Efficient Search over Encrypted Medical Data with Known-Plaintext/Background Models and Unlinkability

SHERIF ABDELFATTAH1, MOHAMED BAZA2, MAHMoud BADR1, MOHAMED MAHMOUD 1, GAUTAM SRIVASTAVA3, FAWAZ ALSOLAMI4, ABDULLAH MARISH ALI4,
1Electrical and Computer Engineering Department, Tennessee Tech University, Cookeville, TN 38505 USA
2Department of Computer Science, College of Charleston, Charleston, SC, 29407, USA
3Department of Math and Computer Science, Brandon University, Canada as well as the Research Centre for Interneural Computing, China Medical University, Taichung, Taiwan
4Department of Computer Science, King Abdulaziz University, Jeddah, Saudi Arabia
Corresponding author: M. Mahmoud (e-mail: mmahmoud@tntech.edu).

ABSTRACT In advanced health care systems, the patients’ medical data can be outsourced to cloud servers to enable remote healthcare service providers to access and analyze patients’ data from any location to provide better treatment. However, outsourcing sensitive medical data makes data owners, i.e., patients, concerned about their privacy because private companies run the cloud service and the data can be accessed by them. Therefore, it is important to encrypt the data in the form of documents before outsourcing them to the cloud in a way that enables a data user, i.e., a doctor, to search over these documents without allowing the cloud provider to learn any private information about patients. Several schemes have been proposed to enable search over encrypted medical cloud data to preserve patient privacy, but the existing schemes suffer from high communication/computation overhead because they are designed for a single-data-owner setting. Moreover, they are not secure against known-plaintext/background and linkability attacks, and do not allow doctors to customize their search to avoid downloading irrelevant documents. In this paper, we develop an efficient search scheme over encrypted data for multi-data-owner setting. To secure our scheme, the cloud server obtains noisy similarity scores and doctors de-noise them to download the most relevant documents. Our scheme enables doctors to prescribe search conditions to customize the search without revealing the conditions to the server. Our formal proof and analysis indicate that our scheme can preserve privacy and is secure against known plaintext/background and linkability attacks, and the results of extensive experiments demonstrate the efficiency of our scheme compared to the existing works.

INDEX TERMS Security, privacy, e-health, searchable encryption schemes, and cloud computing.

I. INTRODUCTION

Due to the cloud computing capability of storing large scale databases [1], the patients’ medical data can be outsourced to cloud servers through high speed cellular network, e.g., 5G network and beyond [2], [3]. The cloud enables remote healthcare service providers to access patients’ data from any location to analyze this data using data mining [4] and machine learning [5] techniques for providing better treatment [6], [7].

Well-known examples for cloud-based health systems is the national e-health infrastructures in Finland and Croatia [8]. Also, the USA is widely implementing cloud-based health services, and the market cap is expected to exceed $40 billion by 2026 [9]. However, outsourcing sensitive medical data makes data owners, i.e., patients, concerned about their privacy because private companies run the cloud service and the data can be accessed by them. For instance, over 113 million clinical records were hacked in the US in 2015 [10].

Therefore, it is essential to encrypt the data in the form of documents before outsourcing them to the cloud in a way that enables a data user, i.e., a doctor, to search over these documents without allowing the cloud provider to learn any private information about patients. To enable doctors to download documents of interest without revealing any information to the server, several schemes have been developed for searching over encrypted data [11]–[15]. The idea is that patients attach with each document an encrypted vector (called index) for the keywords of the document. Then, a
doctor encrypts a vector (called trapdoor) that contains the keywords of the documents he/she wants to download and sends it to the cloud server. The server can compute the similarity score of an index and a trapdoor without being able to learn their keywords and returns to the doctor relevant documents.

Motivations. The existing schemes suffer from several limitations.

Firstly, these schemes suffer from high communication/computation overhead and the need for a large number of keys because they are designed for single-data-owner setting (one patient and multiple doctors). In medical applications, multi-data-owner setting (multiple patients and multiple doctors) is more appropriate because a doctor treats several patients, and thus he should be able to search the documents of these patients efficiently. In the existing schemes, a doctor needs to use a unique key for each patient to be able to search his/her documents, which obviously makes key management inefficient due to using many keys at the doctor side.

Secondly, in the existing schemes, doctors cannot customize their search scope to download only the documents that achieve certain search conditions, which may result in downloading irrelevant documents and, thus wasting communication and computation resources. An example for a search condition is laboratory reports with a certain issuance date.

Thirdly, the existing schemes are vulnerable to known plaintext/background attacks and unlinkability. In the known plaintext attack, an adversary can decrypt encrypted data (indices and trapdoors) if he possesses a set of plaintext/ciphertext pairs. In the known background attacks, an adversary uses background (or statistical) information, such as the frequency of keywords, to infer the keywords of the documents by analyzing the frequency of downloading these documents, which may reveal sensitive information on the patients’ health condition. The existing schemes also suffer from linkability attacks in which the server can link the trapdoors (or indices) that have the same keywords. The existing schemes try to thwart this attack by using random numbers in the encryption so that two trapdoors having the same keywords look different, but this is not enough because the server can link the trapdoors by observing that they give the same scores when they matched to all the documents.

Contributions. To address the aforementioned limitations, we propose EPSM: an Efficient and Privacy-preserving Search over Medical cloud data with known plaintext/background and unlinkability security. We provide a formal proof and privacy analysis for EPSM to prove that our scheme is secure and can preserve the privacy of the patients. Moreover, we conduct extensive experiments to evaluate the performance of our scheme and compare it to the existing works. Specifically, the main contributions of this paper are listed as follows:

- EPSM enables customized search in multi-data-owner and multi-data-user setting so that doctors can prescribe search conditions in trapdoors to limit the search scope to the documents that can satisfy the conditions, without revealing the conditions to the server. In EPSM, the cloud server computes noisy similarity scores for indices and trapdoors and doctors de-noise them to download the most relevant documents. Moreover, unlike the existing schemes, EPSM allows each doctor to use only one key to search the data of all patients he treats.
- Our security analysis proves that EPSM is secure under known plaintext/background models, and the cloud server cannot link two trapdoors (or indices) that have the same keywords.
- Extensive experiments are conducted, and the results indicate that EPSM requires low overhead compared to the existing schemes.

The organization of this paper is as follows. The network and threat models and design goals are presented in Section II. In Section IV, the proposed EPSM is explained in detail. Then, we analyze the security and privacy of EPSM in Section V. In Section VI, we present the performance evaluation of EPSM. Section VII provides the related works. Finally, conclusions are drawn in Section VII.

II. RELATED WORK

In this section, we review the related works and compare them to EPSM.

Song et al. [18] and Boneh et al. [19] have proposed secure searchable symmetric encryption (SSE) schemes based on k nearest neighbour (kNN) technique. However, these schemes are designed to support single keyword search over encrypted data, which is very restrictive because searching documents needs multiple keywords to give accurate results. The schemes also suffer from high computation/communication overheads.

Wang et al. [20] have proposed a ranked search scheme. In this scheme, the cloud server executes the search process and sends back only the top most relevant documents to the user. However, this scheme only considers the single keyword search. Then, Cao et al. [11] have proposed a privacy-preserving multi-keyword ranked search scheme. This scheme has a limitation that it does not consider the keyword frequency, and this may result in inaccurate search

---

**Table 1: Comparison with the related works.**

| Schemes | F₁ | F₂ | F₃ | F₄ | F₅ | F₆ | F₇ | F₈ |
|---------|----|----|----|----|----|----|----|----|
| [16]    | ✓  | ×  | ✓  | ×  | ✓  | √*| ×  | ×  |
| [12]    | ×  | ✓  | ✓  | ✓  | ✓  | ×  | ×  | ×  |
| [17]    | ✓* | ✓* | ✓  | ×  | ×  | ✓* | ×  | ✓  |
| EPSM    | ✓  | ✓  | ✓  | ✓  | ✓  | ✓  | ✓  | ✓  |

F₁: Unlinkability of indices/trapdoors; F₂: Using unique key for each doctor; F₃: Support multi-data-owner setting; F₄: Support multi-data-user setting; F₅: A patient/doctor cannot decrypt the indices/trapdoors of other patients/doctor; F₆: Secure against known plaintext model; F₇: Secure against known background model; F₈: Customized search. Note: ✓ denote a realized feature, ✓* denote a realized feature with lower search accuracy and × denotes an unrealized feature.
result.

Xia et al. [16] have proposed a searchable encryption scheme for single-data-owner and multi-data-users setting. The scheme assumes that the server knows the term frequency of each keyword and it uses this background information to guess the keywords of a trapdoor and an index from the similarity score it computes. In order to thwart this attack, the server ranks the documents using inaccurate similarity scores, but this leads to inaccurate search results and downloading irrelevant documents which may cause misdiagnosis by doctors. Also, the proposed scheme is designed for single-data-owner setting which is not suitable for medical applications where a doctor typically treats several patients and it is inefficient to use single-data-owner schemes in multiple-data-owner setting as explained in Section VI.

Xiangya et al. [17] have proposed a privacy-preserving keyword search scheme for single-data-owner and single-data-user setting. This setting is not suitable for medical applications that have multiple patients and multiple doctors. To secure the scheme against known plaintext model, the server ranks the documents using inaccurate similarity scores which may result in downloading irrelevant documents. Also, there is a tradeoff between accuracy and security because higher security is achieved with increasing the inaccuracy of the similarity scores, but downloading wrong documents is more likely.

Zhang et al. [14], have proposed a scheme for multi-keyword ranked search. The scheme uses an additive order function to retrieve the relevant search results. After receiving a trapdoor from a search user, the cloud server compares each encrypted keyword in the trapdoor with all the keywords of each data owner. Then, the cloud server adds all the document’s scores with all the matched keywords. However, because of comparing the individual keywords in the trapdoor with all the keywords, this scheme requires high computation overhead. In [12], Li et al. have proposed a searchable encryption scheme over medical cloud data. To prevent linking the indices/trapdoors that have the same keywords, the scheme uses random numbers in the encryption so that they look different even if they have the same keywords. However, the scheme is designed for a single-data-owner setting, and the cloud server can link trapdoors (or indices) having the same keywords by observing that they give exactly the same scores when they matched to the documents’ indices (or doctors’ trapdoors).

We provide Table 1 to summarize the differences between EPSM and the aforementioned schemes. Unlike the existing schemes, EPSM supports multi-data-owner and multi-data-user settings. Also, EPSM enables customized search feature that allows doctors to customize their search results. EPSM ensures the unlinkabilty of indices/trapdoors having the same keywords and ensures that the indices (or trapdoors) computed by a patient (or a doctor) cannot be decrypted by other patients (or doctors). Our scheme is secure against known plaintext and know background models.

III. SYSTEM MODELS AND DESIGN GOALS

In this section, we present the network and threat models and design goals considered in this paper.

A. NETWORK MODEL

As shown in Fig. 1 the network model consists of four main entities, including an offline key distribution center (KDC), data owners (DO), data users (DU), and cloud server (CS). The role of each entity and the communication model are explained in this subsection.
Table 2: MAIN NOTATIONS

| Notation | Description |
|----------|-------------|
| \( m \)  | vector size |
| \( e \)  | vector extension size |
| \( SK_1, SK_2 \) | KDC secret keys |
| \( S \)  | Secret binary vector of \( SK_1 \) |
| \( \{M_1, M_2, N_1, ..., N_e\} \) | Server secret matrices of \( SK_1 \) |
| \( J \)  | Secret binary vector of \( SK_2 \) |
| \( \{V_i, V_j, U_1, ..., U_N\} \) | Server secret matrices of \( SK_2 \) |
| \( P_i \) | ID of data owner \( i \) |
| \( SK_{P_i}^1, SK_{P_i}^2 \) | \( P_i \)'s secret keys |
| \( \{A_i, B_i, G_i, D_i\} \) | Random matrices for \( P_i \) of \( SK_{P_i}^1 \) |
| \( \{O_i, P_i, T_i, R_i\} \) | Random matrices for \( P_i \) of \( SK_{P_i}^2 \) |
| \( D_x \) | ID of doctor \( x \) |
| \( SK_{D_x}^{1_p}, SK_{D_x}^{2_p} \) | \( D_x \)'s secret keys |
| \( \{E_x, F_x, G_x, H_x\} \) | Random matrices for \( D_x \) of \( SK_{D_x}^{1_p} \) |
| \( \{W_x, X_x, Y_x, H_x\} \) | Random matrices for \( D_x \) of \( SK_{D_x}^{2_p} \) |
| \( t_{P_i} \) | Number of \( P_i \)’s documents |
| \( DC = \{d_{i_1}, d_{i_2}, ..., d_{i_{n_P}}\} \) | \( P_i \)'s documents |
| \( V_i = \{V_{i_1}, V_{i_2}, ..., V_{i_{n_P}}\} \) | \( P_i \)'s documents' keywords vector |
| \( a_i = \{a_{i_1}, a_{i_2}, ..., a_{i_{n_P}}\} \) | \( P_i \)'s documents' random numbers |
| \( n \) | The length of \( a_{i_{j_3}} \) in bits |
| \( I_{V_i} = \{I_{V_{i_1}}, I_{V_{i_2}}, ..., I_{V_{i_{n_P}}}\} \) | \( P_i \)'s indices |
| \( I_a = \{I_{a_1}, I_{a_2}, ..., I_{a_{n_P}}\} \) | \( P_i \)'s encrypted random numbers |
| \( t_{D_x} \) | Number of \( D_x \)'s queries |
| \( Q_x = \{Q_{x_1}, Q_{x_2}, ..., Q_{x_{n_Q}}\} \) | \( D_x \)'s queries |
| \( L_{Q_x} = \{L_{Q_{x_1}}, L_{Q_{x_2}}, ..., L_{Q_{x_{n_Q}}}\} \) | \( D_x \)'s trapdoors |
| \( b_{x,y} \) | \( D_x \)'s query random number |

- Offline key distribution center (KDC): The KDC is an offline entity that is not involved in the searching process. It computes and distributes the data owners’ and data users’ keys. The KDC can be run by the health department that is interested in the security of the system.
- Data owners (DO): The data owner is either a patient or a hospital, and it manages the patient’s medical records. For each document, DO outsources to the cloud server an encrypted document, an encrypted vector containing the keywords of the document (called index), and an encrypted random number used in the index to mask the similarity scores.
- Data users (DU): Data users include doctors, nurses, pharmacists, researchers, etc. Each data user sends an encrypted query (called trapdoor) containing the keywords of the documents he wants to download from the cloud server. The data user receives the documents’ noisy similarity scores, de-noises the scores, and sends the identifiers of the documents with the highest similarity scores to the cloud server to download them.
- Cloud server (CS): After receiving a trapdoor, the cloud server computes the noisy similarity scores of the trapdoor and the index of each document (that achieves the search conditions) and returns to the user the noisy scores. Then, after receiving the identifiers of the documents requested by the data user, the cloud server sends the documents.

In the rest of the paper, for simplicity, we will refer to DO and DU as patients and doctors, respectively.

B. THREAT MODEL

In EPSM, the attacker can be the cloud server and eavesdroppers. The cloud server is honest-but-curious, where it follows our scheme correctly but it is curious to infer sensitive information, such as the health condition of the patients, by analyzing the data it receives \([16], [20]–[26]\). Specifically, eavesdroppers can capture all the communications in the system and analyze them to infer sensitive information. The server should not be able to infer the keywords of the indices and the trapdoors, or link two given trapdoors (or indices) if they have the same keywords or are sent from the same doctor. Moreover, EPSM should also be secure against the following attack models.

1) Known ciphertext model. In this model, the adversary only knows the encrypted indices and trapdoors \([16], [27]\).
2) Known plaintext model. In this stronger model, the adversary has a set of tuples of indices (or trapdoors) and their corresponding plaintext keyword vectors. Using these plaintext-ciphertext pairs, the adversary may try to infer the keywords or the search conditions of other indices and the trapdoors \([24], [28]\).
3) Known background model. In this model, the adversary possesses statistical information, such as the frequency of some keywords (or search conditions), i.e., the probability of querying documents with certain keywords. Using these information, the adversary tries to identify the keywords and the search conditions of the indices/trapdoors \([16], [29]\).

C. DESIGN GOALS

To enable efficient and privacy-preserving search, EPSM should achieve the following design goals.

(1) Customized Search. EPSM should enable doctors to prescribe conditions in trapdoors so that the server returns only the documents that can satisfy these conditions without being able to learn the conditions.
(2) Security and Privacy Preservation. EPSM should prevent the cloud server from inferring any information about the content of documents, indices, and trapdoors. EPSM should also be secure against the Known plaintext and known background models so that the server cannot identify the keywords or the search conditions of given indices/trapdoors. Also, the trapdoors (and indices) that have the same keywords and conditions or are sent from the same doctor should
not be linkable. The eavesdroppers should not be able to infer any sensitive information.

(3) Scalability and Efficiency. EPSM should efficiently support search for a large number of patients/doctors with a small number of keys for efficient key management. It should also need low search time and computation/communication overhead.

IV. PROPOSED SYSTEM

EPSM consists of four phases. In the system initialization phase, the KDC generates and distributes secret keys to patients and doctors. In the index generation phase, for each document, the patient composes the corresponding index and encrypts the random number used to mask the similarity score and outsources them to the cloud. In the trapdoor generation phase, the doctor encrypts a vector containing the keywords and search conditions of the documents he wants to download, and sends the ciphertext, called trapdoor, to the cloud server. Finally, in the query matching phase, the server calculates the noisy similarity scores of the trapdoor and the indices of the documents that can achieve the search conditions. Then, it returns to the doctor the noisy scores to de-noise them and send to the server the identifiers of the documents he wants to download, i.e., the documents that have the highest scores. Finally, the cloud server returns to the doctor these documents. Table 2 gives the main notations used in the paper. Figure 2 shows an overview of EPSM

A. SYSTEM INITIALIZATION

The KDC runs the following algorithms to compute the secret keys of the patients and the doctors.

Setup(1$^m$) → SK1, SK2: This algorithm takes the security parameter 1$^m$ as an input and outputs two Keys SK1 and SK2. The first key is SK1 = {S, M1, M2, N1, . . . , N8}, where, S is a random binary vector of length (m + e + 2), and {M1, M2, N1, . . . , N8} are a set of random invertible matrices of size (m + e + 2) × (m + e + 2), where m and e are the sizes of the keywords and search conditions, respectively. The second key is SK2 = {J, V1, V2, U1, . . . , U8}, where, J is a random binary vector of length n and {V1, V2, U1, . . . , U8} are a set of random invertible matrices of size (n × n), where n is the bit length of the random number the patient uses to mask the similarity score.

KeyGenPatient(SK1, SK2, Pi) → SKP1, SKP2: For each patient Pi, this algorithm outputs two secret keys SKP1 and SKP2. SKP1 is used to encrypt the keyword vectors to calculate the indices, and it is computed as follows:

$$SKP1 = \{S, N1^{-1}A1, N2^{-1}A1, N3^{-1}A1, N4^{-1}A1, N5^{-1}C1, N6^{-1}D1, N7^{-1}C1, N8^{-1}D1\}$$

(1)

where, \{A1, B1, C1, D1\} are (m + e + 2) × (m + e + 2) matrices of random numbers such that A1 + B1 = M1⁻¹, and C1 + D1 = M2⁻¹.

SKP2 is used to encrypt the random number Pi uses to mask the similarity score, and it is computed as follows:

$$SKP2 = \{J, U1^{-1}O1, U2^{-1}P1, U3^{-1}O1, U4^{-1}P1, U5^{-1}T1, U6^{-1}R1, U7^{-1}T1, U8^{-1}R1\}$$

(2)

where \{O1, P1, T1, R1\} are (n × n) matrices of random numbers such that O1 + P1 = V1⁻¹, and T1 + R1 = V2⁻¹.

Finally, the KDC sends SKP1 and SKP2 to the patient Pi.

KeyGenDoctor(SK1, SK2, Ds) → SKD1s, SKD2s: For each doctor Ds, this algorithm outputs two secret keys SKD1s and SKD2s. SKD1s is used to encrypt the vectors of keywords to compose trapdoors and it is computed as follows:

$$SKD1s = \{S, E_xN1, E_xN2, F_xN3, F_xN4, G_xN5, G_xN6, H_xN7, H_xN8\}$$

(3)

where, \{E_x, F_x, G_x, H_x\} are (m + e + 2) × (m + e + 2) matrices of random numbers such that E_x + F_x = M1, and G_x + H_x = M2.

SKD2s is used to decrypt the random numbers of the patients to de-noise the similarity scores, and it is computed as follows:

$$SKD2s = \{J, W_xU1, W_xU2, X_xU3, X_xU4, Y_xU5, Y_xU6, Z_xU7, Z_xU8\}$$

(4)

where \{W_x, X_x, Y_x, H_x\} are (n × n) matrices of random numbers such that W_x + X_x = V1 and Y_x + Z_x = V2.

Finally, the KDC sends SKD1s and SKD2s to the doctor.

B. INDEX GENERATION

To outsource a document, a patient Pi computes an index and an encrypted random number and sends them to the cloud server. To do so, the patient executes the following algorithm.

CreateIndex(SKP1, SKP2, V′ij, aij) → I′ij, Iaij; This algorithm takes as input the patient’s secret keys SKP1 and SKP2, a keyword vector V′ij corresponding to the document, and a random number aij, and outputs the index of the document (I′ij) and the encrypted random number (Iaij).

For a document j, Pi chooses a keyword set \{w1,1, w1,2, . . . , w1,k\} to generate an m-element keyword vector V′i,j. Every element in V′i,j contains the relevance score of the TF-IDF (Term Frequency - Inverse Document Frequency) [30, 31], which represents the significance of keyword w1,j,k within the whole document collection, and it is computed as follows:

$$TF-IDF(w1,j,k, aij) = freqw1,j,k, aij * \log\left(\frac{N}{freqw1,j,k}\right)$$

(5)

where, the frequency of the keyword w1,j,k is freqw1,j,k, aij, N represents the total number of keywords in the documents set, and freqw1,j,k is the total number of documents the keyword appears in. Then, Pi chooses a random number aij for the (m + 1)−th element in the vector V′i,j. After that, Pi composes an (m + e + 2)-element vector V′i,j = V′i,j || EF′i,j, where EF′i,j has (e + 1) elements for the search conditions.
For example, assuming that there is one condition on the issuance year, an example for $V_{i,j}$ is shown in Fig. 3. The figure shows that, the element that represents the issuance year stores one (2021 in the figure) and the other elements store zeros. $e$ elements are used to represent the years and one element stores one all the time. For simplicity, the figure shows the vector with one condition, but the idea can be extended to include multiple conditions.

Then, in order to encrypt $V_{i,j}$, $P_t$ first splits it into two column vectors $v_i'$ and $v_i''$ using the secret $S$. So, for every element in $V_{i,j}$, $P_t$ checks the value of the corresponding element in $S$. If it is zero, $P_t$ sets the corresponding element in $v_i'$ and $v_i''$ with the same value of the element in $V_{i,j}$. Otherwise, $P_t$ chooses two random numbers for this element in $v_i'$ and $v_i''$ where their summation is equal to the value of the corresponding element in $V_{i,j}$.

\[
I_{V_{i,j}} = \left[ N_1^{-1}A_1v_i'; N_2^{-1}B_1v_i'; N_3^{-1}A_1v_i''; N_4^{-1}B_1v_i'j; N_5^{-1}C_1v_i'; N_6^{-1}D_1v_i''; N_7^{-1}C_1v_i'j; N_8^{-1}D_1v_i'' \right]
\]

(6)

where, $I_{V_{i,j}}$ is an $8(m + e + 2)$-element column vector.

Then, to encrypt the random number $a_{i,j}$, $P_t$ first splits it into two column vectors $a_{i,j}'$ and $a_{i,j}''$ using the secret $J$. So, for every element in $a_{i,j}$, $P_t$ checks the corresponding element in $J$. If it is zero, the corresponding elements in $a_{i,j}'$ and $a_{i,j}''$ are set to the same value of the element of $a_{i,j}$. Otherwise, two random numbers are chosen for this element in $a_{i,j}'$ and $a_{i,j}''$ where their summation is equal to the corresponding element in $a_{i,j}$. Finally, the encryption of $I_{V_{i,j}}$ and $I_{a_{i,j}}$ is done to obtain $I_{Q_{x,y}} = I_{V_{i,j}} + I_{a_{i,j}}$. Then, the server $S$ performs the following steps to obtain $I_{Q_{x,y}}$.

1. Measure the similarity score of the trapdoor and the document’s index.
2. Check if the noisy score is less than the maximum similarity score and return these scores.

The identifiers of the matched and redundant documents are then used to download the desired documents.

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/
Algorithm 1: Search algorithm by the cloud

Input: \( I_{Q,x,y}, I_{V,i,j} \) and \( I_{a,i,j} \)
Output: \( \Lambda \) and the corresponding \( I_{a,i,j} \) of each document

1. \( \Lambda \leftarrow \operatorname{Scorelist}() \)
   // Scorelist is a list to hold the scores of the matched documents

2. for \( j \leftarrow 1 \) to number of documents do
   calculate the matching score
   \[ \text{Score}(Q_{x,y} \cdot V_{i,j}) \leftarrow \text{Match}(I_{Q,x,y}, I_{V,i,j}) \]
   // where \( \text{Score}(Q_{x,y} \cdot V_{i,j}) \) is the dot product noisy score of \( Q_{x,y} \) and \( V_{i,j} \)

3. if \( \text{Score}(Q_{x,y} \cdot V_{i,j}') \geq \maxscore \) then
   ignore this index and continue

4. else
   \( \Lambda \leftarrow \text{Scorelist. Append}(\Lambda, \text{Score}(Q_{x,y} \cdot V_{i,j})) \)

end

Output: Send \( \Lambda \) and the corresponding \( I_{a,i,j} \) of each document to \( D_x \)

\( a_{i,j} (I_{a,i,j}) \) is computed using \( SK^2_{P_i} \) as follow.

\[
I_{a,i,j} = \begin{bmatrix}
U_1^{-1}Q_{a,i,j}; & U_2^{-1}P_{a,i,j}; & U_3^{-1}R_{a,i,j}; \\
U_4^{-1}P_{a,i,j}; & U_5^{-1}T_{a,i,j}; & U_6^{-1}R_{a,i,j}; \\
U_7^{-1}T_{a,i,j}; & U_8^{-1}R_{a,i,j}
\end{bmatrix} \tag{7}
\]

where, \( I_{a,i,j} \) is a column vector of size \( 8n \).

Finally, for each document, \( P_i \) sends the cloud server the corresponding index \( I_{V,i,j} \) and the encryption of \( a_{i,j} (I_{a,i,j}) \).

C. TRAPDOOR GENERATION

In this phase, to search for documents of interest, a doctor composes a query \( Q_{x,y} \) containing the keywords of interest and search conditions, and then uses the following algorithm to encrypt it and obtain the trapdoor \( I_{Q,x,y} \).

\[ \text{CreateTrapdoor}(SK^1_{D_x}, Q_{x,y}) \rightarrow I_{Q,x,y} : \text{This algorithm takes the doctor’s secret key} SK^1_{D_x} \text{ and the query vector} Q_{x,y} \text{ as input, and computes the trapdoor} I_{Q,x,y}. \]

Firstly, the doctor \( D_x \) composes the \( m + e + 2 \)-element query vector \( Q_{x,y} \). The first \( m \) elements contain the keywords of interest where each element stores one or zero to indicate whether or not the corresponding keyword to the element exists in the documents of interest. Specifically, \( Q_{x,y}[k] = 1 \) if the doctor is interested in keyword \( k \), and \( Q_{x,y}[k] = 0 \) if the doctor is not interested in the keyword. Then, a random number \( b_{x,y} \) is selected for the \((m + 1) - th\) element. After that, \( D_x \) uses the following \( e + 1 \) elements to prescribe the search conditions as follows.

\[
Q_{x,y}[k] = \begin{cases} 
-c, & \text{if } k \in F \\
g \times c, & \text{if } k = m + e + 2 \\
0, & \text{other}
\end{cases} \tag{8}
\]

where \( c \) is a random number that is greater than the maximum noisy similarity score, \( F \subset [m + 2, m + e + 1] \) is set of the elements’ positions of the document issuance years that the doctor wants to search, and \( g \) is the length of \( F \).

For example, if the doctor wants to search for the documents issued in 2021 and download them as shown in Fig. 3, he/she stores \(-c \) in the element corresponding to 2021, \( c \) in the last element, and zeros in the other elements. Moreover, as illustrated in Fig. 3, if the document is issued in 2021, the patient stores one in the corresponding element to 2021 and the last element and zero in the other elements of the index vector. By doing so, if the condition is satisfied, the dot product of the elements of the conditions in the index and the trapdoor is equal to zero. Otherwise, it is \( c \) whose value is greater than the maximum noisy similarity score. So, if the noisy similarity score obtained by the cloud server is greater than the maximum score, this indicates that the document does not satisfy the search conditions, otherwise, all the conditions are satisfied. For simplicity, Fig. 3 shows only one condition, but it can be extended to add additional conditions.

To encrypt the query vector \( Q_{x,y} \) and obtain the trapdoor \( I_{Q,x,y} \), \( Q_{x,y} \) is first split into two row vectors \( q'_{x,y} \) and \( q''_{x,y} \) using the secret \( S \), as follows. For every element in \( Q_{x,y} \), \( D_x \) checks the corresponding element in \( S \). If it is one, the corresponding element in \( q'_{x,y} \) and \( q''_{x,y} \) are set to the same value of the element of \( Q_{x,y} \). Otherwise, two random numbers are chosen for this element in \( q'_{x,y} \) and \( q''_{x,y} \) where their summation is equal to the corresponding element in \( Q_{x,y} \). Finally, the trapdoor \( I_{Q,x,y} \) is computed using \( SK^1_{D_x} \) as follow.

\[
I_{Q,x,y} = \begin{bmatrix} q'_{x,y}F_xN_1, q'_{x,y}F_xN_2, q'_{x,y}F_xN_3, q'_{x,y}F_xN_4, q''_{x,y}G_xN_5, q''_{x,y}G_xN_6, q''_{x,y}H_xN_7, q''_{x,y}H_xN_8 \end{bmatrix} \tag{9}
\]

where \( I_{Q,x,y} \) is an \( 8(m + e + 2) \)-element row vector. Finally, \( D_x \) sends the trapdoor \( I_{Q,x,y} \) to the cloud server.

D. QUERY MATCHING

In this phase, the cloud server computes the noisy similarity score of the trapdoor and the index of each document that achieves the search conditions without being able to learn the real score. Then, the server sends to the doctor the noisy scores and the encryptions of the random numbers the patients used to mask the scores as indicated in Algorithm 1. After that, the doctor de-noises the scores and sends to the cloud server the identifiers of the documents he wants to download. These documents include the ones that have high similarity scores in addition to redundant documents that are downloaded to protect against known-background attacks by preventing the server from learning the documents of interest and guessing the keywords of these documents. Finally, the cloud server returns to the doctor the documents he requested. The following algorithms are used in this phase.

\[ \text{Match}(I_{Q,x,y}, I_{V,i,j}) \rightarrow \text{NoisyScore} : \text{This algorithm takes a trapdoor} I_{Q,x,y} \text{ and an index} I_{V,i,j} \text{ as input, and produces the noisy similarity score of} Q_{x,y} \text{ and} V_{i,j} \text{ by computing the dot product} (I_{Q,x,y} \cdot I_{V,i,j}). \]
If all the search conditions prescribed in the trapdoor are satisfied, \( Q_{x,y} \cdot V_{i,j} = \text{KeywordScore} + a_{i,j}b_{x,y} \), which gives the noisy similarity score that is equal to the similarity score of the keywords part in vectors \( V_{i,j} \) and \( Q_{x,y} \) (KeywordScore) masked by the random number \( a_{i,j}b_{x,y} \), where \( a_{i,j} \) is added by the patient in the document index and \( b_{x,y} \) is added by the doctor in the trapdoor. If at least one condition is not satisfied \( Q_{x,y} \cdot V_{i,j} = \text{KeywordScore} + a_{i,j}b_{x,y} + c \) and by selecting \( c \) to be greater than the maximum noisy similarity score, the server can learn that the doctor does not achieve at least one condition and it should discard the document. Finally, for each document that achieves the doctor’s conditions, the cloud server returns to the doctor the noisy similarity score and the encryption of the random number \( a_{i,j} \) (\( I_{a_{i,j}} \)) used by the patient to mask the similarity score.

For each document, the doctor decrypts \( I_{a_{i,j}} \) to obtain \( a_{i,j} \) using the algorithm \( \text{DecryptRandomNumber}() \). Then, using this random number and his trapdoors’ random number \( b_{x,y} \), the doctor de-noises the noisy scores (by subtracting \( a_{i,j}b_{x,y} \)) to obtain the real scores. Then, the doctor sends to the cloud server the identifiers of the documents he wants to download, i.e., the documents that have the highest scores. The doctor should also download redundant documents to protect against known-background attacks by preventing the server from learning the documents of interest and guessing the keywords of these documents. Finally, the cloud server returns to the doctor these documents.

**Theorem IV.1.** The server can obtain the noisy similarity score of indices and trapdoors using dot product operation.

**Proof.**

\[
I_{Q_{x,y} \cdot V_{i,j}} = q'_{xy}E_x A_{i}v'_{ij} + q'_{xy}E_x B_{i}v'_{ij} + q'_{xy}F_x A_{i}v'_{ij} + q'_{xy}F_x B_{i}v'_{ij} + q''_{xy}G_x C_{ij}v''_{ij} + q''_{xy}G_x D_{ij}v''_{ij} + q''_{xy}H_x C_{ij}v''_{ij} + q''_{xy}H_x D_{ij}v''_{ij} = q'_{xy}(E_x + F_x)(A_{i} + B_{i})v'_{ij} + q''_{xy}(G_x + H_x)(C_{ij} + D_{ij})v''_{ij} = q'_{xy}M_1 M_1^{-1}v'_{ij} + q''_{xy}M_2 M_2^{-1}v''_{ij} = Q_{x,y} \cdot V_{i,j}
\]

(10)

V. SECURITY AND PRIVACY ANALYSIS

Our formal proof of the security/privacy-preservation of our scheme follows the logic and model presented in [32]. The goal of the proof is to prove that the cloud server can compute the noisy similarity score of an index and a trapdoor without revealing their keywords and search conditions. We will also prove that external attackers cannot reveal the keywords and search conditions. The server and external attackers cannot also learn the similarity scores of the indices and trapdoors.

**Proposition 1.** The cloud server can calculate the noisy similarity score of an index and a trapdoor without being able to learn the keywords or the search conditions.

**Proof.** History. The history consists of two sets, including a set of \( n \) indices corresponding to the documents of patients \( \{I_{V} = \{I_{V_{i,1}}, I_{V_{i,2}}, \ldots, I_{V_{i,n}}\}\) for each patient \( P_{i} \), generated by encrypting a set of keywords vectors \( V = \{V_{i,1}, V_{i,2}, \ldots, V_{i,n}\} \) and a set of \( u \) trapdoors corresponding to the doctors’ queries \( \{Q_{x} = \{Q_{x,1}, Q_{x,2}, \ldots, Q_{x,u}\}\) for each doctor \( D_{x} \) generated by encrypting a set of queries vectors \( Q = \{Q_{x,1}, Q_{x,2}, \ldots, Q_{x,u}\} \).
Trace. A trace $\text{Trace}(H)$ represents the information of the history $H$ that is deduced by the cloud server, e.g., from the search patterns.

View. The view $W(I_V, I_Q, \text{Trace}(H))$ has the encrypted history and its trace and it is the observation of the server.

A simulator $S$ can produce a fake view $W'$ that is indistinguishable from the original view $W$ by executing these steps.

Step 1: $S$ generates the secret key $sk' = SK'$.

Step 2: $S$ generates a set of random documents $D' = \{d'_1, \ldots, d'_n\}$ such that $|d'_i| = |d_i|$, $1 \leq i \leq n$, $d'_i = \{w_1, w_2, \ldots\}$, where $|d'_i|$ is the number of keywords in $d'_i$.

Step 3: $S$ generates a set of queries as $Q' = \{Q'_{s,1}, Q'_{s,2}, \ldots, Q'_{s,u}\}$, where $Q'$ is a random copy of $Q$.

Step 4: $S$ generates a set of keyword vectors ($V'$) which is a random copy of $V$, where $V' = \{V'_{i,1}, V'_{i,2}, \ldots, V'_{i,n}\}$.

Step 5: $S$ generates indices $I'_V$ and trapdoors $I'_Q$ using the secret $sk'$.

From the previous construction, EPSM is indistinguishable and secure if $S$ has a trace $\text{Trace}(H')$ of the history $H' = (I'_V, I'_Q)$ that is similar to the original trace $\text{Trace}(H)$ such that in no probabilistic polynomial time, an adversary can differentiate between the original view $W(I_V, I_Q)$ and the fake view $W(I'_V, I'_Q)$ with non-negligible advantage, where the correctness of the construction implies this conclusion.

**Preposition 2.** EPSM ensures that adversaries cannot reveal any keyword or search condition from trapdoors and/or indices, i.e., EPSM is secure in the known-ciphertext model.

**Proof.** In EPSM, the confidentiality of the indices and trapdoors is protected using encryption. For each patient/doctor, the matrix $M$ is split into two randomly chosen matrices which are multiplied by another matrix $N$, and thus, no patient/doctor is able to reconstruct the matrix $M$. This is important because by knowing $M$, adversaries can compute the keywords or the search conditions from the indices or trapdoors. This means that the keywords and the conditions are protected in the known-ciphertext model because there is no information can be leaked about them.

**Preposition 3.** EPSM ensures that the indices (or trapdoors) computed by a patient (or a doctor) cannot be decrypted by other patients (or doctors).

**Proof.** If all patients (or doctors) share the same key, then the indices (or the trapdoors) computed by a patient (or a doctor) can be decrypted by other patients (or doctors). Thus, patients’ sensitive information, e.g., the health condition, can be revealed by other patients. To avoid this problem in EPSM, each patient/doctor has a unique key, and, in spite of using different keys to encrypt the indices/trapdoors, the cloud server is still able to obtain the dot product of the keyword and query vectors and obtain the noisy similarity score.

**Preposition 4.** EPSM is secure against known-plaintext model if the random numbers $a_{i,j}$ and $b_{x,y}$ of each index and trapdoor are not known by the adversary.

**Proof.** Under the known-plaintext model, the adversary possesses a set of plaintexts (keyword vector and queries) and their ciphertexts (indices and trapdoors). The adversary tries to use this set to attack the encryption scheme, e.g., by decrypting a new ciphertext. Most of the existing schemes are not secure against known-plaintext model because the server can learn the similarity score, by calculating the dot product of an index and trapdoor. Therefore, if an index has $n$ elements (i.e., $n$ unknowns), the server needs $n$ trapdoors (with known plaintexts) to create $n$ linear equations and solve them to compute the $n$ elements of the index. To protect against this attack in our scheme, the server does not know the similarity score. It only knows noisy similarity score ($r$ in $s + r$, where $s$ is the real similarity score). The random numbers $a_{i,j}$ and $b_{x,y}$ are known only to the patient and doctor. The patients should use a different $a_{i,j}$ for each index and doctors should use a different $b_{x,y}$ in each trapdoor, so that $a_{i,j}b_{x,y}$ is always different even if the same query is used multiple times. By reusing $a_{i,j}$ and $b_{x,y}$, the server can subtract two equations to cancel the term $a_{i,j}b_{x,y}$ and obtain the difference between the two scores, and thus the server can create enough number of equations to obtain the keyword vector of an index. Therefore, by changing $a_{i,j}b_{x,y}$ continuously, the server cannot have enough number of equations to solve because $a_{i,j}b_{x,y}$ introduces a new unknown. Similarly, for the same reasons explained, the server cannot create equations to decrypt trapdoors.

**Preposition 5.** EPSM ensures that the cloud server or an external eavesdropper cannot identify the keywords and the conditions of the documents/trapdoors under the known background model.

**Proof.** In the known background attacks, an adversary uses background (or statistical) information, such as the frequency of keywords, to infer the keywords of the documents by analyzing the frequency of downloading these documents, which may reveal sensitive information on the patients such as their diseases. To protect against this attack in our scheme, the server should not know the real frequency of downloading documents, and this is done by downloading redundant documents (that do not have the highest similarity scores) by the doctors, and because of hiding the similarity scores of the documents in our scheme, the server cannot identify these redundant documents.

**Preposition 6.** EPSM ensures unlinkability of indices/trapdoors sent from the same patient/doctor or having the same keywords and search conditions.

**Proof.** The existing schemes suffer from unlinkability attacks in which the server can link the trapdoors (or indices) that have the same keywords. They try to thwart this attack by using random numbers in the encryption, so that two trapdoors (or indices) having the same keywords look different. However, this is not enough because the server can link two trapdoors (or indices) by observing that they give exactly same similarity scores when they matched to a set of indices (or trapdoors). EPSM ensures that the encrypted in-
dices/trapdoors that have the same keywords or sent from the same patients/doctors look different because of using random numbers in splitting the vectors $V_{x,j}$ and $Q_{x,y}$. Moreover, our scheme also hides the similarity score from the server using the random numbers $a_{i,j}$ and $b_{x,y}$. Therefore, by computing the noisy similarity scores by a server for two trapdoors with same keywords, the scores look different due to using different $b_{x,y}$ in the two trapdoors.

VI. PERFORMANCE EVALUATION

In this section, we compare the performance of EPSM with the existing schemes.

A. EXPERIMENT SETUP AND METRICS

1) Experiment Setup

To evaluate the communication and computation overheads of EPSM, we have performed our experiments using python running on an Intel ® Core i7-8700 CPU @3.20GHz and 16 GB RAM. The computation and communication overheads of EPSM are compared to the proposed schemes in [12] and [16] after using them in a multi-data-owners setting. All the results presented in this section are averaged over 1000 trials for 2,000 documents, 10 patients, 10 doctors, and 2 bytes for each element in the ciphertext vector.

2) Performance Metrics

Three performance metrics are used for comparison and assessment of our scheme.

1) Computation overhead. The time needed by patients/doctors to generate indices/trapdoors to be sent to the server. Also, the time needed to calculate the similarity score by the cloud server to search the documents.

2) Communication overhead. The amount of data transmitted during the communication between the patients/doctors and the server.

3) Key management. The number of a doctor’s keys that is used to search all the documents of all patients.

B. EXPERIMENT RESULTS

1) Computation Overhead

Fig. 4a gives the computation overhead of generating indices/trapdoors versus the number of keywords. The figure shows that the computation overhead increases as the number of keywords increase due to increasing the size of the matrices and vectors. In EPSM, because of supporting the multi-data-owners setting, the doctor needs to calculate one trapdoor to search over the documents of all patients. Also, the same computation time is needed to generate the indices and the trapdoors because their vectors have the same size. It can also be seen from the figure that EPSM is more efficient comparing to [12] and [16], because to use these schemes in a multi-data-owners setting, the doctor needs to calculate one trapdoor for each patient to be able to search their documents.

Fig. 4b gives the time needed to calculate the similarity score by the cloud server versus the number of keywords. As shown in the figure, the computation overhead needed to calculate the similarity score increases as the number of keywords increase because the vector size increases. The figure also shows that EPSM needs less time than [12] and [16]. Although, EPSM increases by higher rate because it needs eight dot product operations to support multi-data-owner setting, the computation time is low (in ms) even with a high number of keywords (2000).
In EPSM, each patient sends an index \((I_{V_{i,j}})\) for each document. The overhead is \(|I_{V_{i,j}}|\), where \(|I_{V_{i,j}}|\) is the size of the index. If each element in the ciphertext is represented by 2 bytes, the ciphertext size in our scheme becomes \(16(m + e + 2)\) bytes. Similarly, the trapdoor vector size is \(16(m + e + 2)\) bytes. Fig. 5 gives the the index/trapdoor communication overhead versus the number of keywords. It can be seen that the communication overhead increases linearly with the number of keywords due to increasing the vector size. Moreover, the schemes \([16]\) and \([12]\) need more overhead comparing to EPSM because they need to extend the vectors by the maximum possible number of documents before encrypting them, and the doctor needs to send multiple trapdoors that are equal to the number of patients to search their documents but in our scheme only one trapdoor is sent to search the data of all patients.

3) Key Management

Fig. 6 gives the number of a doctor’s keys versus the number of patients. As shown in the figure, in EPSM, each doctor has only one key that is used to search all the documents of all patients. However, in \([16]\) and \([12]\), because the schemes are designed for a single-data-owner setting, each doctor needs to share a key with each patient. In e-health application, a doctor typical treats several patients, so multi-data-owner is a proper setting. The figure shows that the number of keys of a doctor increases linearly with the number of patients. Using many keys in the system makes key management inefficient.

VII. CONCLUSION

In this paper, we have proposed, EPSM, an efficient and secure search scheme over encrypted medical cloud data in multi-data-owner setting. To secure EPSM, the cloud server cannot learn the similarity scores of indices and trapdoors, but it computes noisy scores and sends them to the doctor to de-noise them. Moreover, EPSM enables a new feature that allows doctors to customize their search results by expressing search conditions in the trapdoors. Our formal proof and security analysis demonstrate that EPSM can preserve patient privacy and is secure against known plaintext and known background models. Also, EPSM ensures the unlinkability of indices/trapdoors having the same keywords. Finally, our extensive experiments demonstrate that EPSM requires low computation and communication overheads and small number of keys because it is designed for multi-data-owner setting which is more suitable for medical applications.

For the future work, we will investigate denial of service (DoS) attacks against the centralized server. Specifically, we will try to replace the central server with a blockchain network. Also, we will investigate the use of machine learning technology to diagnose diseases e-health system.

ACKNOWLEDGMENT

This project was funded by the Deanship of Scientific Research (DSR) at King Abdulaziz University, Jeddah, under grant no. KEP-16-611-42. The authors, therefore, acknowledge with thanks DSR for technical and financial support.

References

[1] J. C.-W. Lin, Y. Djenouri, and G. Srivastava, “Efficient closed high-utility pattern fusion model in large-scale databases,” Information Fusion, 2021.
[2] M. M. Badr, M. M. Fouda, and A. S. T. Eldien, “A novel vision to mitigate pilot contamination in massive mimo-based 5g networks,” in 2016 11th International Conference on Computer Engineering & Systems (ICCES). IEEE, 2016, pp. 366–371.
[3] J.-C.-W. Lin, G. Srivastava, Y. Zhang, Y. Djenouri, and M. Aloquility, “Privacy-preserving multiobjective sanitization model in 6G IoT environments,” IEEE Internet of Things Journal, vol. 8, no. 7, pp. 5340–5349, 2020.
[4] J. C.-W. Lin, Y. Djenouri, G. Srivastava, U. Yun, and P. Fournier-Viger, “A predictive GA-based model for closed high-utility itemset mining,” Applied Soft Computing, vol. 108, p. 107422, 2021.
[5] Y. Zhao, J. C.-W. Lin, G. Srivastava, D. Guo, H. Zhang, H. Yi, and A. Jolfaei, “Multi-objective neural evolutionary algorithm for combinatorial optimization problems,” IEEE Transactions on Neural Networks and Learning Systems, 2021.

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/
[6] M.-H. Kuo, “Opportunities and challenges of cloud computing to improve health care services,” Journal of Medical Internet Research, vol. 13, no. 3, p. e67, 2011.

[7] S. A. Alansari, M. M. Badr, M. Mahmoud, and W. Alasmary, “Efficient and privacy-preserving contact tracing system for covid-19 using blockchain,” in 2021 IEEE International Conference on Communications Workshops (ICC Workshops). IEEE, 2021, pp. 1–6.

[8] P. D. Ravee, “The world of cloud-based services: storing health data in the cloud,” last accessed: Aug. 2021. [Online]. Available: https://cloud-based-services-storing-health-data-in-the-cloud93053/

[9] “Healthcare cloud computing market to hit US$ 40 Bn by 2026,” last accessed: Aug. 2021. [Online]. Available: https://www.globenewswire.com/news-release/2019/05/08/1819458/0/en/Healthcare-Cloud-Computing-Market-to-Hit-US-40-Bn-by-2026.html

[10] “Record compromised,” last accessed: Aug. 2021. [Online]. Available: https://dashboards.healthit.gov/quickstats/pages/breaches-protected-health-information.php

[11] N. Cao, C. Wang, M. Li, K. Ren, and W. Lou, “Privacy-preserving multi-keyword ranked search over encrypted cloud data,” IEEE Transactions on Parallel and Distributed Systems, vol. 25, no. 1, pp. 222–233, 2014.

[12] H. Li, Y. Yang, Y. Dai, J. Bai, S. Yu, and Y. Xiang, “Achieving secure and efficient dynamic searchable symmetric encryption over medical cloud data,” IEEE Transactions on Cloud Computing, 2017.

[13] Y. Yang, X. Liu, R. H. Deng, and Y. Li, “Lightweight sharable and traceable secure mobile health system,” IEEE Transactions on Dependable and Secure Computing, vol. 17, no. 1, pp. 78–91, 2017.

[14] W. Zhang, S. Xiao, Y. Lin, T. Zhou, and S. Zhou, “Secure ranked multi-keyword search for multiple data owners in cloud computing,” in Dependable Systems and Networks (DSN), 2014 44th Annual IEEE/IFIP International Conference on. IEEE, 2014, pp. 276–286.

[15] W. Zhang, Y. Lin, S. Xiao, J. Wu, and S. Zhou, “Privacy preserving ranked multi-keyword search for multiple data owners in cloud computing,” IEEE Transactions on Computers, vol. 65, no. 5, pp. 1566–1577, 2016.

[16] Z. Xia, X. Wang, X. Sun, and Q. Wang, “A secure and dynamic multi-keyword ranked search scheme over encrypted cloud data,” IEEE Transactions on Parallel and Distributed Systems, vol. 27, no. 2, pp. 340–352, 2015.

[17] W. Xiangyu, J. Ma, M. Yinbin, X. Liu, and Y. Ruikang, “Privacy-preserving diverse keyword search and online pre-diagnosis in cloud computing,” IEEE Transactions on Services Computing, 2019.

[18] D. X. Song, D. Wagner, and A. Perrig, “Practical techniques for searches on encrypted data,” in Security and Privacy, 2000. S&P 2000. Proceedings. 2000 IEEE Symposium on. IEEE, 2000, pp. 44–55.

[19] D. Boneh, G. Di Crescenzo, R. Ostrovsky, and G. Persiano, “Public key encryption with keyword search,” in Eurocrypt, vol. 3027. Springer, 2004, pp. 506–522.

[20] C. Wang, N. Cao, K. Ren, and W. Lou, “Enabling secure and efficient ranked keyword search over outsourced cloud data,” IEEE Transactions on Parallel and Distributed Systems, vol. 23, no. 8, pp. 1467–1479, 2011.

[21] C. Wang, W. Zhang, J. Xu, X. Ju, and N. Yu, “A fast privacy-preserving multi-keyword search scheme on cloud data,” Proceedings of the 2012 International Conference on Cloud Computing and Service Computing, CSC 2012, pp. 104–110, 2012.

[22] W. Sun, B. Wang, N. Cao, M. Li, W. Lou, Y. T. Hou, and H. Li, “Privacy-preserving multi-keyword text search in the cloud supporting similarity-based ranking,” in Proceedings of the 8th ACM SIGSAC symposium on Information, computer and communications security, 2013, pp. 71–82.

[23] R. Li, Z. Xu, W. Kang, K. C. Yow, and C.-Z. Xu, “Efficient multi-keyword ranked query over encrypted data in cloud computing,” Future Generation Computer Systems, vol. 30, pp. 179–190, 2014.

[24] Z. Fu, F. Huang, K. Ren, J. Weng, and C. Wang, “Privacy-preserving smart semantic search based on conceptual graphs over encrypted outsourced data,” IEEE Transactions on Information Forensics and Security, vol. 12, no. 8, pp. 1874–1884, 2017.

[25] H. Li, Y. Yang, T. H. Luan, X. Liang, L. Zhou, and X. S. Shen, “Enabling fine-grained multi-keyword search supporting classified sub-dictionaries over encrypted cloud data,” IEEE Transactions on Dependable and Secure Computing, vol. 13, no. 3, pp. 312–325, 2015.

[26] J. Yu, P. Lu, Y. Zhu, G. Xue, and M. Li, “Toward secure multikeyword top-k retrieval over encrypted cloud data,” IEEE transactions on dependable and secure computing, vol. 10, no. 4, pp. 239–250, 2013.

[27] W. K. Wong, D. W.-I. Cheung, B. Kao, and N. Mamoulis, “Secure KNN computation on encrypted databases,” pp. 139–152, 2009.