Secure Multi-keyword Fuzzy Searches With Enhanced Service Quality in Cloud Computing

Qin Liu, *Member, IEEE*, Yu Peng, *Graduate Student Member, IEEE*, Jie Wu, *Fellow, IEEE*, Tian Wang, *Member, IEEE*, and Guojun Wang, *Member, IEEE*

***Abstract*—With the ever-increasing amount of data resided in a cloud, how to provide users with secure and practical query services has become the key to improve the quality of cloud services. Fuzzy searchable encryption (FSE) is identified as one of the most promising approaches for enabling secure query services, since it allows searching encrypted data by using keywords with spelling errors. However, existing FSE schemes are far from the practical use for the following reasons: (1) *Inflexibility.* It is hard for them to simultaneously support AND and OR semantics in a multi-keyword query. (2) *Inefficiency.* They require sequentially scanning a whole dataset to find matched files, and thus are difficult to apply to a large-scale dataset. (3) *Limited robustness.* It is difficult for them to resist the linear analysis attack in the known-background model. To fix the above problems, this article proposes matrix-based multi-keyword fuzzy search (M2FS) schemes, which support approximate keyword matching by exploiting the indecomposable property of primes. Specifically, we first present a basic scheme, called M2FS-B, where multiple keywords in a query or a file are constructed as prime-related matrices such that the result of matrix multiplication can be employed to determine the level of matching for different query semantics. Then, we construct an advanced scheme, named M2FS-E, which builds a searchable index as a keyword balanced binary (KBB) tree for dynamic and parallel searches, while adding random noises into a query matrix for enhanced robustness. Extensive analyses and experiments demonstrate the validity of our M2FS schemes.**

***Index Terms*—Cloud computing, secure query services, fuzzy searchable encryption, multi-semantic query, parallel search.**

Manuscript received April 23, 2020; revised October 9, 2020; accepted December 13, 2020. Date of publication December 17, 2020; date of current version June 10, 2021. This work was supported in part by NSFC grants 61632009, 61872133, 61872130, 61572181, 61872131, and 61802076; NSF 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). The associate editor coordinating the review of this article and approving it for publication was M. Conti. *(Corresponding author: Qin Liu.)*

Qin Liu and Yu Peng are with the College of Computer Science and Electronic Engineering, Hunan University, Changsha 410082, China (e-mail: gracelq628@hnu.edu.cn; pengyu411@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).

Tian Wang is with Artificial Intelligence and Future Networks, Beijing Normal University and UIC, Zhuhai 519087, China, and also with the College of Computer Science and Technology, Huaqiao University, Xiamen 361021, China (e-mail: cs_tianwang@163.com).

Guojun Wang is with the School of Computer Science and Cyber Engineering, Guangzhou University, Guangzhou 510006, China (e-mail: csgjwang@gzhu.edu.cn).

Digital Object Identifier 10.1109/TNSM.2020.3045467

I. Introduction

Cloud computing, providing a wide variety of services in a pay-as-you-go fashion, is an extremely successful paradigm of service-oriented computing. With the increasing popularity of cloud-based services, consumers are highly motivated to outsource their data and computing services to cloud platforms (e.g., Amazon EC2 and S3, Microsoft Azure, and Google App Engine) for lower costs, higher reliability, and better performance. However, studies and past experience show that cloud platforms may be unreliable, and vulnerable to various threats that cause data leakage intentionally or unwittingly. For security consideration, existing research [1] suggests encrypting data before outsourcing. The new challenge of service quality is emerging since it is hard for traditional encryption methods (e.g., AES and RSA) to support common cloud services like keyword-based searches.

In order to ensure data privacy without invalidating data usability, outsourcing *searchable encrypted data* to cloud platforms has become a prevalent trend in recent years. As a typical application, a user who has a cloud service account uploads encrypted files to a cloud server, and later she generates an encrypted query (referred to as *trapdoor*) for certain keywords to retrieve files of interest. With this trapdoor, the cloud server can find all matching files without decryption. The cryptographic tool enabling keyword-based searches over encrypted outsourced data is referred to as searchable encryption (SE) [2]–[10]. SE has been widely researched since it was proposed by Song *et al.* [2] in 2000. However, most of them handle exact keyword matching, where the misspelling of a query keyword will cause an error result to be returned. While querying the outsourced data, it is a common case that a user forgets the correct spelling of a keyword, but still wants to retrieve files of interest as accurately as possible. Therefore, how to design a SE scheme supporting approximate keyword matching has become an urgent issue, leading to the concept of fuzzy searchable encryption (FSE) [11]–[31].

The first effort of building a FSE scheme comes from Li *et al.* [11], where edit distance [32] and wildcards are exploited to construct a predefined fuzzy set covering all possible keyword misspellings. However, their work permits only a single keyword in a query and enables the size of fuzzy set to increase exponentially with the edit distance. Since then, a few works on multi-keyword FSE [18]–[31] have been proposed with different trade-offs between security and performance. Even so, the transformation of multi-keyword FSE into practical use is far from satisfactory, since they cannot simultaneously satisfy the following requirements.

The first one is *flexibility*. Previous schemes are either focused on conjunctive keyword searches (i.e., AND query semantic), or dedicated to disjunctive keyword searches (i.e., OR query semantic). And yet, very little research has addressed simultaneously supporting AND and OR query semantics. For example, if an FSE scheme supports multiple semantics, a user can retrieve files matching query $\vartheta = \texttt{cloud} \wedge (\texttt{security} \vee \texttt{privacy})$ with a single query. Otherwise, she needs to send two queries $\vartheta_1 = \texttt{cloud}$ and $\vartheta_2 = \texttt{security} \vee \texttt{privacy}$ to the cloud server, which executes each query separately and then performs the intersecion between the resultant file sets.

The next is *efficiency*. To speed up the search process, there are two kinds of approaches to design a searchable index, the forward index that establishes keyword lists per file, and the inverted index that builds file lists per keyword. The inverted index was traditionally used for single-keyword searches. Most of existing multi-keyword FSE schemes build forward indexes to perform sequential searches on a collection of files, resulting in the search time growing linearly with the total number of files. With data volume exploding, it is imperative to design an efficient index structure to facilitate parallel searches on a large-scale dataset.

Finally comes *robustness*. As the work in [19], [20], [26], the secure *k*-nearest neighbor scheme [33] (SKNN for short) that enables *index indistinguishability* and *trapdoor unlinkability*[1] has become a commonly used encryption tool in FSE. However, SKNN is vulnerable to the linear analysis attack launched by an attacker who knows additional background information about the files and queries. That is, given enough query/trapdoor pairs, the attacker can recover file keywords by solving linear equations. Therefore, more sophisticated and robust designs are required to address the potential privacy violation problem in SKNN.

In this article, we propose matrix-based multi-keyword fuzzy search (M2FS) schemes, which exploit the indecomposable property of prime numbers to provide enhanced service quality in cloud computing. Like the work in [11], our M2FS schemes also apply the wildcard technique and edit distance to quantify keywords' similarity. However, our schemes do not require the construction of a predefined fuzzy set and thus are more scalable and practical. Our main idea is to encode a file keyword (resp. a query keyword) into an index vector filled with primes (resp. a query vector filled with reciprocals of primes), such that the result of vectors' inner product is an integer only when two keywords are similar. For flexibility, index vectors and query vectors are organized into prime-related matrices (referred to as query matrix and index matrix, respectively) so that the result of matrix multiplication can be used to determine whether a file matches a multi-semantic query or not. For efficiency, a keyword balanced binary (KBB) tree is built from a collection of index matrices to achieve dynamic and parallel searches. For robustness, a query matrix is extended by random noises before being encrypted with SKNN, such that the level of the match can be quantified based on the encrypted matrices, while breaking the linear relationship between queries and trapdoors. Specifically, we first focus on achieving flexibility, and present a basic M2FS scheme, named M2FS-B, which constructs forward indexes from a collection of index matrices. Then, we construct an advanced M2FS scheme, named M2FS-E, which applies the KBB tree and expansive query matrix to achieve flexibility, efficiency, and robustness simultaneously. Our main contributions are summarized as follows:

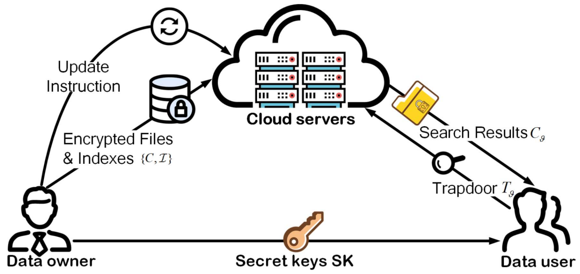


Fig. 1. System model. Secret keys are transmitted through secure channels protected by SSL/SSH.

- To the best of our knowledge, this is the first attempt to devise FSE schemes, which exploit the indecomposable property of prime numbers to achieve practical multi-keyword fuzzy searches.
- Compared with existing work, our M2FS schemes have the following merits: (1) *Greater flexibility.* They allow a user to choose different semantics in a query as required. (2) *Higher efficiency.* They are highly parallelizable and dynamic. (3) *Enhanced robustness.* They can resist linear analyses while achieving index indistinguishability and trapdoor unlinkability.
- We provide formal definitions and proofs on the correctness and security of our M2FS schemes, and conduct experiments on a real dataset to verify their feasibility and practicability.

Paper Organization: We formulate our research problem in Section II before sketching out this work in Section III. After constructing the basic and advanced M2FS schemes in Section IV and V, respectively, we evaluate their performance in Section VI. Finally, we introduce related work in Section VII, before concluding this article in Section VIII.

II. Problem Formulation

A. The System and Threat Models

As shown in Fig. 1, our system model consists of three types of entities: data owner, data user, and cloud server.

• *Data owner* possesses a large-scale collection of files F and decides to outsource them in the encrypted forms C for reduced cost and convenient access. After outsourcing all the above information, she can perform updates (add/delete) on ciphertexts on demand by sending an update instruction to the cloud server. To enable efficient searches, she builds a searchable index from file collection F and a universal keyword set

[1] Given a set of encrypted indexes/queries, it is hard to decide whether they are generated for the same keywords or not.

W before uploading file ciphertexts and an encrypted index, $\{C, \mathcal{I}\}$, to the cloud server. For access authorization, she is responsible for the secure distribution of key information to qualified data users.

• *Data user* provides the cloud server with a trapdoor, T_ϑ, to retrieve files matching query ϑ after obtaining a warrant from the data owner. Specifically, the query ϑ that supports AND and OR semantics may contain multiple fuzzy keywords. Upon receiving search results C_ϑ from the cloud server, he performs decryption locally to recover file contents.

• *Cloud server* centralizes abundant resources and provides data storage and query services to data owners and data users, respectively. Upon receiving the store request from the data owner, it stores the encrypted files and index $\{C, \mathcal{I}\}$ to appropriate locations. Given a trapdoor T_ϑ sent by the data user, it evaluates T_ϑ on the encrypted index $\mathcal{I}$ and returns all matched ciphertexts C_ϑ as the search result. Besides, it also follows the data owner's commands to perform updates on $\{C, \mathcal{I}\}$ appropriately.

In our threat model, the cloud server as the only attacker is considered to be *honest but curious* (HBC) [27], [34], [35]. A HBC cloud server will correctly execute instructions in a predefined protocol, but may try to learn additional information as much as possible from the messages it has seen. According to the information available to the cloud server, we mainly consider the following two models employed by lots of works on secure searches in clouds [19], [20]:

• *Known Ciphertext Model:* The cloud server only knows the encrypted files and index $\{C, \mathcal{I}\}$ uploaded by the data owner, the trapdoors $\mathbb{T} = \{T_\vartheta\}$ submitted by the authorized data user, and the returned search results $\mathbb{C} = \{C_\vartheta\}$. Therefore, the cloud server can conduct ciphertext-only attacks (COA) in this model.

• *Known Background Model:* In this stronger model, the cloud server knows additional background information besides what is available in the known ciphertext model. With this information, the cloud server can identify certain keyword/trapdoor pairs, thereby launching linear analysis attacks.

B. Notations and Cryptographic Preliminaries

Let notation $\lambda \in \mathbb{N}$ denote the security parameter throughout this article. The set of binary strings of length η is denoted by $\{0,1\}^\eta$ and the set of all finite binary strings by $\{0,1\}^*$. Notation $[\![\eta_1, \eta_2]\!]$ represents the set of integers in $\{\eta_1, \ldots, \eta_2\}$, which can be abbreviated as $[\![\eta_2]\!]$ when $\eta_1 = 1$. Given a string s, $\|s\|$ refers to the number of characters in s, and $s[i]$ refers to its i-th character. Given a dictionary of α characters $S = (s_1, \ldots, s_\alpha)$, its i-th character is denoted by $\mathcal{S}[i]$ or s_i, and the concatenation of the first l characters is denoted by $\langle s_1, \ldots, s_l \rangle$. If S is a set, its cardinality is denoted by $|S|$. The most relevant notations are provided as follows:

• $F = \{f_1, \ldots, f_n\}$*:* A set of n files, where each file f_i is associated with an identifier i for $i \in [\![n]\!]$. We assume that file identifier is independent of file contents, and thus is allowed to be exposed to the cloud server.

• $C = \{c_1, \ldots, c_n\}$*:* A set of n ciphertexts, where c_i is the ciphertext of file f_i for $i \in [\![n]\!]$.

• $W = \{w_1, \ldots, w_m\}$*:* A universal set of m keywords.

• $\alpha = \max(\|w_1\|, \ldots, \|w_m\|)$*:* Maximal length of universal keywords.

• $W_i = \{w_{i,1}, \ldots, w_{i,\beta_i}\}$*:* β_i keywords in file f_i.

• $\widetilde{W}_j = \{\widetilde{w}_{j,1}, \ldots, \widetilde{w}_{j,\gamma_j}\}$*:* γ_j keywords in query ϑ_j.

• $\mathcal{I}$*:* An encrypted searchable index.

• T_{ϑ_j}*:* A trapdoor of query ϑ_j.

• $\mathcal{A} = (\text{"}a\text{"}, \ldots, \text{"}z\text{"})$*:* A dictionary of 26 English characters, where its i-th element is denoted by $\mathcal{A}[i]$.

• $\mathcal{S} = (s_1, \ldots, s_\alpha)$*:* A dictionary of α dummy characters, where its i-th element is denoted by s_i or $\mathcal{S}[i]$.

• $\mathbb{P} = (p_1, \ldots, p_\alpha)$*:* A sequence of α primes, where its i-th element is denoted by p_i or $\mathbb{P}[i]$.

Secure k-Nearest Neighbor Scheme (SKNN) [33]: SKNN allows efficient computation of the k-nearest neighbors over encrypted data and is applied to encrypt our index/query matrices. Given a vector $\mathbf{v}$, its i-th element is denoted by $\mathbf{v}[i]$. Given a matrix $\mathbf{M}$, the element in its i-th row and j-th column is denoted by $\mathbf{M}[i][j]$, all elements in its i-th row are denoted by $\mathbf{M}[i][*]$, and all elements in its j-th column are denoted by $\mathbf{M}[*][j]$. Let notation $\mathbf{M}^{-1}$ denote the inverse matrix of $\mathbf{M}$, and let notations " $\cdot$ " and " $\star$ " denote vector inner product and matrix multiplication, respectively. SKNN tailored for our M2FS schemes mainly consists of the following algorithms:

• $sk \leftarrow GenKey(1^\lambda)$*:* It takes a security parameter $\lambda \in \mathbb{N}$ as input and generates secret keys $sk = (\mathbf{M}_1, \mathbf{M}_2, \mathbf{s})$, 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 d must be larger than λ for security. To maximize randomness, the number of 0s is approximately equal to the number of 1s in $\mathbf{s}$.

• $\mathbf{P}' \leftarrow EncI(\mathbf{P}, sk)$*:* It first splits an index matrix $\mathbf{P} \in \mathbb{R}^{\beta \times d}$ into two random matrices, $(\mathbf{P}_a, \mathbf{P}_b) \in \mathbb{R}^{\beta \times d}$, as follows: for $k \in [\![\beta]\!]$ and for $l \in [\![d]\!]$, if $\mathbf{s}[l] = 1$, it sets $\mathbf{P}_a[k][l]$ and $\mathbf{P}_b[k][l]$ to random values such that $\mathbf{P}_a[k][l] + \mathbf{P}_b[k][l] = \mathbf{P}[k][l]$; if $\mathbf{s}[l] = 0$, it sets $\mathbf{P}_a[k][l] = \mathbf{P}_b[k][l] = \mathbf{P}[k][l]$. It then encrypts $(\mathbf{P}_a, \mathbf{P}_b)$ and outputs $\mathbf{P}' = (\mathbf{P}'_a, \mathbf{P}'_b)$ where $\mathbf{P}'_a = \mathbf{P}_a \star \mathbf{M}_1$ and $\mathbf{P}'_b = \mathbf{P}_b \star \mathbf{M}_2$.

• $\mathbf{Q}' \leftarrow EncQ(\mathbf{Q}, sk)$*:* It first splits a query matrix $\mathbf{Q} \in \mathbb{R}^{d \times \gamma}$ into two random matrices, $(\mathbf{Q}_a, \mathbf{Q}_b)$, as follows: for $l \in [\![\gamma]\!]$ and for $k \in [\![d]\!]$, if $\mathbf{s}[k] = 1$, it sets $\mathbf{Q}_a[k][l] = \mathbf{Q}_b[k][l] = \mathbf{Q}[k][l]$; if $\mathbf{s}[k] = 0$, it sets $\mathbf{Q}_a[k][l]$ and $\mathbf{Q}_b[k][l]$ to random values such that $\mathbf{Q}_a[k][l] + \mathbf{Q}_b[k][l] = \mathbf{Q}[k][l]$. It then encrypts $(\mathbf{Q}_a, \mathbf{Q}_b)$ and outputs $\mathbf{Q}' = (\mathbf{Q}'_a, \mathbf{Q}'_b)$ where $\mathbf{Q}'_a = \mathbf{M}_1^{-1} \star \mathbf{Q}_a, \mathbf{Q}'_b = \mathbf{M}_2^{-1} \star \mathbf{Q}_b$.

• $\mathbf{R} \leftarrow Test(\mathbf{P}', \mathbf{Q}')$*:* It generates a test matrix $\mathbf{R} \in \mathbb{R}^{\beta \times \gamma}$ by calculating $\mathbf{R} = \mathbf{P}'_a \star \mathbf{Q}'_a + \mathbf{P}'_b \star \mathbf{Q}'_b$. Note that:

$$\begin{aligned}\mathbf{R} &= \mathbf{P}'_a \star \mathbf{Q}'_a + \mathbf{P}'_b \star \mathbf{Q}'_b \\ &= (\mathbf{P}_a \star \mathbf{M}_1) \star \left(\mathbf{M}_1^{-1} \star \mathbf{Q}_a\right) + (\mathbf{P}_b \star \mathbf{M}_2) \star \left(\mathbf{M}_2^{-1} \star \mathbf{Q}_b\right) \\ &= \mathbf{P}_a \star \mathbf{Q}_a + \mathbf{P}_b \star \mathbf{Q}_b \\ &= \mathbf{P} \star \mathbf{Q}, \qquad (1)\end{aligned}$$

where $\mathbf{R}[k][l] = \mathbf{P}[k][*] \star \mathbf{Q}[*][l]$ for $k \in [\![\beta]\!]$ and $l \in [\![\gamma]\!]$.

Therefore, SKNN enables the multiplication of the plaintext matrices to be calculated based on their encrypted forms. Besides, we encrypt files with symmetric key encryption (SKE) that is secure under chosen-plaintext attacks. We also utilize keyed pseudorandom functions (PRF), which are polynomial-time (PPT) computable functions indistinguishable from random functions when the key is kept secret.

III. Scheme Overview

A. Keywords, Distance, and Matching

Let notation "?" denote a wildcard. Our M2FS schemes classify keywords into *exact keywords* and *fuzzy keywords*. For an exact keyword, all its characters are chosen from the dictionary $\mathcal{A}$, but for a fuzzy keyword, it also contains wildcards, which denotes unsure characters in a keyword. For example, a data user can issue query containing fuzzy keyword "*s?curi??*" to retrieve appropriate files if he is unsure of the second and the last two characters of the keyword "*security*".

To distinguish query keywords from index keywords, we denote the set of keywords associated with file f_i by $W_i = \{w_{i,1}, \ldots, w_{i,\beta_i}\}$ and the set of keywords in query ϑ_j by $\widetilde{W_j} = \{\widetilde{w}_{j,1}, \ldots, \widetilde{w}_{j,\gamma_j}\}$, respectively. We assume that set W_i includes only exact keywords, and that set $\widetilde{W_j}$ contains both kinds of keywords. Given a file keyword $w_{i,k} \in W_i$ and a query keyword $\widetilde{w}_{j,l} \in \widetilde{W_j}$, their distance, denoted by $\Delta(w_{i,k}, \widetilde{w}_{j,l})$, is defined as follows:

Definition 1 (Distance): Let e_1 be the minimal number of insertions, deletions, and substitutions required to transform keyword $w_{i,k}$ into keyword $\widetilde{w}_{j,l}$ and let e_2 be the number of wildcards "?" in substitution operation. We have $\Delta(w_{i,k}, \widetilde{w}_{j,l}) = e_1 - e_2$.

In other word, for keywords $w_{i,k}$ and $\widetilde{w}_{j,l}$, $\Delta(w_{i,k}, \widetilde{w}_{j,l})$ is determined by their edit distance excluding the number of symbol "?"s in keyword $\widetilde{w}_{j,l}$. The similarity of two keywords is determined by their distance defined as follows:

Definition 2 (Similarity): Keywords $w_{i,k}$ and $\widetilde{w}_{j,l}$ are considered similar, denoted by $w_{i,k} \approx \widetilde{w}_{j,l}$, if $\Delta(w_{i,k}, \widetilde{w}_{j,l}) = 0$.

Therefore, "*cloud*" is similar to "*c??ud*" and is dissimilar to "*c?aud*", because $\Delta(\text{"}cloud\text{"}, \text{"}c??ud\text{"}) = 0$ and $\Delta(\text{"}cloud\text{"}, \text{"}c?aud\text{"}) = 1$. In our schemes, a query that contains multiple fuzzy keywords can support AND and OR semantics simultaneously. The matching of a query ϑ_j and a file f_i, denoted by $\vartheta_j \bowtie f_i$, is defined as follows:

Definition 3 (Matching): For an AND query, $\vartheta_j \bowtie f_i$ if, for each keyword $\widetilde{w}_{j,l} \in \widetilde{W_j}$ there exists a keyword $w_{i,k} \in W_i$ such that $\widetilde{w}_{j,l} \approx \underline{w_{i,k}}$. For an OR query, $\vartheta_j \bowtie f_i$ if there exists a keyword $\widetilde{w}_{j,l} \in \widetilde{W_j}$ such that $\widetilde{w}_{j,l} \approx w_{i,k}$ where $w_{i,k} \in W_i$.

That is, for an AND query, $\vartheta_j \bowtie f_i$ if each query keyword $\widetilde{w}_{j,l} \in \widetilde{W_j}$ is similar to a certain file keyword $w_{i,k} \in W_i$. For an OR query, $\vartheta_j \bowtie f_i$ if at least one query keyword $\widetilde{w}_{j,l} \in \widetilde{W_j}$ is similar to a certain file keyword $w_{i,k} \in W_i$. For example, given keyword sets $W_1 = \{\text{"}cloud\text{"}, \text{"}bus\text{"}\}$, $W_2 = \{\text{"}cloud\text{"}, \text{"}apple\text{"}\}$, and $\widetilde{W_1} = \{\text{"}c??ud\text{"}, \text{"}b?s\text{"}\}$, we have $\vartheta_1 \bowtie f_1$ and $\vartheta_1 \not\bowtie f_2$ for AND queries, but $\vartheta_1 \bowtie f_1$ and $\vartheta_1 \bowtie f_2$ for OR queries.

B. Keyword Balanced Binary Tree (KBB)

Inspired by the work in Xia *et al.* [4], an index tree $\mathcal{T}$ is constructed as a KBB tree, where each node *u* is defined as:

$$u = \langle nid, data, fid, lchild, rchild \rangle \quad (2)$$

where *nid* is the unique identifier of node *u* in $\mathcal{T}$, *data* is the data field of node *u*, *fid* stores the identifier of the file associated with node *u*, and *lchild* and *rchild* are the pointers to the left and right child of node *u*, respectively. Each leaf node $u \in \mathcal{T}$ is associated with a distinct file $f \in F$. Therefore, the number of leaf nodes of $\mathcal{T}$ equals the number of files *n* in *F*, and the height of $\mathcal{T}$ is $\lceil \log n \rceil$.

Specifically, if node *u* is a leaf node of $\mathcal{T}$ and is associated with file f_i, *fid* stores the identifier of file f_i, *data* stores the information about file keywords W_i, and both *lchild* and *rchild* are set to null. If node *u* is an internal node of $\mathcal{T}$, *fid* is set to null, *data* stores the information about file keywords associated with its children nodes. The search process is a recursive procedure upon the index tree $\mathcal{T}$. Given a query ϑ_j, the cloud server performs a detection starting from the root node of $\mathcal{T}$. If a node *u* passes the test, the cloud server checks all its children nodes; otherwise, the cloud server stops traversing the subtree rooted at node *u*. When this traversal is over, the cloud server returns all the reached leaves.

C. The Definition of M2FS

Since file privacy can be preserved through standard SKE, e.g., AES, those algorithms related to file encryption/decryption are omitted in this work. As far as the query process is concerned, our M2FS schemes consist of the following algorithms:

- $SK \leftarrow Init(1^\lambda)$*:* The data owner takes the security parameter λ as input and outputs a secret key *SK*.
- $\mathcal{I} \leftarrow BuildIndex(F, W, SK)$*:* Given the secret key *SK*, the data owner builds an encrypted index $\mathcal{I}$ based on the file set *F* and the keyword set *W*.
- $T_{\vartheta_j} \leftarrow GenTrap(\vartheta_j, SK)$*:* Given the secret key *SK*, the data user creates a trapdoor T_{ϑ_j} for query ϑ_j.
- $C_{\vartheta_j} \leftarrow Search(\mathcal{I}, T_{\vartheta_j})$*:* The cloud server evaluates the trapdoor T_{ϑ_j} on the encrypted index $\mathcal{I}$ and outputs a set of ciphertexts C_{ϑ_j} of files matching query ϑ_j.

Definition 4 (Correctness of M2FS): Given the secret key *SK* generated by algorithm *Init*, the encrypted index $\mathcal{I}$ generated by algorithm *BuildIndex*, and the trapdoor T_{ϑ_j} generated by algorithm *GenTrap*, M2FS is correct if, for each ciphertext in C_{ϑ_j} output by algorithm $Search(\mathcal{I}, T_{\vartheta_j})$, corresponding file matches query ϑ_j.

D. Security Definition

Following the approach in [3], we utilize notations *history*, *view*, and *trace* to define the semantic security of our scheme.

- *History:* $\mathcal{H} = (F, W, \mathbb{Q})$ consists of a file collection *F*, a keyword set *W*, and the query set $\mathbb{Q} = (\vartheta_1, \ldots, \vartheta_t)$ that the data user wishes to search for. A history reflects the interaction between the data owner/user and the cloud server.
- *View:* $\mathcal{V} = (C, \mathcal{I}, \mathbb{T})$ consists of the encrypted file set *C*, the encrypted index $\mathcal{I}$, and the set of trapdoors

$\mathbb{T} = (T_{\vartheta_1}, \ldots, T_{\vartheta_t})$. A view reflects what the cloud server can actually see.

- *Trace of History: $Tr(\mathcal{H})$* consists of the information about the history $\mathcal{H}$ that can be leaked to the cloud server. Different threat model causes different search leakages, and the trace of our schemes will be listed in detail in Section IV and Section V.

Let Adv be a PPT adversary, and Sim be a PPT simulator. We consider the following probabilistic experiments:

- $\textbf{Real}_{\text{Adv}}(\lambda)$*:* The challenger runs $\mathit{Init}(1^{\lambda})$ to generate secret key *SK*. Adversary Adv outputs *F* and *W*, and receives $\{\mathcal{I}, C\}$ from the challenger where $\mathcal{I} \leftarrow \mathit{BuildIndex}(F, W, SK)$ and *C* contains ciphertexts of files in *F*. Adv makes a polynomial number of queries $\mathbb{Q} = (\vartheta_1, \ldots, \vartheta_t)$ and for each query $\vartheta_j \in \mathbb{Q}$, Adv receives a trapdoor T_{ϑ_j} from the challenger such that $T_{\vartheta_j} \leftarrow \mathit{GenTrap}(\vartheta_j, SK)$. Finally, Adv outputs view $\mathcal{V} = (\mathcal{I}, C, \mathbb{T})$, where $\mathbb{T} = (T_{\vartheta_1}, \ldots, T_{\vartheta_t})$.
- $\textbf{Ideal}_{\text{Adv,Sim}}(\lambda)$*:* Adversary Adv outputs *F* and *W*. Sim generates and sends $\{\mathcal{I}, C\}$ to Adv. Adv makes a polynomial number of queries $\mathbb{Q} = (\vartheta_1, \ldots, \vartheta_t)$. Given the trace $Tr(\mathcal{H})$, Sim returns an appropriate trapdoor T_{ϑ_j} for each query $\vartheta_j \in \mathbb{Q}$. Finally, Adv outputs the view $\mathcal{V} = (\mathcal{I}, C, \mathbb{T})$, where $\mathbb{T} = (T_{\vartheta_1} \ldots, T_{\vartheta_t})$.

Definition 5 (Semantic Security of M2FS): M2FS is semantically secure if, for all PPT adversaries Adv, there exists a PPT simulator Sim such that the probability of $|\Pr[\textbf{Real}_{\text{Adv}}(\lambda) = 1] - \Pr[\textbf{Ideal}_{\text{Adv,Sim}}(\lambda) = 1]|$ is negligible.

IV. The Basic M2FS Scheme

A. Rationale

Before providing the detailed construction, we first introduce the key techniques used in our basic M2FS scheme.

The Vector Encoding Method: The core idea is to encode the *k*-th keyword $w_{i,k}$ associated with file f_i (resp. the *l*-th keyword $\widetilde{w}_{j,l}$ in query ϑ_j) as an index vector $\mathbf{p}_{i,k} \in \mathbb{R}^d$ (resp. a query vector $\mathbf{q}_{j,l} \in \mathbb{R}^d$), subtly filling $\mathbf{p}_{i,k}$ and $\mathbf{q}_{j,l}$ with specific primes and the reciprocals of those primes, respectively. With the indecomposable property of primes, $\mathbf{p}_{i,k} \cdot \mathbf{q}_{j,l}$, is an integer only when $w_{i,k} \approx \widetilde{w}_{j,l}$.

The Matrix Construction Method: For file f_i that contains β_i keywords, an index matrix $\mathbf{P}_i \in \mathbb{R}^{\beta_i \times d}$ is constructed, where the elements in the *k*-th row of $\mathbf{P}_i$ are set to the elements of $\mathbf{p}_{i,k}$, expressed as $\mathbf{P}_i[k][\,*\,] \leftarrow \mathbf{p}_{i,k}$, for $k \in [\![\beta_i]\!]$. Similarly, for query ϑ_j that contains γ_j keywords, a query matrix $\mathbf{Q}_j \in \mathbb{R}^{d \times \gamma_j}$ is generated, where the elements in the *l*-th column of $\mathbf{Q}_j$ are set to the elements of $\mathbf{q}_{j,l}$, expressed as $\mathbf{Q}_j[*][l] \leftarrow \mathbf{q}_{j,l}$, for $l \in [\![\gamma_j]\!]$.

The Way to Determine Matching: To check whether query ϑ_j matches file f_i or not, $\mathbf{P}_i \star \mathbf{Q}_j$ is calculated yielding a test matrix $\mathbf{R}_{i,j} \in \mathbb{R}^{\beta_i \times \gamma_j}$ where $\mathbf{R}_{i,j}[k][l] = \mathbf{P}_i[k][\,*\,] \star \mathbf{Q}_j[\,*\,][l] = \mathbf{p}_{i,k} \cdot \mathbf{q}_{j,l}$ for $k \in [\![\beta_i]\!]$ and $l \in [\![\gamma_j]\!]$. According to **Definition 3**, for AND queries, $\vartheta_j \bowtie f_i$ if each column of $\mathbf{R}_{i,j}$ contains at least one integer; for OR queries, $\vartheta_j \bowtie f_i$ if $\mathbf{R}_{i,j}$ contains at least one integer element. Note that the order of keywords has nothing to do with the result of matching. Hence, the keywords associated with each file/query can be shuffled before the construction of matrices. Furthermore, both index and query matrices are encrypted with SKNN, which allows the calculation of matrix multiplication based on their encrypted forms.

```
Algorithm 1 Index Vector Encoding
Input: keyword $w_{i,k} \in W_i$, PRF F's secret key $\kappa$, maximal
    length of keywords $\alpha$, vectors' dimension $d$, a sequence of
    $\alpha$ primes $\mathbb{P} = (p_1, \ldots, p_\alpha)$, and a dictionary of $\alpha$ dummy
    characters $\mathcal{S} = (s_1, \ldots, s_\alpha)$
Output: index vector $\mathbf{p}_{i,k} \in \mathbb{R}^d$
if $\|w_{i,k}\| < \alpha$ then
    pad $w_{i,k}$ with $\langle s_1, \ldots, s_{\alpha - \|w_{i,k}\|} \rangle$
for $l \in [\![d]\!]$ do
    set $\mathbf{p}_{i,k}[l] = 1$
for $l \in [\![\alpha]\!]$ do
    set $\sigma_l = \mathrm{F}_\kappa(w[l])$
    set $\mathbf{p}_{i,k}[\sigma_l] = \mathbf{p}_{i,k}[\sigma_l] \times p_l$
choose $\upsilon \in [\![d - \alpha]\!]$ random elements with a value of 1
    from $\mathbf{p}_{i,k}$ and fill them with random primes outside $\mathbb{P}$
```

B. Scheme Construction

Let *SKNN* = (*GenKey*, *EncI*, *EncQ*, *Test*) be a secure KNN scheme as described in Section II-B, and let $\mathrm{F} : \{0,1\}^{\lambda} \times \{0,1\}^{*} \rightarrow \{0,1\}^{\lambda}$ be a PRF. Given the maximal length of keywords α, and a dictionary of 26 English characters $\mathcal{A}$, the basic M2FS scheme is constructed as follows:

- $SK \leftarrow \mathit{Init}(1^{\lambda})$*:* Given the security parameter $\lambda \in \mathbb{N}$, the data owner first runs algorithm *SKNN.GenKey* to generate keys $sk = (\mathbf{M}_1, \mathbf{M}_2, \mathbf{s})$. She then chooses a sequence of α random primes $\mathbb{P}$, and a dictionary of α dummy characters $\mathcal{S} = (s_1, \ldots, s_\alpha)$ such that $\mathcal{S} \cap \mathcal{A} = \emptyset$. Finally, she chooses a random λ-bit string κ as the key of the PRF F, and sets secret keys $SK = (sk, \kappa, \mathbb{P}, \mathcal{S})$.
- $\mathcal{I} \leftarrow \mathit{BuildIndex}(F, W, SK)$*:* For each file $f_i \in F$ that contains β_i keywords $W_i = \{w_{i,1}, \ldots, w_{i,\beta_i}\}$, the data owner constructs an index matrix $\mathbf{P}_i \in \mathbb{R}^{\beta_i \times d}$ as follows: (1) For the *k*-th keyword $w_{i,k}$ in W_i, she constructs a *d*-dimensional vector $\mathbf{p}_{i,k}$ by running Alg. 1. (2) For $k \in [\![\beta_i]\!]$ and $l \in [\![d]\!]$, she sets $\mathbf{P}_i[k][l] = \mathbf{p}_{i,k}[l]$. That is, for $k \in [\![\beta_i]\!]$, she sets the elements in the *k*-th row of matrix $\mathbf{P}_i$ to the elements of vector $\mathbf{p}_{i,k}$, expressed as $\mathbf{P}_i[k][\,*\,] \leftarrow \mathbf{p}_{i,k}$.

The index matrix is built on top of the index vector algorithm (Alg. 1) that consists of the following four steps: (1) **Padding.** This step pads file keyword $w_{i,k}$ with dummy characters in $\mathcal{S}$ to hide its real length. After padding, all keywords are of uniform length α. (2) **Initialization.** This step constructs a *d*-dimensional vector $\mathbf{p}_{i,k}$ where each element is initialized with 1. (3) **Mapping.** This step maps primes in $\mathbb{P}$ to appropriate positions of $\mathbf{p}_{i,k}$. For the *l*-th character of keyword $w_{i,k}$, the corresponding prime is p_l and the corresponding position is $\sigma_l = \mathrm{F}_\kappa(w_{i,k}[l])$. Therefore, $\mathbf{p}_{i,k}[\sigma_l]$ is multiplied by p_l for $l \in [\![\alpha]\!]$. (4) **Randomization.** This step fills partial remaining elements of $\mathbf{p}_{i,k}$ with random primes outside $\mathbb{P}$ to add randomness to the results of vector inner products.

Algorithm 2 Query Vector Encoding

```
Input: keyword $\widetilde{w}_{j,l} \in \widetilde{W}_j$, PRF F's secret key $\kappa$, maximal
    length of keywords $\alpha$, vectors' dimension $d$, a sequence of
    $\alpha$ primes $\mathbb{P} = (p_1, \dots, p_\alpha)$, and a dictionary of $\alpha$ dummy
    characters $\mathcal{S} = (s_1, \dots, s_\alpha)$
Output: query vector $\mathbf{q}_{j,l} \in \mathbb{R}^d$
if $\|\widetilde{w}_{j,l}\| < \alpha$ then
    pad $\widetilde{w}_{j,l}$ with $\langle s_1, \dots, s_{\alpha - \|\widetilde{w}_{j,l}\|} \rangle$
for $k \in [\![d]\!]$ do
    set $\mathbf{q}_{j,l}[k] = 1$
for $k \in [\![\alpha]\!]$ do
    if $\widetilde{w}_{j,l}[k] \neq$ "?" then
        set $\sigma_k = \mathrm{F}_\kappa(\widetilde{w}_{j,l}[k])$
        set $\mathbf{q}_{j,l}[\sigma_k] = \mathbf{q}_{j,l}[\sigma_k] \times 1/p_k$
choose $\upsilon \in [\![d - \alpha]\!]$ random elements with value of 1
    from $\mathbf{q}_{j,l}$ and fill them with random integers
```

Given an index matrix $\mathbf{P}_i$ associated with file f_i, the data owner runs algorithm *SKNN.EncI* to generate a pair of encrypted matrices $\mathbf{P}'_i = (\mathbf{P}'_{i_a}, \mathbf{P}'_{i_b}) \in \mathbb{R}^{\beta_i \times d}$. Therefore, an encrypted index is set as $\mathcal{I} = \{\mathbf{P}'_1, \dots, \mathbf{P}'_n\}$.

- $T_{\vartheta_j} \leftarrow GenTrap(\vartheta_j, SK)$: For query ϑ_j associated with γ_j keywords $\widetilde{W}_j = \{\widetilde{w}_{j,1}, \dots, \widetilde{w}_{j,\gamma_j}\}$, the data user constructs a query matrix $\mathbf{Q}_j \in \mathbb{R}^{d \times \gamma_j}$ as follows: (1) For the *l*-th keyword $\widetilde{w}_{j,l}$ in $\widetilde{W}_j$, he constructs a *d*-dimensional vector $\mathbf{q}_{j,l}$ by running Alg. 2. (2) For $l \in [\![\gamma_j]\!]$ and $k \in [\![d]\!]$, he sets $\mathbf{Q}_j[k][l] = \mathbf{q}_{j,l}[k]$. That is, for $l \in [\![\gamma_j]\!]$, it sets the elements in the *l*-th column of matrix $\mathbf{Q}_j$ to the elements of vector $\mathbf{q}_{j,l}$, expressed as $\mathbf{Q}_j[*][l] \leftarrow \mathbf{q}_{j,l}$.

The query matrix is built on top of the query vector algorithm (Alg. 2) that also consists of four steps as those of Alg. 1. Specifically, the padding and initialization steps of Alg. 2 are similar to those of Alg. 1. In contrast to Alg. 1, its mapping step maps the reciprocals of primes in $\mathbb{P}$ to appropriate positions of $\mathbf{q}_{j,l}$. For the *k*-th character of keyword $\widetilde{w}_{j,l}$, this step calculates $\sigma_k = \mathrm{F}_\kappa(\widetilde{w}_{j,l}[k])$ and multiplies $\mathbf{q}_{j,l}[\sigma_k]$ by $1/p_k$ if $\widetilde{w}_{j,l}[k] \neq$"?", and does nothing otherwise. Its randomization step replaces random 1 elements of $\mathbf{q}_{j,l}$ with random integers. Given a query matrix $\mathbf{Q}_j$ for query ϑ_j, the data user runs algorithm *SKNN.EncQ* to generate a pair of encrypted matrices $\mathbf{Q}'_j = (\mathbf{Q}'_{j_a}, \mathbf{Q}'_{j_b}) \in \mathbb{R}^{d \times \gamma_j}$. Therefore, the trapdoor is set as $T_{\vartheta_j} = \mathbf{Q}'_j$.

- $C_{\vartheta_j} \leftarrow Search(\mathcal{I}, T_{\vartheta_j})$: Given a pair of encrypted matrices $(\mathbf{Q}'_{j_a}, \mathbf{Q}'_{j_b}) \in \mathbb{R}^{d \times \gamma_j}$ in trapdoor T_{ϑ_j}, for each pair of encrypted index matrices $(\mathbf{P}'_{i_a}, \mathbf{P}'_{i_b}) \in \mathbb{R}^{\beta_i \times d}$ in index $\mathcal{I}$, the cloud server runs algorithm *SKNN.Test* to generate a test matrix $\mathbf{R}_{i,j} = \mathbf{P}'_{i_a} \star \mathbf{Q}'_{j_a} + \mathbf{P}'_{i_b} \star \mathbf{Q}'_{j_b} = \mathbf{P}_i \star \mathbf{Q}_j$. Therefore, $\mathbf{R}_{i,j}$ is an $(\beta_i \times \gamma_j)$-dimensional matrix, where $\mathbf{R}_{i,j}[k][l] = \mathbf{P}_i[k][*] \star \mathbf{Q}_j[*][l] = \mathbf{p}_{i,k} \cdot \mathbf{q}_{j,l}$ for $k \in [\![\beta_i]\!]$ and $l \in [\![\gamma_j]\!]$. Finally, the cloud server puts corresponding ciphertext c_i into set C_{ϑ_j} if either of the following cases happens: (1) For an AND query ϑ_j, $\mathbf{R}_{i,j}$ has at least one integer in each column; (2) For an OR query ϑ_j, $\mathbf{R}_{i,j}$ has at least one integer element.

Remark 1: The randomization step in both Alg. 1 and Alg. 2 is necessary for hiding the similarity between keywords. With this step, for two identical file keywords, the generated index vectors are different, rendering different results of inner products with a given query vector. Likewise, this step enables two identical query keywords to have different results of inner products with a given index vector.

Furthermore, the randomization step is helpful to resist COA. Given an index vector $\mathbf{p}_{i,k}$ and a query vector $\mathbf{q}_{j,l}$, the result of vector inner product can be expressed as $\mathbf{p}_{i,k} \cdot \mathbf{q}_{j,l} = \delta + \xi$, where δ is a random integer generated by the joint effect of the randomization step in both algorithms, and ξ is a random value with the following properties:

(1) If $w_{i,k} \approx \widetilde{w}_{j,l}$, $\xi \in [\![\alpha]\!]$ denotes the number of distinct characters in query keyword $\widetilde{w}_{j,l}$. Therefore, $\mathbf{p}_{i,k} \cdot \mathbf{q}_{j,l}$ looks like a random integer leaking nothing to the cloud server.

(2) If $w_{i,k} \not\approx \widetilde{w}_{j,l}$, we consider the simplest case where the *h*-th character is different in two keywords. This means that the reciprocal of primes in $\mathbf{q}_{j,l}[\sigma_h]$ cannot be cancelled, where $\sigma_h = \mathrm{F}_\kappa(\widetilde{w}_{j,l}[h])$. Let x_h and $1/p_h$ denote the value located at $\mathbf{p}_{i,k}[\sigma_h]$ and $\mathbf{q}_{j,l}[\sigma_h]$, respectively. Here, ξ can be rewritten as $y + \frac{x_h}{p_h}$, where *y* relates to the number of identical characters in two keywords. Since x_h equals 1 or is a random prime outside $\mathbb{P}$, $\mathbf{p}_{i,k} \cdot \mathbf{q}_{j,l}$ is a random real. While extending to the case where two keywords have two or more different characters, more random primes will be introduced, it is more difficult for the cloud server to obtain valuable information from the results. Therefore, our vector encoding method can resist COA.

Remark 2: The calculation reciprocals of primes can involve a loss of precision, thereby impacting search accuracy. That is, the result of 1/7 × 7 may be 1.00001 instead of 1. To solve this problem, our method is to round up after the *z*-th decimal point, where *z* can be adjusted in experiments.

Remark 3: To support AND and OR semantics simultaneously, a query will be transformed into disjunctive normal form (DNF) or conjunctive normal form (CNF). Taking DNF as an example, query $\vartheta_j = \texttt{cl??d} \wedge (\texttt{s?curi??} \vee \texttt{privacy})$ can be expressed as $\widetilde{\vartheta}_j = (\texttt{cl??d} \wedge \texttt{s?curi??}) \vee (\texttt{cl??d} \wedge \texttt{privacy})$. From $\widetilde{\vartheta}_j$, we know that a file will be returned if its keywords are similar with either the first and second keywords in the query or the first and third keywords in the query. Therefore, the data user sends the trapdoor T_{ϑ_j} as well as the indicator $I = \{(1, 2)(1, 3)\}$ to the cloud server. For file $f_i \in F$, the cloud server first generates a test matrix $\mathbf{R}_{i,j}$ by running the *Search* algorithm, and then determines $\vartheta_j \bowtie f_i$ if either of the following cases happens: (1) Both the first and second columns of $\mathbf{R}_{i,j}$ contain at least one integer; (2) Both the first and third columns of $\mathbf{R}_{i,j}$ contain at least one integer.

C. Correctness Analysis

The correctness of our basic M2FS scheme is guaranteed by the deterministic mapping of the input and output of PRF. For the sake of clarity, we first assume that a file f_i or a query ϑ_j contains only one keyword, denoted by w_i and $\widetilde{w}_j$, respectively. In this single-keyword setting, the test matrix $\mathbf{R}_{i,j}$ contains only one element that is equal to $\mathbf{p}_i \cdot \mathbf{q}_j$.

Therefore, the basic M2FS scheme is considered incorrect if either of the following cases occurs:

Case 1: The result of $\mathbf{p}_i \cdot \mathbf{q}_j$ is not an integer if $w_i \approx \widetilde{w}_j$.

Case 2: The result of $\mathbf{p}_i \cdot \mathbf{q}_j$ is an integer if $w_i \not\approx \widetilde{w}_j$.

Let $\mathcal{U} = \mathcal{A} \cup \mathcal{S}$ denote the union of the English alphabet $\mathcal{A}$ and the dummy characters $\mathcal{S}$ with $\mathcal{U}[k]$ denoting its k-th element. For **Case 1**, a result where $\mathbf{p}_i \cdot \mathbf{q}_j$ is not an integer means that at least one reciprocal in $\mathbf{q}_j$ cannot be eliminated. Due to the construction of algorithm *GenTrap*, $\mathbf{q}_j[\mathrm{F}_\kappa(\mathcal{U}[k])] = 1/p_l$ means that $\mathcal{U}[k]$ is the l-th character of $\widetilde{w}_j$. If $1/p_l$ cannot be eliminated, $\mathbf{p}_i[\mathrm{F}_\kappa(\mathcal{U}[k])]$ is set to an integer that is indivisible by p_l. According to the construction of algorithm *BuildIndex*, it means that $\mathcal{U}[k]$ cannot appear in the l-th position of w_i. Therefore, two keywords are dissimilar. It contradicts the assumption, and thus **Case 1** is untrue.

For **Case 2**, a result where $\mathbf{p}_i \cdot \mathbf{q}_j$ is an integer means that all reciprocals in $\mathbf{q}_j$ are eliminated. That is, if $\mathbf{q}_j[\mathrm{F}_\kappa(\mathcal{U}[k])] = 1/p_l$, $\mathbf{p}_i[\mathrm{F}_\kappa(\mathcal{U}[k])]$ is set to an integer that is divisible by p_l. Due to the construction of the *BuildIndex* algorithm, it means that $\mathcal{U}[k]$ will appear in the l-th position of w_i. In other words, two keywords are similar, contradicting the assumption. Therefore, **Case 2** is not true, and our basic M2FS scheme is correct in the single-keyword setting.

The correctness of the multi-keyword setting can be derived as follows: From Eq. 1, we know that test matrix $\mathbf{R}_{i,j}$ is an $(\beta_i \times \gamma_j)$-dimensional matrix, where $\mathbf{R}_{i,j}[k][l] = \mathbf{p}_{i,k} \cdot \mathbf{q}_{j,l}$ for $k \in \llbracket \beta_i \rrbracket$ and $l \in \llbracket \gamma_j \rrbracket$. For an AND query, if $\mathbf{R}_{i,j}$ has at least one integer in each column, it means that each keyword in $\widetilde{W}_j$ can find a similar keyword in W_i. For an OR query, if $\mathbf{R}_{i,j}$ has at least one integer element, it means that at least one keyword in $\widetilde{W}_j$ can find a similar keyword in W_i. This argument agrees with **Definition** 3, and our basic M2FS scheme is correct.

D. Security Proof

Theorem 1: Our basic M2FS scheme is semantically secure in the known ciphertext model if F is a secure PRF.

The proof can be found in Appendix A.

V. The Advanced M2FS Scheme

The basic M2FS scheme allows multi-keyword fuzzy search while simultaneously supporting AND and OR semantics. However, it has the following drawbacks: (1) It builds the search index like a forward index, and requires the cloud server to sequentially scan the whole file collection, resulting in $O(n)$ search time. (2) It applies SKNN to encrypt index/query matrices, and thus is vulnerable to linear analyses. To overcome these shortcomings, the advanced M2FS scheme, on one hand, builds a tree-based search index over a file collection to achieve parallel searches; on the other hand, it achieves semantic security in the known-background model by constructing expansive query matrices.

A. The Construction of a Tree-Based Index

Algorithm 3 Index Tree Construction

```
Input: file collection $F = \{f_1, \ldots, f_n\}$, index matrices $\{\mathbf{P}_1, \ldots, \mathbf{P}_n\}$
Output: the index tree $\mathcal{T}$
initialize two empty sets $N$ and $N'$
for each file $f$ in $F$ do
    construct a leaf node $u$ for $f$ with Eq. (3)
    insert $u$ to set $N$
while $|N| > 1$ do
    while ($|N| > 3$) OR ($|N|$ is even AND $|N| > 0$) do
        for each pair of nodes $u_i$ and $u_j$ in $N$ do
            generate a parent node $v$ with Eq. (4)
            remove nodes $u_i$ and $u_j$ from $N$
            insert $v$ to set $N'$
    if $|N|$ equals 3 then
        generate a parent node $v$ with Eq. (4) for a pair of nodes $u_i$ and $u_j$ in $N$
        generate a parent node $v'$ with Eq. (4) for the parent node $v$ and the last node $u_l$ in $N$
        remove nodes $u_i$, $u_j$, $u_l$ from $N$
        insert $v'$ to set $N'$
    replace $N$ with $N'$ and clear $N'$
return the only node left in $N$, namely, the root $v_0$ of index tree $\mathcal{T}$
```

An index tree $\mathcal{T}$ is constructed as a KBB tree as illustrated in Section III-B. Let FID and NID be functions that output a unique identifier for a file $f \in F$ and a node $u \in \mathcal{T}$, respectively. For each file $f \in F$, the corresponding leaf node $u \in \mathcal{T}$ is constructed with Eq. (3):

$$\begin{aligned} &u.nid = \mathrm{NID}(u), u.data = \mathbf{P}_{\mathrm{FID}(f)}, u.fid = \mathrm{FID}(f), \\ &u.lchild = null, u.rchild = null \end{aligned} \tag{3}$$

where $\mathbf{P}_{\mathrm{FID}(f)} \in \mathbb{R}^{\beta_{\mathrm{FID}(f)} \times d}$ is an index matrix of file f. For each internal node v with left child v_l and right child v_r, its fields are computed as follows:

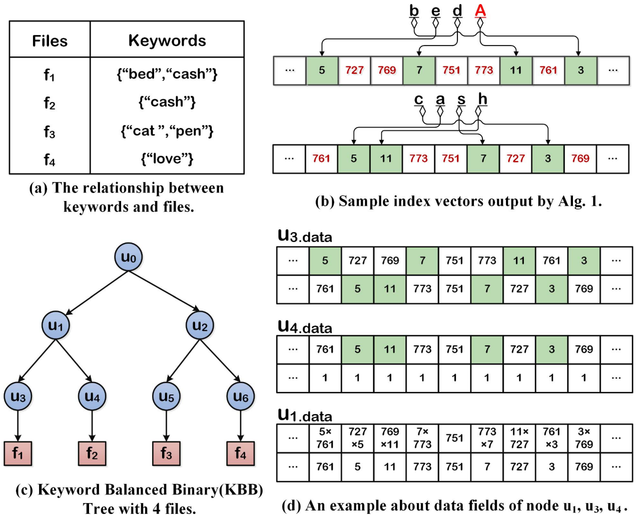
$$\begin{aligned} &v.nid = \mathrm{NID}(v), v.data = v_l.data \odot v_r.data, \\ &v.fid = null, v.lchild = v_l, v.rchild = v_r \end{aligned} \tag{4}$$

In Eq. (4), the data field of node v is computed as $v.data = v_l.data \odot v_r.data$, where $\odot$ denotes element-wise multiplication operation of two matrices. However, each file is associated with a different number of keywords, and the number of rows in each index matrix may be different. Let $\beta = \max(\beta_1, \ldots, \beta_n)$ be the maximal number of keywords associated with files in F. To make operation $\odot$ reasonable, each index matrix is expanded to $(\beta \times d)$ dimensions by padding with dummy rows where all elements are set to 1. Let $v.\mathbf{P}$ denote the matrix in the data field of node v. Therefore, matrix $v.\mathbf{P} \in \mathbb{R}^{\beta \times d}$ is constructed with Eq. (5):

$$v.\mathbf{P}[k][l] = v_l.\mathbf{P}[k][l] \times v_r.\mathbf{P}[k][l] \tag{5}$$

where $k \in \llbracket \beta \rrbracket$ and $l \in \llbracket d \rrbracket$.

With the above definitions about tree nodes, Alg. 3 constructs a KBB tree $\mathcal{T}$ in an iterative manner. Given two empty sets N and N' used to hold the currently processing children nodes and the newly generated parent nodes, respectively, this algorithm generates n leaf nodes for tree $\mathcal{T}$ and puts all of them into set N. Next, this algorithm builds tree $\mathcal{T}$ iteratively from bottom to top in the following way: If $|N| = 1$, it outputs



Files	Keywords
f_1	{"bed","cash"}
f_2	{"cash"}
f_3	{"cat ","pen"}
f_4	{"love"}

(a) The relationship between keywords and files.

(b) Sample index vectors output by Alg. 1.

(c) Keyword Balanced Binary(KBB) Tree with 4 files.

u3.data

…	5	727	769	7	751	773	11	761	3	…
…	761	5	11	773	751	7	727	3	769	…

u4.data

…	761	5	11	773	751	7	727	3	769	…
…	1	1	1	1	1	1	1	1	1	…

u1.data

…	5× 761	727 ×5	769 ×11	7× 773	751	773 ×7	11× 727	761 ×3	3× 769	…
…	761	5	11	773	751	7	727	3	769	…

(d) An example about data fields of node u_1, u_3, u_4.

Fig. 2. Construction process of Keyword Balanced Binary(KBB) tree.

the unique element in N as the root of tree $\mathcal{T}$ and terminates; otherwise, it executes lines 7-10. This process will not stop until either of the following cases happens: (1) If $|N| = 3$, it executes lines 12-15. (2) If $|N| = 0$, it executes line 16 and then goes back to line 5.

Example 1: Provided that the relationship between files and keywords is as shown in Fig. 2-(a). Given the maximal length of keywords $\alpha = 4$, random primes $\mathbb{P} = (3, 5, 7, 11)$ and dummy characters $\mathcal{S} = (\text{"}A\text{"}, \text{"}B\text{"}, \text{"}C\text{"}, \text{"}D\text{"})$, the index keyword vectors could be constructed as shown in Fig. 2-(b) according to Alg. 1. In the index vectors, the filled random numbers are marked in red. Fig. 2-(c) illustrates a KBB tree $\mathcal{T}$ built from file collection $F = \{f_1, f_2, f_3, f_4\}$ according to Alg. 3, where the sample matrices are shown in Fig. 2-(d). In this example, the maximal number of keywords is 2, and thus the matrix of node $\mathbf{u}_4$ is padded with one dummy row.

B. The Construction of an Expansive Query Matrix

In the known-background model, the cloud server may obtain certain statistic information, e.g., the keyword frequency and distribution. With such information, the cloud server can infer certain keyword/trapdoor pairs. As proven in [36], SKNN is vulnerable to linear analyses if the number of exposed pairs is large enough. To illustrate, let us review the process of generating test matrix $\mathbf{R}_{i,j} \in \mathbb{R}^{\beta \times \gamma_j}$:

$$\mathbf{R}_{i,j} = \mathbf{P}'_{i_a} \star \mathbf{Q}'_{j_a} + \mathbf{P}'_{i_b} \star \mathbf{Q}'_{j_b} = \mathbf{P}_i \star \mathbf{Q}_j \tag{6}$$

where there are only $\beta \times d$ unknown variables (i.e., the elements of $\mathbf{P}_i$) if query matrix $\mathbf{Q}_j$ are exposed. Thus, the cloud server can solve linear equations to recover plaintext index matrices with sufficient query matrix and trapdoor pairs. To resist such an attack, the advanced scheme expands a query matrix with random noises so that Eq. (6) holds with a negligible probability. The main trick is that the random noises tactfully filled will not impact the correctness of our scheme. That is, for the expansive query matrix, the integral/non-integral properties of its elements remain unchanged.

Algorithm 4 Query Matrix Expansion

```
Input: $\mu \in \llbracket 2 \sim d \rrbracket$, query matrix $\mathbf{Q}_j \in \mathbb{R}^{d \times \gamma_j}$ for query $\vartheta_j$
Output: Expansive query matrix $\widehat{\mathbf{Q}}_j \in \mathbb{R}^{d \times (\mu\gamma_j)}$
for $l \in \left\llbracket \gamma_j \right\rrbracket$ do
    for $k \in \llbracket d \rrbracket$ do
        set $\mathbf{Q}_j[k][l]$ at a random position of the $k$-th row of $\widehat{\mathbf{Q}}_j\llbracket (l-1)\mu + 1, l\mu \rrbracket$ under Requirement 1
        fill the remaining elements in the $k$-th row of $\widehat{\mathbf{Q}}_j\llbracket (l-1)\mu + 1, l\mu \rrbracket$ with random numbers under Requirement 2
```

Let $\mu \in \llbracket 2, d \rrbracket$ be a security parameter determining the amount of random noises to be added. Given a query matrix $\mathbf{Q}_j \in \mathbb{R}^{d \times \gamma_j}$, Alg. 4 generates an expansive query matrix $\widehat{\mathbf{Q}}_j \in \mathbb{R}^{d \times (\mu\gamma_j)}$ as follows. Let $\widehat{\mathbf{Q}}_j\llbracket x, y \rrbracket$ be a submatrix made up of matrix columns $\widehat{\mathbf{Q}}_j[*][x], \ldots, \widehat{\mathbf{Q}}_j[*][y]$. For $l \in \llbracket \gamma_j \rrbracket$, this algorithm first randomly scatters the elements of $\mathbf{Q}_j[*][l]$ to submatrix $\widehat{\mathbf{Q}}_j\llbracket (l-1)\mu + 1, l\mu \rrbracket$, and then it fills submatrix $\widehat{\mathbf{Q}}_j\llbracket (l-1)\mu + 1, l\mu \rrbracket$ with random numbers. Specifically, given the value of l, for $k \in \llbracket d \rrbracket$, this algorithm places $\mathbf{Q}_j[k][l]$ at a random position of the k-th row of $\widehat{\mathbf{Q}}_j\llbracket (l-1)\mu + 1, l\mu \rrbracket$ such that **Requirement 1** is satisfied, and then it fills the remaining elements in the k-th row of $\widehat{\mathbf{Q}}_j\llbracket (l-1)\mu + 1, l\mu \rrbracket$ with random numbers such that **Requirement 2** is satisfied.

- *Requirement 1:* Each column of $\widehat{\mathbf{Q}}_j\llbracket (l-1)\mu + 1, l\mu \rrbracket$ contains at least one element of $\mathbf{Q}_j[*][l]$.
- *Requirement 2:* The sum of random numbers at the k-th row of $\widehat{\mathbf{Q}}_j\llbracket (l-1)\mu + 1, l\mu \rrbracket$, denoted as δ_k, is equal to $t_k\mathbf{Q}_j[k][l]$ where $t_k = 0$ or $(t_k + 1)$ is a prime outside $\mathbb{P}$.

Given an index matrix $\mathbf{P}_i \in \mathbb{R}^{\beta \times d}$ generated for file f_i and an expansive query matrix $\widehat{\mathbf{Q}}_j \in \mathbb{R}^{d \times (\mu\gamma_j)}$ generated for query ϑ_j, an intermediate matrix $\widehat{\mathbf{R}}_{i,j} \in \mathbb{R}^{\beta \times (\mu\gamma_j)}$ is obtained by calculating $\mathbf{P}_i \star \widehat{\mathbf{Q}}_j$. The test matrix $\mathbf{R}_{i,j} \in \mathbb{R}^{\beta \times \gamma_j}$ is obtained by merging each μ columns of $\widehat{\mathbf{R}}_{i,j}$ together as follows: For $k \in \llbracket d \rrbracket$ and $l \in \llbracket \gamma_j \rrbracket$, we set $\mathbf{R}_{i,j}[k][l] = \sum_{y=1}^{\mu} \widehat{\mathbf{R}}_{i,j}[k][(l-1)\mu + y]$. With the test matrix $\mathbf{R}_{i,j}$, the matching of file f_i and query ϑ_j can be determined in the same way as the basic M2FS scheme.

C. Scheme Construction

Given parameters $\beta = \max(\beta_1, \ldots, \beta_n)$ and $\mu \in \llbracket 2, d \rrbracket$, our advanced M2FS scheme is constructed as follows.

- $SK \leftarrow Init(1^\lambda)$*:* The data owner runs the basic *Init* algorithm to generate the secret key $SK = (sk, \kappa, \mathbb{P}, \mathcal{S})$.
- $\mathcal{I} \leftarrow BuildIndex(F, W, SK)$*:* The data owner first generates a set of index matrices $\{\mathbf{P}_1, \ldots, \mathbf{P}_n\}$ by running the basic *BuildIndex* algorithm. Then, it builds a tree-based index $\mathcal{T}$ by running Alg. 3. For each node $u \in \mathcal{T}$, it encrypts $u.\mathbf{P} \in \mathbb{R}^{\beta \times d}$ in the data field by running algorithm $SKNN.EncI$, and outputs a pair of encrypted matrices $u.\mathbf{P}' = (u.\mathbf{P}'_a, u.\mathbf{P}'_b)$. The encrypted index is set as $\mathcal{I} = v_0$, where v_0 is the root node of tree $\mathcal{T}$.
- $T_{\vartheta_j} \leftarrow GenTrap(\vartheta_j, SK)$*:* The data user first generates a query matrix $\mathbf{Q}_j \in \mathbb{R}^{d \times \gamma_j}$ by running the Basic *GenTrap* algorithm. Then, it generates an expansive query matrix $\widehat{\mathbf{Q}}_j \in$

$\mathbb{R}^{d\times(\mu\gamma_j)}$ by running Alg. 4. Next, it encrypts query matrix $\widehat{\mathbf{Q}}_j$ by running algorithm *SKNN.EncQ* and outputs a pair of encrypted matrices $\widehat{\mathbf{Q}}'_j = (\widehat{\mathbf{Q}}'_{j_a}, \widehat{\mathbf{Q}}'_{j_b})$. The trapdoor is set as $T_{\vartheta_j} = \widehat{\mathbf{Q}}'_j$.

- $C_{\vartheta_j} \leftarrow$ *Search*$(\mathcal{I}, T_{\vartheta_j})$: Given a pair of encrypted matrices $(\widehat{\mathbf{Q}}'_{j_a}, \widehat{\mathbf{Q}}'_{j_b}) \in \mathbb{R}^{d\times(\mu\gamma_j)}$ in trapdoor T_{ϑ_j}, the cloud server executes detection starting from the root node v_0 of index tree $\mathcal{T}$. Given a pair of encrypted matrices $(v_0.\mathbf{P}'_a, v_0.\mathbf{P}'_b) \in \mathbb{R}^{\beta\times d}$ in the data field of node v_0, it runs algorithm *SKNN.Test* to calculate an intermediate matrix $v_0.\widehat{\mathbf{R}}_j \in \mathbb{R}^{\beta\times(\mu\gamma_j)}$:

$$v_0.\widehat{\mathbf{R}}_j = v_0.\mathbf{P}'_a \star \widehat{\mathbf{Q}}'_{j_a} + v_0.\mathbf{P}'_b \star \widehat{\mathbf{Q}}'_{j_b} = v_0.\mathbf{P} \star \widehat{\mathbf{Q}}_j \tag{7}$$

To determine whether node v_0 passes the test or not, it generates a test matrix $v_0.\mathbf{R}_j \in \mathbb{R}^{\beta\times\gamma_j}$ by merging every μ columns of $v_0.\widehat{\mathbf{R}}_j$ as follows: For $k \in [\![\beta]\!]$ and $l \in [\![\gamma_j]\!]$, its sets $v_0.\mathbf{R}_j[k][l] = \sum_{y=1}^{\mu} v_0.\widehat{\mathbf{R}}_j[k][(l-1)\mu + y]$. As our basic M2FS scheme, node v_0 passes the test if each column of $v_0.\mathbf{R}_j$ has at least one integer for an AND query or if $v_0.\mathbf{R}_j$ has at least one integer element for an OR query. The search process is a recursive procedure upon the index tree $\mathcal{T}$. If node $v \in \mathcal{T}$ passes the test, all of its children nodes will be checked. When this traversal is over, it puts files associated with the reached leaves into C_{ϑ_j}.

D. Correctness Analysis

The main difference from the basic M2FS scheme is that the advanced version adds algorithms for the constructions of an index tree and an expansive query matrix. As the correctness of the basic M2FS scheme has been proven in Section IV-C, the correctness of our advanced M2FS scheme can be directly deduced from that of the index tree construction algorithm (Alg. 3) and the query expansion algorithm (Alg. 4).

Let $L_v = \{u\}$ denote a set of leaf nodes which are the descendants of node *v* in tree $\mathcal{T}$, and let $v.\mathbf{R}_j = v.\mathbf{P} \star \mathbf{Q}_j$ denote the test matrix for node *v* and query ϑ_j. First, we demonstrate the correctness of Alg. 3 without consideration of an expansive query matrix. This algorithm is considered incorrect if either of the following cases occurs:

Case 1: There is a node *u* in L_v passing the test of query ϑ_j, when node *v* fails the test.

Case 2: All nodes in L_v fail the test of ϑ_j, when node *v* passes the test.

Suppose that node *v* is node *u*'s direct ancestor, with node *u* being its left child. In Alg. 3, node *v*'s data field is calculated with Eq. (5). Therefore, we have:

$$\begin{aligned} v.\mathbf{R}_j[k][l] &= v.\mathbf{P}[k][\,*\,] \star \mathbf{Q}_j[\,*\,][l] \\ &= \sum_{x=1}^{d} v.\mathbf{P}[k][x] \times \mathbf{Q}_j[x][l] \\ &= \sum_{x=1}^{d} v_r.\mathbf{P}[k][x] \times u.\mathbf{P}[k][x] \times \mathbf{Q}_j[x][l] \end{aligned} \tag{8}$$

where $k \in [\![\beta]\!]$, and $l \in [\![\gamma_j]\!]$.

For **Case 1** where node *u* in L_v passes the test of query ϑ_j, it means that each column of $u.\mathbf{R}_j$ has an integer element for an AND query, and $u.\mathbf{R}_j$ has at least one integer element for an OR query. If $u.\mathbf{R}_j[k][l]$ is an integer, i.e., $\sum_{x=1}^{d} u.\mathbf{P}[k][x] \times \mathbf{Q}_j[x][l]$ is an integer, all reciprocals in $\mathbf{Q}_j[\,*\,][l]$ will be cancelled due to the indecomposable property of primes. Since each element in $v_r.\mathbf{P}[k][*]$ is an integer, $v.\mathbf{R}_j[k][l]$ is also an integer. Therefore, node *v* passes the test of query ϑ_j and **Case 1** is false.

For **Case 2** where node *v* passes the test of query ϑ_j, it means that each column of $v.\mathbf{R}_j$ has an integer for an AND query, and $v.\mathbf{R}_j$ has at least one integer element for an OR query. Due to the indecomposable property of primes, if $v.\mathbf{R}_j[k][l]$ is an integer, the reciprocal in $\mathbf{Q}_j[x][l]$ can be eliminated by either $u.\mathbf{P}[k][x]$ or $v_r.\mathbf{P}[k][x]$ for $x \in [\![d]\!]$. That is, either node *u* or node v_r passes the test of query ϑ_j, rendering **Case 2** false. Since the index tree $\mathcal{T}$ is constructed from bottom to top in an recursive way, the correctness of Alg. 3 is derived.

Next, we will prove the correctness of Alg. 4 by testifying that the introduced randomness in a query matrix will not impact the integral/non-integral property of results. In other word, $v.\mathbf{R}_j[k][l]$ is an integer if and only if $v.\mathbf{P}[k][*] \star \mathbf{Q}_j[*][l]$ is an integer for $k \in [\![\beta]\!]$, and $l \in [\![\gamma_j]\!]$. Given the matrix $v.\mathbf{P}_i$ and the expansive query matrix $\widehat{\mathbf{Q}}_j$, the intermediate matrix is calculated as $v.\widehat{\mathbf{R}}_j = v.\mathbf{P} \star \widehat{\mathbf{Q}}_j$, where $v.\widehat{\mathbf{R}}_j[k][l] = \sum_{x=1}^{d} v.\mathbf{P}[k][x] \times \widehat{\mathbf{Q}}_j[x][l]$ for $k \in [\![\beta]\!]$, and $l \in [\![\mu\gamma_j]\!]$. To obtain the test matrix, we merge every μ columns of $v.\widehat{\mathbf{R}}_{i,j}$ by setting $v.\mathbf{R}_j[k][l] = \sum_{y=1}^{\mu} v.\widehat{\mathbf{R}}_j[k][(l-1)\mu + y]$, and infer Eq. (10), shown at the bottom of the next page, based on **Requirement** 1 and **Requirement** 2. For $x \in [\![d]\!]$, $t_x = 0$ or $(t_x + 1)$ is a prime that is outside $\mathbb{P}$, and the result of $(t_x + 1)\mathbf{P}_i[k][x] \times \mathbf{Q}_j[x][l]$ is an integer only when $v.\mathbf{P}[k][x] \times \mathbf{Q}_j[x][l]$ is an integer. Due to the indecomposable property of primes, $v.\mathbf{R}_j[k][l]$ has the same integral/non-integral property as the result of $v.\mathbf{P}[k][*] \star \mathbf{Q}_j[*][l]$. Therefore, Alg. 4 is also correct, and the correctness of our advanced M2FS scheme is proven.

E. Security Proof

Theorem 2: The SKNN scheme with expansive query matrix can resist linear analyses.

The proof can be found in Appendix B.

Theorem 3: Our advanced M2FS scheme is semantically secure in the known background model if F is a secure PRF.

The proof can be found in Appendix C.

F. Discussion

Optimization of Index Matrices: In the process of constructing the matrix associated with parent node *v*, if $v_l.\mathbf{P}[k][l]$ and $v_r.\mathbf{P}[k][l]$ are non-coprime, their common divisors will be multiplied twice in Eq. (5). To keep the product value from getting too large, matrix $v.\mathbf{P}$ can be calculated with Eq. (9) instead: for $k \in [\![\beta]\!]$ and $l \in [\![d]\!]$,

$$v.\mathbf{P}[k][l] = \frac{v_l.\mathbf{P}[k][l] \times v_r.\mathbf{P}[k][l]}{\prod_{p\in\mathbb{L}_{k,l}} p} \tag{9}$$

where $\mathbb{L}_{k,l} = \{p\}$ is a set of primes that are the common divisors of $v_l.\mathbf{P}[k][l]$ and $v_r.\mathbf{P}[k][l]$. This makes sure that each common divisor will be multiplied once.

The Impact of Parameter μ: The number of columns μ in query matrix mainly impacts the performance of algorithms *GenTrap* and *Search* in our advanced M2FS scheme. As shown in Fig. 4-(d) and Fig. 5-(d), the larger the value of μ, the higher the incurred costs. In terms of security level, even if $\mu = 2$, the adversary cannot obtain any useful information from a single column of an intermediate matrix directly. We believe that even for this low value of μ, there is a sufficient measure of security provided.

Parallel and Dynamic Search: With the tree-based indexes, the parallel search is executed as follows. Let $P = \{\rho_0, \ldots, \rho_t\}$ be a set of *t* available processors in the system. Given query ϑ_j, an idle processor $\rho_i \in P$ is chosen to perform searches starting from the root node v_0 of the index tree. If v_0 passes the test of ϑ_j, its children nodes v_{0_l} and v_{0_r} will be traversed. Specifically, processor ρ_i continues to search v_{0_l} and assigns another available processor $\rho_j \in P$ to explore v_{0_r}. The above process is recursively applied for v_0's children nodes until reaching leaf nodes. If the current processor $\rho_i \in P$ is processing node v_x that passes the test, but there is no idle processor available during the search process, then ρ_i simply selects the left child of v_x to continue, and puts the right child of v_x into a waiting queue. Once there is an idle processor, it pops the first node in the queue to continue the search.

The tree-based index also allows for dynamic updates. Similar to the work in [7], the insertion/deletion of a file *f* corresponds to the adding/removal of a leaf node *u* in tree $\mathcal{T}$. The structural update involves the necessary rebalancing, so that the tree height is maintained to be logarithmic. The update process is relatively efficient, since each update only causes the reconstruction of a subtree $\mathcal{T}_u$. Specifically, the data owner first downloads the involved subtree $\mathcal{T}_u$ from the cloud server, and updates its structure by adding/removing a leaf node. Next, the data owner re-calculates and re-encrypts the data field of each node in the subtree and uploads the new subtree $\mathcal{T}'_u$ to the cloud server. Finally, the cloud server replaces $\mathcal{T}_u$ with $\mathcal{T}'_u$ to finish the update process.

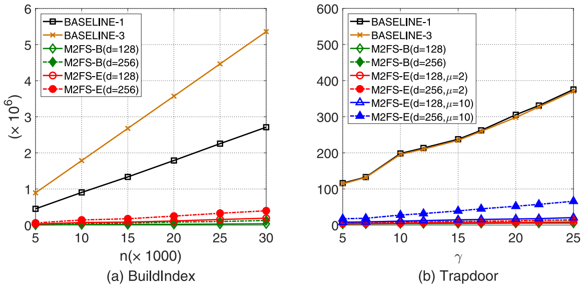
VI. Evaluation

This section will evaluate the performance of our M2FS schemes in terms of execution time and result accuracy. To show their effectiveness, we compare our schemes with the multi-keyword FSE schemes proposed in [19], [20], and [28], (denoted by BASELINE-1, BASELINE-2, and BASELINE-3, respectively), because those schemes also utilize SKNN as a cryptographic tool. As main techniques of their schemes, a Bloom filter of 8,000 entries and a 2-stable $(\sqrt{3}, 2, p_1, p_2)$-locality-sensitive hashing (LSH) family are employed to support 1 difference between keywords. In BASELINE-1, keywords are first transformed into bi-gram vectors that will be mapped into Bloom filters by using LSH functions. Then, SKNN is employed to encrypt Bloom filters, such that the level of matching can be quantified by computing inner product between two encrypted Bloom filters. Compared to BASELINE-1, BASELINE-2 applies a stemming algorithm and a uni-gram-based keyword transformation method to reduce Euclidean distance, and thus is more accurate. Based on BASELINE-2, BASELINE-3 constructs a balanced binary tree to improve search efficiency while keeping the same accuracy with BASELINE-2. Therefore, we conduct comparisons with BASELINE-1 and BASELINE-3 for performance, and conduct comparisons as BASELINE-2 for accuracy, respectively.

A. Parameter Setting

In our M2FS schemes, algorithms *Init*, *BuildIndex*, and *GenTrap* are run by either the data owner or the data user, and the relevant experiments are conducted on a local machine running the Windows 10 Enterprise operating system with an Intel Core i5 CPU running at 3.20 GHz and 16 GB memory. To simulate the *Search* algorithm run by the cloud server, the relevant experiments are tested on a server with two Intel(R) Xeon(R) CPU E5-2620 Processors (2.0 GHz), which has 16 processor cores and supports 16 parallel threads. The programs implemented in Java are evaluated on a real dataset, Enron

$$
\begin{aligned}
v.\mathbf{R}_j[k][l] &= \sum_{y=1}^{\mu} v.\widehat{\mathbf{R}}_j[k][(l-1)\mu + y] \\
&= \sum_{y=1}^{\mu} \sum_{x=1}^{d} v.\mathbf{P}[k][x] \times \widehat{\mathbf{Q}}_j[x][(l-1)\mu + y] \\
&= \left(\sum_{x=1}^{d} v.\mathbf{P}[k][x] \times \widehat{\mathbf{Q}}_j[x][(l-1)\mu + 1] \right) + \cdots + \left(\sum_{x=1}^{d} v.\mathbf{P}[k][x] \times \widehat{\mathbf{Q}}_j[x][l\mu] \right) \\
&= v.\mathbf{P}[k][1]\Big(\widehat{\mathbf{Q}}_j[1][(l-1)\mu + 1] + \cdots + \widehat{\mathbf{Q}}_j[1][l\mu]\Big) + \cdots + v.\mathbf{P}[k][d]\Big(\widehat{\mathbf{Q}}_j[d][(l-1)\mu + 1] + \cdots + \widehat{\mathbf{Q}}_j[d][l\mu]\Big) \\
&= v.\mathbf{P}[k][1]\big(\mathbf{Q}_j[1][l] + \delta_1\big) + \cdots + v.\mathbf{P}[k][d]\big(\mathbf{Q}_j[d][l] + \delta_d\big) \\
&= v.\mathbf{P}[k][1]\big(\mathbf{Q}_j[1][l] + t_1\mathbf{Q}_j[1][l]\big) + \cdots + v.\mathbf{P}[k][d]\big(\mathbf{Q}_j[d][l] + t_d\mathbf{Q}_j[d][l]\big) \\
&= (1 + t_1)v.\mathbf{P}[k][1] \times \mathbf{Q}_j[1][l] + \cdots + (1 + t_d)v.\mathbf{P}[k][d] \times \mathbf{Q}_j[d][l] \\
&= v.\mathbf{P}[k][\,*\,] \star \mathbf{Q}_j[\,*\,][l] + X
\end{aligned} \tag{10}
$$



(a) BuildIndex

(b) Trapdoor

Fig. 3. Comparison of the execution time(*ms*) at data owner/user side. (a) The time for generating an index from *n* files. (b) The time for generating a trapdoor for γ keywords.

Email Data Set,[2] which contains 30109 emails sent by about 150 different users.

To validate the feasibility of our schemes, a file collection F is constructed by randomly selecting $n \in [\![5000, 30000]\!]$ files from this dataset, where each file f_i is associated with $\beta_i \in [\![5, 25]\!]$ keywords. The number of universal keywords is $m \in [\![59706, 449703]\!]$, where the maximal length of keywords is $\alpha = 30$. The data user will query with $\gamma \in [\![5, 25]\!]$ fuzzy keywords, where each fuzzy keyword contains $\varphi \in [\![2, 5]\!]$ symbol "?"s. To generate a fuzzy query, we randomly choose φ characters from a keyword and replace them with symbol "?". For security consideration, the dimension of vectors d is set as $\{128,\ 256\}$ and the number of columns μ in an expansive query matrix is set as $[\![2, d]\!]$. To minimize deviation, each simulation is run at least 100 times to obtain the average value.

B. Efficiency

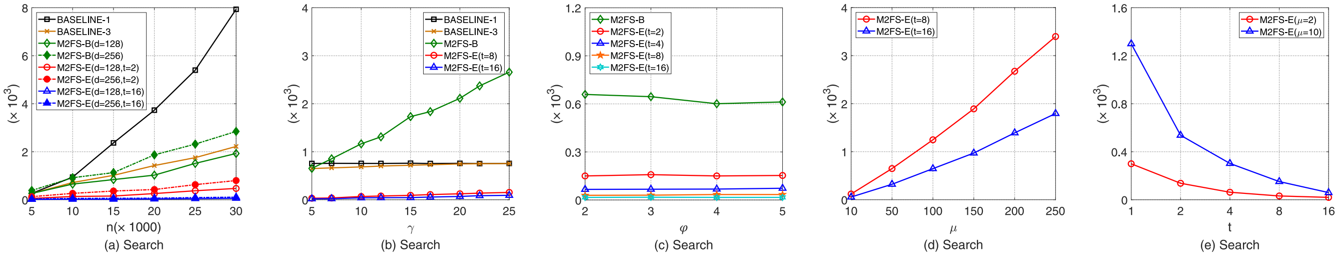
We only consider the most expensive operation in both M2FS-B and M2FS-E, i.e., the calculation of matrix multiplication. Let us start with analyzing the computation costs of M2FS-B. The basic *Init* algorithm generates two invertible matrices $\mathbf{M}_1, \mathbf{M}_2 \in \mathbb{R}^{d \times d}$ as keys, the complexity of which is $O(d^2)$. For file $f_i \in F$ containing β_i keywords, the basic *BuildIndex* algorithm generates a pair of index matrices $\mathbf{P}'_i = (\mathbf{P}'_{i_a}, \mathbf{P}'_{i_b}) \in \mathbb{R}^{\beta_i \times d,}$ resulting in a complexity $O(\beta_i \times d^2)$. Given a collection of *n* files, this algorithm builds a forward index with complexity $O(\sum_{i=1}^{n} \beta_i \times d^2)$. For a query containing γ keywords, the basic *GenTrap* algorithm outputs a pair of query matrices $\mathbf{Q}'_j = (\mathbf{Q}'_{j_a}, \mathbf{Q}'_{j_b}) \in \mathbb{R}^{d \times \gamma}$, the complexity of which is $O(\gamma \times d^2)$. Given $\mathbf{P}'_i$ and $\mathbf{Q}'_j$, the basic *Search* algorithm calculates matrix multiplication to generate a test matrix $\mathbf{R}_{i,j} \in \mathbb{R}^{\beta_i \times \gamma}$ with a complexity $O(\beta_i \times \gamma \times d)$. To find all matched files, this algorithm needs to evaluate the query on each file in *F*, and thus the complexity is $O(\sum_{i=1}^{n} \beta_i \times \gamma \times d)$.

M2FS-E as an improved version differs from M2FS-B in the following aspects. First, the advanced *BuildIndex* algorithm needs to build a KBB tree from a collection of *n* files, the complexity of which is $O(n \times \beta \times d^2)$, where β is the maximal number of keywords associated with files in *F*. Second, the advanced *GenTrap* algorithm outputs a pair of expansive query matrices $\widehat{\mathbf{Q}}'_j = (\widehat{\mathbf{Q}}'_{j_a}, \widehat{\mathbf{Q}}'_{j_b}) \in \mathbb{R}^{d \times (\mu\gamma)}$ for a query containing γ keywords, the complexity of which is $O(\mu \times \gamma \times d^2)$. Finally, the advanced *Search* algorithm traverses the KBB tree-based index and generates an intermediate matrix $v.\widehat{\mathbf{R}}_j \in \mathbb{R}^{\beta \times (\mu\gamma)}$ for each tree node *v*. Therefore, the sequential search time is $O(\log n \times \beta \times \mu \times \gamma \times d)$ and a parallel search can be achieved in $O(\frac{r}{t} \times \log n \times \beta \times \mu \times \gamma \times d)$, where *r* is the number of matched files and *t* is the number of processors. If *t* is sufficiently large, the parallel search time is even less than the optimal sublinear search time *O*(*r*).

Fig. 3-Fig. 5 show the comparison results in terms of efficiency among our M2FS schemes, BASELINE-1, and BASELINE-3. Note that key generation is a one-time cost (about 73ms for $d = 256$ in our schemes and 2842s in BASELINE-1/BASELINE-3, respectively), and thus the related experiment results are omitted in comparisons. For generating encrypted indexes, all schemes apply SKNN for encryption. However, BASELINE-1 needs to calculate LSH functions to generate *n* Bloom filters of 8000 entries, and BASELINE-3 requires additional calculation of (*n* − 1) Bloom filters to construct tree-based index. M2FS-B requires only matrix multiplication to generate *n* index matrices of ($\beta_i \times d$) dimensions, and M2FS-E needs to build a KBB tree in addition compared to M2FS-B. Since *d* is much smaller than the size of Bloom filters, our M2FS schemes are much more efficient in general. For example, in Fig. 3-(a), for building indexes from $n = 25000$ files, BASELINE-1 and BASELINE-3 cost about 2255s and 4465s, respectively, but M2FS-B and M2FS-E (when $d = 128$) incur only about 28s and 155s, respectively. The process of generating trapdoors is similar. As shown in Fig. 3-(b), the trapdoor generation time of BASELINE-1 and BASELINE-3 are almost same duo to the identical procedure, M2FS-B costs the least time, and BASELINE-1 costs the most time. For example, to generate a trapdoor containing $\gamma = 10$ keywords, BASELINE-1 costs about 198 ms, BASELINE-3 costs about 194 ms, but M2FS-B (when $d = 128$) and M2FS-E (when $\mu = 10$ and $d = 128$) incur about 4ms and 11 ms, respectively. However, the larger value of μ, the higher cost of generating a trapdoor in M2FS-E. For example, when $d = 128$, the execution time increases from 7 ms to 17 ms for generating 20 keywords, as μ increases from 2 to 10.

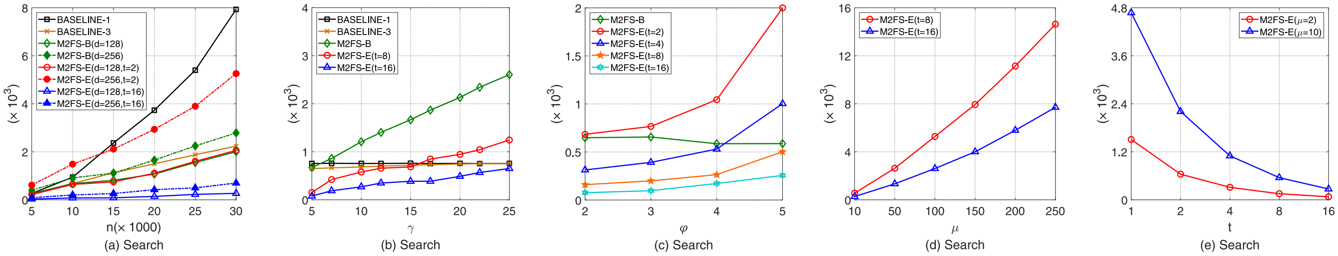
The results of search time for AND queries and OR queries are shown in Fig. 4 and Fig. 5, respectively. For all schemes, the search time is increasing as the number of documents *n* grows. In terms of search speed, M2FS-E supporting parallel processing is faster than M2FS-B when using enough processors. For example, in Fig. 4-(a), when $d = 128$, given $t = 16$ available processors, the execution time for searching $\gamma = 5$ keywords over a collection of $n = 25000$ files in M2FS-B and M2FS-E is 1515 ms and 47 ms, respectively. As for M2FS-E, AND queries will prune more subtrees than OR queries while traversing the KBB tree. Therefore, OR queries consume more execution time under the same conditions. For example, when $d = 128$, $\varphi = 2$, $\mu = 2$ and $t = 16$, the execution time for searching $\gamma = 5$ keywords over a collection of $n = 20000$ files is 33 ms (in AND queries) and 142 ms (in OR queries), respectively. Compared with BASELINE-1, BASELINE-3 that uses a tree-based index incurs less search time. Compared with the baseline schemes, M2FS-B and M2FS-E incur less

[2] https://www.cs.cmu.edu/∼./enron/



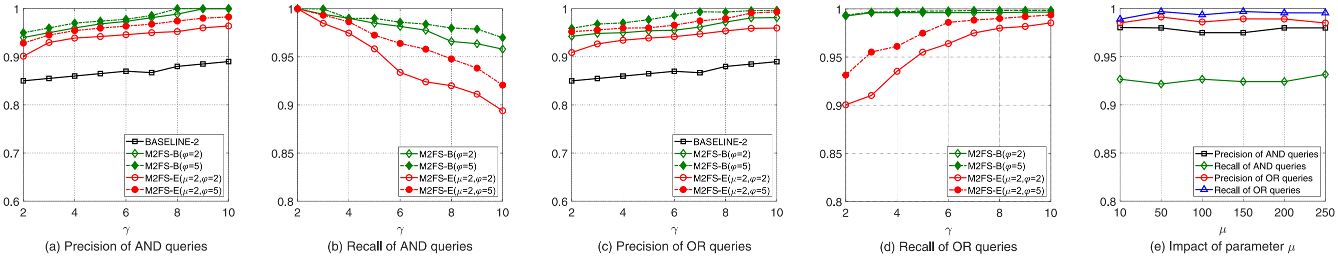
(a) Search (b) Search (c) Search (d) Search (e) Search

Fig. 4. Comparison of the execution time(*ms*) for AND queries at cloud server side. (a) The time for searching n files with fixed values $\gamma = 5$, $\varphi = 2$ and $\mu = 2$. (b) The time for searching γ keywords with fixed values $d = 128$, $n = 10000$, $\varphi = 2$ and $\mu = 2$. (c) The search time of our M2FS schemes under different φ with fixed values $d = 128$, $n = 10000$, $\gamma = 5$ and $\mu = 2$. (d) The search time of M2FS-E under different μ with fixed values $d = 128$, $n = 10000$, $\gamma = 5$, and $\varphi = 2$. (e) The search time of M2FS-E under different t with fixed values $d = 128$, $n = 10000$, $\gamma = 5$ and $\varphi = 2$.



(a) Search (b) Search (c) Search (d) Search (e) Search

Fig. 5. Comparison of the execution time(*ms*) for OR queries at cloud server side. (a) The time for searching n files with fixed values $\gamma = 5$, $\varphi = 2$ and $\mu = 2$. (b) The time for searching γ keywords with fixed values $d = 128$, $n = 10000$, $\varphi = 2$ and $\mu = 2$. (c) The search time of our M2FS schemes under different φ with fixed values $d = 128$, $n = 10000$, $\gamma = 5$ and $\mu = 2$. (d) The search time of M2FS-E under different μ with fixed values $d = 128$, $n = 10000$, $\gamma = 5$ and $\varphi = 2$. (e) The search time of M2FS-E under different t with fixed values $d = 128$, $n = 10000$, $\gamma = 5$ and $\varphi = 2$.



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

Fig. 6. The accuracy of our M2FS schemes. (a) Precision for AND queries of γ keywords. (b) Recall for AND queries of γ keywords. (c) Precision for OR queries of γ keywords. (d) Recall for OR queries of γ keywords. (e)Accuracy impacted by parameter μ with $\gamma = 10$ and $\varphi = 5$.

execution time when the number of query keywords γ is small. For example, in Fig. 4-(b), when $\gamma = 5$, BASELINE-1 (resp. BASELINE-3) costs about 755 ms (resp. 717 ms) to search $n = 10000$ files, while M2FS-B and M2FS-E ($\mu = 2$ and $t = 8$) cost only about 653 ms and 32 ms, respectively. However, the search complexity of M2FS-B and M2FS-E increases as γ increases. For example, in Fig. 5-(b), when $n = 10000$, $\mu = 2$ and $t = 8$, the search cost of M2FS-B (resp. M2FS-E) increases from 673 ms to 2605 ms (resp. from 152 ms to 1245 ms), while γ ranges from 5 to 25. In M2FS-E, the complexity of both AND and OR queries is positively impacted by the values of γ and μ, but negatively impacted by the value of t. Besides, the complexity of OR queries is positively impacted by the values of φ. For example, in Fig. 5-(c), given $n = 10000$ files, when $\mu = 2$, $\gamma = 5$ and $t = 4$, the search time grows from 313 ms to 1003 ms as φ increases from 2 to 5 for OR queries, but the trend of AND queries time is basically unchanged in Fig. 4-(c); given $n = 10000$ files, when $\gamma = 5$ and $t = 16$, the search time of AND queries grows from 56 ms to 1795 ms as μ increases from 5 to 250 in Fig. 4-(d), and when $\gamma = 5$, $\varphi = 2$ and $\mu = 2$, the search time of OR queries decreases from 1508 ms to 76 ms as t increases from 1 to 16 in Fig. 5-(e).

C. Accuracy

The accuracy is measured by the widely used performance metrics, *precision* and *recall*, denoted as ϵ_p and ϵ_r, respectively. Let t_p, f_p, and f_n denote true positive, false positive, and false negative, respectively. We have $\epsilon_p = t_p/(t_p + f_p)$ and $\epsilon_r = t_p/(t_p + f_n)$. Here, if $\vartheta_j \bowtie f_i$ and c_i is in the search results C_{ϑ_j}, a true positive t_p is yielded; if $\vartheta_j \not\bowtie f_i$ and c_i is in the search results C_{ϑ_j}, a false positive f_p is yielded; if $\vartheta_j \bowtie f_i$ and c_i is not in the search results C_{ϑ_j}, a false negative f_n is yielded.

The loss of accuracy is caused by precision loss in the calculation of reciprocals of primes. To determine whether a value is an integer or not, we round up after the z-th decimal point, where $z \in [\![3, 10]\!]$. The reason is that a large portion of fractional numbers, e.g., 3.000459, will be rounded to integers when the value of z is small, and this increases the rate of false positives f_p. When the value of z is large, a large portion

TABLE I
COMPARISON OF MULTI-KEYWORD FSE SCHEMES

	Parallel Search	Flexibility	Search Time	Communication Cost	Storage Cost
BASELINE-1 [19]	×	✓	$O(n \times l)$	$O(l)$	$O(n \times l)$
BASELINE-2 [20]	×	✓	$O(n \times l)$	$O(l)$	$O(n \times l)$
EliMFS-E [26]	×	×	$O(\lvert F(w_1)\rvert \times l)$	$O(l)$	$O(\sum_{i=1}^{m} \lvert F(w_i)\rvert \times l)$
BASELINE-3 [28]	×	✓	$O(r \times \log n \times l)$	$O(l)$	$O(n \times l)$
M2FS-B	×	✓	$O(\sum_{i=1}^{n} \beta_i \times \gamma \times d)$	$O(\gamma \times d)$	$O(\sum_{i=1}^{n} \beta_i \times d)$
M2FS-E	✓	✓	$O(\frac{r}{t} \times \log n \times \beta \times \mu \times \gamma \times d)$	$O(\mu \times \gamma \times d)$	$O(n \times \beta \times d)$

$n = |F|$ is the number of files, $|F(w_i)|$ is the number of files matching a keyword $w_i \in W$, l is the length of Bloom filter, $m = |W|$ is the total number of keywords in W, β_i is the number of keywords for file $f_i \in F$, β is the maximal number of keywords associated with files in F, γ is the number of keywords in a query, d is the dimension of index/query vector of M2FS, ($d \ll l$ shown in our experiments), $\mu \in [\![2, d]\!]$ is a security parameter determining the amount of random noises to be added, r is the number of matched files for the given γ keywords, and t is the number of processors.

of fractional numbers, e.g., 3.000000007, will be rounded to reals, increasing the rate of false negatives f_n. To obtain a reasonable value for z, we evaluate the accuracy of our M2FS schemes under different values of z, and find that $z = 6$ enables the highest accuracy.

With the fixed $z = 6$ and $d = 128$, we evaluate the accuracy of our M2FS schemes under different settings of φ, γ, and μ. From Fig. 6-(e), we know that μ has only a minor impact on the accuracy of M2FS-E. However, φ and γ exert a tremendous influence on the accuracy of our M2FS schemes. In particular, the larger φ, the higher accuracy of AND queries and OR queries. For example, given an AND query containing $\gamma = 5$ keywords, both precision and recall of M2FS-B increase from 96% to 97% and from 98% to 99%, respectively, as φ increases from 2 to 5; given an OR query containing $\gamma = 5$ keywords, when $\mu = 2$, both precision and recall of M2FS-E increase from 93% to 96% and from 95% to 97%, respectively, as φ increases from 2 to 5. The reason is that a larger φ implies less reciprocals of primes in a query matrix, incurring a minor loss of precision.

Furthermore, an increase of γ in AND queries causes an increase of precision or a decrease of recall. For example, when $\varphi = 2$, precision of M2FS-B increases from 94% to 100% and recall decreases from 100% to 96% as γ increases from 2 to 10. This is because a larger γ implies a larger number of columns in the test 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). Besides, M2FS-E may filter out some matched files, this might also lead to a decrease of recall for AND queries. Then, an increase of γ in OR queries causes an increase of precision and recall. For example, when $\mu = 2$ and $\varphi = 5$, both precision and recall of M2FS-E increase from 95% to 99% and from 93% to 99%, respectively, as γ increases from 2 to 10. This is because a larger γ implies a larger number of matched files (a higher t_p).

Meanwhile, we compare the accuracy between our scheme and BASELINE-2 that supports AND/OR queries by changing a threshold value. From the comparison results, we know that our M2FS scheme is much more accurate than BASELINE-2 in precision under different parameter settings. As for the recall rate, BASELINE-2 usually sends back the top-K result files instead of all the relevant files. Therefore, the recall of BASELINE-2 is relatively low, and will not be shown in the comparison results.

VII. RELATED WORK

SE allows a user to retrieve data of interest in a privacy-preserving way, and it has been an active research filed for securing cloud services in recent years. Song *et al.* [2] proposed the first SE scheme, where the search cost grew linearly with the size of the dataset. As a seminal work, Curtmola *et al.* [3] provided a rigorous security definition for SE and built an inverted index to achieve the optimal search time $O(|F(w)|)$, where $|F(w)|$ is the number of files matching a query keyword w. Subsequently, abundant works have been proposed to provide various functionalities [4]–[6], dynamic update [7], [8], and verifiability [9], [10]. While these works provide solutions with different trade-offs among security and performance, most of them only support exact match. In cloud computing, a user may want to retrieve files as accurately as possible even if she is unsure of the exact spelling of a query keyword. Therefore, FSE with the feature of supporting approximate keyword matching is especially important for improving the service quality of cloud computing.

A. Single-Keyword FSE

Li *et al.* [11] proposed the first single-keyword FSE scheme, which quantified the similarity of keywords with edit distance. The shortcoming of their scheme was the large index size which increased exponentially with the edit distance difference. To reduce the index size, Liu *et al.* [12] proposed a dictionary-based fuzzy set construction that limited the scope of fuzzy keywords. Boldyreva and Chenette [13] improved the security of [11] by revealing only pairwise neighbor relationships instead of neighbor sets. However, all these schemes required a predefined dictionary, and thus making it difficult to perform updates on a file collection. The LSH function [37], which returns records within a distance of a given query with a high probability, is a useful tool for fast similarity search. Kuzu *et al.* [14] was the first to apply the LSH function on the Bloom filter [38] to support efficient fuzzy keyword searches. In their scheme, LSH-based Bloom filters were treated as keywords and an encrypted inverted index was constructed for sublinear search time. To reduce space and communication cost, the work in [15], [16] leveraged a set of advanced hash-based algorithms including multiple-choice hashing, cuckoo hashing, and all-pairs LSH functions for compact indexes and trapdoors. Yuan *et al.* [17] improved the security of collision counting LSH functions by hiding frequency of queries.

However, the above FSE schemes support only single-keyword searches, which usually return results in a coarse-grained fashion. Obviously, multi-keyword FSE schemes, especially those supporting multi-semantic queries, can efficiently improve user experience.

B. Multi-Keyword FSE

Chuah and Hu [18] transformed multi-keyword into a single keyword through pre-defined phrases, but rendered the index size to increase with the edit distance. Wang *et al.* [19] encoded keywords as bi-grams and quantified keywords similarity based on Euclidean distance. The multi-keyword fuzzy search function was achieved by forward indexes built based on a collection of LSH-based Bloom filters. To improve search accuracy, Fu *et al.* [20] developed a keyword transformation method so that keywords with the same root could be queried using a stemming algorithm. Wang *et al.* [21] proposed a GPSE scheme that allowed users to query by using generalized wildcard-based string patterns. The construction of [22] is based on homomorphic encryption rather than Bloom filter and hence eliminates the false probability caused by Bloom filter. The main drawback of these schemes [19]–[22] is that the search time is linear with the number of files in the dataset. To improve search efficiency, Moataz *et al.* [23] employed letter orthogonalization to allow testing of string membership by computing inner products. Hahn *et al.* [24] transformed the problem of secure substring search into range queries that could be answered in an efficient way. Shao *et al.* [25] constructed a keyword-based radix tree for fast execution time. Chen *et al.* [26] proposed a two-stage index structure that enabled the search time to be independent of file set size. However, the above schemes [23]–[26] supported only single semantic in sequential searches. To enrich the search patterns, Ding *et al.* [27] proposed a multi-keyword top-*K* similarity search scheme based on the idea of partition. Zhong *et al.* [28] constructed a balanced binary tree on the basis of [20] to support efficient and dynamic top-*K* search. Fu *et al.* [29] exploited multi-dimension index tree structure to implement logic query. Ananthi *et al.* [30] proposed a fuzzy logic-based semantic search. However, the above schemes [27]–[30] never consider the problem of parallel searches. Our previous work [31] constructed a wildcard-based multi-keyword fuzzy search (WMFS) scheme based on the indecomposable property of prime numbers. However, it supported only conjunctive semantic and required the search time to be linear with the number of files. Table I demonstrates the comparison results between our M2FS schemes and the state-of-the-art multi-keyword FSE schemes based on SKNN.

VIII. Conclusion

In this article, we develop M2FS schemes that exploit the indecomposable property of primes to achieve practical multi-keyword fuzzy searches in cloud computing. The proposed M2FS schemes can simultaneously supports AND and OR query semantics, and thus give the data user a great flexibility during the query process. For high efficiency and enhanced robustness, we build an index as a KBB tree and expand a query with random noises to provide parallel searches while resisting linear analysis inherit from SKNN.

However, in the theoretical and experimental results, the search time of our schemes increases with the number of query keywords. The main reason is that each query is encoded as a matrix, the sizes of which grows linearly with the number of keywords. The adoption of matrix-based construction enables flexibility, but causes performance degradation. Therefore, one direction of our future work is to design an efficient FSE scheme with constant-sized query matrix while preserving flexibility. Furthermore, our threat model assumes the mutual trust between the data owner and the data user. This assumption applies only to the case of single data owner, rather than the multi-owner scenario. Therefore, how to remove such an assumption is another research direction. Finally, in view of the research hotspots of SE, we can improve our M2FS schemes from the following aspects: (1) *verifiability* that allows the data user to verify the correctness and integrity of the search results returned by a malicious cloud server; (2) *ranked search* that allows the data user to perform a top-*K* search to retrieve the best-matched files.

Appendix A
Proof of Theorem 1

Proof: We consider a special case, where a file f_i (resp. a query ϑ_j) contains only one keyword, denoted by w_i (resp. $\tilde{w}_j$). The security of our basic M2FS scheme can be easily derived, because the multi-keyword setting is built on top of such a special case. In the known-ciphertext model, the cloud server can only observe the encrypted forms of file set, index, and queries, as well as the search results. Due to the randomness introduced in the index/query vectors, and the random splitting step in the SKNN scheme, index indistinguishability and trapdoor unlinkability are achieved. Therefore, for history $\mathcal{H} = (F, W, \mathbb{Q})$, the trace $Tr(\mathcal{H})$ includes the following information:

- *Size Pattern:* The length of the documents in F.
- *Access Pattern:* The search results $(\mathcal{R}_1, \ldots, \mathcal{R}_t)$, where $\mathcal{R}_j = \{(i, \mathbf{R}_{i,j}) | \vartheta_j \bowtie f_i, i \in [\![n]\!]\}$, for $j \in [\![t]\!]$.

Let Sim denote a simulator that can simulate a view $\bar{\mathcal{V}}$. Our scheme is secure if the cloud server cannot distinguish $\bar{\mathcal{V}}$ from $\mathcal{V}$ given two histories with the same trace. To achieve this, simulator Sim performs the following:

- Simulator Sim runs algorithm $SKNN.GenKey$ to generate $\bar{sk} = \{\bar{\mathbf{M}}_1, \bar{\mathbf{M}}_2, \bar{\mathbf{s}}\}$. It then chooses α random primes $\bar{\mathbb{P}} = \{\bar{p}_1, \ldots, \bar{p}_\alpha\}$, and π random integers $\mathcal{X} = \{\sigma_1, \ldots, \sigma_\pi\}$ where $\pi = 26 + \alpha$ and $1 \leq \sigma_i \leq d$ for $i \in [\![\pi]\!]$. Finally, it sets $\bar{SK} = (\bar{sk}, \bar{\mathbb{P}}, \mathcal{X})$.
- Simulator Sim randomly selects $\bar{f}_i \in \{0,1\}^{|f_i|}$ for $f_i \in F$ and outputs $\bar{C} = \{\bar{c}_1, \ldots, \bar{c}_n\}$ where $|f_i|$ is the bit length of file f_i and $\bar{c}_i$ is the ciphertext of $\bar{f}_i$ encrypted with SKE.
- To generate $\bar{\mathcal{I}}$, simulator Sim generates a $(1 \times d)$-dimensional matrix $\bar{\mathbf{P}}'_i$ for $1 \leq i \leq n$ as follows:

1) It constructs a $(1 \times d)$-dimensional matrix $\bar{\mathbf{P}}_i$ where each element is set to 1.
2) For $l \in [\![\alpha]\!]$, it chooses a random integer σ from $\mathcal{X}$ and sets $\bar{\mathbf{P}}_i[1][\sigma] = \bar{\mathbf{P}}_i[1][\sigma] \times \bar{p}_l$.

3) It randomly chooses $\upsilon \in \llbracket d-\alpha \rrbracket$ elements with a value of 1 from $\bar{\mathbf{P}}_i$ and fills them with random primes outside $\bar{\mathbb{P}}$.

4) It runs the *SKNN.EncI* algorithm to output $\bar{\mathbf{P}}'_i$.

Therefore, $\bar{\mathcal{I}} = \{\bar{\mathbf{P}}'_1, \ldots, \bar{\mathbf{P}}'_n\}$.

- To generate $\bar{\mathbb{T}}$, simulator Sim constructs a $(d \times 1)$-dimensional matrix $\bar{\mathbf{Q}}'_j$ for $1 \leq j \leq t$ as follows:

1) It constructs an empty set *R* and a matrix $\bar{\mathbf{Q}}_j \in \mathbb{R}^{d\times 1}$ where each element is set to 1.

2) For $i \in \llbracket n \rrbracket$, if $\vartheta_j \bowtie f_i$, it puts $\bar{\mathbf{P}}_i$ in set *R*.

3) For $l \in \llbracket d \rrbracket$, it performs as follows: a) It constructs an empty set *Y*. b) For each index matrix $\bar{\mathbf{P}}_i \in R$, if $\bar{\mathbf{P}}_i[1][l] \neq 1$, it puts $\bar{\mathbf{P}}_i[1][l]$ into *Y*. c) If $|Y| > 0$, it sets $\bar{\mathbf{Q}}_j[l][1] = 1/y$ where *y* is the least common multiple of elements in *Y*. d) If $|Y| = 0$, it sets $\bar{\mathbf{Q}}_j[l][1]$ to a random integer.

4) It runs the *SKNN.EncQ* algorithm to output $\bar{\mathbf{Q}}'_j$.

Therefore, $\bar{\mathbb{T}} = \{\bar{\mathbf{Q}}'_1, \ldots, \bar{\mathbf{Q}}'_t\}$.

- Simulator Sim outputs the view $\bar{\mathcal{V}} = (\bar{C}, \bar{\mathcal{I}}, \bar{\mathbb{T}})$. Due to the semantic security of SKE, no PPT adversary can distinguish between $\bar{C}$ and *C*. The indistinguishability of indexes and trapdoors is achieved as follows: First, a set of random primes and a PRF are introduced to build vectors in our construction. Without knowing $\mathcal{P}$ and the secret key of PRF, it is hard for the cloud server to recover the vectors. For example, if all primes are randomly chosen from [1, 10000], then the total number of primes that can be used is $l = 1231$. For $l = 1231$ and $L = 30$, there are $C(l, L) = 2^{199}$ possible ways to construct these primes in brute force attacks. In terms of strength, this is more powerful than 128-bit symmetric keys. Second, SKNN is proven to be COA secure, rendering the encrypted indexes $\bar{\mathcal{I}}$ and the trapdoors $\bar{\mathbb{T}}$ to generate the same trace as the one that the cloud server has. Therefore, we claim that no PPT adversary can distinguish $\bar{\mathcal{V}}$ from $\mathcal{V}$, and that M2FS-B scheme is secure in the known ciphertext model. ■

Appendix B
Proof of Theorem 2

Proof: The purpose of expanding a query matrix is to add random noises to the search results so that Eq. (6) holds with a negligible probability. First, we show the security of the test matrix $v.\mathbf{R}_j$. That is, $v.\mathbf{R}_j[k][l] \neq v.\mathbf{P}[k][\,*\,] \star \mathbf{Q}_j[\,*\,][l]$ for $k \in \llbracket \beta \rrbracket$ and $l \in \llbracket \gamma_j \rrbracket$. From Eq. (10), we know that $v.\mathbf{R}_j[k][l] = v.\mathbf{P}[k][\,*\,] \star \mathbf{Q}_j[\,*\,][l] + X$, where $X \in \mathbb{R}$ is a random number that has no linear relationship with the result of $v.\mathbf{P}[k][\,*\,] \star \mathbf{Q}_j[\,*\,][l]$. Therefore, it is impossible for the adversary to decompose $v.\mathbf{P}[k][*]\star\mathbf{Q}_j[*][l]$ from $v.\mathbf{R}_j[k][l]$. In our advanced M2FS scheme, the *Init* algorithm is constructed in the same way as our basic scheme, and an index tree $\mathcal{T}$ is constructed based on the index matrices output by the basic *BuildIndex* algorithm. Our main security concern is that, given an expansive query matrix $\widehat{\mathbf{Q}}_j$, the advanced *Search* algorithm outputs an intermediate matrix $v.\widehat{\mathbf{R}}_j$, which may leak certain sensitive information.

Then, we show the security of the intermediate matrix. From Eq. (7), we know that $v.\widehat{\mathbf{R}}_j = v.\mathbf{P} \star \widehat{\mathbf{Q}}_j$, where $v.\widehat{\mathbf{R}}_j[k][l] = v.\mathbf{P}[k][\,*\,] \star \widehat{\mathbf{Q}}_j[\,*\,][l]$, for $k \in \llbracket \beta \rrbracket$ and $l \in \llbracket \mu\gamma_j \rrbracket$. In our construction, an expansive query matrix $\widehat{\mathbf{Q}}_j$ is made up of γ_j sub-matrices of $(\beta \times \mu)$ dimensions, denoted by $\widehat{\mathbf{Q}}_j\llbracket 1, \mu \rrbracket, \ldots, \widehat{\mathbf{Q}}_j\llbracket (\gamma_j - 1)\mu + 1, \mu\gamma_j \rrbracket$. With a given value of *l*, each column of the sub-matrix $\widehat{\mathbf{Q}}_j\llbracket (l-1)\mu + 1, l\mu \rrbracket$ contains $(d - \mu + 1)$ query values at most and $(\mu - 1)$ random numbers at least. Although the sum of the random numbers at each row of $\widehat{\mathbf{Q}}_j\llbracket (l-1)\mu + 1, l\mu \rrbracket$ is related to the query value ($\delta_k = t_k\mathbf{Q}_j[k][l]$), the random numbers in each column are independent from both the index and query matrices. Therefore, the adversary cannot infer any useful information from $v.\widehat{\mathbf{R}}_j$ directly. Now, let us consider the adversary constructing linear equations from an arbitrary combination of elements in $v.\widehat{\mathbf{R}}_j$. Given $y < \mu$ elements, the number of unknown variables is larger than the number of linear equations, and thus the adversary cannot solve an equation. Given $y \geq \mu$ elements, the most advantageous attack is to add up μ elements at the *k*-th row to obtain one element in $v.\mathbf{R}_j$. As the security of the test matrix has already been proven, the SKNN scheme with expansive query matrix is secure in linear analyses. ■

Appendix C
Proof of Theorem 3

Proof: In the known-background model, the cloud server may infer certain keyword/trapdoor pairs besides what it can observe in the known-ciphertext model. Therefore, the trace includes a set of keyword/trapdoor pairs in addition to the size pattern and access pattern. In the advanced M2FS scheme, the construction of index/query matrices is based on that of the basic M2FS scheme, and the SKNN scheme with expansive query matrix is proven to be secure in linear analyses. Therefore, we claim that no PPT adversary can distinguish a view simulated by a simulator from a view generated by a real experiment, and that our advanced M2FS scheme is semantically secure in the known background model. ■

References

[1] K. Gu, N. Wu, B. Yin, and W. Jia, "Secure data query framework for cloud and fog computing," *IEEE Trans. Netw. Service Manag.*, vol. 17, no. 1, pp. 332–345, Mar. 2020.

[2] D. X. Song, D. Wagner, and A. Perrig, "Practical techniques for searches on encrypted data," in *Proc. IEEE Symp. Security Privacy (S&P)*, 2000, pp. 44–55.

[3] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky, "Searchable symmetric encryption: Improved definitions and efficient constructions," in *Proc. CCS*, 2006, pp. 79–88.

[4] 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.

[5] Q. Liu, Y. Peng, S. Pei, J. Wu, T. Peng, and G. Wang, "Prime inner product encoding for effective wildcard-based multi-keyword fuzzy search," *IEEE Trans. Services Comput.*, early access, Sep. 1, 2020, doi: 10.1109/TSC.2020.3020688.

[6] 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.

[7] S. Kamara and C. Papamanthou, "Parallel and dynamic searchable symmetric encryption," in *Proc. Int. Conf. Financ. Cryptogr. Data Security (FC)*, 2013, pp. 258–274.

[8] J. Li *et al.*, "Searchable symmetric encryption with forward search privacy," *IEEE Trans. Depend. Secure Comput.*, early access, Jan. 22, 2019, doi: 10.1109/TDSC.2019.2894411.

[9] Q. Liu, Y. Tian, J. Wu, T. Peng, and G. Wang, "Enabling verifiable and dynamic ranked search over outsourced data," *IEEE Trans. Services Comput.*, early access, Jun. 11, 2019, doi: 10.1109/TSC.2019.2922177.

[10] X. Ge *et al.*, "Towards achieving keyword search over dynamic encrypted cloud data with symmetric-key based verification," *IEEE Trans. Depend. Secure Comput.*, early access, Jan. 30, 2019, doi: 10.1109/TDSC.2019.2896258.

[11] J. Li, Q. Wang, C. Wang, N. Cao, K. Ren, and W. Lou, "Fuzzy keyword search over encrypted data in cloud computing," in *Proc. INFOCOM*, 2010, pp. 441–445.

[12] 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. (CCIS)*, 2011, pp. 269–273.

[13] A. Boldyreva and N. Chenette, "Efficient fuzzy search on encrypted data," in *Proc. Int. Workshop Fast Softw. Encryption (FSE)*, 2014, pp. 613–633.

[14] M. Kuzu, M. S. Islam, and M. Kantarcioglu, "Efficient similarity search over encrypted data," in *Proc. IEEE Int. Conf. Data Eng. (ICDE)*, 2012, pp. 1156–1167.

[15] X. Yuan, H. Cui, X. Wang, and C. Wang, "Enabling privacy-assured similarity retrieval over millions of encrypted records," in *Proc. Eur. Symp. Res. Comput. Security (ESORICS)*, 2015, pp. 40–60.

[16] X. Liu, X. Yuan, and C. Wang, "EncSIM: An encrypted similarity search service for distributed high-dimensional datasets," in *Proc. IEEE/ACM 25th Int. Symp. Qual. Service (IWQoS)*, 2017, pp. 1–10.

[17] X. Yuan, X. Wang, C. Wang, C. Yu, and S. Nutanong, "Privacy-preserving similarity joins over encrypted data," *IEEE Trans. Inf. Forensics Security*, vol. 12, no. 11, pp. 2763–2775, Nov. 2017.

[18] M. Chuah and W. Hu, "Privacy-aware bedtree based solution for fuzzy multi-keyword search over encrypted data," in *Proc. ICDCS Workshops*, 2011, pp. 273–281.

[19] B. Wang, S. Yu, W. Lou, and Y. T. Hou, "Privacy-preserving multi-keyword fuzzy search over encrypted data in the cloud," in *Proc. INFOCOM*, 2014, pp. 2112–2120.

[20] 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.

[21] 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. INFOCOM*, 2015, pp. 2101–2109.

[22] Y. Yang, X. Liu, R. H. Deng, and J. Weng, "Flexible wildcard searchable encryption system," *IEEE Trans. Services Comput.*, vol. 13, no. 3, pp. 464–477, May/Jun. 2020.

[23] T. Moataz, I. Ray, I. Ray, A. Shikfa, F. Cuppens, and N. Cuppens, "Substring search over encrypted data," *J. Comput. Security*, vol. 26, no. 1, pp. 1–30, Nov. 2018.

[24] F. Hahn, N. Loza, and F. Kerschbaum, "Practical and secure substring search," in *Proc. SIGMOD*, 2018, pp. 163–176.

[25] J. Shao, R. Lu, Y. Guan, and G. Wei, "Achieve efficient and verifiable conjunctive and fuzzy queries over encrypted data in cloud," *IEEE Trans. Services Comput.*, early access, Jun. 24, 2019, doi: 10.1109/TSC.2019.2924372.

[26] J. Chen *et al.*, "EliMFS: Achieving efficient, leakage-resilient, and multi-keyword fuzzy search on encrypted cloud data," *IEEE Trans. Services Comput.*, vol. 14, no. 6, pp. 1072–1085, Nov./Dec. 2020, doi: 10.1109/TSC.2017.2765323.

[27] X. Ding, P. Liu, and H. Jin, "Privacy-preserving multi-keyword top-k similarity search over encrypted data," *IEEE Trans. Depend. Secure Comput.*, vol. 16, no. 2, pp. 344–357, Mar./Apr. 2019.

[28] H. Zhong, Z. Li, J. Cui, Y. Sun, and L. Liu, "Efficient dynamic multi-keyword fuzzy search over encrypted cloud data," *J. Netw. Comput. Appl.*, vol. 149, Jan. 2020, Art. no. 102469, doi: 10.1016/j.jnca.2019.102469.

[29] S. Fu, Q. Zhang, N. Jia, and M. Xu, "A privacy-preserving fuzzy search scheme supporting logic query over encrypted cloud data," *Mobile Netw. Appl.*, to be published, doi: 10.1007/s11036-019-01493-3.

[30] M. Ananthi, R. Sabitha, S. Karthik, and J. Shanthini, "FSS-SDD: Fuzzy-based semantic search for secure data discovery from outsourced cloud data," *Soft Comput.*, vol. 24, no. 16, pp. 12633–12642, Aug. 2020.

[31] Q. Liu, S. Pei, K. Xie, J. Wu, T. Peng, and G. Wang, "Achieving secure and effective search services in cloud computing," in *Proc. TrustCom*, 2018, pp. 1386–1391.

[32] G. Navarro, "A guided tour to approximate string matching," *ACM Comput. Surveys*, vol. 33, no. 1, pp. 31–88, Mar. 2001.

[33] W. K. Wong, D. W.-l. Cheung, B. Kao, and N. Mamoulis, "Secure kNN computation on encrypted databases," in *Proc. SIGMOD*, 2009, pp. 139–152.

[34] Q. Liu, G. Wang, F. Li, S. Yang, and J. Wu, "Preserving privacy with probabilistic indistinguishability in weighted social networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 5, pp. 1417–1429, May 2017.

[35] Q. Liu, P. Hou, G. Wang, T. Peng, and S. Zhang, "Intelligent route planning on large road networks with efficiency and privacy," *J. Parallel Distrib. Comput.*, vol. 133, pp. 93–106, Nov. 2019.

[36] B. Yao, F. Li, and X. Xiao, "Secure nearest neighbor revisited," in *Proc. Int. Conf. Data Eng. (ICDE)*, 2013, pp. 733–744.

[37] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni, "Locality-sensitive hashing scheme based on p-stable distributions," in *Proc. SoCG*, 2004, pp. 253–262.

[38] 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.

Qin Liu (Member, IEEE) received the B.Sc. degree in computer science from Hunan Normal University, China, in 2004, the M.Sc. degree in computer science in 2007, and the Ph.D. degree in computer science from Central South University, China, in 2012. She has been a visiting student with Temple University, USA. She is currently an Associate Professor with the College of Computer Science and Electronic Engineering, Hunan University, China. Her research interests include security and privacy issues in cloud computing.

Yu Peng (Graduate Student Member, IEEE) received the B.Sc. degree in software engineering from China West Normal University, China, in 2018. He is currently pursuing the Ph.D. degree with the College of Computer Science and Electronic Engineering, Hunan University, China. His research interests include security and privacy issues in cloud computing. He is a Student Member of the China Computer Federation.

Jie Wu (Fellow, IEEE) was a Program Director with the National Science Foundation and a Distinguished Professor with Florida Atlantic University. He is the Chair and a Laura H. Carnell Professor with the Department of Computer and Information Sciences, Temple University, Philadelphia, PA, USA. 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 was a recipient of the 2011 China Computer Federation (CCF) Overseas Outstanding Achievement Award. He serves on several editorial boards, including IEEE TRANSACTIONS ON SERVICE COMPUTING and *Journal of Parallel and Distributed Computing*. He was the General Co-Chair/Chair for IEEE MASS 2006, IEEE IPDPS 2008, IEEE ICDCS 2013, and ACM MobiHoc 2014, as well as the 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. He is a CCF Distinguished Speaker.

Tian Wang (Member, IEEE) received the B.Sc. and M.Sc. degrees in computer science from Central South University in 2004 and 2007, respectively, and the Ph.D. degree from the City University of Hong Kong in 2011. He is currently a joint Professor with the Artificial Intelligence and Future Networks, Beijing Normal University snd UIC and Huaqiao University, China. His research interests include the Internet of Things, edge computing, and artificial intelligence.

Guojun Wang (Member, IEEE) received the B.Sc. degree in geophysics, the M.Sc. degree in computer science, and the Ph.D. degree in computer science from Central South University, 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 the Vice Dean of the School of Computer Science and Cyber Engineering, Guangzhou University, China, and the Director of Institute of Computer Networks, Guangzhou University. He has been listed in Chinese Most Cited Researchers (Computer Science) by Elsevier 2014 to 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, a Member of ACM and IEICE.