Secure Multi-Character Searchable Encryption Supporting Rich Search Functionalities

Qing Wang, Donghui Hu, *Member, IEEE*, Meng Li, *Senior Member, IEEE*, Yan Qiao, *Member, IEEE*, Guomin Yang, *Senior Member, IEEE*, and Mauro Conti, *Fellow, IEEE*

***Abstract*—Wildcard Keyword Searchable Encryption (WKSE) has grown into a ubiquitous tool. It enables clients to search desired files with wildcard expressions. Although promising, previous schemes confront three barriers: (1) An adversary can launch a correlation attack to acquire the similarity between keywords. (2) The WKSE schemes exhibit false positives which can lead to wrong search results. (3) Existing feature extraction strategies limit the flexibility of search expressions. In this paper, we propose a Multi-Character Searchable Encryption scheme (MCSE) that overcomes the aforementioned barriers. To resist correlation attacks, we design the randomize-pad model to encrypt the vector. To eradicate false positives, we apply the vector space model and complete feature extraction strategies so that a feature set uniquely identifies a keyword or expression. To enhance search flexibility, we introduce three distinct feature extraction strategies for keyword expressions, wildcard expressions, and logical expressions, enabling effective multi-character search. These strategies enable indexes to accommodate the search of diverse expressions. Finally, we prove that MCSE is indistinguishable against chosen-feature attacks and implement MCSE on two real datasets. Compared with state-of-the-art schemes, the experiment results show that MCSE achieves good performance.**

***Index Terms*—Searchable encryption, wildcard expression, logical expression, correlation attack, feature extraction strategy.**

Received 10 September 2024; revised 27 October 2025; accepted 26 December 2025. Date of publication 5 January 2026; date of current version 13 February 2026. This work was supported in part by the National Natural Science Foundation of China (NSFC) under Grant U23A20303, Grant 62372149, and Grant 62572168, in part by the Anhui Provincial Natural Science Foundation under Grant 2508085MF151, in part by the China Scholarship Council (CSC), and in part by the Key Laboratory of Knowledge Engineering with Big Data (the Ministry of Education of China) under Grant BigKEOpen2025-04. The work of Guomin Yang was supported by the Lee Kong Chian Fellowship awarded by Singapore Management University. Recommended for acceptance by Z. Wang. *(Corresponding authors: Donghui Hu; Meng Li.)*

Qing Wang and Donghui Hu are with the School of Computer Science and Information Engineering, Hefei University of Technology, Hefei 230601, China (e-mail: qingwang@mail.hfut.edu.cn; hudh@hfut.edu.cn).

Meng Li and Yan Qiao are with the School of Computer Science and Information Engineering, Hefei University of Technology, Hefei 230601, China, also with the State Key Laboratory of Networking and Switching Technology (Beijing University of Posts and Telecommunications), Beijing 100876, China, and also with the Key Laboratory of Knowledge Engineering with Big Data (Hefei University of Technology), Ministry of Education, Hefei 230002, China (e-mail: mengli@hfut.edu.cn; qiaoyan@hfut.edu.cn).

Guomin Yang is with the School of Computing and Information Systems, Singapore Management University, Singapore 188065 (e-mail: gmyang@smu.edu.sg).

Mauro Conti is with the Department of Mathematics and HIT Research Center, University of Padova, 35122 Padova, Italy, and also with Örebro University, 701 82 Örebro, Sweden (e-mail: mauro.conti@unipd.it).

Digital Object Identifier 10.1109/TKDE.2025.3650082

I. Introduction

CLOUD storage [1], [2] provides massive storage and elastic computation services. To protect data privacy and ensure data availability, Searchable Encryption (SE) [3] is proposed, which is a keyword search technique that enables the cloud server to search on encrypted data without decryption. Most existing SE schemes [4], [5], [6], [7], [8] focus on the indexes built by complete keywords. However, the user may issue a query that contains wildcards. To solve the issue, recent works [9], [10] have explored Wildcard Keyword Searchable Encryption (WKSE) technology that splits each keyword into multiple characters and maps them to a Bloom Filter. With the technology, users can use search expressions that contain wildcards to search over encrypted data. The common wildcards have two types: single-character wildcard "?" and multi-character wildcard "$*$". "?" denotes any individual character, while "$*$" signifies any sequence of characters.

Most WKSE schemes [11], [12] based on Bloom Filters only offer weak security, where adversaries can launch correlation attacks [13], [14]. The specific details are as follows: the data owner splits a keyword into multiple characters and maps them into a Bloom Filter. Since the same method is used for all keywords, the identical characters are in the same position in the Bloom Filters corresponding to different keywords. Adversaries can infer the similarity of keywords by comparing the overlapping parts of the Bloom Filters. The correlation attacks lead to data breaches when the adversary has background knowledge. To resist correlation attacks, Bosch et al. [15] used a pseudo-random generator to encrypt each bit of each Bloom Filter. However, this method requires users to possess significant computational power. Another solution [16] also introduced an unkeyed hash function to encrypt the index. But it causes correlation attacks when the data user performs multiple searches. SPWSE [17] and SFWQ [18] used inner product encryption and key aggregation encryption to resist correlation attacks, but they both leak the wildcard positions.

False positives need to be eliminated in WKSE. WKSE schemes [10], [16] extracted feature sets for keywords and expressions to build indexes and generate trapdoors. However, incomplete feature extraction strategies can lead to the feature set of one keyword or expression may be the feature set of another keyword or expression. For example, in SSE-MCW [12], the feature set of $a*b*c$ is the same as the feature set of $a*c$. Therefore, incomplete feature extraction strategies can result in false positives. Another reason for false positives in WKSE

is the utilization of cryptographic tools. A Bloom Filter [19], [20] is a common cryptographic tool that can efficiently check whether an element is within a set or not, but it inherently has false positives due to hash collision. The presence of false positives in search results can lead to serious misunderstandings and erroneous judgments. Yang et al. [21] achieved wildcard expression search by slicing the ciphertext to have no false positive, but this requires multiple rounds of interaction between two non-colluding servers. Wang et al. [18] adopted the interval matching and Li et al. [17] leveraged inner product encryption to achieve WKSE without false positive. However, the trapdoor will leak the positions where wildcards appear in the expression.

To further accommodate the practical necessity of users, search expressions need support "and", "or" and "not" operations between characters. The expressions can be viewed as logical expressions. The "or" operation means that one of the provided characters can appear in a specific position. We use "()" to represent "or". For example, (abc) requires any one of a, b and c to be in the first position. The "not" operation denotes that the provided characters cannot appear at a specific position. We use "[]" to represent "not". For example, $[abc]$ requires a, b, and c should not appear in the first position. The "and" operation indicates that the specified character appears in each position. The "and" operation needs not to be symbolized. For example, $a(abd)$ requires a to be in the first position while any one of a, b, and d should be in the second position. To further emphasize the importance of logical expression, we consider the following banking system. The bank classifies all clients into **r**etail clients, **p**rofessional clients, or **e**ligible counterparties to provide different investment services. For management purposes, the bank extracts a label for each client. The first position of the label represents the category that the client belongs to and the second position of the label represents the type of business the client is engaged in. One day, the bank needs to find clients are retail clients or professional clients and they specialize in the automotive industry. The corresponding logical expression is $(rp)a$. To the best of our knowledge, WKSE is unable to address the problem beyond the naive method, which requires the user to supply two search expressions, denoted as ra and pa, for the search process.

In summary, the existing WKSE schemes suffer from the following issues: (a) The correlation attacks for WKSE can compromise the similarity of keywords. (b) Incomplete feature extraction strategies and natural properties of Bloom Filters may causes false positives in the search results. (c) Logical expressions are mostly used for pattern matching with strings, but most SE works do not consider logical expressions search.

To solve the above issues, we need to address three technical challenges: **Challenge 1: High efficiency while resistant to correlation attacks.** The adversary can obtain keyword similarities from unencrypted Bloom Filters. Encrypting Bloom Filters using traditional encryption schemes incurs significant communication and computation costs for clients, as the server must return the encrypted Bloom Filters to the client for local decryption. Similarly, homomorphic encryption for Bloom Filters results in substantial computation costs. Thus, designing keyword encrypted search schemes that are both resistant to correlation attacks and efficient in the search process is a challenge. **Challenge 2: Complete feature extraction strategies.** Characters, character positions, relative positions, and n-grams can be keyword features. It appears that as the number of features extracted increases, the probability of false positives decreases. Even so, it cannot eliminate the errors completely. If the feature set does not uniquely correspond to a search expression, then multiple distinct search expressions may share an identical feature set. Consequently, the search results will be a combination of the results of all of these search expressions, introducing false positives to any one of them. Thus, the challenge is to develop a complete feature extraction strategy that enables the feature set to be unique to keywords or expressions. **Challenge 3: Flexible feature extraction strategies.** Feature extraction strategies for wildcard expressions fail to ensure proper execution of logical expressions, because a wildcard can represent any character while logical operators specify concrete characters that either occur or do not occur at specific positions. Alternatively, we extract all characters within "()" for the "or" operation, then the feature set of the logical expression will be larger than the feature set of the keyword. If the feature set of a keyword is included in the feature set of a logical expression to consider that the two match, then the keyword that does not match can also be in the search results. Thus, it is challenging to extract features for logical expressions and achieve correct matches.

To address these technical challenges, we make the following contributions:

Security: To resist correlation attacks and safeguard keyword confidentiality, an index encryption phase is introduced. The randomized-pad mode is proposed, which randomizes vectors before padding. It ensures that computation results for the same index but different trapdoors do not produce the same variable. In addition, this method requires only a single additional dimension for padding, reducing computation cost. During the index-building process, features are extracted from keywords and encrypted into indexes using the randomized-pad mode and invertible matrices, ensuring that keyword information remains confidential.

Accuracy: To avoid false positives, we design complete feature extraction. We extract each character along with its precise positional information and the length of the expression. For wildcard expressions, we also extract the length of the desired keyword. These features comprising character, position and length information ensure that each keyword or expression is uniquely represented. In addition, we apply vector space models to replace Bloom Filters and complete feature extraction strategies to eradicate false positives. The vector space model guarantees that a vector component is 1 provided that the feature set must contain the feature corresponding to that component.

Functionality: To support multi-character search, we propose feature extraction strategies corresponding to keyword, wildcard, and logical expressions. For keyword and wildcard expressions, we extract characters, positions, and length features. The size of the feature set is viewed as an indicator, which is used as a reference for matching. For logical expressions, we consider each operation ("or" and "not") as a unit. Within a unit, all characters are assigned the same positional value. Then

TABLE I
COMPARISON OF FUNCTIONALITIES WITH EXISTING WORKS

Property	Keyword expression	Wildcard expression	Logical expression	No false positive	Resistance to correlation attacks	Not revealing wildcard positions
SKS [10]	✓	✓				✓
SSE-MCW [22]	✓	✓				✓
EWFSS [11]	✓	✓				✓
SPWSE [17]	✓	✓		✓	✓	
SFWQ [18]	✓	✓		✓	✓	
TSEWS [23]	✓	✓		✓	✓	✓
MCSE	✓	✓	✓	✓	✓	✓

we extract the characters, positions, and length features. The length of the logical expression plus one indicator is used as a reference for each matching.

To summarize, we propose a multi-character searchable encryption (MCSE). We compare MCSE with existing works in Table I.

- MCSE resists correlation attacks and hides wildcard positions. By security proof, MCSE is secure under chosen-feature attacks which catch correlation attacks.
- MCSE has no false positives. We design complete feature extraction strategies and apply vector space models so that can any vector to determine a keyword or an expression uniquely.
- MCSE supports keyword, wildcard, and logical expression searches. We design three feature extraction strategies. These strategies enable indexes to accommodate the search of diverse expressions.
- MCSE achieves significant better performance and so on. Compared with the state-of-the-art schemes, the experiment results show that MCSE brings a saving of approximately 99.87% in the search process.

The paper is organized as outlined below. We review the related work in Section II. Section III formalizes the problem. Section IV introduces the necessary preliminaries. In Section V, we present the details of E-MRSE. In section VI, we propose a multi-character searchable encryption scheme that adopts the encryption method in E-MRSE, followed by the security analysis in Section VII and performance evaluation in Section VIII, respectively. We discuss about scalability, security, efficiency, and flexibility of MCSE in Section IX. Finally, we conclude the paper in Section X.

II. RELATED WORK

In this section, we describe related works on Wildcard Keyword Searchable Encryption.

A. Symmetric Searchable Encryption

Song et al. [3] introduced Symmetric Searchable Encryption (SSE) in 2000 and proposed schemes with provable secrecy, query isolation, controlled search, hidden queries, and feasibility. However, the underlying plaintext distribution is vulnerable to statistical attacks. Goh et al. [13] defined a security model for SSE known as semantic security against adaptive chosen keyword attack (IND-CKA) and proposed an efficient SSE scheme based on Bloom Filter (BF) that proved secure under the IND-CKA model. Considering the information leaked by trapdoors, Curtmola et al. [24] introduced game-based and simulation-based definitions to protect the privacy of indexes and trapdoors and formalized the keyword-file index structure, a landmark work in the field of SSE. To date, more researchers have devoted to improving the performance and enhancing the functionality of SSE [25], [26], [27], [28].

B. Secure K-Nearest Neighbor Computation

Wong et al. [29] proposed a k-nearest neighbor (knn) computation technology that searches for k points on encrypted data that are the nearest to a given query point. They proposed two secure knn schemes based on asymmetric scalar product preserving encryption. The schemes have been enormously impactful, engendering numerous applications [30], [31], [32], [33].

Cao et al. [34] proposed Multi-keyword Ranked Searchable Encryption (MRSE) based on knn. For security, MRSE-2 was padded with more dummy keywords. Xia et al. [35] transformed the index into a special tree structure to achieve sub-linear search and support dynamic updates of documents. Chen et al. [36] considered the similarity between files and proposed a k-means based hierarchical clustering method to speed up the search phase. The method clusters files based on a minimum relevance threshold and then divides the resulting clusters into sub-clusters until all sub-clusters do not exceed the pre-specified cluster size. Fuzzy keyword search technology allows users to provide query keywords with minor misspellings or inconsistencies in the form of stored keywords. Li et al. [37] used edit distance to quantify keyword similarity to support fuzzy keyword search. [38], [39] solved the problem of fuzzy keyword search ranking. To enable the correctness and completeness of search results, Sun et al. [40] proposed a verifiable multi-keyword ranked search upon the index tree structure. All these schemes require padding dummy keywords for security.

C. Wildcard Keyword Searchable Encryption

Existing works achieve WKSE based on several techniques: hidden vector encryption, homomorphic encryption, inner product encryption, and Bloom Filter. Fuzzy keyword search technology [32] appears to offer a potential solution for implementing wildcard expression searches. However, the index generation process depends on the edit distance, so search results may be incomplete.

Hidden vector encryption technology [9], [41], [42] can be transformed into SE that support wildcard expressions search.

For example, Sedghi et al. [9] proposed a public-key hidden vector encryption scheme. The data owner encrypts a public message using the keyword. The trapdoor sent to the server is actually a decryption key derived from a query expression. If the server decrypts the ciphertext to obtain the public message, then the two keywords are the same except for the wildcard positions. But the general drawback of this approach is that the data user needs to provide the server with locations where wildcards appear in the keyword.

The appearance of homomorphic encryption [21], [43], [44] has attracted the attention of more researchers. For example, Yasuda et al. [44] computed the sum of multiple modified inner products between two vectors and determined whether it is 0. Saha et al. [45] split the expression from the location that wildcard appears to support repetitive wildcards. Kim et al. [46] proposed a pattern-matching algorithm that takes an encrypted string and an encrypted pattern along with auxiliary encryptions as inputs and returns an encryption of 1 or 0. The main application is searching for DNA sequences in a genome database. Yang et al. [21] proposed a wildcard search scheme in multi-user scenarios based on homomorphic encryption, but it is under the dual-server model.

Inner product encryption is a type of functional encryption in which the ciphertext and the key each correspond to a vector. With the correct key, the decryption result is the inner product of the two vectors. Li et al. [17] transformed the keyword and the wildcard keyword into vectors and leveraged inner product encryption to process these vectors. If the two match, the inner product is 0. However, the search expression cannot contain multi-character wildcards. Wang et al. [18] adopted the interval matching method instead of position matching to support multi-character wildcard keyword search, but the user has to provide the locations where wildcards appear in the keyword.

The prevailing approach for wildcard expressions search in symmetric searchable encryption is based on Bloom Filter. The first method [15], [22] to support wildcard expressions search is to transform a wildcard search into a lookup for an exact match. For example, Bosch et al. [15] proposed a masked index scheme that can resist the correlation attack. However, the encrypted index is not available for the server, and users have to decrypt themselves to match. Hu et al. [22] considered dynamic operations on the documents. The data owners in these schemes are obliged to pre-process the keywords and insert not only the keyword to the index but all wildcard versions of each keyword. So the search is not flexible, the data user can only search for wildcard expressions in the index. Another flexible method is to compare the characters of the keywords in the index and the trapdoor. Suga et al. [10] proposed a wildcard keyword searchable encryption scheme. During index building phase, the data owner extracts the characters of the keywords, concatenates their positions, and incorporates these features into BF. Trapdoors are also generated in this way based on wildcard expressions in addition to hopping over wildcards. So if a keyword in the index matches the wildcard expression, the index is 1 in all positions where the trapdoor is 1. Hu et al. [12] extracted the character in keyword concatenated its inverted position additionally to support expressions containing "$*$".

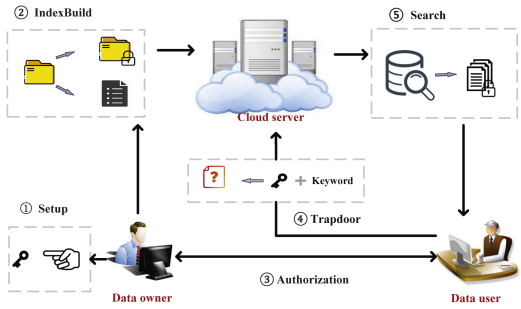


Fig. 1. System model.

Zhao et al. [16] considered the correlation attack derived from unencrypted BF and introduced an unkeyed hash function to xor the unencrypted BF but it is also vulnerable to correlation attacks. Based on [16], Weener et al. [47] proposed a framework of backward private WKSE, but did not provide a concrete scheme. Hua et al. [11] proposed an enhanced wildcard-based fuzzy search scheme that extends functionality to client-proxy-database frameworks like CryptDB [48], but the scheme did not consider correlation attacks.

In summary, prevailing-based WKSE schemes either cannot resist correlation attacks or users need to perform decryption operations. In addition, none of the WKSE schemes support logical expression searches.

III. Problem Statement

In this section, we introduce the system model, security model, and design objectives of MCSE.

A. System Model

As illustrated in Fig. 1, the system model of MCSE includes three parties: the cloud server, the data owner, and the data user.

Cloud server (CS): The cloud server has huge storage space and can provide efficient and convenient data services. In our system model, we consider the server to be honest-but-curious. It may try to learn any useful information about files and keywords from indexes, trapdoors, and similar inner products, but it cannot deviate from the protocol.

Data owner (DO): The data owner, who can be an individual or an enterprise, is responsible for building indexes. Also, he/she can search for files and authorize other users to search.

Data user (DU): The data user can search for files of interest by generating trapdoors with keys provided by the data owner.

MCSE system is composed of four algorithms, namely Setup, IndexBuild, Trapdoor and Search. In Fig. 1, step 1 is Setup algorithm where the DO chooses encryption keys. For step 2, the DO extracts keywords from files, builds indexes based on the keyword-file relationship and uploads the encrypted files and indexes to the cloud server. When a DU wants to search for files containing desired search expressions, he/she first sends a search query to the DO. Upon receiving the request, the DO, if granting authorization, transmits the key to the DU via a secure channel. The DU then uses this key to generate the trapdoor and sends it to the cloud server at step 4. Once the server receives

a trapdoor from the DU, it searches the index with the trapdoor and returns matched files at step 5. We focus on the index for queries, which is deemed as the "structure-only" SSE [49], [50].

B. Security Model

In this system, we consider that the cloud server is honest-but-curious which has been widely adopted in searchable encryption [51], [52], [53]. It assumes that the server follows the scheme algorithms honestly but attempts to learn about keywords and documents with the help of messages transmitted by the data user.

We define the security model that indistinguishability against chosen-feature attacks (IND-CFA) for multi-character searchable encryption based on IND-CKA [13]. The indistinguishability security is modeled by a game played by a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$. During their interaction, $\mathcal{C}$ simulates an MCSE scheme Π, which $\mathcal{A}$ tries to break.

Chosen-feature attacks capture the ability of an adversary to exercise control over what indexes are built by data owners and what expression data users query. First, $\mathcal{C}$ generates a key set $\mathcal{K}$ and keeps it secret. $\mathcal{A}$ makes sub-index queries on keywords by his option and then outputs two distinct keywords w_0, w_1 to be challenged. $\mathcal{C}$ generates a challenge sub-index I on a randomly chosen keyword from w_0, w_1. After that, $\mathcal{A}$ can also adaptively make sub-index queries on any keyword and make trapdoor queries on expressions that do not contain character features that are in w_0 but not in w_1 or character features that are in w_1 but not in w_0. Finally, $\mathcal{A}$ outputs a guess for the challenge sub-index. Formally, the IND-CFA security model is described as follows.

Setup: Given a security parameter, the challenger $\mathcal{C}$ generates the key set $\mathcal{K}$.

Query1: $\mathcal{A}$ chooses keyword adaptively and issues sub-index queries and trapdoor queries.

1) For sub-index queries, $\mathcal{C}$ runs $\mathsf{IndexBuild}$ to generate a sub-index and return it to $\mathcal{A}$.
2) For trapdoor queries, $\mathcal{C}$ runs $\mathsf{Trapdoor}$ to generate a trapdoor and return it to $\mathcal{A}$.

Challenge: $\mathcal{A}$ outputs two distinct keywords w_0 and w_1 under the restriction that the expression W has not been queried where $W[i] = w_0[i]$ or $w_1[i]$ but $w_0[i] \neq w_1[i]$. $\mathcal{C}$ randomly chooses $b \in \{0, 1\}$ and runs $\mathsf{IndexBuild}$ on V_b to get the index I, which is given to $\mathcal{A}$. We refer to I as the challenge sub-index.

Query2: $\mathcal{A}$ issues sub-index queries and trapdoor queries.

1) For a sub-index query, $\mathcal{C}$ responds to sub-index queries in the same way as in **Query1** without restriction.
2) For a trapdoor query, $\mathcal{C}$ runs $\mathsf{Trapdoor}$ to generates a trapdoor and returns it to $\mathcal{A}$. But the expressions do not contain character features that are in w_0 but not in w_1 or character features that are in w_1 but not in w_0.

Guess: $\mathcal{A}$ outputs a bit b' as its guess for b.

Since the adversary adaptively issues sub-index queries for keywords during phase (1) of **Query2**, chosen-feature attacks catch correlation attacks.

Definition 3.1. A multi-character searchable encryption scheme $\Pi = (\mathsf{Setup}, \mathsf{IndexBuild}, \mathsf{Trapdoor}, \mathsf{Search})$ is IND-CFA secure, if there exists no probabilistic polynomial-time adversary who can win the above game with non-negligible advantage over random guess after it makes trapdoor queries and sub-index queries.

TABLE II
KEY NOTATIONS

FL	feature list
$\|FL\|$	the size of FL
ML	length of the longest keyword
w	keyword (expression)
$w[i]$	the $i+1$-th character of w
$\|\|$	concatenation
$?$	single-character wildcard
$*$	multi-character wildcard
(abc)	a, b, or c
$[abc]$	not a, not b and not c
ir	indicator
$\mathcal{W}$	keyword set
$\mathcal{D}$	file set
D_i	the i-th file
w_i	the i-th keyword
V_i	the initial vector of D_i (w_i) in E-MRSE (MCSE)
$V[i]$	the $i+1$-th entry of V
$\hat{V}$	processed vector
$E_1(V)$	encrypted vector
M_1, M_2	invertible matrix

C. Design Objectives

We have three design objectives, namely privacy, functionality, and efficiency.

- *Functionality:* The designed scheme needs to support keyword expression, wildcard expression, and Logical expression searches.
- *Privacy:* (1) Index privacy. The server cannot learn the keyword corresponding to the sub-index. (2) Trapdoor privacy. The server cannot obtain the information of expression from trapdoors. (3) Resist correlation attacks. The server cannot recover indexes even if it captures some feature-trapdoor pairs.
- *Efficiency:* The designed scheme should be efficient in terms of computational and communication costs. Specifically, the search protocol requires only a single round of interaction between the user and the server. Furthermore, there are no complex computations on the user side.

IV. PRELIMINARIES

In this section, we introduce key notations and review the multi-keyword ranked searchable encryption scheme MRSE-1.

A. Notations

This paper utilizes several notations, which are systematically presented in Table II.

B. MRSE-1

Wong et al. [29] proposed a secure knn technique in encrypted databases, which can compare the Euclidean distance between two points in the database and the query point in privacy and select the k nearest neighbor points to the query point. Traditional knn encryption techniques have been used in numerous

searchable encryption schemes for multi-keyword search. We first briefly review the algorithms of these schemes.

Suppose the file set as $\mathcal{D} = \{D_1, \ldots, D_n\}$, the keyword set as $\mathcal{W} = \{w_1, \ldots, w_m\}$. The secret key consists of a bit-vector S of length $m+2$ and two invertible matrices M_1, M_2 of dimension $m+2$. The data owner generates a file vector V_i for each file in $\mathcal{D}$. If file D_i contains keyword w_j, set the j-th bit of V_i as 1. Otherwise, set it as 0. For each file vector V_i, it needs to extend to $(m+2)$-dimensional vector $\hat{V}_i$, where the $(m+1)$-th entry is set to a random value ε_i and the $(m+2)$-th entry is set to 1. According to S, $\hat{V}_i$ is split into two vectors $\hat{V}_{i,1}, \hat{V}_{i,2}$. If the j-th bit of S is 0, $\hat{V}_{i,1}[j], \hat{V}_{i,2}[j]$ are set as the same as $\hat{V}_i[j]$. Otherwise, set $\hat{V}_{i,1}[j], \hat{V}_{i,2}[j]$ to random numbers that satisfy $\hat{V}_{i,1}[j] + \hat{V}_{i,2}[j] = \hat{V}_i[j]$. At last, the file vector is encrypted as $E_1(V_i) = \{M_1^T \hat{V}_{i,1}, M_2^T \hat{V}_{i,2}\}$.

The data user generates a query vector Q for the keywords of interest. The query vector is generated in the same way as the file vector, i.e. the corresponding entry of the keywords of interest in the vector is set to 1 and the other entries are set to 0. And Q also needs to be extended by 2 dimensions to obtain $\hat{Q}$, where the $(m+2)$-th entry is set to a random value t, the $(m+1)$-th entry is set to a random number r, and the first m entries are multiplied by r. Accordingly, the data user divides $\hat{Q}$ into two vectors $\hat{Q}_1, \hat{Q}_2$. Here, the rules for splitting vectors are the reversal of those for file vectors. If the i-th bit of S is 0, $\hat{Q}_1[i], \hat{Q}_2[i]$ are set to random numbers that satisfy $\hat{Q}_1[i] + \hat{Q}_2[i] = \hat{Q}[i]$. Otherwise, $\hat{Q}_1[i], \hat{Q}_2[i]$ are set as the same as $\hat{Q}[i]$. At last, the trapdoor $E_2(Q)$ is generated by $\{M_1^{-1}\hat{Q}_1, \hat{M}_2^{-1}Q_2\}$.

During the query phase, the server calculates (1) for $i = 1, \ldots, n$, and returns the top-k files. $r(V_i \cdot Q + \varepsilon_i) + t$ preserves the inner product between the file vector and query vector and scale relationship for two queries on the same keyword.

$$\begin{aligned} & E_1(V_i) \cdot E_2(Q) \\ &= (M_1^T \hat{V}_{i,1}, M_2^T \hat{V}_{i,2}) \cdot (M_1^{-1}\hat{Q}_1, M_2^{-1}\hat{Q}_2) \\ &= \hat{V}_{i,1} \cdot \hat{Q}^1 + \hat{V}_{i,2} \cdot \hat{Q}_2 \\ &= (V_i, \varepsilon_i, 1) \cdot (rQ, r, t) \\ &= r(V_i \cdot Q + \varepsilon_i) + t \end{aligned} \tag{1}$$

Since similar inner products computed by an index I_i and different trapdoors all contain the same variable ε_i, Cao et al. [34] identified that the MRSE-1 scheme is susceptible to scale analysis attacks. To address this vulnerability, they proposed MRSE-2 which adds $U+1$ dummy entries into the vector. So MRSE-2 brings more computation overhead.

V. E-MRSE

In this section, we propose E-MRSE which can resist scale analysis attacks. The vector processes of MRSE-1, MRSE-2 and E-MRSE are shown in Fig. 2. MRSE-1 and MRSE-2 pad the vectors before randomization. In contrast, in E-MRSE, we randomize the vectors before padding. During the pad phase, MCSE, MRSE-1, and MRSE-2 pad 1, 2, and $U+1 (U \gg 1)$ entries, respectively. During the randomize phase, MCSE, MRSE-1, and MRSE-2 execute m, $m+2$, and $m+U+1$ randomized processes.

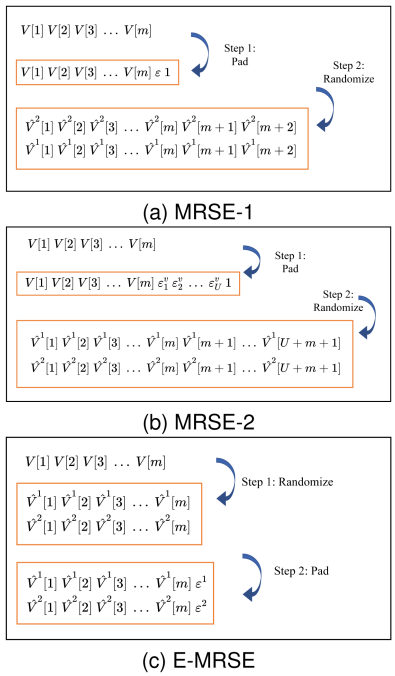


Fig. 2. Vector processing system.

A. Scheme

The four algorithms included in MRSE are described below:

- $\mathsf{Setup}\ (1^m) \to \mathcal{K}$: The data owner randomly chooses a m-bit vector S and two $(m+1)$-dimensional invertible matrices M_1, M_2. The secret key is $\mathcal{K} = \{S, M_1, M_2\}$.
- $\mathsf{IndexBuild}(\mathcal{D}, \mathcal{W}, \mathcal{K}) \to \mathcal{I}$: For each $D_i \in \mathcal{D}$, the data owner generates a file vector V_i. The initial state of V_i is a zero vector of length m. If D_i contains $w_j, j \in [1, s]$, set the j-th bit of V_i as 1. Otherwise, set it as 0. Instead of extending the vector before splitting, V_i is split into two m-bit vectors according to the secure knn technique. Subsequently, the two m-bit vectors needs to be extended to $m+1$ bits, the $(m+1)$-th entries are set as random numbers $\varepsilon_{i,1} \in E$ and $\varepsilon_{i,2} \in E$, respectively. So $V_{i,1}$ and $V_{i,2}$ are obtained. Finally, the data owner computes the sub-index $I_i = E_1(V_i) = \{M_1^T V_{i,1}, M_2^T V_{i,2}\}$. The index is $\mathcal{I} = \{I_1, \ldots, I_n\}$.
- $\mathsf{Trapdoor}(\mathcal{W}_\mathcal{Q}, \mathcal{K}) \to \mathcal{T}$: For a set of query keywords $\mathcal{W}_\mathcal{Q}$, the data user generates a query vector Q. The initial state of Q is a zero vector of length m. If $\mathcal{W}_\mathcal{Q}$ contains $w_j, j \in [1, s]$, set the j-th bit of Q as 1. Otherwise, set it as 0. First, each entry of Q is multiplied by a random number $r \in R$ and then Q is split into two vectors according to the knn technology. Subsequently, the two vectors are extended into $(m+1)$ bits. The $(m+1)$-th entry of Q_1, Q_2 are inserted into the random numbers $r_1 \in E$ and $r_2 \in E$, respectively. The data user computes the encrypted query

vector $E_2(Q) = \{M_1^{-1}Q_1, M_2^{-1}Q_2\}$. Finally, the trapdoor $\mathcal{T} = E_2(Q)$ is uploaded.

- $\mathsf{Search}(\mathcal{I}, \mathcal{T}) \rightarrow IScore$: The cloud server computes the similar inner product of the encrypted query vector and file vector via (2). After sorting all scores, the cloud server returns the top-k ranked files.

$$\begin{aligned} I_i \cdot T &= (M_1^T V_{i,1}, M_2^T V_{i,2}) \cdot (M_1^{-1}Q_1, M_2^{-1}Q_2) \\ &= V_{i,1} \cdot Q_1 + V_{i,2} \cdot Q_2 \\ &= r(V_i \cdot Q) + \varepsilon_{i,1} r_1 + \varepsilon_{i,2} r_2 \end{aligned} \tag{2}$$

B. Analysis

We analyze the E-MRSE in terms of correctness , efficiency and privacy. We compare it with MRSE-1 and MRSE-2.

- *Correctness:* The form of the similar inner product is $r(V_i \cdot Q) + \varepsilon_{i,1} r_1 + \varepsilon_{i,2} r_2$. If $Inf(R) > 2|Sup(E) - Inf(E)| \cdot |Sup(E)|$ where Sup, Inf denote the upper and lower bounds of a set, respectively, then the similarity-based ranking preserves the correct order established by the actual inner products. The reasons are follows.
  Let V_1, V_2 denote the file vectors of D_1, D_2 respectively, and let Q be a query vector. The actual inner products are defined as $IP_1 = V_1 \cdot Q$ and $IP_2 = V_2 \cdot Q$. The similar inner products are $SIP_1 = r(V_1 \cdot Q) + \varepsilon_1 r_1 + \varepsilon_2 r_2$ and $SIP_2 = r(V_2 \cdot Q) + \varepsilon_3 r_1 + \varepsilon_4 r_2$. Since V_i, Q are binary vectors, the difference between IP_1, IP_2 is at least 1 if $IP_1 \neq IP_2$. Without loss of generality, we may assume that $IP_1 = IP_2 + 1$. For the similar inner product to yield the correct ranking, the condition $SIP_1 > SIP_2$ must hold. That is $SIP_1 - SIP_2 = r(IP_1 - IP_2) + r_1(\varepsilon_1 - \varepsilon_3) + r_2(\varepsilon_2 - \varepsilon_4) > 0$. Given $IP_1 = IP_2 + 1$, we can obtain $r > r_1(\varepsilon_3 - \varepsilon_1) + r_2(\varepsilon_4 - \varepsilon_2)$. Since $\varepsilon_1, \varepsilon_2, \varepsilon_3, \varepsilon_4, r_1, r_2$ are chosen from E, then $\varepsilon_{3(4)} - \varepsilon_{1(2)} < |Sup(E) - Inf(E)|$ and $r_{1(2)} < |SupE|$. Therefore, $r_1(\varepsilon_3 - \varepsilon_1) + r_2(\varepsilon_4 - \varepsilon_2) < 2|Sup(E) - Inf(E)| \cdot |SupE|$. Since r is chosen from R, we set the lower bound condition of R as $Inf(R) > 2|Sup(E) - Inf(E)| \cdot |Sup(E)|$. Therefore, $r_1(\varepsilon_3 - \varepsilon_1) + r_2(\varepsilon_4 - \varepsilon_2) < r$. For example, given $E = [0.3, 1]$, we may choose R such that $\inf(R) > 1.4$.
- *Privacy:* The privacy of indexes and trapdoors are protected if $\mathcal{K}$ is kept secret.
  The details of the security proof of E-MRSE are as follows: The security game is played by an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$.
  Setup: Given a security parameter λ, $\mathcal{C}$ randomly chooses a m-dimensional vector, two $(m+1)$-dimensional invertible matrices.
  Query1: $\mathcal{A}$ is allowed to makes sub-index queries and trapdoor queries. For an index query, $\mathcal{C}$ runs $\mathsf{IndexBuild}$ to generates an index and returns it to $\mathcal{A}$. For a trapdoor query, $\mathcal{C}$ runs $\mathsf{Trapdoor}$ to generates a trapdoor and returns it to $\mathcal{A}$.
  Challenge: After queries, $\mathcal{A}$ chooses two distinct files d_0 and d_1 adaptively, under the constraint that d_0, d_1 have not been queried. $\mathcal{C}$ randomly chooses $b \in \{0, 1\}$ and runs $\mathsf{IndexBuild}$ on V_b to get the index I_b, which is given to $\mathcal{A}$.

TABLE III
COMPARISON ON COMPUTATION COMPLEXITY

	MRSE-2	E-MRSE
Setup	$\mathcal{O}((U+m+1)^3)$	$\mathcal{O}((m+1)^3)$
IndexBuild	$\mathcal{O}((U+m+1)^2)$	$\mathcal{O}((m+1)^2)$
Trapdoor	$\mathcal{O}((U+m+1)^2)$	$\mathcal{O}((m+1)^2)$
Search	$\mathcal{O}(N)$	$\mathcal{O}(N)$
Index Space	$\mathcal{O}(U+m+1)$	$\mathcal{O}(m+1)$
Trapdoor Space	$\mathcal{O}(U+m+1)$	$\mathcal{O}(m+1)$

N is the number of Files in the dataset, and m is the number of keywords in the dataset.

Query2: Like **Query1**, $\mathcal{A}$ can make index and trapdoor queries under the restriction that d_0 and d_1 have not been queried.
Response: $\mathcal{A}$ outputs a bit b' as its guess for b.
$\mathcal{C}$ transforms d_b into a challenge index by randomization, padding and matrix multiplication. The index composes two vectors, consisting of random numbers.
One way is the adversary only learns knowledge from the index or trapdoor. This corresponds to the chosen-plaintext attack in cryptography. The $\mathsf{IndexBuild}$ algorithm is similar to $\mathsf{trapdoor}$ algorithm, so we will only consider the index case. From the adversary's standpoint, the index is two $m+1$-dimensional random vectors. If $\mathcal{A}$ guesses b correctly with the non-negligible advantage, $\mathcal{A}$ will be able to recover M_1 and M_2 or know the secret information directly from the index. For the former, since the index query only replies to the final index vector and contains no intermediate vectors, it is not feasible to build the equation to recover M_1 and M_2. For the latter, $\mathcal{A}$ analyzes V_b is equal to any vector in $R \times R \times \cdots \times E$ with a different probability. It goes against the randomness.
The other way is that the adversary obtains information from the inner product scores. The restriction on the trapdoor query shows $V_0 \cdot \mathcal{Q} = V_1 \cdot \mathcal{Q}$, where V_0 and V_1 represent the vectors of files d_0 and d_1 respectively, and $\mathcal{Q}$ denotes the vector of the queried trapdoor. If $\mathcal{A}$ can break E-MRSE with a non-negligible advantage, then $\mathcal{A}$ analyzes $\varepsilon_{b,1} r_1 + \varepsilon_{b,2} r_2$ is equal to any vector in E with a different probability. It goes against the randomness.
Therefore, $\mathcal{A}$ guesses b with non-negligible advantage.

- *Efficiency:* During $\mathsf{IndexBuild}$ or $\mathsf{Trapdoor}$, the transformation procedure for m-dimensional file vectors to sub-indexes or trapdoors includes randomization, padding and encryption with matrices. The main computational overhead consists of two multiplications of a $m+1$-dimensional matrix with a $m+1$-dimensional vector. We compare E-MRSE with MRSE-1 and MRSE-2, their indexes are $m+1$, $m+2$, and $m+U+1$ dimensions, respectively. MRSE-1 is a special case of MRSE-2 when $U = 1$. Table III shows the complexity analysis of of MRSE-2 and E-MRSE and Fig. 3 shows the time cost and communication overhead of $\mathsf{IndexBuild}$. We fix the initial vector to be 1500 dimensions, i.e. $m = 1500$. The index vector of E-MRSE has only 1 more dimension than the file vector, so the vector processing time and index size do not vary with U shown as a horizontal line in Fig. 3(a)

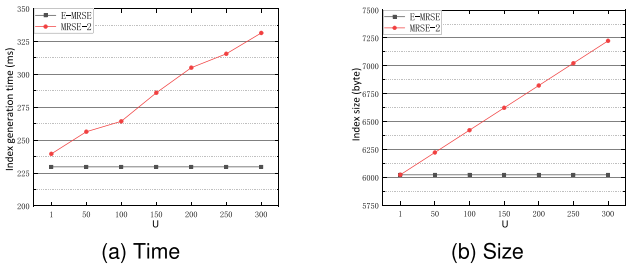


Fig. 3. Time cost and communication overhead of IndexBuild (Detailed experimental configurations are described in Section VIII).

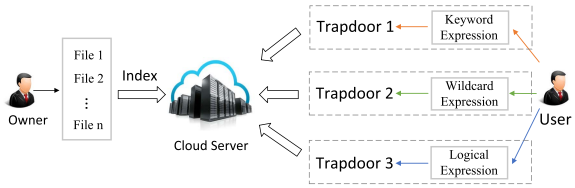


Fig. 4. The architecture of MCSE.

and (b). For MRSE-2, both time cost and communication overheads keep increasing as U increases.

VI. Multi-Character Search Encryption

In this section, we propose a multi-character search encryption scheme MCSE based on E-MRSE.

A. Overview

The architecture of MCSE is shown in Fig. 4. The user can provide keyword expressions, wildcard expressions, and logical expressions to search on the index. In MCSE, three feature extraction methods are designed to accommodate the three types of expressions. For keyword expressions, we extract each character, its positions, and the length of the keyword expression. For wildcard expressions, we additionally extract the desired keyword information. For logical expressions, we view each operation ("or" and "not") as a single unit. Within a unit, all characters are assigned the same positional value. Then the extraction process is like keyword expressions.

A match between an index and a search expression is a match between their feature sets. If the feature set of the keyword contains all the elements of the feature set of the search expression, then two match. For privacy, we use vectors to represent feature sets and use the radomize-pad model and invertible matrices to encrypt vectors in Section V to encrypt vectors. In this way, the inner product of indexes and trapdoors is the similar inner product of the original vectors. If the similarity inner product is greater than the user-supplied indicator, the keyword corresponding to the index matches the expression corresponding to the trapdoor. To illustrate this more clearly, we first present feature extraction strategies for different search expressions. Secondly, we show the vector conversion method. At last, we illustrate our scheme MCSE.

Algorithm 1: Feature Extraction for Keyword.

Input: The keyword W
Output: The feature set $FSet$
1 Initialize $FSet$ to the empty set;
2 Set the length of W as Lw;
3 **for** $i = 0$ *to* $Lw - 1$ **do**
4 *Extract feature* $W[i] \parallel i + 1$;
5 Append it to $FSet$;
6 Append $- \parallel (Lw + 1)$ to $FSet$.

B. Feature Extraction Strategy

Wildcard expressions and logical expressions are composed of letters and symbols rather than complete keywords, then if we build an index based on complete keywords it is difficult to match the trapdoor to the index. Hence, we use a feature extraction strategy to extract character features from expressions. For keywords, we introduce a terminator " $-$ " to mark the length of keywords. For wildcard expressions, the data user needs to provide L to specify the length of desired keywords, and he/she concatenates " $-$ " and $L + 1$ as a feature used to check whether the length of the keyword in the index is L. It is efficient to avoid false positives from just prefix matching [10].

1) Keyword: Keywords are complete and do not contain any symbols. The feature extraction strategy for keywords is illustrated in Algorithm 1. The feature extraction strategy is to concatenate each character of the keyword and its position in the keyword. For example, the feature set extracted for the keyword "abstract" is $\{a \parallel 1, b \parallel 2, s \parallel 3, t \parallel 4, r \parallel 5, a \parallel 6, c \parallel 7, t \parallel 8, - \parallel 9\}$.

We also adopt this strategy to extract features for keyword expressions in the trapdoor generation phase. Note that an indicator is required to be introduced during the trapdoor generation phase, which is used to match an expression to indexes. Here, the indicator is the size of $FSet$.

2) Wildcard Expression: We consider two types of wildcards "?" and " $*$ ". "?" denotes a single-character wildcard and " $*$ " denotes a multi-character wildcard. In our feature extraction strategies, we consider any single-wildcard and two or less multi-character wildcards. The details are shown in Algorithm 2. The algorithm employs A and B to represent the minimum and maximum number of characters, respectively, that the first wildcard "*" can represent. Additionally, it uses j_1 and j_2 to mark the positions of the multi-character wildcard "*", with both initialized to L_w. Depending on the number of wildcards n in the expression, the process is structured as follows: If no wildcard is present ($n = 0$), features are extracted according to lines 12-16. If a single-character wildcard occurs ($n = 1$), its position is denoted by j_1, and feature extraction follows lines 12-21. For expressions containing two wildcards ($n = 2$), the positions j_1 and j_2 are used, and lines 12-24 are executed accordingly.

The feature extraction strategy is to encode characters and not encode wildcards. The single-character wildcard represents any one character. For example, if the expression is $ab?, L = 3$, only $\{a \parallel 1, b \parallel 2, - \parallel 4\}$ is extracted. If the expression is

Algorithm 2: Feature Extraction for Wildcard Expression.

Input: The wildcard expression W and the length of desired keywords L
Output: The feature set $FSet$ and an indicator ir

Set Lw as the length of W and n as the number of $*$;
Initialize j_1, j_2 as Lw and $A = 0, B = 0$;
Extract feature $- \parallel (L+1)$;
if $n == 1$ **then**
  Let j_1 represent the location that $*$ occurs;
  $A = L - Lw + 1, B = L - Lw + 1$;
if $n == 2$ **then**
  Let j_1, j_2 represent locations that $*$ occurs;
  $A = 0, B = L - Lw + 2$;
for $k = A$ *to* B **do**
  Initialize $FSet[k]$ to the empty set;
  for $i = 0$ *to* $j_1 - 1$ **do**
    if $W[i] \neq ?$ **then**
      Extract feature $W[i] \parallel i + 1$;
  if $j_1 = Lw$ **then**
    break // Exit the outer loop
  for $i = j_1 + 1$ *to* $j_2 - 1$ **do**
    if $W[i] \neq ?$ **then**
      Extract feature $W[i] \parallel i + k$;
  if $j_2 = Lw$ **then**
    break // Exit the outer loop
  for $i = j_2 + 1$ *to* $Lw - 1$ **do**
    if $W[i] \neq ?$ **then**
      Extract feature $W[i] \parallel i + L - Lw$;
  Append all features to $FSet[k]$;
  Set $ir_k = |FSet[k]|$

$*ab, L = 8$, the feature set is $\{a \parallel 7, b \parallel 8, - \parallel 9\}$. The indicator is set to the size of the feature set, i.e. 3. The multi-character wildcard can represent a string of any length. Since the data user provides the length of the desired keyword L, the length range represented by the multi-character wildcard can be determined. If an expression has two multi-wildcards, the clients need to extract $L - Lw + 3$ $FSet$s. For example, if $a * p * b, L = 5$, the feature sets are $\{a \parallel 1, p \parallel 2, b \parallel 5, - \parallel 6\}$, $\{a \parallel 1, p \parallel 3, b \parallel 5, - \parallel 6\}$, and $\{a \parallel 1, p \parallel 4, b \parallel 5, - \parallel 6\}$. The indicator is also set to the size of the feature set.

We can also use the idea of Algorithm 2 to extract features when the number of "$*$" is bigger than 2. The size of the trapdoor is relevant to the length of the expression, the number of wildcards and the length of the desired keyword. Let the length of an expression is k, the number of "$*$" is x, and the length of the desired keyword is L. When $x = 1$, the trapdoor includes a pair of vectors. When $x = 2$, the trapdoor includes $L - k + 3$ pairs of vectors. When $x = 3$, the trapdoor includes $(L - k + 4)(L - k + 5)/2$ pairs of vectors. Note that, keywords with more than two multi-wildcards are seldomly used in practice [21]. Therefore, we consider two multi-character wildcards or less.

Algorithm 3: Feature Extraction for Logical Expression.

Input: The logical expression W
Output: The feature set $FSet$ and an indicator ir

Initialize $FSet$ to the empty set;
Set the length of W as Lw;
Set a counter cr as 1;
for $i = 0$ *to* $Lw - 1$ **do**
  if (*occurs* $W[i]$ **then**
    Concatenate elements in parentheses ( ) *with* cr *separately;*
    Append them to $FSet$;
  else if [*occurs* $W[i]$ **then**
    Concatenate the elements in the alphabet but not in square bracket [ ] *with* cr *separately*;
    Append them to $FSet$;
  else
    Concatenate $W[i] \parallel cr$
  Increase cr by 1;
Append $- \parallel (Lw + 1)$ to $FSet$;
Set $ir = Lw + 1$.

3) Logical Expression: Logical expressions are sequences of characters that define a search pattern, mostly for use in pattern matching with strings. Here, we consider three types of logical operators: "and", "or" and "not". Logical expression supports that the data user specifies a range of characters in each position. The details of feature extraction are shown in Algorithm 3. The "and" operators are naturally satisfied, i.e. all features in each location are placed in the feature set. For $(\cdots)$, character features are extracted by concatenating each character and the position of "(". In addition, for $[\cdots]$, character complement features are extracted by connecting each character in the alphabet but not in $[\cdots]$ and the position of "[". For example, $(wpy)an$ can be used to search for wan, pan, or yan. The feature set is $\{w \parallel 1, p \parallel 1, y \parallel 1, a \parallel 2, n \parallel 3, - \parallel 4\}$ and the indicator is 4. This indicator is determined as the length of the logical expression plus one.

Complete feature extraction strategies mean the extracted feature set uniquely determines the keyword or expression. The three algorithms provide complete feature extraction strategies. Take Algorithm 1 for example, the proof of the complete feature extraction strategy is as follow: Suppose two keywords have the same feature sets, but these two keywords are different. The reason that the keywords are different may be that the keywords have different lengths or the keywords are different at a certain position. Suppose keywords A and B have different characters at the i-th position, one is a and the other is b. Then when using Algorithm 1 to extract features, the feature set of A contains $a \parallel i$ and the feature set of B contains $b \parallel i$, then the feature sets of A and B cannot be the same. Suppose A and B have different lengths, say A has length 7 and B has length 8, then when using Algorithm 1 to extract the features, the feature set of A contains $- \parallel 8$ and the feature set of B contains $- \parallel 9$, then the feature

DO runs $\mathsf{Setup}(1^m)$:

1. Generate a vector $S \leftarrow \{0,1\}^m$ and two $m+1$-dimensional invertible matrices M_1, M_2 randomly.
2. Sample $k \leftarrow \Gamma.KGen(1^m)$.
3. Save S, M_1, M_2 and k secretly.

DO runs $\mathsf{IndexBuild}(\mathcal{D}, \mathcal{W}, S, M_1, M_2, k)$:

- Initialization for files:
  1. Scan each file in D and extract keywords.
  2. Integrate all different keyword to generate a keyword set $\mathcal{W} = w_1, w_2, \cdots, w_N$.
  3. Invoke $\Gamma.Enc$ to encrypt each file.
- Build index: for $w_i \in \mathcal{W}$,
  1. Extract its character features using Algorithm 1.
  2. Invoke Algorithm 4 to generate keyword vector V_i.
  3. Similar to E-MRSE, convert V_i into two $m+1$-dimensional vectors $V_{i,1}, V_{i,2}$.
  4. Encrypt the vectors using Eq. (3) to obtain a sub-index $E_1(V_i)$.
$$E_1(V_i) = \{M_1^T V_{i,1}, M_2^T V_{i,2}\} \tag{3}$$
  5. Create an array A_i to store all file identifiers in $D(w_i)$.
- Upload to the server:
  Output $\mathcal{I} = \{(I_1, A_1), \cdots, (I_N, A_N)\}$.

DU runs $\mathsf{Trapdoor}(W, S, M_1, M_2)$:

- Generate trapdoor:
  1. Extract character features for search expression W using Algorithm 1, 2, or 3, and an indicator can be obtained, denoted as v.
  2. Generate the corresponding vector Q according to Algorithm 4.
  3. Similar to E-MRSE, convert Q into two $m+1$-dimensional vectors Q_1, Q_2.
  4. Encrypt the vectors using Eq. (4) to obtain encrypted vectors $E_2(Q)$.
$$E_2(Q) = \{M_1^{-1} Q_1, M_2^{-1} Q_2\} \tag{4}$$
  5. Compute a new indicator I using Eq. (5) for vector matching, where r is a random number multiplied by the vector Q.
$$I = v * r \tag{5}$$
- Upload to the server:
  Output the trapdoor $\mathcal{T} = \{I, E_2(Q)\}$.

CS runs $\mathsf{Search}(\mathcal{I}, \mathcal{T})$:

- Match:
  1. For $I_i, i \in [1, n]$, compute Eq. (6) and compare $I_i \cdot T$ with I. If it is greater than I, append file identifiers in A_i to the search result.
  2. Find encrypted files for each file identifier in the search result.
  3. Add the files to the search result.
$$\begin{aligned} I_i \cdot T &= (M_1^T V_{i,1}, M_2^T V_{i,2}) \cdot (M_1^T Q_1, M_2^T Q_2) \\ &= V_{i,1} \cdot Q_1 + V_{i,2} \cdot Q_2 \\ &= r(V_i \cdot Q) + \varepsilon_{i,1} r_1 + \varepsilon_{i,2} r_2 \end{aligned} \tag{6}$$
- Return to DU:
  Output the search result.

Fig. 5. Multi-character Searchable Encryption Scheme.

Algorithm 4: Vector Conversion.

Input: The feature list FL and a feature set $FSet$
Output: A vector V

```
Let |FL| represent the length of FL;
Initialize V to zero vector of length |FL|;
for i = 0 to |FL| − 1 do
    if FL[i] occurs FSet then
        Set V[i] as 1.
```

sets of A and B cannot be the same. Therefore, the feature sets of two different keywords must be different. Therefore, the feature extraction method of Algorithm 1 uniquely identifies a keyword.

C. Vector Conversion

A feature list FL needs to be defined in advance. The features in FL depend on the application scenario. The dataset used for our experiments is from the Enron and Paper, so only characters in the alphabet are considered. Let the length of the longest keyword is ML (ML can be obtained by scanning all the keywords), and FL contains $27 * ML + 1$ elements as $\{a \parallel 1, \ldots, a \parallel ML, b \parallel 1, \ldots, b \parallel ML, \ldots, z \parallel ML, - \parallel 1, - \parallel 2, \ldots, - \parallel (ML+1)\}$.

The length of the vector is fixed, and it is pre-defined in MCSE. Algorithm 4 is to convert the feature set to the feature vector which based on the relationship between the elements in the feature set and the feature list FL. The initial value of the vector is a zero vector. The i-th bit of this vector is set to 1 if the i-th character feature in the FL is in the feature set.

D. MCSE

Based on the feature extraction strategies and E-MRSE above, we construct a multi-character search scheme (MCSE) in Fig. 5. Suppose that $\Gamma = (KGen(\cdot), Enc_k(\cdot), Dec_k(\cdot))$ is a secure symmetric encryption scheme with indistinguishability against chosen-plaintext attacks (IND-CPA).

Correctness requires that if $r(V_i \cdot Q) + \varepsilon_{i,1} r_1 + \varepsilon_{i,2} r_2 > I$, w_i contains all character features in the query expression. The random numbers $r, r_1, r_2, \varepsilon_{i,1}, \varepsilon_{i,2}$ are used to obfuscate the actual inner product, thus making matching based on similar inner products less accurate. So for correctness, the range of random numbers needs to satisfy this condition $Inf(R) > 2|Sup(E) - Inf(E)| \cdot |Sup(E)|$.

VII. Security Analysis

In this section, we prove that MCSE is indistinguishable secure under the IND-CFA security model.

Theorem 1. The above scheme is IND-CFA for any Probabilistic Polynomial-Time (PPT) adversary.

Proof. We prove the theorem by reduction. Suppose that there exists a PPT $\mathcal{A}$ who breaks the above scheme with a non-negligible advantage. We use $\mathcal{A}$ to construct an algorithm D. D can break the randomness of a random value.

Setup: Given a security parameter λ, $\mathcal{C}$ randomly chooses a m-dimensional vector, two $(m+1)$-dimensional invertible matrices.

Query1: $\mathcal{A}$ is allowed to makes sub-index queries and trapdoor queries. For a sub-index query, $\mathcal{C}$ runs IndexBuild to generates a sub-index and returns it to $\mathcal{A}$. For a trapdoor query, $\mathcal{C}$ runs Trapdoor to generates a trapdoor and returns it to $\mathcal{A}$.

Challenge: After making some sub-index queries, $\mathcal{A}$ outputs two distinct keywords w_0 and w_1 under the restriction that the expression W has not been queried where $W[i] = w_0[i]$ or $w_1[i]$ but $w_0[i] \neq w_1[i]$. $\mathcal{C}$ randomly chooses $b \in \{0, 1\}$ and runs IndexBuild on V_b to get the index I_b, which is given to $\mathcal{A}$.

Query2: $\mathcal{A}$ is allowed to makes sub-index queries and trapdoor queries. For a sub-index query, $\mathcal{C}$ responds to sub-index queries in the same way as in **Query1** without restriction. For a trapdoor query, $\mathcal{C}$ runs Trapdoor to generates a trapdoor and returns it to $\mathcal{A}$. But the query expressions do not contain character features that are in w_0 but not in w_1 or character features that are in w_0 but not in w_1.

Response: $\mathcal{A}$ outputs a bit b' as its guess for b.

$\mathcal{C}$ transforms w_b into a challenge index by randomization, padding, and matrix multiplication. The index is composed of two vectors, consisting of random numbers.

One way is the adversary only gets information from the index or trapdoor. This corresponds to the chosen-plaintext attack in cryptography. The IndexBuild algorithm is similar to trapdoor algorithm, so we will only consider the index case. From the adversary's standpoint, the index is two $m+1$-dimensional random vectors. If $\mathcal{A}$ guesses b correctly with the non-negligible advantage, $\mathcal{A}$ will be able to recover M_1 and M_2 or know the secret information directly from the index. For the former, since the index query only replies to the final index vector and contains no intermediate vectors, it is not feasible to build the equation to recover M_1 and M_2. For the latter, $\mathcal{A}$ analyzes V_b is equal to any vector in $R \times R \times \cdots \times E$[1] with a different probability. It goes against the randomness.

Another way that the adversary gets information from the inner product scores. The restriction on the trapdoor query shows $V_0 \cdot \mathcal{Q} = V_1 \cdot \mathcal{Q}$, where V_0 and V_1 represent the vectors of keywords w_0 and w_1 respectively, and $\mathcal{Q}$ denotes the vector of the queried trapdoor. If $\mathcal{A}$ can break MCSE with a non-negligible advantage, then $\mathcal{A}$ analyzes $\varepsilon_{b,1} r_1 + \varepsilon_{b,2} r_2$ is equal to any vector in E with a different probability. It goes against the randomness.

Therefore, $\mathcal{A}$ guesses b with the non-negligible advantage.

[1]the Cartesian product of two sets A and B, denoted $A \times B$, is the set of all ordered pairs (a, b) where a is in A and b is in B

VIII. Performance Analysis

Dataset. The Enron and the Paper datasets are selected as benchmark datasets due to their highly suitability for modeling keyword/value pair structures. We treat each email and each paper's abstract as an individual file and view the content of the file as a collection of keywords for processing and analysis. These datasets and source codes of MCSE are on Github https://github.com/UbiPLab/MCSE.

Metrics. We evaluate the running time of index generation, index encryption, trapdoor generation and search of MCSE. We compare MCSE with the classical schemes, SKS [10], SSE-MCW [22] and EWFSS [11] all of which support wildcard-based keyword searches. MCSE, SKS, SSE-MCW, and EWFSS also utilize feature-based matching mechanisms. Notably, none of these schemes disclose the positions of wildcards. Since SKS, SSE-MCW, and EWFSS do not have index encryption processes, we compare MCSE with the three schemes with respect to index generation. We also compare MCSE with the state-of-the-art, SPWSE [17], and SFWQ [18]. Since SPWSE and SFWQ can resist correlation attacks, we compare our scheme with SPWSE for total index building and total search process.

Experiment setting. We implemented all experiments on a computer with 11th Gen Intel(R) Core(TM) i5-1135G7 running on Windows 11 64-bit operation system. All algorithms are programmed using Java language. We use JPBC library to realize the group operations and bilinear pairs that are used in SFWQ and SPWSE.

Theoretical analysis. Table IV summarizes the computational and space complexity of the schemes SKS, SSE-MCW, EWFSS, SPWSE, SFWQ, and MCSE across the Setup, IndexBuild, Trapdoor, and Search phases. In the Setup phase, MCSE involves the computation of an inverse matrix, resulting in a computational complexity of $\mathcal{O}(s_{fl}^3)$ During the IndexBuild phase, MCSE performs vector-matrix multiplication, leading to a computational complexity of $\mathcal{O}(s_{fl}^2)$. While this complexity is higher than that of SKS, SSE-MCW, and EWFSS, they cannot to resist correlation attacks. Notably, SPWSE and SFWQ exhibit lower computational complexity; however, their operations are group-based computations, which brings large overhead.

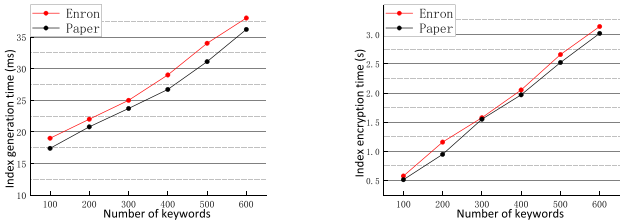
A. Index Building

The index building process can be divided into 4 phases: feature extraction, vector conversion, vector processing, and index encryption. The first two phases are collectively referred to as the index generation process. The last two phases are collectively referred to as the index encryption process. We fix $ML = 16$ and evaluate the time cost of index generation on two datasets. Fig. 6(a) shows that the time cost of index generation has a roughly linear relationship with the number of keywords. From Fig. 6(b), we can see that the encryption time is almost linear to the number of keywords. The experiment results show an expected higher time spent encrypting indexes rather than generating indexes. But the encryption process enables MCSE to resist correlation attacks.

TABLE IV
COMPARISON ON COMPUTATION AND SPACE COMPLEXITY WITH EXISTING WORKS

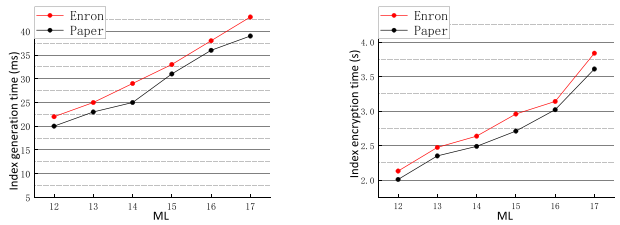
Property	Setup	Build	Trapdoor	Search	Index Space	Trapdoor Space
SKS [10]	$\mathcal{O}(s_{bf}+k)$	$\mathcal{O}(s_{bf}+kn_f)$	$\mathcal{O}(s_{bf}+kn_f)$	$\mathcal{O}(N)$	$\mathcal{O}(s_{bf})$	$\mathcal{O}(s_{bf})$
SSE-MCW [22]	$\mathcal{O}(s_{bf}+k)$	$\mathcal{O}(s_{bf}+kn_f)$	$\mathcal{O}(s_{bf}+kn_f)$	$\mathcal{O}(N)$	$\mathcal{O}(s_{bf})$	$\mathcal{O}(s_{bf})$
EWFSS [11]	$\mathcal{O}(s_{bf}+k)$	$\mathcal{O}((s_{bf}+kn_f))$	$\mathcal{O}(s_{bf}+kn_f)$	$\mathcal{O}(N)$	$\mathcal{O}(s_{bf})$	$\mathcal{O}(s_{bf})$
SPWSE [17]	$\mathcal{O}(2l_v+1)$	$\mathcal{O}(l_vloglv)$	$\mathcal{O}(l_v)$	$\mathcal{O}(N)$	$\mathcal{O}(l_v)$	$\mathcal{O}(l_v)$
SFWQ [18]	$\mathcal{O}(l_v)$	$\mathcal{O}(l_v)$	$\mathcal{O}(l_v)$	$\mathcal{O}(N)$	$\mathcal{O}(l_v)$	$\mathcal{O}(l_v)$
TSEWS [23]	$\mathcal{O}(l_v^3)$	$\mathcal{O}(l_v^2)$	$\mathcal{O}(l_v^2)$	$\mathcal{O}(N)$	$\mathcal{O}(l_v)$	$\mathcal{O}(l_v)$
MCSE	$\mathcal{O}(s_{fl}^3)$	$\mathcal{O}(s_{fl}^2)$	$\mathcal{O}(s_{fl}^2)$	$\mathcal{O}(N)$	$\mathcal{O}(s_{fl})$	$\mathcal{O}(s_{fl})$

s_{bf} is the size of Bloom Filter, k is the number of hash functions, N is the number of keywords in the dataset, n_f is the number of the features, l_v is the max length of the keywords, and s_{fl} is the size of the feature list.Note, the operations in SPWSE, SFWQ, and TSEWS are group-based computations.



(a) Index generation (b) Index encryption

Fig. 6. Time cost.



(a) Index generation (b) Index encryption

Fig. 7. Time cost.

ML will affect the performance of our scheme to a great extent. We took ML to be 12, 13, 14, 15, 16 and 17 to test its impact on the index generation process and the index encryption process. As shown in Fig. 7(a), the index generation time is linear with ML. It is because the dimension of the vector is $27 * ML + 1$. As shown in Fig. 7(b), the encryption time is almost quadratic with the value of ML. The reason is that the encryption algorithm includes four multiplications of vector and matrix. But overall, it requires approximately 3.14 s to encrypt indexes containing 600 keywords, which is an affordable security trade-off for correlation resistance.

We conduct extensive experiments on Enron and Paper datasets. As illustrated in Figs. 6 and 7, the red line represents the time on Enron dataset, and the black line represents the time overhead on Paper dataset. We can obtain that the index building time is consistent overall across different datasets.

In terms of index generation, we further compared MCSE with SKS [10], SSE-MCW [22] and EWFSS [11]. From Fig. 8, MCSE has minimal time cost compared to these schemes and performs well.

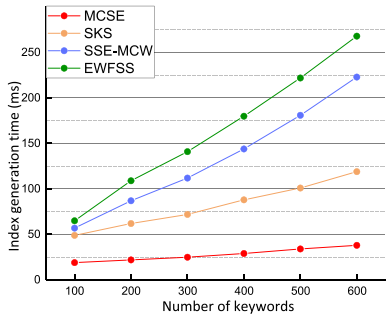


Fig. 8. Comparison of index generation time.

TABLE V
TIME COST FOR GENERATING TRAPDOORS

KE	SWE	LE	1MWE	2MWE
4ms	4ms	4ms	4ms	24ms

B. Search

Our scheme supports keyword expressions (KE), single-character wildcard expressions (SWE), logical expressions (LE), and multi-character wildcard expressions (MWE). Furthermore, we design two types of MWE: MWE contains one multi-character wildcard (1MWE) and MWE contains two multi-character wildcards (2MWE).

The trapdoor generation process is composed of feature extraction, vector conversion, vector processing, and trapdoor encryption. KE, SWE, LE, and 1MWE all require only one feature set to be extracted. It is not surprising that the trapdoor generation times of these expressions are nearly identical from Table V. For 2MWE, we choose an expressions $a * h * n$ and set L as 8. The two wildcards indicate 5 characters due to $L = 8$ and the number of extracted feature sets is 6. From Table V, the trapdoor generation time of 2MWE is longer than the other four search expressions.

Fig. 9 demonstrates the efficiency of MCSE varying with different number of keywords. The search time of KE, SW, LEW and 1MWE are nearly identical. There is roughly a linear relationship between the keyword search time and the number of keywords. When the number of keywords in the dataset is 600, the search time for all four expressions is only 11 ms. For 2MWE, the search time is still linearly related to the number of keywords. When the number of keywords in the dataset is 600, the search time for 2MWE is approximately 64 ms. The trapdoor

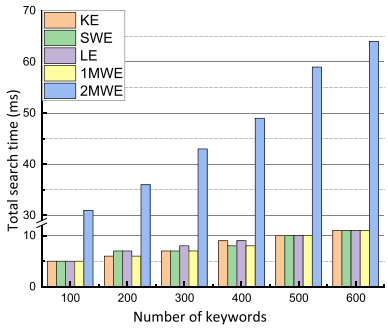


Fig. 9. Time cost for search.

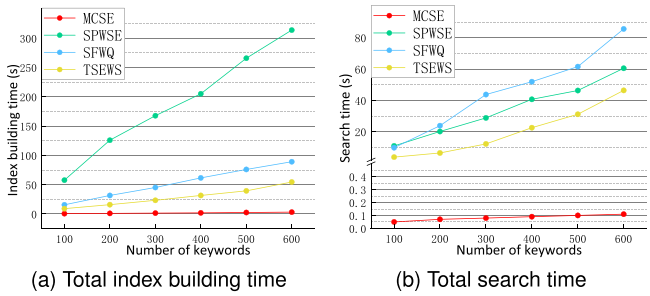


Fig. 10. Comparison SPWSE, SFWQ and TSEWS.

of 2MWE has 6 pairs of vectors is 6 times as long as KE, so the search time is also about 6 times as long as KE.

C. Comparison With SPWSE and SFWQ

SPWSE [17], SFWQ [18], and TSEWS [23] are three wildcard keyword searchable encryption schemes that can resist correlation attacks. We compare our scheme with SPWSE, SFWQ, and TSEWS with regard to different numbers of keywords. Fig. 10(a) illustrates the total index building time of the four schemes. As the volume of keywords in the dataset increases, the computational time of MCSE exhibits only a marginal rise. In contrast, the time overhead of SPWSE is approximately two orders of magnitude higher than that of MCSE, while TSEWS requires about 18 times the computational time of MCSE. The superior efficiency of MCSE stems primarily from its lightweight computational framework, whereas both SPWSE and TSEWS involve operations in bilinear pairing groups, which inherently incur higher computational complexity. Fig. 10(b) illustrates the total search time of MCSE, SPWSE, TSEWS, and SFWQ. The search time of MCSE is linearly related to the number of keywords with a small increase. Compared to SFWQ, when the dataset contains 600 keywords, MCSE can save approximately 99.87% of the time cost. This is attributed to the fact that our scheme employs a trapdoor to match all indexes, whereas SFWQ needs to adjust the trapdoor to a specific one tailored for each index. In addition, we evaluate the communication cost of an index and a trapdoor in Table VI. Compared with SPWSE and SFWQ, MCSE achieves lower overall communication overhead. Although MCSE introduces higher communication costs than TSEWS, this is a trade-off that provides enhanced flexibility in supporting expressive search operations.

TABLE VI
COMPARISON OF COMMUNICATION COSTS

Index (Data Owner → Server)			
MCSE	SPWSE	SFWQ	TSEWS
3.38KB	4.25KB	6.4KB	2.1KB
Trapdoor (Data User → Server)			
MCSE	SPWSE	SFWQ	TSEWS
3.38KB	4.25KB	1.1KB	2.1KB

IX. DISCUSSIONS

Scalability and update: The index updates are supported through an independent indexing mechanism, allowing new data to be indexed without requiring a complete rebuild of the existing indexes. This ensures the system remains scalable and responsive to dynamic data environments. As larger datasets inevitably lead to an increase in the number of keywords, the size of the indexes grows proportionally with the keyword volume. Consequently, the search overhead also scales linearly with the number of keywords. To mitigate storage demands, the server can maintain a history of search expressions, thereby optimizing search efficiency, like [54].

Balance between efficiency and security: To resist correlation attacks, the encrypted search process requires computing the inner product of encrypted vectors. While this introduces additional computational overhead, it significantly enhances security. The encryption method is the randomized-pad mode which requires only a single additional dimension for padding, reducing computation cost.

Forward and backward privacy: Forward privacy ensures that adversaries cannot obtain any information about the keyword from the newly added files. It also means the server cannot use the old trapdoor to match newly added keywords. So we need to add timestamps when building the index. But it will result in that it needs to generate multiple trapdoors corresponding to multiple indexes when a search. We will study this problem in our future work. Backward privacy ensures search queries cannot leak matching entries after they have been deleted. We can encrypt the identifiers of files to achieve weak backward privacy.

Search pattern leakage and side-channel attacks: A randomization process is incorporated during trapdoor generation, so the trapdoors generated by multiple calls to the trapdoor generation algorithm are different. The adversary can infer the search pattern by analyzing the search results. Specifically, if the results of two searches are identical, it is highly probable that the search expressions are the same. To mitigate this risk, we can introduce padding by adding fake files, ensuring that the number of files associated with each keyword remains consistent. Additionally, the identifiers of these files should be encrypted. When the user retrieves the search results, they can filter out the fake files and extract real files. The match process in MCSE is to compute the inner product of fixed-dimension vectors. So the adversary cannot launch side-channel attacks by testing the match time.

Balance of search flexibility and security: We design three feature extraction strategies to support flexibility and design the randomize-pad encryption mode to guarantee security. Since the encryption method uses symmetric keys, MCSE can only support multi-user scenarios in a trivial manner—for instance, by sharing the same key across all users or maintaining separate keys for each user.

X. Conclusion

In this paper, we propose a multi-character searchable encryption scheme MCSE that effectively resists the correlation attack, eliminates false positives and supports rich search functionalities, including keyword expression, wildcard expression and logical expression searches. We enhance the security model of wildcard keyword searchable encryption by considering the correlation attack. We provide formal proof of MCSE to demonstrate its security. Furthermore, we realize MCSE to demonstrate its feasibility and efficiency.

References

[1] J. Liu, J. Qin, X. Zhang, and H. Wang, "Efficient key-aggregate cryptosystem with user revocation for selective group data sharing in cloud storage," *IEEE Trans. Knowl. Data Eng.*, vol. 36, no. 11, pp. 6042–6055, Nov. 2024.

[2] J.-N. Liu et al., "Enabling efficient, secure and privacy-preserving mobile cloud storage," *IEEE Trans. Dependable Secure Comput.*, vol. 19, no. 3, pp. 1518–1531, May/Jun. 2022.

[3] D. X. Song, D. Wagner, and A. Perrig, "Practical techniques for searches on encrypted data," in *Proc. 21st IEEE Symp. Secur. Privacy*, 2000, pp. 44–55.

[4] B. Ferreira, B. Portela, T. Oliveira, G. Borges, H. Domingos, and J. Leitão, "Boolean searchable symmetric encryption with filters on trusted hardware," *IEEE Trans. Dependable Secure Comput.*, vol. 19, no. 2, pp. 1307–1319, Mar./Apr. 2022.

[5] B. Minaud and M. Reichle, "Dynamic local searchable symmetric encryption," in *Proc. 42nd Int. Cryptol. Conf.*, 2022, pp. 91–120.

[6] Z. Shi, X. Fu, X. Li, and K. Zhu, "ESVSSE: Enabling efficient, secure, verifiable searchable symmetric encryption," *IEEE Trans. Knowl. Data Eng.*, vol. 34, no. 7, pp. 3241–3254, Jul. 2022.

[7] C. Cai, J. Weng, X. Yuan, and C. Wang, "Enabling reliable keyword search in encrypted decentralized storage with fairness," *IEEE Trans. Dependable Secure Comput.*, vol. 18, no. 1, pp. 131–144, Jan./Feb. 2021.

[8] C. Jiang, C. Xu, and G. Yang, "Device-enhanced secure cloud storage with keyword searchable encryption and deduplication," in *Proc. 29th Eur. Symp. Res. Comput. Secur.*, 2024, pp. 396–413.

[9] S. Sedghi, P. v. Liesdonk, S. Nikova, P. Hartel, and W. Jonker, "Searching keywords with wildcards on encrypted data," in *Proc. Int. Conf. Secur. Cryptogr. Netw.*, 2010, pp. 138–153.

[10] T. Suga, T. Nishide, and K. Sakurai, "Secure keyword search using bloom filter with specified character positions," in *Proc. 6th Int. Conf. Provable Secur.*, 2012, pp. 235–252.

[11] J. Hua, Y. Liu, H. Chen, X. Tian, and C. Jin, "An enhanced wildcard-based fuzzy searching scheme in encrypted databases," *World Wide Web*, vol. 23, no. 3, pp. 2185–2214, 2020.

[12] C. Hu and L. Han, "Efficient wildcard search over encrypted data," *Int. J. Inf. Secur.*, vol. 15, no. 5, pp. 539–547, 2016.

[13] E.-J. Goh, "Secure indexes," *Cryptol. ePrint Arch.*, vol. 216, pp. 1–18, 2003.

[14] S. Chatterjee, M. Kesarwani, J. Modi, S. Mukherjee, S. K. P. Puria, and A. Shah, "Secure and efficient wildcard search over encrypted data," *Int. J. Inf. Secur.*, vol. 20, no. 2, pp. 199–244, 2021.

[15] C. Bösch, R. Brinkman, P. Hartel, and W. Jonker, "Conjunctive wildcard search over encrypted data," in *Proc. 10th Workshop Secure Data Manage.*, 2011, pp. 114–127.

[16] F. Zhao and T. Nishide, "Searchable symmetric encryption supporting queries with multiple-character wildcards," in *Proc. 10th Int. Conf. Netw. Syst. Secur.*, 2016, pp. 266–282.

[17] Y. Li, J. Ning, and J. Chen, "Secure and practical wildcard searchable encryption system based on inner product," *IEEE Trans. Serv. Comput.*, vol. 16, no. 3, pp. 2178–2190, May/Jun. 2023.

[18] Q. Wang, D. Hu, M. Li, and G. Yang, "Secure and flexible wildcard queries," *IEEE Trans. Inf. Forensics Secur.*, vol. 19, pp. 7374–7388, 2024.

[19] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Commun. ACM*, vol. 13, no. 7, pp. 422–426, 1970.

[20] Y. Wu et al., "Elastic bloom filter: Deletable and expandable filter using elastic fingerprints," *IEEE Trans. Comput.*, vol. 71, no. 4, pp. 984–991, Apr. 2022.

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

[22] C. Hu, L. Han, and S. M. Yiu, "Efficient and secure multi-functional searchable symmetric encryption schemes," *Secur. Commun. Netw.*, vol. 9, no. 1, pp. 34–42, 2016.

[23] C. Hahn, "Towards secure and efficient wildcard search for cloud storage," *IEEE Trans. Dependable Secure Comput.*, vol. 22, no. 6, pp. 6968–6982, Nov./Dec. 2025.

[24] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky, "Searchable symmetric encryption: Improved definitions and efficient constructions," *J. Comput. Secur.*, vol. 19, no. 5, pp. 895–934, 2011.

[25] N. Cui et al., "Enabling efficient, verifiable, and secure conjunctive keyword search in hybrid-storage blockchains," *IEEE Trans. Knowl. Data Eng.*, vol. 36, no. 6, pp. 2445–2460, Jun. 2024.

[26] Q. Tong, Y. Miao, J. Weng, X. Liu, K. R. Choo, and R. H. Deng, "Verifiable fuzzy multi-keyword search over encrypted data with adaptive security," *IEEE Trans. Knowl. Data Eng.*, vol. 35, no. 5, pp. 5386–5399, May 2023.

[27] V. Vo, X. Yuan, S.-F. Sun, J. K. Liu, S. Nepal, and C. Wang, "ShieldDB: An encrypted document database with padding countermeasures," *IEEE Trans. Knowl. Data Eng.*, vol. 35, no. 4, pp. 4236–4252, Apr. 2023.

[28] W. Yang, Y. Geng, L. Li, X. Xie, and L. Huang, "Achieving secure and dynamic range queries over encrypted cloud data," *IEEE Trans. Knowl. Data Eng.*, vol. 34, no. 1, pp. 107–121, Jan. 2022.

[29] W. K. Wong, D. W. Cheung, B. Kao, and N. Mamoulis, "Secure kNN computation on encrypted databases," in *Proc. 35th ACM SIGMOD Int. Conf. Manage. Data*, 2009, pp. 139–152.

[30] M. Li, Y. Chen, S. Zheng, D. Hu, C. Lal, and M. Conti, "Privacy-preserving navigation supporting similar queries in vehicular networks," *IEEE Trans. Dependable Secure Comput.*, vol. 19, no. 2, pp. 1133–1148, Mar./Apr. 2022.

[31] Q. Liu, Y. Peng, J. Wu, T. Wang, and G. Wang, "Secure multi-keyword fuzzy searches with enhanced service quality in cloud computing," *IEEE Trans. Netw. Service Manag.*, vol. 18, no. 2, pp. 2046–2062, Jun. 2021.

[32] 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. Serv. Comput.*, vol. 15, no. 4, pp. 1799–1812, Jul./Aug. 2022.

[33] X. Li et al., "VRFMS: Verifiable ranked fuzzy multi-keyword search over encrypted data," *IEEE Trans. Serv. Comput.*, vol. 16, no. 1, pp. 698–710, Jan. 2024.

[34] N. Cao, C. Wang, M. Li, K. Ren, and W. Lou, "Privacy-preserving multi-keyword ranked search over encrypted cloud data," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 1, pp. 222–233, Jan. 2014.

[35] 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.

[36] C. Chen et al., "An efficient privacy-preserving ranked keyword search method," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 4, pp. 951–963, Apr. 2016.

[37] J. Li, Q. Wang, C. Wang, N. Cao, K. Ren, and W. Lou, "Fuzzy keyword search over encrypted data in cloud computing," in *Proc. 29th IEEE Int. Conf. Comput. Commun.*, 2010, pp. 1–5.

[38] S. A. Mittal and C. R. Krishna, "Privacy preserving synonym based fuzzy multi-keyword ranked search over encrypted cloud data," in *Proc. Int. Conf. Comput., Commun. Automat.*, 2016, pp. 1187–1194.

[39] J. Wang, X. Yu, and M. Zhao, "Privacy-preserving ranked multi-keyword fuzzy search on cloud encrypted data supporting range query," *Arabian J. Sci. Eng.*, vol. 40, no. 8, pp. 2375–2388, 2015.

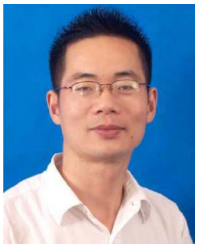
[40] W. Sun et al., "Verifiable privacy-preserving multi-keyword text search in the cloud supporting similarity-based ranking," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 11, pp. 3025–3035, Nov. 2014.

[41] S. Lai et al., "Result pattern hiding searchable encryption for conjunctive queries," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2018, pp. 745–762.

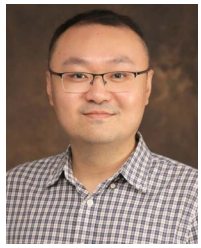
[42] D. Boneh and B. Waters, "Conjunctive, subset, and range queries on encrypted data," in *Proc. 4th Theory Cryptol. Conf.*, 2007, vol. 4392, pp. 535–554.

[43] M. Li et al., "Accurate, secure, and efficient semi-constrained navigation over encrypted city maps," *IEEE Trans. Dependable Secure Comput.*, vol. 22, no. 3, pp. 2642–2658, May/Jun. 2025.
[44] M. Yasuda, T. Shimoyama, J. Kogure, K. Yokoyama, and T. Koshiba, "Privacy-preserving wildcards pattern matching using symmetric somewhat homomorphic encryption," in *Proc 19th Australas. Conf. Inf. Secur. Privacy*, 2014, pp. 338–353.
[45] T. K. Saha and T. Koshiba, "An enhancement of privacy-preserving wildcards pattern matching," in *Proc. 9th Int. Symp. Foundations Pract. Secur.*, 2016, pp. 145–160.
[46] M. Kim, H. T. Lee, S. Ling, B. H. M. Tan, and H. Wang, "Private compound wildcard queries using fully homomorphic encryption," *IEEE Trans. Dependable Secure Comput.*, vol. 16, no. 5, pp. 743–756, Sep./Oct. 2019.
[47] J. Weener, F. Hahn, and A. Peter, "Libertas: Backward private dynamic searchable symmetric encryption supporting wildcards," in *Proc. 36th Annu. IFIP WG 11.3 Conf. Data Appl. Secur. Privacy*, 2022, pp. 215–235.
[48] R. A. Popa, C. M. Redfield, N. Zeldovich, and H. Balakrishnan, "Cryptdb: Protecting confidentiality with encrypted query processing," in *Proc. 23rd ACM Symp. Operating Syst. Princ.*, 2011, pp. 85–100.
[49] Z. Gui, K. G. Paterson, and S. Patranabis, "Rethinking searchable symmetric encryption," in *Proc. 44th IEEE Symp. Secur. Privacy*, 2023, pp. 1401–1418.
[50] C. Huang, D. Liu, A. Yang, R. Lu, and X. Shen, "Multi-client secure and efficient DPF-based keyword search for cloud storage," *IEEE Trans. Dependable Secure Comput.*, vol. 21, no. 1, pp. 353–371, Jan./Feb. 2024.
[51] K. He, J. Chen, Q. Zhou, R. Du, and Y. Xiang, "Secure dynamic searchable symmetric encryption with constant client storage cost," *IEEE Trans. Inf. Forensics Secur.*, vol. 16, pp. 1538–1549, 2021.
[52] A. Bossuat, R. Bost, P.-A. Fouque, B. Minaud, and M. Reichle, "SSE and SSD: Page-efficient searchable symmetric encryption," in *Proc. 41st Int. Cryptol. Conf.*, T. Malkin and C. Peikert, Eds., 2021, pp. 157–184.
[53] F. Luo, X. Yan, H. Yang, and X. Zheng, "PAEWS: Public-key authenticated encryption with wildcard search over outsourced encrypted data," *IEEE Trans. Inf. Forensics Secur.*, vol. 20, pp. 2212–2223, 2025.
[54] J. Wang and S. S. M. Chow, "Omnes pro uno: Practical multi-writer encrypted database," in *Proc. 31st USENIX Secur. Symp.*, 2022, pp. 2371–2388.

Qing Wang received the BS degree in information and computational science from Harbin Normal University, in 2019, and the MS degree in foundation of mathematics from Shandong University, in 2022. She is currently working toward the PhD degree in computer science and technology from the Hefei Univeristy of Technology. She was a visiting PhD student with the School of Computing and Information Systems, Singapore Management University. Her research focuses on applied cryptography.

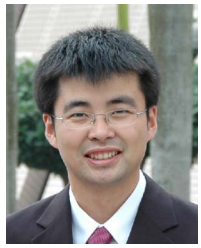


Donghui Hu (Member, IEEE) received the BS degree from Anhui Normal University, in 1995, the MS degree in computer science and technology from the University of Science and Technology of China, in 2004, and the PhD degree in information security from Wuhan University, in 2010. He was a visiting researcher with UNCC from 2013 to 2014. He is currently a professor with the College of Computer Science and Information Engineering, Hefei University of Technology. His research interests include network security, information hiding and privacy protection.



Meng Li (Senior Member, IEEE) received the PhD degree in computer science and technology from the School of Computer Science and Technology, Beijing Institute of Technology, China, in 2019. He was a postdoc researcher with the the Department of Mathematics and HIT Center, University of Padua, Italy, where he is led by Prof. Mauro Conti (IEEE Fellow). He is currently a professor with the School of Computer Science and Information Engineering, Hefei University of Technology, China. He was sponsored by ERCIM 'Alain Bensoussan' Fellowship Programme to conduct postdoc research supervised by Prof. Fabio Martinelli at CNR, Italy. He was sponsored by China Scholarship Council as a joint PhD student supervised by Prof. Xiaodong Lin (IEEE Fellow). He is supported by CSC as a visiting scholar collaborating with Prof. Mauro Conti (IEEE Fellow) with HIT Center, University of Padua, Italy. His research interests include security, privacy, applied cryptography, blockchain, TEE, and Internet of Vehicles. He is also an associate editor for *IEEE Transactions on Information Forensics and Security*, *IEEE Transactions on Dependable and Secure Computing*, and *IEEE Transactions on Network and Service Management*. He was the recipient of 2024 IEEE HITC Award for Excellence (Early Career Researcher) and 2025 IEEE TCSVC Rising Star Award. He was selected into the IEEE Computer Society Computing's Top 30 Early Career Professionals for 2025.

Yan Qiao (Member, IEEE) received the PhD degree in computer science from the Beijing University of Posts and Telecommunications, in 2012. She was a Postdoctoral fellow with the School of Computer Engineering, Nanyang Technological University, Singapore. She is currently an associate professor with the School of Computer Science and Information Engineering, Hefei University of Technology, China. Her research interests include network monitoring, anomaly detection for time series, and artificial intelligence.



Guomin Yang (Senior Member, IEEE) received the PhD degree in computer science from the City University of Hong Kong in 2009. In 2022, he was an associate professor with the University of Wollongong, Australia. He is currently an associate professor with the School of Computing and Information Systems, Singapore Management University (SMU). His research focuses on applied cryptography and privacy enhancing technologies. In 2015, he was the recipient of Australian Research Council Discovery Early Career Researcher Award.

Mauro Conti (Fellow, IEEE) received the PhD degree from the Sapienza University of Rome, Italy, in 2009. He was a postdoc researcher with Vrije Universiteit Amsterdam, The Netherlands. He has been a visiting researcher with GMU, UCLA, UCI, TU Darmstadt, UF, and FIU. He is currently a full professor with the University of Padua, Italy. He is also affiliated with TU Delft and University of Washington, Seattle. His research focuses on the area of security and privacy.