Achieving Privacy-Preserving DSSE for Intelligent IoT Healthcare System

Yaru Liu, Jia Yu *Member, IEEE*, Jianxi Fan, Pandi Vijayakumar *Senior Member, IEEE* and Victor Chang

***Abstract*—As the product of combining Internet of Things (IoT), cloud computing and traditional healthcare, Intelligent IoT Healthcare (IIoTH) brings us a lot of convenience, meanwhile security and privacy issues have attracted great attention. Dynamic searchable symmetric encryption (DSSE) technique can make the user search the dynamic healthcare information from IIoTH system under the condition that the privacy is protected. In this paper, a novel privacy-preserving DSSE scheme for IIoTH system is proposed. It is the first DSSE scheme designed for PHR files database with forward security. We construct the secure index based on hash chain and realize trapdoor updates for resisting file injection attacks. In addition, we realize fine-grained search over encrypted PHR files database of attribute-value type. When the user executes search operations, he/she gets only a matched attribute value instead of the whole file. As a result, the communication cost is reduced and the disclosure of patient's privacy is minimized. The proposed scheme also achieves attribute access control, which allows users have different access authorities to attribute values. The specific security analysis and experiments show the security and the efficiency of the proposed scheme.**

***Index Terms*—Intelligent Internet of Things Healthcare, forward security, attribute access control, searchable encryption, privacy preserving**

I. Introduction

INTELLIGENT IoT Healthcare (IIoTH) system [1–3] has been rapidly developed in recent years. It is generated by integrating Internet of Things (IoT) and cloud computing into traditional healthcare. In an IIoTH system, a set of IoT devices (e.g, smart bracelets) that construct wireless body area networks, can continuously collect patient's health data (e.g, blood presure). These data are periodically aggregated into Personal Health Records (PHRs) by the IoT gateway, and then are uploaded to the cloud. In this way, IIoTH not only provides timely diagnosis and evidence of health misdiagnosis, but also greatly facilitates access and share with low local storage costs.

Protecting the privacy of patients is a big challenge in IIoTH system [4, 5]. Therefore, it is necessary to encrypt the PHR files before the PHR files are outsourced to the cloud. Whereas, the encrypted data loses the ability of being searched based on keyword. Searchable Encryption (SE) technology successfully realizes the direct search on the encrypted data. The concept of searchable encryption was firstly proposed in [6]. In order to improve efficiency, Goh et al. [7] designed a secure index based on bloom filter for constructing Searchable Symmetric Encryption (SSE) scheme. Curmola et al. [8] employed the structure of inverted index to construct SSE scheme, which greatly improved the search efficiency. Boneh et al. [9] first proposed Public key Encryption with Keyword Search (PEKS) mechanism, which was motivated by the retrieval of encrypted email system. Compared with SSE technology, the PEKS mechanism brings higher computation overhead. However, it can easily achieve richer and more complex functions [10–14].

In IIoTH system, PHR files usually adopt standardized data format, that is, the data format of attribute-value type. In attribute-value type database [15, 16], each entry has a unique identifier which represents a file containing multiple attributes. Each attribute of the file corresponds to an attribute value. Some searchable encryption schemes for healthcare have been proposed [2, 17]. Nonetheless, these schemes must return the entire PHR file when the user performs search operation. Actually, the user may need to get only a certain attribute value but not the whole file. For example, the researcher should search to get only the allergy information of some patient but not the patient identity and others in the encrypted PHR from IIoTH system. Clearly, the existing schemes not only bring communication burden, but also leak extra information to users. In addition, PHR files include much sensitive attribute information of patients, which should not be accessed by anyone. Only the users authorized by hospital (such as doctors) can access the attribute value of PHR files. Some attribute access control keyword search and data sharing schemes are proposed [18–20]. However, these schemes adopt public key encryption, which incurs high computation overhead.

In IIoTH system, PHR files often require to be dynamically updated. As a result, it is significant to design SSE schemes that can support dynamic updates. Some dynamic searchable symmetric encryption (DSSE) schemes have been put forward [21, 22]. However, these schemes cannot resist file injection attacks [23] when the PHR file is updated. The cloud server is able to infer the encrypted keyword in the trapdoor by injecting the forged files into searched database. This kind of powerful attacks seriously damage the privacy of patients. How to resist file injection attacks has been becoming a hot topic of DSSE. Stefanov et al. [24] firstly introduced the definition of forward security for resisting this attack, which requires the update trapdoor cannot be linked with previous trapdoors.

Y. Liu is with the College of Computer Science and Technology, Qingdao University, Qingdao 266071, China. E-mail: lyr_0616@163.com

(*Corresponding author: J. Yu*) J. Yu is with the College of Computer Science and Technology, Qingdao University, Qingdao 266071, China, with Guangxi Key Laboratory of Cryptography and Information Security, Guilin, 541004, China. E-mail:qduyujia@gmail.com.

J. Fan is with the School of Computer Science and Technology, Soochow University, Suzhou 215006, China (Email: jxfan@suda.edu.cn).

P. Vijayakumar is with the Department of Computer Science and Engineering, University College of Engineering Tindivanam, Tindivanam, Tamil Nadu, India. Email: vijibond2000@gmail.com.

V. Chang is with the Artificial Intelligence and Information Systems Research Group, School of Computing, Engineering and Digital Technologies, Teesside University, Middlesbrough, UK. Email: ic.victor.chang@gmail.com

Bost proposed $\Sigma o\phi o\varsigma$ scheme with forward security [25], in which one-way trapdoor permutation function is used to update trapdoors. Song et al. [26] presented two efficient DSSE schemes with forward security. The proposed schemes adopted the similar hash chain with $\Sigma o\phi o\varsigma$ scheme. Zuo et al. [27] presented a forward secure DSSE scheme based on bitmap index and homomorphic addition encryption. Unfortunately, the existing DSSE schemes with forward security cannot be directly applied to IIoTH system in which PHR files are stored in the form of attribute-value type due to the following reasons. Firstly, existing DSSE schemes with forward security are designed for traditional file system but not for attribute-type database. Using previous schemes, cloud server may return redundant files to the user. The reason is that different attribute values may contain the same keyword in PHRs. Assume a user wants to search files whose attribute att value contains keyword w. Using previous schemes, the user sends trapdoor obtained by encrypting keyword w to the cloud server. And then the cloud server returns all the files contain keyword w. These returned files also contain some files whose another attribute att' value contains the keyword w, except the files with attribute att value containing the keyword w. So some of these returned files are not what the user requires and may contain the privacy information of patients. Secondly, the user might often need to get only one certain attribute value but not the entire file by performing search operations. Using previous schemes, the user will have to get the whole file containing extra attribute values, which obviously leaks the extra information of patients. In conclusion, previous schemes not only bring communication burden to the user, but also cause more information leakage. As a result, how to design forward secure DSSE scheme for IIoTH system is still an unsolved problem.

In this paper, a novel Privacy-Preserving DSSE (PPDSSE) scheme for IIoTH system is proposed. It is the first DSSE scheme that can be well applied to PHR files database and be against file injection attacks. In order to achieve forward security, we construct a hash chain for each keyword of each attribute. For each update, a fixed length bit string is randomly chosen as a new state, which is used to update trapdoor. The update trapdoor cannot match the previous search trapdoors. Moreover, our scheme can realize fine-grained search and condition search over PHR database of attribute-value type. Condition search refers to searching the value of another attribute att' on the condition that a certain attribute att value contains search keyword w. Specially, based on the keyword w contained in the attribute att value, the user can only know another attribute att' value by executing condition search. Condition search, as a novel notion, is suitable for IIoTH system. As a result, the disclosure of patient's privacy is minimized and the communication cost is reduced. Our scheme also achieves attribute access control, which allows different users have different access authorities to attribute values. Finally, the specific security analysis and experimental results indicate our scheme is secure and efficient.

We organize the rest of the paper as follows. The next section presents the system model, design goals, notations, personal health record and bitmap index. Section III introduces our scheme in detail. In Section IV and section V, we respectively give security analysis and experimental evaluation. We conclude this paper in the last section.

II. Problem Formulation

A. System Model

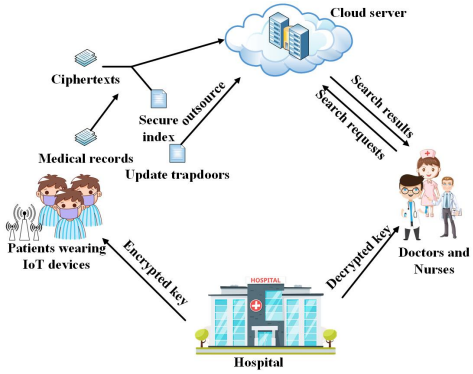

Fig. 1: Overview of the system model

As shown in the Fig. 1, there are multiple different kinds of entities in the system model: Patients, Doctors and Nurses, Hospital and Cloud server.

- Patients: As the data owner, they would like to store the collected vital signal data from wearing IoT devices on the cloud server. First of all, the collected data is periodically integrated into PHR files by honest IoT gateway. Then, IoT gateway constructs secure index and encrypts each attribute value of each file. Finally, the secure index and the set of encrypted PHR files are both uploaded to the cloud server.
- Doctors and Nurses: As the data users, they would like to search some patients' personal health information or do research on a certain kind of disease. They require to generate the search trapdoor corresponding to the search keyword. And then send the trapdoor to the cloud server. For convenience of description, we use users to represent doctors and nurses in the next section.
- Cloud server: After receiving the trapdoor, the cloud server performs search operations based on the secure index. And then it judges whether the user has access to the attributes of matched files. Finally, the cloud server returns the matched value that user is allowed to access. In addition, cloud server is assumed to be honest and curious. It honestly executes the search operation, update operation and returns the correct file set containing the search keyword to user. However, the cloud server is curious about the contents of files. It desires to get more valuable knowledge from the received data.
- Hospital: Hospital is responsible for generating the encrypted/decrypted keys of PHR files. Only patients and users authorized by the hospital are able to encrypt or decrypt files.

TABLE I: **Notations and Descriptions**

Notations	Descriptions
DB	The PHR files database
EDB	The encrypted PHR files database
λ	The secure parameter
w	The keyword
t_w	The trapdoor
ATT	The set of attribute $ATT = \{att_1, att_2, ..., att_n\}$
att	The attribute
n	The number of attributes
$\mathcal{W}$	The keyword set $\mathcal{W} = \{W_{att_1}, W_{att_2}, ...\}$
$\|\mathcal{W}\|$	The total number of keywords included in $\mathcal{W}$
W_{att}	The keyword set related to the attribute att: $W_{att} = \{w_1, w_2, ..., w_i, ...\}$
$\|W_{att}\|$	The number of keywords included in W_{att}
st_c	The latest state
bs	The identifier of PHR file
$V_{bs,att}$	The value corresponding to the attribute att in bs
$C_{bs,att}$	The encrypted value set corresponding to $V_{bs,att}$
$V_{att,w}$	The value set corresponding to the attribute att including keyword w
$C_{att,w}$	The encrypted value set corresponding to $V_{att,w}$
$DB(w)$	The set of files containing keyword w
$DB(u, att)$	The set of files in which user u is allowed to access attribute att
R_{bs}	The encrypted files corresponding to bs
m	The largest number of files

TABLE II: **Attribute-value type database**

Value / Attribute / Ind	att_1	att_2	att_3	...	att_n
ind_1	$V_{1,1}$	$V_{1,2}$	$\perp$	...	$V_{1,n}$
ind_2	$\perp$	$V_{2,2}$	$V_{2,3}$	...	$V_{2,n}$
...	...	...	...	...	...
ind_m	$V_{m,1}$	$V_{m,2}$	$V_{m,3}$	...	$V_{m,n}$

B. Design Goals

The following requirements should be satisfied in the proposed scheme.

- Searchability. The proposed scheme should support keyword search and condition search. Compared with other DSSE schemes, the proposed scheme should allow user to obtain an attribute value of the file.
- Dynamic update. The proposed scheme should be able to dynamically update users and files.
- Forward security. A newly updated trapdoor cannot be linked with previous search trapdoors in the proposed scheme.
- Attribute access control. In PHR files, some privacy information (i.e., ID number, mobile phone number, home address) is very important for patients, which should not be allowed to access arbitrarily. The proposed scheme should support different users have different access authorities to attribute values.
- Privacy preserving. The proposed scheme should ensure that the meaningful knowledge from the encrypted PHR files and the stored secure index are not known by the cloud server, except for permitted leakage.
- Efficiency. The cloud server should efficiently return the matched search results to the user.

C. Notations

The common notations in this paper are described in Table I.

D. Personal Health Record(PHR)

We show a simple attribute-value type database for PHR in TABLE.II. In such a database, each row represents a PHR record(file) identified by a unique identifier. Each PHR has multiple attributes and each attribute corresponds to one value. If the PHR does not contain one attribute, the corresponding value is set to $\perp$. As shown in TABLE.II, there are m records, n attributes and each attribute of each record has a corresponding value. For record ind_1, the value of att_1 is $V_{1,1}$. The value of att_3 is $\perp$, which represents the record ind_1 does not contain the attribute att_3. In addition, the same attribute for different records may have the same value.

E. Bitmap Index

Bitmap index: Assume there are total m files at most. In our scheme, the file identifier is denoted by bitmap index that actually is a bit string with m bits. We set the bits of bitmap from the right to the left. If the file f_i exists, we set the i-th bit of the bitmap to 1; otherwise, set it to 0. We show an example with $m = 4$ shown in Fig. 2. Suppose that there exists one file f_1 (Fig. 2.(a)) initially. When a file f_2 is added, we require to produce a bit string $2^2 = 0100$ and XOR with the original bit string (Fig. 2.(b)). If the file f_1 is deleted, we also require to produce a bit string $2^1 = 0010$ and XOR with the original bit string (Fig. 2.(c)).

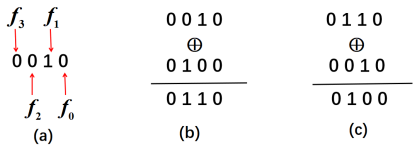

Fig. 2: Bitmap index

III. Construction of Our Work

In this section, we describe the constructed PPDSSE scheme in detail.

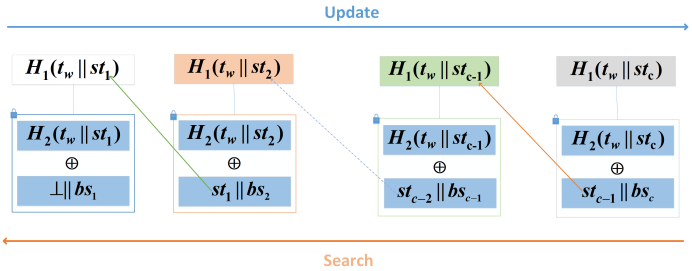

Fig. 3: Secure index

A. The Detailed Construction

The proposed scheme makes the definition of Π = {$Setup$, $BuildIndex$, $Update$, $Search$, Dec}. First of all, we illustrate some used symbols and functions: f, g, y, φ are pseudo-random functions; H_1, H_2 are hash functions. We use λ to

denote the secure parameter for secret keys and the bit length of state. We use μ, l, η, ς to denote the output length of pseudo-random functions. Let m be the maximum number of files that scheme can support. Namely, the length of bitmap index is m. Hash function H_1 and H_2 are both used to encrypt the trapdoor and the state. So we use $l+\lambda$ to denote the input length of hash functions and use γ to denote the output length of hash function H_1. Because the output of H_2 requires to XOR with previous state and bitmap index, the output length of H_2 is $m+\lambda$. We use $\{0,1\}^*$ to denote the set of bit strings with arbitrary length. Let k_{u_i}, k_1 and k_2 be the secret keys with λ bits for pseudo-random functions f, y, φ respectively. Let k_w be the secret key with ς bits for pseudo-random function g. These functions are defined as follows: $f : \{0,1\}^* \times k_{u_i} \rightarrow \{0,1\}^{\mu}$, $g : \{0,1\}^* \times k_w \rightarrow \{0,1\}^{l}$, $y : \{0,1\}^* \times k_1 \rightarrow \{0,1\}^{\eta}$, $\varphi : \{0,1\}^* \times k_2 \rightarrow \{0,1\}^{\varsigma}$, $H_1 : \{0,1\}^{l+\lambda} \rightarrow \{0,1\}^{\gamma}$, $H_2 : \{0,1\}^{l+\lambda} \rightarrow \{0,1\}^{m+\lambda}$. The detailed algorithms in the proposed scheme are described as the following:

Setup(1^λ): Input a secure parameter λ to generate a key set $\{k_1, k_2, k_s, k_{u_i}\}$. Specially, k_1, k_2 is used to encrypt attributes; k_s is used to encrypt search keyword; k_{u_i} is the secret key of user u_i, which is used to encrypt the user's identification and the accessed attribute. The hospital generates a symmetric key k_e used to encrypt/decrypt files. Only patients and users with hospital authorization can get the symmetric key k_e to encrypt or decrypt files.

Algorithm 1 Setup

Input: The security parameter λ.
Output: The secret key set K.
Patient:
$k_1, k_2, k_s, k_{u_i} \leftarrow \{0,1\}^{\lambda}$;
Hospital:
$k_e \leftarrow SKE.Gen(1^{\lambda})$;
Return $K = \{k_1, k_2, k_s, k_{u_i}, k_e\}$.

BuildIndex$(K, DB; \perp)$:

1) Scan the PHR file database DB to extract different keywords and construct keyword set $\mathcal{W}$=$\{\{W_{att_1}\}, \{W_{att_2}\}, \cdots, \{W_{att_n}\}\}$;
2) Each attribute value of each PHR file is encrypted separately using symmetric encryption primitive $C_{bs,att}$=$SKE.Enc(k_e, V_{bs,att})$;
3) In order to improve search efficiency, initialize an empty map T. The map T is used to construct the secure index in the form of dynamic list with two columns. The first column is used to store newly updated trapdoor information and the second column is used to store newly updated PHR identifier information. One new row will be added into secure index for each update. Fig. 3 shows the secure index of the keyword w of attribute att. The secure index has been updated c times. It is stored on the cloud server;
4) Initialize an empty map Σ which is a dynamic list with two rows to map the relation between keyword and the latest state. The first row is used to store keyword and the second row is used to store the latest state. A new row will be added into Σ when a new keyword is added. It is stored by patients and users;
5) Initialize an empty map A to map the relation between user's identifier information and user's attribute access authority. The map A is a dynamic list with two columns. The first column is used to store user's identifier and access attribute information, and the second column is used to store user's attribute access authority. User's attribute access authority is represented by bit string. If the user is allowed to access this attribute of the PHR, the corresponding PHR position is set to 1, otherwise is set to 0. We will add (delete) n rows in A when a user is added (deleted). It is stored on the cloud server.

Algorithm 2 BuildIndex

Input: The PHR file database DB.
Output: The keyword set $\mathcal{W}$, encrypted PHR file and empty maps T, Σ and A.
IoT gateway:
for each $att \in ATT$ **do**
  for each $bs \in DB$ **do**
    Extract keyword and construct keyword set W_{att};
    $C_{bs,att} = SKE.Enc(k_e, V_{bs,att})$;
  end for
end for
Initialize three empty maps T, Σ and A.
Return $\mathcal{W}, \{C_{bs,att}\}, T, \Sigma$ and A.

Algorithm 3 Update$_f$

Input: The secret key set K, the file identifier bs, the attribute att, the keyword w, the state st_c and user u_i.
Output: The latest state st_{c+1}, the index location v and the update index information e.
Patient:
$\{t\} \leftarrow \emptyset$;
$st_c \leftarrow \Sigma[att||w]$, $k_{att} \leftarrow y_{k_1}(att)$;
$k_w \leftarrow \varphi_{k_2}(att)$, $t_w \leftarrow g_{k_w}(w)$;
if $st_c = \perp$ **then**
  $st_1 \leftarrow \{0,1\}^{\lambda}$, $v \leftarrow H_1(t_w||st_1)$;
  $e \leftarrow H_2(t_w||st_1) \oplus (\perp||bs)$;
else
  $st_{c+1} \leftarrow \{0,1\}^{\lambda}$, $v \leftarrow H_1(t_w||st_{c+1})$;
  $e \leftarrow H_2(t_w||st_{c+1}) \oplus (st_c||bs)$;
end if
$\Sigma[att||w] \leftarrow st_{c+1}$;
for all users u_i allowed to access attribute att **do**
  $t_i \leftarrow f_{k_{u_i}}(u_i||att)$, $\{t\} \leftarrow \{t\} \cup t_i$;
end for
Send $(v, e, \{t\}, bs)$ to the cloud server.
Cloud server:
$T[v] \leftarrow e$;
for $t_i \in \{t\}$ **do**
  $z \leftarrow A[t_i] \oplus bs$, $A[t_i] \leftarrow z$;
end for
Return T, A.

Update$(K, bs, att, w, st; EDB, u, A)$: When the patient wants to update (i.e.,add or delete) a file bs which includes the value of attribute att containing keyword w. She/He firstly encrypts the attribute att to generate a key k_w, which is used for encrypting the keyword w to generate the update trapdoor t_w. Forward security requires the update trapdoor cannot match previous search trapdoors. Therefore, for achieving forward security, the trapdoor should be updated when an update happens. The patient randomly chooses λ bit string as a new state to generate a new location in the index T. The generated new index information is stored in the newly updated location.

The previous state is embedded in the newly generated index information. In addition, attribute access array also needs to be updated. The update of attribute access array includes two aspects: one is the update of file; the other is the update of the user.

- **File update.** If the user can access the certain attribute of the update file, the corresponding value of access attribute in attribute access array A should be updated. The patient sends the updated file identifier and the location information to the cloud server. The location information denotes the entry location of the attribute that the user is allowed to access in A. According to the location information of A, the cloud server retrieves the corresponding value, which is performed XOR operation with the updated file identifier. The array A is updated in this way.
- **User update.** If the user u_i leaves the hospital, the user u_i cannot to access the files in IIoTH system any more. The hospital should revoke all access authorities of the user u_i. All entries associated with the user u_i are deleted from A by the cloud server. If the user u_j joins the hospital, the hospital authorizes the user u_j to access certain attributes of PHR files. Attribute entries associated with the user u_j should be added into A by the cloud server.

Algorithm 4 Update$_u$

Input: The secret key k_{u_i}, k_{u_j}, the added user identifier u_i, the allowed access attribute att_i and the deleted user identifier u_j.
Output: The updated information $\{a\}, \{d\}, \{bs\}$.
Hospital:
$\{d\}, \{a\}, \{bs\} \leftarrow \emptyset$;
for $att_i \in ATT$ **do**
  $a_i = f_{k_{u_i}}(u_i||att_i)$, $d_i = f_{k_{u_j}}(u_j||att_i)$;
  $\{a\} \leftarrow \{a\} \cup a_i, \{d\} \leftarrow \{d\} \cup d_i$;
  The corresponding bit positions of the files that can be accessed are set to 1 and the remaining are set to 0, generating bs_i;
  $\{bs\} \leftarrow \{bs\} \cup bs_i$
end for
Send $(\{a\}, \{bs\}, \{d\})$ to the cloud server.
Cloud server:
for $a_i \in \{a\}, bs_i \in \{bs\}$ **do**
  $A[a_i] \leftarrow A[a_i] \oplus bs_i$;
end for
for $d_i \in \{d\}$ **do**
  Remove $A[d_i]$;
end for
Return A.

Keyword Search$(K, att, w, st_c; EDB, T, A)$: When the user wants to search the value of the certain attribute containing an interested keyword, she/he needs to encrypt the search attribute and the keyword to generate search trapdoor. Then she/he sends the search trapdoor t_w, the encrypted attribute k_{att} and the latest state st_c to the cloud server. The latest state can be got from the state list Σ. When the cloud server receives above information, it is able to compute the location in secure index T based on the search trapdoor and the latest state st_c. The cloud server can obtain all previous states and bitmap by decrypting the index information corresponding to the index location. Finally, the cloud server XORs the bit map bs_i to get the final result sum_e, which is composed by the identifiers of all files containing the search keyword. Specially, in order to save storage space of the cloud server, after each search, the cloud server removes the entries related to the search keyword and stores the final result sum_e corresponding to the current state st_c in the secure index. Different users have different access authorities to the attributes of files. In order to achieve fine-grained access control, we construct a dynamic list with two columns. The first column is user's information that is obtained by encrypting user's identifier and accessed attribute, and the second column is access authority information that indicates which PHRs contain the attribute the user can access. Access authority information is represented by bitmap index. Specially, the length of bitmap index is equal to the maximum number of records that the scheme can support. If the user has access authority to the attribute of the PHR, the corresponding bit position is set to 1, otherwise is set to 0. In order to implement fine-grained access control, the user only needs to send the user's information generated by encrypting identifier and accessed attribute with his/her own private key to the cloud server. After searching PHRs containing the interested keyword based on the trapdoor, the cloud server needs to judge whether the user can access the attribute of PHRs. It performs AND operation on the access authority corresponding to user's information in A and the search result sum_e (Algorithm 5, lines 22-29). The bit position 1 of the result indicates that the user has the access authority to the attribute of the file; otherwise, does not have. In this way, the cloud server can judge whether the user has access to PHRs or attributes.

Condition Search$(K, att', att, w, st_c; EDB, T, A)$: When the user would like to search the value of another attribute att' on the matched condition that the attribute att contains the search

Algorithm 5 Keyword search

Input: The secret key set K, the latest state st_c, the attribute att, the search keyword w and the user identifier u.
Output: The search result bs.
User:
1: $st_c \leftarrow \Sigma[att||w]$;
2: **if** $st_c \doteq \perp$ **then**
3: Return $\varnothing$;
4: **else**
5: $k_{att} \leftarrow y_{k_1}(att)$, $k_w \leftarrow \varphi_{k_2}(att)$;
6: $t_w \leftarrow g_{k_w}(w)$, $s \leftarrow f_{k_u}(u||att)$;
7: **end if**
8: Send (k_{att}, st_c, t_w, s) to the cloud server.
Cloud server:
9: $sum_e \leftarrow 0$, $st \leftarrow st_c$;
10: **while** $st_c \neq \perp$ **do**
11: $v \leftarrow H_1(t_w||st_c)$, $e \leftarrow T[v]$;
12: $(st_{c-1}||bs) \leftarrow e \oplus H_2(t_w||st_c)$;
13: $st_c \leftarrow st_{c-1}$;
14: $sum_e \leftarrow sum_e \oplus bs$;
15: Remove $T[v]$;
16: **end while**
17: $T[v] \leftarrow H_2(t_w||st) \oplus (\perp||sum_e)$.
18: $z \leftarrow sum_e \wedge A[s]$;
19: **for** $(i = 0; i < m; i++)$ **do**
20: **if** (z[i]=1) **then**
21: The search result $r[i] = 1$, other $m - 1$ bits are 0.
22: The file identifier obtained above is used to locate the row and the encrypted attribute k_{att} to locate the column in the PHR files database. The intersection of the row and column is the search attribute value.
23: $r[i] \leftarrow 0$;
24: **end if**
25: **end for**
26: Send the encrypted attribute value $C_{r,att}$ to the user.

keyword w, the user generates the trapdoors by encrypting keyword w contained in attribute att value and attributes att'. Especially, the key for encrypting the keyword w is obtained by encrypting attribute att. And then the user sends the trapdoor to the cloud server. The cloud server searches for finding the file identifiers by using the trapdoor of the keyword w contained attribute att value. It firstly locates the corresponding row in the encrypted PHRs database, and then locates the corresponding column by the trapdoor of attribute att'. The intersection of the row and the column is the value of searched attribute.

Dec$(k_e, C_{r,att})$: When the user receives the encrypted value, he/she decrypts the encrypted value with symmetric encryption key by computing $V_{r,att} \leftarrow Dec(k_e, C_{r,att})$.

B. A Toy Example

In order to help reader better understand, we analyze a simple example to illustrate the scheme.

TABLE III: **an example of attribute-value type EHR files**

Ind	Name	Gender	Allergy	Symptom
0001	Alice	Female	$\perp$	Cough, runny nose, sore throat
0010	Bob	Male	Penicillin	Fatigue, stuffy, fever, headache
0100	Alice	Female	$\perp$	Stomachache

Assume the largest number of files is 4 (i.e., $m = 4$). We take Table III as an example. As shown in the Table III, there are three files and four attributes. Each entry is identified with a unique bit string and the attribute set is ATT=$\{Name, Gender, Allergy, Symptom\}$. Suppose one doctor (i.e., u_d) and one nurse (i.e., u_n) can access the PHR database. They both can access attributes $Name$, $Gender$ and $Symptom$. But only one of them can access $Allgery$ allowed u_d. Now, we construct the secure index and the attribute access array based on the attribute-value type database. Note that we have illustrated only one of these attributes (i.e., $Name$) in detail.

1) Constructing secure index T
   - Extract keywords for each attribute to generate keyword set $\mathcal{W}$: $\mathcal{W}$=$\{W_{Name}, W_{Gender}, W_{Allgery}, W_{Symptom}\}$, where $W_{Name} = \{Alice, Bob\}$ et al.;
   - Construct the file set containing keyword: $DB(Alice) = \{0001, 0100\}$, $DB(Bob) = \{0001\}$;
   - Encrypt attribute: k_{Name1}=y_{k_1}(Name), k_{Name2}=φ_{k_2}(Name);
   - Encrypt keyword: $t_{Alice} = g_{k_{Name2}}(Alice)$, $t_{Bob} = g_{k_{Name2}}(Bob)$;
   - For each update, choose a bit string as new state $st_c \leftarrow \{0,1\}^{\lambda}$;
   - Encrypt each attribute value in PHR files database: $C_{bs,att} = SKE.Enc(k_e, V_{bs,att})$.
2) Constructing the attribute access array A
   - Build the set of files for attributes that each user can access: $DB(u_d, Name) = \{0111\}$, $DB(u_d, Gender) = \{0111\}$, $DB(u_d, Allergy) = \{0111\}$, $DB(u_d, Symptom) = \{0111\}$. User u_n adopts the same method to generate $DB(u_n, att)$.

The secure index for the attribute $Name$ is shown in Fig. 4.(a). The attribute access array A is shown in Table IV. In addition, patients and users need to store a state array Σ containing the latest state of each keyword for each attribute. We show it in Fig. 4.(b). We give a simple example of condition search to

TABLE IV: **Attribute access array** A

User information	Access authority
$f_{k_{u_d}}(u_d, Name)$	0111
$f_{k_{u_d}}(u_d, Gender)$	0111
$f_{k_{u_d}}(u_d, Allergy)$	0111
$f_{k_{u_d}}(u_d, Symptom)$	0111
$f_{k_{u_n}}(u_n, Name)$	0111
$f_{k_{u_n}}(u_n, Gender)$	0111
$f_{k_{u_n}}(u_n, Allergy)$	0000
$f_{k_{u_n}}(u_n, Symptom)$	0111

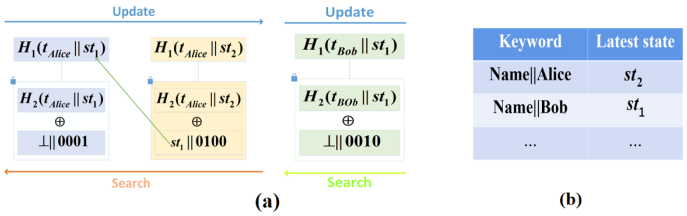

Fig. 4: Secure index and state array

illustrate the search process. Assume doctor u_d desires to know the allergy of Alice. That is to say, u_d wants to search the value of $Allergy$ on the matched condition that $Name = Alice$.

First, u_d needs to generate search trapdoors: $t_{Alice} = g_{k_{Name}}(Alice), k_{Name} = \varphi_{k_2}(Name), k_{Allergy} = y_{k_1}(Allergy)$. And then sends the search trapdoors and the latest state st_2 for keyword $Alice$ to the cloud server. Second, when the cloud server receives above information, it gets the location in the index T by computing $H_1(t_{Alice}||st_2)$. In this way, the cloud server can get file identifiers containing the keyword $Alice$: $0100 \oplus 0001 = 0101$. Third, the cloud server needs to know whether user u_d can access the attribute $Allergy$ of files 0101. The cloud server receives the information $s = f_{k_{u_d}}(u_d, Allergy)$ from u_d. It is able to search $A[s]$ (i.e., $A[s] = 0111$). We have $A[s] \wedge 0101 = 0101$. It means u_{d_1} is allowed to access the $Allergy$ of file 0001 and file 0100. Finally, the cloud server searches the corresponding attribute value from the encrypted database in Table III and then returns $c_{0001,Allergy}$=$SKE.Enc\{k_e, \perp\}$, $c_{0100,Allergy}$=$SKE.Enc\{k_e, \perp\}$ to u_d.

IV. Security Analysis.

The proposed scheme satisfies the security definition of secure searchable encryption [7]. The update query does not leak the updated keywords, which makes the proposed scheme satisfy the definition of forward security [24].

Theorem 1: Suppose y, φ and g are PRFs, and H_1 and H_2 are Hash functions. Leakage functions $\mathcal{L} = (\mathcal{L}_{Setup}, \mathcal{L}_{Update}, \mathcal{L}_{Search})$ can be defined as follows:

$$\begin{cases} \mathcal{L}_{Setup} = (\perp), \\ \mathcal{L}_{Update}(i, bs_i, w) = (i, bs_i), \\ \mathcal{L}_{Search}(att, w) = (sp(w), ap(w)), \end{cases} \quad (1)$$

where $sp(w)$=$\{i|$for each query $(i,w)\}$ denotes search pattern, $ap(w)=\{t_1,t_2,...,t_Q\}$ denotes access pattern, $t_i=\{i,DB_i\}$ denotes the search query and $t_i=\{i,bs_i\}$ denotes the update query. Then our proposed scheme is an $\mathcal{L}-adaptively-secure$ SSE scheme with forward security.

Proof. We prove this Theorem through a series of games. Starting from $\mathbf{Real}^{\Pi}_{\mathcal{A}}(\lambda)$, we give total five games. Any game except the first is lightly different from the previous one. We will show these games are indistinguishable. The final game is $\mathbf{Ideal}^{\Pi}_{\mathcal{A},\mathcal{S}}(\lambda)$. Therefore, we know $\mathbf{Real}^{\Pi}_{\mathcal{A}}(\lambda)$ is indistinguishable from $\mathbf{Ideal}^{\Pi}_{\mathcal{A},\mathcal{S}}(\lambda)$ according to the property of indistinguishability.

Game G_1: G_1 has no difference with $\mathbf{Real}^{\Pi}_{\mathcal{A}}(\lambda)$ except that instead of using y to generate k_{att} and using g to generate t_w, the game uses maps $\mathbf{Token}_1$ and $\mathbf{Token}_2$ to store $(w||t_w)$ and $(att||k_{att})$ pairs respectively. When t_w needs to be used in the search algorithm, the game first checks whether $\mathbf{Token}_1$ contains an entry related to w. If the entry related to w can be found, the game returns t_w; otherwise, an l bit string is chosen randomly and stored in $\mathbf{Token}_1$ in the form of (w,t_w) pairs. When k_{att} needs to be used in the search algorithm, the game does the same as that for searching t_w. The adversary cannot distinguish pseudo-random functions g, y and truly random functions.

$$Pr[\mathbf{Real}^{\Pi}_{\mathcal{A}}(\lambda)=1]\text{-}Pr[G_1=1]\leq Adv(\lambda).$$

Game G_2: G_2 has no difference with G_1 except using random oracle to replace H_1 to generate u. In G_2, for each update query, we randomly choose a string from $\{0,1\}^n$ as the update trapdoor and store it in the map $\mathbf{L}$. For each search query, we randomly choose a string to perform the random oracle $\mathbf{H}_1$ such that $\mathbf{H}_1[(t_w||st_c)]$=$\mathbf{L}[t_w||st_c]$. In G_2, Σ requires to maintain all states since the execution of the random oracle requires all states of search keyword. If $\mathcal{A}$ makes a query on $\mathbf{H}_1$ using $t_w||st_{c+1}$, a value u' will be returned to $\mathcal{A}$. It is very likely that $u'\neq u$. The reason is that $\mathbf{L}[t_w||st_{c+1}]$ hasn't been added to $\mathbf{H}_1$ and u' is a string randomly picked by the oracle. If the adversary $\mathcal{A}$ would like to query $\mathbf{H}_1$ with $t_w||st_{c+1}$, the adversary $\mathcal{A}$ needs to guess the state st_{c+1}, and the probability is $1/2^{\lambda}$ for the reason that each state is randomly chosen from $\{0,1\}^{\lambda}$. Suppose a PPT $\mathcal{A}$ can make poly(λ) times queries at most, we have

$$Pr[G_2=1]\text{-}Pr[G_1=1]\leq p(\lambda)/(2^{\lambda}+neg(\lambda)).$$

Game G_3: G_3 has no difference with G_2 except that instead of using H_2 to generate e, H_2 is modeled as a random oracle. Thus, we have

$$Pr[G_3\text{=}1]\text{-}Pr[G_2\text{=}1]\leq p(\lambda)/(2^{\lambda}+neg(\lambda)).$$

Game G_4: G_4 has no difference with G_3 except that in G_4, the number of update is recorded by a counter v and the update requests after the last search are recorded by a map **up**. The counter v is set to 0 in the Setup algorithm. For each update, v is increased by 1. In G_3, adversary queries random oracle by $t_w||st_{c+1}$. In contract, the random oracle is chosen at random without knowing st in G_4. For each update, cloud server receives two random strings in G_3 and G_4. Thus, it is impossible for adversary to distinguish G_3 and G_4. For each search, cloud server receives four random strings and does the same as that in G_3. From the view of adversary $\mathcal{A}$, what it sees in G_4 and that in G_3 are distinguishable.

$$Pr[G_4\text{=}1]\text{-}Pr[G_3\text{=}1]\text{=}0.$$

$\mathbf{Ideal}^{\Pi}_{\mathcal{A},\mathcal{S}}(\lambda)$: The $\mathbf{Ideal}^{\Pi}_{\mathcal{A},\mathcal{S}}(\lambda)$ game has no difference with G_4 except using $\mathbf{sp}(w)$ and $\mathbf{uh}(w)$ to replace actual searches and updates, where $\mathbf{uh}(w)$=$\{(j,bs_j)|q_j=(j,bs_j,w)(1\leq j\leq Q)\}$. The simulator $\mathcal{S}$ keeps two maps used to simulate random oracle queries and uses a counter to store the update number. For each update, $\mathcal{S}$ randomly chooses two strings. In the search algorithm, $\underline{w}=min$ $\mathbf{SP}(w)$ is used to denote the first index of w that was in the $\mathbf{SP}(w)$. It is because the trapdoor t_w can be generated by $\underline{w}$. $\mathcal{S}$ can use $\mathbf{uh}$ instead of $\mathbf{up}$ in G_4 to decide the update queries about keyword w. The simulator $\mathcal{S}$ generates the view that is difficult to be distinguished from the one generated in G_4.

$$Pr[\mathbf{Ideal}^{\Pi}_{\mathcal{A},\mathcal{S}}(\lambda)\text{=}1]-Pr[G_4=1]\text{=}0.$$

From above all, we have

$$Pr[\mathbf{Real}^{\Pi}_{\mathcal{A}}(\lambda)\text{=}1]\text{-}Pr[\mathbf{Ideal}^{\Pi}_{\mathcal{A},\mathcal{S}}(\lambda)\text{=}1]\leq negl(\lambda).$$

V. Experimental Evaluation

In this section, we make comparison of functions with other SSE schemes and analyze the performance of proposed scheme with real-world database by experiments.

A. Functional Comparison

As we can see in TABLE V, only our proposed scheme can simultaneously achieve keyword search, condition search, file update, user update, forward security and attribute access control.

TABLE V: **Functional comparison of various schemes**

Properties	[8]	[21]	[25]	[28]	[29]	Our scheme
Keyword search	√	√	√	√	√	√
Condition search	×	×	×	×	×	√
File update	×	√	√	√	√	√
User update	×	×	√	×	√	√
Forward security	×	×	√	√	×	√
Attribute access control	×	×	×	√	√	√

B. Performance Analysis

Experiments are run on a Linux OS equipped with 2.4GHz Intel(R) Core(TM) i5 CPU 4GB RAM and 2GB RAM, which are used to simulate cloud server, user and IoT gateway. We use JAVA language on the Indian Liver Patient Dataset (ILPD) dataset from the popular medical dataset [30]. The ILPD dataset includes 583 instances and 11 attributes. We instantiate pseudo-random functions f,g,y,φ with MD5, hash functions H_1 with SHA-1, H_2 with SHA-256 and SSE with AES. So the values of μ, l, η and ς are 128, the value of γ is 160, the value of λ is 80 and the value of m is 176. Note that in our system, wearing devices are used to collect patients vital signs. After that, these collected vital signs are integrated into Personal Heath Records through IoT gateway.

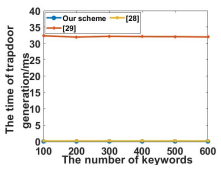

Fig. 5: The time of trapdoor generation

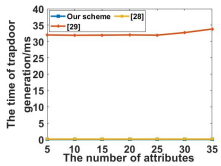

Fig. 6: The time of trapdoor generation

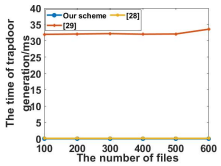

Fig. 7: The time of trapdoor generation

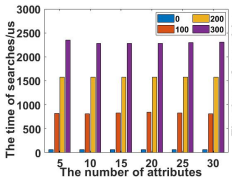

Fig. 8: The time of search

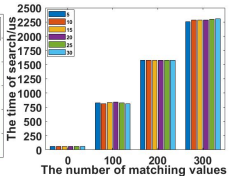

Fig. 9: The time of search

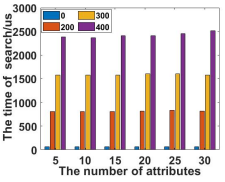

Fig. 10: The time of search

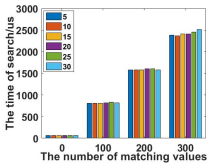

Fig. 11: The time of search

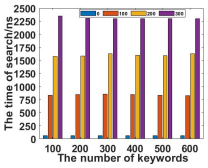

Fig. 12: The time of search

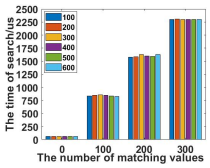

Fig. 13: The time of search

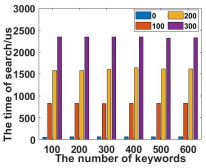

Fig. 14: The time of search

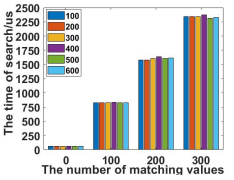

Fig. 15: The time of search

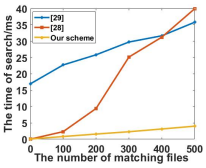

Fig. 16: The time of search

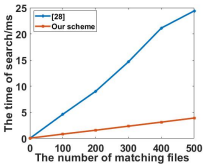

Fig. 17: The time of search

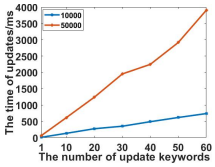

Fig. 18: The time of updates

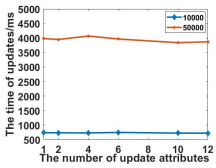

Fig. 19: The time of updates

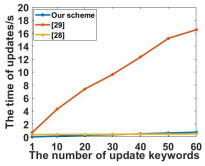

Fig. 20: The time of updates

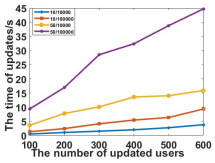

Fig. 21: The time of updates

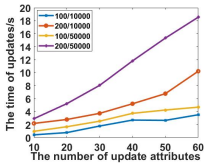

Fig. 22: The time of updates

Our scheme mainly realizes search, dynamic update, forward security and attribute access control based on these Personal Health Records. Collecting data is not the focus of our scheme. Therefore, wearing devices are not involved in our experiment. We use a constrained platform (i.e., 2.4GHz Intel(R) Core(TM) i5 CPU 2GB RAM) to simulate IoT gateway. In order to evaluate the performance of our scheme, we also make comprehensive comparisons with existing schemes [31, 32].

Efficiency of trapdoor generation. When a user (i.e., doctor or nurse) wants to search attribute value containing an interested keyword, she/he generates a search trapdoor and sends it to the cloud server. In order to evaluate the efficiency of trapdoor generation, we show trapdoor generation time according to different numbers of keywords, attributes and files, respectively. As shown in Figs. 5,6,7, the time of trapdoor generation is nearly constant, which is about 0.5 us. Therefore, the time of trapdoor generation is not related with the numbers of keywords, attributes and files. In addition, compared with schemes [31, 32], our scheme is much more efficient for trapdoor generation. Scheme [32] is the least efficient in these three schemes because it involves public key encryption.

Efficiency of Search. In the proposed scheme, the cloud server needs to search based on the secure index and attribute access array. In Figs. 8-15, we show the relationship between the time of searching index T and the number of keywords, the number of attributes and the number returned matched values, respectively. To enhance the accuracy, we repeat 10 times for each experiment and take the average. We show how the number of attributes and the number of matched values impact on the search time when the number of keywords is unchanged in Fig. 8 and Fig. 9, As shown in Fig. 8, we use different colors to represent different numbers of matched values. When the number of keywords and the number of attributes are unchanged, the search time increases with the number of matched values increasing. As shown in Fig. 9, we use different colors to represent different numbers of attributes contained in PHR database. When the number of keywords and the number of matched values are unchanged, the search time is almost unchanged even if the number of attributes increases. We show how the number of attributes and the number of matched values impact on the condition search time when the number of keywords is unchanged in Fig. 10 and Fig. 11. We can observe the experimental result is similar with previous analysis.

We show how the number of keywords and the number of matched values impact on the search time when the number of attributes is unchanged in Fig. 12 and Fig. 13. As shown in Fig. 12, we use different colors to represent different numbers of matched value. When the number of keywords and the number of attributes are fixed, the search time increases with the number of matched values increasing. In Fig. 13, we use different colors to represent the different numbers of attributes. When the number of attributes and matched value is unchanged, as the number of keywords changes, the search time is almost unchanged. Finally, we show how the number of keywords and the number of matching values impact on the condition search time when the number of attributes is unchanged in Fig. 14 and Fig. 15. Moreover, we compare the keyword search efficiency and the condition search efficiency of our scheme with those of schemes [31, 32]. As shown

in Fig. 16, the keyword search efficiency of our scheme is obviously better than schemes [31, 32]. Since the scheme [32] cannot realize condition search, we only compare the condition search efficiency of our scheme with scheme [31]. As presented in Fig. 17, the condition search efficiency of our scheme is obviously better than scheme [31].

Efficiency of secure index update. In Fig. 18 and Fig. 19, we show the time of secure index update. The maximum number of files is set to 10000 and 50000 in these two figures, respectively. We show how the update time changes when the number of attributes is fixed and the number of keywords changes in Fig. 18. As the number of updated keywords increases, the update time increases. We show how the update time changes when the number of update keywords is fixed and the number of attributes changes in Fig. 19. As the number of update attributes increases, the update time is almost unchanged. However, when the largest number of files increases, the update time increases. We compare the update time of our scheme with those of schemes [31, 32], Fig. 20 shows that the update time of scheme [32] is much more than that of our scheme and scheme [31], and the update time of our scheme is almost the same as that of scheme [31].

Efficiency of attribute access array update. In Fig. 21 and Fig. 22, we show the time of attribute access array update. In Fig. 21, we show how the number of update users impacts on the update time. Legend n/m denotes the number of attributes/the allowed largest number of files in the scheme. With the number of updated users increasing, the update time increases linearly. We show how the number of added attributes impacts on the update time in Fig. 22. Legend n/m represents the number of added users/the allowed largest number of files. With the number of added attributes increasing, the update time increases linearly. Moreover, we can observe that the update time increases when the allowed largest number of files increases.

VI. Conclusion

In this paper, we explore how to achieve forward-secure privacy-preserving search based on keyword for IIoTH system. We propose the first DSSE scheme that can be well applied into PHR files database and resist file injection attacks. In order to achieve the design goal, we construct the secure index based on hash chain and realize fine-grained search on the database of attribute-value type. In addition, we achieve efficient access control to protect the privacy of patients' PHR files in our scheme. The detailed security analysis and experiments illustrate the security and efficiency of our scheme.

Acknowledgments

This work was supported in part by the National Natural Science Foundation of China under Grant 61572267; in part by the Joint Found of the National Natural Science Foundation of China under Grant U1905211; in part by the Major Scientific and Technological Innovation project of Shandong Province under Grant 2020CXGC010114; in part by the Key Research and Development Project of Qingdao under Grant 21-1-2-21-XX; and in part by Guangxi Key Laboratory of Cryptography and Information Security under Grant GCIS202101.

References

[1] F. Farshad, A. Rahmani, K. Mankodiya, M. Badaroglu, G.V. Merrett, P. Wong, and B. Farahani. Internet-of-things and big data for smarter healthcare: From device to architecture, applications and analytics. *Future Generations Computer Systems*, 78:583–586, 2018.

[2] L. Yang, Q. Zheng, and X. Fan. RSPP: A Reliable, Searchable and Privacy-Preserving e-Healthcare System for Cloud-Assisted Body Area Networks. In *IEEE Conf. Comput. Commun.(INFOCOM)*, pages 1–9, 2017.

[3] G. Yang, L. Xie, M. Mantysalo, X. Zhou, Z. Pang, L. D. Xu, S. Kao-Walter, Q. Chen, and L. Zheng. A Health-IoT Platform Based on the Integration of Intelligent Packaging, Unobtrusive Bio-Sensor, and Intelligent Medicine Box. *IEEE Transactions on Industrial Informatics*, 10(4):2180–2191, 2014.

[4] H. Huang, T. Gong, N. Ye, R. Wang, and Y. Dou. Private and Secured Medical Data Transmission and Analysis for Wireless Sensing Healthcare System. *IEEE Transactions on Industrial Informatics*, 13(3):1227–1237, 2017.

[5] C. Guo, P. Tian, and K. K. R. Choo. Enabling Privacy-Assured Fog-Based Data Aggregation in E-Healthcare Systems. *IEEE Transactions on Industrial Informatics*, 17(3):1948–1957, 2021.

[6] D. Song, D. Wagner, and A. Perrig. Practical Techniques for Searches on Encrypted Data. In *IEEE Computer Society Symposium on Research in Security and Privacy*, pages 44–55, 2002.

[7] E. Goh. Secure Indexes. *IACR Cryptology ePrint Archive*, 2003:216–234, 2003.

[8] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky. Searchable symmetric encryption: Improved definitions and efficient constructions. *Journal of Computer Security*, 19:895–934, 2011.

[9] D. Boneh, G. D. Crescenzo, R. Ostrovsky, and G. Persiano. Public key encryption with keyword search. In *EUROCRYPT 2004*, pages 506–522, 2004.

[10] Y. Miao, Q. Tong, K. Choo, X. Liu, R. Deng, and H. Li. Secure online/offline data sharing framework for cloud-assisted industrial internet of things. *IEEE Internet of Things Journal*, 6(5):8681–8691, 2019.

[11] Y. Miao, X. Liu, R. Deng, H. Wu, H. Li, J. Li, and D. Wu. Hybrid keyword-field search with efficient key management for industrial internet of things. *IEEE Transactions on Industrial Informatics*, 15(6):3206–3217, 2019.

[12] Y. Miao, X. Liu, K. Choo, R. Deng, H. Wu, and H. Li. Fair and dynamic data sharing framework in cloud-assisted internet of everything. *IEEE Internet of Things Journal*, 6(4):7201–7212, 2019.

[13] Z. Fu, K. Ren, J. Shu, X. Sun, and F. Huang. Enabling personalized search over encrypted outsourced data with efficiency improvement. *IEEE Transactions on Parallel and Distributed Systems*, 27(9):2546–2559, 2016.

[14] Z. Fu, X. Wu, Q. Wang, and K.Ren. Enabling central keyword-based semantic extension search over encrypted outsourced data. *IEEE Transactions on Information Forensics and Security*, 12(12):2986–2997, 2017.

[15] P. Nadkarni and B. Cynthia. Data extraction and ad hoc query of an entity-attribute-value database. *Journal of the American Medical Informatics Association Jamia*, 7:511, 1998.

[16] P. Nadkarni, C. Brandt, and L. Marenco. WebEAV: Automatic Metadata-driven Generation of Web Interfaces to Entity-Attribute-Value Databases. *Journal of the American Medical Informatics Association*, 7:343–356, 2000.

[17] C. Xu, N. Wang, L. Zhu, K. Sharif, and C. Zhang. Achieving Searchable and Privacy-Preserving Data Sharing for Cloud-Assisted E-Healthcare System. *IEEE Internet of Things Journal*, 6(5):8345–8356, 2019.

[18] Y. Miao, J. Ma, X. Liu, F. Wei, and Z. Liu. m^2-abks: Attribute-Based multi-keyword search over encrypted personal health records in multi-owner setting. *Journal of Medical Systems*, 40:246–258, 2016.

[19] Y. Miao, R. Deng, X. Liu, K. Choo, and H. Li. Multi-authority Attribute-Based Keyword Search over Encrypted Cloud Data. *IEEE Transactions on Dependable and Secure Computing*, 2019.

[20] Y. Zhang, D. He, M. S. Obaidat, P. Vijayakumar, and K. Hsiao. Efficient Identity-Based Distributed Decryption Scheme for Electronic Personal Health Record Sharing System. *IEEE Journal on Selected Areas in Communications*, pages 1–11, 2020.

[21] S. Kamara, C. Papamanthou, and T. Roeder. Dynamic searchable symmetric encryption. In *Computer and Communications Security*, pages 965–976, 2012.

[22] X. Ge, J. Yu, H. Zhang, C. Hu, Z. Li, Z. Qin, and R. Hao. Towards Achieving Keyword Search over Dynamic Encrypted Cloud Data with Symmetric-Key based Verification. *IEEE Transactions on Dependable Secure Computing*, 2019.

[23] Y. Zhang, J. Katz, and C. Papamanthou. All your queries are belong to us: the power of file-injection attacks on searchable encryption. In *the 25th USENIX Conference on Security*, pages 707–720, 2016.

[24] E. Stefanov, C. Papamanthou, and E. Shi. Practical Dynamic Searchable Encryption with Small Leakage. *Network and Distributed System Security*, pages 1–15, 2013.

[25] R. Bost. $\sum o\phi o\varsigma$: Forward Secure Searchable Encryption. In *Computer Communications Security*, pages 1143–1154, 2016.

[26] X. Song, C. Dong, D. Yuan, Q. Xu, and M. Zhao. Forward Private Searchable Symmetric Encryption with Optimized I/O Efficiency. *IEEE Transactions on Dependable and Secure Computing*, 17(5):912–927, 2019.

[27] C. Zuo, S. Sun, J. Liu, J. Shao, and J. Pieprzyk. Dynamic Searchable Symmetric Encryption with forward and stronger backward privacy. *European Symposium on Research in Computer Security*, pages 283–303, 2019.

[28] H. Li, Y. Yang, Y. Dai, S. Yu, and Y. Xiang. Achieving secure and efficient dynamic searchable symmetric encryption over medical cloud data. *IEEE Transactions on Cloud Computing*, 8:484–494, 2020.

[29] M. Li, S. Yu, K. Ren, and W. Lou. Securing Personal Health Records in Cloud Computing: Patient-Centric and Fine-Grained Data Access Control in Multi-Owner Settings. In *Security and Privacy in Communication Networks*, pages 89–106, 2010.

[30] UC Irvine Machine Learning Repository. http://archive.ics.uci.edu/ml/index.php.

[31] S. Li, C. Xu, Y. Zhang, X. Wen, K. Chen, and J. Ma. Efficient data retrieval over encrypted attribute-value type databases in cloud-assisted ehealth systems. *IEEE Systems Journal*, pages 1–12, 2021.

[32] P. Xu, S. He, W. Wang, W. Susilo, and H. Jin. Lightweight searchable public-key encryption for cloud-assisted wireless sensor networks. *IEEE Transactions on Industrial Informatics*, 14(8):3712–3723, 2018.

Yaru Liu received the BE degree in information science and engineering from Shandong Normal University, in 2018. She is currently working toward the masters degree in computer science and technology at Qingdao University. Her research interests include cloud computing security and searchable encryption.

Jia Yu received the BS and MS degrees from the School of Computer Science and Technology, Shandong University, in 2000 and 2003, respectively, and the PhD degree from the Institute of Network Security, Shandong University, in 2006. He was a visiting professor with the Department of Computer Science and Engineering, The State University of New York at Buffalo, Buffalo, NY, from 2013 to 2014. He is currently a professor with the College of Computer Science and Technology, Qingdao University. His research interests include cloud computing security, key evolving cryptography, digital signature, and network security.

Jianxi Fan received the BS degree in computer science from Shandong Normal University in 1988, the MS degree in computer science from Shandong University in 1991, and the PhD degree in computer science from City University of Hong Kong, China, in 2006. He is currently a professor of computer science in the School of Computer Science and Technology at Soochow University, China. He visited as a research fellow in the Department of Computer Science, City University of Hong Kong, Hong Kong (october 2006-March 2007, June 2009-August 2009). His research interests include parallel and distributed systems, interconnection architectures, design and analysis of algorithms, and graph theory.

Pandi Vijayakumar received the B.E. degree in computer science and engineering from Madurai Kamaraj University, Madurai, India, in 2002, the M.E. degree in computer science and engineering from the Karunya Institute of Technology, Coimbatore, India, in 2005, and the Ph.D. degree in computer science and engineering from Anna University, Chennai, India, in 2013. He is the former Dean and currently an Assistant Professor with the Department of Computer Science and Engineering, University College of Engineering Tindivanam, Melpakkam, India. He has completed four Ph.D. candidates successfully. He has also authored or coauthored many quality papers in various IEEE transactions/journals, ACM transactions, Elsevier, IET, and Springer journals. His current research interests include key management in network security, VANET security, and multicasting in computer networks. Till now he has authored four books for various subjects that belong to the department of computer science and engineering.

Victor Chang is a Full Professor of Data Science and Information Systems, School of Computing and Digital Technologies, Teesside University, Middlesbrough, UK, since September 2019. He leads Artificial Intelligence and Information Systems Research Group at Teesside University. He was a Senior Associate Professor, Director of Ph.D. and MRes Programs at International Business School Suzhou (IBSS), Xi'an Jiaotong-Liverpool University (XJTLU), Suzhou, China. He joined XJTLU in June 2016. He is still a Visiting Researcher at the University of Southampton, UK. Previously he worked as a Senior Lecturer at Leeds Beckett University, UK, for 3.5 years. Within 4 years, he completed Ph.D. (CS, Southampton) and PGCert (Higher Education, Fellow, Greenwich) while working full time.