Hindawi
Mobile Information Systems
Volume 2017, Article ID 7514867, 15 pages
https://doi.org/10.1155/2017/7514867

Research Article

Flexible and Lightweight Access Control for Online Healthcare Social Networks in the Context of the Internet of Things

Zhen Qin, Jianfei Sun, Dajiang Chen, and Hu Xiong

School of Information and Software Engineering, University of Electronic Science and Technology of China, Chengdu 610054, China

Correspondence should be addressed to Hu Xiong; xionghu.uestc@gmail.com

Received 27 November 2016; Revised 6 March 2017; Accepted 12 April 2017; Published 28 May 2017

Academic Editor: Tao Han

Online healthcare social networks (OHSNs) play an essential role in sharing information among medical experts and patients who are equipped with similar experiences. To access other patients' data or experts' diagnosis anywhere and anytime, it is necessary to integrate the OHSN into the Internet as part of the Internet of Things (IoT). Therefore, it is crucial to design an efficient and versatile access control scheme that can grant and revoke a user to access the OHSN. In this paper, we propose novel attribute-based encryption (ABE) features with user revocation and verifiable decryption outsourcing to control the access privilege of the users. The security of the proposed ABE scheme is given in the well-studied random oracle model. With the proposed ABE scheme, the malicious users can be excluded from the system and the user can offload most of the overhead in the decryption to an untrusted cloud server in a verifiable manner. An access control scheme for the OHSN has been given in the context of the IoT based on the proposed ABE scheme. The simulation demonstrates that our access control mechanism is practical.

1. Introduction

Fueled with wireless medical sensors implanted or worn on human body, personal health information (PHI) can be extracted anytime and from any location handily. To share the PHI with the medical experts or other patients from the same community, the online healthcare social networks (OHSNs) [1–3] can be formed and integrated into the Internet of Things (IoT) [4–6] by exchanging the PHI via the portable devices such as the smart phones (Figure 1). With the support of OHSN, users can easily identify other users with certain symptoms in the community and share diagnosis information with each other. Furthermore, accurate and positive treatment information from medical experts can also be disseminated in OHSN to improve the online healthcare environment.

In OHSNs, PHI data are extremely sensitive since these data are closely related to the patients' health status. Naturally, one main requirement of PHI data sharing is to ensure that the data owners could fully control the access to their PHI data and hinder unauthorized users from obtaining the PHI data. One simple approach to achieve access control on the PHI data is to encrypt the shared data with the conventional asymmetric encryption by considering the bulky key management overhead of the symmetric encryption. In asymmetric cryptography (also known as public key cryptography), each user is equipped with a public/private key pair where the public key is distributed in the system and the private key is kept a secret by the user. To share PHI data with close friend or medical expert securely, data owner needs to encrypt the shared data under the public key of the receiver. In this way, only the receiver can read this data with his/her private key. However, this approach is not scalable in the system with a huge number of users. Let us consider the following scenario. Assume the user Alice intends to share her PHI data with her attending physician Bob, her colleague Carol, and her friend David with the conventional public key cryptography. To ensure her data can be accessed by these three users, Alice has to perform the encryption over the shared data three times under the public key of Bob, Carol, and David, respectively. Thus, it is not a trivial task to construct an access control mechanism for the OHSNs in the context of the IoT due to the large-scale nature of OHSNs.

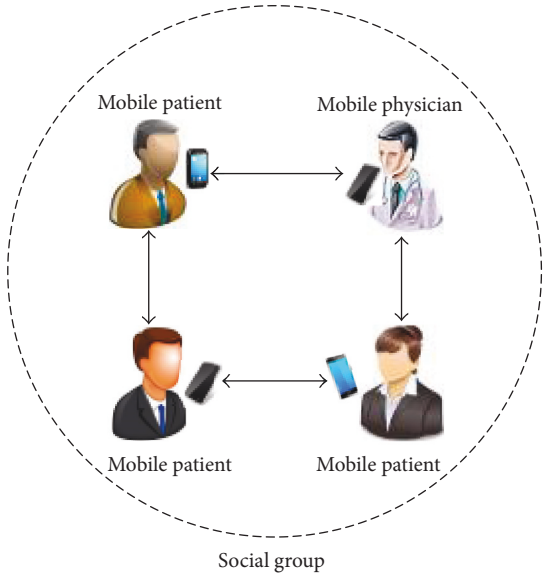

Figure 1: PHI data sharing system.

Attribute-based encryption (ABE) [7, 8] seems to be a possible promising solution to provide flexible and versatile access control over the encrypted sharing data due to its one-to-many encryption pattern. In an (ciphertext policy) ABE system, user's private key and ciphertext are, respectively, labeled with a set of attributes and an access policy. Without enumerating the public key of all of the intended receivers, the ciphertext can be read by a group of users as long as the set of attributes associated with the users satisfy the access structure embedded in the ciphertext. That is to say, the one-to-many encryption can be achieved by ABE directly, which makes the ABE primitive particularly appropriate to the large-scale OHSNs. However, two major barriers impede ABE schemes from direct deployment in OHSNs. First, tremendous amounts of wireless and portable sensors/devices interconnected in OHSNs can be easily infiltrated or breached by hackers; therefore, the function with revocation should be considered in ABE schemes to deal with this situation. Second, the ciphertext size and the computational cost of ABE usually grow with the complexity of the access policy because of its expressiveness. By considering the limited battery and computational capabilities of the wireless and portable sensors/devices [9, 10], the existing ABE scheme cannot be directly used to secure the OHSNs.

The motivation of the paper is to construct a practical access control mechanism suitable for the large-scale OHSNs in the context of the IoT. In OHSNs, the users equipped with portable devices may be compromised and thus excluded from the system. By considering the limited computation capabilities of the mobile users, the mechanism should be designed to be lightweight. To tackle the above-mentioned issues, we design our ABE scheme featured with user revocation and verifiable decryption outsourcing and apply the proposed ABE scheme to OHSNs. The contributions of this paper include the following two points:

(1) We design a novel ABE scheme with user revocation by incorporating the ciphertext update and key update function such that any legitimate users in social group can access the PHI data from other data owners in the community and revoked users cannot read the encrypted data again even if they intend to collude their attributes with other legitimate social group users. Moreover, our scheme offloads most of the computation task on the data sharers' side to the untrusted cloud server provider. Concretely, the data sharer sends the transformation key to the cloud server, which in turn sends the partially decrypted ciphertexts to the data sharers and thus contributes to offloading the original decryption task. Moreover, the outsourced result can be easily validated by users whether it is true or not.

(2) Based on our ABE scheme, we present a versatile personal health information (PHI) data sharing architecture to achieve flexible and fine-grained access control. Security proof in the random oracle model and simulation result demonstrate that our scheme is secure and practical.

Organization. The rest of this paper is organized as follows. Related works are illustrated in Section 2. Some preliminaries are presented in Section 3. Section 4 describes the formal definition of proposed scheme, our system model, and the security model. In Section 5, the concrete construction is given. Sections 6 and 7 show the security analysis and the performance analysis, respectively. Finally, our conclusion is stated in Section 8.

2. Related Work

2.1. Attribute-Based Encryption. The notion of attribute-based encryption, which was suggested by Sahai and Waters as fuzzy identity-based encryption in [11], was extended by Goyal et al. [7]. Up to now, two flavors of ABE, key policy attribute-based encryption (KP-ABE) [7, 12–14] and ciphertext policy attribute-based encryption (CP-ABE) [8, 15, 16], have been proposed referring to the fact that access control policy is embedded to the private key or the ciphertext. In a KP-ABE system, keys are associated with an access policy and the ciphertext with a set of attributes. Contrarily, in a CP-ABE system, ciphertexts are assigned to an access structure and the key to a set of attributes. A user can decipher the ciphertext, only if the set of attributes holds the access structure.

Although ABE enjoys high expressive and versatile access control policies, the ciphertext size and the computational cost grow with the complexity of the access policy in existing ABE schemes. To offload high computational overload, Green et al. [17] introduced the notion of outsourced decryption into ABE systems, which largely eliminates the decryption overhead for users. In [17], user's secret key was blinded with a random number and then was delivered to a cloud server to translate the ABE ciphertext into a simper ciphertext. Unfortunately, the verifiability of the cloud's transformation is not guaranteed for users. By offloading all the access

policy and attribute related operation to the key generation service provider (KGSP) and the decryption cloud server provider (DSP), respectively, a secure OABE scheme, which was introduced by Li et al. [18], supplied checkability with supporting both outsourced key issuing and decryption. Presently, Lai et al. [19] formally defined the verifiability of ABE and presented an ABE scheme with verifiable outsourced decryption. Although [19] achieved the desirable effect of the verifiability, double overhead on both the length of ciphertext and the computational cost is reluctant to be accepted for users. To resolve the double overhead problem, Ma et al. [20] suggested a verifiable and exculpable outsourced OABE scheme, which not only largely leverages the ciphertext size and computational cost, but also brings the strong verifiability and exculpability, which effectively addresses the dispute between a user and an outsource computation service provider.

2.2. User Revocation. The issue of applying ABE to the data outsourcing architecture also faces many challenges with regard to user revocation or attribute revocation. Key revocation mechanisms in KP-ABE and CP-ABE, respectively, first proposed by Bethencourt et al. [8] and Boldyreva et al. [21], was realized by incorporating with encrypting the message to the attribute and its validation time. However, it was confirmed that these schemes [8, 21] had the security degradation problem considering the forward and backward secrecy [22]. In such a data sharing revocable ABE scenario, a revoked user might still be able to retrieve the original data by attribute-collusion even if his/her attributes do not hold access policy. Of course, a new user might be able to access the previous encrypted data before joining the system until the data are reencrypted with the current updated attribute keys. It is not desirable that a user, revoked from a single attribute group, loses all the access rights to the system in many pragmatic scenarios since the other attributes may still be valid. Attrapadung and Imai [23] realized a conjunctive broadcast and attribute-based encryption scheme with revocation ability; in [23], the data owner was able to perform the user direct revocation by maintaining all user membership lists for each attribute group. However, this scheme was unsuitable to be applied to the data sharing system mainly considering that the data owner will no longer take direct control of data after outsourcing data to the cloud storage server. These schemes [24–26] also addressed the user revocation in the ABE-based data sharing system. In [25], in order to revoke users, the trusted authority should generate all secret keys including the proxy key; simultaneously, the server would reencrypt the ciphertext to hamper revoked users from deciphering the ciphertext after receiving the proxy key from the trusted authority. Therefore, the key escrow problem described in [26] may appear. To address this problem, Hur [26] integrated a key issuing protocol with the proxy encryption mechanism. Very recently, Li et al. [24] presented a flexible and fine-grained attribute-based data storage in cloud computing; however, several issues, key-unblinding, unverifiability, and high key-updating times, cannot be well addressed.

2.3. Online Healthcare Social Networks. Several research works that are closely related to the online healthcare social networks (OHSNs) have been introduced [27–30]. Chen et al. [27] present an event-aided packet forwarding (EPF) protocol, which enables patients who suffer from the same diseases to discuss health conditions with other wardmates in OHSNs. Zhou et al. [29] show a secure and privacy-preserving key management scheme for cloud-assisted wireless body area network in OHSNs; in [29], it can tamper both time-based and location-based mobile attacks from the collaboration of the patients having the same diseases in the same social group. Jiang et al. [30] propose an efficient and privacy-preserving personal health information sharing scheme in OHSNs; in [30], it provides the privacy-preserving of data recipients by hidden access policy and the lightweight decryption overhead on decryptors' sides by the server-aided outsourcing technique. However, the above schemes do not consider the dynamicity of social group in OHSNs. Featuring with dynamic group can provide the OHSNs with extendibility, flexibility, and practicability.

Most of the existing ABE schemes applied to online healthcare social networks for data sharing can only protect the data confidentiality/privacy. They do not consider the dynamic characteristics of social group and the decryption overhead on recovering the original message. In our paper, the above gaps are bridged well by suggesting a flexible and lightweight access control for online healthcare social networks in the context of the Internet of Things.

3. Preliminaries

In this section, we briefly describe notion description, bilinear map, hardness assumption, linear secret-sharing scheme, proxy reencryption, and key derivation function.

3.1. Notions. The notions used in this paper are listed in Table 1.

3.2. Bilinear Pairing. Let $\mathcal{G}$ represent an algorithm that takes as input a security parameter λ, it outputs a group tuple $(p, \mathbb{G}, \mathbb{G}_T, e)$, where $\mathbb{G}$, $\mathbb{G}_T$ denote multiplicative cyclic groups with prime order p, and $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ is a computable bilinear map such that the following are defined:

(1) *Bilinearity*: $e(g_1^u, g_2^v) = e(g_1, g_2)^{uv}$ for all $g_1, g_2 \in \mathbb{G}$, and $u, v \in \mathbb{Z}_p^*$.

(2) *Nondegeneracy*: $e(g_1, g_2) \neq 1$, whenever $g_1, g_2 \neq 1_{\mathbb{G}}$.

Definition 1 (DCDH assumption). Given a group tuple $(\mathbb{G}, g, g^s, g^t)$ with its generator g, where $s, t \overset{R}{\leftarrow} \mathbb{Z}_p^*$, the divisible computation Diffie-Hellman (DCDH) problem is to output $g^{s/t}$. Let $\mathcal{G}$ be a bilinear generator; we say that the DCDH holds for $\mathcal{G}$ if, for all probabilistic polynomial-time (PPT) algorithms $\mathcal{A}$, the function $Adv_{\mathcal{A}}^{\mathrm{DCDH}}(\lambda)$ is a negligible function of λ.

3.3. Linear Secret-Sharing Scheme (LSSS). A secret-sharing scheme Π over a set of parties $\mathcal{P}$ is called linear (over $\mathbb{Z}_p^*$)

Table 1: Notions for our proposed system.

Notions	Description
$\mathbb{G}, \mathbb{G}_T$	Two cyclic multiplicative groups with the same prime order p
$e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$	A computable bilinear map
$\mathbb{Z}_p^*$	The integer modulo p
$s \xleftarrow{R} \mathbb{Z}_p^*$	Select a random number s from $\mathbb{Z}_p^*$
$H : \{0,1\}^* \to \mathbb{G}$	A function maps bit strings to a group element in $\mathbb{G}$
$H_1 : \mathbb{G} \to \mathbb{Z}_p^*$	A function maps an element in $\mathbb{G}$ to a random number in $\mathbb{Z}_p^*$
uid	User's identity
gid	Group identity
pk/*msk*	System public key/system master secret key
gpk/*gmk*	Group public key/group master secret key
Rekey	Reencryption key
tk/*rk*	Transformation key/retrieval key
dsk	User's decryption secret key
key/*ssk*	Encapsulated key/session key
TA	The trusted authority
GA	The trusted group administrator
CSS	The cloud storage server
DSP	The decryption cloud service provider
PPT	Probabilistic polynomial time
KDF	Key derivation function

with two properties: (1) A vector v over $\mathbb{Z}_p^*$ is shared by a set of parties. (2) For the secret-sharing scheme Π, a share-generating matrix $\mathcal{M}$ with ℓ rows and n columns can be established. Besides, there also exists a function ρ that maps each row of the matrix to an associated party. In other words, for $i = 1, \ldots, \ell$, the value $\rho(i)$ equals the party associated with row i. When we share a secret $s \in \mathbb{Z}_p$, the column vector $v = (s, r_2, \ldots, r_n)$ is established, where $r_2, \ldots, r_n \in \mathbb{Z}_p^*$. Then $\mathcal{M}v$ is the vector of ℓ shares of the secret s, and the share $(\mathcal{M}v)_i$ belongs to party $\rho(i)$.

A linear secret-sharing scheme Π also enjoys the linear reconstruction property: Suppose that there exists a LSSS for the access structure $\mathbb{A}$. Let $S \in \mathbb{A}$ denote the attribute set S which satisfies the access structure $\mathbb{A}$, and let $I \subset \{1, 2, \ldots, \ell\}$ be defined as $I = \{i : (i) \in S\}$. Then, there exist constants $\{\omega_i \in \mathbb{Z}_p\}_{i \in I}$ such that if $\{\lambda_i\}$ are valid shares of any secret s according to Π, then $s = \sum_{i \in I} \omega_i \lambda_i$.

3.4. Proxy Reencryption. Proxy reencryption [24] allows an entity of honest but curious proxy cloud server, using reencryption key, to transform an encrypted message m under A's public key into an encrypted same message m under B's public key without exposing any valuable information about m.

3.5. Key Derivation Function (KDF). Key derivation function (KDF [31]), as a cryptographic primitive, provides a key-expansion capability that converts the initial keying material containing semi-secret randomness into one or more pseudo-random keys. The pseudorandom keys derived from KDF are indistinguishable from a randomly and uniformly distributed string of the same length. Furthermore, portion of the bits generated by the KDF cannot disclose knowledge on the other bits. The definition and the security of the KDF are illustrated as follows.

Definition 2 (KDF). Given a sampled value s derived from initial keying material and a length value l, a KDF generates a string with length l.

Definition 3 (security of KDF). A KDF is said to be secure against any PPT adversary $\mathcal{A}$, in case the following advantage is negligible:

$$\left|\Pr\left[\mathcal{A}\left(\mathsf{KDF}\left(s, l\right)\right) = 1\right] - \Pr\left[\mathcal{A}\left(R\right) = 1\right]\right|, \quad (1)$$

where R refers to a randomly and uniformly chosen string of the length l bits.

4. Formal Definition, System, and Security Model

In this section, the definition of our proposed scheme, system model, and security model are presented as follows.

4.1. Formal Definition of Our Proposed Scheme. We now describe the framework of our online healthcare social network by exploiting CP-ABE system. Let $\mathcal{S}$ represent a set of attributes, and $L = (\mathcal{M}, \rho)$ stands for an access structure LSSS. Our scheme consists of the following algorithms.

$SystemSetup(\lambda)$. The *SystemSetup* algorithm, implemented by trusted authority, outputs the system master key *msk* and public key *pk* taking as input a security parameter λ.

$GroupSetup(pk)$. The *GroupSetup* algorithm, executed by trusted group administrator, generates the group master key *gmk*, the group public key *gpk*, and a dictionary *dic* with recording the version status.

$CertGen(pk, uid, gmk_{ver})$. The *CertGen* algorithm, also performed by group administrator, takes as input the system public key *pk*, the user's identity *uid,* and the group master secret key gmk_{ver}; it generates a certificate δ_{ver}.

$KeyGen(pk, msk, \mathcal{S}, gpk_{ver}, uid, \delta_{ver})$. The *KeyGen* algorithm, carried out by trusted authority, takes as input the system public key *pk*, the system master secret key *msk*, attribute set $\mathcal{S}$, the public group public key gpk_{ver}, the user's identity *uid*, and the user's authenticated certificate δ_{ver}; it produces the user decryption keys dsk_{ver} and the user tuple up_{ver}.

$Encrypt(pk, gpk_{ver}, M, (\mathcal{M}, \varrho))$. With the system public key *pk*, the group public key gpk_{ver}, a message M, and a LSSS access structure $L = (\mathcal{M}, \varrho)$, the *Encrypt* algorithm, run by PHI data owner, generates the ciphertext CT_{ver} stored in cloud server.

$GroupUpdate(pk, gmk_{ver}, dic_{ver})$. Given the system public key *pk*, the group master key gmk_{ver}, and a dictionary dic_{ver}, the *GroupUpdate* algorithm, executed by group administrator, updates the group master key as gmk_{ver+1} and the group public key as gpk_{ver+1}; besides, it outputs a new dictionary dic_{ver+1} and generates a reencryption key $Rekey_{ver \to ver+1}$ delivered to proxy server.

$UserUpdate(dsk_{ver}, up_{ver+1})$. The *UserUpdate* algorithm, performed by users themselves, takes as input the user's decryption secret key dsk_{ver} and current tuple up_{ver+1}; it outputs the updated decryption secret key dsk_{ver+1}.

$ReEncrypt(CT_{ver}, Rekey_{ver \to ver+1})$. The *ReEncrypt* algorithm, executed by proxy server, takes as input the ciphertext CT_{ver} and the reencryption key $ReKey_{ver \to ver+1}$; it generates a updated ciphertext CT_{ver+1}.

$GenTK_{out}(pk, dsk_{ver+1})$. The *GenTK* algorithm, run by data sharer, takes as input the system public key *pk* and the user's decryption secret key dsk_{ver+1}; it outputs a blinded decryption secret key dsk'_{ver+1} and a transformation key tk_{ver+1}.

$Transform_{out}(CT_{ver+1}, tk_{ver+1})$. The *Transform* algorithm, implemented by DSP, takes as input the ciphertext CT_{ver+1} and the transformation key tk_{ver+1} from data sharer (i.e., mobile patients or mobile physicians); if $\mathcal{S}$ does not satisfy access structure, it then outputs $\perp$. Otherwise, it outputs a transformation ciphertext CT'_{ver+1}.

$Decrypt_{verify}(CT'_{ver+1}, dsk_{ver+1})$. The *Decrypt* algorithm, run by data sharers, takes as input the transformation ciphertext CT'_{ver+1} and the updated decryption secret key dsk_{ver+1}; it outputs the decrypted message M if validating the correctness of transformed ciphertext CT'_{ver+1}.

4.2. System Model. We consider an efficient personal health information (PHI) data sharing architecture [32, 33] by an example that mobile patients featured with the same symptoms [3] or physicians can form a social group and can rent a cloud server to store and share PHI data with each other in a flexible access manner. Based upon the above premise, several different entities are involved in our system model (Figure 2): PHI cloud storage server, trusted authority, decryption cloud server provider, trusted group administrator, and a large number of social group users including PHI data owners and PHI data sharers. The trusted authority undertakes the responsibility concerning attribute authentication and key distribution. The trusted group administrator takes charge of group management, certificate generation, key update for social group users, and ciphertext update for reencryption requests. The users (i.e., mobile patients and mobile physicians) in the same social group share their health conditions and medical care experiences with their mobile devices. PHI cloud storage server, which is assumed to be a honest but curious entity, provides social group users with some storage and reencryption services; that is, it faithfully implements all operations requested by users and purposefully retrieves the stored ciphertext to collect additional valuable PHI information. The decryption cloud server provider [20] offers the services that can help users to convert a complex decryption task into a simple one.

4.3. Security Model. We now give the definition of the chosen plaintext security (CPA) security for CP-ABE scheme with verifiability. In this security model, the revoked user may collude with unrevoked user to obtain some unauthorized data [34]. We suppose that the revoked user can get private keys that satisfy the access structure; the version however differs from the current version. Contrarily, the unrevoked user can achieve private keys that do not satisfy the access structure, but the version is the current version. To formalize the security model, the game is described between an adversary $\mathcal{A}$ and a challenger $\mathcal{B}$ as follows.

Init. The adversary $\mathcal{A}$ gives the challenger $\mathcal{B}$ its challenge LSSS access structure $L^* = \{\mathcal{M}^*, \varrho^*\}$, group identity gid^*, and the version ver^*.

Setup. $\mathcal{B}$ first executes *SystemSetup()* algorithm to obtain the system master secret key *msk* and the public parameter *pk*. $\mathcal{B}$ then performs *GroupSetup()* algorithm to achieve the group master key gmk_0 and the group public key gpk_0 for gid^*. Moreover, $\mathcal{B}$ runs *GroupUpdate()* algorithm to get the group master key gmk_{ver}, the group public key gpk_{ver}, and reencryption key $Rekey_{ver-1 \to ver}$, where $ver = \{1, 2, \ldots, ver^*\}$. $\mathcal{B}$ finally sends $\mathcal{A}$ the public parameter *pk* and the group public keys $\{gpk_{ver}\}_{0 \le ver \le ver^*}$ and keeps the system mater key *mk*, the group master keys $\{gpk_{ver}\}_{0 \le ver \le ver^*}$, and the reencryption key $\{Rekey_{ver-1 \to ver}\}_{ver=\{1,2,\ldots,ver^*\}}$.

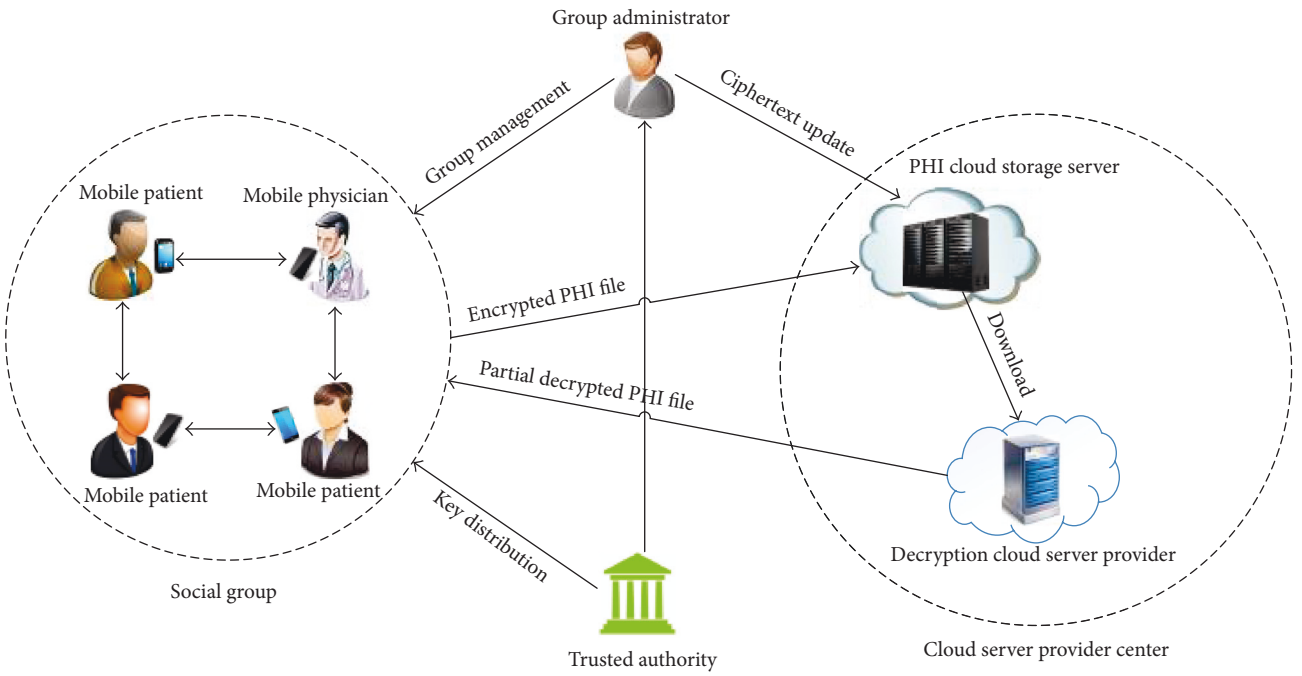

Figure 2: Architecture of the proposed PHI data sharing protocol.

Phase 1. $\mathcal{A}$ repeatedly issues queries as follows, including Type-*A* query, Type-*B* query, User update query, and Reencryption query.

(i) Type-*A* query:

(1) *Certificate* query, on input user's identity *uid*, group identity gid^*, and the version number *ver*, where $ver < ver^*$: $\mathcal{B}$ runs $\delta_{ver} \leftarrow CerGen(pk, uid, gmk_{ver})$, and then it returns to $\mathcal{A}$ the certificate δ_{ver}.

(2) *Private key* query, on input user's identity *uid*, group identity gid^*, a set of attributes $\mathcal{S}$ satisfying the access structure L^*, and the certificate $\{\delta_{ver}\}_{ver<ver^*}$: $\mathcal{B}$ executes $dsk_{ver} \leftarrow KeyGen(pk, msk, gpk_{ver}, \mathcal{S}, uid, \delta_{ver})$ and then issues the private secret key dsk_{ver} to $\mathcal{A}$.

(3) *Transformation key* query, on input the system public key *pk*, the version number *ver*, and the corresponding private key dsk_{ver}: $\mathcal{B}$ runs $dsk'_{ver} \leftarrow GenTK_{out}(pk, dsk_{ver})$, where $dsk'_{ver} = (rk, tk)$. Then it issues the new private secret key dsk'_{ver} to $\mathcal{A}$.

(ii) Type-*B* query:

(1) *Certificate* query, on input user's identity *uid*, group identity gid^*, and the version number ver^*: $\mathcal{B}$ runs $\delta_{ver^*} \leftarrow CerGen(pk, uid, gmk_{ver^*})$ and then returns the certificate δ_{ver^*} to $\mathcal{A}$.

(2) *Private key* query, on input user's identity *uid*, group identity gid^*, a set of attributes $\mathcal{S}$ dissatisfying the access structure L^*, and the certificate δ_{ver^*}: $\mathcal{B}$ executes $dsk_{ver^*} \leftarrow KeyGen(pk, mk, gpk_{ver^*}, \mathcal{S}, uid, \delta_{ver^*})$ and then issues the private key dsk_{ver^*} to $\mathcal{A}$.

(3) *Transformation key* query, on input the system public key *PK*, the version number ver^*, and the corresponding private key dsk_{ver^*}: $\mathcal{B}$ runs $dsk'_{ver^*} \leftarrow GenTK_{out}(pk, dsk_{ver^*})$, where $dsk'_{ver^*} = (tk, tk)$. Then it issues the new private secret key dsk'_{ver^*} to $\mathcal{A}$.

(iii) *User update* query, on user's identify *uid* and the decryption private $\{dsk_{ver}\}_{ver<ver^*-1}$: $\mathcal{B}$ sends to $\mathcal{A}$ the tuple up_{ver+1}. Then $\mathcal{A}$ performs $dsk_{ver+1} \leftarrow UserUpdate(dsk_{ver}, up_{ver+1})$.

(iv) *Reencryption* query, on input $\{CT_{ver}\}_{ver<ver^*}$: $\mathcal{B}$ runs $CT_{ver+1} \leftarrow ReEncrypt(CT_{ver}, Rekey_{ver \to ver+1})$ and returns CT_{ver+1} to $\mathcal{A}$.

Challenge. $\mathcal{A}$ submits two equal length messages M_0 and M_1 to $\mathcal{B}$. $\mathcal{B}$ picks a random bit $\eta \in \{0, 1\}$, sets $CT^* \leftarrow Encyrpt(pk, gpk_{ver}, M_\eta, L^*)$, and sends CT^* to $\mathcal{A}$.

Phase 2. $\mathcal{A}$ continues to send *Type-A, Type-B, User update, and Reencryption* queries as in Phase 1.

Guess. $\mathcal{A}$ outputs its guess $\eta' \in \{0, 1\}$ for η and wins the game if $\eta = \eta'$.

The advantage of the adversary $\mathcal{A}$ in this game is defined as $Adv_{\mathcal{A}} = |\Pr[\eta = \eta'] - 1/2|$.

5. Concrete Construction

Our scheme is based on flexible and fine-grained attribute-based data storage [24]. In this section, we suggest an efficient CP-ABE scheme based on LSSS structure under the aforementioned DCDH assumption.

$(\mathbb{G}, \mathbb{G}_T)$ can be denoted as a multiplicative cyclic bilinear group pair and these two group elements in pair have the same large prime order p. Let g be a generator of group $\mathbb{G}$ and $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ be a bilinear map. Let $H : \{0,1\}^* \to \mathbb{G}$ and $H_1 : \mathbb{G} \to \mathbb{Z}_p^*$ denote that a hash function H maps an identity or an attribute to a group value in $\mathbb{G}$ and a hash function H_1 maps an elements in $\mathbb{G}$ to a random number in $\mathbb{Z}_p$, respectively.

$SystemSetup(\lambda) \to (pk, msk)$. This algorithm is executed by trusted authority (TA). TA first picks $\alpha, \theta, a \in \mathbb{Z}_p^*$ and $u, w, v \in \mathbb{G}$ and chooses a key derivation function (KDF) with the output length l. Then it calculates $g^\alpha, g^\theta, g^{1/\theta}, g^b, e(g,g)^\alpha$. Finally, it publishes public parameters $pk = (\mathbb{G}, g, u, w, v, g^\theta, e(g,g)^\alpha, g^a, \mathsf{KDF}, l)$, keeps the master key $msk = (g^\alpha, \theta)$, and simultaneously issues $g^{1/\theta}$ to trusted group administrator (GA) to execute the revocation operation.

$GroupSetup(pk) \to (gmk_0, gpk_0, dic_0)$. This algorithm is run by GA. GA first selects a random $\beta \in \mathbb{Z}_p^*$ and sets the group master secret key $gmk = \beta$. Then, GA calculates $g^\beta, e(g,g)^\beta$. Lastly, it publishes the group public parameters $gpk = (gid, g^\beta, e(g,g)^\beta)$, where gid denotes group identity. dic_0 stands for a dictionary which initializes an empty version. In our system, for example, 0 denotes initial version. The version will be updated to a new version when any user leaves the system. Let ver be the current version.

$CertGen(pk, uid, gmk_{ver}) \to \delta_{ver}$. This algorithm is implemented by GA. When a new user who wants to join the group system requests a group certificate. Once the GA accepts his/her request, it produces a certificate as $\delta_{ver} = g^{\theta\beta_{ver}} \times H(uid)^{\beta_{ver}}$ and sends it to the user.

$KeyGen(pk, msk, \mathcal{S}, gpk_{ver}, uid, \delta_{ver}) \to (dsk_{ver}, up_{ver})$. This algorithm is performed by TA. User's attributes and identity could be authenticated by TA and then this algorithm is run to generate decryption secret key as follows:

(1) TA first validates the authenticity of certificate to recognize whether the user is a group member or not by $e(g, \sigma_{ver}) \stackrel{?}{=} e(g^\theta \times H(uid), g^{\beta_{ver}})$. If the equation is true, then the certificate is valid and goes to the next step. Otherwise, it returns an error symbol $\perp$.

(2) TA generates decryption secret key dsk_{ver} for user according to his/her identity uid and a set of user attributes $\mathcal{S}$. TA picks $s_1, s_2, r, \tau, \sigma \in \mathbb{Z}_p^*$ at random and computes $H(uid)^{\beta_{ver}} = \delta_{ver}/(g^{\beta_{vet}})^\theta$; then the private key is produced as follows:

$$\begin{aligned} dsk_{\mathcal{S}} &= \left\{D_1 = g^{s_1(\alpha+r)/(\theta+\sigma)} \cdot v^{s_1},\ L = g^{s_1},\ D_2 = \sigma,\ D_3 \right. \\ &\left. = g^{as_1/\theta} \cdot H(uid)^{\beta_{ver}s_1/\theta},\ \left\{K_x = H^{s_1}(x)\right\}_{x\in\mathcal{S}}\right\}, \\ dsk_{gid} &= \left\{D_4 = g^{\beta_{ver}s_2},\ D_5 = H(uid)^{\beta_{ver}s_2},\ D_6 = g^{s_2 r}\right. \\ &\left. \cdot H(gid)^{s_2\tau} \cdot v^{(\theta+\sigma)s_2},\ D_7 = g^{s_2\tau}\right\}. \end{aligned} \tag{2}$$

(3) TA sets the decryption key $dsk_{ver} = (dsk_{\mathcal{S}}, dsk_{gid}, s_1, s_2)$ and sends it to the user in the group; besides, it also delivers the current tuple $up_{ver} = (uid, d_1 = H(uid)^{1/\theta}, d_2 = g^{1/\theta})$ to GA.

$Encrypt(pk, gpk_{ver}, M, (\mathcal{M}, \rho)) \to CT_{ver}$. This algorithm is run by PHI data owner. This algorithm takes as input the system public key pk, the group public key gpk, a plaintext message M, and a LSSS access structure $L = (\mathcal{M}, \varrho)$, where $\mathcal{M}$ denotes a $l \times n$ matrix and $\mathcal{M}_k$ is the vector corresponding to the kth row of $\mathcal{M}$. ϱ is a map from each $\mathcal{M}_i$ of $\mathcal{M}$ to the party ϱ_i. The algorithm first picks a random vector $\vec{v} = (t_2, v_2, \ldots, v_n) \in \mathbb{Z}_p^{*n}$ such that t_2 can be shared. For $k = 1$ to l, it is easy to compute $\lambda_k = \vec{v} \cdot \mathcal{M}_i$. The detailed steps by PHR data owner proceed as follows:

(1) Choose $t_1 \in \mathbb{Z}_p^*$ randomly and generate a new session key ssk with the encapsulated key $key = e(g,g)^{\alpha t_1}$; besides, compute

$$\begin{aligned} \mathsf{KDF}(key, l) &= ssk \parallel f, \\ \widehat{C} &= u^{H_1(ssk)} \cdot w^{H_1(f)}, \\ C_{ver} &= M \cdot e(g,g)^{\alpha t_1 + \beta_{ver} t_2}, \\ C_1' &= g^{\theta t_1}, \\ C_2' &= g^{\theta t_2}, \\ C_1'' &= g^{t_1}, \\ C_2'' &= g^{t_2}, \\ C_3' &= H(gid)^{t_1}. \end{aligned} \tag{3}$$

(2) Pick $r_1, \ldots, r_l \in \mathbb{Z}_p^*$ randomly, and calculate

$$\begin{aligned} &\left(C_1 = g^{a\lambda_1} \cdot H(\rho(1))^{-r_1},\ F_1 = g^{r_1}\right), \ldots, \\ &\left(C_l = g^{a\lambda_l} \cdot H(\rho(l))^{-r_l},\ F_l = g^{r_l}\right). \end{aligned} \tag{4}$$

(3) Output the ciphertext

$$\begin{aligned} &CT_{ver} \\ &= \left(ver, L, \widehat{C}, C_{ver}, C_1', C_2', C_1'', C_2'', C_3', \left\{(C_k, F_k)\right\}_{k\in\{1,l\}}\right). \end{aligned} \tag{5}$$

And, then upload CT_{ver} to CSS.

$GroupUpdate(pk, gmk_{ver}, dic_{ver}) \rightarrow (gmk_{ver+1}, gpk_{ver+1}, Rekey_{ver\rightarrow ver+1}, dic_{ver+1})$. This algorithm is carried out by GA. When social group members leave the group, the group master key and public key should be updated by GA as follows:

(1) Pick $\beta_{ver+1} \in \mathbb{Z}_p^*$ as a updated group master key $gmk_{ver+1} = \beta_{ver+1}$, and update the group public key $gpk_{ver+1} = (gid, g^{\beta_{ver+1}}, e(g,g)^{\beta_{ver+1}})$. Moreover, compute a reencryption key $Rekey_{ver\rightarrow ver+1} = g^{(\beta_{ver+1}-\beta_{ver})/\theta}$.

(2) Update the current tuple $up_{ver+1} = (uid, d_1 = H(uid)^{(\beta_{ver+1}-\beta_{ver})/\theta}, d_2 = g^{(\beta_{ver+1}-\beta_{ver})})$ and issue up_{ver+1} to each group member.

$UserUpdate(dsk_{ver}, up_{ver+1}) \rightarrow dsk_{ver+1}$. This algorithm is performed by PHI data sharer in the social group. When a sharer leaves the group, the group keys need to be updated by GA as above. Besides, other sharers in the group also need to update their decryption keys by themselves. dsk_{ver+1} is updated as follows:

(1) Calculate

$$\begin{aligned} D_3 &= D_3 \times d_1^{s_1} = g^{as_1/\theta} \times H(uid)^{\beta_{ver+1}s_1/\theta}, \\ D_4 &= D_4 \times d_2^{s_2} = g^{\beta_{ver+1}s_2}, \\ D_5 &= D_5 \times d_1^{s_2} = H(uid)^{\beta_{ver+1}s_2}. \end{aligned} \tag{6}$$

(2) Update

$$\begin{aligned} dsk_{\mathcal{S}} &= \left\{D_1 = g^{s_1(\alpha+r)/(\theta+\sigma)} \cdot v^{s_1},\ D_2 = \sigma,\ D_3 = g^{as_1/\theta} \cdot H(uid)^{\beta_{ver+1}s_1/\theta},\ L = g^{s_1},\ \left\{K_x = H^{s_1}(x)\right\}_{x\in\mathcal{S}}\right\}, \\ dsk_{gid} &= \left\{D_4 = g^{\beta_{ver+1}s_2},\ D_5 = H(uid)^{\beta_{ver+1}s_2},\ D_6 = g^{s_2 r} \cdot H(gid)^{s_2\tau} \cdot v^{(\theta+\sigma)s_2},\ D_7 = g^{s_2\tau}\right\}. \end{aligned} \tag{7}$$

(3) The updated decryption secret key dsk_{ver+1} is denoted as

$$dsk_{ver+1} = \left(s_1, s_2, dsk_{\mathcal{S}}, dsk_{gid}\right). \tag{8}$$

$ReEncrypt(CT_{ver}, Rekey_{ver\rightarrow ver+1}) \rightarrow CT_{ver+1}$. This algorithm is executed by CSS. After updating user's decryption keys, the user who uses his/her updated key dsk cannot decrypt the original ciphertext stored in CSS anymore; therefore, the ciphertext needs to be updated by reencryption operations as follows:

(1) Calculate

$$\begin{aligned} C_{ver+1} &= C_{ver} \cdot e\left(Rekey_{ver\rightarrow ver+1}, C_2'\right) \\ &= Me(g,g)^{\alpha t_1 + \beta_{ver} t_2}\, e\left(g^{(\beta_{ver+1}-\beta_{ver})/\theta}, g^{\theta t_2}\right) \\ &= M \cdot e(g,g)^{\alpha t_1 + \beta_{ver+1} t_2}. \end{aligned} \tag{9}$$

(2) Output the updated ciphertext

$$CT_{ver+1} = \left(ver+1, L, \widehat{C}, C_{ver+1}, C_1', C_2', C_1'', C_2'', C_3', \left\{\left(C_k, D_k'\right)\right\}_{k\in\{1,l\}}\right). \tag{10}$$

$GenTK_{out}(pk, dsk_{ver+1}) \rightarrow dsk'_{ver+1}$. This algorithm is run by the existing social group sharer. Given the user's decryption secret key dsk_{ver+1}, it can convert the original decryption secret key into a blinded decryption secret key in the following:

(1) Pick a random $z \in \mathbb{Z}_p^*$ and compute

$$\begin{aligned} s_1' &= \frac{s_1}{z}, \\ s_2' &= \frac{s_2}{z}, \\ dsk'_{\mathcal{S}} &= \left\{D_1' = g^{s_1'(\alpha+r)/(\theta+\sigma)} \cdot v^{s_1'},\ D_2' = \sigma,\ D_3' = g^{as_1'/\theta} \cdot H(uid)^{\beta_{ver}s_1'/\theta},\ L = g^{s_1'},\ \left\{K_x' = H^{s_1'}(x)\right\}_{x\in\mathcal{S}}\right\}, \\ dsk'_{gid} &= \left\{D_4' = g^{\beta_{ver}s_2'},\ D_5' = H(uid)^{\beta_{ver}s_2'},\ D_6' = g^{s_2' r} \cdot H(gid)^{s_2'\tau} \cdot v^{(\theta+\sigma)s_2'},\ D_7' = g^{s_2'\tau}\right\}. \end{aligned} \tag{11}$$

Note that herein we set $D_2 = D_2' = \sigma$ which is not blinded.

(2) Set the decryption secret key $dsk'_{ver+1} = (rk, tk_{ver+1})$, where the blinded transformation key $tk_{ver+1} = (dsk'_{\mathcal{S}}, dsk'_{gid})$ and the retrieval key $rk = (s_1', s_2')$, and then send tk_{ver+1} to the DSP.

$Transform_{out}(CT_{ver+1}, tk_{ver+1}) \rightarrow CT'_{ver+1}$. This algorithm is performed by DSP. The *Transform* algorithm takes as input the ciphertext CT_{ver+1} and the transformation key TK_{ver+1} from user; if $\mathcal{S}$ does not satisfy access structure, it then outputs $\perp$. Assume that $\mathcal{S}$ satisfies the access structure and let $I \subseteq \{1, 2, \ldots, l\}$ be defined as $I = \{k : \varrho(k) \in \mathcal{S}\}$. It could calculate constants $\{\omega_k \in \mathbb{Z}_p^*\}_{k\in I}$ such that if $\{\lambda_k\}$ are valid shares of any secret s according to $\mathcal{M}$, then $\sum_{k\in I} \omega_k \lambda_k = t_2$. The algorithm partially deciphers the ciphertext as follows and sends the partially decrypted ciphertext CT' to the PHI data sharer.

(1) Calculate

$$\begin{aligned} T_1 &= \frac{e\left(C_2', D_3'\right)}{\left(e\left(\Pi_{k\in I} C_k^{\omega_k}, L'\right) \cdot \Pi_{k\in I} e\left(F_k^{\omega_k}, K'_{\varrho(k)}\right)\right)} \\ &= \frac{e\left(g^{\theta t_2}, g^{as_1/\theta} \times H(uid)^{\beta_{ver+1}s_1/\theta}\right)^{1/z}}{\left(\Pi_{k\in I} e(g,g)^{s_1 a \lambda_k \omega_k}\right)^{1/z}} \end{aligned}$$

$$= \frac{e\left(g^{\theta t_2}, g^{as_1/\theta} \times H(uid)^{\beta_{ver+1}s_1/\theta}\right)^{1/z}}{e(g,g)^{s_1 a t_2/z}}$$

$$= e(g, H(uid))^{\beta_{ver+1}t_2 s_1/z},$$

$$T_2 = \frac{e\left(C_1'', D_6'\right)}{e\left(C_3', D_7'\right)}$$

$$= \frac{e\left(g^{t_1}, g^{s_2 r} \cdot H(gid)^{s_2\tau} \cdot v^{(\theta+\sigma)s_2}\right)^{1/z}}{e\left(H(gid)^{t_1}, g^{s_2\tau}\right)^{1/z}}$$

$$= e(g,g)^{t_1 s_2 r/z} \cdot e(g,v)^{(\theta+\sigma)t_1 s_2/z}. \tag{12}$$

(2) Similarly, compute

$$E_1 = e\left(C_1''^{D_2'} C_1', D_1'\right)$$

$$= e\left(g^{(\theta+\sigma)t_1}, g^{s_1(\alpha+r)/(\theta+\sigma)} \cdot v^{s_1}\right)^{1/z}$$

$$= e(g,g)^{t_1(\alpha+r)s_1/z} e(g,v)^{(\theta+\sigma)t_1 s_1/z},$$

$$E_2 = e\left(C_2'', D_4'\right) = e\left(g^{t_2}, g^{\beta_{ver+1}s_2}\right)^{1/z} \tag{13}$$

$$= e(g,g)^{\beta_{ver+1}s_2 t_2/z},$$

$$E_3 = e\left(C_2'', D_5'\right) = e\left(g^{t_2}, H(uid)^{\beta_{ver+1}s_2}\right)^{1/z}$$

$$= e(g, H(uid))^{\beta_{ver+1}s_2 t_2/z}.$$

(3) DSP issues the partial decrypted ciphertext $CT'_{ver+1} = \{\widehat{C}, C_{ver+1}, T_1, T_2, E_1, E_2, E_3\}$ to the social group sharers.

$Decrypt_{verify}(CT'_{ver}, dsk'_{ver+1}) \rightarrow M$. This algorithm is executed by PHI data sharer. the algorithm could decipher the decrypted ciphertext as follows:

(1) Calculate F_r', F_g', E_1', E_2' as

$$T_1' = T_1^{1/s_1'} = e(g, H(uid))^{\beta_{ver+1}t_2},$$

$$T_2' = T_2^{1/s_2'} = e(g,g)^{t_1 r} \cdot e(g,v)^{(\theta+\sigma)t_1},$$

$$E_1' = E_1^{1/s_1'} = e(g,g)^{t_1(\alpha+r)} e(g,v)^{(\theta+\sigma)t_1}, \tag{14}$$

$$E_2' = E_2^{1/s_2'} = e(g,g)^{\beta_{ver+1}t_2},$$

$$E_3' = E_3^{1/s_2'} = e(g, H(uid))^{\beta_{ver+1}t_2}.$$

(2) Compute the encapsulated key and the session key ssk as

$$key = \frac{E_1' \cdot E_3'}{T_1' \cdot T_2'}$$

$$= \frac{e(g,g)^{t_1(\alpha+r)} e(g,v)^{(\theta+\sigma)t_1} \cdot e(g, H(uid))^{\beta_{ver+1}t_2}}{e(g, H(uid))^{\beta_{ver+1}t_2} \cdot e(g,g)^{t_1 r} \cdot e(g,v)^{(\theta+\sigma)t_1}} \tag{15}$$

$$= e(g,g)^{\alpha t_1},$$

$$\mathsf{KDF}(key, l) = ssk \parallel f.$$

(3) If $\widehat{C} = u^{H_1(ssk)} w^{H_1(f)}$, it outputs the encapsulated key key and then retrieves the message M as $M = CT_{ver+1} \cdot (key \cdot E_2')^{-1}$. Otherwise, it outputs $\perp$.

6. Security Analysis

In this section, we present the security for our proposed CP-ABE scheme. The main issue in our scheme is also to resist the collusion attack between the revoked users and existing legitimate users.

Theorem 4. *Suppose that the construction of Li [24] is a selectively CPA secure CP-ABE scheme; then our scheme proposed is also selectively CPA secure. In our construction, provided that the hash function H is a random oracle, if there exists a probabilistic polynomial-time adversary $\mathscr{A}$ that can break our scheme with a nonnegligible advantage $\varepsilon > 0$ after l_1 Type-A queries and l_2 Type-B queries, then there exists a challenger $\mathscr{B}$ that can solve the DCDH problem with the advantage*

$$Adv_{\mathscr{A}} \leq \frac{\varepsilon}{2^{l_1+l_2}} \cdot l_1 \cdot l_2. \tag{16}$$

Let $\mathscr{B}$ be a DCDH attacker who receives a random instance $\{p, \mathbb{G}, \mathbb{G}_T, e, g, (P,Q) = (g^b, g^d)\}$ of DCDH problem in $\mathbb{G}$ and has to calculate the value of $g^{b/d}$. $\mathscr{A}$ is an adversary who interacts with the attacker $\mathscr{B}$ as modeled in aforementioned game of security model. We present how the attacker $\mathscr{B}$ can use the adversary $\mathscr{A}$ to solve the DCDH problem, that is, how to compute the value of $g^{b/d}$.

To easily understand the proof, the reduction is briefly presented. When the game starts, the attacker $\mathscr{B}$ first initializes the instance $(P,Q) = (g^b, g^d)$ of the DCDH problem and then simulates hash functions as random oracles. During the simulation, the attacker $\mathscr{B}$ needs to guess the adversary $\mathscr{A}$'s target identity and message. the attacker $\mathscr{B}$ finally will set dsk_{A-i} and dsk_{B-i} (please consult our proof for the settings). At the end of game, the adversary $\mathscr{A}$ can output the value of $g^{b/d} = g^{z_i'/z_i\beta_{ver}}$ as his answer by the evidence $Q^{z_i'} = (H_i)^{\beta_{ver}}$.

Proof. $\mathscr{B}$ is given a challenge instance

$$\left\{p, G, G_T, e, g, (P,Q) = \left(g^b, g^d\right)\right\}, \tag{17}$$

where $b, d \in \mathbb{Z}_p^*$. $\mathscr{B}$ interacts with $\mathscr{A}$ as follows.

Init. The adversary $\mathcal{A}$ gives the challenger $\mathcal{B}$ its challenge LSSS access structure $L^* = \{\mathcal{M}^*, \rho^*\}$, group identity gid^*, and the version ver^*.

Setup. $\mathcal{B}$ does the following steps to set the public parameter pk and the group public key gpk:

(1) Choose random $a, x, y, \alpha, \theta \in \mathbb{Z}_p^*$, and pick a key derivation function (KDF) with the output length l. Also pick a collision-resistant hash function $H_1 : \mathbb{G} \to \mathbb{Z}_p^*$, and then calculate $g^a, g^\theta, e(g,g)^\alpha$, and set $pk = (g, G, g^a, u' = g^x, w' = g^y, g^\theta, e(g,g)^\alpha, \mathsf{KDF}, l, H_1)$, $msk = (\theta, g^\alpha)$.

(2) Pick $\beta_0, \beta_1, \ldots, \beta_{ver^*-1} \in \mathbb{Z}_p^*$ at random, compute $\{g^{\beta_{ver}}\}_{0\le ver<ver^*} \cup \{Q\}_{ver^*}$, $\{e(g,g)^{\beta_{ver}}\}_{0\le ver<ver^*} \cup \{e(g,Q)\}_{ver^*}$, and set $gpk_{ver} = \{gid^*, g^{\beta_{ver}}, e(g,g)^{\beta_{ver}}\}_{0\le ver<ver^*} \cup \{gid^*, B, e(g,Q)\}_{ver^*}$, $gmk = \{\beta_{ver}\}_{0\le ver<ver^*} \cup \{d\}_{ver^*}$. Note that d is unknown to $\mathcal{B}$.

(3) Issue pk, gpk_{ver} to $\mathcal{A}$, and keep msk, gmk in himself.

H-Queries. $\mathcal{B}$ uses three lists H_{uid}^{list}, H_{gid}^{list}, H_{ϱ}^{list} to reply $\mathcal{A}$'s queries (user's identity, group identity, and $\varrho(k)$ queries).

(1) User's identity query (H_{uid_i}): if uid_i already exists in $H_{uid_i}^{list}$ as $\langle uid_i, H_i, z_i, r_i\rangle$, then $\mathcal{B}$ returns $H_{uid_i} = H_i$. Otherwise, $\mathcal{B}$ flips a random coin r_i with the probability $\Pr[r_i = 1] = \epsilon$. If $r_i = 0$, then $\mathcal{B}$ picks $z_i \in Z_p^*$ randomly and inserts the tuple $\langle uid_i, H_i = g^{z_i}, z_i, r_i\rangle$ into H_{uid}^{list}. If $r_i = 1$, then $\mathcal{B}$ chooses random $z_i \in Z_p^*$ and adds the tuple $\langle uid_i, H_i = P^{z_i}, z_i, r_i\rangle$ into H_{uid}^{list}. $\mathcal{B}$ gives $\mathcal{A}$ the user's identity query $H(uid_i) = H_i$.

(2) Group identity query (H_{gid_j}): if gid_j already exists in $H_{gid_j}^{list}$ as $\langle gid_j, H_j, m_j\rangle$, then $\mathcal{B}$ returns $H_{gid_j} = H_j$. Otherwise, $\mathcal{B}$ chooses random $m_i \in \mathbb{Z}_p^*$ and adds the tuple $\langle gid_j, H_j = g^{m_j}, m_j\rangle$ into the H_{gid}^{list}. $\mathcal{B}$ gives $\mathcal{A}$ the group identity query $H(gid_j) = H_j$.

(3) $\varrho(k)$ query ($H(\varrho(k))$): if $\varrho(k)$ already exists in H_{ϱ}^{list} as $\langle \rho(k), H_k, n_k\rangle$, then $\mathcal{B}$ returns $H(\varrho(k)) = H_k$. Otherwise, $\mathcal{B}$ chooses random $n_k \in \mathbb{Z}_p^*$ and adds the tuple $\langle \varrho(k), H_k = g^{n_k}, n_k\rangle$ into H_{ϱ}^{list}. $\mathcal{B}$ gives $\mathcal{A}$ the ρ query $H(\varrho(k)) = H_k$.

Phase 1. $\mathcal{A}$ repeatedly issues queries to $\mathcal{B}$ as follows; $\mathcal{B}$ utilizes two lists to send queries Key_A^{list}, Key_B^{list} (initially empty).

(i) Type-*A* query:

(1) *Certificate* query $\langle uid_i, gid^*, ver\rangle_{ver<ver^*}$: suppose that uid_i exists in H_{uid}^{list} as $\langle uid_i, H_i, z_i, r_i\rangle$; otherwise, $\mathcal{B}$ runs a request himself as in H_{uid_i}. If $r_i = 1$, then $\mathcal{B}$ generates a certificate $\delta_{ver} = g^{\theta\beta_{ver}} \times P^{z_i\beta_{ver}}$, and if $r_i = 0$, then $\mathcal{B}$ outputs $\perp$.

(2) *Private key* query $\langle uid_i, gid^*, \mathcal{S}, ver\rangle_{ver<ver^*}$, where $\mathcal{S}$ satisfies a LSSS access structure L^*: if uid_i has already appeared in Key_B^{list} as $\langle uid_i, dsk_{B-i}\rangle$, then $\mathcal{B}$ outputs $\perp$. Otherwise, $\mathcal{B}$ first obtains $\langle uid_i, H_i, z_i, r_i\rangle$, $\langle gid^*, H^*, m^*\rangle$ from H_{uid}^{list}, H_{gid}^{list}, respectively, and $\mathcal{B}$ picks $s_1, s_2, r, \tau, \sigma \in \mathbb{Z}_p^*$ randomly and calculates the private key as follows:

$$dsk_{\mathcal{S}} = \Big\{D_1 = g^{s_1(\alpha+r)/(\theta+\sigma)} \cdot v^{s_1},\ D_2 = \sigma,\ D_3 = g^{as_1/\theta} \cdot P^{z_i\beta_{ver}s_1/\theta},\ L = g^{s_1},\ \{K_x = H^{s_1}(x)\}_{x\in\mathcal{S}\wedge x=\varrho(k)}\Big\},$$
$$dsk_{gid} = \Big\{D_4 = g^{\beta_{ver}s_2},\ D_5 = P^{z_i\beta_{ver}s_2},\ D_6 = g^{s_2 r} \cdot H^{*s_2\tau} \cdot v^{(\theta+\sigma)s_2},\ D_7 = g^{s_2\tau}\Big\}. \tag{18}$$

(3) *Transformation key* query $\langle s_1, s_2, ver, dsk_{\mathcal{S}}, dsk_{gid^*}\rangle_{ver<ver^*}$: $\mathcal{B}$ chooses a random exponent $z \in \mathbb{Z}_p^*$ and then sets

$$s_1' = \frac{s_1}{z},$$
$$s_2' = \frac{s_2}{z},$$
$$dsk_{\mathcal{S}}' = \Big\{D_1' = g^{s_1'(\alpha+r)/(\theta+\sigma)} \cdot v^{s_1'},\ D_2' = \sigma,\ D_3' = g^{as_1'/\theta} \cdot P^{z_i\beta_{ver}s_1'/\theta},\ L' = g^{s_1'},\ \Big\{K_x' = H^{s_1'}(x)\Big\}_{x\in\mathcal{S}}\Big\},$$
$$dsk_{gid}' = \Big\{D_4' = g^{\beta_{ver}s_2'},\ D_5' = P^{z_i\beta_{ver}s_2'},\ D_6' = g^{s_2' r} \cdot (H^*)^{s_2'\tau} \cdot v^{(\theta+\sigma)s_2'},\ D_7' = g^{s_2'\tau}\Big\}. \tag{19}$$

$\mathcal{B}$ finally sends $dsk_{A-i} = \{rk = (s'_1, s'_2), tk_{\mathcal{S}} = (\mathcal{S}, dsk_{\mathcal{S}}', dsk_{gid^*}')\}$ to $\mathcal{A}$ and adds $\langle uid_i, dsk_{A-i}\rangle$ into the list Key_A^{list}.

(ii) Type-*B* query:

(1) *Certificate* query $\langle uid_i, gid^*, ver^*\rangle$: suppose that uid_i exists in H_{uid}^{list} as $\langle uid_i, H_i, z_i, r_i\rangle$; otherwise, $\mathcal{B}$ runs a request himself as in H_{uid_i}. If $r_i = 0$, then $\mathcal{B}$ generates a certificate $\delta_{ver^*} = Q^\theta \times Q^{z_i}$, and if $r_i = 1$, then $\mathcal{B}$ outputs $\perp$.

(2) *Private key* query $\langle uid_i, gid^*, \mathcal{S}, ver\rangle_{ver=ver^*}$, where $\mathcal{S}$ dissatisfies a LSSS access structure L^*: if uid_i has already appeared in Key_A^{list} as $\langle uid_i, dsk_{A-i}\rangle$, then $\mathcal{B}$ outputs $\perp$. Otherwise, $\mathcal{B}$ first obtains $\langle uid_i, H_i, z_i, r_i\rangle$, $\langle gid^*, H^*, m^*\rangle$ from H_{uid}^{list}, H_{gid}^{list}, respectively, and $\mathcal{B}$ picks s_1, s_2,

$r, \tau, \sigma \in \mathbb{Z}_p^*$ randomly and calculates the private key as follows:

$$dsk_{\mathcal{S}} = \Big\{D_1 = g^{s_1(\alpha+r)/(\theta+\sigma)} \cdot v^{s_1},\ D_2 = \sigma,\ D_3 = g^{as_1/\theta} \cdot Q^{z_i s_1/\theta},\ L' = g^{s_1},\ \{K_x = H^{s_1}(x)\}_{x\in\mathcal{S}\wedge x=\varrho(k)}\Big\},$$
$$dsk_{gid} = \Big\{D_4 = Q^{s_2},\ D_5 = Q^{z_i s_2},\ D_6 = g^{s_2 r} \cdot (H^*)^{s_2\tau} \cdot v^{(\theta+\sigma)s_2},\ D_7 = g^{s_2\tau}\Big\}. \tag{20}$$

(3) *Transformation key* query $\langle s_1, s_2, ver, dsk_{\mathcal{S}}, dsk_{gid^*}\rangle_{ver=ver^*}$: $\mathcal{B}$ chooses a random exponent $z \in \mathbb{Z}_p^*$ and then sets

$$s_1' = \frac{s_1}{z},$$
$$s_2' = \frac{s_2}{z},$$
$$dsk_{\mathcal{S}}' = \Big\{D_1' = g^{s_1'(\alpha+r)/(\theta+\sigma)} \cdot v^{s_1'},\ D_2' = \sigma,\ D_3' = g^{as_1'/\theta} \cdot Q^{z_i s_1'/\theta},\ L' = g^{s_1'},\ \Big\{K_x' = H^{s_1'}(x)\Big\}_{x\in\mathcal{S}}\Big\},$$
$$dsk_{gid}' = \Big\{D_4' = Q^{s_2'},\ D_5' = Q^{z_i s_2'},\ D_6' = g^{s_2' r} \cdot (H^*)^{s_2'\tau} \cdot v^{(\theta+\sigma)s_2'},\ D_7' = g^{s_2'\tau}\Big\}. \tag{21}$$

$\mathcal{B}$ finally sends $dsk_{B-i} = \{rk = (s_1', s_2'), tk_{\mathcal{S}} = (\mathcal{S}, dsk_{\mathcal{S}}', dsk_{gid^*}')\}$ to $\mathcal{A}$ and adds $\langle uid_i, dsk_{B-i}\rangle$ into the list Key_B^{list}.

(iii) *User update* query $\langle uid_i, ver, dsk_{A-i}\rangle_{ver<ver^*-1}$, where uid_i already exists in Type-A private key query. $\mathcal{B}$ achieves $\langle uid_i, H_i, z_i, r_i\rangle$ from H_{uid}^{list} and calculates $d_1 = P^{z_i\cdot(\beta_{ver+1}-\beta_{ver})/\theta}$, $d_2 = g^{(\beta_{ver+1}-\beta_{ver})}$; then $\mathcal{B}$ transmits the tuple $up_{ver+1} = \langle uid_i, d_1, d_2\rangle$ to $\mathcal{A}$; finally $\mathcal{A}$ uses the algorithm $UserUpdate(dsk_{A,i}, up_{ver+1})$ to update his private key.

(iv) *Reencryption* query $\langle CT_{ver}, gid^*\rangle_{ver<ver^*}$: $\mathcal{B}$ first calculates reencryption key as $Rekey_{ver\to ver+1} = g^{(\beta_{ver+1}-\beta_{ver})/\theta}$ and executes $CT_{ver+1} \leftarrow ReEncrypt(CT_{ver}, Rekey_{ver\to ver+1})$. Then $\mathcal{B}$ returns CT_{ver+1} to $\mathcal{A}$.

Challenge. $\mathcal{A}$ submits two equal length messages M_0, M_1; $\mathcal{B}$ then picks a random η and proceeds as follows:

(1) $\mathcal{B}$ selects $s_1, s_2 \in \mathbb{Z}_p^*$ randomly and achieves $\langle gid^*, H^*, m^*\rangle$ from H_{gid}^{list}. Moreover, $\mathcal{B}$ chooses $c_1, \ldots, c_l \in \mathbb{Z}_p^*$ at random and obtains $\langle \varrho(k), H_k, n_k\rangle$ from H_ϱ^{list}.

(2) $\mathcal{B}$ then generates the challenge ciphertext in the following.

$$CT_u = \Big\{version = ver^*, L^*, gid^*,\ C_\eta = M_\eta \cdot e(g,g)^{\alpha t_1} \cdot e(g,Q)^{t_2},\ C_1' = g^{\theta t_1},\ C_2' = g^{\theta t_2},\ C_1'' = g^{t_1},\ C_2'' = g^{t_2},\ C_3' = (H^*)^{t_1},\ \Big(C_k = g^{a\lambda_k} \cdot H(\varrho(k))^{-c_k},\ F_k = g^{c_k}\Big)_{k\in[0,l]}\Big\}. \tag{22}$$

(3) $\mathcal{A}$ runs *Encrypt* algorithm to obtain $key^* = e(g,g)^{\alpha s_1}$ and selects a random bit $r \in \{0,1\}$. If $r = 0$, it sets $key = key^*$. If $r = 1$, it chooses a random key R^* and sets $key = R^*$ and then sends (key, CT_η) to $\mathcal{B}$. $\mathcal{B}$ computes $\mathsf{KDF}(key, l) = ssk^* \parallel f^*$, $\widehat{C} = u'^{H_1(ssk^*)} w'^{H_1(f^*)}$.

(4) $\mathcal{B}$ finally delivers $(CT_\eta, \widehat{C})$ to $\mathcal{A}$.

Phase 2. $\mathcal{A}$ continues to request the above queries not issues as in Phase 1, and $\mathcal{B}$ sends the queries as in Phase 1.

Guess. If $\mathcal{A}$ wins the game, then $\mathcal{B}$ can compute

$$g^{x\cdot H_1(ssk^*)+yH_1(f^*)} = u'^{H_1(SSK^*)} w'^{H_1(f^*)} = \widehat{C}. \tag{23}$$

Eventually, $\mathcal{B}$ omits $\mathcal{A}'s$ output and chooses dsk_{A-i} from Key_A^{list} and dsk_{B-i} from Key_B^{list} randomly. If $\mathcal{A}$ wants to decipher the challenge ciphertext, he needs to obtain keys D_4', D_5' of dsk_{B-i} and D_3' of dsk_{A-i}. Namely, D_j can be achieved from $\langle uid_i', H_i', z_i', r_i' = 0\rangle$, and D_2 can be obtained from $\langle uid_i, H_i, z_i, r_i = 1\rangle$. Therefore, we could formalize the above theory evidence as $Q^{z_i'} = (H_i)^{\beta_{ver}}$ and we have the equation $g^{dz_i'} = g^{b\cdot z_i\cdot\beta_{ver}}$. $\mathcal{B}$ finally outputs $g^{b/d} = g^{z_i'/z_i\beta_{ver}}$ as the solution of DCDH problem.

Provided that $\mathcal{A}$ makes l_1 Type-A queries and l_2 Type-B queries, the probability that $\mathcal{B}$ does not return $\perp$ in Phase 1 is $\epsilon^{l_1}\cdot(1-\epsilon)^{l_2}$. The equation value reaches maximum when $\epsilon = 1/2$. Also, the probability that $\mathcal{B}$ always selects correct dsk_{A-i}, dsk_{B-i} is $1/l_1 l_2$. Therefore, it is very easy to compute $\mathcal{B}'s$ advantage at most $\varepsilon/2^{l_1+l_2}\cdot l_1\cdot l_2$.

Analysis. In our security model, we define the attack capability for an adversary $\mathcal{A}$ who can not only get private keys (transformation key) that satisfy the access policy, the version not being the current version, but also can receive private keys (transformation key) that do not satisfy the access policy, the version being the current version. The aforementioned statements imply that if the adversary can break our scheme, she/he can of course get

$$dsk_{\mathcal{S}} = \Big\{D_1 = g^{s_1(\alpha+r)/(\theta+\sigma)} \cdot v^{s_1},\ D_2 = \sigma,\ D_3 = g^{as_1/\theta} \cdot P^{z_i\beta_{ver}s_1/\theta},\ L = g^{s_1},\ \{K_x = H^{s_1}(x)\}_{x\in\mathcal{S}\wedge x=\varrho(k)}\Big\} \tag{24}$$

Table 2: Comparison of performances.

Scheme	Key update (user)	Decryption (server)	Decryption (user)	Ciphertext Size	Transformed ciphertext size	Access policy	Revocation	Verifiability
Scheme [24]	$(l+1)E_G$	$(2l+5)P+lE_{G_T}$	$4E_{G_T}$	$(2l+4)\lvert\mathbb{G}\rvert + \lvert\mathbb{G}_T\rvert + 2\lvert\mathbb{Z}_p\rvert$	$5\lvert\mathbb{G}_T\rvert$	Access tree	√	×
Our Scheme	$3E_G$	$(l+7)P + (2l+1)E_G$	$2E_G + 5E_{G_T}$	$(2l+6)\lvert\mathbb{G}\rvert + \lvert\mathbb{G}_T\rvert + 2\lvert\mathbb{Z}_p\rvert$	$\lvert\mathbb{G}\rvert + 6\lvert\mathbb{G}_T\rvert$	LSSS	√	√

Above l refers to an LSSS access structure with an $l \times n$ matrix.

from $dsk_{A,i}$ and

$$dsk_{gid} = \left\{D_4 = Q^{s_2},\ D_5 = Q^{z_i s_2},\ D_6 = g^{s_2 r} \cdot (H^*)^{s_2 \tau} \cdot v^{(\theta+\sigma)s_2},\ D_7 = g^{s_2 \tau}\right\} \tag{25}$$

from $dsk_{B,i}$. These two private keys would be used to decrypt the challenge ciphertext as follows:

$$\begin{aligned}
T_1' &= (T_1)^{s_1} = \frac{e\left(C_2', D_3'\right)}{\left(e\left(\Pi_{k\in I} C_k^{\omega_k}, L'\right) \cdot \Pi_{k\in I} e\left(F_k^{\omega_k}, K'_{\varrho(k)}\right)\right)} \\
&= e\left(g, P\right)^{\beta_{ver} z_i t_2}, \\
T_2' &= (T_2)^{s_2} = \frac{e\left(C_1'', D_6'\right)}{e\left(C_3', D_7'\right)} \\
&= e\left(g, g\right)^{t_1 r} \cdot e\left(g, v\right)^{(\theta+\sigma)t_1}, \\
E_1' &= (E_1)^{s_1} = e\left(C_1''^{D_2'} C_1', D_1'\right) \\
&= e\left(g, g\right)^{t_1(\alpha+r)} e\left(g, v\right)^{(\theta+\sigma)t_1}, \\
E_2' &= (E_2)^{s_2} = e\left(C_2'', D_4'\right) = e\left(g, Q\right)^{t_2}, \\
E_3' &= (E_3)^{s_2} = e\left(C_2'', D_5'\right) = e\left(g, Q\right)^{z_i' s_2 t_2}.
\end{aligned} \tag{26}$$

If an adversary $\mathcal{A}$ can break our scheme, then the adversary $\mathcal{A}$ can compute

$$\begin{aligned}
key &= \frac{E_1' \cdot E_3'}{T_1' \cdot T_2'} \cdot E_2' \\
&= \frac{e\left(g, g\right)^{\alpha t_1} \cdot e\left(g, Q\right)^{z_i' t_2} \cdot e\left(g, Q\right)^{t_2}}{e\left(g, P\right)^{z_i \beta_{ver} t_2}}.
\end{aligned} \tag{27}$$

And the *key* can successfully decipher the challenge ciphertext. That is to say, we can obtain $key = (e(g,g)^{\alpha t_1} \cdot e(g,Q)^{z_i' t_2} \cdot e(g,Q)^{t_2})/e(g,P)^{z_i \beta_{ver} t_2} = e(g,g)^{\alpha t_1} e(g,Q)^{t_2}$. Therefore, we can get the equation $e(g,Q)^{z_i' t_2} = e(g,P)^{z_i \beta_{ver} t_2}$. If the adversary can break our scheme such that $Q^{z_i'} = P^{z_i \beta_{ver}}$, the result $g^{b/d} = g^{z_i'/z_i \beta_{ver}}$ can be taken as his/her answer. □

7. Performance Analysis

In this chapter, the performance of our system is first theoretically evaluated concerning the computational overhead of key update, decryption for DSP and user, and communication cost, and the quantitative analysis of our scheme then is given compared to previous Li et al.'s scheme [24].

In Li et al.'s proposed scheme [24], the private keys have not been blinded, but partially, to be delivered to the trusted third party (DSP) to transform the original ciphertext into a simple ciphertext. Once receiving the transformed ciphertexts, the decryptor can easily recover the plaintext message without checking the correctness of transformed ciphertexts. However, it is not suitable for real outsourced applications because the third party is generally assumed as an untrusted one. In our proposed scheme, the private keys, first blinded by GenTK algorithm, are divided into the transformation key and retrieval key. After that, the blinded transformation key is sent to the untrusted third party (DSP) to translate the complex ciphertext into a simple one. Finally, once partial decrypted ciphertexts are verified as true, the plaintext message can be recovered by the retrieval key.

We express by P, $E_{\mathbb{G}}$, and $E_{\mathbb{G}_T}$ the time to a pairing computation, an exponentiation in $\mathbb{G}$, and an exponentiation in $\mathbb{G}_T$, respectively (other operations are ignored). From Table 2, we can learn that the computational cost of user's decryption key updating in Li et al.'s [24] follows a linear relationship with the number of attributes while the corresponding cost in our scheme only achieves constant. As for the time-cost of decryption (for server or user), from Table 2, we explicitly learn that the time-efficiency for decryption for DSP in ours is higher than Li et al.'s [24], which indirectly illustrates that the efficiency for decryption for user in ours is higher than Li et al.'s [24] without outsourcing. Besides, our scheme achieves a similar time-efficiency for decryption for user with outsourcing as Li et al.'s [24]. Moreover, our scheme provides the verifiable property of outsourcing while the scheme in Li et al.'s [24] fails. For the communication overhead, our scheme and Li et al.'s [24] also have an approximate size in terms of the ciphertext size and the transformed ciphertext size. Besides, both our scheme and Li et al.'s [24] scheme feature the property of revocation by incorporating the ciphertext update and key update function such that any legitimate users in group can access the data from other data owners in the community. Moreover, revoked users cannot read the encrypted data again even if they intend to collude their attributes with other legitimate group users or revoked group users.

Quantitative analyses for ours and Li's [24] are given as follows. Here we use the experimental results in [35] on MICA2. The experiment in Shim et al.'s [35, 36] is based on a super-singular curve $z^2 + z = x^3 + x$. Table 3 shows the sizes

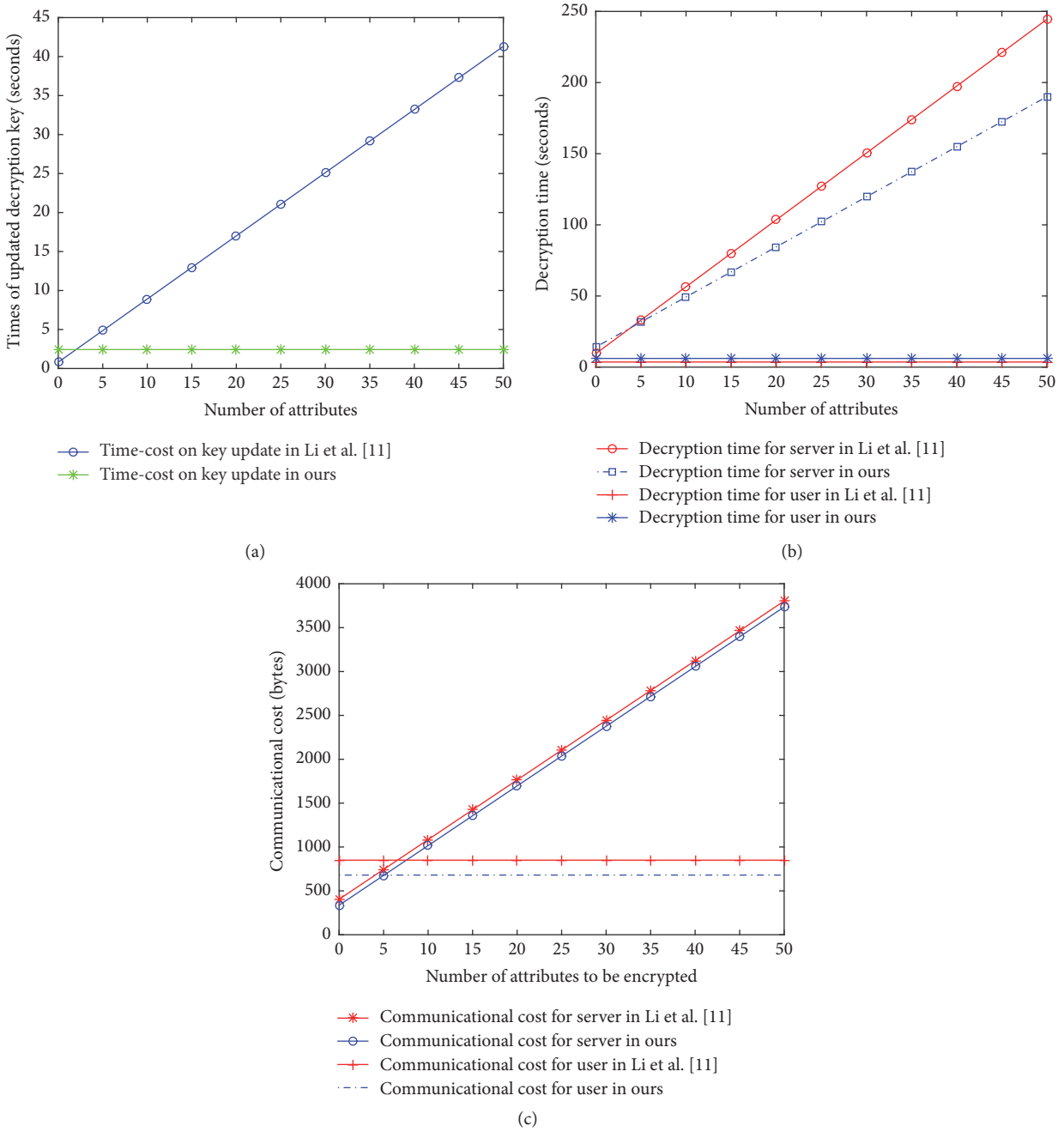

FIGURE 3: (a) Comparison of the time-costs for key update. (b) Comparison of the decryption time-costs. (c) Communicational cost comparison of the ciphertext.

as well as exponentiation and pairing computational time-cost of the elements at an 80-bit security level. Each value in Table 3 was measured based on pabe toolkit and Pairing-Based Cryptography (PBC) library [37] using an Intel(R) Core(TM) i5-4460 CPU @3.2 GHz and 4 G ROM running Windows XP system and VC++ 6.0. From Table 3, the time-costs to calculate a pairing, an exponentiation in $\mathbb{G}$, and an exponentiation in $\mathbb{G}_T$ need to take 1.9 s, 0.81 s, and 0.9 s. The computational times in ours are 2.43 s, $(3.52l + 14.92)$ s, and 6.12 s in terms of user's key updating, decryption for server, and decryption for user. However, the values in Li's [24] are $(0.81l + 0.81)$ s, $(4.7l + 9.5)$ s, and 3.6 s. For communication cost, assuming $|gid| = 80$ bits, according to [35], the element in group $\mathbb{G}$ and group $\mathbb{G}_T$ can be reduced to 34 bytes and 136 bytes by accessing standard compression technique [35]. Therefore, the communication cost in Li et al.'s [24] and ours can be denoted as $(2 * l + 4) * 34 + 136 + 2 * 32 = (64l + 336)$ bytes and $5 * 136 = 618$ bytes and $(2l + 6) * 34 + 136 + 2 * 32 = (64l + 404)$ bytes and $6 * 136 + 34 = 850$ bytes, respectively.

From Figure 3(a), we could find that the efficiency of the user's decryption key updating in our scheme obviously outperforms Li et al.'s [24]. As depicted in Figure 3(b), the

Table 3: Sizes and time-costs.

$\lvert\mathbb{Z}_p^*\rvert$	$\lvert\mathbb{G}\rvert$	$\lvert\mathbb{G}_T\rvert$	E_G	E_{G_T}	P
Byte	34 bytes	136 bytes	0.81 s	0.9 s	1.9 s

performance in ours is better than Li et al.'s [24] without outsourcing. Importantly, our scheme has a similar efficiency to Li et al.'s [24] with achieving the feature of outsourcing. As illustrated in Figure 3(c), communication cost under the same condition in ours and Li et al.'s [24] has an approximate size performance.

8. Conclusion

In our paper, we suggested a secure and efficient attribute-based encryption scheme applied to construct an online healthcare social network. With our proposed ABE scheme, we enabled supporting dynamic social group securely and efficiently; when users (i.e., mobile patients or mobile physicians) were revoked from the social group, after updating ciphertext and secret keys with the aid of PHI cloud storage server, revoked users could not access the cloud again even if they colluded their attributes with other legitimate social group users. Moreover, a lightweight decryption efficiency could be achieved by employing the technique of verifiable outsourcing; thus mobile patients or mobile physicians could effectively share personal health information or medical treatment experience with other users. Finally, the security proof and experiment simulations illustrated that our scheme was selective IND-CPA secure and practical.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This work was supported in part by the National Science Foundation of China (no. 61370026), the National High Technology Research and Development Program of China (no. 2015AA016007), the Sichuan Key Technology Support Program (no. 2014GZ0106), Science Technology Project of Guangdong Province (no. 2016A010101002), and Fundamental Research Funds for the Central Universities under Grant ZYGX2016J091. It was also supported by the Major International (Regional) Joint Research Project of China National Science Foundation (no. 61520106007), Fundamental Research Funds for the Central Universities under Grant ZYGX2015J072, and the Sichuan Science Technology Support Plan Program (no. 2016GZ0065).

References

[1] J. Su, D. Cao, B. Zhao, X. Wang, and I. You, "EPASS: an expressive attribute-based signature scheme with privacy and an unforgeability guarantee for the Internet of Things," *Future Generation Computer Systems*, vol. 33, pp. 11–18, 2014.

[2] L. Guo, C. Zhang, J. Sun, and Y. Fang, "A privacy-preserving attribute-based authentication system for mobile health networks," *IEEE Transactions on Mobile Computing*, vol. 13, no. 9, pp. 1927–1941, 2014.

[3] J. Zhou, Z. Cao, X. Dong, X. Lin, and A. Vasilakos, "Securing m-healthcare social networks: challenges, countermeasures and future directions," *IEEE Wireless Communications*, vol. 20, no. 4, pp. 12–21, 2013.

[4] K. Zhang, X. Liang, R. Lu, and X. Shen, "Sybil attacks and their defenses in the internet of things," *IEEE Internet of Things Journal*, vol. 1, no. 5, pp. 372–383, 2014.

[5] C. Perera, C. H. Liu, and S. Jayawardena, "The emerging internet of things marketplace from an industrial perspective: a survey," *IEEE Transactions on Emerging Topics in Computing*, vol. 3, no. 4, pp. 585–598, 2015.

[6] S. Ray, J. Park, and S. Bhunia, "Wearables, implants, and internet of things: the technology needs in the evolving landscape," *IEEE Transactions on Multi-Scale Computing Systems*, vol. 2, no. 2, pp. 123–128, 2016.

[7] V. Goyal, O. Pandey, A. Sahai, and B. Waters, "Attribute-based encryption for fine-grained access control of encrypted data," in *Proceedings of the 13th ACM Conference on Computer and Communications Security (CCS '06)*, pp. 89–98, November 2006.

[8] J. Bethencourt, A. Sahai, and B. Waters, "Ciphertext-policy attribute-based encryption," in *Proceedings of the IEEE Symposium on Security and Privacy (SP '07)*, pp. 321–334, May 2007.

[9] D. Chen, N. Zhang, Z. Qin et al., "S2M: a lightweight acoustic fingerprints based wireless device authentication protocol," *IEEE Internet of Things Journal*, vol. 4, no. 1, pp. 88–100, 2017.

[10] D. Chen, Z. Qin, X. Mao, P. Yang, Z. Qin, and R. Wang, "SmokeGrenade: an efficient key generation protocol with artificial interference," *IEEE Transactions on Information Forensics and Security*, vol. 8, no. 11, pp. 1731–1745, 2013.

[11] A. Sahai and B. Waters, "Fuzzy identity-based encryption," in *Advances in Cryptology—EUROCRYPT 2005*, vol. 3494 of *Lecture Notes in Computer Science*, pp. 457–473, Springer, Berlin, Germany, 2005.

[12] R. Ostrovsky, A. Sahai, and B. Waters, "Attribute-based encryption with non-monotonic access structures," in *Proceedings of the 14th ACM Conference on Computer and Communications Security (CCS '07)*, pp. 195–203, November 2007.

[13] A. Lewko, T. Okamoto, A. Sahai, K. Takashima, and B. Waters, "Fully secure functional encryption: attribute-based encryption and (hierarchical) inner product encryption," in *Advances in Cryptology—EUROCRYPT 2010*, vol. 6110 of *Lecture Notes in Computer Science*, pp. 62–91, Springer, Berlin, Germany, 2010.

[14] T. Okamoto and K. Takashima, "Fully secure functional encryption with general relations from the decisional linear assumption," in *Advances in Cryptology—CRYPTO 2010*, vol. 6223 of *Lecture Notes in Computer Science*, pp. 191–208, Springer, Berlin, Germany, 2010.

[15] B. Waters, "Ciphertext-policy attribute-based encryption: an expressive, efficient, and provably secure realization," in *Proceedings of the 14th International Conference on Practice and Theory in Public Key Cryptography (PKC '11)*, vol. 6571 of *Lecture Notes in Computer Science*, pp. 53–70, Springer, Heidelberg, Germany.

[16] L. Cheung and C. Newport, "Provably secure ciphertext policy ABE," in *Proceedings of the 14th ACM Conference on Computer and Communications Security (CCS '07)*, pp. 456–465, November 2007.

[17] M. Green, S. Hohenberger, and B. Waters, "Outsourcing the decryption of abe ciphertexts," in *Proceedings of the USENIX Security Symposium '11*, vol. 2011.

[18] J. Li, X. Y. Huang, J. W. Li, X. F. Chen, and Y. Xiang, "Securely outsourcing attribute-based encryption with checkability," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 8, pp. 2201–2210, 2014.

[19] J.-Z. Lai, R. H. Deng, C. Guan, and J. Weng, "Attribute-based encryption with verifiable outsourced decryption," *IEEE Transactions on Information Forensics and Security*, vol. 8, no. 8, pp. 1343–1354, 2013.

[20] H. Ma, R. Zhang, Z. Wan, Y. Lu, and S. Lin, "Verifiable and exculpable outsourced attribute-based encryption for access control in cloud computing," *IEEE Transactions on Dependable and Secure Computing*, 2016.

[21] A. Boldyreva, V. Goyal, and V. Kumart, "Identity-based encryption with efficient revocation," in *Proceedings of the 15th ACM Conference on Computer and Communications Security (CCS '08)*, pp. 417–426, October 2008.

[22] S. Rafaeli and D. Hutchison, "A Survey of Key Management for Secure Group Communication," *ACM Computing Surveys*, vol. 35, no. 3, pp. 309–329, 2003.

[23] N. Attrapadung and H. Imai, "Conjunctive broadcast and attribute-based encryption," in *Proceedings of the 3rd International Conference on Pairing-Based Cryptography (Pairing '09)*, vol. 5671 of *Lecture Notes in Computer Science*, pp. 248–265, Springer, Berlin, Germany, 2009.

[24] J. Li, W. Yao, Y. Zhang, H. Qian, and J. Han, "Flexible and fine-grained attribute-based data storage in cloud computing," *IEEE Transactions on Services Computing (TSC)*, 2016.

[25] S. Yu, C. Wang, K. Ren, and W. Lou, "Attribute based data sharing with attribute revocation," in *Proceedings of the 5th ACM Symposium on Information, Computer and Communication Security, (ASIACCS '10)*, pp. 261–270, April 2010.

[26] J. Hur, "Improving security and efficiency in attribute-based data sharing," *IEEE Transactions on Knowledge and Data Engineering*, vol. 25, no. 10, pp. 2271–2282, 2013.

[27] L. Chen, Z. Cao, R. Lu, X. Liang, and X. Shen, "EPF: An event-aided packet forwarding protocol for privacy-preserving mobile healthcare social networks," in *Proceedings of the 54th Annual IEEE Global Telecommunications Conference: "Energizing Global Communications"*, vol. 2011, pp. 1–5, usa, December 2011.

[28] N. Zhang, N. Lu, N. Cheng, J. W. Mark, and X. S. Shen, "Cooperative spectrum access towards secure information transfer for CRNs," *IEEE Journal on Selected Areas in Communications*, vol. 31, no. 11, pp. 2453–2464, 2013.

[29] J. Zhou, Z. Cao, X. Dong, N. Xiong, and A. V. Vasilakos, "4S: a secure and privacy-preserving key management scheme for cloud-assisted wireless body area network in m-healthcare social networks," *Information Sciences*, vol. 314, pp. 255–276, 2015.

[30] S. Jiang, X. Zhu, and L. Wang, "EPPS: efficient and privacy-preserving personal health information sharing in mobile healthcare social networks," *Sensors*, vol. 15, no. 9, pp. 22419–22438, 2015.

[31] H. Krawczyk, "Cryptographic extraction and key derivation: the HKDF scheme," in *Advances in Cryptology—CRYPTO 2010*, vol. 6223 of *Lecture Notes in Computer Science*, pp. 631–648, Springer, Berlin, Germany, 2010.

[32] H. Xiong, "Cost-effective scalable and anonymous certificateless remote authentication protocol," *IEEE Transactions on Information Forensics and Security*, vol. 9, no. 12, pp. 2327–2339, 2014.

[33] H. Xiong and Z. Qin, "Revocable and scalable certificateless remote authentication protocol with anonymity for wireless body area networks," *IEEE Transactions on Information Forensics and Security*, vol. 10, no. 7, pp. 1442–1455, 2015.

[34] N. Zhang, N. Cheng, N. Lu, X. Zhang, J. W. Mark, and X. Shen, "Partner selection and incentive mechanism for physical layer security," *IEEE Transactions on Wireless Communications*, vol. 14, no. 8, pp. 4265–4276, 2015.

[35] K.-A. Shim, Y.-R. Lee, and C.-M. Park, "EIBAS: an efficient identity-based broadcast authentication scheme in wireless sensor networks," *Ad Hoc Networks*, vol. 11, no. 1, pp. 182–189, 2013.

[36] N. Gura, A. Patel, A. Wander et al., "Comparing elliptic curve cryptography and RSA on 8-bit CPUs," in *Proceedings of 6th International Workshop on Cryptographic Hardware and Embedded Systems*, vol. 3156 of *Lecture Notes in Computer Science*, pp. 119–132, Springer, 2004.

[37] B. Lynn, "The stanford pairing based crypto library," http://crypto.stanford.edu/pbc/.