot \mathbf{q}'_b \\ &= (\mathbf{M}_1^T \star \mathbf{p}_a) \cdot (\mathbf{M}_1^{-1} \star \mathbf{q}_a) + (\mathbf{M}_2^T \star \mathbf{p}_b) \cdot (\mathbf{M}_2^{-1} \star \mathbf{q}_b) \\ &= \mathbf{p}_a^T \star \mathbf{q}_a + \mathbf{p}_b^T \star \mathbf{q}_b = \mathbf{p} \cdot \mathbf{q}. \end{aligned}$$

Therefore, the sum of inner products of encrypted vectors is equivalent to the inner product of the original vectors. Moreover, we also utilize pseudorandom functions (PRF), which are polynomial-time computable functions indistinguishable from random functions by any probabilistic polynomial-time (PPT) adversary.

3 Scheme Overview

This section will introduce related notations and definitions, and outline the working process of PIPE.

3.1 Notations and Definitions

In our PIPE scheme, the set of all binary strings of length η is denoted by $\{0,1\}^\eta$ and the set of finite binary strings by $\{0,1\}^*$. Notations $[\eta]$ and $[\eta_1 \sim \eta_2]$ represent the set of integers in $\{1, \ldots, \eta\}$ and $\{\eta_1, \ldots, \eta_2\}$, respectively. Given a vector $\mathbf{v}$, its ith element is denoted by $\mathbf{v}[i]$. Given a matrix $\mathbf{M}$, the element in its ith row and jth column is denoted by $\mathbf{M}[i][j]$, all elements in its ith row are denoted by $\mathbf{M}[i][*]$, and all elements in its jth column are denoted by $\mathbf{M}[*][j]$. If s is a string, then $||s||$ refers to its length and $s[i]$ refers to its ith character. Given a dictionary of L characters $\mathbb{S} = (s_1, \ldots, s_L)$, its ith character is denoted by $\mathbb{S}[i]$, and the concatenation of the first l characters is denoted by $\langle s_1, \ldots, s_l \rangle$. For quick reference, the most relevant notations are shown in Table 1.

There are two types of keywords: *exact* keywords and *fuzzy* keywords. Each character of an exact keyword is chosen from the English alphabet $\mathbb{A}$. A fuzzy keyword contains symbol "$*$", which suggests that the character that should be placed at the relevant position is uncertain. We assume that $\mathcal{W}_i$ associated with file D_i includes only exact keywords where $\mathcal{W} = \mathcal{W}_1 \cup \cdots \cup \mathcal{W}_n$, and that $\widetilde{\mathcal{W}}_j$ associated with

query Q_j may contain both kinds of keywords. Let $w_{i,k}$ and $\widetilde{w}_{j,k}$ denote the kth keyword in $\mathcal{W}_i$ and $\widetilde{\mathcal{W}}_j$, respectively. Two keywords are considered *similar* if their distance is 0. The distance between keywords $w_{i,k} \in \mathcal{W}_i$ and $\widetilde{w}_{j,l} \in \widetilde{\mathcal{W}}_j$, denoted by $dist(w_{i,k}, \widetilde{w}_{j,l})$, is defined as follows:

Definition 1 (Distance). *Let e_1 be the number of operations required to transform w_i to w_j and let e_2 be the number of "$*$"s in keywords. We have $dist(w_i, w_j) = e_1 - e_2$.*

That is, for keywords w_i and w_j, $dist(w_i, w_j)$ is determined by the edit distance excluding the number of symbol "$*$"s. For example, $dist("hello","h*llo") = 0$ and $dist("hello","hol*o") = 1$. Therefore, "$hello$" is similar to "$h*llo$" and is dissimilar to "$hol*o$".

A cloud user issues multi-keyword AND/OR queries. Each keyword in an AND query is able to further narrow down the search results, but the opposite is true in an OR query. A query Q_j matching a file D_i, denoted by $Q_j \bowtie D_i$ (mismatching is denoted by $Q_j \not\bowtie D_i$), is defined as follows:

Definition 2 (Matching). *For an AND query, $Q_j \bowtie D_i$ if, for each keyword $\widetilde{w}_{j,l} \in \widetilde{\mathcal{W}}_j$ there exists a keyword $w_{i,k} \in \mathcal{W}_i$ such that $dist(\widetilde{w}_{j,l}, w_{i,k}) = 0$. For an OR query, $Q_j \bowtie D_i$ if there exists a keyword $\widetilde{w}_{j,l} \in \widetilde{\mathcal{W}}_j$ such that $dist(\widetilde{w}_{j,l}, w_{i,k}) = 0$ where $w_{i,k} \in \mathcal{W}_i$. such that $dist(\widetilde{w}_{j,l}, w_{i,k}) = 0$ where $w_{i,k} \in \mathcal{W}_i$.*

In other words, for AND queries, $Q_j \bowtie D_i$ if each keyword in $\widetilde{\mathcal{W}}_j$ is similar to a certain keyword in $\mathcal{W}_i$. For OR queries, $Q_j \bowtie D_i$ if at least one keyword in $\widetilde{\mathcal{W}}_j$ is similar to a certain keyword in $\mathcal{W}_i$. For example, given keyword sets $\mathcal{W}_1 = \{"hello","key"\}$, $\mathcal{W}_2 = \{"hello","world"\}$, and $\widetilde{\mathcal{W}} = \{"h*llo","k*y"\}$, we have $Q \bowtie D_1$ and $Q \not\bowtie D_2$ for AND queries, but $Q \bowtie D_1$ and $Q \bowtie D_2$ for OR queries.

3.2 The Definition of PIPE

A PIPE scheme is a protocol between the cloud user and the CSP as follows:

- $GenKey(1^\kappa) \rightarrow SK$: The cloud user takes the security parameter κ as input and outputs the secret key SK.
- $BuildIndex(\mathcal{W}_i, SK) \rightarrow I_i$: Given a set of m_i keywords $\mathcal{W}_i$, the cloud user builds an encrypted index I_i for file D_i with the secret key SK. A collection of encrypted indexes is set to $\mathcal{I} = \{I_1, \ldots, I_n\}$.
- $Trapdoor(\widetilde{\mathcal{W}}_j, SK) \rightarrow T_{Q_j}$: Given a set of n_j keywords query $\widetilde{\mathcal{W}}_j$, the cloud user creates a trapdoor T_{Q_j} for query Q_j based on the secret key SK.
- $Search(I_i, T_{Q_j}) \rightarrow \{0, 1\}$: The CSP evaluates the trapdoor T_{Q_j} on the encrypted index I_i and output 1 if $Q_j \bowtie D_i$, otherwise it outputs 0.

Definition 3 (Correctness of PIPE). *Given the secret key SK generated by algorithm $GenKey$, an encrypted index I_i generated by algorithm $BuildIndex$, and a trapdoor T_{Q_j} generated by algorithm $Trapdoor$, our PIPE scheme is correct if the following holds: algorithm $Search(I_i, T_{Q_j})$ output 1 if $Q_j \bowtie D_i$, and it outputs 0 if $Q_j \not\bowtie D_i$.*

3.3 Security Definition

We recall the semantic security definitions from [3], and follow their approach that utilizes history $\mathcal{H}$, view $\mathcal{V}$, and trace $Tr(\mathcal{H})$ to capture what is being revealed during the search process. As defined in [3], a history $\mathcal{H}$ consists of a file collection and a set of keywords that the cloud user wishes to search for. Given a history, we refer to what an adversary actually gets to "see", which consists of encrypted searchable indexes and trapdoors, as view $\mathcal{V}$. The trace of a history $Tr(\mathcal{H})$ consists of exactly the information we are willing to leak about the history and nothing else. Different adversary models cause different search leakages, and the trace of our PIPE schemes will be listed in detail in Sections 4 and 5, respectively. Let $\kappa \in \mathbb{N}$ be the security parameter, $\mathcal{A}$ be an adversary, and $\mathcal{S}$ be a simulator. We consider the following probabilistic experiments:

- $\mathbf{Real}_{\mathcal{A}}(\kappa)$: The challenger runs $GenKey(1^\kappa)$ to generate secret key SK. Adversary $\mathcal{A}$ outputs $\mathcal{D}$ and receives $(\mathcal{I} = \{I_1, \ldots, I_n\}, \mathcal{C} = \{C_1, \ldots, C_n\})$ from the challenger so that for $i \in [n] I_i \leftarrow BuildIndex(\mathcal{W}_i, SK)$ and C_i is the ciphertext of D_i encrypted with SKE. $\mathcal{A}$ makes a polynomial number of queries $\mathbb{Q} = (Q_1, \ldots, Q_q)$ and for each query $Q_j \in \mathbb{Q}$, $\mathcal{A}$ receives a trapdoor T_{Q_j} from the challenger such that $T_{Q_j} \leftarrow Trapdoor(\widetilde{\mathcal{W}}_j, SK)$. Finally, $\mathcal{A}$ outputs view $\mathcal{V} = (\mathcal{I}, \mathcal{C}, \mathbb{T})$, where $\mathbb{T} = (T_{Q_1} \ldots, T_{Q_q})$.
- $\mathbf{Sim}_{\mathcal{A},\mathcal{S}}(\kappa)$: Adversary $\mathcal{A}$ outputs $\mathcal{D}$. Given history $\mathcal{H}$, simulator $\mathcal{S}$ generates and sends a pair $(\mathcal{I}, \mathcal{C})$ to $\mathcal{A}$. $\mathcal{A}$ makes a polynomial number of queries $\mathbb{Q} = (Q_1, \ldots, Q_q)$. Given trace $Tr(\mathcal{H})$, $\mathcal{S}$ returns an appropriate trapdoor T_{Q_j} for each query $Q_j \in \mathbb{Q}$. Finally, $\mathcal{A}$ outputs view $\mathcal{V} = (\mathcal{I}, \mathcal{C}, \mathbb{T})$, where $\mathbb{T} = (T_{Q_1} \ldots, T_{Q_q})$.

Definition 4 (Semantic security of PIPE_0). *Given two* Views *with the same* trace *where one view is generated by a real experiment and the other is simulated by a simulator. Our PIPE_0 scheme is semantically secure under the known-ciphertext model if, the probability of distinguishing them is negligible for all PPT adversaries $\mathcal{A}$.*

Definition 5 (Semantic security of PIPE_S). *Given a set of (query keyword, trapdoor) pairs included in the* trace *and two* Views *with the same* trace *where one view is generated by a real experiment and the other is simulated by a simulator. Our PIPE_S scheme is semantically secure under the known-background model if, the probability of distinguishing them is negligible for all PPT adversaries $\mathcal{A}$.*

4 Basic PIPE Construction

This section will present the basic PIPE construction, which is secure in the known-ciphertext model.

4.1 Main Idea

Our PIPE scheme is built on top of the index vector encoding algorithm (Algorithm 1) and the query vector encoding algorithm (Algorithm 2). The basic idea is to encode either a query keyword or an index keyword into a vector, the elements of which are set to primes or the reciprocals of primes, ensuring that the result of vectors' inner product is an integer only when two keywords are similar.

The first step (lines 1-2) of Algorithm 1 is to pad each keyword with dummy characters in $\mathbb{S}$ to hide its genuine length. Let L denote the max length of universal keywords.

After padding, the length of each keyword is equal to L. The second step (lines 3-4) is to initialize each element of the index vector with 1. The third step (lines 5-7) is to place the reciprocals of specified primes in appropriate positions of the index vector. For the lth character $w_{i,k}[l]$ of keyword $w_{i,k}$, the corresponding prime is p_l and the corresponding position is calculated as $pos_l = F_{k_f}(w_{i,k}[l])$. Therefore, $\mathbf{p}_{i,k}[pos_l]$ will be multiplied by $1/p_l$, for $l \in [L]$.

In Algorithm 2, the padding step (lines 1-2) and the initialization step (lines 3-4) are the same as those of Algorithm 1. The third step (lines 5-15) will place specified primes in appropriate positions of the index vector. For the lth character $\widetilde{w}_{j,k}[l]$ of keyword $\widetilde{w}_{j,k}$, there are two cases: (1) If $\widetilde{w}_{j,k}[l] \neq'' *''$, lines 6-8 calculate $pos_l = F_{k_f}(\widetilde{w}_{j,k}[l])$ and multiply $\mathbf{q}_{j,k}[pos_l]$ by p_l; (2) If $\widetilde{w}_{j,k}[l] ='' *''$, it means that an arbitrary character in $\mathbb{A} \cup \mathbb{S}$ could be located at the lth position of a keyword. Lines 10-15 calculate $pos_{l_i} = F_{k_f}(\mathbb{A}[i])$ for $1 \leq i \leq 26$ and $pos_{l_i} = F_{k_f}(\mathbb{S}[i-26])$ for $26 < i \leq 26 + L$, and multiply $\mathbf{q}_{j,k}[pos_{l_x}]$ by p_l for $x \in [26 + L]$. To add randomness to the results of inner products, the last line of both algorithms fills partial remaining elements with random numbers.

Based on the above encoding algorithms, we construct matrices from vectors to support multi-keyword queries. Specifically, $\mathcal{W}_i$ that contains m_i keywords will be constructed as an $m_i \times d$ matrix $\mathbf{A}_i$. For $k \in [m_i]$, the elements in the kth row of $\mathbf{A}_i$ are set to the elements of the kth index keyword's vector, expressed as $\mathbf{A}_i[k][*] \leftarrow \mathbf{p}_{i,k}$. Similarly, $\widetilde{\mathcal{W}}_j$ associated with n_j keywords will be constructed as a $d \times n_j$ matrix $\mathbf{B}_j$. For $k \in [n_j]$, the elements in the kth column of $\mathbf{B}_j$ are set to the elements of the kth query keyword's vector, expressed as $\mathbf{B}_j[*][k] \leftarrow \mathbf{q}_{j,k}$. The matching of Q_j and D_i is determined by the result of matrix multiplication. Given an $m_i \times d$ index matrix $\mathbf{A}_i$ and a $d \times n_j$ query matrix $\mathbf{B}_j$, the CSP calculates $\mathbf{A}_i^{\star}\mathbf{B}_j$ and obtains an $m_i \times n_j$ result matrix $\mathbf{R}_{i,j}$. For AND queries, $Q_j \bowtie D_i$ if each column of $\mathbf{R}_{i,j}$ contains one or more integers; For OR queries, $Q_j \bowtie D_i$ if at least one column of $\mathbf{R}_{i,j}$ contains one or more integers. Note that the order of keywords will not impact the result of matching. Hence, the keywords associated with each file/query can be shuffled before the construction of matrix.

Algorithm 1. Index Vector Encoding

Input: the kth keyword $w_{i,k}$ in file D_i, pseudorandom function F's secret key k_f, a sequence of primes $\mathcal{P} = (p_1, \ldots, p_L)$, and a dictionary of dummy characters $\mathbb{S} = (s_1, \ldots, s_L)$

OutPut: index vector $\mathbf{p}_{i,k} \in \mathbb{R}^d$

1: **if** $||w_{i,k}|| < L$ **then**
2: pad $w_{i,k}$ with $\langle s_1, \ldots, s_{L-||w_{i,k}||} \rangle$
3: **for** $1 \leq l \leq d$ **do**
4: set $\mathbf{p}_{i,k}[l] = 1$
5: **for** $1 \leq l \leq L$ **do**
6: set $pos_l = F_{k_f}(w_{i,k}[l])$
7: set $\mathbf{p}_{i,k}[pos_l] = \mathbf{p}_{i,k}[pos_l] \times 1/p_l$
8: choose a random number $z \in [d - L]$ and set z random 1 elements of $\mathbf{p}_{i,k}$ with random integers

4.2 Scheme Construction

Let $\kappa \in \mathbb{N}$ be a security parameter, let $L = \max(||w_1||, \ldots, ||w_m||)$ be the max length of universal keywords, and let $F : \{0,1\}^{\kappa} \times \{0,1\}^{*} \rightarrow \{0,1\}^{\kappa}$ be a PRF. Given a secure KNN scheme as described in Section 2.4, we provide our basic construction, $\text{PIPE}_0 = (GenKey_0, BuildIndex_0, Trapdoor_0, Search_0)$, as follows:

- $GenKey_0(1^{\kappa}) \rightarrow SK$: Given the security parameter κ, the cloud user runs $KNN.Key(1^{\kappa})$ to generate sk. Then, he randomly chooses a sequence of L primes $\mathcal{P} = (p_1, \ldots, p_L)$, a dictionary of L dummy characters $\mathbb{S} = (s_1, \ldots, s_L)$ such that $\mathbb{S} \cap \mathbb{A} = \emptyset$, and κ-bit string k_f. The secret key SK is set as $(sk, k_f, \mathcal{P}, \mathbb{S})$.
- $BuildIndex_0(\mathcal{W}_i, SK) \rightarrow I_i$: For file D_i with m_i keywords $\mathcal{W}_i = \{w_{i,1}, \ldots, w_{i,m_i}\}$, the cloud user builds an $m_i \times d$ matrix $\mathbf{A}_i$ as follows:
  1) For $k \in [m_i]$, he runs Algorithm 1 to output a d-dimensional vector $\mathbf{p}_{i,k}$ for the kth keyword $w_{i,k}$ in $\mathcal{W}_i$.
  2) For $k \in [m_i]$, he sets $\mathbf{A}_i[k][*] \leftarrow \mathbf{p}_{i,k}$.

For $k \in [m_i]$, he runs $KNN.EncI(\mathbf{A}_i[k][*], sk)$ to encrypt the kth row of matrix $\mathbf{A}_i$ and outputs a pair of encrypted vectors $(\mathbf{A}'_{i_a}[k][*], \mathbf{A}'_{i_b}[k][*])$. The encrypted index I_i is set as a pair of encrypted matrices $\mathbf{A}'_i = (\mathbf{A}'_{i_a}, \mathbf{A}'_{i_b})$.

- $Trapdoor_0(\widetilde{\mathcal{W}}_j, SK) \rightarrow T_{Q_j}$: For query Q_j with n_j keywords $\widetilde{\mathcal{W}}_j = \{\widetilde{w}_{j,1}, \ldots, \widetilde{w}_{j,n_j}\}$, the cloud user constructs a $d \times n_j$ matrix $\mathbf{B}_j$ as follows:
  1) For $k \in [n_j]$, he runs Algorithm 2 to output a d-dimensional vector $\mathbf{q}_{j,k}$ for the kth keyword $\widetilde{w}_{j,k}$ in $\widetilde{\mathcal{W}}_j$.
  2) For $k \in [n_j]$, he sets $\mathbf{B}_j[*][k] \leftarrow \mathbf{q}_{j,k}$.

For $k \in [n_j]$, he runs $KNN.EncQ(\mathbf{B}_j[*][k], sk)$ to encrypt the kth column of matrix $\mathbf{B}_j$ and outputs a pair of encrypted vectors $(\mathbf{B}'_{j_a}[*][k], \mathbf{B}'_{j_b}[*][k])$. The trapdoor T_{Q_j} is set as a pair of encrypted matrices $\mathbf{B}'_j = (\mathbf{B}'_{j_a}, \mathbf{B}'_{j_b})$.

- $Search_0(I_i, T_{Q_j}) \rightarrow \{0,1\}$: The CSP computes $\mathbf{R}_{i,j} = \mathbf{A}'_{i_a} \star \mathbf{B}'_{j_a} + \mathbf{A}'_{i_b} \star \mathbf{B}'_{j_b} = \mathbf{A}_i^{\star}\mathbf{B}_j$. For an AND query, it outputs 1 if $\mathbf{R}_{i,j}$ has at least one integer in each column, and 0 otherwise; for an OR query, it outputs 1 if $\mathbf{R}_{i,j}$ has at least one integer element, and 0 otherwise.

Remark 1. The division operation may result in a loss of precision which may impact search accuracy. Thus, the result of $1/3 \times 3$ may be 1.00001 instead of 1. To solve this problem, our method is to round up after the xth decimal point, where x can be tuned in experiments.

Example 1. Suppose that the relationship between files and keywords is as shown in Fig. 2a, and that the cloud user issues queries $Q_1 = \{''hel * o'', '' k * y''\}$ and $Q_2 = \{''hel * o'', '' w * r * d''\}$ to retrieve appropriate files. Given the maximal length of keywords $L = 5$, random primes $\mathcal{P} = (3, 5, 7, 11, 13)$ and dummy characters $\mathbb{S} = (''A'', '' B'', '' C'', '' D'', '' E'')$, index/query vectors are constructed as shown in Fig. 3. To illustrate the working process of the vector encoding algorithms, let us take index keyword "key" and query keyword "$k * y$" as an example. In the first step, two keywords will be padded with characters "A" and "B" to have the maximal length 5. In the second step, two d-dimensional vectors, $\mathbf{p}_{key}$ and $\mathbf{q}_{k*y}$, are constructed and initialized with 1. For the index keyword "key", the reciprocals of primes are placed as follows: since "k" is the first character, $\mathbf{p}_{key}[F_{k_f}(k)]$ will be

Files	Keywords
D_1	{"hello","key"}
D_2	{"key"}
D_3	{"hello","world"}

(a) The relationship between keywords and files.

$$\mathbf{A}_1=\begin{bmatrix}\mathbf{p}_{hello}\\ \mathbf{p}_{key}\end{bmatrix},\mathbf{A}_2=\begin{bmatrix}\mathbf{p}_{key}\end{bmatrix},\mathbf{A}_3=\begin{bmatrix}\mathbf{p}_{hello}\\ \mathbf{p}_{world}\end{bmatrix}$$

$$\mathbf{B}_1=\begin{bmatrix}\mathbf{q}_{hel*o} & \mathbf{q}_{k*y}\end{bmatrix},\mathbf{B}_2=\begin{bmatrix}\mathbf{q}_{hel*o} & \mathbf{q}_{w*r*d}\end{bmatrix}$$

(b) Sample index and query matrices.

$$\mathbf{R}_{1,1}=\mathbf{A}_1\star\mathbf{B}_1=\begin{bmatrix}\mathbf{p}_{hello}\bullet\mathbf{q}_{hel*o} & \mathbf{p}_{hello}\bullet\mathbf{q}_{k*y}\\ \mathbf{p}_{key}\bullet\mathbf{q}_{hel*o} & \mathbf{p}_{key}\bullet\mathbf{q}_{k*y}\end{bmatrix}=\begin{bmatrix}34 & 3.11\\ 18.08 & 21\end{bmatrix},\quad \mathbf{R}_{1,2}=\mathbf{A}_1\star\mathbf{B}_2=\begin{bmatrix}\mathbf{p}_{hello}\bullet\mathbf{q}_{hel*o} & \mathbf{p}_{hello}\bullet\mathbf{q}_{w*r*d}\\ \mathbf{p}_{key}\bullet\mathbf{q}_{hel*o} & \mathbf{p}_{key}\bullet\mathbf{q}_{w*r*d}\end{bmatrix}=\begin{bmatrix}34 & 34.27\\ 18.08 & 46.42\end{bmatrix}$$

$$\mathbf{R}_{2,1}=\mathbf{A}_2\star\mathbf{B}_1=\begin{bmatrix}\mathbf{p}_{key}\bullet\mathbf{q}_{hel*o} & \mathbf{p}_{key}\bullet\mathbf{q}_{k*y}\end{bmatrix}=\begin{bmatrix}18.08 & 21\end{bmatrix},\quad \mathbf{R}_{2,2}=\mathbf{A}_2\star\mathbf{B}_2=\begin{bmatrix}\mathbf{p}_{key}\bullet\mathbf{q}_{hel*o} & \mathbf{p}_{key}\bullet\mathbf{q}_{w*r*d}\end{bmatrix}=\begin{bmatrix}18.08 & 46.42\end{bmatrix}$$

$$\mathbf{R}_{3,1}=\mathbf{A}_3\star\mathbf{B}_1=\begin{bmatrix}\mathbf{p}_{hello}\bullet\mathbf{q}_{hel*o} & \mathbf{p}_{hello}\bullet\mathbf{q}_{k*y}\\ \mathbf{p}_{world}\bullet\mathbf{q}_{hel*o} & \mathbf{p}_{world}\bullet\mathbf{q}_{k*y}\end{bmatrix}=\begin{bmatrix}34 & 3.11\\ 41.68 & 4.22\end{bmatrix},\quad \mathbf{R}_{3,2}=\mathbf{A}_3\star\mathbf{B}_2=\begin{bmatrix}\mathbf{p}_{hello}\bullet\mathbf{q}_{hel*o} & \mathbf{p}_{hello}\bullet\mathbf{q}_{w*r*d}\\ \mathbf{p}_{world}\bullet\mathbf{q}_{hel*o} & \mathbf{p}_{world}\bullet\mathbf{q}_{w*r*d}\end{bmatrix}=\begin{bmatrix}34 & 34.27\\ 41.68 & 181\end{bmatrix}$$

(c) Sample result matrices.

Fig. 2. Working process of PIPE_0. $\mathbf{p}_{w_{i,k}}$ and $\mathbf{q}_{\widetilde{w}_{j,k}}$ are used to denote $\mathbf{p}_{i,k}$ and $\mathbf{q}_{j,k}$, respectively.

multiplied by 1/3; since "e" is the second character, $\mathbf{p}_{key}[F_{k_f}(e)]$ will be multiplied by 1/5; since "y" is the third character, $\mathbf{p}_{key}[F_{k_f}(y)]$ will be multiplied by 1/7; since "A" is the fourth character, $\mathbf{p}_{key}[F_{k_f}(A)]$ will be multiplied by 1/11; since "B" is the fifth character, $\mathbf{p}_{key}[F_{k_f}(B)]$ will be multiplied by 1/13. For the query keyword "$k*y$", the primes are placed as follows: since "k" is the first character, $\mathbf{q}_{k*y}[F_{k_f}(k)]$ will be multiplied by 3; since "$*$" is the second character, an arbitrary element in $\mathbb{A}$ and $\mathbb{S}$ could be located at this position, and $\mathbf{q}_{k*y}[F_{k_f}(a)],\ldots,\mathbf{q}_{k*y}[F_{k_f}(E)]$ will be multiplied by 5; since "y" is the third character, $\mathbf{q}_{k*y}[F_{k_f}(y)]$ will be multiplied by 7; since "A" is the fourth character, $\mathbf{q}_{k*y}[F_{k_f}(A)]$ will be multiplied by 11; since "B" is the fifth character, $\mathbf{q}_{k*y}[F_{k_f}(B)]$ will be multiplied by 13. Due to the indecomposable property of prime numbers, the result of $\mathbf{p}_{key}\cdot\mathbf{q}_{k*y}$ is an integer (equal to 21). The results are in accordance with the fact that two keywords are similar.

Algorithm 2 Query Vector Encoding

Input: the kth keyword $\widetilde{w}_{j,k}$ in query Q_j, PRF F's secret key k_f, a sequence of primes $\mathcal{P}=(p_1,\ldots,p_L)$, an English alphabet $\mathbb{A}$ of length 26, and a dictionary of dummy characters $\mathbb{S}=(s_1,\ldots,s_L)$
OutPut: query vector $\mathbf{q}_{j,k}\in\mathbb{R}^d$

```
if ||w̃_{j,k}|| < L then
    pad w̃_{j,k} with ⟨s_1, ..., s_{L−||w̃_{j,k}||}⟩
for 1 ≤ l ≤ d do
    set q_{j,k}[l] = 1
for 1 ≤ l ≤ L do
    if w̃_{j,k}[l] ≠' *' then
        set pos_l = F_{k_f}(w̃_{j,k}[l])
        set q_{j,k}[pos_l] = q_{j,k}[pos_l] × p_l
    else
        for 1 ≤ i ≤ 26 + L do
            if 1 ≤ i ≤ 26 then
                set pos_{l_i} = F_{k_f}(A[i])
            else
                set pos_{l_i} = F_{k_f}(S[i − 26])
            set q_{j,k}[pos_{l_i}] = q_{j,k}[pos_{l_i}] × p_l
choose a random number z ∈ [d − 26 − L] and set z random 1
    elements of q_{j,k} with random primes outside P
```

Given index/query vectors shown in Fig. 3, corresponding sample matrices are shown in Fig. 2b. If Q_1 is an AND query, D_1 will be returned, since only $\mathbf{R}_{1,1}$ has an integer element in each column; if Q_1 is an OR query, $\{D_1,D_2,D_3\}$ will be returned, since $\mathbf{R}_{1,1},\mathbf{R}_{2,1}$, and $\mathbf{R}_{3,1}$ have at least one integer element. If Q_2 is an AND query, D_3 will be returned, since only $\mathbf{R}_{3,2}$ has an integer element in each column; if Q_2 is an OR query, $\{D_1,D_3\}$ will be returned, since $\mathbf{R}_{1,2}$ and $\mathbf{R}_{3,2}$ have at least one integer element.

4.3 Correctness Analysis

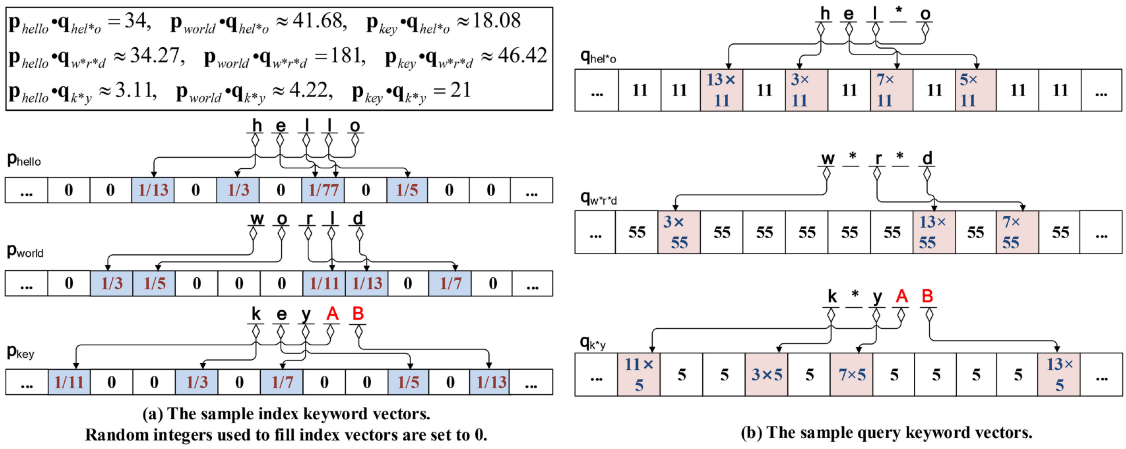
Let $\mathbb{U}=\mathbb{A}\cup\mathbb{S}$ denote the union of the English alphabet and the dummy characters, where $\mathbb{U}[i]$ denotes the ith element in $\mathbb{U}$. We first consider a single-keyword setting, where a file D_i or a query Q_j contains only one keyword, denoted by $w_{i,1}$ and $\widetilde{w}_{j,1}$, respectively. Our basic PIPE construction is considered incorrect if the following occurs:

Case 1. The result of $\mathbf{p}_{i,1}\cdot\mathbf{q}_{j,1}$ is not an integer if keywords $w_{i,1}$ and $\widetilde{w}_{j,1}$ are similar.

Case 2. The result of $\mathbf{p}_{i,1}\cdot\mathbf{q}_{j,1}$ is an integer if keywords $w_{i,1}$ and $\widetilde{w}_{j,1}$ are dissimilar.

For Case 1, a result where $\mathbf{p}_{i,1}\cdot\mathbf{q}_{j,1}$ is not an integer means that at least one reciprocal in $\mathbf{p}_{i,1}$ cannot be eliminated. Due to the construction of the $BuildIndex_0$ algorithm, $\mathbf{p}_{i,1}[F_{k_f}(\mathbb{U}[i])]=1/p_l$ means that $\mathbb{U}[i]$ is the lth character of $w_{i,1}$. If $1/p_l$ cannot be eliminated, $\mathbf{q}_{j,1}[F_{k_f}(\mathbb{U}[i])]$ is set to an integer V that is indivisible by p_l. Due to the construction of the $Trapdoor_0$ algorithm, neither $\mathbb{U}[i]$ nor symbol "$*$" appears in the lth position of $\widetilde{w}_{j,1}$. Therefore, two keywords are dissimilar, which contradicts the assumption, and Case 1 is untrue. For Case 2, a result where $\mathbf{p}_{i,1}\cdot\mathbf{q}_{j,1}$ is an integer means that all reciprocals in $\mathbf{p}_{i,1}$ are eliminated. Given $\mathbf{p}_{i,1}[F_{k_f}(\mathbb{U}[i])]=1/p_l$ denoting that $\mathbb{U}[i]$ is the lth character of $w_{i,1}$, $\mathbf{q}_{j,1}[F_{k_f}(\mathbb{U}[i])]$ is set to p_l or an integer V that is divisible by p_l. Due to the construction of the $Trapdoor_0$ algorithm, it means that either $\mathbb{U}[i]$ or symbol "$*$" is in the lth position of $\widetilde{w}_{j,1}$. In any case, we have two keywords that are similar to contradict the assumption. Therefore, Case 2 is not true, and PIPE_0 is correct in the single-keyword setting.

While extending to the multi-keyword setting, the correctness of our PIPE scheme can be derived as follows: The element in the kth row and lth column of the result matrix $\mathbf{R}_{i,j}$ is the result of $\mathbf{p}_{i,k}\cdot\mathbf{q}_{j,l}$, which is an integer if the kth keyword in $\mathcal{W}_i$ is similar to the lth keyword in $\widetilde{\mathcal{W}}_j$. For AND queries, if each column of $\mathbf{R}_{i,j}$ has at least one integer, i.e., the kth keyword in $\mathcal{W}_i$ is similar to at least one keyword



(a) The sample index keyword vectors. Random integers used to fill index vectors are set to 0.

(b) The sample query keyword vectors.

Fig. 3. Sample vectors output by Algorithms 1 and 2.

in $\widetilde{\mathcal{W}}_j$ for $k \in [m_i]$, then $Q_j \bowtie D_i$ is considered true; For OR queries, if at least one element in $\mathbf{R}_{i,j}$ is an integer, i.e., there exists one keyword in $\mathcal{W}_i$ matching at least one keyword in $\widetilde{\mathcal{W}}_j$, then $Q_j \bowtie D_i$ is considered true. This argument agrees with the definition of matching, and PIPE_0 is correct.

4.4 Security Proof

We first consider the the security of our scheme in the single-keyword setting, where a file D_i or a query Q_j contains only one keyword, denoted by $w_{i,1}$ and $\widetilde{w}_{j,1}$, respectively. Before stating our security theorem, we provide a more formal and concise description of our scheme's leakage:

- *History* $\mathcal{H} = (\mathcal{D}, \mathcal{W}, \mathbb{Q})$ where $\mathcal{D}$ is a collection of files $\{D_1, \ldots, D_n\}$, $\mathcal{W}$ is a set of keywords $\{w_1, \ldots, w_m\}$, and $\mathbb{Q}$ is a sequence of submitted queries $(Q_1, \ldots, Q_q)$.
- *View* $\mathcal{V} = (\mathcal{C}, \mathcal{I}, \mathbb{T})$ is the encrypted form of history under the secret key SK. That is, $\mathcal{C} = \{C_1, \ldots, C_n\}$ is a set of ciphertexts (where C_i is the ciphertext of file $D_i \in \mathcal{D}$ encrypted with SKE), $\mathcal{I} = \{I_1, \ldots, I_n\}$ is a set of encrypted indexes (where I_i is built for file $D_i \in \mathcal{D}$), and $\mathbb{T} = (T_{Q_1}, \ldots, T_{Q_q})$ is a sequence of trapdoors (where T_{Q_j} is the trapdoor of query $Q_j \in \mathbb{Q}$). The CSP can only see views.
- *Trace of history* captures the information that can be learned by the CSP. For history $\mathcal{H} = (\mathcal{D}, \mathcal{W}, \mathbb{Q})$, the trace $Tr(\mathcal{H})$ includes the following information:
  - Size pattern: $(n, |D_1|, \ldots, |D_j|)$, where n is the number of files in $\mathcal{D}$ and $|D_j|$ is the bit length of file D_j.
  - Access pattern: The search results $(\mathcal{R}_1, \ldots, \mathcal{R}_k)$, where $\mathcal{R}_j = \{(i, \mathbf{R}_{i,j}) | Q_j \bowtie D_i, i \in [n]\}$, for $j \in [k]$. The result matrix $\mathbf{R}_{i,j}$ contains only one random integer that equals $\mathbf{p}_{i,1} \cdot \mathbf{q}_{j,1}$.

Theorem 1. *Our basic PIPE scheme is semantically secure in the known ciphertext model if F is a secure PRF and SKE is secure under chosen-plaintext attacks (CPA-secure).*

Proof. We adopt a simulation-based proof similar to the one used in [3]. Let $\mathcal{S}$ denote a simulator that can simulate a view $\bar{\mathcal{V}}$. Our scheme is secure if the CSP cannot distinguish $\bar{\mathcal{V}}$ from $\mathcal{V}$. To achieve this, $\mathcal{S}$ performs the following:

- It randomly picks two invertible matrices $\bar{\mathbf{M}}_1, \bar{\mathbf{M}}_2 \in \mathbb{R}^{d \times d}$ and a d-dimensional binary vector $\bar{\mathbf{s}}$, L primes $\bar{\mathcal{P}} = \{\bar{p}_1, \ldots, \bar{p}_L\}$, and t random integers $\mathcal{X} = \{pos_1, \ldots, pos_t\}$ where $t = 26 + L$ and $1 \le pos_i \le d$ for $i \in [t]$. $\mathcal{S}$ sets $\bar{SK} = (\bar{sk}, \bar{\mathcal{P}}, \mathcal{X})$ where $\bar{sk} = (\bar{\mathbf{M}}_1, \bar{\mathbf{M}}_2, \bar{\mathbf{s}})$.
- $\mathcal{S}$ selects a random $\bar{D}_i \in \{0,1\}^{|D_i|}$ for $D_i \in \mathcal{D}$ and outputs $\bar{\mathcal{C}} = \{\bar{C}_1, \ldots, \bar{C}_n\}$ where $\bar{C}_i$ is the ciphertext of $\bar{D}_i$ encrypted with SKE for $i \in [n]$.
- To generate $\bar{\mathcal{I}}$, $\mathcal{S}$ generates a d-dimensional vector $\bar{\mathbf{p}}'_{i,1}$ for $1 \le i \le n$ as follows:
  1) It constructs a d-dimensional vector $\bar{\mathbf{p}}_{i,1}$ where each element is set to 1.
  2) For $l \in [L]$, it chooses a random integer pos from $\mathcal{X}$ and sets $\bar{\mathbf{p}}_{i,1}[pos] = \bar{\mathbf{p}}_{i,1}[pos] \times 1/\bar{p}_l$.
  3) Other elements of $\bar{\mathbf{p}}_{i,1}$ are filled with random integers.
  4) It runs $KNN.EncI(\bar{\mathbf{p}}_{i,1}, \bar{sk})$ to output $\bar{\mathbf{p}}'_{i,1}$.

  Therefore, $\bar{\mathcal{I}} = \{\bar{\mathbf{p}}'_{1,1}, \ldots, \bar{\mathbf{p}}'_{n,1}\}$.
- To generate $\bar{\mathbb{T}}$, $\mathcal{S}$ constructs a d-dimensional vector $\bar{\mathbf{q}}'_{j,1}$ for $1 \le j \le q$ as follows:
  1) It constructs an empty set R and a d-dimensional vector $\bar{\mathbf{q}}_{j,1}$ where each element is set to 1.
  2) For $i \in [n]$, if $Q_j \bowtie D_i$, it puts $\bar{\mathbf{p}}_{i,1}$ in set R.
  3) For $1 \le l \le d$, it performs as follows:
     a) It constructs an empty set Y.
     b) For each index vector $\bar{\mathbf{p}}_{i,1} \in R$, if $\bar{\mathbf{p}}_{i,1}[l]$ is not an integer, it puts $1/\bar{\mathbf{p}}_{i,1}[l]$ into Y.
     c) If Y is not empty, it sets $\bar{\mathbf{q}}_{j,1}[l] = y$ where y is the least common multiple of elements in Y.
     d) If Y is empty, it sets $\bar{\mathbf{q}}_{j,1}[l]$ to a random prime outside $\bar{\mathcal{P}}$.
  4) It runs $KNN.EncQ(\bar{\mathbf{q}}_{j,1}, \bar{sk})$ to output $\bar{\mathbf{q}}'_{j,1}$.

  Therefore, $\bar{\mathbb{T}} = \{\bar{\mathbf{q}}'_{1,1}, \ldots, \bar{\mathbf{q}}'_{q,1}\}$.
- $\mathcal{S}$ outputs the view $\bar{\mathcal{V}} = (\bar{\mathcal{C}}, \bar{\mathcal{I}}, \bar{\mathbb{T}})$.

Since SKE is secure under chosen-plaintext attacks, no PPT adversary can distinguish between $\bar{\mathcal{C}}$ and $\mathcal{C}$. The indistinguishability of indexes and trapdoors is based on the indistinguishability of KNN and the introduced randomness. KNN is proven to be secure against ciphertext only attacks. The encrypted indexes $\bar{\mathcal{I}}$ and the trapdoors

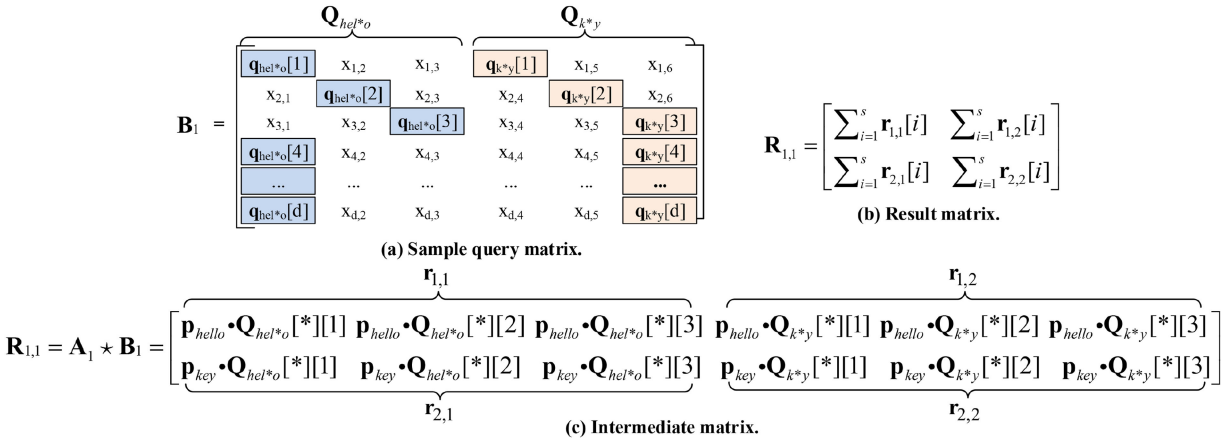


Fig. 4. Working process of $\mathrm{PIPE_S}$. $\mathbf{p}_{w_{i,k}}$ and $\mathbf{Q}_{\widetilde{w}_{j,k}}$ are used to denote $\mathbf{p}_{i,k}$ and $\mathbf{Q}_{j,k}$, respectively.

$\bar{\mathbb{T}}$ generate the same trace as the one that the CSP has. Therefore, we claim that no PPT adversary can distinguish $\bar{\mathcal{V}}$ from $\mathcal{V}$. $\mathrm{PIPE_0}$ is built based on the single-keyword setting, and it is semantically secure in the known ciphertext model. □

5 Advanced PIPE Construction

This section will provide a security-enhanced KNN scheme to resist linear analysis.

5.1 Main Idea

From the construction of KNN [31], we know that the inner product of an index vector and a query vector can be calculated from their encrypted forms: $\mathbf{p} \cdot \mathbf{q} = \mathbf{p}'_a \cdot \mathbf{q}'_a + \mathbf{p}'_b \cdot \mathbf{q}'_b$, where only d variables are unknown to the cloud server, i.e., d elements of the index vector $\mathbf{p}$. In the known-background model, the cloud server can infer certain (plaintext, ciphertext) pairs based on the available background information. As proven in Yao *et al*. [32], an adversary with d query vectors and their encryptions can recover index vectors by solving linear equations.

To resist such an attack, we extend query vectors to random matrices thereby adding noises to the results of inner product. The hardness for initiating such attacks comes from the negligible probability of constructing d correct equations. For all index vectors we have

$$\mathbf{p} \cdot \mathbf{q} \neq \mathbf{p'}_a \cdot \mathbf{q}'_a + \mathbf{p'}_b \cdot \mathbf{q}'_b. \tag{1}$$

Our solution is based on the following key techniques:

Encoding Query Matrices. A query vector $\mathbf{q}_{j,k}$ generated by Algorithm 2 will be encoded as a $d \times s$ matrix $\mathbf{Q}_{j,k}$ by Algorithm 3. For $l \in [d]$, Algorithm 3 sets $\mathbf{q}_{j,k}[l]$ at a random position of the lth row of $\mathbf{Q}_{j,k}$ and fills other positions with random numbers while the following conditions are satisfied:

Condition 1. Each column of matrix $\mathbf{Q}_{j,k}$ contains at least one element of vector $\mathbf{q}_{j,k}$.

Condition 2. The sum of the random numbers at the lth row, denoted as δ_l, is equal to $t_l \mathbf{q}_{j,k}[l]$ where $t_l = 0$ or $(t_l + 1)$ is a prime outside $\mathcal{P}$.

The Way to Determine Similarity. Given an index vector $\mathbf{p}_{i,k} \in \mathbb{R}^d$ generated for keyword $w_{i,k} \in \mathcal{W}_i$ and a query matrix $\mathbf{Q}_{j,l} \in \mathbb{R}^{d \times s}$ generated for keyword $\widetilde{w}_{j,l} \in \widetilde{\mathcal{W}}_j$, an intermediate vector $\mathbf{r}_{k,l} \in \mathbb{R}^s$ is obtained by calculating $\mathbf{p}_{i,k}^T \star \mathbf{Q}_{j,l}$. If the sum of elements in $\mathbf{r}_{k,l}$, $\alpha_{k,l} = \sum_{x=1}^{s} \mathbf{r}_{k,l}[x]$, is an integer, keywords $w_{i,k}$ and $\widetilde{w}_{j,l}$ are considered similar.

Algorithm 3. Query Matrix Encoding

Input: $s \in [2 \sim d]$, query vector $\mathbf{q}_{j,k} \in \mathbb{R}^d$
OutPut: query matrix $\mathbf{Q}_{j,k} \in \mathbb{R}^{d \times s}$
1: construct a matrix $\mathbf{Q}_{j,k} \in \mathbb{R}^{d \times s}$ and initialize each element of $\mathbf{Q}_{j,k}$ with 0
2: **for** $l \in [d]$ **do**
3: set $\mathbf{q}_{j,k}[l]$ at a random position of the lth row of $\mathbf{Q}_{j,k}$ with the constraint of **Condition 1**
4: fill the remaining elements in the lth row of $\mathbf{Q}_{j,k}$ with random numbers with the constraint of **Condition 2**

5.2 Scheme Construction

The main differences compared with $\mathrm{PIPE_0}$ lie in algorithms $Trapdoor_S$ and $Search_S$ as follows:

- $Trapdoor_S(\widetilde{\mathcal{W}}_j, \widetilde{SK}) \rightarrow T_{Q_j}$: For query Q_j containing n_j keywords $\widetilde{\mathcal{W}}_j$, the cloud user chooses a random integer $s \in [2 \sim d]$. To generate a $d \times (sn_j)$ matrix $\widehat{\mathbf{B}}_j$ for $k \in [n_j]$, he performs as follows:
  1) He runs Algorithm 2 to output a d-dimensional vector $\mathbf{q}_{j,k}$ for the kth keyword $\widetilde{w}_{j,k} \in \widetilde{\mathcal{W}}_j$
  2) He runs Algorithm 3 to extend $\mathbf{q}_{j,k}$ to a $d \times s$ matrix $\mathbf{Q}_{j,k}$.
  3) For $l \in [s]$, he sets $\widehat{\mathbf{B}}_j[*][(k-1)s + l] \leftarrow \mathbf{Q}_{j,k}[*][l]$. Matrix $\widehat{\mathbf{B}}_j$ is actually the combination of n_j query keywords' matrices (see Fig. 4a).

For $k \in [sn_j]$, he runs $KNN.EncQ(\widehat{\mathbf{B}}_j[*][k], sk)$ to encrypt the kth column of matrix $\widehat{\mathbf{B}}_j$ and obtains a pair of encrypted vectors $(\widehat{\mathbf{B}}'_{j_a}[*][k], \widehat{\mathbf{B}}'_{j_b}[*][k])$. The trapdoor T_{Q_j} is set as a pair of encrypted matrices $(\widehat{\mathbf{B}}'_{j_a}, \widehat{\mathbf{B}}'_{j_b})$.

- $Search_S(I_i, T_{Q_j}) \rightarrow \{0, 1\}$: Given index matrices $\mathbf{A}'_i = (\mathbf{A}'_{i_a}, \mathbf{A}'_{i_b})$ and query matrices $\widehat{\mathbf{B}}'_j = (\widehat{\mathbf{B}}'_{j_a}, \widehat{\mathbf{B}}'_{j_b})$, the CSP generates an $m_i \times (sn_j)$ intermediate matrix $\widehat{\mathbf{R}}_{i,j}$ by setting $\widehat{\mathbf{R}}_{i,j} = \mathbf{A}'_{i_a} \star \widehat{\mathbf{B}}'_{j_a} + \mathbf{A}'_{i_b} \star \widehat{\mathbf{B}}'_{j_b} = \mathbf{A}^*_i \widehat{\mathbf{B}}_j$. Matrix $\widehat{\mathbf{R}}_{i,j}$ is consisted of $m_i \times n_j$ intermediate vectors (see Fig. 4c), where the elements of $\mathbf{r}_{k,l}$ locate at $\widehat{\mathbf{R}}_{i,j}[k][(l -$

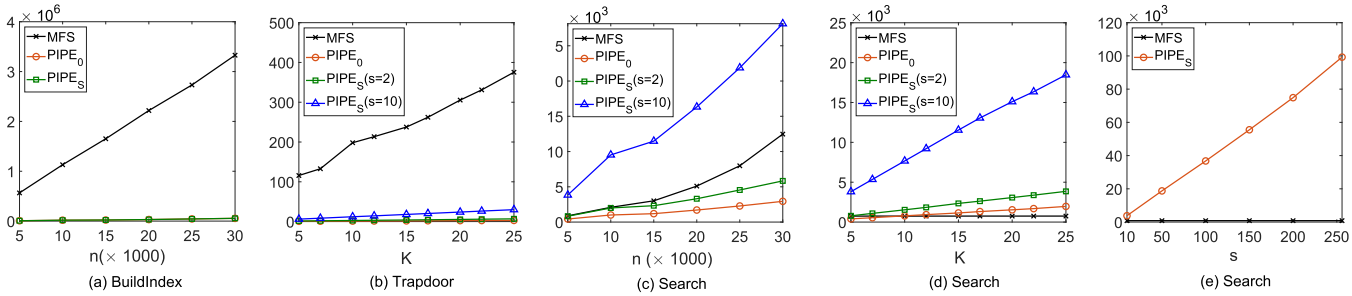


Fig. 5. Comparison of the execution time (ms) between PIPE and MFS. (a) The time for the encryption of n indexes. (b) The time for the encryption of a query of K keywords. (c) The time for searching n files with fixed query keywords $K = 5$. (d) The time for searching K keywords with fixed file size $n = 10,000$. (e) The search time of $\mathrm{PIPE_S}$ under different s with fixed file size $n = 10,000$ and query keywords $K = 5$.

$1)s+1], \ldots, \widehat{\mathbf{R}}_{i,j}[k][ls]$. To determine whether $Q_j \bowtie D_i$ or not, he performs as follows:

1) He constructs a result matrix $\mathbf{R}_{i,j} \in \mathbb{R}^{m_i \times n_j}$
2) For $k \in [m_i]$ and $l \in [n_j]$, he sets $\mathbf{R}_{i,j}[k][l] = \sum_{y=(l-1)s+1}^{y=ls} \widehat{\mathbf{R}}_{i,j}[k][y]$. This is equivalent to setting $\mathbf{R}_{i,j}[k][l] = \sum_{y=1}^{s} \mathbf{r}_{k,l}[y]$ (see Fig. 4b).
3) For an AND query, it outputs 1 if $\mathbf{R}_{i,j}$ has at least one integer in each column, and 0 otherwise; for an OR query, it outputs 1 if $\mathbf{R}_{i,j}$ has at least one integer element, and 0 otherwise.

Remark 2. The number of columns s in query matrix mainly impact the performance of algorithms $Trapdoor_S$ and $Search_S$. As shown in Figs. 5b and 5e, the larger the value of s, the higher the incurred costs. In terms of security level, even if $s = 2$, the adversary cannot obtain any useful information from a single column of $\mathbf{R}$ directly. We believe that even for this low value of s, there is a sufficient measure of security provided.

Example 2. Suppose that $s = 3$, the sample query matrix $\widehat{\mathbf{B}}_1$ for query $Q_1 = \{''hel*o'', ''k*y''\}$ is shown in Fig. 4a, where $\mathbf{Q}_{hel*o}$ and $\mathbf{Q}_{k*y}$ situate at the first s columns and the last s columns of $\widehat{\mathbf{B}}_1$, respectively. Given matrix $\mathbf{A}_1$ as shown in Fig. 2, the result matrix $\mathbf{R}_{i,j}$ and the intermediate matrix $\widehat{\mathbf{R}}_{i,j}$ are shown in Figs. 4b and 4c, respectively.

5.3 Correctness Analysis

The correctness of $\mathrm{PIPE_S}$ is based on the assumption that the introduced randomness in a query matrix will not impact the integral/non-integral property of results. Given an index vector $\mathbf{p}$ and a query vector $\mathbf{q}$, the intermediate vector can be derived as follows:

$$\mathbf{r} = \mathbf{p}_\star^T \mathbf{Q} = (\mathbf{p} \cdot \mathbf{Q}[*][1], \ldots, \mathbf{p} \cdot \mathbf{Q}[*][s]) = \left(\sum_{j=1}^{d} \mathbf{p}[j] x_{j,1}, \sum_{j=1}^{d} \mathbf{p}[j] x_{j,2}, \ldots, \sum_{j=1}^{d} \mathbf{p}[j] x_{j,s}\right),$$

where $\mathbf{Q}$ is $\mathbf{q}$'s query matrix generated by Algorithm 3 and $x_{i,j}$ denotes the element in the ith row and jth column of $\mathbf{Q}$ for $i \in [d]$ and $j \in [s]$. We derive the following equation:

$$\begin{aligned} \alpha &= \sum_{i=1}^{s} \mathbf{r}[i] = \sum_{j=1}^{d} \mathbf{p}[j] x_{j,1} + \cdots + \sum_{j=1}^{d} \mathbf{p}[j] x_{j,s} \\ &= \mathbf{p}[1](x_{1,1} + \cdots + x_{1,s}) + \cdots + \mathbf{p}[d](x_{d,1} + \cdots + x_{d,s}). \end{aligned} \tag{2}$$

Based on *Condition* 1-2, Eq. (2) can be converted to Eq. (3)

$$\begin{aligned} \alpha &= \mathbf{p}[1](\mathbf{q}[1] + \delta_1) + \cdots + \mathbf{p}[d](\mathbf{q}[d] + \delta_d) \\ &= \mathbf{p}[1](\mathbf{q}[1] + t_1 \mathbf{q}[1]) + \cdots + \mathbf{p}[d](\mathbf{q}[d] + t_d \mathbf{q}[d]) \\ &= (1 + t_1)(\mathbf{p}[1]\mathbf{q}[1]) + \cdots + (1 + t_d)(\mathbf{p}[d]\mathbf{q}[d]) \\ &= \sum_{j=1}^{d} \mathbf{p}[j]\mathbf{q}[j] + \sum_{j=1}^{d} t_j \mathbf{p}[j]\mathbf{q}[j] \\ &= \mathbf{p} \cdot \mathbf{q} + X. \end{aligned} \tag{3}$$

For $j \in [d]$, $t_j = 0$ or $(t_j + 1)$ is a prime that is outside $\mathcal{P}$, and the result of $(t_j + 1)\mathbf{p}[j]\mathbf{q}[j]$ is an integer only when $\mathbf{p}[i]\mathbf{q}[i]$ is an integer. Due to the indecomposable property of primes, $\alpha = \sum_{i=1}^{s} \mathbf{r}[i]$ is an integer only when $\mathbf{p} \cdot \mathbf{q}$ is an integer meaning that two keywords are similar. Therefore, $\mathrm{PIPE_S}$ is correct.

5.4 Security Proof

Theorem 2. *The KNN scheme is resilient to linear analyses, if the query vectors are extended into random matrices.*

Proof (sketch). The purpose of extending a query vector to a matrix is to add random noises to the search results so that Eq. (1) holds for all index vectors. First, we show the security of $\alpha = \sum_{i=1}^{s} \mathbf{r}[i]$ as follows: from Eq. (3), we know that $\alpha = \mathbf{p} \cdot \mathbf{q} + X$, where $X \in \mathbb{R}$ is a random number that has no linear relationship with the result of $\mathbf{p} \cdot \mathbf{q}$. Therefore, it is impossible for the adversary to decompose $\mathbf{p} \cdot \mathbf{q}$ from α.

In $\mathrm{PIPE_S}$, query matrices are constructed from query vectors output by Algorithm 2, and algorithms $GenKey_S$ and $BuildIndex_S$ are constructed in the same way as the algorithms in the basic construction. Our main security concern is that the $Search_S$ algorithm leaks an intermediate vector $\mathbf{r}$, which may be used to infer certain sensitive information about underlying data sets and submitted queries. We show the security of $\mathbf{r}$ as follows: for $k \in [s]$, the kth element of $\mathbf{r}$ is the result of the inner product of $\mathbf{p}$ and $\mathbf{Q}[*][k]$. In our construction, each column of a matrix contains $(d - s + 1)$ query values at most and $(s - 1)$ random numbers at least. Although the sum of the random numbers at each row is related to the query value ($\delta_k = t_k \mathbf{q}[k]$), the random numbers in each column are independent from both the index and query vectors. Therefore, the adversary cannot infer any useful information from $\mathbf{r}$ directly. Now, let us consider the adversary constructing linear equations from an arbitrary combination of $\mathbf{r}[1], \ldots, \mathbf{r}[s]$. If the adversary

adds up $\mathbf{r}[i], \ldots, \mathbf{r}[j]$ where $i, j \in [s]$ and $|i - j| < s - 1$, the number of unknown variables is larger than the number of linear equations. If the adversary adds up $\mathbf{r}[1], \ldots, \mathbf{r}[s]$, it obtains α, whose security we have already proved. Therefore, the KNN scheme with random extending query matrix can resist linear analyses. □

Theorem 3. *Our advanced PIPE scheme is semantically secure in the known background model if* F *is a secure PRF and SKE is CPA-secure.*

Proof (sketch). In the known background model, the adversary (i.e., the cloud server) may obtain statistic information (e.g., keyword frequency and file distribution) besides what can be acquired in the known ciphertext model. With such background information, the adversary can derive a certain number of query keywords and their corresponding ciphertexts. For history $\mathcal{H}_S = (\mathcal{D}, \mathcal{W}, \mathbb{Q})$, the trace in PIPE_S, denoted by $Tr_S(\mathcal{H})$, includes a set of (query keyword, trapdoor) pairs and the following information:

- Size pattern: $(n, |D_1|, \ldots, |D_j|)$, where n is the number of files in $\mathcal{D}$ and $|D_j|$ is the bit length of file D_j.
- Access pattern: The search results $(\mathcal{R}_1, \ldots, \mathcal{R}_k)$, where $\mathcal{R}_j = \{(i, \widehat{\mathbf{R}}_{i,j}, \mathbf{R}_{i,j}) | Q_j \bowtie D_i, i \in [n]\}$, for $j \in [k]$. The intermediate matrix $\widehat{\mathbf{R}}_{i,j}$ equals $\mathbf{p}^{\star}_{i,1}\mathbf{Q}_{j,1}$ thereby contains s random numbers and the result matrix $\mathbf{R}_{i,j}$ contains only one random integer that equals $\mathbf{p}_{i,1} \cdot \mathbf{q}_{j,1} + X$ where $X \in \mathbb{Z}$ is a random integer that has no linear relationship with the result of $\mathbf{p}_{i,1} \cdot \mathbf{q}_{j,1}$.

In PIPE_S, the index is constructed in the same way as that in PIPE_0, and the construction of query matrices is based on that of PIPE_0. Based on Theorem 2, we claim that given two *Views* with the same trace $Tr_S(\mathcal{H})$, the probability for any PPT adversary to distinguish a view simulated by a simulator from a view generated by a real experiment is negligible. Therefore, PIPE_S is semantically secure in the known background model. □

6 Evaluation

This section will evaluate the performance of our PIPE scheme in terms of execution time and result accuracy. To show the effectiveness of our PIPE scheme, we compare it with the multi-keyword fuzzy search proposed in [21] (denoted by MFS), which also utilizes KNN to preserve data privacy. As main techniques of their work, a Bloom filter of 8,000 entries and a 2-stable $(\sqrt{3}, 2, p_1, p_2)$-LSH family are employed to support 1 edit distance difference between indexes and queries.

6.1 Parameter Setting

Experiments are conducted on a local machine running the Microsoft Windows 7 Ultimate operating system with an Intel Core i5 CPU running at 2.6 GHz and 16 GB memory. To validate the feasibility of our scheme in practice, we conduct a performance evaluation on the Enron Email Data Set;[1] this contains 30,109 emails sent by about 150 different users.

1. https://www.cs.cmu.edu/~enron/

The number of files n is set to $[5,000 \sim 30,000]$. For each file D_i, we extract $m_i = [5 \sim 25]$ keywords to build its index and the maximal length of keyword L is set to 30. The universal keyword set contains $m = [59,706 \sim 449,703]$ keywords. Each user will query with $K = [5 \sim 25]$ keywords, where each query contains $F = [2 \sim 5]$ symbol "$*$"s. To generate a fuzzy keyword for a query, we randomly choose one character from a keyword and replace it with symbol "$*$". For security considerations, the size of vectors is set as $d = 256$ for both PIPE_0 and PIPE_S. In PIPE_S, we set the number of columns of query matrices as $s = [2 \sim d]$.

6.2 Experiment Results

1) Efficiency. Both algorithms $GenKey_0$ and $GenKey_S$ generate two $d \times d$ invertible matrices as keys, the complexity of which is $O(d^2)$. Key generation is a one-time cost which will be omitted in comparisons. Both algorithms $BuildIndex_0$ and $BuildIndex_S$ generate a pair of $m_i \times d$ matrices $\mathbf{A}'_i = (\mathbf{A}'_{i_a}, \mathbf{A}'_{i_b})$ for file D_i containing m_i keywords, the complexity of which is $O(m_i d^2)$. The $Trapdoor_0$ algorithm outputs a pair of $d \times n_j$ matrices $\mathbf{B}'_j = (\mathbf{B}'_{j_a}, \mathbf{B}'_{j_b})$ for query Q_j containing n_j keywords, the complexity of which is $O(n_j d^2)$. The $Trapdoor_S$ algorithm outputs a pair of $d \times (sn_j)$ matrices $\widehat{\mathbf{B}}'_j = (\widehat{\mathbf{B}}'_{j_a}, \widehat{\mathbf{B}}'_{j_b})$ for query Q_j containing n_j keywords, the complexity of which is $O(sn_j d^2)$. The $Search_0$ algorithm generates an $m_i \times n_j$ matrix $R_{i,j}$ with a complexity $O(m_i n_j d)$, and the $Search_S$ algorithm generates an $m_i \times (sn_j)$ matrix $\widehat{R}_{i,j}$ with a complexity $O(sm_i n_j d)$.

Fig. 5 shows the comparison results between our PIPE scheme and MFS. For generating encrypted indexes and trapdoors, MFS requires the computation of multiple LSHs to map elements to a Bloom filter of 8,000 entries; our PIPE scheme requires only multiplication, and the size of vectors is much smaller. This makes it much more efficient in general. For example, in terms of building $n = 25,000$ encrypted indexes, MFS costs about 2,731 s, but our scheme incurs only about 43 s; in terms of generating a trapdoor for $K = 10$ keywords, MFS costs about 198 ms, but PIPE_0 and PIPE_S (when $s = 10$) incur only about 1 and 12 ms, respectively.

In terms of search speed, PIPE_0 is faster than MFS when the number of query keywords K is small. For example, when $K = 5$, MFS costs about 745 ms to search $n = 10,000$, while PIPE_0 costs only about 397 ms. However, the search complexity of PIPE_0 linearly increases as K increases. For example, when $n = 10,000$ and K ranges from 5 to 25, the search cost of PIPE_0 increases from 397 to 1,964 ms. In PIPE_S, the search complexity is impacted by the values of K and s. For example, when $s = 10$, the execution time of searching over $n = 10,000$ files grows from 3,800 ms to 18 s as K increases from 5 to 25; when $K = 5$, the execution time of searching over $n = 10,000$ files grows from 775 ms to 96 s as s increases from 10 to 250.

2) Accuracy. The accuracy of our scheme is measured by the definitions of the widely used performance metrics: *precision* and *recall*. Let t_p, f_p, and f_n denote true positive, false positive, and false negative, respectively. Precision can be calculated as $t_p/(t_p + f_p)$ and recall can be calculated as $t_p/(t_p + f_n)$. The loss of accuracy is caused by precision loss in division operations. To determine if a value is an integer or not, we round up after the xth decimal point, where $x \in$

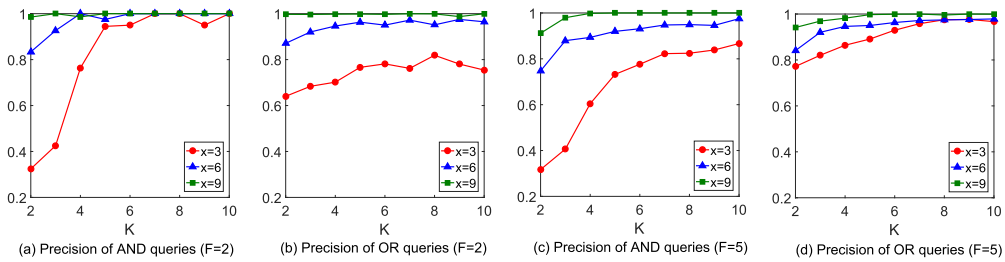


Fig. 6. The precision of the basic PIPE construction while rounding up after the xth decimal point (n is fixed with 10,000 and K ranges from 2 to 10). (a) AND queries of K keywords containing $F = 2$ wildcards. (b) OR queries of K keywords containing $F = 2$ wildcards. (c) AND queries of K keywords containing $F = 5$ wildcards. (d) OR queries of K keywords containing $F = 5$ wildcards.

$[3 \sim 10]$. This is because a large portion of fractional numbers, e.g., 3.000459, will be determined as integers if x is small, and this increases the rate of false positives. When x is large, many values, e.g., 3.000000007, will be determined as fractional numbers, and this increases the rate of false negatives. From Figs. 6 and 7, we know that $x = 6$ enables our PIPE scheme to have the highest accuracy, i.e., either the precision or the recall is almost 100 percent.

With the fixed $x = 6$, we evaluate the accuracy of our PIPE scheme under different settings of F, K, and s. From Fig. 8, we know that F and K have a significant impact on the accuracy of our PIPE scheme, but s has only a minor impact. In particular, the larger F, the lower accuracy of AND queries. For example, when $K = 5$, both precision and recall of PIPE_0 decrease from 100 to 81 percent and from 100 to 95 percent, respectively, as F increases from 2 to 5. Since a larger F implies more prime multiplications in vectors, a greater loss of precision will occur. Furthermore, an increase of K in AND queries causes an increase of precision or a decrease of recall. For example, when $F = 2$, precision of PIPE_0 increases from 83 to 100 percent and recall decreases from 100 to 90 percent as K increases from 2 to 10. This is because a larger K implies a larger number of columns in the result matrix, lowering the probability of turning non-integers of each column to integers (a lower f_p), but increasing the probability of turning integers in one column to non-integers (a higher f_n). Meanwhile, we compare the accuracy of OR queries between our scheme and MFS since MFS can support OR queries by changing a threshold value. From comparison results, we know that our PIPE scheme is much more accurate than MFS under different parameter settings. An increase of K in OR queries causes an increase of precision. This is because a larger K implies a larger number of matched files (a higher t_p). However, recall of OR queries is always very high (almost 100 percent) no matter how parameters change.

7 Related Work

The first SE scheme where both queries and data were encrypted under a symmetric key was proposed by Song *et al.* [1]. The main drawback of their scheme was that the search cost grew linearly with the database. To improve the query efficiency, Goh [2] developed a secure searchable index scheme based on Bloom filters. As a seminal work in SE, Curtmola *et al.* [3] provided a rigorous security definition and constructed schemes based on an inverted index. Subsequently, researchers enhanced the variety of SE schemes from query refinement [4], efficient update [5] and verifiability [6]. However, the above SE schemes only support single-keyword search.

7.1 Multi-Keyword Searchable Encryption

OXT [7] is the first SE scheme that supports sub-linear time for conjunctive keyword searches. Later, Lai *et al.* [8] proposed an HXT protocol by adopting lightweight symmetric-key hidden vector encryption and Bloom filters to eliminate the leakage of Keyword Pair Result Pattern (KPRP) in OXT. The main limitation of their scheme is multiple rounds of interaction between the cloud user and the cloud server during the search phase. Kamara *et al.* [9] made use of the inclusion-exclusion principle and proposed efficient disjunctive and boolean search schemes with worst-case sub-linear search complexity. Xia *et al.* [10] proposed a dynamic multi-keyword search scheme, where a user could efficiently update the encrypted outsourced data. Wan *et al.* [11] constructed a verifiable multi-keyword search scheme in which a user could detect any cheating behavior from malicious servers. Ranked search schemes allow users to retrieve the best-matched files to reduce communication costs. Cao *et al.* [12] proposed a multi-keyword ranked search scheme by leveraging the secure KNN technique [31] and TF-IDF model to rank the relevance between files and the queries. Chen

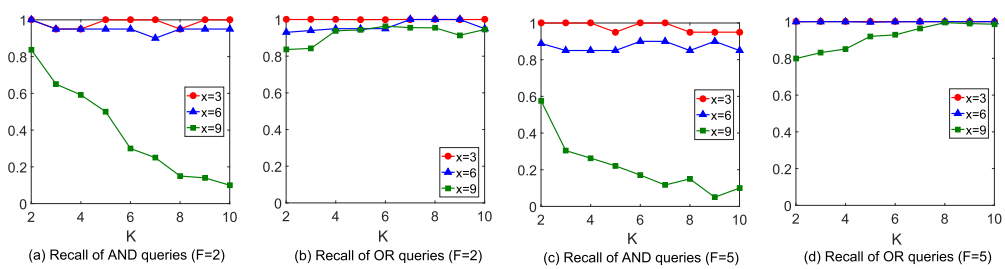
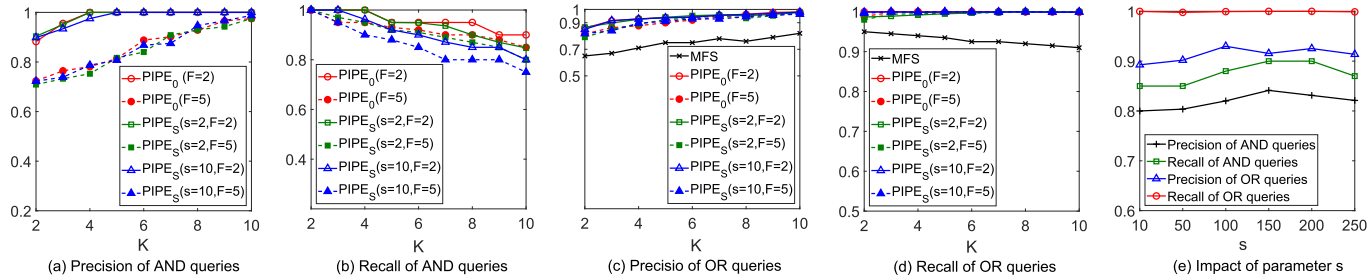


Fig. 7. The recall of the basic PIPE construction while rounding up after the xth decimal point (n is fixed with 10,000 and K ranges from 2 to 10). (a) AND queries of K keywords containing $F = 2$ wildcards. (b) OR queries of K keywords containing $F = 2$ wildcards. (c) AND queries of K keywords containing $F = 5$ wildcards. (d) OR queries of K keywords containing $F = 5$ wildcards.



(a) Precision of AND queries (b) Recall of AND queries (c) Precisio of OR queries (d) Recall of OR queries (e) Impact of parameter s

Fig. 8. The accuracy of our PIPE scheme. (a) Precision for AND queries of K keywords. (b) Recall for AND queries of K keywords. (c) Precision for OR queries of K keywords. (d) Recall for OR queries of K keywords. (e) Accuracy impacted by parameter s with $K = 5$ and $F = 5$.

TABLE 2
Comparison of Fuzzy Search Schemes

	Multi-keyword Search	Flexibility	Search Time	Communication Cost	Storage Cost
Li *et al.* [15]	×	×	$O(\sum_{i=1}^{m} l_{w_i}^e)$	$O(l_{w_q}^k)$	$O(\sum_{i=1}^{m} l_{w_i}^e)$
F2SE [19]	×	×	$O(m \times d)$	$O(d)$	$O(m \times d)$
MFS [21]	✓	✓	$O(n \times l_b)$	$O(l_b)$	$O(n \times l_b)$
Fu *et al.* [22]	✓	✓	$O(n \times l_b)$	$O(l_b)$	$O(n \times l_b)$
EliMFS-E [23]	✓	×	$O(\lvert\mathcal{D}(w_1)\rvert \times l_b)$	$O(l_b)$	$O(\sum_{i=1}^{m} \lvert\mathcal{D}(w_i)\rvert \times l_b)$
PIPE_0	✓	✓	$O(n \times m_i \times K \times d)$	$O(K \times d)$	$O(n \times m_i \times d)$
PIPE_S	✓	✓	$O(s \times n \times m_i \times K \times d)$	$O(s \times K \times d)$	$O(n \times m_i \times d)$

Notations: $n = |\mathcal{D}|$ is the number of files, $m = |\mathcal{W}|$ is the total number of keywords in $\mathcal{W}$, m_i is the number of keywords in file $\mathcal{D}_i$, and $|\mathcal{D}(w_i)|$ is the number of files matching keyword w_i. l_{w_i} is the length of keyword w_i and l_{w_q} is the length of query keyword w_q. e is the edit distance to form the wildcard-based fuzzy set of w_i, and k is the edit distance to form the trapdoor set of w_q, $(k < e)$. l_b is the length of Bloom filter and d is the dimension of index/query vector in PIPE and F2SE, $(d \ll l_b$ shown in our experiments). K is the number of keywords in a query, and $s \in [2, d]$ is a security parameter determining the amount of random noises to be added into a query matrix.

et al. [13] adopted the backtracking algorithm upon hierarchical clustering method to improve the rank privacy. However, the above multi-keyword search schemes only support exact keyword searches over encrypted data.

7.2 Single-Keyword Fuzzy Searchable Encryption

To improve search experiences, Refs. [14], [15] proposed wildcard-based fuzzy search schemes, which tolerated keyword misspellings in the query. The limitation of their scheme is the requirement of a predefined dictionary, the size of which increases exponentially with the edit distance. Liu *et al.* [16] improved the scheme by reducing the index size. Wang *et al.* [17] proposed a symbol-based trie-traverse mechanism to achieve similarity search with edit distance as the similarity metric. In [18], LSH functions were used to deal with similarity search. Wang *et al.* [19] adopted the keyword fingerprint extraction algorithm to transform a string into a fingerprint vector which would be utilized to evaluate the similarity between keywords.

7.3 Multi-Keyword Fuzzy Searchable Encryption

Chuah *et al.* [20] proposed a bedtree-based fuzzy search scheme to enable efficient updates but the index size is increasing with the edit distance. Refs. [21], [22] proposed a multi-keyword fuzzy search scheme which supported the constant size indexes. However, the combined effect of false positives (introduced by Bloom filters) and false negatives (introduced by LSH) seriously impacted the accuracy [23]. To enrich the search patterns, Fu *et al.* [24] designed a content-based symmetric SE scheme to enable efficient semantic searches; Wang *et al.* [25] proposed a scheme for a generalized pattern-matching string-search. To improve the security, Ding *et al.* [26] proposed a random traversal algorithm, which produced different visiting paths on the index for the identical queries with different keys. Boldyreva *et al.* [27] improved the security of existing fuzzy search schemes based on closeness graphs. To improve search efficiency, Moataz *et al.* [28] employed letter orthogonalization to allow testing of string membership by computing inner products; Hahn *et al.* [29] transformed the problem of secure substring search into range queries for fast execution time. Our previous work [30] constructed a single-keyword fuzzy search scheme based on the indecomposable property of prime numbers, and extended it to conjunctive keyword setting by utilizing collision-free hash functions. However, [30] requires an order among the keywords in index construction and trapdoor generation to avoid false negative, and thus it lacks flexibility and practicability. The comparison between our work and the state-of-the-art fuzzy search schemes is shown in Table 2.

8 Conclusion

In this paper, we propose a PIPE scheme to achieve secure and effective search services in cloud computing. The proposed scheme supports wildcard-based multi-keyword fuzzy searches over the encrypted data by exploiting the indecomposable property of primes. Experiment results demonstrate that our scheme is efficient and accurate. However, our adversary model assumes the mutual trust between the data owner and the data user. As part of our future work, we will try to design a secure PIPE scheme without such an assumption.

ACKNOWLEDGMENT

This work was supported in part by NSFC Grants 61632009, 61872133, 61872130, 61572181 and 61802076; US National Science Foundation Grants CNS 1824440, CNS 1828363, CNS 1757533, CNS 1629746, CNS 1651947, and CNS 1564128; the CERNET Innovation Project (NGII20190409); the Guangdong Provincial Natural Science Foundation (No. 2017A030308006), and the Hunan Provincial Natural Science Foundation of China (Grant No. 2020JJ3015).

REFERENCES

[1] D. X. Song, D. Wagner, and A. Perrig, "Practical techniques for searches on encrypted data," in *Proc. IEEE Symp. Secur. Privacy*, 2000, pp. 44–55.
[2] E.-J. Goh, "Secure indexes," *IACR Cryptol. ePrint Archive*, 2003. [Online]. Available: http://eprint.iacr.org/2003/216
[3] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky, "Searchable symmetric encryption: Improved definitions and efficient constructions," in *Proc. 13th ACM Conf. Comput. Commun. Secur.*, 2006, pp. 79–88.
[4] Q. Liu, Y. Tian, J. Wu, T. Peng, and G. Wang, "Enabling verifiable and dynamic ranked search over outsourced data," *IEEE Trans. Services Comput.*, to be published, doi: 10.1109/TSC.2019.2922177.
[5] S. Kamara, C. Papamanthou, and T. Roeder, "Dynamic searchable symmetric encryption," in *Proc. ACM Conf. Comput. Commun. Secur.*, 2012, pp. 965–976.
[6] K. Kurosawa and Y. Ohtaki, "UC-secure searchable symmetric encryption," in *Proc. Int. Conf. Financial Cryptography Data Secur.*, 2012, pp. 285–298.
[7] 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 *Proc. Annu. Cryptol. Conf.*, 2013, pp. 353–373.
[8] S. Lai *et al.* "Result pattern hiding searchable encryption for conjunctive queries," in *Proc. ACM Conf. Comput. Commun. Secur.*, 2018, pp. 745–762.
[9] S. Kamara and T. Moataz, "Boolean searchable symmetric encryption with worst-case sub-linear complexity," in *Proc. Annu. Int. Conf. Theory Appl. Cryptogr. Techn.*, 2017, pp. 94–124.
[10] Z. Xia, X. Wang, X. Sun, and Q. Wang, "A secure and dynamic multi-keyword ranked search scheme over encrypted cloud data," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 2, pp. 340–352, Feb. 2016.
[11] Z. Wan and R. H. Deng, "VPSearch: Achieving verifiability for privacy-preserving multi-keyword search over encrypted cloud data," *IEEE Trans. Dependable Secure Comput.*, vol. 15, no. 6, pp. 1083–1095, Nov./Dec. 2018.
[12] N. Cao, C. Wang, M. Li, K. Ren, and W. Lou, "Privacy-preserving multi-keyword ranked search over encrypted cloud data," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 1, pp. 222–233, Jan. 2014.
[13] C. Chen *et al.* "An efficient privacy-preserving ranked keyword search method," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 4, pp. 951–963, Apr. 2016.
[14] J. Li, Q. Wang, C. Wang, N. Cao, K. Ren, and W. Lou, "Enabling efficient fuzzy keyword search over encrypted data in cloud computing," *IACR Cryptol. ePrint Archive*, 2009. [Online]. Available: http://eprint.iacr.org/2009/593
[15] J. Li, Q. Wang, C. Wang, N. Cao, K. Ren, and W. Lou, "Fuzzy keyword search over encrypted data in cloud computing," in *Proc. IEEE INFOCOM*, 2010, pp. 1–5.
[16] C. Liu, L. Zhu, L. Li, and Y. Tan, "Fuzzy keyword search on encrypted cloud storage data with small index," in *Proc. IEEE Int. Conf. Cloud Comput. Intell. Syst.*, 2011, pp. 269–273.
[17] C. Wang, K. Ren, S. Yu, and K. M. R. Urs, "Achieving usable and privacy-assured similarity search over outsourced cloud data," in *Proc. IEEE INFOCOM*, 2012, pp. 451–459.
[18] M. Kuzu, M. S. Islam, and M. Kantarcioglu, "Efficient similarity search over encrypted data," in *Proc. IEEE 28th Int. Conf. Data Eng.*, 2012, pp. 1156–1167.
[19] D. Wang, S. Fu, and M. Xu, "A privacy-preserving fuzzy keyword search scheme over encrypted cloud data," in *Proc. IEEE 5th Int. Conf. Cloud Comput. Technol. Sci.*, 2013, pp. 663–670.
[20] M. Chuah and W. Hu, "Privacy-aware BedTree based solution for fuzzy multi-keyword search over encrypted data," in *Proc. 31st Int. Conf. Distrib. Comput. Syst. Workshops*, 2011, pp. 273–281.
[21] B. Wang, S. Yu, W. Lou, and Y. T. Hou, "Privacy-preserving multi-keyword fuzzy search over encrypted data in the cloud," in *Proc. IEEE INFOCOM*, 2014, pp. 2112–2120.
[22] 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 Trans. Inf. Forensics Security*, vol. 11, no. 12, pp. 2706–2716, Dec. 2016.
[23] J. Chen *et al.* "EliMFS: Achieving efficient, leakage-resilient, and multi-keyword fuzzy search on encrypted cloud data," *IEEE Trans. Services Comput.*, to be published, doi: 10.1109/TSC.2017.2765323.
[24] Z. Fu, L. Xia, X. Sun, A. X. Liu, and G. Xie, "Semantic-aware searching over encrypted data for cloud computing," *IEEE Trans. Inf. Forensics Security*, vol. 13, no. 9, pp. 2359–2371, Sep. 2018.
[25] D. Wang, X. Jia, C. Wang, K. Yang, S. Fu, and M. Xu, "Generalized pattern matching string search on encrypted data in cloud systems," in *Proc. IEEE INFOCOM*, 2015, pp. 2101–2109.
[26] X. Ding, P. Liu, and H. Jin, "Privacy-preserving multi-keyword top-k similarity search over encrypted data," *IEEE Trans. Dependable Secure Comput.*, vol. 16, no. 2, pp. 344–357, Mar./Apr. 2019.
[27] A. Boldyreva and N. Chenette, "Efficient fuzzy search on encrypted data," in *Proc. Int. Workshop Fast Softw. Encryption*, 2014, pp. 613–633.
[28] T. Moataz, I. Ray, I. Ray, A. Shikfa, F. Cuppens, and N. Cuppens, "Substring search over encrypted data," *J. Comput. Secur.*, vol. 26, no. 1, pp. 1–30, Jan. 2018.
[29] F. Hahn, N. Loza, and F. Kerschbaum, "Practical and secure substring search," in *Proc. Int. Conf. Manage. Data*, 2018, pp. 163–176.
[30] Q. Liu, S. Pei, K. Xie, J. Wu, T. Peng, and G. Wang, "Achieving secure and effective search services in cloud computing," in *Proc. 17th IEEE Int. Conf. Trust Secur. Privacy Comput. Commun./ 12th IEEE Int. Conf. Big Data Sci. Eng.*, 2018, pp. 1386–1391.
[31] W. K. Wong, D. W.-L. Cheung, B. Kao, and N. Mamoulis, "Secure kNN computation on encrypted databases," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2009, pp. 139–152.
[32] B. Yao, F. Li, and X. Xiao, "Secure nearest neighbor revisited," in *Proc. IEEE 29th Int. Conf. Data Eng.*, 2013, pp. 733–744.
[33] D. Guo, J. Wu, H. Chen, Y. Yuan, and X. Luo, "The dynamic bloom filters," *IEEE Trans. Knowl. Data Eng.*, vol. 22, no. 1, pp. 120–133, Jan. 2010.
[34] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni, "Locality-sensitive hashing scheme based on p-stable distributions," in *Proc. 20th Annu. Symp. Comput. Geometry*, 2004, pp. 253–262.
[35] Q. Wang, M. He, M. Du, S. S. M. Chow, R. W. F. Lai, and Q. Zhou, "Searchable encryption over feature-rich data," *IEEE Trans. Dependable Secure Comput.*, vol. 15, no. 3, pp. 496–510, May/Jun. 2018.

Qin Liu (Member, IEEE) received the BSc degree in computer science from Hunan Normal University, Changsha, China, in 2004, and the MSc and PhD degrees in computer science from Central South University, Changsha, China, in 2007 and 2012, respectively. She has been a visiting student with Temple University, Philadelphia, Pennsylvania. Her research interests include security and privacy issues in cloud computing. Currently, she is an associate professor with the College of Computer Science and Electronic Engineering, Hunan University, China.

Yu Peng (Student Member, IEEE) received the BSc degree in software engineering from the China West Normal University, Nanchong, China, in 2018. Currently, he is working toward the PhD degree with the College of Computer Science and Electronic Engineering, Hunan University, Changsha, China. His research interests include security and privacy issues in cloud computing. He is a student member of the China Computer Federation (CCF).

Shuyu Pei (Student Member, IEEE) received the BSc degree in information security and the MSc degree in computer science from Hunan University, Changsha, China, in 2015 and 2018. She is currently working toward the PhD degree with the College of Computer Science and Electronic Engineering, Hunan University, Changsha, China. Her research interests include security and privacy issues in cloud computing.

Jie Wu (Fellow, IEEE) is the chair and a Laura H. Carnell professor with the Department of Computer and Information Sciences, Temple University, Philadelphia, Pennsylvania. Prior to joining Temple University, he was a program director with the National Science Foundation and a distinguished professor with Florida Atlantic University. His current research interests include mobile computing and wireless networks, routing protocols, cloud and green computing, network trust and security, and social network applications. He has regularly published in scholarly journals, conference proceedings, and books. He serves on several editorial boards, including the *IEEE Transactions on Service Computing*, and the *Journal of Parallel and Distributed Computing*. He was general co-chair/chair for IEEE MASS 2006, IEEE IPDPS 2008, IEEE ICDCS 2013, and ACM MobiHoc 2014, as well as program co-chair for IEEE INFOCOM 2011 and CCF CNCC 2013. He was an IEEE Computer Society distinguished visitor, ACM distinguished speaker, and chair for the IEEE Technical Committee on Distributed Processing (TCDP). He is a CCF distinguished speaker. He is the recipient of the 2011 China Computer Federation (CCF) Overseas Outstanding Achievement Award.

Tao Peng (Member, IEEE) received the BSc degree in computer science from Xiangtan University, Xiangtan, China, in 2004, the MSc degree in circuits and systems from Hunan Normal University, Changsha, China, in 2007, and the PhD degree in computer science from Central South University, Changsha, China, in 2017. Currently, she is an associate professor with the School of Computer Science and Cyber Engineering, Guangzhou University, China. Her research interests include network and information security issues.

Guojun Wang (Member, IEEE) received the BSc degree in geophysics, MSc degree in computer science, and PhD degree in computer science, both from Central South University, Changsha, China, in 1992, 1996, and 2002, respectively. He is a Pearl River Scholarship distinguished professor of higher education in Guangdong Province, a doctoral supervisor and vice dean with the School of Computer Science and Cyber Engineering, Guangzhou University, China, and the director with the Institute of Computer Networks, Guangzhou University. He has been listed in Chinese Most Cited Researchers (Computer Science) by Elsevier in the past six consecutive years (2014-2019). His research interests include artificial intelligence, big data, cloud computing, Internet of Things (IoT), blockchain, trustworthy/dependable computing, network security, privacy preserving, recommendation systems, and smart cities. He is a distinguished member of CCF, and a member of the ACM and IEICE.