1-2018

A lightweight policy preserving EHR sharing scheme in the cloud

Zuobin YING

Lu WEI

Qi LI

Ximeng LIU
Singapore Management University, xmliu@smu.edu.sg

Jie CUI

IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received July 28, 2018, accepted September 14, 2018, date of publication September 19, 2018, date of current version October 17, 2018.

Digital Object Identifier 10.1109/ACCESS.2018.2871170

A Lightweight Policy Preserving EHR Sharing Scheme in the Cloud

ZUOBIN YING[1], (Member, IEEE), LU WEI[1], QI LI[2], XIMENG LIU[3,4], (Member, IEEE), AND JIE CUI[1]

[1]School of Computer Science and Technology, Anhui University, Hefei 230601, China
[2]School of Computer Science, Nanjing University of Posts and Telecommunications, Nanjing 210023, China
[3]College of Mathematics and Computer Science, Fuzhou University, Fuzhou 350116, China
[4]School of Information Systems, Singapore Management University, Singapore 188065

Corresponding author: Jie Cui (cuijie@mail.ustc.edu.cn)

This work was supported in part by the National Natural Science Foundation of China under Grant U1405255, Grant 61572001, Grant 61502008, Grant 61702005, Grant 61502248, Grant 61427801, and Grant 61702105, and in part by the China Postdoctoral Science Foundation under Grant 2018M632350.

ABSTRACT Electronic Health Record (EHR) is a digital health documentary. It contains not only the health-related records but also the personal sensitive information. Therefore, how to reliably share EHR through the cloud is a challenging issue. Ciphertext-policy attribute-based encryption (CP-ABE) is a promising cryptography prototype, which can achieve fine-grained access control as well as one-to-many encryption. In CP-ABE, access policy is attached to the ciphertext, and however, the access policy is not protected, which will also cause some privacy leakage. In this paper, we propose a policy preserving EHR system on the basis of CP-ABE. Specifically, we designed an algorithm, which can hide the entire access policy as well as recover the hidden attributes from the access matrix. The subsequent evaluation of element insert, lookup, and recovery shows that our proposed scheme only introduces light-weighted overhead cost. The security analysis indicates that the scheme is selectively secure under q-BDHE assumption.

INDEX TERMS Attribute cuckoo filter (ACF), access control, cloud computing, CP-ABE, Electronic Health Record (EHR), policy preserving.

I. INTRODUCTION

Electronic Medical Record (EMR) is a systematized digital record which contains the detailed health information of a patient and population. The initially conception of EMR is to take place of traditional papery medical record, so as to improve the management of cases in a health-care institution. However, due to the increasingly concern of self-health, general population also want to obtain and manage their own health information in some way. Thus, a novel personalized health information management system called Electronic Health Record (EHR) became popular. By using EHR, on the one hand, users are able to manage their own health records. They can fill the corresponding columns with some physical symptoms obtained from wearable devices or the physical examinations from medical institutions at anytime. On the other hand, users are able to monitor their own health variation trends up to date. Therefore, the new health management style attracts more and more attention world-widely. Health-care enterprises tailored for different individuals with personalized EHR systems, such as, Praxis, WRS Health, Medent, etc [1]. Health-care institutes also permit the *patient-centric* management pattern. Healthy ecospheres are built upon the basis of these EHR systems.

If all the benefits mentioned above are from the users' aspect, then the following plenty of advantages are listed on account of diagnosis. According to athenahealth "2012 Physician Sentiment Index" [2]. 81% of physicians said they believe EHRs improve access to clinical data. More than two-thirds said an EHR can actually improve patient care. EHR reduces the paperwork of the clinician, and help to establish and maintain effective clinical work-flows. More importantly, EHR provides an opportunity to interact with affiliated hospitals, clinics, labs, and pharmacies seamlessly.

In the modern health-care environment, cloud computing plays an important role. EHR providers are willing to outsource their EHR to the cloud for the purpose of accessing them at *anytime* and *anywhere*. However, the cloud server is assumed to be semi-trusted, which means it will sniffer

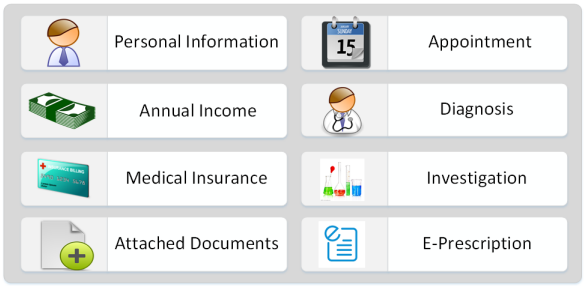


FIGURE 1. Functions of an EHR.

as much personal information as it could besides following the orders given by the users. It can be seen in Fig.1 that EHR consists of various types of items more than health-related only. For example, the annual income, certificate of medical insurance, e-prescription, etc. Thus, the outsourcing of plaintext EHR to the cloud will lead to a high potential risk of personal privacy leakage. Recently, for instance [3], the health ministry of Singapore confess that hackers attacked an EHR database of a health-care corporation. About a million and a half users' private information suffers from illegal access. This means one out of four individuals of Singapore, including the prime minister, encounter personal privacy disclosure. Therefore, the privacy protection of the outsourcing EHR became an ongoing research focus [4]. The mainstream of protecting EHR in the cloud can be summarized into *Cryptography* and *Non-cryptology* methods in general. Here, we focus on the avoidance of unauthorized disclosure of sensitive information, videlicet, secure fine-grained access control.

Fine-grained access control mechanisms have been widely studied and adopted in various research fields. Whereas, classical access control mechanisms (e.g., access control list) cannot meet the needs of EHR in the cloud environment. Firstly, the outsourcing of plaintext EHR to the cloud without any assurances of privacy and security protection, the benefits of the EHR providers will be compromised. Secondly, traditional cryptography scheme is incapable to achieve one-to-many encryption which is affirmatively required in the cloud sharing scenarios. Therefore, a brand-new cryptography prototype named Fuzzy identity-based encryption was introduced, which is also known as Attribute-Based Encryption (ABE) [5]. ABE can be evolved into Key-Policy ABE (KP-ABE) [6] or Ciphertext-Poicy ABE (CP-ABE) [7] by applying policy either to the key side or the ciphertext side. With the help of CP-ABE, EHR providers are able to formulate the appropriate policies according to the content of the EHRs they want to share through the cloud. EHR consumers (e.g. clinician, assurance company employee) can decrypt the EHR files only if their attributes could satisfy the policy attached in the ciphertext. It seems that CP-ABE is a perfectly matched method for the secure sharing of EHR in the cloud environment. However, it suffers from a serious problem: The access policy would reveal the privacy, for the access policy is not in a ciphertext form, thus adversary can obtain some sensitive information about the EHR consumers as well as the EHR providers through the policy. For example, Bob have cardiopathy, he encrypts his health record files using the policy ((''*cardiologist*'' and ''*female*'') and ''*Peking Union Medical College Hospital*''). An adversary could easily get that Bob has some cardiologist disease, and he/she may still guess that Bob is in Peking Union Medical College Hospital (PUMCH) in all probability, which reveals the privacy of Bob. Additionally, if he/she finds that Alice could fetch and get access to the EHR. He/she can speculate that Alice is a cardiologist doctor working in PUMCH, which also reveals the privacy of Alice.

In order to prevent privacy leakage through the access policy. Some policy preserving schemes have been proposed. A straightforward way is to hide the attributes in the policy [8], [9]. However, decryption became challenging since neither the authorized users nor the adversaries know what attributes are contained in the access policy. Therefore, the subsequent schemes try to divide the attributes into two parts: ''attribute name'' and ''attribute value'' (e.g., *occupation*: *cardiologist*) [10], [11]. The attribute values are hidden by using the wildcards while the attribute names can still be seen in the access structure which will reveal some information anyway. Moreover, these partially hidden policy schemes either have a special request on the expression of access structure (e.g., AND-gates on multi-valued attributes) or have a low efficiency in practical terms [12].

The main idea of our work is to hide the entire access structure of the EHR instead of hiding the attribute values only. Besides, the expressiveness of the access structure is also the main aspect of our architecture. We decide to use the basic idea of linear secret sharing scheme (*LSSS*) since it is a widely acknowledged access structure. The *LSSS* structure is formulated as $(\mathbb{M}, \rho)$, in which $(\mathbb{M})$ represents a policy matrix and function ρ maps each row M_i of $\mathbb{M}$ to an attribute [13]. For the purpose of hiding the access policy, the function ρ should be replaced. Moreover, since the policy is completely hidden, an attribute matching algorithm should be designed to decide and precisely locate the attributes right in the anonymous access policy. Therefore, we designed a new element filter named Attribute Cuckoo Filter (ACF). ACF helps to locate the attributes in the hidden access policy, and can also save a lot of computation cost as well as storage overhead. The main contribution of our work can be summarized as follows.

1) A novel EHR system with policy hidden CP-ABE is constructed. The entire policy of EHR is hidden instead of hiding the attribute values only to enhance the security property. In detail, before outsourcing EHR to the cloud server, we separate the access structure from the ciphertext, then convert the access matrix into a hidden status. Afterward, the converted access matrix will be outsourced to the cloud along with the ciphertext. Since nobody knows the specific attribute in the access policy, thus the policy security can be guaranteed. However, the authorized EHR consumer wants to decrypt the EHR, there has to be another reliable and efficient

way to match their attributes to the ciphertext in the hidden policy matrix.

2) We designed a feasible and efficient attribute matching algorithm to decide whether an attribute is in an anonymous access policy which we named Attribute Cuckoo Filter (ACF). If so, ACF is also capable to locate and recover the attributes accurately. Specifically, we put forward a multiple table ACF to avoid the element inserting collision. Compared with the schemes of same kind. Our proposed scheme have an obvious improvement in efficiency.

3) We proved the security of our scheme, and the experiment result indicates that our scheme can protect the access policy by only introducing a small overhead.

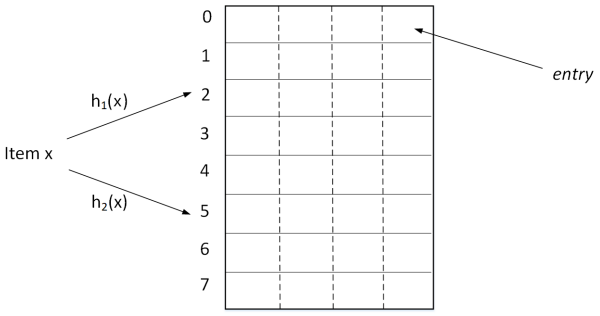


FIGURE 2. A demonstration of cuckoo filter, two hash per item and four entries per bucket.

II. RELATED WORK

Cloud storage separates the ownership and the control benefit of the EHR provider. Besides, it is widely believed that the cloud server cannot be fully trusted (i.e. semi-trusted). That is to say, it will follow the protocols, but tries to explore as much privacy of the data as it could [14], [15]. Intuitively, encrypt the EHR before outsourcing them to the cloud is an effective solution. However, traditional public key encryption (PKE) cannot satisfy the needs of one-to-many encryption. It has to distribute different private key for different users for decryption. Meanwhile, there should be the same amount of copies of ciphertext. To tackle this problem, Attribute-Based Encryption (ABE) was put forward [5]. Yu *et al*. [16] construct the first fine-grained access control scheme in the cloud by using KP-ABE. After that, fruitful researches of using KP-ABE and CP-ABE lay the foundation for preserving the privacy in the cloud [17], [18]. However, policies in these works are not protected. Kapadia *et al*. [8] proposed a secure Attribute-Based publishing scheme with hidden credential and hidden policies, it uses the wildcards to hide the attributes in the policy. But the scheme cannot resist collusion attack. Nishide *et al*. [10] put forward two ABE schemes with policy partially hidden, nevertheless, the two constructions support the same expressiveness of policy (e.g. AND-gates on multi-valued attributes with wildcards). Li *et al*. [11] improved the attribute hiding method by using a hash value to demonstrate the corresponding value of an attribute. Lai *et al*. presented a fully secure policy hiding CP-ABE. The security model is better than [10] and [11], but the policy expression is still restricted to be AND-gates on multi-valued attributes with wildcards. Moving one step forward, Lai *et al*. proposed a fully secure CP-ABE with partially hidden policy using *LSSS* structure [12], but the scheme is constructed on the composite-order group, so the efficiency is extremely lower than the constructions build on the prime order group.

Yang *et al*. [19] attempt to solve the policy preserving issue through another aspect. Their key to solving the problem is how to match the attributes correctly to the hidden policy. They proposed a novel attribute matching algorithm called Attribute Bloom Filter (ABF) to find and correspond the attributes to the relevant row in the access matrix. However, bloom filter (BF) has a non-negligible false positive probability. That is to say, BF could estimate an attribute which is actually not in the policy to be in the access structure by mistake, which will cause some unauthorized access. Besides, the proposed ABF cannot completely solve the hash value conflicting issue.

The remaining of this paper contains the following contents. First, we give a brief introduction of the related works in Section II. In Section III, the preliminaries are presented. After that, we provide the system model, then the definition of the scheme as well as the security model in Section IV. Our CP-ABE scheme is detailed in Section V. Security analysis and the performance evaluation are discussed in Section VI. Finally, conclusion of the paper is given in Section VII.

III. PRELIMINARIES

A. CUCKOO FILTER

Cuckoo Filter (CF) is a novel way to approximately estimate whether a given item is in a set [22]. Cuckoo filter consists of an array of buckets as shown in Fig.2. The buckets contains multiple basic units called *entry*. Each entry stores one fingerprint.[1] In order to optimize the space efficiency, CF utilizes a technique named *partially-key cuckoo hashing* to generate the fingerprint of the inserting item. The fingerprint will be stored at either two of the candidate locations in the bucket.

Initially, the buckets are all empty. To add an element x to the bucket, CF first use *partially-key cuckoo hashing* to generate the fingerprint of x. Then the alternate location can be determined based on the fingerprint by using the following equations:

$$\begin{aligned} h_1(x) &= hash(x), \\ h_2(x) &= h_1(x) \oplus hash(x's\ fingerprint). \end{aligned} \tag{1}$$

The important property of xor operation in (1) makes the calculation of $h_1(x)$ from $h_2(x)$ and the fingerprint using the same formula and vice versa.

To check whether a given element y is in the bucket. The algorithm first calculates the fingerprint of y, then the two candidate buckets according to (1). CF returns true if any existing fingerprint in either bucket matches. Otherwise,

[1]The fingerprint is a constant-sized hash value of the item to be insert

CF returns false, which means the element y is not in the bucket.

B. BILINEAR PAIRING

A bilinear pairing has the properties of bilinear and non-degenerate as well as computable detailed as follows: Let $\mathbb{G}$ and $\mathbb{G}_T$ be two groups of multiplicative. Let g be a generator of $\mathbb{G}$. Then, the bilinear mapping function $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$ has the following properties:

1) Computable: For all $u, v \in \mathbb{G}$, $e(u, v)$ could be computed efficiently.

2) Bilinear: For all $x, y \in \mathbb{Z}_p$, $e(g^x, g^y) = e(g, g)^{xy}$.

3) Non-degenerate: $e(u, v) \neq 1$ where $u, v \in \mathbb{G}$.

C. LINEAR SECRET SHARING SCHEMES

Definition 1 ((LSSS) [13]): A secret-sharing scheme Π over a set of parties $\mathcal{P}$ is referred to as linear over $\mathbb{Z}_p$ if:

1) The shares for each party form a vector over $\mathbb{Z}_p$.

2) A matrix A with l rows and n columns exists, which is called the share-generating matrix for Π. For any i'th row of A, where $i = 1, \ldots, l$, let $\rho(i)$ be defined as the party labeling row i. Then the secret $s \in \mathbb{Z}_p$ to be shared can be constructed by a column vector $v = (s, r_2, \ldots, r_n)$, where $r_2, \ldots, r_n$ are randomly chosen from $\mathbb{Z}_p$. Then the vector Av is the vector of l shares of the secret s according to Π. The share $(Av)_i$ belongs to party $\rho(i)$.

Beimel *et al*. [21] indicates that any linear secret sharing-scheme according to the definition above also have the property of *linearreconstruction*. The definition is as follows: Suppose Π is an *LSSS* for access structure $\mathbb{A}$. Let $S \in \mathbb{A}$ be any authorized sets, and let $I \subset 1, 2, \ldots, l$ be defined as $I = i : \rho(i) \in S$. Then there exist constants $\{\omega \in \mathbb{Z}_p\}_{i \in I}$ so that $\Sigma_{i \in I} \omega_i \lambda_i = s$ if λ_i are valid shares of any secret s according to Π. Moreover, constants $\{\omega_i\}$ can be found in poly-time in the size of the share-generating matrix A.

D. COMPLEXITY ASSUMPTION (DECISIONAL Q-BDHE ASSUMPTION)

Choose a group $\mathbb{G}$ of prime order p and with g as a generator according to the security parameter λ. Randomly choose $a, s \in \mathbb{Z}_p^*$. Denote g_i as g^{a^i}. The adversary must distinguish $e(g, g)^{a^{q+1}s} \in \mathbb{G}_T$ from a random element R in $\mathbb{G}_T$ when given $\vec{y} = (g, g_1, \ldots, g_q, g_{q+2}, \ldots, g_{2q}, g^s)$. An algorithm $\mathcal{B}$ has advantage ϵ in solving decisional q-BDHE problem in $\mathbb{G}$ if

$$|Pr[\mathcal{B}(\vec{y}, T = e(g, g)^{a^{q+1}s}) = 0] - Pr[\mathcal{B}(\vec{y}, T = R) = 0]| \geq \epsilon.$$

Definition 2: We say that the Decisional q-BDHE assumption holds if no poly-time algorithm has a non-negligible advantage in solving the q-BDHE problem.

IV. DEFINITIONS

In this section, we give a brief introduction of the policy hidden EHR system model and the designing goal of it. Then we define the proposed scheme as well as the security model.

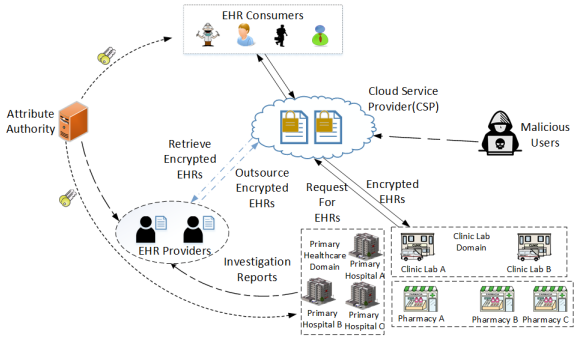


FIGURE 3. Policy Preserving EHR System Overview.

A. SYSTEM MODEL

We consider the system model demonstrated in Fig.3 consists of five parts, namely Cloud Service Provider (CSP), Attribute Authority (AA), EHR Providers (EP), EHR Consumer (EC) and Malicious Users (MU).

1) CSP provides with efficient computing, mass storage and other Internet services to the EP and EC. CSP will follow the instructions issued by all the consumers (include the malicious users). In our scheme, CSP is assumed not to collude with the malicious users. However, CSP itself will try to sniffer as much privacy as it could. So it cannot be fully trusted.

2) All the attributes in the system is managed by AA. It generates the public parameter and the master secret key. AA also responses for issuing private keys to the EHR consumers according to their attributes. AA is assumed to be fully trusted in our scheme.

3) EP are entities who own the EHRs and intend to outsource them to the CSP. They answer for their own EHRs' creation, management and completion. They also have the right to formulate the access policy to decide what kind of EHR consumer is authorized to the data. EP are assumed to be honest.

4) ECs in our proposed policy preserving EHR can be of variant kinds of entities, including patients, clinicians, assurance company employees, health-care institutes, etc. EC requests the authorization of encrypted data from CSP. The decryption can only be processed if their attributes satisfy the access policy attached to the encrypted EHRs. However, sometimes, a single EC may not have enough attributes to satisfy the policy, so they may collude with each other to gain more profits. ECs cannot be fully trusted.

5) Malicious Users represent the hackers as well as the unauthorized users. They try to break through our proposed scheme to mining both ECs' and EPs' privacy through access policies attached with EHR ciphertexts. Specifically, MU in our system have the polynomial time capacity.

Our proposed policy preserving EHR system consists of four algorithms listed as follows: ***Setup***, ***KeyGen***, ***Encrypt*** and ***Decrypt***.

TABLE 1. Feature comparison.

	Expressiveness of access structure	Security	Policy Hidden
Nishide [10]	AND-gates on multi-valued attributes with wildcards	Selective	Partially Hidden
Lai [12]	*LSSS*	Adaptive	Partially Hidden
Yang [20]	*LSSS*	Selective	Fully Hidden
Ours	*LSSS*	Selective	Fully Hidden

• ***Setup***$(1^{\lambda}) \rightarrow (PK,MSK)$: The Setup algorithm takes the security parameter λ as a input. It outputs the public key and the master secret key.

• ***KeyGen*** $(PK,MSK,U) \rightarrow (SK)$: AA initials the key generation algorithm. It takes as inputs the public key PK, the master secret key MSK as well as a set of attribute U. It outputs the corresponding secret key SK.

• ***Encrypt*** $(PK,(\mathbb{M}, \rho), M) \rightarrow (CT, ACF)$: The encrypt algorithm is composed of two sub-algorithm, namely, ***Enc*** and ***ACF-Create***.

– ***Enc*** $(PK, M, (\mathbb{M}, \rho)) \rightarrow CT$: This sub-algorithm takes as inputs the PK, the access structure $(\mathbb{M}, \rho)$ as well as the plaintext M. It takes the ciphertext CT as an output.

– ***ACF-Create***$(\mathbb{M}, \rho) \rightarrow ACF$: This sub-algorithm takes as inputs the access structure $(\mathbb{M}, \rho)$ and outputs the ACF table.

• ***Decrypt*** $(PK, ACF, \mathbb{M}, SK, CT) \rightarrow M$: The decrypt algorithm also contains two sub-algorithm, namely, ***ACF-Check*** and ***Dec***.

– ***ACF-Check*** $(PK, ACF, U) \rightarrow \rho'$: This sub-algorithm takes as inputs the public key PK, the attribute set S and the ACF table. It outputs the attribute in the access policy as well as the location in the $LSSS$ matrix.

– ***Dec***$(SK, CT, ((\mathbb{M}), \rho')) \rightarrow M$ or $\perp$: This sub-algorithm takes the secret key SK, the ciphertext CT and the $LSSS$ matrix $\mathbb{M}$ as well as the outputs of sub-algorithm ACF-Check. It outputs plaintext M if the attributes satisfy the policy, or $\perp$ if not.

B. DESIGN GOAL

Based on the aforementioned system model as well as the security threats, we develop an efficient and reliable fine-grained EHR access control scheme with policy preserving became our design goal. The features of our proposed scheme compared with the other state-of-art schemes are listed in Table 1.

• *Reliable*: Without the basic requirements of security, personal information, clinical diagnosis, and other individual privacy will be disclosed to the malicious users, no EP will be rest assured towards the whole system. So we should consider about the security of EHR first.

• *Fine-grained access control*: In the cloud-based EHR system, providers hope to formulate the appropriate access policy towards different consumers. That is to say, the fine-grained access control mechanism should be involved in our system. It helps the EP to strictly limit the access to their own EHR files to decrease the privacy leakage risk.

• *Policy preserving*: As mentioned before, policies attached with the ciphertext in traditional attribute-based EHR systems are not protected, which will reveal the privacy of both the EP and the EC. So this is the main design goal of our system. Moreover, we should design a policy matching algorithm in order to match the attributes within the hidden policy, then recover the corresponding ones. The ABF scheme proposed by Yang *et al.* [19] launches a Garbled Bloom Filter to deal with the policy hiding issue. It opens up a new idea of solving the policy preserving problem. Moving one step forward, we plan to design a better algorithm in order to solve the element conflict problem besides policy preserving.

• *Efficiency*: Most of the existing CP-ABE scheme with policy hidden or partially hidden is build on the composite order group, which is inefficient compared with the ones build on the prime order group.

C. SECURITY MODEL

We take the indistinguishability against selective chosen plaintext attacks (CPA) into consideration. It is designed on the bases of a game between an adversary $\mathcal{A}$ and a simulator $\mathcal{B}$.

• *Init*: The adversary $\mathcal{A}$ chooses a challenge access structure $(\mathbb{M}^*, ACF^*)$, where $\mathbb{M}^*$ is an $l^* \times n^*$ matrix.

• *Setup*: The challenger runs the Setup algorithm and gives the public parameter PK to the adversary $mathcalA$.

• *Phase* 1: In this phase, the adversary $\mathcal{A}$ issues secret key queries correlated with attribute list Att_Q.

– If $Att_Q \in (\mathbb{M}^*, ACF^*)$, then aborts.

– Otherwise,the simulator generates a secret key related to Att_Q for the adversary $\mathcal{A}$.

• *Challenge*: The adversary $\mathcal{A}$ submits two equal length messages M_0 and M_1 to the simulator $\mathcal{B}$. $\mathcal{B}$ chooses $\tau \in \{0, 1\}$ at random and encrypts M_{tau} under the challenge access structure $(\mathbb{M}^*, ACF^*)$. Finally it sends the generated challenge ciphertext CT^* to $\mathcal{A}$.

• *Phase* 2: *Phase* 2 is the same as *Phase* 1.

• *Guess*: the adversary outputs a guess τ' of τ. The advantage of $\mathcal{A}$ in this game is defined as:

$$Adv(\mathcal{A}) = |Pr[\tau' = \tau] - 1/2|.$$

V. CONSTRUCTIONS OF OUR SCHEME

We construct our scheme by utilizing the Waters CP-ABE [13] as a building block. Apparently, our scheme can easily extend to other CP-ABE schemes with the structure expressed in $LSSS$ form. For making the proposed scheme more comprehensive, we give the notation of the parameters in Table 2. The simplified schematic diagram is demonstrated in Fig.4.

We briefly describe the entire workflow of our system.

1) After initializing, Attribute Authority generates the public parameter and the master secret key MSK, the public

TABLE 2. Notations.

Symbols	Descriptions
$PK,\ SK$	public/private key pairs
MSK	system master secret key
$(\mathbb{M}, \rho)$	access policy in original ABE
$ACF,\ ACF*$	attribute cuckoo filter
CT	ciphertext
H_e	ACF hash for the insert element e
H_f	ACF hash to generate the fingerprint of element e
$h_1, \ldots, h_n$	hash the attributes into group $\mathbb{G}$
L_{rnum}	string length of row number in the *LSSS* matrix
L_{att}	string length of attribute in the *LSSS* matrix
U	a set of attribues

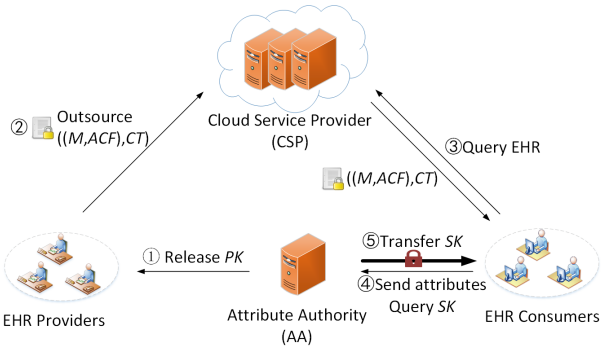


FIGURE 4. Simplified schematic diagram.

parameter will be distributed to the whole system. AA can switch to offline mode until the EHR consumer request for the decryption key.

2) EHR provider formulate the appropriate policy by using an *LSSS* access structure $(\mathbb{M}, \rho)$. EHR will be encrypted under this access policy. Then the EHR provider will start the ***ACF-Create*** to hide the access matrix $(\mathbb{M}, \rho)$. Afterwards, the encrypted EHR with hidden policy will be outsourced to the cloud.

3) When the EHR consumer wants to get access to an EHR, he/she first downloads the EHR according to personal interests. Then he/she will query to AA for the secret key *SK*.

4) AA will be activated from the offline mode and generates the *SK* according to the attributes owned by the EHR consumer. Then the secret key will be transfered to the EHR consumer through a preset secure channel. Then AA will switch back to offline mode again.

5) After receiving the *SK*, EHR consumer will initial the ***ACF-Check*** to match his/her attribute with those in the hidden policy. The decryption of EHR ciphertext succeed only if the match up procedure correctly match and recover the attributes from the hidden access policy. Otherwise, the system will not decrypt the EHR ciphertext, neither can the EHR consumer know any information from the hidden policy.

A. SYSTEM SETUP

In this algorithm, AA initials the ***Setup*** algorithm. Denote $\mathbb{G}$ and $\mathbb{G}_T$ to be two multiplicative cyclic groups of prime order p. $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ is a bilinear map. AA chooses a generator $g \in \mathbb{G}$ and N random group elements $h_1, \ldots, h_N \in \mathbb{G}$ associated with the N attributes in the system. Randomly choose $\mu, u \in \mathbb{Z}_p^*$. Let L_{att} and L_{rnum} represent the maximum bit length of the attributes as well as the maximum bit length of row numbers of the LSSS matrix respectively. Let H_f be the collision-resistent hash functions of generating fingerprint of an element. Let H_e be the collision-resistent hash function which maps an element to an entry in the ACF buckets.

The public parameter is

$PK = \langle g, e(g,g)^{\mu}, g^{u}, h_1, \ldots, h_N, L_{att}, L_{rnum}, H_f, H_e \rangle$.

The master secret key is set to be $MSK = g^{\mu}$.

B. KEY GENERATION

To get access to the encrypted data in the cloud, data consumers have to apply for the secret key from AA. AA will check whether the data consumer is legal. If so, AA allocates the attributes U according to the data consumer's characteristic. AA also generates the corresponding secret key for the data consumer based on the assigned attributes by using the key generation algorithm. It takes the input as PK, MSK and U, randomly chooses $t \in \mathbb{Z}_p^*$, then computes

$$E = g^{\mu} g^{ut}, \quad I = g^{t}, \ \{E_x = h_x^t\}_x \in U.$$

Then the secret key is set to be:

$$SK = \langle E, I, \{E_x\}_{x \in U}, U \rangle.$$

C. DATA ENCRYPTION AND ACF-CREATION

In this algorithm, the data owner first runs the ***Enc*** sub-algorithm. It takes as input the PK, the plaintext M and the access metrix $(\mathbb{M}, \rho)$. $\mathbb{M}$ is an $l \times n$ matrix by using function ρ to map attributes to rows of $\mathbb{M}$. The sub-algorithm starts with randomly selects a vector $v = (s, y_2, \ldots, y_n) \in \mathbb{Z}_p^*$, in which $y_2, \ldots, y_n$ are used to share the encryption secret s. For $i = 1$ to l, it calculates $\lambda_i = M_i \cdot v$, where M_i is the vector related to the ith row of $\mathbb{M}$. Then it outputs the ciphertext:

$$CT = \langle C = Me(g,g)^{\mu s}, C' = g^{s}, \{C_i = g^{u\lambda_i} h_{\rho(i)}^{-s}\}_{i=1,\ldots,l} \rangle.$$

In the original ABE schemes, after encryption, the ciphertext CT will be outsourced to the cloud server associated with the access policy $(\mathbb{M}, \rho)$. Although the message is encrypted, the policy is in a plaintext form whereas. Which will cause the privacy leakage of both the data owner and the data consumer. The leakage is caused by the mapping function ρ by the observation by Yang *et al*. [19]. In order to replace the mapping function ρ, they create a novel element location and recovery algorithm called Attribute Bloom Filter (ABF) by using the reformative Bloom Filter named garbled bloom filter present in [23]. However, due to the property of bloom filter, the algorithm has some problems in dealing with the position occupation conflict.

Here we construct a combined algorithm called Attribute Cuckoo Filter (ACF), which consists of two sub-algorithms ***ACF-Create*** and ***ACF-Check***. We introduce the cuckoo filter (CF) as a building block of our ACF algorithm. CF defines the basic unit of storage as an entry. Fingerprint of each element

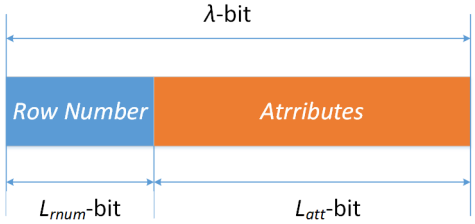


FIGURE 5. λ-bit length element of ACF with L_{rnum}-bit row number string and L_{att}-bit attribute string.

will be stored in each entry. The CF hash table consists of an array of buckets. A bucket can have multiple entries as demonstrated in Fig.5. For each element x to be inserted to the hash table, CF first calculates the fingerprint of x, then use (1) to determine two candidate buckets. If either bucket has an empty entry, the fingerprint will be inserted successfully. Otherwise, the algorithm will randomly chooses a bucket, then an random entry, and replaces the existing fingerprint by the new one. The kicked out fingerprint will then enter a kicking out circulation, until the empty entry is found. This will result in the unrecoverable previous inserted fingerprint. So, we made some modification based on the original CF and formulate some regulations in order to prevent the displace of the existing fingerprint. The detailed analysis can be found in VI-B.

To precisely locate attributes to the relevant row number in the *LSSS* matrix. We use an array of λ-bit as shown in Figure 4. the element is concatenated of two fixed length parts: The L_{rnum} represents the row number, and L_{att} denotes the attribute, where $\lambda = L_{rnum} + L_{att}$.

When the encryption finishes, the procedure moves to the sub-algorithm ***ACF-Create***.

The sub-algorithm takes as input the access policy $(\mathbb{M}, \rho)$. It binds the attributes with the relevant row number in the access matrix $\mathbb{M}$ and generate a set of elements $U_x = \{i \| att_x\}_{i\in[1,l]}$, in which the i-th row of the access matrix maps to the attribute $att_x = \rho(i)$. Then the row number i and the att_x will be expanded to the maximum bit length by zero bit filling on the left side of the string. Afterwards, ACF-Create algorithm is able to create the ACF by taking the U_x as a input. When we have to insert a new element x in the set U_x to the ACF, the algorithm first shares the element x with (k, k) secret sharing scheme by randomly generating $k-1$ λ-bit length strings $c_{1,x}, \ldots, c_{k-1,x}$ and set:

$$c_{k,x} = c_{1,x} \oplus \ldots \oplus c_{k-1,x} \oplus x.$$

Then, it hashes the attribute att_x associated with the element x by using Eq.(1) and gets $H_f(att_x)$ as the fingerprint of x as well as the $H_e(att_x)$ denotes one of the candidate buckets. It then stores the i-th element share c_i to the ACF as

$c_{i,x} \rightarrow H_f(att_x)$ position in ACF.

When we continue to add new elements y to the ACF, some entries may already stored the fingerprint $H_f(x')$ of element x'. If neither of the two candidate bucket have any empty entries, the collision occurs. To overcome this problem, we replicate a new ACF hash table exactly the same as the first ACF, and place y into the new ACF firstly. Secondly, we restrict the space efficiency to be no more than 50%. Algorithm 1 demonstrate the ACF-Create procedure. Finally, the data owner outsources the ciphertext along with the $\mathbb{M}$ and *ACF* to the cloud in the form of $(CT, ACF, \mathbb{M})$.

Algorithm 1 ACF-Create

Input: An *LSSS* access matrix $(\mathbb{M}, \rho)$, λ
Input: ACF parameters: bucket size b, capacity l,
Input: H_e, H_f denotes the hash functions
Output: ACF_j

Create an element set U_x from the access matrix $(\mathbb{M}, \rho)$;
$c = 0.5 * b * l$ ◁ *space occupation boundary is 50%*
$f = H_f(x)$;
$i_1 = H_e(x)$;
$i_2 = i_1 \oplus H_e(f)$;
for $j = 0; j < 4; j$++ **do**
  if $\frac{count}{b*l} < c$ **then**
    ◁ *count is the number of existed elements*
    if $ACF_j.bucket[i_1]$ or $ACF_j.bucket[i_2]$ has an empty entry **then**
      add $\langle f, f \oplus x\rangle$ to that bucket;
      count++;
    else
      continue;
    end if
  end if
end for
return Failure;

D. ACF-CHECK AND DECRYPTION

When data consumer wants to get access to the encrypted data in the cloud, the access control mechanism initials to check whether the attributes of the data consumer could satisfy the access policy or not. In traditional ABE schemes, it is easy to figure out the result for the access policy $\mathbb{M}, \rho$ can be seen. In our proposed scheme, however, the mapping function ρ is hidden, so ***ACF-Check*** will be initialed to checkout which attributes of the data consumer is in the access matrix.

– ***ACF-Check*** takes as input the *PK*, the attribute set U of the data consumer and *ACF* as well. For each attribue *att* in the set U, The algorithm computes first computes $H_e(att)$ to locate the first bucket, then it calculates the fingerprint of *att* by using H_f and gets $H_f(att)$, by which the position indices of the candidate bucket could be found. If either bucket have the fingerprint, then the string in the entry will be fetched in the ACF as follows:

$H_f(att)$ position in ACF $\rightarrow c_{i,x}$.

Then the element constructs as follows:

$$\begin{aligned} x &= c_{1,x} \oplus \ldots \oplus c_{k-1,x} \oplus c_{i,x}, \\ &= c_{1,x} \oplus \ldots \oplus c_{k-1,x} \oplus c_{1,x} \oplus \ldots \oplus c_{k-1,x} \oplus x \end{aligned}$$

Here the element x is formed as $x = \{i \| att_x\}$. Then the algorithm will automatically remove the zero bits to the left of the string L_{att} to get the attribute att_x, then the same operation to obtain the row number i from L_{rnum}. Otherwise, the attribute att does not exit in the access policy if att is not the same as att_x, Finally, the new attribute mapping function ρ' will be reconstructed as:

$$\rho' = \{rnum, att\}_{att \in U}.$$

Then the row number in the matrix $\mathbb{M}$ will be determined. The ACF check algorithm is presented in Algorithm 2.

Algorithm 2 ACF-Check

Input: Attribute Cuckoo Filters ACF_j, a set of attributes U
Input: Maximum row number string length L_{rnum}
Input: Maximum attribute string length L_{att}
Input: H_e, H_f denotes the hash functions
Output: $\rho' = \{\text{rnum}, \text{att}\}_{\text{att} \in U}$

for each att $\in U$ **do**
  ReCov $= \{0\}^{\lambda}$ ◁ *initial the recovery string*
  $f' = H_f(att)$;
  $i_1 = H_e(att)$;
  $i_2 = i_1 \oplus H_e(f')$;
  for $j = 0; j < 4; j$++ **do**
    if $ACF_j.bucket[i_1]$ or $ACF_j.bucket[i_2]$ has f' **then**
      ReCov $= f' \oplus f \oplus x$;
      $att_{Str} = Get_{L_{att}}$(ReCov)
      $att' = RMZ(att_{Str})$ ◁ *remove the filling zeros*
      if $att' == att$ **then**
        $rnum_{Str} = Get_{L_{rnum}}$(ReCov)
        $rnum = RMZ(rnum_{Str})$
        ◁ *remove the filling zeros*
        Add $(rnum, att)$ into ρ'
      end if
    else
      continue;
    end if
  end for
end for

When the access policy $(\mathbb{M}, \rho')$ is obtained. The final decryption algorithm can proceed just the same as in the original CP-ABE scheme.

– ***Dec*** sub-algorithm takes as input the SK, CT and the reconstructed mapping function ρ' as well as the access matrix $\mathbb{M}$. If the attributes could satisfy the access policy, coefficients $\{\tau_i \mid i \in I\}$ can be found so that $\sum_{i \in I} \tau_i \lambda_i = s$, where $I = \{\tau_i : \rho'(i) \in U\} \subset \{1, 2, \ldots, l\}$. Then the data consumer computes:

$$e(C', E) / \prod_{i \in I} (e(C_i, I)) e(C', E_{\rho'(i)})^{\tau_i} = e(g, g)^{\mu s}.$$

So, the plaintext data can be divide from C by $M = C / e(g, g)^{\mu s}$. Otherwise, a $\perp$ outputs to indicate the decryption fails.

VI. SECURITY AND PERFORMANCE ANALYSIS

A. SECURITY ANALYSIS

Theorem 1: There is no poly-time adversary can selectively break the proposed scheme under the decisional q-BDHE assumption with an $l^* \times w^*(w^* \leq q)$ challenge matrix.

• *Proof:* Our proposed scheme is built on the bases of the CP-ABE scheme proposed in [13], which has been proved to be selectively secure under the decisional q-BDHE assumption. If there exists a adversary $\mathcal{A}$ who can break our scheme with a non-negligible advantage $\epsilon = Adv_{\mathcal{A}}$, a simulater $\mathcal{B}$ can be built to solve the q-BDHE problem with the same non-negligible advantages.

To prove the security of our scheme, we imitate the game between the adversary and the simulator. If there exists a adversary $\mathcal{A}$ who has a non-negligible advantage $\epsilon = Adv_{\mathcal{A}}$ in winning the selective security game, we can create a simulator $\mathcal{B}'$ which can break the decisional q-BDHE assumptions with the same non-negligible advantages. The creation of $\mathcal{B}$ is quite alike the creation of $\mathcal{B}$ in [13]. The operation in ***Init*** phase is the same, In the ***Setup*** phase, we choose the CF hash to be the random oracle besides the existing steps. The secret key query phase is the same, but the ***challenge*** phase have some differences. Our encryption algorithm is composed of two sub-algorithms. The simulator $\mathcal{B}'$ make queries about secret keys from the ***ACF-Create*** oracle. Then in the sub-algorithm of $\mathcal{B}'$.***Enc*** $= \mathcal{B}$.***Encrypt***. For that the challenge matrix has been selected by the adversary before the ***Init*** phase, so the construction of ACF will not increase the advantages in winning the game. So we can conclude that $\mathcal{B}'$ has a negligible advantage in breaking the q-BDHE assumption. ■

Theorem 2: Our proposed scheme is policy preserving against the poly-time adversary in the security parameter λ.

• *Proof:* In our proposed scheme, only the data consumers who have the attributes could recover the attribute string from the attribute space N. A polynomial time adversary can only guess the attribute strings in a brute force way. Therefore, they cannot sniffer any sensitive information from the modified access policy formed as $(\mathcal{M}, ACF)$. Data consumers can only check the attributes of their own to see whether they are in the hidden access policy. They cannot check all the attributes in the attribute universe, unless by collude with each other.

Since the ACF is constructed with the cuckoo filter with λ-bit secure parameter, the false-positive rate can be less than $2b/2^{\lambda}$. ■

B. ANALYSIS & SOLUTION OF THE ACF INSERT COLLISION

In most cases of our proposed schemes, if there is an empty entry in the candidate buckets, ACF will not kick out the existing fingerprints. However, if collision happens, we have to find another entry for the subsequent fingerprints. In original Cuckoo Filter, there is only one hash table, so the kicking out circulation stops either at finds an empty entry or at the maximum times kicking (e.g. 100 times) which would

implicit the hash table is almost full or have an high space occupation. So we made some improvements.

First, we set the threshold value of the space occupation to be no more than 50% (Although original CF can support an efficient space occupation at about 95%). When the space occupation reaches the threshold value, if we want to insert a new element, the ***ACF-Create*** sub-algorithm creates a new hash table same as the first one, in which all the entries are empty. Then the fingerprint of the new element will be stored in the new ACF table. Considering the situation when there is a collision occurred when the space occupation have not reach 50%, the newer element will also be inserted into the new ACF table. Hereafter, if collision happens in second cuckoo hash table, then the third hash table will be created and so on.

We give the probability of collision of inserting item x to the ACF as follows:

$$P = \begin{cases} 0 & x \leq 2b \\ \dfrac{1}{s(s-1)} & 2b < x \leq 3b \\ \dfrac{C_3^2}{s(s-1)} & 3b < x \leq 4b \\ \vdots & \\ \dfrac{C_{2^{l-2}}^2}{s(s-1)} & 2^{l-2}b < x \leq 2^{l-1}b \end{cases}$$

Here, b is the number of entries per bucket(usually, b is set to be 2, 4 or 8). m is the number of the elements to be inserted. σ denotes the space occupation(50% in our ACF). $l = \lceil log(\frac{m}{\sigma}/b) \rceil$, and we have the buckets size of $s = 2^l$.

Here is an instance. Suppose there are 16 elements to be inserted. Let $b = 4$, then we can calculate $l == \lceil log(\frac{m}{\sigma}/b) \rceil = 3$. Then we begin to insert the element. The first 8 elements will cause any collisions. The insertion of 9 - 12 elements will cause a collision at the probability of 1.786%, 13 - 16 elements will raise up to 5.357%. After that, if we still want to insert new items, ACF will create a new hash table to do so, for the space occupation of the first ACF hash table has reached the threshold percentage of 50.

C. THEORETICAL PERFORMANCE ANALYSIS

Our proposed policy preserving EHR system is constructed on the basis of Waters CP-ABE [13]. We modified the original *encryption* and *decryption* phase in order to achieve policy hiding. In this section, we will analyze the cost of introducing the two sub-algorithms theoretically.

Let Γ denotes the number of attributes involved in encryption phase, which also represents how many rows have taken part in the *LSSS* Matrix $(\mathbb{M}, \rho)$. Δ refers to the number of rows in the *LSSS* that will be used for decryption. Ω is the hash function used in ACF. i, j represent the bilinear pairing and exponential operation respectively. Although in ACF, there have two hash functions, one for normal hash of an element, the other calculates the fingerprint of the same element. We can reckon they have the same efficiency in a rough. Let ϕ denotes the *XOR* calculation. In our encryption phase, EP have to run an extra sub-algorithm called *ACF-Create* besides data encryption. This sub-algorithm only have to calculate the hash value and the relevant fingerprint of an attribute, then *XOR* the original value with the fingerprint. Both the hash and the *XOR* calculation are extraordinary efficient compared with the bilinear pairing operation. So it will not increase much overhead.

TABLE 3. Computation complexity comparison.

	Yang [20]	Ours
Policy Hidding	$k\Omega + \phi$	n*$(2\Omega + \phi)$
Attribute Recovery	$k\Omega + \phi$	n*$(2\Omega + \mathcal{O}_{(1)}) + \phi$
Encryption	$j * (2 + 2\Gamma) + k\Omega + \phi$	$j * (2 + 2\Gamma) + 2\Omega + \phi$
Decryption	$i*(2\Delta+1)+j\Delta+k\Omega+\phi$	$i * (2\Delta + 1) + j\Delta + n * (2\Omega) + \phi$

It is worth noting that we solve the inserting conflict by setting the space occupation of no more than 50%. Besides, if collision happens ahead of the space occupation threshold value, then a new ACF table will be created. All the conflict elements will be placed to the next table and so on. We denote the number of ACF table as n. However, due to the low space occupation, the conflict rarely happens in the real evaluation.

In the decryption phase, EC will first download the encrypted EHR according to their needs. Then, before the EHR decryption, EC have to initial the *ACF-Check*, the algorithm will use the fingerprint to locate the candidate buckets. Then for each bucket, the element query overhead will be $\mathcal{O}_{(n)}$, where n is the bucket size. After locating the entry of exact attribute, an *XOR* operation will be used to recover the value. Similar to the encryption phase, the overhead is tiny in contract with the decryption of EHR. The ACF computation complexity comparison with ABF proposed by Yang *et al.* [19] can be found in Table 3.

D. PERFORMANCE ANALYSIS

We proposed two sub-algorithms on the basis of Cuckoo Filter. The ***ACF-Create*** algorithm is added in the phase of encryption, while the ***ACF-Check*** algorithm runs in the phase of decryption. In order to figure out how much computation cost have been introduced by ACF algorithm, we deploy the experiment environment on the Ubuntu Linux Desktop 64-bit system with an Intel Core i7 CPU at 3.4GHz and 8.00 GB RAM. The code utilizes the *charm* library version 0.50,[2] and an asymmetric elliptic curve α-curve, where the base field size is 512-bit and the embedding degree is 2, so that the security parameter is 1024-bit. To implement our ACF, we employ the *MurmurHash3*.[3] All the experimental result are the mean of 100 trails.

Specifically, we perform the comparison experiment with the original CP-ABE [13] without policy preserving as well as the CP-ABE with ABF to hide the policy matrix by Yang *et al*. [19]. Fig.6 and Fig.7 demonstrates the cost of

[2]Available: https://github.com/JHUISI/charm
[3]Available: https://github.com/hajimes/mmh3

TABLE 4. The comparison of encryption/decryption time with ACF & ABF.

Number of Attributes	n = 8	n = 10	n = 12	n = 14	n = 16	n = 18	n = 20
Encryption Procedure (*Second*)							
EHR Encryption	35.1×10^{-3}	42.4×10^{-3}	51.2×10^{-3}	58.7×10^{-3}	66.7×10^{-3}	75.8×10^{-3}	82.5×10^{-3}
ABF-Building (8 Hash)	25.9×10^{-5}	32.1×10^{-5}	41.8×10^{-5}	45.4×10^{-5}	54.9×10^{-5}	58.6×10^{-5}	68.1×10^{-5}
ABF-Building (16 Hash)	55.8×10^{-5}	64.0×10^{-5}	83.6×10^{-5}	96.3×10^{-5}	111.6×10^{-5}	126.6×10^{-5}	135.5×10^{-5}
ACF-Create	2.9×10^{-5}	3.5×10^{-5}	4.4×10^{-5}	5.0×10^{-5}	5.6×10^{-5}	6.2×10^{-5}	7.3×10^{-5}
Decryption Procedure(*Second*)							
EHR Decryption	4.87×10^{-3}	4.92×10^{-3}	5.3×10^{-3}	5.5×10^{-3}	6.3×10^{-3}	6.7×10^{-3}	7.0×10^{-3}
ABF-Query (8 Hash)	25.8×10^{-5}	32.4×10^{-5}	41.7×10^{-5}	51.9×10^{-5}	56.9×10^{-5}	59.5×10^{-5}	73.8×10^{-5}
ABF-Query (16 Hash)	52.1×10^{-5}	69.3×10^{-5}	79.4×10^{-5}	94.5×10^{-5}	112.3×10^{-5}	121.6×10^{-5}	140.3×10^{-5}
ACF-Check	2.5×10^{-5}	3.1×10^{-5}	3.7×10^{-5}	4.3×10^{-5}	4.9×10^{-5}	5.5×10^{-5}	6.2×10^{-5}

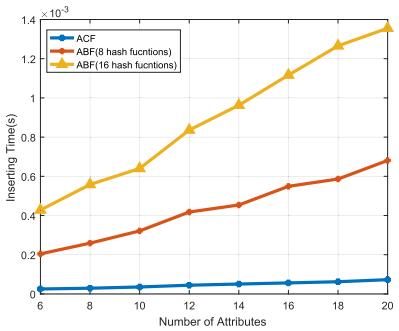


FIGURE 6. Attribute inserting time between ACF & ABF.

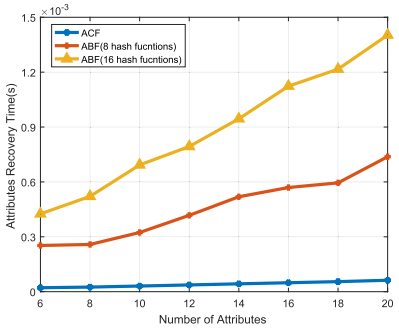


FIGURE 7. Attribute recovery time between ACF & ABF.

inserting items into the ABF and ACF as well as recovering the attributes from the Filter respectively. Obviously, our ACF is more efficient than ABF scheme under the same condition.

We also evaluate the encryption and decryption overhead. The encryption phase in our proposed scheme is composed of ACF-creation and data encryption. The decryption phase in our proposed scheme is composed of ACF-check and data decryption. The extra overhead comparison is listed in Table 4. Apparently, our scheme can preserve the access policy privacy without introducing much computation overhead in both encryption and decryption procedure. Compared with the ABF algorithm, we also have an obvious efficiency improvement.

VII. CONCLUSION

We proposed an efficient policy preserving CP-ABE scheme in the cloud sharing EHR environment. Different from the policy partially-hidden schemes which hides only the attribute value, our scheme could hide the entire access policy. To deal with the problem of attribute matching and recovering from the hidden policy, we designed an ACF matrix to precisely locate the row number and the relevant attribute. We proved the security of our scheme, and the evaluation result shows that our scheme can achieve policy preserving as well as the attribute recovery without introducing much overhead.

ACKNOWLEDGMENT

The authors appreciate the help from the unsigned researchers and students for their constructive advises and suggestions.

REFERENCES

[1] American EHR. *2017 Top 10 Best EHR Systems*. Accessed: Aug. 1, 2016. [Online]. Available: http://www.americanehr.com/ratings/ehr_ratings/ehr-top10ratings.aspx

[2] Athenahealth. *2012 Physician Sentiment Index*. Accessed: Jun. 14, 2012. [Online]. Available: https://www.athenahealth.com/PSI/meaningful-use-incentives.php

[3] Gadgets. *Singapore Health Database Hit by Cyber-Attack, Leaking Details of 1.5 Million Including PM*. Accessed: Jul. 20, 2018. [Online]. Available: https://gadgets.ndtv.com/internet/news/singapore-health-database-hit-by-cyber-attack-leaking-details-of-1-5-million-including-pm-1886937

[4] X. Liu, R. Deng, K. K. R. Choo, and Y. Yang, "Privacy-preserving outsourced clinical decision support system in the cloud," *IEEE Trans. Services Comput.*, to be published, doi: 10.1109/TSC.2017.2773604.

[5] A. Sahai and B. Waters, "Fuzzy identity-based encryption," in *Proc. Annu. Int. Conf. Theory Appl. Cryptograph. Techn.* Berlin, Germany: Springer, 2005, pp. 457–473.

[6] V. Goyal, O. Pandey, A. Sahai, and B. Waters, "Attribute-based encryption for fine-grained access control of encrypted data," in *Proc. 13th ACM Conf. Comput. Commun. Secur.*, 2006, pp. 89–98.

[7] J. Bethencourt, A. Sahai, and B. Waters, "Ciphertext-policy attribute-based encryption," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2007, pp. 321–334.

[8] A. Kapadia, P. P. Tsang, and S. W. Smith, "Attribute-based publishing with hidden credentials and hidden policies," in *Proc. NDSS*, vol. 7, 2007, pp. 179–192.

[9] L. Cheung and C. Newport, "Provably secure ciphertext policy ABE," in *Proc. 14th ACM Conf. Comput. Commun. Secur.*, 2007, pp. 456–465.

[10] T. Nishide, K. Yoneyama, and K. Ohta, "Attribute-based encryption with partially hidden encryptor-specified access structures," in *Proc. Int. Conf. Appl. Cryptogr. Netw. Secur.* Berlin, Germany: Springer, 2008.

[11] J. Li, K. Ren, and B. Zhu, "Privacy-aware attribute-based encryption with user accountability," in *Proc. Int. Conf. Inf. Secur.* Berlin, Germany: Springer, 2009, pp. 347–362.

[12] J. Lai, R. H. Deng, and Y. Li, "Expressive CP-ABE with partially hidden access structures," in *Proc. 7th ACM Symp. Inf., Comput. Commun. Secur.*, 2012, pp. 18–19.

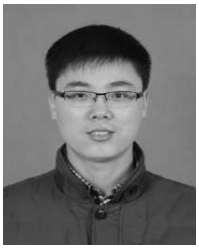
[13] B. Waters, "Ciphertext-policy attribute-based encryption: An expressive, efficient, and provably secure realization," in *Proc. Int. Workshop Public Key Cryptogr.* Berlin, Germany: Springer, 2011, pp. 53–70.
[14] Z. Ying, H. Li, J. Ma, J. Zhang, and J. Cui, "Adaptively secure ciphertext-policy attribute-based encryption with dynamic policy updating," *Sci. China Inf. Sci.*, vol. 59, p. 042701, Apr. 2016.
[15] J. Li, Y. Zhang, X. Chen, and Y. Xiang, "Secure attribute-based data sharing for resource-limited users in cloud computing," *Comput. Secur.*, vol. 72, pp. 1–12, Jan. 2018.
[16] S. Yu, C. Wang, K. Ren, and W. Lou, "Achieving secure, scalable, and fine-grained data access control in cloud computing," in *Proc. IEEE INFOCOM*, Mar. 2010, pp. 1–9.
[17] J. Li, X. Lin, Y. Zhang, and J. Han, "KSF-OABE: Outsourced attribute-based encryption with keyword search function for cloud storage," *IEEE Trans. Services Comput.*, vol. 10, no. 5, pp. 715–725, Sep./Oct. 2017.
[18] M. Qiu, K. Gai, B. Thuraisingham, L. Tao, and H. Zhao, "Proactive user-centric secure data scheme using attribute-based semantic access controls for mobile clouds in financial industry," *Future Gener. Comput. Syst.*, vol. 80, pp. 421–429, Mar. 2018.
[19] K. Yang, Q. Han, H. Li, K. Zheng, Z. Su, and X. Shen, "An efficient and fine-grained big data access control scheme with privacy-preserving policy," *IEEE Internet Things J.*, vol. 4, no. 2, pp. 563–571, Apr. 2017.
[20] Q. Han, Y. Zhang, and H. Li, "Efficient and robust attribute-based encryption supporting access policy hiding in Internet of Things," *Future Gener. Comput. Syst.*, vol. 83, pp. 269–277, Jun. 2018.
[21] A. Beimel, "Secure schemes for secret sharing and key distribution," Ph.D. dissertation, Fac. Comput. Sci., Technion-Israel Inst. Technol., Haifa, Israe, 1996.
[22] B. Fan, D. G. Andersen, and M. Kaminsky, "Cuckoo filter: Practically better than bloom," in *Proc. 10th ACM Int. Conf. Emerg. Netw. Exp. Technol.*, 2014, pp. 75–88.
[23] C. Dong, L. Chen, and Z. Wen, "When private set intersection meets big data: an efficient and scalable protocol," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2013, pp. 789–800.

ZUOBIN YING (M'17) received the Ph.D. degree in cryptography from Xidian University, Xi'an, China, in 2016. He is currently a Lecturer with the School of Computer Science and Technology, Anhui University, China. His research interests include cloud security and applied cryptography.

LU WEI is currently a Research Student with the School of Computer Science and Technology, Anhui University, Hefei, China. His research focuses on vehicle ad hoc networks.

QI LI received the Ph.D. degree in computer system architecture from Xidian University, Xi'an, China, in 2014. He is currently a Lecturer with the School of Computer Science, Nanjing University of Posts and Telecommunications, China. His research interests include information security and applied cryptography.

XIMENG LIU (S'13–M'16) received the B.Sc. degree in electronic engineering and the Ph.D. degree in cryptography from Xidian University, Xi'an, China, in 2010 and 2015, respectively. He is currently a Full Professor with the College of Mathematics and Computer Science, Fuzhou University, China. He is also a Research Fellow with the School of Information System, Singapore Management University, Singapore. He has published over 100 research articles, including IEEE TIFS, IEEE TDSC, IEEE TC, IEEE TII, IEEE TSC, and IEEE TCC. His research interests include cloud security, applied cryptography, and big data security.



JIE CUI was born in Zhoukou, Henan, China, in 1980. He received the Ph.D. degree from the University of Science and Technology of China in 2012. He is currently an Associate Professor with the School of Computer Science and Technology, Anhui University. He has published over 50 papers. His current research interests include applied cryptography, IoT security, vehicular ad hoc networks, and software-defined networking.