IJARSCT
ISSN (Online) 2581-9429
International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)
Volume 2, Issue 1, October 2022
Impact Factor: 6.252

Preserving Security, Privacy of Health Passport Over Patient in Cloud

Fanny Chatola[1] and Dr. G. Glorindal[2]
Student, School of Computer Science and Information Technology[1]
Supervisor, School of Computer Science and Information Technology[1]
DMI-St Eugene University, Chibombo, Zambia
fionachatola@gmail.com and pg@dmiseu.edu.zm

Abstract: *The passport is an expandable document in which key personal, social care and clinical information is recorded. Everyday people move to difference places whether abroad or within the country and some people it might happen they forgot their health passport or they are not registered in any health sectors as a result it might be difficult to have treatment without a health passport. So we designed a Cloud health passport system which is secure, fast and private which will be helpful, using block chain algorithm to allow continuous updating by family and health professionals, acting as a 'live' synchronized record of changing need, service. The passport is set up by the patient (or their agreed representative) and carried on their mobile device across different healthcare settings using Dynamic Searchable Symmetric Encryption (DSSE). It stores demographic details and pertinent clinical information. By having a digital record of this information, valuable patient (and administrative) time is not wasted at registration, improving the patient experience. By incorporating an individual health identifier, it allows a patient to be readily identified across any healthcare setting. The option of storing the dates of upcoming appointments could lead to a reduction in the out-patient non-attendance rate currently estimated at 15%. The increased likelihood that important clinical information is filed to the correct patient record frees up more time for direct patient care.*

Keywords: Authorization Query, Cloud Computing, Personal Health Record (PHR), Privacy-Preserving Query, Intelligent Healthcare, Forward Security, Attribute Access Control, Searchable Encryption

I. INTRODUCTION

The passport is an expandable document in which key personal, social care and clinical information is recorded.**T**he Block – Health Passport & Contact Tracing (HPCT)is constructed to allow a compendium of information about the person with diseases such as their close family members and friends, activities, interests, and beliefs.

The Dynamic searchable symmetric encryption (DSSE) technique can make the user search the dynamic healthcare information from system under the condition that the privacy is protected

Block Chain Algorithm is designed to allow continuous updating by family and health professionals, acting as a 'live' synchronized record of changing need, service contact and information provision. The secure index based on hash chain used to realize trapdoor updates for resisting file injection attacks. In addition, we realize fine-grained search over encrypted Personal Health Records 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.

II. PROBLEM FORMULATION

As shown in the Fig., 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 register to a health sector from cloud using mobile or any system on the cloud server. First of all. They will create an account using the mobile number or emailthen,

IJARSCT
ISSN (Online) 2581-9429
International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)
Volume 2, Issue 1, October 2022
Impact Factor: 6.252

mobile gateway constructs secure index and encrypts each attribute value of each file. Finally, the secure index and the set of encrypted Personal Health Records files are both uploaded to the cloud server, which will formulate a digital passport showing the details of the patient with a QR code.

- Hospital: As the data users, they will access the patient records using the digital passport formulated by the data owners, and registration to their system.
- 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.

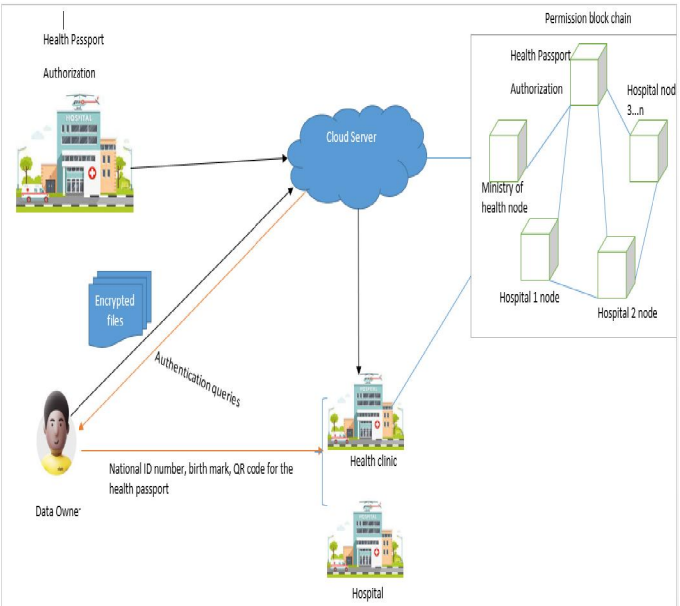


TABLE I: Notations and Descriptions

TABLE II: Attribute-value type database

Value \ Attribute / Ind	att_1	att_2	att_3	...	att_n
ind_1	$V1,1$	$V1,2$	$\perp$	...	$V1,n$
ind_2	$\perp$	$V2,2$	$V2,3$	...	$V2,n$
...	...	...	...	...	...
ind_m	$Vm,1$	$Vm,2$	$Vm,3$	...	Vm,n

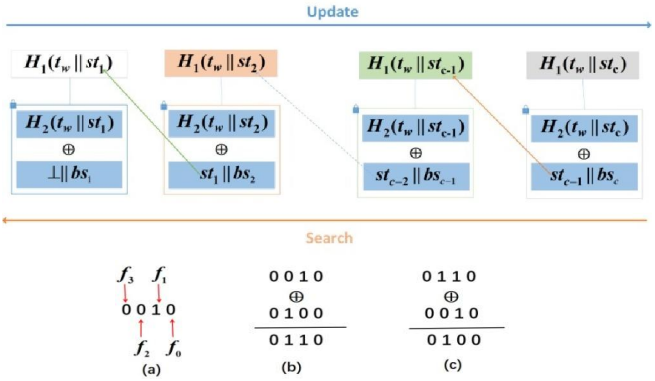
2.1 Design Goals

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.

IJARSCT
ISSN (Online) 2581-9429
International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)
Volume 2, Issue 1, October 2022
Impact Factor: 6.252

- 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)).



Bitmap index

Fig. 3: Secure index

III. CONSTRUCTION OF OUR WORK

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

3.1 The Detailed Construction

The proposed scheme makes the definition of $\Pi = \{Setup, BuildIndex, Update, Search, Dec\}$. First of all, we illustrate some used symbols and functions: $f,g,y,\phi$are pseudorandom 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_{ui}, 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 pseudorandom function g. These functions are defined as follows:

$$f: \{0,1\}^* \times k_{ui} \rightarrow \{0,1\}^{\mu}, g: \{0,1\}^* \times k_w \rightarrow \{0,1\}^{l},$$
$$y: \{0,1\}^* \times k_1 \rightarrow \{0,1\}^{\eta}, \varphi : \{0,1\}^{*}_{\lambda} \times k_2 \rightarrow \{ \quad ^{\varsigma},$$
$$H_1: \{0,1\}^{l+\lambda} \rightarrow \{0,1\}^{\gamma},$$
$$H_2: \{0,1\}^{l+} \rightarrow \{0,1\}^{m+}.$$

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 $\{k_1, k2,ks,kui\}$. Specially, k_1, $k_2$is used to encrypt attributes; k_s is used to encrypt search keyword; k_{ui} 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:

1: $k_1,k_2,k_s,k_{ui} \leftarrow \{0,1\}^{\lambda}$;

Hospital:

2: $k_e \leftarrow SKE.Gen(1^{\lambda})$;

3: Return $K = \{k_1,k_2,k_s,k_{ui},k_e\}$.

IJARSCT
ISSN (Online) 2581-9429
International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)
Volume 2, Issue 1, October 2022
Impact Factor: 6.252

BuildIndex(K, DB; ⊥):

1. Scan the PHR file database *DB* to extract different keywords and construct keyword set W={{*Watt*1},{*Watt*2},⋯,{*Wattn*}};

2. Each attribute value of each PHR file is encrypted separately using symmetric encryption primitive *Cbs,att=SKE.Enc*(*ke,Vbs,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 W, encrypted PHR file and empty maps *T,*Σand *A*. IoT gateway:

1: for each *att*∈ *ATT* do
2: for each *bs*∈ *DB* do
3: Extract keyword and construct keyword set W_{att};
4: *Cbs,att= SKE.Enc*(*ke,Vbs,att*);
5: end for
6: end for
7: Initialize three empty maps *T,*Σand *A*. 8: Return W, $\{C_{bs,att}\}$, *T,*Σ and *A*.

Algorithm 3 Update

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:

1: $\{t\}\leftarrow\emptyset$;
2: $st_c\leftarrow \Sigma[att||w]$, $k_{att}\leftarrow y_{k1}(att)$;
3: $k_w\leftarrow \phi_{k2}(att)$, $t_w\leftarrow g_{kw}(w)$;
4: if $st_c=\perp$ then
5: $st_1\leftarrow\{0,1\}^{\lambda}$, $v\leftarrow H_1(t_w||st_1)$;
6: $e\leftarrow H_2(t_w||st_1)\oplus(\perp||bs)$;
7: else
8: $st_{c+1}\leftarrow\{0,1\}^{\lambda}$, $v\leftarrow H_1(t_w||st_{c+1})$;
9: $e\leftarrow H_2(t_w||st_{c+1})\oplus(st_c||bs)$;
10: **end if**
11: $\Sigma[att||w]\leftarrow st_{c+1}$;
12: for all users $u_i$allowed to access attribute *att*do
13: $t_i\leftarrow f_{ku}i(u_i||att)$, $\{t\}\leftarrow\{t\}\cup t_i$;

IJARSCT
ISSN (Online) 2581-9429
International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)
Volume 2, Issue 1, October 2022
Impact Factor: 6.252

14: end for
15: Send $(v,e,\{t\},bs)$ to the cloud server.
16: $T[v] \leftarrow e$;
17: for $t_i \in \{t\}$ do
18: $z \leftarrow A[t_i] \oplus bs$, $A[t_i] \leftarrow z$;
19: end for 20: 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
Input: The secret key k_{ui},k_{uj}, 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:
1: $\{d\},\{a\},\{bs\} \leftarrow \emptyset$;
2: for $att_i \in ATT$ do
3: $a_i = f_{kui}(u_i||att_i)$, $d_i = f_{kuj}(u_j||att_i)$;
4: $\{a\} \leftarrow \{a\} \cup a_i$, $\{d\} \leftarrow \{d\} \cup d_i$;
5: 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;
6: $\{bs\} \leftarrow \{bs\} \cup bs_i$
7: end for
8: Send $(\{a\},\{bs\},\{d\})$ to the cloud server.
9: for $a_i \in \{a\}, bs_i \in \{bs\}$ do
10: $A[a_i] \leftarrow A[a_i] \oplus bs_i$;
11: end for
12: for $d_i \in \{d\}$ do
13: Remove $A[d_i]$;
14: end for 15: Return A.

IJARSCT
ISSN (Online) 2581-9429
International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)
Volume 2, Issue 1, October 2022
Impact Factor: 6.252

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_cin the secure index. Different users have different access authorities to the attributes of files.

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 $st_c=\perp$ then
3: Return ∅;
4: else
5: $k_{att} \leftarrow y_{k1}(att)$, $k_w \leftarrow \phi_{k2}(att)$;
6: $t_w \leftarrow g_{kw}(w)$, $s \leftarrow f_{ku}(u||att)$;
7: end if
8: Send (k_{att}, st_c, t_w, s) to the cloud server.
9: $sum_e \leftarrow 0$, $st \leftarrow st_c$;
10: while $st_c 6=\perp$ 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.

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

IJARSCT
ISSN (Online) 2581-9429
International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)
Volume 2, Issue 1, October 2022
Impact Factor: 6.252

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$(K,att^0,att,w,st_c;EDB,T,A)$: When the user would like to search the value of another attribute att^0 on the matched condition that the attribute *att*contains the search keyword w, the user generates the trapdoors by encrypting keyword w contained in attribute *att*value and attributes att^0. 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^0. 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 $V_{r,att} \leftarrow Dec(k_e,C_{r,att})$.

3.2 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	⊥	Cough, runny nose, sore throat
0010	Bob	Male	Penicillin	Fatigue, stuffy, fever, headache
0100	Alice	Female	⊥	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 W:
  W={*WName,WGender,WAllgery,WSymptom*}, where W_{Name}= {*Alice,Bob*} et al.;
- Construct the file set containing keyword:
  DB(*Alice*) = {0001,0100}, *DB*(*Bob*) = {0001};
- Encrypt attribute: $k_{Name1}=y_{k1}$(Name), *kName*2=ϕk2(Name);
- Encrypt keyword: $t_{Alice}= g_{kName2}(Alice)$, *tBob*= *gkName*2(*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:
  Cbs,att= *SKE.Enc*(*ke,Vbs,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

IJARSCT
ISSN (Online) 2581-9429
International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)
Volume 2, Issue 1, October 2022
Impact Factor: 6.252

TABLE IV: Attribute access array A

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

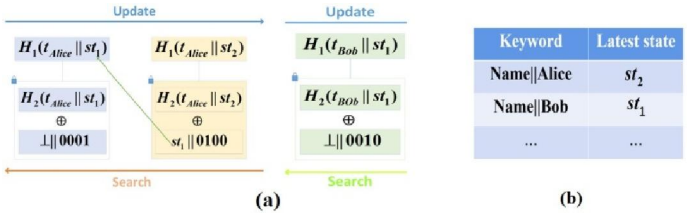


Fig. 4: Secure index and state array

Fig. 4.(b). We give a simple example of condition search to 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 trap doors:

$tAlice= gkName(Alice), kName= \phi k2(Name), kAllergy= y_{k1}(Allergy)$. 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 $s = f_{ku}d(u_d,Allergy)$ 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_{d1} 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 $y,\phi$and g are PRFs, and H_1 and H_2 are Hash functions. Leakage functions L = (L*Setup,* L*Update,* L*Search*) can be defined as follows:

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

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 L− *adaptively* − *secure* SSE scheme with forward security. Proof. We prove this Theorem through a series of games. Starting from $Real^{\Pi}_{A}(\lambda)$, 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 $Ideal^{\Pi}_{\mathcal{A},\mathcal{S}}(\lambda)$. Therefore, we know $Real^{\Pi}_{A}(\lambda)$ is indistinguishable from $Ideal^{\Pi}_{\mathcal{A},\mathcal{S}}(\lambda)$ according to the property of indistinguishability.

Game G_1: G_1 has no difference with $Real^{\Pi}_{A}(\lambda)$ except that instead of using y to generate $k_{att}$and using g to generate t_w, the game uses maps $Token_1$ and $Token_2$ 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 $Token_1$ 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 $Token_1$ in the form of (w,t_w) pairs.

IJARSCT
ISSN (Online) 2581-9429
International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)
Volume 2, Issue 1, October 2022
Impact Factor: 6.252

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.

Can use uh instead of up in G_4 to decide the update queries about keyword w. The simulator S generates the view that is difficult to be distinguished from the one generated in G_4. $Pr[\text{Ideal}^{\Pi}_{\mathcal{A},S}(\lambda)=1]-Pr[G_4=1]=($

From above all, we have

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

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.

5.1 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	×	×	×			

5.2 Performance Analysis

Experiments are run on a Linux OS equipped with 2.4 GHz 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 patient's vital signs. After that, these collected vital signs are integrated into Personal Heath Records through IoT gateway.

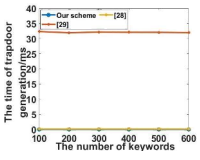


Fig. 5: The time of trapdoorgeneration

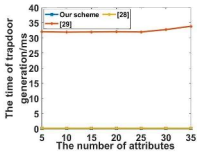


Fig. 6: The time of trapdoorgeneration

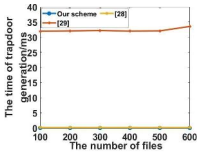


Fig. 7: The time of trapdoorgeneration

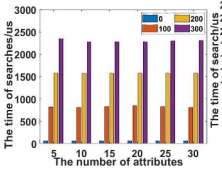


Fig. 8: The time of search

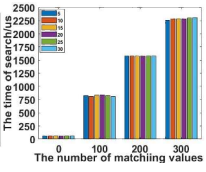


Fig. 9: The time of search

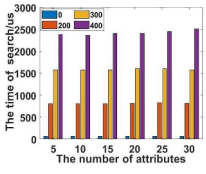


Fig. 10: The time of search

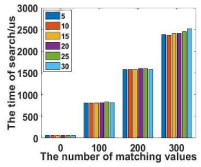


Fig. 11: The time of search

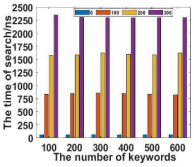


Fig. 12: The time of search

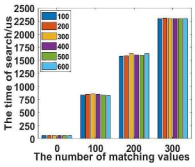


Fig. 13: The time of search

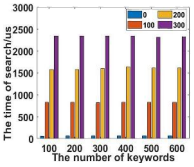


Fig. 14: The time of search

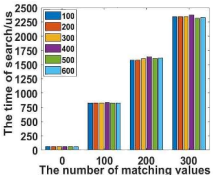


Fig. 15: The time of search

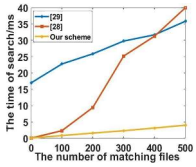


Fig. 16: The time of search

IJARSCT
ISSN (Online) 2581-9429
International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)
Volume 2, Issue 1, October 2022
Impact Factor: 6.252

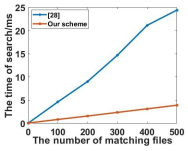


Fig. 17: The time of search

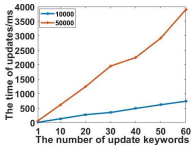


Fig. 18: The time of updates

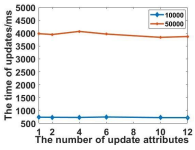


Fig. 19: The time of updates

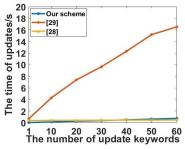


Fig. 20: The time of updates

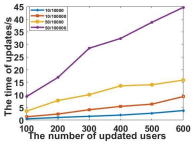


Fig. 21: The time of updates

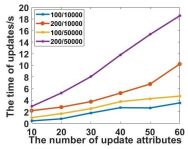


Fig. 22: The time of updates

Our scheme mainly realizes search, dynamic update, and 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

IJARSCT
ISSN (Online) 2581-9429
International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)
Volume 2, Issue 1, October 2022
Impact Factor: 6.252

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 in health and creation of a health passport 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.

REFERENCES

[1]. S. M. Metev and V. P. Veiko, Laser Assisted Microtechnology, 2nd ed.,R. M. Osgood, Jr., Ed. Berlin, Germany: Springer-Verlag, 1998.

[2]. J. Breckling, Ed., The Analysis of Directional Time Series: Applications to Wind Speed and Direction, ser. Lecture Notes in Statistics. Berlin, Germany: Springer, 1989, vol. 61.

[3]. S. Zhang, C. Zhu, J. K. O. Sin, and P. K. T. Mok, "A novel ultrathin elevated channel low-temperature poly-Si TFT," IEEE Electron Device Lett., vol. 20, pp. 569–571, Nov. 1999.

[4]. M. Wegmuller, J. P. von der Weid, P. Oberson, and N. Gisin, "Highresolution fiber distributed measurements with coherent OFDR," in Proc. ECOC'00, 2000, paper 11.3.4, p. 109.

[5]. R. E. Sorace, V. S. Reinhardt, and S. A. Vaughn, "High-speed digital-to-RF converter," U.S. Patent 5 668 842, Sept. 16, 1997.

[6]. (2002) The IEEE website. [Online]. Available: http://www.ieee.org/

[7]. M. Shell. (2002) IEEEtran homepage on CTAN. [Online]. Available: http://www.ctan.org/tex-archive/macros/latex/contrib/supported/IEEEtran/

[8]. FLEXChip Signal Processor (MC68175/D), Motorola, 1996.

[9]. "PDCA12-70 data sheet," Opto Speed SA, Mezzovico, Switzerland.

[10]. Karnik, "Performance of TCP congestion control with rate feedback:TCP/ABR and rate adaptive TCP/IP," M. Eng. thesis, Indian Institute ofScience, Bangalore, India, Jan. 1999.

[11]. J. Padhye, V. Firoiu, and D. Towsley, "A stochastic model of TCP Renocongestion avoidance and control," Univ. of Massachusetts, Amherst,MA, CMPSCI Tech. Rep. 99-02, 1999.

[12]. Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specification,IEEE Std. 802.11, 1997.