VRFMS: Verifiable Ranked Fuzzy Multi-Keyword Search Over Encrypted Data

Xinghua Li, *Member, IEEE*, Qiuyun Tong, Jinwei Zhao, Yinbin Miao, Siqi Ma, Jian Weng, Jianfeng Ma, *Member, IEEE*, and Kim-Kwang Raymond Choo, *Senior Member, IEEE*

Abstract—Searchable encryption(SE) allows users to efficiently retrieve data over encrypted cloud data, but most existing SE schemes only support exact keyword search, resulting in false results due to minor typos or format inconsistencies of queried keywords. The fuzzy keyword search can avoid this limitation, but still incurs low search accuracy and efficiency. Besides, most of fuzzy keyword search schemes do not consider malicious cloud servers which may execute a fraction of search operations or forge some results due to various interest incentives such as saving computation or storage resources. To solve these problems, we propose an efficient and Verifiable Ranked Fuzzy Multi-keyword Search scheme, called VRFMS. VRFMS uses locality-sensitive hashing and bloom filter to implement fuzzy keyword search, and employs Term Frequency-Inverse Document Frequency(TF-IDF) to sort the relevant results. Aiming to further improve the search accuracy, we design an improved bi-gram keyword transformation method. Furthermore, the homomorphic MAC technique and a random challenge technique are utilized to verify the correctness and completeness of returned results, respectively. Formal security analysis and empirical experiments demonstrate that VRFMS is secure and efficient in practical applications, respectively.

Index Terms—Searchable encryption, ranked fuzzy multi-keyword search, locality-sensitive hashing, verifiability, homomorphic MAC

1 Introduction

More and more users outsource their local data to the clouds to enjoy the convenience and flexibility of cloud computing. However, it inevitably leads to security and privacy concerns as the clouds and users are not in the same trusted domain. The most effective way of solving this problem is to encrypt the data before uploading them to the cloud, but makes the retrieval over encrypted data a challenging task. The SE technique [1] which allows users to search encrypted files of interest has gained increasing attention from both academic and industrial fields. Various SE schemes have been proposed based on different demands and preconditions [2], [3], [4], [5], [6], [7], [8], [9], [10], [11], [12], [13], but these schemes only support exact keyword search rather than fuzzy keyword search. Thus, the fuzzy keyword search dealing with typos or approximate queries has been extensively studied [14], [15], [16], [17], [18], [19].

Existing fuzzy keyword search schemes are mainly divided into two architectures: 1) using edit distance; 2) using locality-sensitive hashing (LSH) and bloom filter (BF). For the former, a set of fuzzy keywords that are similar to original keywords are constructed by using wildcard and edit distance. However, the edit distance-based architecture has many disadvantages, such as non-supporting ranked multi-keyword search and low search efficiency. When users make search queries, it is unrealistic to return all ciphertexts meeting the demands. For the latter, users generate the bloom filter for each file via LSH. The bloom filters are encrypted as the indexes and then outsourced to the cloud server. This architecture outperforms the first one as it can support fuzzy multi-keyword search efficiently, but its search accuracy is at most 87%. In addition, these schemes only use the term frequency to sort the returned results. Therefore, it is very necessary to provide a ranked fuzzy keyword search scheme with high accuracy.

Besides, most SE schemes [2], [3], [4], [5], [6], [7], [8], [9], [10], [11], [14], [15], [16], [17], [18], [19] assume that cloud servers are honest-but-curious, which is not always true in

- *Xinghua Li is with the State Key Laboratory of Integrated Service Networks, School of Cyber Engineering, Xidian University, Xi'an 710071, China, and also with Engineering Research Center of Big data Security, Ministry of Education, Xi'an 710071, China. E-mail: xhli1@mail.xidian.edu.cn.*
- *Qiuyun Tong and Jinwei Zhao are with the State Key Laboratory of Integrated Services Networks, School of Cyber Engineering, Xidian University, Xi'an 710071, China. E-mail: qytong0820@163.com, jinweizhao@stu.xidian.edu.cn.*
- *Yinbin Miao is with the School of Cyber Engineering, Xidian University, Xi'an 710071, China. E-mail: ybmiao@xidian.edu.cn.*
- *Siqi Ma is with the School of Information Technology and Electrical Engineering, University of Queensland, QLD 4072, Australia. E-mail: xdmasiqi@hotmail.com.*
- *Jian Weng is with the College of Information Science and Technology, Jinan University, Guangzhou 510632, China. E-mail: cryptjweng@gmail.com.*
- *Jianfeng Ma is with the School of Cyber Engineering, Shaanxi Key Laboratory of Network and System Security, Xidian University, Xi'an 710071, China. E-mail: jfma@mail.xidian.edu.cn.*
- *Kim-Kwang Raymond Choo is with the Department of Information Systems and Cyber Security, The University of Texas at San Antonio, San Antonio, TX 78249 USA. E-mail: raymond.choo@fulbrightmail.org.*

Manuscript received 29 June 2020; revised 13 Aug. 2021; accepted 30 Dec. 2021. Date of publication 4 Jan. 2022; date of current version 6 Feb. 2023.
This work was supported in part by the National Natural Science Foundation of China under Grants 62125205, U1708262, U1736203, and 62072361, in part by the Key Research and Development Program of Shaanxi under Grant 2021ZDLGY05-04, in part by the Fundamental Research Funds for the Central Universities under Grant JB211505, and in part by the Guangxi Key Laboratory of Cryptography and Information Security under Grant GCIS201917, and the work of Kim-Kwang Raymond Choo was supported only by the Cloud Technology Endowed Professorship.
(Corresponding author: Yinbin Miao.)
Recommended for acceptance by B. Carminati.
Digital Object Identifier no. 10.1109/TSC.2021.3140092

actual scenarios. The malicious cloud servers may execute a fraction of search operations or forge some results due to various interest incentives such as saving computation and storage resources, as many schemes [12], [13], [20], [21], [22], [23], [24], [25], [26], [27], [28], [29] assume. Moreover, these schemes provide some verification mechanism (e.g., Merkle hash tree, RSA accumulator, homomorphic MAC, auditing technology) to detect the malicious behaviors of cloud servers. However, most of these schemes just support correctness verification of search results and do not consider fuzzy search. Therefore, we need to design a solution that can both support fuzzy search and verify the correctness[1] and completeness[2] of the results.

To solve these problems, we propose an efficient and Verifiable Ranked Fuzzy Multi-keyword Search scheme VRFMS. We first improve the existing LSH and BF architectures to achieve the high accuracy of fuzzy keyword search. Subsequently, we introduce the homomorphic MAC and a random challenge technique to verify the correctness and completeness of returned results, respectively. Specifically, the main contributions of this paper are summarized as follows:

1) Our scheme VRFMS uses LSH and TF-IDF to support fuzzy multi-keyword search, which enables the cloud server to return most relevant documents. In addition, our scheme VRFMS designs an improved bi-gram keyword transformation method considering the appearance of the same bi-gram to further improve the accuracy of fuzzy keyword search. The highest accuracy can reach 91%.
2) Our scheme VRFMS utilizes Homomorphic MAC and a random challenge technique to ensure the correctness and completeness of search results. In comparison to previous schemes using Merkle hash tree or accumulator structure, our scheme VRFMS uses Homomorphic MAC to ensure the correctness of similarity computation result apart from the correctness of outsourced indexes and ciphertexts. Besides, our scheme supports more efficient result verification [25].
3) We proved that VRFMS is secure under both known ciphertext model and known background model through rigorous security analysis. We implement and evaluate VRFMS using a real-world dataset. The results demonstrate that VRFMS has higher efficiency in search and verification phases, and has higher search accuracy when compared with existing schemes.

The remainder of this paper is organized as follows. In Section 2, we review the existing work related to fuzzy keyword search and verifiable keyword search. In Section 3, some preliminaries used in VRFMS are provided. Then the system model, threat model and design goals in Section 4. The main idea, algorithm definitions, and concrete construction of VRFMS are presented in Section 5. Security analysis and experiment performance are shown in Section 6. Finally, we conclude this paper in Section 7.

1. The correctness means the results should not be forged or tampered by malicious servers.

2. The completeness denotes that the returned top-k results are exactly the top-k most relevant results over all encrypted data.

2 Related Work

Song *et al.* [1] first proposed SE, which effectively solved the problem of searching over encrypted data and ensured that the data privacy was not leaked. Since then, researchers have proposed various SE schemes according to different demands and preconditions. To be consistent with the discussed topic, we just introduce the related work about fuzzy keyword search and verifiable keyword search.

2.1 Fuzzy Keyword Search

Most existing SE schemes only support exact keyword search, and cannot return expected results when users misspell the words or only remember the approximate wording of keywords. So far only part of researches have been carried out on fuzzy keyword search. Li *et al.* [14] proposed the first solution to support fuzzy keyword search. This scheme used edit distance to indicate the relevance of keywords and utilized wildcard $*$ to construct fuzzy keyword sets. For example, a fuzzy keyword set with an edit distance of 1 from "cat" is $S_{CAT,1} = \{CAT, *CAT, *AT, C*AT, C*T, CA*T, CA*, CAT*\}$. But this solution only supports fuzzy single-keyword search, which plays a very limited role in solving typos and formatting errors. Besides, this scheme incurs high storage overhead and low search accuracy. Kuzu *et al.* [15] used minhash to transform keywords, and then utilized jaccard distance to determine the similarity between keywords to avoid the limitation in the fuzzy single-keyword search. However, none of above solutions can sort the returned results, which are still unsuitable for the actual production environment. Although the scheme [30] designed a two-factor ranking function combining keyword weight with keyword morphology similarity to rank search results, it cannot avoid the limitation of single-keyword search.

Wang *et al.* [16] proposed MFSE (Multi-keyword Fuzzy Search over Encrypted data), using the architecture of LSH and BF. This scheme can sort the returned results without predefining fuzzy keyword sets. Fu *et al.* [18] improved scheme [16] by introducing porter stemming algorithm and keyword transformation algorithm based on the uni-gram model, which improves the fuzzy search accuracy greatly. However, only the term frequency is used to sort the results, which leads to inaccurate sorting results and high computation overhead. Based on the architecture of LSH and BF, Zhong *et al.* [19] constructed a balanced binary tree for the index and proposed an algorithm to search top-k results, which improves the search accuracy.

2.2 Verifiable Keyword Search

Most existing SE schemes also do not support result verification due to the existence of malicious cloud servers, which ultimately leads to data privacy leakage. Chai *et al.* [31] implemented a verifiable SE scheme for the first time, using tree-based indexing and hash chain technology. Kurpsawa *et al.* [32] proposed a verifiable SE scheme that can achieve UC(Universal Composability) security. It can verify whether returned results are modified or deleted, but it incurs high verification overhead. Jiang *et al.* [33] proposed

TABLE 1
Functionality Comparison Between Our Scheme and Previous Schemes

Scheme	VPSearch[21]	scheme [26]	VDFS [27]	VFKS [28]	VESFS [29]	VRFMS
Multi-Keyword Search	✓	✗	✗	✗	✗	✓
Architecture of Fuzzy Search	-	edit distance	edit distance	edit distance	edit distance	LSH and BF
Dynamic Update	✓	✗	✓	✗	✗	✓
Verification Correctness	✓	✓	✓	✓	✓	✓
Verification Completeness	✓	✗	✗	✗	✗	✓

a verifiable multi-keyword retrieval scheme. They constructed a special data structure to achieve efficient retrieval and introduced relevance score to sort returned results. Li *et al.* [34] verified the correctness and completeness of encrypted search results by using the keyed-Hash Message Authentication Code (HMAC) and Paillier encryption. Wan *et al.* [21] improved the original homomorphic MAC and proposed the scheme VPSearch (Verifiable Privacy-preserving keyword Search). However, these researches only support the verification of exact keyword search. Tong *et al.* [25] also used the adapted homomorphic MAC to achieve result correctness verification, and it focused on encrypted image retrieval. Next, we will briefly introduce the existing verifiable fuzzy keyword search schemes.

Existing verifiable fuzzy keyword search schemes [26], [27], [28], [29] still use the edit distance-based architecture, rather than the architecture of LSH and BF. Wang *et al.* [26] first proposed a verifiable fuzzy search scheme. Zhu *et al.* [27] implemented a verifiable fuzzy search scheme by introducing the RSA accumulator. This scheme constructs RSA accumulators for encrypted data and indexes to achieve the verification of returned results, which not only achieves the UC-security [35] but also supports dynamic updates. Ge *et al.* [28] generated a verification label for each fuzzy keyword in advance, but it just verifies the correctness of returned results. Huang *et al.* [29] used RSA accumulators to verify the correctness of the returned results, and designed a challenge-response mechanism to improve the verification efficiency, but it still cannot verify the completeness of returned results.

Hu *et al.* [36] showed that the existing verifiable SE schemes do not have a universal verification scheme for different demands, and there is no efficient punishment. Therefore, by introducing blockchain and smart contract, they ensure the results are correct and immutable. So users do not need to perform additional verification. However, due to high overhead and expensive cost, it still cannot be used in the actual production environment. Table 1 summarizes existing verifiable schemes in terms of various functionalities.

3 Preliminaries

LSH [16] and BF [18] are used in VRFMS to implement fuzzy search, and the TF-IDF rule [2] is introduced to rank queried results. These three techniques will be presented first. Then, the labelled program [21] and homomorphic MAC [21] used in the verification phase are introduced.

1) *Locality-Sensitive Hashing(LSH)* [16]: LSH is an algorithm for solving the approximate or exact near neighbor search in high dimensional spaces. LSH hashes input items so that similar items are mapped to the same buckets with high probability. A hash function family H is $(\mathrm{r}_1, \mathrm{r}_2, \mathrm{p}_1, \mathrm{p}_2)$-sensitive if any two points x, y and $\mathrm{h} \in H$ satisfy:

$$\text{if } \mathrm{d}(\mathrm{x},\mathrm{y}) \le \mathrm{d}_1, \mathrm{P}(\mathrm{h}(\mathrm{x}) = \mathrm{h}(\mathrm{y})) \ge \mathrm{p}_1;$$
$$\text{if } \mathrm{d}(\mathrm{x},\mathrm{y}) \ge \mathrm{d}_2, \mathrm{P}(\mathrm{h}(\mathrm{x}) = \mathrm{h}(\mathrm{y})) \le \mathrm{p}_2,$$

where $\mathrm{d}(\mathrm{x},\mathrm{y})$ is the distance between x and y, P is the probability that different items are hashed to the same value.

2) *Bloom Filter(BF)* [18]: A bloom filter consists of a fairly long binary vector and a series of hash functions. A bloom filter is a binary vector with m bits, all of which are set to 0 first. When adding an item to the bloom filter, it uses l independent hash functions to insert it into the bloom filter by setting its corresponding bits to 1. To check whether an item is in the bloom filter, it uses l independent hash functions to calculate and get l values. If any position is 0 in the l positions corresponding to the l values, this item does not belong to the set; otherwise, this item belongs to the set or it is false positive. The BF incurs less space and search overhead, but has a certain false recognition and difficulty in deleting items. Fig. 1 illustrates a simple example of it.

3) *Term Frequency-Inverse Document Frequency (TF-IDF)* [2]: It is actually the product of the TF value and the IDF value. It is a statistical method used to evaluate the importance of a word in a file collection. It is often used as a weighting factor in information retrieval and text mining. There is a hypothesis that the most meaningful words for distinguishing documents should be those that appear frequently in a single document and less frequently in other documents in the entire file collection. The value of TF-IDF is shown in Eq. (1). f_{w,F_i} is the number of times the keyword w appears in the file F_i, $|F_i|$ is the total number of keywords in the file F_i, $|F|$ is the total number of file in the file collection F, and $\left|\{j : w \in F_j\}\right|$ is the total number of file collection containing the keyword w.

$$\begin{aligned} Score_{w,F_i} &= TF_{w,F_i} \times IDF_w \\ &= \frac{f_{w,F_i}}{|F_i|} \times \log \frac{|F|}{\left|\{j : w \in F_j\}\right| + 1}. \end{aligned} \tag{1}$$

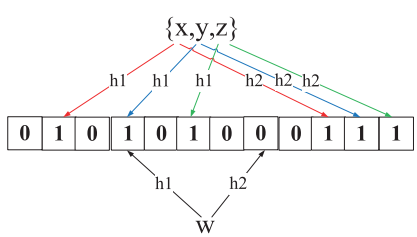


Fig. 1. A simple example of bloom filter. We assume m=12, l=2, {h1,h2} are two hash functions and {x,y,z} is a set. The user wants to query the keyword w, and the result shows that w does not exist in the set.

4) *Labelled Program* [21]: The labelled program is composed of a function f and n input variables with each input assigned a label L_i. L_i is a unique string to label the variable of the function f. For example, a class outsources its students' report cards R to the cloud server and then asks the cloud server to compute f as the average score of all students. Then L_i can be constructed as ("the ith student's score"), where i is the index of a student. Then the score s_i can be authenticated with respect to the label L_i, which essentially binds the data m with the corresponding label.

5) *Homomorphic MAC* [21]: Any message m_i is encrypted as a degree-1 polynomial $\mathrm{y}(x)$, namely $\mathrm{y}(0) = m_i$ and $\mathrm{y}(\alpha) = \gamma_i$, where α and γ_i are only known to the verifier. That is,

$$\mathrm{y}(x) = m_i + (\gamma_i - m_i) \cdot x/\alpha \tag{2}$$

is established. Since the cloud server needs to perform some basic arithmetic operations when calculating function f, homomorphic encryption needs to be introduced for direct calculation over encrypted data. Then the verifier can verify Eq. (4) on condition that Eq. (3) is established. Thus, this technology can verify whether the cloud server honestly executes user-defined search algorithms.

$$f(\gamma_1, \ldots, \gamma_n) = y(\alpha). \tag{3}$$

$$f(m_1, \ldots, m_n) = y(0). \tag{4}$$

Since the traditional homomorphic MAC only supports the calculation of a finite field, VRFMS uses the RealHomMAC proposed in paper [21]. The improvement of RealHomMAC is to treat all messages as real numbers encoded by a format like the double-precision floating point format defined in IEEE 754 standard. That is, the new algorithm can handle real numbers while retaining the homomorphism of the original algorithm. The concrete algorithms of RealHomMAC can be referred to the reference [21].

4 Problem Formulation

We formulate the system model, threat model, and design goals of VRFMS, then introduce some preliminaries.

4.1 System Model

The system model of VRFMS consists of four entities, namely data owner, data users, proxy server, and cloud server, as shown in Fig. 2. Specifically, the role of each entity is shown as follows:

1) *Data owner*. The data owner is responsible for generating the secret key and managing data users' query permissions.
2) *Data users*. Data users make search queries based on the queried keywords.
3) *Proxy server*. The proxy server is responsible for generating verifiable indexes, encrypting files, verifiable trapdoors, and then verifying the correctness and completeness of returned results.
4) *Cloud server*. The cloud server executes search operations and returns top-k query results and corresponding proofs.

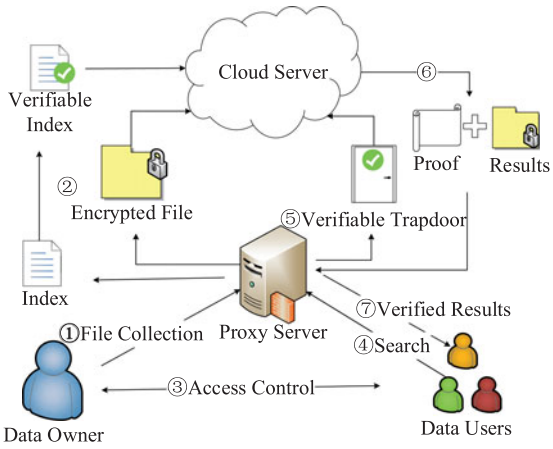


Fig. 2. System architecture of VRFMS.

In our system model, the data owner has a file collection $F = \{f_1, f_2, \ldots, f_n\}$. He needs to encrypt these files and then outsource them to the cloud server. Because the data owner's computing power is limited, the file collection will be transferred to the proxy server(Step ①), which will be encrypted as file ciphertext set C. To efficiently retrieve the encrypted data, the proxy server constructs secure and retrievable index set I. To verify results returned by the cloud server, the proxy server will generate verification tags based on the encrypted indexes. Then encrypted data C and verifiable indexes will be outsourced to the cloud server(Step ②). Only data users authorized by the data owner can communicate with the proxy server directly(Step ③). When a certain data user wants to retrieve the relevant files based on the queried keywords, he sends the queried keywords to the proxy server(Step ④). The proxy server generates a verifiable trapdoor Q and sends Q to the cloud server(Step ⑤). After receiving the trapdoor Q, the cloud server retrieves top-k search results according to Q and generates one proof for each search result (Step ⑥). Then the cloud server sends the top-k search results and their proofs to the proxy server. The proxy server verifies whether the results are valid. If they are invalid, it is considered that the cloud server has malicious behavior and the proxy server will reject the results. Otherwise, it means that the cloud server honestly computes over all encrypted data, and the proxy server returns the results to the data user(Step ⑦). Note that we assume that the file updating does not introduce new meaningful keywords.

4.2 Threat Model

In our system architecture, data owner, authorized data users, and proxy server are assumed to be fully trusted in the entire process. Different from the traditional honest-but-curious cloud servers[3], the cloud server in VRFMS is malicious, which may execute a fraction of search operations or forge some results due to various interest incentives such as saving storage and computation resources.

As analyzed in existing SE schemes, there are two kinds of threat models according to the knowledge available to the cloud server:

3. The cloud server honestly executes established protocols but may be curious to deduce some sensitive information.

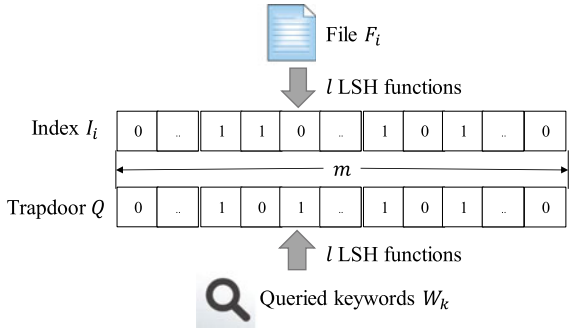


Fig. 3. The main idea of LSH and BF architecture. File F_i is transformed to the m-bit index I_i by using l LSH functions, and queried keywords W_k are converted to trapdoor Q using the same l LSH functions. The inner product between index I_i and trapdoor Q can be considered the correlation between a certain file F_i and the queried keywords W_k. The higher the inner product, the more relevant the file F_i is to the queried keywords W_k.

1) Known ciphertext model: the cloud server only knows the encrypted files and the encrypted indexes;
2) Known background model: the cloud server also knows some additional background knowledge, e.g., the cloud server may record the search results corresponding to each trapdoor, and statistics of the outsourced documents or relationship between different trapdoors.

4.3 Design Goals

VRFMS is designed to achieve the following goals:

1) *Verification*. VRFMS should verify both the correctness and completeness of returned results.
2) *Multi-keyword Search*. VRFMS should support fuzzy multi-keyword search over encrypted data, which can not only improve the accuracy of fuzzy search but also increase user satisfaction.
3) *Ranking*. The returned results should be sorted according to the relevance scores, which can reduce the time overhead caused by decrypting irrelevant information.
4) *Privacy-Preserving*. The cloud server cannot obtain any plaintext information from the encrypted data, encrypted indexes, and encrypted trapdoors generated during each query.
5) *Efficiency*. In VRFMS, the proxy server should be able to efficiently verify the results returned by the cloud server, and the cloud server should be able to efficiently execute the search process.

5 Proposed VRFMS

In this section, we present the construction of VRFMS. We first give the main idea and the algorithm definitions of VRFMS, then introduce each algorithm in detail. Finally, we discuss the specific methods used in VRFMS to improve the accuracy of fuzzy search and implement completeness verification.

5.1 Main Idea

Our VRFMS is built on top of the LSH and BF architecture. The main idea of this architecture is shown in Fig. 3. In this architecture, a file F_i is transformed to an index I_i. The index I_i, containing all the keywords in F_i, is a m-bit BF. To support fuzzy multi-keyword search, VRFMS first transforms the keywords to a bi-gram based vector and then uses LSH functions to insert the keywords to index I_i. Thus, the inner product between index I_i and trapdoor Q can represent the correlation between a certain file F_i and the queried keywords W_k. For a certain query, the cloud server ranks the results in terms of the inner product between each index I_i and trapdoor Q, and returns the top-k results to the data user. Therefore, VRFMS implements the ranked fuzzy multi-keyword search.

The verifiability of VRFMS is based on the homomorphic MAC technique. Let M be the size of the encrypted index. Each element i_j in the index $I_i = (i_1, \ldots, i_M)$ is encoded as a degree-1 polynomial $y_{i,j}(x)$ such that $y_{i,j}(0) = i_j, y_{i,j}(\alpha) = r_{i,j}$, where $\alpha, r_{i,j}$ are secrets known only to the verifier. In this way, the authentication tag for the index I_i is a polynomial vector $\sigma_{I_i} = (y_i^{(1)}, \ldots, y_i^{(m)})$, where $y_i^{(j)} = (i_j, (r_{i,j} - i_j)/\alpha)$ is composed of the coefficients of corresponding polynomial $y_{i,j}(x)$ and we set $\vec{r}_{I_i} = (r_{i,1}, \ldots, r_{i,M})$. Similarly, the authentication tag for the trapdoor Q is $\sigma_Q = (y_Q^{(1)}, \ldots, y_Q^{(M)})$, where $y_Q^{(j)} = (q_j, (r_{Q,j} - q_j)/\alpha)$ is composed of the coefficients of corresponding polynomial $y_{i,j}(x)$ and we set $\vec{r}_Q = (r_{Q,1}, \ldots, r_{Q,M})$. As elementary arithmetic operations over polynomials are homomorphic and the fuzzy keyword search function f is actually the inner product composed of addition and multiplication operations, the cloud server can perform inner product of the index's authentication tag and trapdoor's authentication tag to obtain a 2-degree polynomial $g(x) = y_0 + y_1 x + y_2 x^2$, i.e., $f(\sigma_{I_i}, \sigma_Q) = (y_0, y_1, y_2)$. Thus, we can use $f(I_i, Q) = g(0)$, $f(\vec{r}_{I_i}, \vec{r}_Q) = g(\alpha)$ to verify whether the cloud server honestly executes inner product of correct indexes and correct trapdoor. A random challenge technique based on ranking is proposed to greatly improve the efficiency of completeness verification.

5.2 Algorithm Definitions

VRFMS consists of seven polynomial-time algorithms (KeyGen, BuildIndex, Trapdoor, Auth, Search, Verify, Update).

1) KeyGen$(1^\lambda, m) \rightarrow SK$ is a probabilistic key generation algorithm run by the data owner. It takes a random secure parameter λ and index length m as input, and outputs a secret key set $SK = (K, \alpha, M_1, M_2, S)$.
2) BuildIndex$(F, SK) \rightarrow Enc_{sk}(I)$ is an index building algorithm run by the proxy server. This algorithm takes the file collection F and secret key SK as input, and outputs encrypted indexes $Enc_{sk}(I) = \{Enc_{sk}(I_1), \ldots, Enc_{sk}(I_n)\}$, n is the number of files in the file collection.
3) Trapdoor$(W_k, SK) \rightarrow Enc_{sk}(Q)$ is a trapdoor generation algorithm run by the proxy server. This algorithm takes the collection of queried keywords $W_k = w_1, \ldots, w_k$ and secret key SK as input, and outputs the encrypted trapdoor $Enc_{sk}(Q)$.
4) Auth$(SK, L, Enc_{sk}(I_i)\ or\ Enc_{sk}(Q)) \rightarrow \sigma_{I_i}$ or σ_Q is an authentication tag generation algorithm run by the proxy server. It takes the secret key SK, string L and encrypted index or encrypted trapdoor as input, and outputs the authentication tag σ_{I_i} or σ_Q.

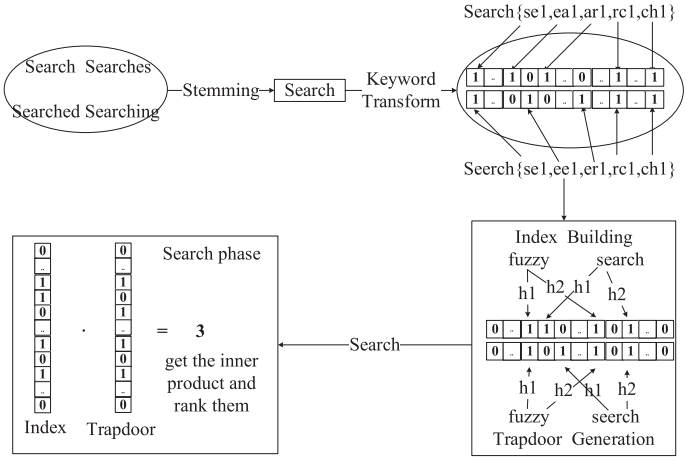


Fig. 4. The architecture of LSH and BF. It mainly represents the index building phase and searching phase. It can be divided into stemming keyword, keyword transformation, index building or trapdoor generation and search.

5) Search$\left(f, \sigma_I, \sigma_Q\right) \rightarrow \sigma$ is a deterministic fuzzy keyword search algorithm run by the cloud server. It takes fuzzy keyword search function f, index tag set σ_I and trapdoor tag σ_Q as input, and outputs the results $\sigma = (\sigma_0, \ldots, \sigma_k)$.
6) Verify$(SK, \mathcal{P}, sco_i, \sigma_i)$ is a verification algorithm run by the proxy server. It takes the secret key SK, fuzzy keyword search function f, message ms as well as results σ, and outputs *accept* or *reject*.
7) Update$\left(F_{temp}, SK\right)$. This algorithm takes the file to be updated F_{temp} and secret key SK as input. This algorithm consists of three operations such as file addition, file deletion and file modification.

5.3 Concrete Construction

In this section, we will introduce each algorithm in VRFMS.

1) KeyGen$(1^\lambda, m)$: According to the security parameters λ, the data owner invokes $\mathrm{RealHomMac.KeyGen}\left(1^\lambda\right)$ to generate the secret key (K, α). According to the index length m, two invertible matrices $(M_1, M_2) \in R^{(m+2)\times(m+2)}$ and $S \in \{0,1\}^{m+2}$ are generated randomly. The final secret key is $SK = (K, \alpha, M_1, M_2, S)$, and will be sent to the proxy server.
2) BuildIndex(F, SK): The index building algorithm includes the following steps:
   - Data preprocessing: For a given file collection $F = \{f_1, f_2, \ldots, f_n\}$, the data owner directly passes it to the proxy server. The proxy server first extracts keywords from F to form a keyword set $K = \{k_1, k_2, \ldots, k_N\}$. Then it uses the Porter stemming algorithm [37] to ascertain the root of the words, and gets the stemming keyword set $ST = \{st_1, st_2, \ldots, st_N\}$. This step corresponds to the stemming part in Fig. 4. The proxy server calculates the TF and IDF values of the keyword according to ST later, so as to analyze the relevance of the queried keywords and the files.
   - Keyword transformation: The proxy server first uses the improved keyword transformation method to transform each keyword into a bi-gram set BS, where the appearance of the same bi-gram is considered. For example, the bi-gram set BS of the keyword "$represent$" is $BS = \{re_1, ep_1, pr_1, re_2, es_1, se_1, en_1, nt_1\}$. Then, proxy server transforms the set BS into a fixed-length binary vector BV, as shown in the keyword conversion stage in Fig. 4.
   - Construction of BF – based index: For each document f_i, the proxy server first generates a m-bit bloom filter I_i and initializes each bit to 0. Then, for each keyword $w_j \in f_i$, using the bi-gram vector BV_j as the input of l LSH functions, the proxy server calculates corresponding positions in I_i and sets each of them to $TF_{i,j}/l$, where $TF_{i,j} = 1 + |w_j|/|f_i|$ is the w_j's term frequency in the document f_i, $|w_j|$ is the number of keyword w_j in f_i, and $|f_i|$ is the number of total keywords in f_i. Note that if different keywords are hashed into the same position, we use their average value as the insert, as the scheme [18] does. For example, if the document f_i is composed of keywords w_1, w_2 with $TF_{i,1} = 1.4, TF_{i,2} = 1.6$ and $l = 2$ LSH functions are used to map the two keywords' bi-gram vectors to positions 1,6 and 3,6, respectively, then the TF values in positions 1,3,6 are $TF_{i,1}/2 = 0.7, TF_{i,2}/2 = 0.8, (TF_{i,1}/2 + TF_{i,2}/2)/2 = 0.75$, respectively. The effect is shown in the index building part of Fig. 4, but it is directly set to 1 for simplicity. Finally, I_i is expanded to (m+2)-bit, and the expansion values are ε_1 and 1.
   - Index encryption: The index vector I_i is a vector of length m+2. The proxy server will encrypt it to I_i' and I_i'' to protect the index privacy. The encryption follows the rule: set $i_j' = i_j'' = i_j$ if $s_j \in S$ is 1; otherwise $i_j' = \frac{1}{2} i_j + r, i_j'' = \frac{1}{2} i_j - r$, where r is a random number. Then the proxy server encrypts I_i' and I_i'' into $\left\{M_1^T \cdot I_i', M_2^T \cdot I_i''\right\}$.

Example. We show the index building process by an instance. For simplicity, we assume a file has two keywords "ranked" and "searched", namely K ={ranked, searched}. We set m = 10, S = [1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1] and assume (M_1, M_2) are all identity matrixes. K is first stemmed to ST = {rank, search}. The keyword "rank" is transformed to the bi-gram set $BS_1 = \{\mathrm{ra}_1, an_1, nk_1\}$, and the 14th position of BV_1 is set to 1 as the item an_1 exists in BS_1. BS_2 and BV_2 are generated for the keyword "search" in the same method. Then, we use BV_1 and BV_2 as two inputs of LSH functions respectively, and set calculation results' positions in I_i to its keyword's TF value 0.5. Thus we have I_i = [0.5, 0, 0, 0, 0, 0.5, 0, 0.5, 0, 0.5], and it is expanded to I_i = [0.5, 0, 0, 0, 0, 0.5, 0, 0.5, 0, 0.5, 0.5, 1] using values 0.5 and 1. Subsequently, I_i is split to I_i'' as well as I_i'' by using S, and I_i' = [0.5, 0.1, 0.1, 0, 0.1, 0.35, 0, 0.5, 0.1, 0.35, 0.5, 1], I_i'' = [0.5, -0.1, -0.1, 0, -0.1, 0.15, 0, 0.5, -0.1, 0.15, 0.5, 1]. Finally, I_i' and I_i'' are encrypted with the matrices (M_1, M_2), and the final result is $Enc_{sk}(I_i) = \left\{M_1^T \cdot I_i', M_2^T \cdot I_i''\right\}$ = [0.5, 0.1, 0.1, 0, 0.1, 0.35, 0, 0.5, 0.1, 0.35, 0.5, 0.5, -0.1, -0.1, 0, -0.1, 0.15, 0, 0.5, -0.1, 0.15, 0.5, 1].

3) Trapdoor(W_k, SK): The trapdoor generation algorithm includes the following steps:

- Keyword transformation: It is exactly the same as the first two steps of index building, preprocessing keywords and transforming them to the bi-gram based vector BV.
- Construction of BF – based trapdoor: The proxy server first generates an m-bit bloom filter Q and initializes each bit to 0. Then, the proxy server uses l LSH functions to map each queried keyword's bi-gram vector into the bloom filter Q and set the corresponding positions of Q to its IDF value. Finally, the proxy server multiplies the trapdoor vector Q by ε_2, and expands it to a (m+2)-bit vector by expanding ε_2 and t.
- Trapdoor encryption: The trapdoor vector Q is a vector of length m+2. The proxy server will encrypt it to Q' and Q'' to protect trapdoor privacy. The encryption follows the rule: set $q'_j = q''_j = q_j$ if $s_j \in S$ is 0; otherwise $q'_j = \frac{1}{2}q_j + r, q''_j = \frac{1}{2}q_j - r$, where r is a random number. Then the proxy server encrypts Q' and Q'' into $\{M_1^{-1} \cdot Q', M_2^{-1} \cdot Q''\}$.

Example. We show the trapdoor generation process by an instance. We assume the queried keywords are "ranked" and "seerch". $\{BS_1, BV_1, BS_2, BV_2\}$ are generated using the stemmed keywords {rank, seerch} by the same method in the previous example. Then, we use BV_1 and BV_2 as two inputs of LSH functions respectively, and set calculation results' positions in Q to its keyword's IDF value log(0.5)=-0.3. Then Q = [0, -0.3, 0, 0, 0, -0.3, 0, -0.3, 0, -0.3], and it is expanded to Q = [0, -0.3, 0, 0, 0, -0.3, 0, -0.3, 0, -0.3, 1, -0.1] using values 1 and -0.1. Subsequently, Q is split to Q' = [0.1, -0.3, 0, 0.1, 0, -0.3, 0.1, -0.05, 0, -0.3, 0.6, 0.05] and Q'' = [-0.1, -0.3, 0, -0.1, 0, -0.3, -0.1, -0.25, 0, -0.3, -0.6, -0.15] by using S. Finally, Q' and Q'' are encrypted to $Enc_{sk}(Q) = \{M_1^{-1} \cdot Q', M_2^{-1} \cdot Q''\}$ = [0.1, -0.3, 0, 0.1, 0, -0.3, 0.1, -0.05, 0, -0.3, 0.6, 0.05, -0.1, -0.3, 0, -0.1, 0, -0.3, -0.1, -0.25, 0, -0.3, -0.6, -0.15].

4) Auth$(SK, Enc_{sk}(I_i)$ *or* $Enc_{sk}(Q))$: This algorithm generates an authentication tag for each encrypted index $Enc_{sk}(I_i)$ or the encrypted trapdoor $Enc_{sk}(Q)$. Specifically, the proxy server first labels each item of $Enc_{sk}(I_i)$ or $Enc_{sk}(Q)$. For example, $Enc_{sk}(I_i)[j]$ is labeled as $L_{Enc_{sk}(I_i),j}$: "the jth item in the ith index I''_i; $Enc_{sk}(Q)[j]$ is labeled as $L_{Enc_{sk}(Q),j}$: "the jth item in the trapdoor Q''. Then, for each item $Enc_{sk}(I_i)[j]$ in the encrypted index $Enc_{sk}(I_i)$, the proxy server invokes RealHomMAC.Auth$\left(sk, L_{Enc_{sk}(I_i),j}, Enc_{sk}(I_i)[j]\right)$ to output $(y_0^{(j)}, y_1^{(j)}) = (Enc_{sk}(I_i)[j], (r_{i,j} - Enc_{sk}(I_i)[j])/\alpha)$, where $r_{i,j} = F_K(L_{Enc_{sk}(I_i),j})$. In this way, the proxy generates the authentication tag σ_{I_i} for the encrypted index $Enc_{sk}(I_i)$. Similarly, the proxy server generates the authentication tag σ_Q for the encrypted trapdoor $Enc_{sk}(Q)$.

5) Search$(Enc_{sk}(I_i), Enc_{sk}(Q), \sigma_{I_i}, \sigma_Q)$: For each encrypted index $Enc_{sk}(I_i)$ $(1 \le i \le n)$, the cloud server first computes the inner product of it and encrypted trapdoor $Enc_{sk}(Q)$ to obtain a relevance score sco_i, i.e., $sco_i = f(Enc_{sk}(I_i), Enc_{sk}(Q)) = Enc_{sk}(I_i)^\top \cdot Enc_{sk}(Q)$. Then, the cloud server selects k search results with top-k relevance scores. Finally, for each of these k search results such as f_i, the cloud server invokes RealHomMAC.Eval$\left(f, \{\sigma_{I_i}, \sigma_Q\}\right)$ to output its proof $\sigma_i = (y_0, y_1, y_2)$ in Fig. 5.

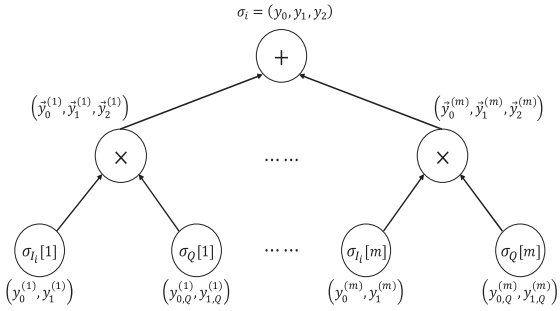


Fig. 5. Fuzzy keyword search function f combined with the RealHomMAC algorithm.

6) Verify$(SK, \mathcal{P}, sco_i, \sigma_i)$: Let $\mathcal{P} = (f, L_{Enc_{sk}(I_i)}, L_{Enc_{sk}(Q)})$. The proxy server invokes RealHomMAC.Ver$(SK, \mathcal{P}, sco_i, \sigma_i)$ to verify sco_i's correctness according to Eq. (5). If the Eq. (5) holds, the proxy server accepts f_i as a search result; otherwise it rejects f_i.

$$sco_i = y_0,$$
$$f(\vec{r}_{I_i}, \vec{r}_Q) = \sum_{j=0}^{2} y_j \alpha^j. \tag{5}$$

7) Update(F_{temp}, SK): This algorithm includes three operations: adding file, deleting file and modifying file. When adding a file, the proxy server encrypts the file, invokes Auth to generate an authentication tag for it, and finally outsources it to the cloud server. When deleting a file, the proxy server interacts with the cloud server to delete the corresponding ciphertext and index. As for file modification, it is equivalent to deleting the original file and then adding a new file.

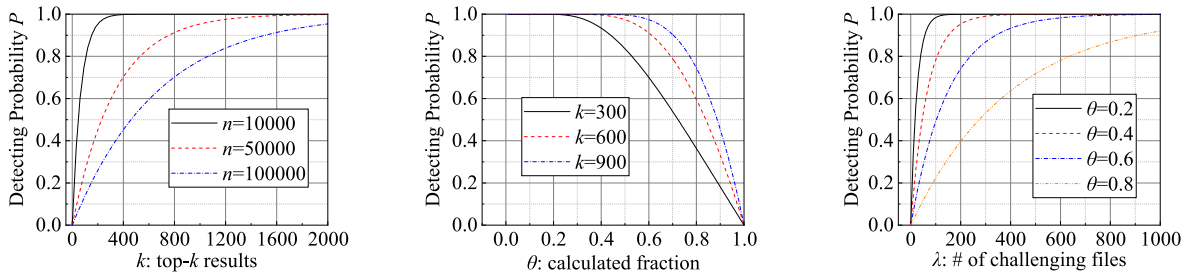
5.4 Discussion

This section introduces some discussions of VRFMS in terms of search accuracy and completeness verification of returned results.

5.4.1 Accuracy of Fuzzy Search

In the architecture of LSH and BF, if the original keyword and its fuzzy keyword are calculated with the same l LSH functions, the more the same results, the higher the final similarity score. Therefore, the most important factor that affects the accuracy of fuzzy keyword search in this architecture is the similarity between the original keyword and the fuzzy keyword. However, using the uni-gram method to transform keyword cannot reflect the order between keywords, e.g., keywords "eat" and "tea" will be transformed to the same vector using uni-gram, which will reduce the search accuracy.

Thus, we propose an improved bi-gram keyword transformation method as shown in Algorithm 1. Different from the traditional bi-gram keyword transformation method, the improved one considers the appearance of the same bi-gram. For example, keyword "$represent$" is transformed to



(a) P for $\lambda = 600, \theta = 0.8$. (b) P for $\lambda = 600, n = 50000$. (c) P for $k = 300, n = 30000$.

Fig. 6. P as a function, n, k, λ and θ are randomly changed.

the bi-gram set $\{re_1, ep_1, pr_1, re_2, es_1, se_1, en_1, nt_1\}$. In practice, the length of the vector can be determined by the actual needs of data users. So, the accuracy of fuzzy keyword search is improved by using the bi-gram based keyword transformation algorithm. With this method, even if the keyword is misspelled, the vector's distance between the correct spelling keyword and the misspelling one will be smaller. So, the hashing result will be the same, which will improve search accuracy. In VRFMS, we introduce the TF-IDF rule to sort the returned results, which can return the most relevant results.

Algorithm 1. Generate Bi-Gram Based Keyword Vector

Input: Stemming keyword set ST
Output: Bi-gram based vector set V

for each ST_i in ST **do**
  Set V_{ST_i} all the position to 0 in vector $\{0,1\}^{2*26^2}$;
  Compute st_i=$length(ST_i)$;
  generate a vector $\{y|y[j] = 1, 0 < j < st_i\}$;
  for each $ST_i[j, j+1]$ in ST_i **do**
    for $k = 1 : j-1$ **do**
      if $ST_i[k, k+1]$=$ST_i[j, j+1]$ **then**
        y[j]++;
      end if
    end for
    $ST_{Temp}[j]$=$ST_i[j, j+1]$+ $y[j]$;
  end for
  for each $ST_{Temp}[j]$ in ST_{Temp} **do**
    Set V_{ST_i} all corresponding positions to 1;
  end for
  add the vector V_{ST_i} for stemming keyword ST_i to V;
end for
return V=$V_{ST_i}|ST_i \in ST$ for all stemming keywords in set ST.

5.4.2 Completeness Verification

One of the most important parts of VRFMS is the result verification, which is mainly divided into two parts: correctness verification and completeness verification. The correctness verification is realized by the homomorphic MAC technology. However, this algorithm cannot verify the completeness of the returned results.

Since it is unrealistic for the cloud server to return all encrypted results when the data are fairly large, most solutions will sort the results and then return the top-k results, note that k is far less than the number of files. Assuming that the cloud server is malicious, it may only calculate part of the data, and return the top-k results of them. In this case, since the proxy server does not know all relevance scores, it is difficult to find this malicious behavior. In order to prevent it, the simplest method is to verify the relevance scores of other files. Only when the relevance scores of other files are lower than those of top-k results, the top-k results returned by the cloud server are considered correct and complete. Suppose that the data owner outsources n files to the cloud server, and the data user only wants to obtain top-k results when searching, where k is far less than n. The cloud server only detects a fraction of $\theta \in (0.1]$ part and then returns the top-k results. The proxy server randomly selects λ remaining files and asks the cloud server to return the relevance scores. Therefore, the probability of detecting the malicious behavior of the cloud server is

$$P = 1 - \left(1 - \frac{(1-\theta)k}{\theta(n-k)}\right)^{\lambda}. \tag{6}$$

According to Eq. (6), we plot the influence of various parameters on the probability P of detecting malicious behavior in Fig. 6. Fig. 7a shows that when $\lambda = 600, \theta = 0.8$, the detection probability P gradually increases with the increase of k. Fig. 7b shows when $\lambda = 600, n = 50000$, with the increase of θ, the detection probability P gradually decreases to 0. Fig. 7c shows that when $k = 300, n = 30000$, with the increase of λ, the detection probability P gradually increases.

As can be seen from Fig. 6, only if λ and k are large, the probability of detecting malicious behavior is higher than 90%, which is very inefficient. Therefore, we propose a random challenge technique based on ranking. The details are as follows: First, when the cloud server performs search operation, it sorts the identities of all accessible outsourced

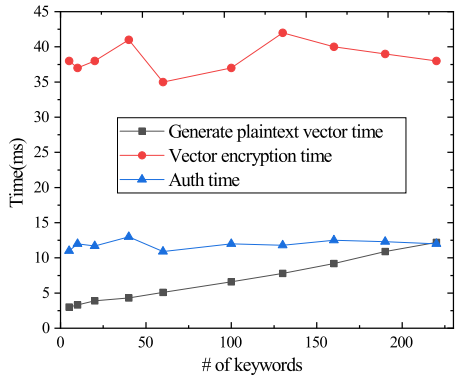


Fig. 7. Time cost of index building in a single file.

documents according to their relevance scores, and returns the sorted list along with the top-k search results. Then, the proxy server randomly selects λ files, and computes their relevance scores. If the relevance scores of these documents are all lower than those of the top-k results and the order of these documents' identities ranked according to their relevance scores is correct, it is convinced that the cloud server has executed the search algorithm honestly and the top-k results are complete with the probability $\mathrm{P} = 1 - (p\lambda)!/\lambda!$, where p is the faction of documents the cloud servers retrieves. It can be seen that the detection probability has nothing to do with the total number of files n and k, and it is only related to the part of calculated p and challenging samples λ. Let $\lambda = 10, p = 0.8$, the final detection probability is about 99.0%. Compared with Fig. 7b, under the same conditions, the detection probability is only 20%. Therefore, the random challenge technology based on ranking not only improves the detecting probability but also improves the efficiency of completeness verification. Due to the existence of detection technology, the cloud server will honestly execute users' queries and return integral results. That is, the random challenge technology based on ranking successfully implements the completeness verification of returned results.

6 Security and Performance Analysis

We first analyze the security of VRFMS, then evaluate the theoretical performance and actual performance.

6.1 Security

In this section, we proof our scheme is secure in both known ciphertext and known background knowledge. Before giving the proof, we introduce some notations.

1) $H(\mathrm{history}) = (F, I, W_k)$. F is a file collection, I is the index and $W_k = w_1, \ldots, w_k$ are the queried keywords.
2) *V(H)*= $\left(Enc_{sk}(F), \sigma_I, \sigma_{W_k}\right)$. *V(H)*(View) is the encrypted results of H with the secret key sk. The cloud server only knows *V(H)*.
3) *Tr(H)*. It is a set of traces of queried keywords $\{Tr(w_1), Tr(w_2) \cdots, Tr(w_k)\}$. Exactly, $Tr(w_i) = \left\{\left(f_j, s_j\right)_{w_i \subset f_j}, 1 \le j \le |F|\right\}$ is the relevance score between the file f_j and the trapdoor w_i.

Theorem 1. *Our scheme is secure under the known ciphertext model.*

Proof. In the known ciphertext model, given two histories with the same trace, if the cloud server cannot distinguish which of them is generated by the simulator, it cannot learn additional information from the *V(H)*.

We adopt a similar simulation-based proof used in paper [16]. Let S be a simulator that can simulate a view $V'(H)$ indistinguishable form a cloud server's view *V(H)*, then the S can conduct the following process:

1) S selects a random $f_i' \in \{0,1\}^{|f_i|}, f_i \in F, 1 \le i \le |F|$, and outputs $F' = \{f_i', 1 \le i \le |F|\}$.
2) S generates the secret key $sk' = \left\{K', \alpha', M_1', M_2', S'\right\}$ randomly by the security parameter m and λ.
3) To generate σ_I', S first generates a m-bit vector for each file as the index. Then S invokes BuildIndex algorithm to get $Enc_{sk'}\left(W_k'\right) = \left\{Enc_{sk'}\left(w_1'\right), \ldots, Enc_{sk'}\left(w_k'\right)\right\}$. Finally, S invokes Auth algorithm to get the final output σ_I'.
4) To generate σ_{W_k}', S generates w_i' by w_i, ensures that the number of 1s is same. So S gets $W' = \left\{w_i', 1 \le i \le k\right\}$, and then encrypts it to $Enc_{sk'}\left(W_k'\right) = \left\{Enc_{sk'}\left(w_1'\right), \ldots, Enc_{sk'}\left(w_k'\right)\right\}$. At last, S invokes Auth algorithm to generate σ_{W_k}'.
5) S outputs $V' = (F', \sigma_I', \sigma_{W_k}')$.

The secure indexes and trapdoors use the same method to generate as the one that the cloud server has. We claim that no probabilistic polynomial-time (P.P.T.) adversary can distinguish between the view $V'(H)$ and $V(H)$. Particularly, due to the semantic security of the symmetric encryption, no P.P.T adversary can distinguish between $Enc_{sk'}(F)$ and $Enc_{sk}(F)$. And the indistinguishability of indexes and trapdoors is based on the indistinguishability of the secure kNN encryption and the random number introduced in the index building and trapdoor generation phases.

Based on these aspects, we observe that VRFMS is secure in the known ciphertext model. This completes the proof of Theorem 1. □

In addition, if $F_K : \{0,1\}^* \to \mathbb{R}_\lambda$ is a pseudo-random function and the secret key (K, α) keeps in secret, RealHomMAC is secure against forgeries who may forge or temper authentication tag to return false results, which has been proved in the scheme [25]. The main idea is that each real number can be encoded into an integer in the finite field, and the scheme [38] has rigorously proved the security of HomMAC in finite fields if $F_K : \{0,1\}^* \to \mathbb{Z}_p$ is an pseudo-random function and (K, α) keeps in secret. Therefore, RealHomMAC is secure against forgeries if $F_K : \{0,1\}^* \to \mathbb{R}_\lambda$ is a pseudo-random function and the secret key (K, α) keeps in secret.

6.2 Performance

6.2.1 Theoretical Performance

This section shows the theoretical performance of VRFMS. In the following analysis, n is the number of files and m is the length of the index/trapdoor.

BuildIndex. This stage is mainly divided into two parts: generating plaintext index and encrypting index. The time cost required to generate plaintext index is mainly determined by the number of keywords contained in the file. The index encryption mainly depends on the matrix multiplication calculation, so its time complexity is $O(m^2 n)$. This stage only needs to be performed once, so it will not affect the running performance.

Trapdoor. This stage also mainly includes the matrix multiplication calculation, so its time cost is $O(m^2)$ for each query.

Auth. The time cost of generating the verification label is proportional to the index length m. So the time complexity of generating the verification label for each index or trapdoor is $O(m)$. This process only needs to be performed once, so it will not affect the running performance.

Search. This stage is to calculate the inner product of the trapdoor and all the indexes, thus its time complexity is $O\ mn$.

TABLE 2
Computation Complexities in Various Schemes

Scheme	scheme [18]	scheme [26]	VDFS [27]	VFKS [28]	VESFS [29]	VRFMS
Index Building Cost	$O(m^2n)$	$O(NM)$	$O(NM)$	$O(NM)$	$O(NM)$	$O(m^2n)$
Trapdoor Generation Cost	$O(m^2)$	$O(M)$	$O(M)$	$O(M)$	$O(M)$	$O(m^2)$
Search Cost	$O(mn)$	$O(M\log N)$	$O(M\log N)$	$O(M)$	$O(M\log N)$	$O(mn)$
Verification Cost	-	$O(k)$	$O(k)$	$O(k)$	$O(k)$	$O(mk)$

[1]*We denote n as the number of files in the cloud, N as the number of exact keywords, M as the maximum number of fuzzy keywords, m as the length of index in VRFMS, and k as the number of results to be verified.*

Verify. The time complexity of verifying the correctness of the top-k results is $O(mk)$. As the phase for verifying the completeness of the results needs to sort all the results, so its time complexity is $O(n\log n)$.

In Table 2, we perform the efficiency comparison in the following aspects: index building, trapdoor generation, search and verification. We denote n as the number of files in the cloud, N as the number of exact keywords, M as the maximum number of fuzzy keywords, m as the length of index in VRFMS, and k as the number of results to be verified. VRFMS's index building time complexity is $O(m^2n)$, those of other verifiable schemes are $O(NM)$. VRFMS's trapdoor generation time complexity is $O(m^2)$, those of other verifiable schemes are $O(M)$. VRFMS's search time complexity is $O(mn)$, those of other verifiable schemes are $O(M\log N)$. VRFMS's verification time complexity is $O(mk)$, those of other verifiable schemes are $O(k)$. m can be considered as a constant in VRFMS, and $n \ll N \ll M$. Therefore, VRFMS is more efficient than existing schemes in the phases of index building, trapdoor generation and search.

6.2.2 Experimental Performance

In this section, we estimate the overall performance of VRFMS using Java language on a 64-bit Windows 10 Home Chinese version server with Intel Core i7-8700 CPU running at 3.20GHz. We use the Request For Comment (RFC) dataset[4]. We randomly select 3000 files to form the dataset, and extract 65277 keywords in total. The maximum number of keywords in a single file is 3819, and the minimum is 113. We set the number of LSH functions $l = 30$, and the length of the index/trapdoor $m = 8000$. Similar to the original scheme, we randomly select one letter among the keywords and then replace it with another letter to construct fuzzy keywords.

This section mainly compares VRFMS with the existing verifiable fuzzy keyword search schemes [26], [28], [29] in terms of index building, fuzzy search, and verification.

Index Building. The single index/trapdoor generation in VRFMS mainly includes generating plaintext vector, encrypting vector, and generating verification tags for them. Fig. 7 shows that in a single index, as the number of keywords increases, the time cost of generating plaintext vector gradually increases, as this process needs to calculate the keywords individually. The time cost of the vector encryption and generation verification tags changes within a range as the number of keywords increases. This is because the time cost of these two stages is only related to the length of the vector m, regardless of the number of keywords in the file. There is no comparison with other existing schemes in this figure because the index generation method is not the same. Fig. 8 shows that when the program initializes to generate indexes for all files, as the number of files increases, the time cost of the entire process of index generation increases. As shown in this figure, VRFMS has the lowest time-consuming during the index generation stage. This is because the architecture of fuzzy keyword search is different. VRFMS's time-consuming index generation is mainly related to the number of files. However, those of other schemes are mainly related to the number of keywords over all files. And the time overhead of VESFS [29] is the largest, as it uses the RSA accumulator for verification.

Fuzzy Search. Since the existing verifiable fuzzy keyword search schemes only support fuzzy single-keyword search, so we use the fuzzy single-keyword search to conduct comparative experiments. It can be seen from Fig. 9 that the time required for the search stage increases linearly with the number of files in the cloud server. VRFMS is optimal in efficiency, and when the number of files is 3000, the search stage takes only 3.5 seconds, which is far lower than those of other schemes. The time cost of the scheme [26] and VFKS [28] are

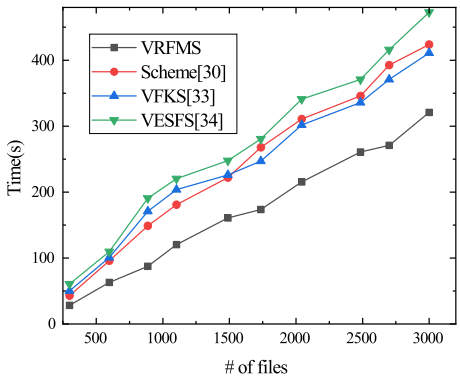


Fig. 8. Time cost of index building in file set.

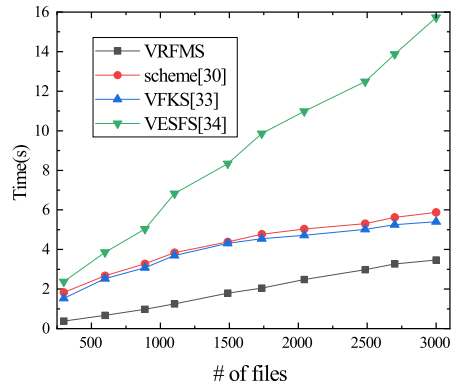


Fig. 9. Search time of different schemes.

4. RFC dataset contains almost all important information about the Internet. Available: http://www.ietf.org/rfc.html.

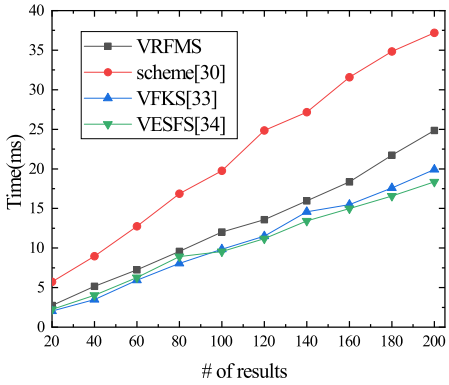


Fig. 10. Verification time of different schemes.

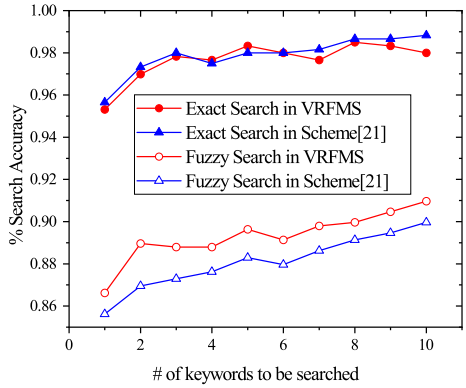


Fig. 11. The accuracy of exact search and fuzzy search.

similar, and after the number of files is greater than 1000, the increase in search time is significantly reduced. This is because in these two schemes, one keyword corresponds to one index, so the search time is positively correlated with the number of keywords. As the number of files increases, the growth rate of the number of different keywords slows down, thus the increase rate of search time slows down as shown in Fig. 9. The time overhead of search in VESFS [29] is still the largest due to the use of the RSA accumulator.

Verification. VRFMS supports correctness verification and completeness verification, however, most existing solutions only support correctness verification. Therefore, this part mainly compares VRFMS with other schemes through experiments regarding the correctness verification. Since the existing schemes do not support ranked search, it is impossible to select top-k results for verification, so this part only conducts comparative experiments by calculating the time overhead required for the operations in the verification phase. As shown in Fig. 10, with the increase of the number of returned results, the time cost required to verify the correctness of results increases linearly. VRFMS only needs 25 milliseconds when verifying 200 returned results. In real life, this overhead is completely acceptable.

The implement of completeness verification is that the proxy server randomly selects λ files to challenge the cloud server. The cloud server calculates its relevance scores as well as verification tags and then returns them to the proxy server. The proxy server only needs to verify that the calculation of results is correct, and then verify the order of them is correct. The time overhead at this stage is actually the time required to verify λ files. From Section 5.4.2, we know that we have 99% probability to detect the malicious behavior when $\lambda = 10$. As shown in Fig. 10, verifying the correctness of 10 files requires less than 2 milliseconds, so the time required to verify the completeness is also extremely low.

Search Accuracy. Search accuracy is measured by using the precision of the results. LSH technology can only guarantee the calculation results are equal with high probability, so there may be a case where a certain file does not contain the queried keyword but the file exists in the returned results, that is, false positive f_p. Then we denote the true positive as t_p. Therefore, the search accuracy in VRFMS is $t_p/(t_p + f_p)$.

A key factor that affects search accuracy is the number of queried keywords. Fig. 11 shows the accuracy of the results by varying the number of keywords in VRFMS and scheme [18] in the exact keyword search and fuzzy keyword search. It can be seen from Fig. 11 that whether it is an exact keyword search or a fuzzy keyword search, as the number of keywords increases, the search accuracy generally increases. The more keywords, the easier we distinguish what we want. Due to the use of the improved bi-gram keyword transformation method, the accuracy of fuzzy keyword search has reached almost 91% in VRFMS.

7 Conclusion and Future Work

We proposed an efficient and verifiable ranked fuzzy multi-keyword search scheme VRFMS in this paper. Based on the existing fuzzy search scheme, we introduced the TF-IDF rule to sort the returned results. We proposed a keyword transformation method based on bi-gram to further improve the accuracy of fuzzy keyword search. We introduced the homomorphic MAC technique to achieve correctness verification and proposed a random challenge technology based on ranking to achieve completeness verification. Formal security analysis and empirical experiments demonstrate that VRFMS is secure and efficient in practical applications.

However, the highest accuracy of fuzzy keyword search in VRFMS is only 91%, and there is still much room for improvement compared to exact keyword search. As part of our future work, we will focus on further improving the search accuracy.

References

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

[2] 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.

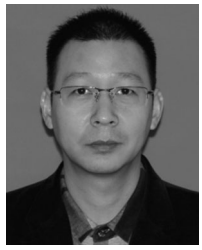
[3] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky, "Searchable symmetric encryption: Improved definitions and efficient constructions," *J. Comput. Secur.*, vol. 19, no. 5, pp. 895–934, 2011.

[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] D. Cash, S. Jarecki, C. Jutla, H. Krawczyk, M.-C. Roşu, and M. Steiner, "Highly-scalable searchable symmetric encryption with support for boolean queries," in *Proc. Annu. Cryptol. Conf.*, 2013, pp. 353–373.

[6] Z. Fu, X. Wu, Q. Wang, and K. Ren, "Enabling central keyword-based semantic extension search over encrypted outsourced data," *IEEE Trans. Inf. Forensics Secur.*, vol. 12, no. 12, pp. 2986–2997, Dec. 2017.

[7] F. Kerschbaum, "Frequency-hiding order-preserving encryption," in *Proc. 22nd ACM SIGSAC Conf. Comput. Commun. Secur.*, 2015, pp. 656–667.
[8] D. S. Roche, D. Apon, S. G. Choi, and A. Yerukhimovich, "POPE: Partial order preserving encoding," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2016, pp. 1131–1142.
[9] S. Lai *et al.*, "Result pattern hiding searchable encryption for conjunctive queries," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2018, pp. 745–762.
[10] Y. Miao, R. Deng, X. Liu, K.-K. R. Choo, H. Wu, and H. Li, "Multi-authority attribute-based keyword search over encrypted cloud data," *IEEE Trans. Dependable Secure Comput.*, vol. 18, no. 4, pp. 1667–1680, Jul./Aug. 2021.
[11] Y. Miao *et al.*, "Privacy-preserving attribute-based keyword search in shared multi-owner setting," *IEEE Trans. Dependable Secure Comput.*, vol. 18, no. 3, pp. 1080–1094, May/Jun. 2021.
[12] Y. Miao, R. Deng, K.-K. R. Choo, X. Liu, J. Ning, and H. Li, "Optimized verifiable fine-grained keyword search in dynamic multi-owner settings," *IEEE Trans. Dependable Secure Comput.*, vol. 18, no. 4, pp. 1804–1820, Jul./Aug. 2021.
[13] Q. Tong, Y. Miao, X. Liu, K.-K. R. Choo, R. Deng, and H. Li, "VPSL: Verifiable privacy-preserving data search for cloud-assisted Internet Of Things," *IEEE Trans. Cloud Comput.*, to be published, doi: 10.1109/TCC.2020.3031209.
[14] J. Li, Q. Wang, C. Wang, N. Cao, K. Ren, and W. Lou, "Fuzzy keyword search over encrypted data in cloud computing," in *Proc. IEEE INFOCOM*, 2010, pp. 1–5.
[15] M. Kuzu, M. S. Islam, and M. Kantarcioglu, "Efficient similarity search over encrypted data," in *Proc. IEEE 28th Int. Conf. Data Eng.*, 2012, pp. 1156–1167.
[16] B. Wang, S. Yu, W. Lou, and Y. T. Hou, "Privacy-preserving multi-keyword fuzzy search over encrypted data in the cloud," in *Proc. IEEE INFOCOM Conf. Comput. Commun.*, 2014, pp. 2112–2120.
[17] 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.
[18] 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 Secur.*, vol. 11, no. 12, pp. 2706–2716, Dec. 2016.
[19] 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, 2020, Art. no. 102469.
[20] W. Sun, X. Liu, W. Lou, Y. T. Hou, and H. Li, "Catch you if you lie to me: Efficient verifiable conjunctive keyword search over large dynamic encrypted cloud data," in *Proc. IEEE Conf. Comput. Commun.*, 2015, pp. 2110–2118.
[21] Z. Wan and R. H. Deng, "VPSearch: Achieving verifiability for privacy-preserving multi-keyword search over encrypted cloud data," *IEEE Trans. Dependable Secure Comput.*, vol. 15, no. 6, pp. 1083–1095, Nov./Dec. 2016.
[22] J. Zhu, Q. Li, C. Wang, X. Yuan, Q. Wang, and K. Ren, "Enabling generic, verifiable, and secure data search in cloud services," *IEEE Trans. Parallel Distrib. Syst.*, vol. 29, no. 8, pp. 1721–1735, Aug. 2018.
[23] C. Xu, C. Zhang, and J. Xu, "vchain: Enabling verifiable boolean range queries over blockchain databases," in *Proc. Int. Conf. Manage. Data*, 2019, pp. 141–158.
[24] W. Yang and Y. Zhu, "A verifiable semantic searching scheme by optimal matching over encrypted data in public cloud," *IEEE Trans., Inf. Forensics Secur.*, vol. 16, pp. 100–115, Jun. 2020, doi: 10.1109/TIFS.2020.3001728.
[25] Q. Tong *et al.*, "VFIRM: Verifiable fine-grained encrypted image retrieval in multi-owner multi-user settings," *IEEE Trans. Serv. Comput.*, to be published, doi: 10.1109/TSC.2021.3083512.
[26] J. Wang *et al.*, "Efficient verifiable fuzzy keyword search over encrypted data in cloud computing," *Comput. Sci. Inf. Syst.*, vol. 10, no. 2, pp. 667–684, 2013.
[27] X. Zhu, Q. Liu, and G. Wang, "A novel verifiable and dynamic fuzzy keyword search scheme over encrypted data in cloud computing," in *Proc. IEEE Trustcom/BigDataSE/ISPA.*, 2016, pp. 845–851.
[28] X. Ge, J. Yu, C. Hu, H. Zhang, and R. Hao, "Enabling efficient verifiable fuzzy keyword search over encrypted data in cloud computing," *IEEE Access*, vol. 6, pp. 45725–45739, 2018.
[29] R. Huang, Z. Li, and G. Wu, "A verifiable encryption scheme supporting fuzzy search," in *Proc. Int. Conf. Secur. Privacy Anonymity Comput. Commun. Storage.*, 2019, pp. 397–411.
[30] H. Zhang, S. Zhao, Z. Guo, Q. Wen, W. Li, and F. Gao, "Scalable fuzzy keyword ranked search over encrypted data on hybrid clouds," *IEEE Trans. Cloud Comput.*, to be published, doi: 10.1109/TCC.2021.3092358.
[31] Q. Chai and G. Gong, "Verifiable symmetric searchable encryption for semi-honest-but-curious cloud servers," in *Proc. IEEE Int. Conf. Commun.*, 2012, pp. 917–922.
[32] K. Kurosawa and Y. Ohtaki, "UC-secure searchable symmetric encryption," in *Proc. Int. Conf. Financial Cryptogr. Data Secur.*, 2012, pp. 285–298.
[33] X. Jiang, J. Yu, J. Yan, and R. Hao, "Enabling efficient and verifiable multi-keyword ranked search over encrypted cloud data," *Inf. Sci.*, vol. 403, pp. 22–41, 2017.
[34] J. Li *et al.*, "Verifiable semantic-aware ranked keyword search in cloud-assisted edge computing," *IEEE Trans. Serv. Comput.*, to be published, doi: 10.1109/TSC.2021.3098864.
[35] K. Kurosawa and Y. Ohtaki, "Uc-secure searchable symmetric encryption," in *Proc. Int. Conf. Financial Cryptogr. Data Secur.*, 2012, pp. 285–298.
[36] S. Hu, C. Cai, Q. Wang, C. Wang, X. Luo, and K. Ren, "Searching an encrypted cloud meets blockchain: A decentralized, reliable and fair realization," in *Proc. IEEE INFOCOM Conf. Comput. Commun.*, 2018, pp. 792–800.
[37] P. Willett, "The porter stemming algorithm: Then and now," *Program*, vol. 40, no. 3, pp. 219–223, Jul. 2006.
[38] D. Catalano and D. Fiore, "Practical homomorphic message authenticators for arithmetic circuits," *J. Cryptol.*, vol. 31, no. 1, pp. 23–59, 2018.



Xinghua Li (Member, IEEE) received the ME and PhD degrees in computer science from Xidian University in 2004 and 2007, respectively. He is currently a professor with the School of Cyber Engineering, Xidian University, China. His research interests include wireless networks security, privacy protection, cloud computing, and security protocol formal methodology.

Qiuyun Tong received the BS degree from the Department of Information and Computing Science, Shaanxi Normal University, Xi'an, China, in 2019. She is currently working toward the PhD degree with the Department of Cyber Engineering, Xidian University, Xi'an, China. Her research interests include information security and applied cryptography.

Jinwei Zhao received the BE degree in information security from the University of Electronic Science and Technology of China in 2013. He is currently working toward the master's degree in computer science with Xidian University. His research interests include data security and searchable encryption.

Yinbin Miao received the BE degree from the Department of Telecommunication Engineering, Jilin University, Changchun, China, in 2011 and the PhD degree from the Department of Telecommunication Engineering, Xidian University, Xi'an, China, in 2016. From September 2018 to September 2019, he was a postdoctor with Nanyang Technological University. He is currently a lecturer with the Department of Cyber Engineering, Xidian University, Xi'an, China.

Siqi Ma received the PhD degree in information system from Singapore Management University in 2018. She is currently a lecturer with the School of Information Technology and Electrical Engineering, the University of Queensland. Her research interests include mobile security, web security, and IoT security.

Jian Weng received the PhD degree with Shanghai Jiao Tong University, Shanghai, China, in 2008. He is currently a professor and the executive dean of the College of Information Science and Technology, Jinan University, Guangzhou. His research interests include public key cryptography, cloud security, and blockchain. He was the PC co-chair or a PC member for more than 20 international conferences.

Jianfeng Ma (Member, IEEE) received the ME and PhD degrees in computer software and communications engineering from Xidian University in 1988 and 1995, respectively. He is currently a professor and the PhD supervisor with the School of Cyber Engineering, Xidian University, China. He is also the director of the Shaanxi Key Laboratory of Network and System Security. His research interests include information and network security, coding theory, and cryptography.

Kim-Kwang Raymond Choo (Senior Member, IEEE) received the PhD degree in information security from the Queensland University of Technology, Brisbane, Australia, in 2006, and currently holds the Cloud Technology endowed professorship at the University of Texas at San Antonio, San Antonio, Texas. He is the founding co-editor-in-chief of the *ACM Distributed Ledger Technologies: Research & Practice*, founding chair of the IEEE TEMS Technical Committee on Blockchain and Distributed Ledger Technologies, an ACM distinguished speaker and IEEE Computer Society distinguished visitor (2021-2023), and a Web of Science's highly cited researcher (Computer Science-2021, Cross-Field-2020). He is also the recipient of the 2019 IEEE Technical Committee on Scalable Computing Award for excellence in scalable computing (middle career researcher).