International Journal on Applications in Information and Communication Engineering
Volume 4: Issue 2: May 2018, pp 41 - 54. www.aetsjournal.com ISSN (Online) : 2394 - 6237

AN EFFICIENT SECURE DATA SHARING FOR MOBILE COMPUTING

A.THAMIZHEZHILAN , K.NANDHINI , K.ASHWINI , S.SATHISH , DHIVYA.D , M.MAKURU

Abstract— With the popularity of cloud computing, mobile devices can store/retrieve personal data from anywhere at any time. Consequently, the data security problem in mobile cloud becomes more and more severe and prevents further development of mobile cloud. There are substantial studies that have been conducted to improve the cloud security. However, most of them are not applicable for mobile cloud since mobile devices only have limited computing resources and power. Solutions with low computational overhead are in great need for mobile cloud applications. In this paper, we propose a lightweight data sharing scheme (LDSS) for mobile cloud computing. It adopts CP-ABE, an access control technology used in normal cloud environment, but changes the structure of access control tree to make it suitable for mobile cloud environments. LDSS moves a large portion of the computational intensive access control tree transformation in CP-ABE from mobile devices to external proxy servers. Furthermore, to reduce the user revocation cost, it introduces attribute description fields to implement lazy-revocation, which is a thorny issue in program based CP-ABE systems. The experimental results show that LDSS can effectively reduce the overhead on the mobile device side when users are sharing data in mobile cloud environments.

Keywords— mobile cloud computing, data encryption, access control, user revocation .

I. INTRODUCTION

With the development of cloud computing and the popularity of smart mobile devices, people are gradually getting accustomed to a new era of data sharing model in which the data is stored on the cloud and the mobile devices are used to store/retrieve the data from the cloud. Typically, mobile devices only have limited storage space and computing power. On the contrary, the cloud has enormous amount of resources. In such a scenario, to achieve the satisfactory performance, it is essential to use the resources provided by the cloudservice provider (CSP) to store and share the data.

A.Thamizhezhilan, Final Year CSE, Meenakshi Ramaswamy Engineering College, Thathanur, Tamilnadu, India .
K.Nandhini, Final Year CSE, Meenakshi Ramaswamy Engineering College, Thathanur, Tamilnadu, India .
K.Ashwini, Final Year CSE, Meenakshi Ramaswamy Engineering College, Thathanur, Tamilnadu, India .
S.Sathish, Final Year CSE, Meenakshi Ramaswamy Engineering College, Thathanur, Tamilnadu, India .
Dhivya.D Final Year CSE, Meenakshi Ramaswamy Engineering College, Thathanur, Tamilnadu, India .
M.Makuru , HOD/CSE, Meenakshi Ramaswamy Engineering College, Thathanur, Tamilnadu, India .

Nowadays, various cloud mobile applications have been widely used. In these applications, people (data owners) can upload their photos, videos, documents and other files to the cloud and share these data with other people (data users) they like to share. CSPs also provide data management functionality for data owners. Since personal data files are sensitive, data owners are allowed to choose whether to make their data files public or can only be shared with specific data users. Clearly, data privacy of the personal sensitive data is a big concern for many data owners.

The state-of-the-art privilege management/access control mechanisms provided by the CSP are either not sufficient or not very convenient. They cannot meet all the requirements of data owners. First, when people upload their data files onto the cloud, they are leaving the data in a place where is out of their control, and the CSP may spy on user data for its commercial interests and/or other reasons. Second, people have to send password to each data user if they only want to share the encrypted data with certain users, which is very cumbersome. To simplify the privilege management, the data owner can divide data users into different groups and send password to the groups which they want to share the data. However, this approach requires fine-grained access control. In both cases, password management is a big issue.

Apparently, to solve the above problems, personal sensitive data should be encrypted before uploaded onto the cloud so that the data is secure against the CSP. However, the data encryption brings new problems. How to provide efficient access control mechanism on ciphertext decryption so that only the authorized users can access the plaintext data is challenging. In addition, system must offer data owners effective user privilege management capability, so they can grant/revoke data access privileges easily on the data users. There have been substantial researches on the issue of data access control over ciphertext. In these researches, they have the following common assumptions. First, the CSP is considered honest and curious. Second, all the sensitive data are encrypted before uploaded to the Cloud. Third, user authorization on certain data is achieved through encryption/decryption key distribution. In general, we can divide these approaches into four categories: simple ciphertext access control, hierarchical access control, access control based on fully homomorphic encryption [1][2] and access control based on attribute-based encryption (ABE). All these proposals are designed for non-mobile cloud environment.

They consume large amount of storage and computation resources, which are not available for mobile devices. According to the experimental results in [26], the basic ABE operations take much longer time on mobile devices than laptop or desktop computers. It is at least 27 times longer to execute on a smart phone than a personal computer (PC). This means that an encryption operation which takes one minute on a PC will take about half an hour to finish on a mobile device. Furthermore, current solutions don't solve the user privilege change problem very well. Such an operation could result in very high revocation cost. This is no proper solution which can effectively solve the secure data sharing problem in mobile cloud. As the mobile cloud becomes more and more popular, providing an efficient secure data sharing mechanism in mobile cloud is in urgent need.

To address this issue, in this paper, we propose a Lightweight Data Sharing Scheme (LDSS) for mobile cloud computing environment.

The main contributions of LDSS are as follows:

(1) We design an algorithm called LDSS-CP-ABE based on Attribute-Based Encryption (ABE) method to offer efficient access control over ciphertext.

(2) We use proxy servers for encryption and decryption operations. In our approach, computational intensive operations in ABE are conducted on proxy servers, which greatly reduce the computational overhead on client side mobile devices. Meanwhile, in LDSS-CP-ABE, in order to maintain data privacy, a version attribute is also added to the access structure. The decryption key format is modified so that it can be sent to the proxy servers in a secure way.

(3) We introduce lazy re-encryption and description field of attributes to reduce the revocation overhead when dealing with the user revocation problem.

(4) Finally, we implement a data sharing prototype framework based on LDSS. The experiments show that LDSS can greatly reduce the overhead on the client side, which only introduces a minimal additional cost on the server side. Such an approach is beneficial to implement a realistic data sharing security scheme on mobile devices. The results also show that LDSS has better performance compared to the existing ABE based access control schemes over ciphertext.

The rest of this paper is organized as follows. Section 2 presents some fundamental concepts in secure mobile cloud data sharing and the security premise. Section 3 gives the detailed design of LDSS. Section 4 and 5 give the safety assessment and performance evaluation, respectively. Section 6 presents related works. Finally, Section 7 concludes our work with the future work.

II. PRELIMINARIES AND ASSUMPTIONS

In this section, we first briefly present the technique preliminaries closely related to LDSS, and then present the system model and some security assumptions in LDSS.

A. Preliminary Techniques

1) Bilinear Pairing

Define a function e as follows:

$e : G_0 * G_0 = G_1$

In this function, both G_0 and G_1cyclic groups of the prime order p.

Assume that g is a generator of G_0 , Zp is a finite field.Then e is a bilinear pairing if e has the following Properties .

1) Bilinear:

$$\forall u, v \in G_0, \forall a, b \in Z_p,\ e(u^a, v^b) = e(u, v)^{ab}$$

2) Non-degeneracy: e(g, g) is a member of G_1 if g is a member of G_0 .

$$\forall u, v \in G$$

3) Computability:$_0$, e(u, v) can be calculated.

In our implementation, we usually take G_0 as a group consisting points on an elliptic curve, G_1 as a multiplicative subgroup of a finite field, e as a Weil or the Tate pairing based on an elliptic curve over a finite field. Further descriptions on how these parameters are defined and generated can be found in [28].

2) Attribute-Based Encryption

Attribute-based encryption (ABE) is proposed by Sahai and Waters [29]. It is derived from the Identity-Based Encryption (IBE) and is particularly suitable for one-to- many data sharing scenarios in a distributed and open cloud environment. Attribute-based encryption is divided into two categories: one is the Ciphertext-Policy Attribute Based Encryption (CP-ABE), in which the access control policy is embedded into ciphertext; the other one is Key- Policy Attribute Based Encryption (KP-ABE), in which the access control policy is embedded in the user's key attributes. In real applications, CP-ABE is more suitable since it resembles role-based access control. In CP-ABE, the data owner designs the access control policy and assigns attributes to data users. A user can decrypt the data properly if the user's attributes satisfy the access control policy.

3) Secret Sharing Scheme

Shamir secret sharing scheme [30] is used to protect secret information. It can be explained as below.

Assume that p is a prime number, the secret information to share is $k \in K = Z_p$. Divide k into n pieces through the following steps:

(1) Randomly select one (t-1)-order polynomial $h(x) = a_{t-1} x^{t-1} + ...+ a_1 x + a_0 \in Z_p [x]$, and let $a_0 = k$.

(2) Select n non-zero and distinct elements Xi from Zp, calculate $y_i = h(x_i),1 \le i \le n$.

(3) Distribute y_i $(1 \le i \le n)$ as shares and publish the corresponding $x_1 , x_2 ,...,x_n$.

The process to reconstruct $h(x)$ out of t random shares through the Lagrange polynomial interpolation is as follows:

$$h(x)=\sum_{s=1}^{t} y_{i_s} \prod_{j=1, j\neq s}^{t}(x-x_{i_j})/(x_{i_s}-x_{i_j})$$

All these operations are done on *Zp*, namely, they are all p-mode operations.

After obtaining $h(x)$, we can get the secret k □ a_0 □ $h(0)$:

$$k=h(0)=\sum_{s=1} y_{i_s} \prod_{j=1, j\neq s} \frac{\square - x_{i_j}}{x_{i_s} - x_{i_j}}$$

Since x_1 , x_2 ,...,x_n is public, we can get Lagrange coefficients in advance:

$$\lambda_s=\prod_{j=1, j\neq s}^{t}\frac{-x_{i_j}}{x_{i_s}-x_{i_j}}$$

Thus, the formula to recover the secret k can be put in a simpler way:

$$k=\sum_{s=1}^{t}\lambda_s y_{i_s}$$

B. Security Assumptions

1) Semi-trusted Server

LDSS is designed under the same assumptions proposed in 0 that the CSP is honest but curious, which means that the CSP will faithfully execute the operations requested by users, but it will peek on what users have stored in the cloud. The CSP will faithfully store users' data, undertake an initial access control, update data according to users' requests. However, CSP may do malicious actions such as collusion with users to get the data in plain text.

In LDSS, proxy encryption server and proxy decryption server are introduced to assist users to encrypt and decrypt data so that user-side overhead can be minimized. In essence, proxy servers are also machines in the cloud. Thus, we consider that they are honest but curious just as the CSP.

2) Trusted Authority

In this paper, to make LDSS feasible in practice, a trusted authority (TA) is introduced. It is responsible of generating public and private keys, and distributing attribute keys to users. With this mechanism, users can share and access data without being aware of the encryption and decryption operations.

We assume TA is entirely credible, and a trusted channel exists between the TA and every user. The fact that a trusted channel exists doesn't mean that the data can be shared through the trusted channel, for the data can be in a large amount. TA is only used to transfer keys (in a small amount) securely between users. In addition, it's requested that TA is online all the time because data users may access data at any time and need TA to update attribute keys.

3) Lazy Re-encryption

In ciphertext access control, data needs to be re-encrypted when some users' access privileges to the data are revoked. However, frequent re-encryption brings heavy computational overhead, and the accessed plaintext data may already be stored on these data users. Therefore, this paper adopts the lazy re-encryption method proposed in [3]. With lazy re-encryption, when a user's access privilege is revoked, data is not re-encrypted until the data owner updates the data.

In our approach, when the data owner revokes a user's privilege, the file of the access control policy that contains these attributes will be marked. Later, when the data owner updates this file, it first checks the mark to see if it has been marked as revoked. If that is the case, this file will be re-encrypted.

III. OUR PROPOSED MECHANISM

In this section, we describe the LDSS system design. First, we give the overview of LDSS, and then we present LDSS-CP-ABE algorithm and system operations, which are the base of LDSS algorithm. Finally, we describe LDSS in details.

1) Overview

We propose LDSS, a framework of lightweight data-sharing scheme in mobile cloud (see Fig. 1). It has the following six components.

(1) Data Owner (DO): DO uploads data to the mobile cloud and share it with friends. DO determines the access control policies.
(2) Data User (DU): DU retrieves data from the mobile cloud.
(3) Trust Authority (TA): TA is responsible for generating and distributing attribute keys.
(4) Encryption Service Provider (ESP): ESP provides data encryption operations for DO.
(5) Decryption Service Provider (DSP): DSP provides data decryption operations for DU.
(6) Cloud Service Provider (CSP): CSP stores the data for DO. It faithfully executes the operations requested by DO, while it may peek over data that DO has stored in the cloud.

As shown in Fig. 1, a DO sends data to the cloud. Since the cloud is not credible, data has to be encrypted before it is uploaded. The DO defines access control policy in the form of access control tree (refer to Definition 2 in Section 3.2) on data files to assign which attributes a DU should obtain if he wants to access a certain data file. In LDSS, data files are all encrypted with the symmetric encryption mechanism, and the symmetric key for data encryption is also encrypted using attribute based encryption (ABE). The access control policy is embedded in the ciphertext of the symmetric key. Only a DU who obtains attribute keys that satisfy the access control policy can decrypt the ciphertext and retrieve the symmetric key. As the encryption and decryption are both computationally intensive, they introduce heavy burden for mobile users. To relieve the overhead on the client side mobile devices, encryption service provider (ESP) and decryption service provider (DSP) are used. Both the encryption service provider and the decryption service provider are also semi-trusted. We modify the traditional CP-ABE algorithm and design an LDSS-CP-ABE algorithm to ensure the data privacy when outsourcing computational tasks to ESP and DSP.

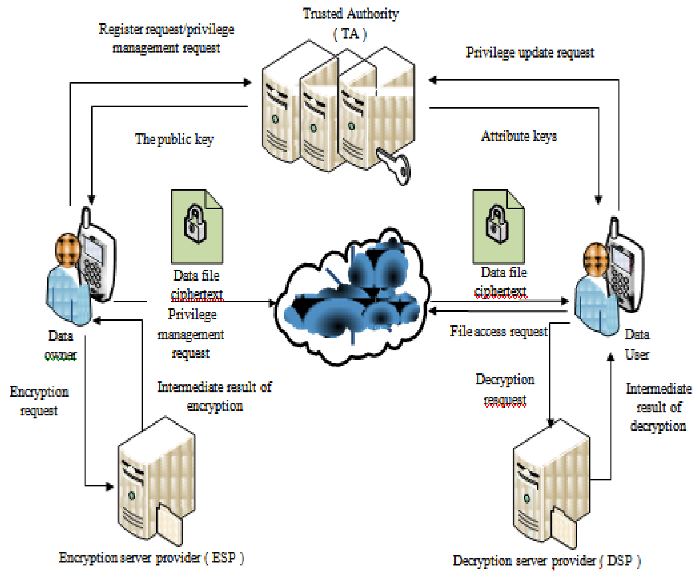
Trusted Authority
(TA)
Register request/privilege management request
Privilege update request
The public key
Attribute keys
Data file ciphertext
Data file ciphertext
Privilege management request
File access request
Data owner
Data User
Encryption request
Intermediate result of encryption
Decryption resquest
Intermediate result of decryption
Encryption server provider (ESP)
Decryption server provider (DSP)

Figure 1. A lightweight data-sharing scheme (LDSS) framework.

2) LDSS-CP-ABE Algorithm

To better illustrate LDSS-CP-ABE algorithm, we first define the following terms.

Definition 1: Attribute

An attribute defines the access privilege for a certain data file. Attributes are assigned to data users by data owners. A data user can have multiple attributes corresponding to multiple data files. A data owner can define a set of attributes for its data files. The data accesses are managed by access control policy specified by data owners.

Let *A* = {*A*1, *A*2, *A*3, ..., *A*n} be the set of attributes for a data owner. Each data user *u* also has a set of attributes *A*u, which is a non-empty subset of *A*, namely *A*u ⊆ {*A*1, *A*2, *A*3, ..., *A*n}.

For example, assume *A* is {*relatives, colleagues, classmates, friends, teachers, peers, Hubei, Beijing, Shanghai, degree of intimacy*}. A data user's subset *A*u could be {*friend, Hubei, degree of intimacy*=3}. The access control policy for a data file *M* could be: ((*friends and degree of intimacy > 1 and Hubei*) *or* (*relatives and peers*)), which means a data user cannot access *M* unless these conditions are met.

Definition 2: Access Control Tree

Access control tree is the specific expression of access control policies, in which the leaf nodes are attributes, and non-leaf nodes are relational operators such as *and, or, n of m threshold*. Each node in an access control tree represents a secret, and the secret of a top node can be split into multiple secrets by secret sharing scheme and distribute to lower level nodes. Correspondingly, if we know the secrets of leaf nodes, we can deduce the secret of non-leaf nodes by calculating recursively from bottom to top.

Fig. 2 shows the access control tree for the example described in Definition 1.

Definition 3: Version Attribute.

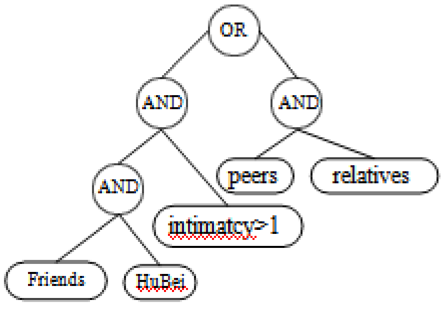

Figure 2. The access control tree.

Version attribute is introduced in LDSS-CP-ABE algorithm to ensure security. It is an addition to the original access control tree, forming a new root node of *and*. We have the following definitions.

T: The new access tree with version attributes.

S: The secret related to the root of *T*.

*T*a, *R*a, *S*a: *T*a is the initial access control tree and the left subtree of *T*. *R*a is the root of *T*a. *S*a is the secret related to *R*a.

*T*v, *R*v, *S*v: *T*v is the right subtree of *T* and contains only one node, which represents the version attribute *R*v. *S*v is the secret related to *R*v.

Both *S*a and *S*v are derived from *S* based on the secret sharing scheme.

For the example described in Definition 1, the access control tree with version attributes is shown in Figure 3.

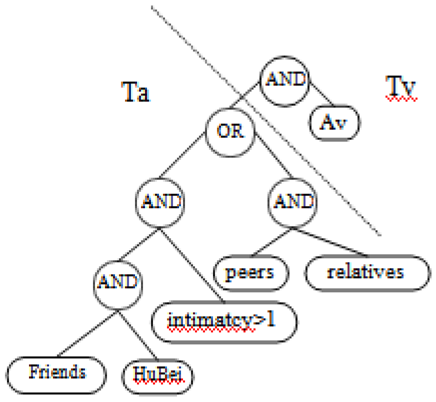

Figure 3. The access control tree with version attributes.

LDSS-CP-ABE algorithm is designed using above definitions. It includes four sub-functions:

Setup(*A*, *V*): Generate the master key *MK*, the public key *PK* based on attribute set *A* of the Data Owner and the version attribute *V* .

KeyGen(*A*u, *MK*): Generate attribute keys *SK*u for a data user *U* based on his attribute set *A*u and the master key *MK*.

Encryption(*K, PK, T*): Generate the ciphertext *CT* based on the symmetric key *K*, public key *PK* and access control tree *T* .

Decryption(*CT,T,SK*u): Decrypt the ciphertext *CT* using the access control tree *T* and the attribute keys *SK*u .

We explain all of these functions specifically below.

First, function Setup() is called by the trusted third party (TA) to generate the master key and the public key. The master key is used to generate attribute keys and the public key is used to encrypt data files. The process of this function is given in Function 1.

Function 1: Setup()

INPUT: The attribute set A , the version attribute V.
OUTPUT: The master key MK, the public key PK.

1. Construct a p-order bilinear group G_0 of generator g and a bilinear mapping $e : G_0 * G_0 = G_1$.
2. Randomly choose $a,b \in Z_p$ and calculate g^b , $e(g, g)^a$.
3. For each attribute a_i in A, randomly choose $t_i \in Z_p$, and calculate $X_i = g^{t_i}$.
4. For V, randomly choose $t_v \in Z_p$, and calculate $X_v = g^{t_v}$.
5. Return the master key MK and the public key PK, Wherein $MK=\{a,b\}$, $PK=\{ G_0 , g , g^b, e(g,g)^a, \{X_i\}^k{}_{i=1}, X_v\}$.

Second, function KeyGen() is used to generate attribute keys for users, as shown in Function 2.

Function 2: KeyGen()

INPUT: The attribute set A_u, the master key $MK=\{a,b\}$.
OUTPUT: Attribute keys associated with A_u.

1. Randomly choose a parameter $r \in Z_p$, and calculate $SK_r = g^{(a+r)/b}$.
2. For each attribute a_i in A_u, randomly choose $r_i \in Z_p$, and calculate $SK_a = \{g^{r_i} , g^r \cdot X_i^{r_i} \}_{i=1}^{j}$.
3. For V, randomly choose $r_v \in Z_p$, and calculate $SK_v = \{g^{r_v} , g^r \cdot X_v^{r_v} \}$.
4. Return $SK_u = \{SK_r , SK_a , SK_v \}$.

Third, function Encryption( ) is used to encrypt the symmetric key. DO executes function Encryption( ) and gets *S*a in step 2, then sends it to ESP with *T*a. ESP takes *T*a and *S*a as input and deduces s_i for each leaf node, calculating $CT_a = \{g^{S_i} , g^r \cdot X_i^{S_i} \}^{num}$. Then DO gets *CTa* from ESP and has the final ciphertext *CT*. The function Encryption( ) is shown in Function 3.

Function 3: Encryption()

INPUT: The symmetric key K, public key PK, access control tree T (including the left subtree T_a, right subtree T_v, and left subtree has *num* leaf nodes).
OUTPUT: The ciphertext CT.

1. Randomly choose $S \in Z_p$ as the secret of T, and calculate $CT_k=\{g^{bS}, K \cdot e(g,g)^{aS}\}$.
2. Get the value of the two children (namely S_a, S_v) of the root node according to the access control tree.
3. Calculate $CT_v=\{ g^{S_v} , g^r \cdot X_v^{S_v} \}$.
4. Return $CT=\{ CT_k, CT_a, CT_v \}$.

Fourth, DU uses Decryption( ) to decrypt the symmetric key *K*. DU first executes step 1 to get *SK*u' and sends it to DSP with *CT*. DSP executes step 2 to step 3 to get DecryptLeaf( ), which will be sent to DU. Then DU executes the last step to get the plaintext of *K*. The function Decryption( ) is shown in Function 4.

Function 4: Decryption()

INPUT: Ciphertext CT, the access control tree T (including the left subtree T_a, right subtree T_v, and left subtree has *num* leaf nodes), SK_u (attribute keys of user U).
OUTPUT: The plaintext of K.

1. Randomly choose t, and get $SK_u'=\{ SK_r'= SK_r{}^t, SK_a, SK_v \}$.
2. For every leaf node z of T_a, calculate $DecryptLeaf(CT_a, SK_u', z) = e(g , g)^{q_z(0)}$.
3. For the leaf node in right subtree, calculate $DecryptLeaf(CT_v, SK_u', V) = e(g , g)^{q_v(0)}$.
4. Let $CT_k\text{-}1= g^{bS}$, $CT_k\text{-}2= K \cdot e(g, g)^{aS}$, calculate $K = \dfrac{CT_k-2}{e(CT_k -1, SK_r')^{\frac{1}{t}}/Fx} \quad \dfrac{CT_k-2}{e(CT_k -1, SK_r')^{\frac{1}{t}}/e(g, g)^{rS}}$.

In Function 4, the specific processes of step 2 and step 3 are as follows.

Let *SK*a-*1*= g^{ri}, *SK*a-*2*= $g^r \bullet Xi^{ri}$, (*i* starts from 1 to *num* and *num* is the number of the leaf nodes of the access control tree; let *CT*a-*1*= g^{Si}, *CT*a-*2*=Xi^{Si}; for every leaf node *z* of *T*a, define the following functions:

$$DecryptLeaf(CT_i, SK_u', z) = \frac{e(SKa-2, CTa-1)}{e(SKa-1, CTa-2)} = \frac{e(g^r \cdot Xi^{ri}, g^{Si})}{e(g^{ri}, Xi^{Si})} = e(g, g)^{rSi} = e(g, g)^{q_z(0)}.$$

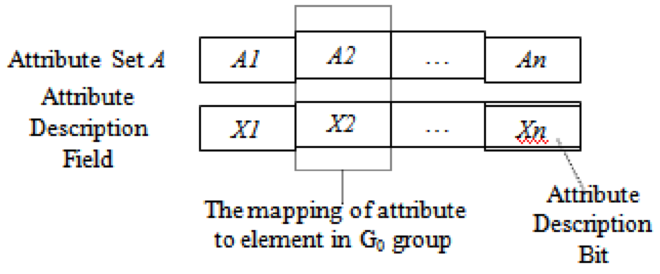
Attribute Set A
A1
A2
...
An
Attribute Description Field
X1
X2
...
Xn
The mapping of attribute to element in G0 group
Attribute Description Bit

Figure 4. The attribute description field of data owner.

Similarly, for the nodes in the right subtree, let $SK_v\text{-}1=g^{rv}$, $SK_v\text{-}2=g^r \bullet X_v^{rv}$, $CT_v\text{-}1= g^{Sv}$, $CT_v\text{-}2=X_v^{Sv}$, then

$$\mathrm{DecryptLeaf}(CT_V,\ SK_u',\ V)= \frac{e(SKv-2, CTv-1)}{e(SKv-1, CTv-2)} = \frac{e(g^r \cdot Xv^{rv}, g^{Sv})}{e(g^{rv}, Xv^{Sv})} = e(g,g)^{rSv} = e(g,g)^{q_v(0)}.$$

The specific process of step 4 is as follows.

For a non-leaf node x, assume that z is a child of x, then F_z = DecryptLeaf(CT_a, SK_u', z)= $e(g, g)^{q_z(0)}$.

Let S_x be the set of x's children, and the size of S_x is k_x, $i = index(x)$, $S'_x = \{index(z) : z \in S_x\}$, according to secret sharing scheme(refer to section 2.1.3), we can get:

$$Fx = \prod_{z\in Sx} Fz^{\Delta_{i,S'x}(0)}$$

$$= \prod_{z\in Sx} (e(g,g)^{r\cdot q_z(0)})^{\Delta_{i,S'x}(0)}$$

$$= \prod_{z\in Sx} (e(g,g)^{r\cdot q_{parent(z)}(index(z))})^{\Delta_{i,S'x}(0)}$$

$$= \prod_{z\in Sx} (e(g,g)^{r\cdot q_x(i)})^{\Delta_{i,S'x}(0)}$$

$$= e(g,g)^{r\cdot q_x(0)}$$

$$= e(g,g)^{rs}.$$

3) Attribute Description Field in LDSS-CP-ABE

Attribute description field is introduced in LDSS for dynamic user privilege management. It keeps access control strategy secret against the cloud.

To better illustrate the attribute description field, we have the following definitions.

Definition 4: Attribute Description Field. Attribute description field is a string of binary bits, which describes attribute information related to DO, DU and data files.

Definition 5: Attribute Description Bit. Attribute description bit is every bit in Attribute description field corresponding to an attribute.

Clearly, attribute description field is composed of several attribute description bits. The size of attribute description field equals to the number of elements in the attribute set A. Each DO defines its own set of attributes. Attribute description fields of different DOs are used to

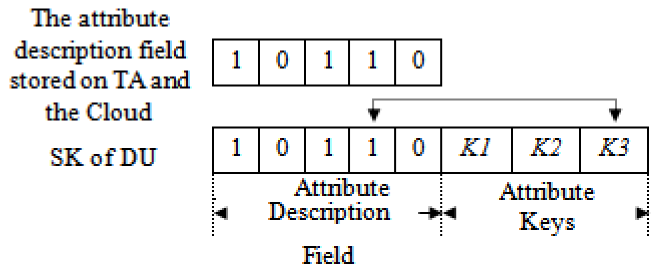
The attribute description field stored on TA and the Cloud
1 0 1 1 0
SK of DU
1 0 1 1 0 K1 K2 K3
Attribute Description Field
Attribute Keys

Figure 5. A sample attribute description field of data user.

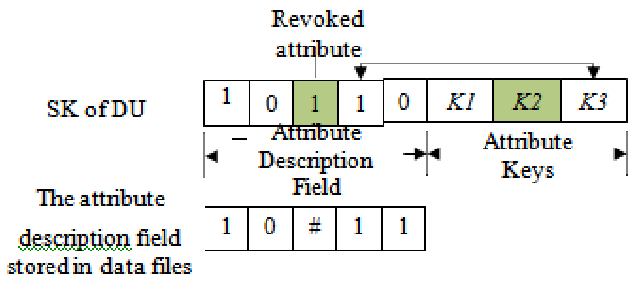
Revoked attribute
SK of DU
1 0 1 1 0 K1 K2 K3
Attribute Description Field
Attribute Keys
The attribute description field stored in data files
1 0 # 1 1

Figure 6. The attribute description field of data files.

There are three kinds of Attribute Description fields, namely, the Attribute Description field of DO, the attribute description field of DU and the attribute description field of data file.

The attribute description field of DO is generated by the TA. When a data owner registered with TA, it sends its own attribute set to TA. TA then generates attribute description field, in which each attribute bit represents a value in G0. TA keeps the attribute description field in the DO-PK/MK-information table. The attribute description field of DO is shown in Fig. 4.

The attribute description field of a data user (DU) is generated by TA and the cloud under the supervision of the data owner. TA and the cloud keep it in contacts- information table. TA and the cloud keep up-to-date information of DU's attribute description fields according to the data owner. Each data user also maintains an attribute description field which may contains out-dated control information. Data users obtain their attribute description fields from TA when TA generates attribute keys for them. The attribute description field is sent together with the attribute keys. In the attribute description field of DU, every bit is either 1 or 0. A 1 denotes that the DU owns the attribute while a 0 denotes the opposite. For example, if the data owner has 5 attributes, a sample attribute description field is shown in Fig. 5.

The attribute description field of data files is stored on DO. It represents which attributes are assigned in data files' access control policy. If an attribute is included in the access control policy, the corresponding bit in the description field is 1, otherwise it's 0. '#' may appear in the attribute description field when an attribute is included in the access control policy and some data users have this attribute revoked. For a data owner who has five attributes, an example of the attribute description field of data files is shown in Fig. 6. control accesses on their own data files, thus they might have different meanings.

Assume the data owner's attribute set is {A, B, C, D, E}, and it has a file of which the access control policy is "A and C and D and E". A contact of the data owner has three attributes: {A, C, D} and C is revoked. Then the description field of this data file is shown in Fig. 6.

To enforce access control, the access control policy should be uploaded to the cloud. It is also described by multiple attribute description bits, which is a combination of 1 and 0. Thus, it can protect the access control policy against the cloud.

4) System Operations of LDSS

LDSS scheme is designed for data sharing in mobile cloud. The whole process of LDSS includes system initialization, file sharing, user authorization, and file access operations. It also has to support attribute revocation and file update operations.

1) System Initialization

In system initialization, Function 1 is executed. The specific process is described as follows.

(1) When the data owner (DO) registers on TA, TA runs the algorithm Setup() to generate a public key PK and a master key MK. PK is sent to DO while MK is kept on TA itself.
(2) DO defines its own attribute set and assigns attributes to its contacts. All these information will be sent to TA and the cloud.
(3) TA and the cloud receive the information and store it.

2) File Sharing

The process of file sharing uses Function 3 to encrypt data files. The specific process is described as follows.

(1) DO selects a file *M* which is to be uploaded and encrypts it using a symmetric cryptographic mechanism (such as AES, 3DES algorithm) with a symmetric key *K*, generating ciphertext *C*.
(2) DO assigns access control policy for *M* and encrypts *K* with the assistance of ESP using Function 3, generating the ciphertext of *K* (*CT*).
(3) DO uploads C, CT and access control policy to the cloud.

3) User Authorization

The process of user authorization executes Function 2 to generate attribute keys for data users. The specific process is described as follows.

(1) DU logins onto the system and sends, an authorization request to TA. The authorization request includes attribute keys (*SK*) which DU already has.
(2) TA accepts the authorization request and checks whether DU has logged on before. If the user hasn't logged on before, go to step (3) , otherwise go to step (4).
(3) TA calls Function 2 to generate attribute keys (*SK*) for DU.
(4) TA compares the attribute description field in the attribute key with the attribute description field stored in database. If they are not match, go to step (5), otherwise go to step (6).

For each inconsistent bit in description field, if it is 1 on data user's side and 0 on TA's side, it indicates that DU's attribute has been revoked, then TA does nothing on this bit. If it is reversed scenario, it indicates that DU has been assigned with a new attribute, then TA generates the corresponding attribute key for DU.

(5) TA checks the version of every attribute key of DU. If it's not the same with the current version, then TA updates the corresponding attribute key for DU.

In the stage of user authorization, TA updates attribute keys for DU according to the attribute description field, which is stored with *SK*. It describes which attributes DU has and their corresponding versions. TA also keeps attribute description field of DU in database. When DO changes the attribute of DU, the attribute description field on the TA side is also updated. Thus, when DU logins on the system, the attribute description field on itself may be different from that of TA. TA has to update the attribute keys for DU according to the attribute description field just as described above.

4) Access Files

When DU requests to access a certain data file, Function 4 is used to decrypt data. The specific process is described as follows:

(1) DU sends a request for data to the cloud.
(2) Cloud receives the request and checks if the DU meets the access requirement. If DU can't meet the requirement, it refuses the request, otherwise it sends the ciphertext to DU.
(3) DU receives the ciphertext, which includes ciphertext of data files and ciphertext of the symmetric key. Then DU executes the Function 4 to decrypt the ciphertext of the symmetric key with the assistance of DSP.
(4) DU uses the symmetric key to decrypt the ciphertext of data files.

5) Privilege Revoked

DO can revoke attributes from a DU. The process is as follows.

(1) DO informs TA and the cloud that one attribute has been revoked from a specific DU.
(2) TA and the cloud update the information of DU in database.
(3) DO marks the corresponding bit of the attribute description field of data files.

This strategy implements the asynchronous processing of attribute revocation and attribute keys update operations. When DO wants to revoke one attribute from a DU, TA only updates the database and doesn't update attribute keys for DU simultaneously.

6) Documentation Updates

As a result of lazy re-encryption, when DO revokes one attribute from a DU, the revoked attribute is not updated. When the data file is updated, if it has one attribute that has been revoked, this attribute should be updated. The specific process is as follows.

(1) DO checks if there is any bit in the description field of data files has been set to '#'.
(2) DO informs TA which attributes should be updated. All the attributes that should be updated form a set is called *Anew*.
(3) TA chooses a new value in *G0* for every attribute in *Anew* to replace the original one, and updates the description field of DO in DO-PK/MK table, changing the corresponding attribute description bit to the new value.
TA sends a new *PK* to DO, and DO uses the new *PK* to encrypt data files.
(4) DO sets the '#' bit of the description field of the corresponding data file to 1.

This operation is critical for lazy re-encryption. If the system updates attributes immediately after the attribute revocation operation, excessive overhead occurs. Taking into account that DU already know the content of a data file after accessing it, there is no need to re-encrypt this data file with a new symmetric key immediately. The DU who has been revoked the access privileges should not be able to access the updated content. In this situation, the system should re-encrypt the data file. Thus, in LDSS, attribute updates are delayed until related data files are updated. In order to decide which attribute should be updated, the corresponding bit in the description field has to be marked as '#'.

IV. SECURITY ANALYSIS

The security assessment is based on the security assumptions we described in Section 3. The possible scenarios that malicious users may expose plaintext to others are not discussed.

1) Security Analysis of LDSS-CP-ABE:

LDSS-CP-ABE algorithm is designed on top of Attribute-Based Encryption (ABE). The security of ABE is based on the bilinear diffie-hellman assumptions.

2) Bileardiffie-hellman assumptions:

When attackers only have a, b, c, $z \in Zp$, there exists no polynomial algorithm that can get the relationship between ($A=g^{a}$, $B=g^{b}$, $C=g^{c}$, $Z=e(g, g)^{ab/c}$) and ($A=g^{a}$, $B=g^{b}$, $C=g^{c}$, $Z=e(g, g)^{z}$). In other words, attackers cannot get $Z=e(g, g)^{z}$ that corresponds to $e(g, g)^{ab/c}$.

The security of CP-ABE is proved in BSW CP-ABE [27] based on above assumptions. Since LDSS-CP-ABE is a variation of the original BSW CP-ABE, the structure of the ciphertext used in LDSS-CP-ABE is similar to that of original BSW CP-ABE, thus the encryption and decryption processes are safe. The difference between our work and BSW CP-ABE is that a version attribute is added to the access control tree. It only changes the structure of the access tree slightly. It contains two sub trees in our work: *Ta* and *Tv*. If a DO chooses a first-order polynomial *q (x)*, and let *S = q(0)*, *S1 = q (1)*, *S2 = q (2)*. The tuple {*S1*, *Ta*} is sent to ESP. According to the secret sharing scheme, even if S1 is exposed to DO, *S2* and *S* are safe.

3) Data Confidentiality against Conspiracy

The data confidentiality is taken into account from two aspects. In LDSS, data are encrypted with a symmetric key. The security of this part is guaranteed by symmetric encryption mechanism. Next, the symmetric key is encrypted by attribute encryption. The security of this part depends on the encryption process.

The security of the core algorithm in the encryption process is proved in the previous section. Here, we

discuss the situation that the symmetric key is safe even if a malicious user, ESP and DSP conspired to get the key. The conspiracy attack can be divided into several kinds, namely conspiracy between different users, DSP and ESP, users and cloud.

First, consider the conspiracy between different users. It can be proven that different users with different attributes cannot combine their attributes to decrypt data files. Since users get different *r* from TA, which is used to generate attribute keys for users, different users with same attributes get different keys. When decrypting data files, only when all the keys are generated from the same *r* can they be combined to decrypt data files, thus effectively preventing the conspiracy between users.

Second, consider the conspiracy between ESP and DSP. ESP gets {*S1* , *Ta*} and PK from DO and TA, and DSP gets SK_u', CT from DU. Combining all these information, ESP and DSP can finally get $e(g, g)^{t(a\Box r) / S}$, $e(g, g)^{rS}$, $e(g, g)^{a}$, which cannot deduce $e(g, g)^{aS}$ thanks to the **bilinear diffie-hellman assumptions**, thus protecting CTk.

Last, consider the conspiracy between the cloud and DU. The cloud may send data packets to whom do not meet the access control policy. However, even if DU illegally obtains ciphertext, it cannot get the plain context since it doesn't have the right attribute keys.

4) Confidentiality of Access Control Policy

The security of access control policy is that no participants could know the specific content of the access control policy except data owners. LDSS introduces attribute description field so that access control policy is described by the corresponding attribute description bit. ESP and the Cloud can only get the relationships between different attribute description bits, but not the specific content of access control strategy, thus protecting the access control strategy.

V. PERFORMANCE EVALUATION

In this section, we evaluate the performance of LDSS in terms of computational and storage overheads, respectively.

1) *Experimental Settings*

To evaluate the efficiency of the proposed solution, we conduct several experiments. The test of LDSS is done on a Core 2 DUO machine, which has 2.0GHz CPU with the Linux operating system (Ubuntu 12.10) installed.

The core algorithm of LDSS takes advantage of the CP-ABE tools developed by Bethencourt et al [15]. It's based on 160-bit elliptic curve group, which derives from the super singular curve $y^{2} = x^{3} \square x$ over a 512-bit finite field. CP-ABE tools have three basic operations, namely exponentiation and pairing on $G0$ and exponentiation on $G1$. These three operations take 4.99ms, 4.98ms and 0.58ms respectively in our experimental environment.

TABLE 1 - COMPUTATIONAL OVERHEAD OF BASIC OPERATIONS OF ABE SCHEMES

Types of Devices	Pairing	Exponentiation	Multiplication
PC	20 ms	5 ms	0.7 ms
Mobile	550 ms	177 ms	26 ms

The cost of access control mechanisms is closely related to the size of access control policy. To reflect closely to the reality, in our experiment, the number of attributes owned by individual users is fixed, and the size of access control policy varies. We assume that the average number of attributes owned by DO is 10, and the number of attributes included in the access policies varies from 1 to 32.

In order to simplify the representation, we define the following symbols:

$|A|$: The number of attributes owned by DO.

$|A_u|$: The number of attributes owned by DU.

$|T_a|$: The number of leaf nodes in the access control tree .

$|T|$: The number of leaf nodes in the access control tree with version attribute, and $|T| = |T_a| +1$.

L_{G0}, L_{G1}, L_z: The size of an element in G_0 group, G_1 group and Z.

T_G_0: The time needed for exponentiation operation in group G_0.

T_G_m: The time needed for multiplication operation in group G_m.

T_G_e: The time needed for pairing operation in group G0.

T_G_1: The time needed for exponentiation operation in group G_1.

2) *Computational Overhead Evaluation*

We first evaluate the computational overhead of LDSS and compare it with existing access control schemes.

A. *Computational Overhead of LDSS*

According to [26], the basic operations of attribute based encryption mechanisms (pairing, exponentiation, multiplication) vary a lot between mobile devices and PCs. The experimental results are shown in Table 1.

It is clear that a single pairing operation, exponentiation operation, multiplication operation take much longer time on mobile devices than on PCs, which is 27, 35 and 38 times of that on PCs. We focus the analysis of computational overhead on pairing operation and exponentiation operation. Other operations that take little time are ignored.

B. *User registration*

The overhead of user registration comes from the function Setup(), which only needs to be executed once and the overhead is on the TA's side. The main overhead of this execution includes one exponentiation operation and one pairing operation on $G0$ and one exponentiation operation on $G1$, namely. The main overhead is: $T\_G0 + T\_Ge + T\_G1$.

TABLE 2 - COMPUTATIONAL OVERHEAD OF DATA SHARING

	Exponentiation on G_0	Exponentiation on G_1	Paring on G_0
ESP	$2\,|T_a|$	0	0
DO	3	1	0

TABLE 3 -COMPUTATIONAL OVERHEAD OF DATA ACCESS

	Exponentiation on G_0	Exponentiation on G_1	Paring on G_0
DSP	0	$|T_a|$	$2\,|T_a|+1$
DO	0	1	0

C. *Data sharing*

The cost of data sharing comes from the execution of the function Encryption(), which is executed every time when sharing data files. The function Encryption() includes exponentiation operation on $G0$ (the number of operations is proportional to the number of attributes included in the access strategy) and one exponentiation operation on $G1$. The cost of this function depends on which one does the encryption operation. Before introducing ESP, the cost is on DO. After the usage of ESP, the cost on DO is reduced to a constant value, and is no longer associated with the number of attributes in access control strategies. The overhead on ESP and DO is shown in Table 2.

D. User authorization

The cost of user authorization comes from function KeyGen(), which is executed the first time a DU tries to read a DO's data. TA executes this function for authorization. It includes exponentiation on $G0$ and multiplication on $G0$, of which the number is proportional to the number of attributes owned by DU . The overhead is: (2 |Au| +1) T_G0 + |Au| T_Gm.

E. Accessing data files

The cost of accessing data files comes from function Decryption( ), which is executed every time a file is accessed. This function includes pairing operations on $G0$, multiplication operations on $G0$ and exponentiation operation on $G1$. The number of these three kinds of operations is all proportional to the number of attributes included in the access strategy. The cost of accessing data files depends on which one does the decryption operation. Before introducing DSP, the overhead is on DU. After the introduction of DSP, the cost on DU is reduced to a constant value. The overhead of decryption is related to the number of attributes involved in the data file and how these attributes are combined. In the worst case, all the attributes keys related to the access control strategy are needed for decryption. In this case, the overhead of ESP and DO is shown in Table 3.

TABLE 4 - COMPUTATIONAL OVERHEAD WITH DIFFERENT CP-ABES

	Bethencourt	BSW CP-ABE	LDSS
Data sharing	$(2\|T_a\|+1)T\_G_0+T\_G_1$	$(4\|T_a\|+1)T\_G_0+T\_G_1$	$3T\_G_0+T\_G_m$
Data access	$(2\|A_u\|+1)T\_G_e$	$(2\|A_u\|+1)T\_G_e$	$T\_G_0+T\_G_m$

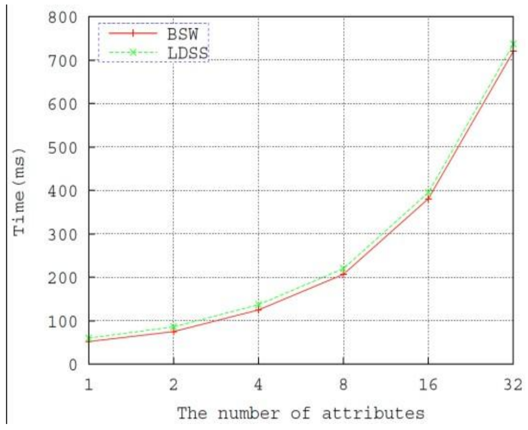

Figure 7. The computational overhead of authorization.

F. User revocation

LDSS uses lazy re-encryption. If there is user revocation operation, TA and the cloud only need to update the contact-attribute-information table. Only when data files are updated should the attributed be updated and data files be re-encrypted. As a result, multiple revocation operations are merged into one, reducing the overall overhead. The cost of data re-encryption is the same with sharing data files. Thus, no further discussion is placed here.

G. Computational Overhead with Different CP-ABE Schemes

DO's overhead in different ABE schemes is shown in Table 4. As shown in Table 4, in existing programs, the overhead on mobile user DU's side is proportional to the number of attributes in access control policy. In LDSS, the overhead is a small constant value.

H. Measurement of Computational Overhead of LDSS

We measure the computational overhead of LDSS through experiments. The results are as follows.

(1) Registration cost

The average registration time for a single user is 50ms.

(2) Authorization cost

The time needed for authorization is proportional to the number of attributes owned by DU. Fig. 7 shows the time needed for user authorization when the number of attributes owned by user is 2,4,8,16,32.

As can be seen in Fig. 7, the time of authorization is proportional to the number of attributes in both BSW CP-ABE [27] and LDSS.

In both scenarios, the authorization time is still lower than 1s when the number of attributes rises to 32. Authorization time in LDSS is just slightly longer because it introduces the version attribute.

(3) The cost of encryption and decryption

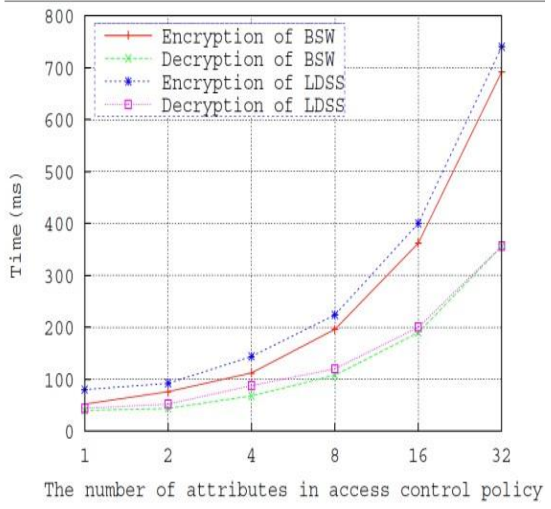

Figure 8. The relationship between encryption and decryption time and the size of access control policy.

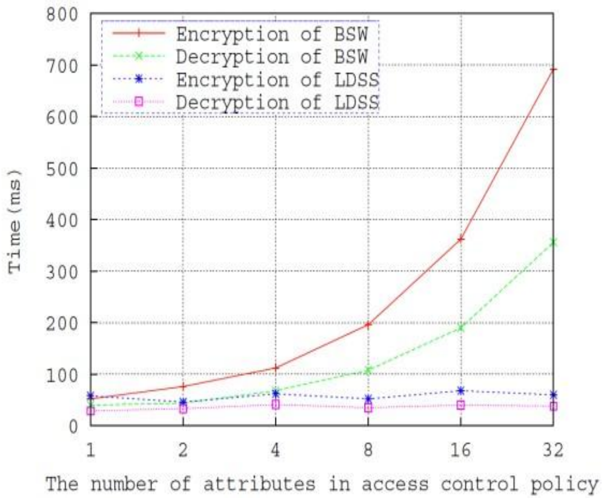

Figure 9. The relationship between users' overhead and the size of access control policy.

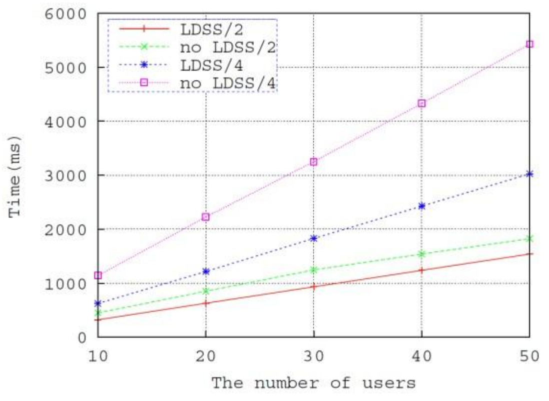

Figure 10. The overhead of attribute revocation.

The time needed for encryption and decryption is shown in Fig. 8.

As can be seen from Fig. 8, the overhead of encryption and decryption operations is proportional to the number of attributes in access control policy. In LDSS, it takes a little longer. Besides, the encryption and decryption time are lower than 1s when the number of attributes rises to 32 in both schemes.

Fig. 9 shows how the overhead on user side in BSW CP-ABE and LDSS changes with the size of access control policy. In LDSS, since the main encryption and decryption operations are given to the proxy server, the overhead on users' side is basically a constant value, on longer changing with the size of access control policy.

I. The overhead of user revocation

From the above analysis, the main overhead of user revocation comes from user attribute update operations. The overhead is related to the number of revoked attributes and related users. Assume that there are 32 attributes in the attribute set, and the average number of attributes owned by DU is 10. Fig. 10 shows how the overhead of user revocation changes with the number of data users when the number of revoked attributes is 2 and 4, respectively.

TABLE 5 - STORAGE OVERHEAD WITH DIFFERENT CP-ABES

CP-ABEs	PK	MK	SK	CT
BSW[27]	$3\,L_{G0}+L_{G1}$	L_z+L_{G0}	$(2\lvert A_u\rvert+1)\,L_{G0}$	$(2\lvert T_a\rvert+1)\,L_{G0}+L_{G1}$
Waters[30]	$(\lvert A\rvert+2)\,L_{G0}+L_{G1}$	L_{G0}	$(\lvert A_u\rvert+2)\,L_{G0}$	$(2\lvert T_a\rvert+1)\,L_{G0}+L_{G1}$
LDSS	$3\,L_{G0}+L_{G1}$	L_{G0}	$(\lvert A_u\rvert+4)\,L_{G0}$	$(2\lvert T_a\rvert+3)\,L_{G0}+L_{G1}$

As shown in Fig. 10, the overhead of user revocation is proportional to the number of data users, and LDSS works better than other CP-ABE. When the number of revoked attributes grows bigger, this advantage becomes more obvious.

In a word, the experimental results show that LDSS reduces the overhead on users' side significantly at a small cost of the overall growth on storage and computation. It also performs better in user revocation operations.

J. Storage Overhead Evaluation

We also evaluate the storage overhead of LDSS and compare it with existing CP-ABE schemes.

Storage Overhead with Different CP-ABE Schemes

DO needs to keep PK, which is of the size $(|A|+3)LG0+LG1$. DU also needs to keep SK, which is of the size $(|Au|+4)\,LG0$. TA needs to keep PK and MK. MK is of the size $LG0$. The cloud needs to keep the symmetric key ciphertext CT, which is of the size $(2|Ta|+3)\,LG0+$ LG1. DSP / ESP only do calculations and need not retain any value.

Table 5 shows the comparison of storage overhead with different CP-ABE schemes.

K. Measurement of Storage Overhead of LDSS

LDSS is based on 160-bit elliptic curve group, which is derived from the super singular curve $y^2 = x^3 + x$ over a 512-bit finite field. The size of $LG0$ 、 $LG1$ 、 Lz is 40B, 64B and 20B, separately.

In LDSS, the storage overhead needed for access control is the storage of PK/MK, SK and CT. PK and MK is 156B and 888B separately. The size of CT grows with the number of attributes in access control policy and the size of SK grows with the number of attributes in DU's attribute set.

When sharing data files, the data files is encrypted with symmetric key, then the symmetric key itself is encrypted by CP-ABE. Since the size of data files remains the same after encryption, we only evaluate the size change of the symmetric key.

Figure 11 shows the size of symmetric key after encryption when the number of attributes in access control policy is 1, 2, 4, 8, 16 and 32. It can be concluded that the size of ciphertext rises with the number of attributes in access control policy in both BSW CP-ABE [15] and LDSS. The size of symmetric key ciphertext of BSW CP-ABE is a little bigger than that of LDSS. When the number of attributes rises to 32, the size of symmetric key ciphertext is smaller than 10KB in both schemes, which is very small compared to the data files.

For DU authorization, the size of SK is linear with the number of attributes in DU's attribute set. Fig. 12 shows the size of SK when the number of attributes in DU's attribute set is 2, 4, 8 and 32, respectively.

When the number of attributes in the attribute set rises to 32, DU's SK is smaller than 1KB in both schemes, which is very small compared to the size of data files. The size of SK in LDSS is a little bigger for introducing an attribute version, but the difference is small.

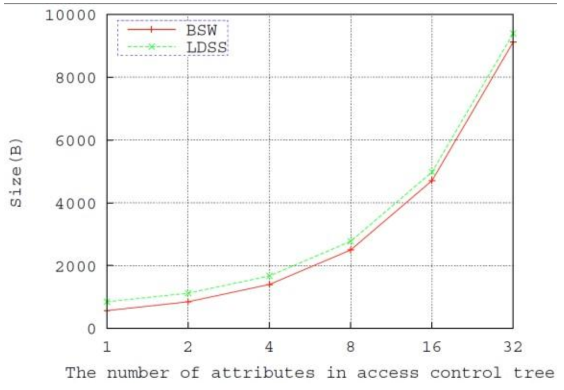

Figure 11. The relationship between symmetric key ciphertext and access control policy.

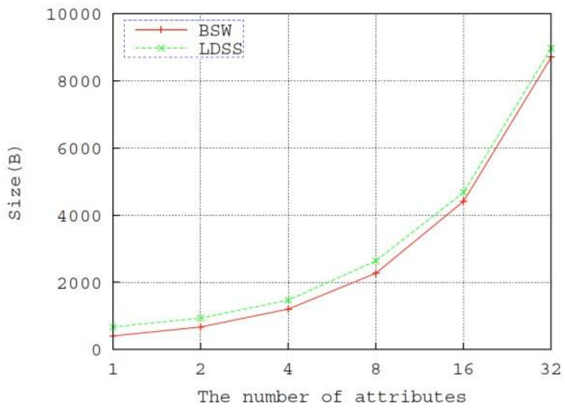

Figure 12. The storage overhead of SK.

In sum, in LDSS, the storage overhead needed for access control is very small compared to data files.

L. Communication Overhead Evaluation

The communication overhead of access control happens when TA sends keys to DO/DU at the stage of system initialization and user authorization, and DO/DU encrypt/decrypt the symmetric key which is used to encrypt the data files. According to the experimental results of Section 5.3, the key sent to TA is the MK of size 888B. The keys sent to DU are the attribute keys which are 8969B when the number of attributes owned by DU rises to 32. According to function 3 and 4, intermediate results of encryption/decryption transferred between DO/DU and ESP/DSP are of the size smaller than CT, which is lower than 10000B. Since the transferred data are in small amount, the communication cost is negligible.

VI. RELATED WORKS

In this section, we focus on the works of ciphertext access control schemes which are closely related to our research. Access control is an important mechanism of data privacy protection to ensure that data can only be acquired by legitimate users. There has been substantial research on the issues of data access control in the cloud, mostly focusing on access control over ciphertext. Typically, the cloud is considered honest and curious.

Sensitive data has to be encrypted before sending to the cloud.User authorization is achieved through key distribution. The research can be generally divided into four areas: simple ciphertext access control, hierarchical access control, access control based on fully homomorphic encryption [1][2] and access control based on attribute-based encryption (ABE).

Simple ciphertext access control refers to that after data file encryption, the encryption keys are distributed in a secure way to achieve authorization for trusted users [3]. To reduce the overhead of massive user key distribution, Skillen and Mannan [4] designed a system called Mobiflage that enables PDE (plausibly deniable encryption) on mobile devices by hiding encrypted volumes via random data on a device's external storage. However, the system needs to obtain large amount of information of keys. [5] borrows the access control method used in conventional distributed storage [4][6][12][14], separating users into different groups according to access rights and assign different keys to groups. This reduces the overhead of key management, but it cannot satisfy the demand for fine-grained access control.

Hierarchical access control has good performance in reducing the overhead of key distribution in ciphertext access control [7]. As a result, there are substantial research on ciphertext access control [8][9][10][11] based on hierarchical access control method. In hierarchical access control method, keys can be derived from private keys and a public token table. However, the operation on token table is complicated and generates high cost. Besides, the token table is stored in the cloud. Its privacy and security cannot be guaranteed [12].

Full homomorphic encryption algorithm can operate directly on the ciphertext. Its operating results are the same with operating on plaintext and then encrypting the data. [13] uses full homomorphic encryption algorithm to do operations such as retrieval and calculation directly on ciphertext. It can solve the problem that the cloud is untrustworthy fundamentally because all data update operations and user privilege change operations can be done directly on ciphertext. However, this encryption scheme is too complex to implement in practical applications.

Attribute-based encryption algorithm is derived from identity-based encryption. It embeds decryption rules in the encryption algorithm, which avoids frequent key distribution. Lai et al [14] and Bethencourt et al [15] proposed key-policy attribute-based encryption (KP-ABE) and ciphertext-policy attribute-based encryption (CP- ABE). In practical applications, CP-ABE has been extensively studied [16][17][18] since it is similar to role- based access control (RBAC) scheme [19]. In CP-ABE, the possession of one attribute key means that the key owner owns corresponding attribute, and attribute keys cannot be reclaimed once they are distributed. As a result, when a data user's attribute is revoked, how to ensure data privacy becomes a difficult issue [14]. Liang et al [16] propose attribute-based proxy re-encryption (ABPRE) scheme to solve this problem. However, in their solution, when a user's attribute is revoked, all other users who own this attribute will lose this attribute at the same time, which cannot satisfy fine-grained access control needs. Tian et al [20] combine CP-ABE and public key cryptography to achieve ciphertext access control. However, it brings high cost to data owners. Di Vimercati et al [21] add a time stamp to attributes to limit the use of attribute keys to deal with attribute revocation problem. However, in this scenario, data users need to periodically apply for attribute keys and the users' attribute cannot be revoked before the time stamp expires. Yu et al [22] propose some work of revocation can be outsourced to CSP, whereas CSP should have a certain credibility, and access control policy that contains "or" relationship or "threshold" relationship is not supported. Yu et al [23] also proposed a scheme to address the cloud computing challenging that keep sensitive user data confidential against untrusted servers by exploiting and uniquely combining techniques of attribute-based encryption (ABE), proxy re-encryption, and lazy re-encryption. Yang et al. [22] proposed a novel scheme that enabling efficient access control with dynamic policy updating for big data in the cloud that focusing on developing an outsourced policy updating method for ABE systems. It also designed policy updating algorithms for different types of access policies.

All the above works focus on the issue of data access control in the cloud. They are mainly for non-mobile devices and cannot be applied for data sharing in mobile cloud environment. Regarding to data privacy in mobile cloud, some works have been done in this field [23]. Huang et al [24] propose MobiCloud, in which traditional Mobile Ad-hoc NETworks (MANETs) is transformed into service-oriented communication architecture. In this architecture, each mobile device is regarded as a service node, and the operations are outsourced to the cloud. However, in MobiCloud, users need to completely trust the cloud, which is not the case in reality. Livshits and Jung [25] designed and implemented a graph theoretic algorithm to place mediation prompts that protect every
resource access, while avoiding repetitive prompting and prompting in background tasks or third-party libraries, for the problem of mediating resource accesses in mobile applications. Zhou et al [26] proposed an ABDS scheme to achieve secure data storage in the cloud. However, this scheme is not suitable for data sharing and has no clear solution for attribute revocation. Tysowski et al. [27] considered a specific cloud computing environment where data are accessed by resource-constrained mobile devices, and proposed novel modifications to ABE, which assigned the higher computational overhead of cryptographic operations to the cloud provider and lowered the total communication cost for the mobile user.

In summary, current proposals on data access control in the cloud are mostly for non-mobile terminals, which is not suitable for mobile devices. Besides, current solutions don't solve the problem of user privilege change scenarios very well since they bring high revocation cost. This is not applicable for mobile devices which only have limited computing capacity and power. Existing studies on mobile cloud don't have a good solution to secure data sharing when servers are not credible. In a word, there is no proper solution that can solve the problem of secure data sharing in mobile cloud. In this paper, we propose a lightweight data sharing scheme (LDSS) for mobile cloud applications. It adopts CP-ABE, a technology used in access control in the normal cloud environment, but changes the structure of access control tree to make it suitable for mobile cloud. LDSS is provably secure, and is demonstrated to be more efficient and scalable than state- of-the-art ABE schemes.

VII. CONCLUSION AND FUTURE WORK

In recent years, many studies on access control in cloud are based on attribute-based encryption algorithm (ABE). However, traditional ABE is not suitable for mobile cloud because it is computationally intensive and mobile devices only have limited resources. In this paper, we propose LDSS to address this issue. It introduces a novel LDSS-CP-ABE algorithm to migrate major computation overhead from mobile devices onto proxy servers, thus it can solve the secure data sharing problem in mobile cloud. The experimental results show that LDSS can ensure data privacy in mobile cloud and reduce the overhead on users' side in mobile cloud. In the future work, we will design new approaches to ensure data integrity. To further tap the potential of mobile cloud, we will also study how to do ciphertext retrieval over existing data sharing schemes.

VIII. ACKNOWLEDGMENT

This work is supported by National Natural Science Foundation of China under grants 61173170, 61300222, 61370230, 61433006 and U1401258, Innovation Fund of Huazhong University of Science and Technology under grants 2015TS069 and 2015TS071, Science and Technology Support Program of Hubei Province under grant 2014BCH270 and 2015AAA013, Science and Technology Program of Guangdong Province under grant 2014B010111007, and and Youth Talent Project of Science and Technology Research Program of Hubei Provincial Education Department under grant Q20151111.

References

[1] Gentry C, Halevi S. Implementing gentry's fully-homomorphic encryption scheme. in: Advances in Cryptology–EUROCRYPT 2011. Berlin, Heidelberg: Springer press, pp. 129-148, 2011.

[2] Brakerski Z, Vaikuntanathan V. Efficient fully homomorphic encryption from (standard) LWE. in: Proceeding of IEEE Symposium on Foundations of Computer Science. California, USA: IEEE press, pp. 97-106, Oct. 2011.

[3] Qihua Wang, Hongxia Jin. "Data leakage mitigation for discertionary access control in collaboration clouds". the 16th ACM Symposium on Access Control Models and Technologies (SACMAT), pp.103-122, Jun. 2011.

[4] Adam Skillen and Mohammad Mannan. On Implementing Deniable Storage Encryption for Mobile Devices. the 20th Annual Network and Distributed System Security Symposium (NDSS), Feb. 2013.

[5] Wang W, Li Z, Owens R, et al. Secure and efficient access to outsourced data. in: Proceedings of the 2009 ACM workshop on Cloud computing security. Chicago, USA: ACM pp. 55-66, 2009.

[6] Maheshwari U, Vingralek R, Shapiro W. How to build a trusted database system on untrusted storage. in: Proceedings of the 4th conference on Symposium on Operating System Design & Implementation-Volume 4. USENIX Association, pp. 10-12, 2000.

[7] Kan Yang, Xiaohua Jia, Kui Ren: Attribute-based fine-grained access control with efficient revocation in cloud storage systems. ASIACCS 2013, pp. 523-528, 2013.

[8] Crampton J, Martin K, Wild P. On key assignment for hierarchical access control. in: Computer Security Foundations Workshop. IEEE press, pp. 14-111, 2006.

[9] Shi E, Bethencourt J, Chan T H H, et al. Multi-dimensional range query over encrypted data. in: Proceedings of Symposium on Security and Privacy (SP), IEEE press, 2007. 350- 364

[10] Cong Wang, Kui Ren, Shucheng Yu, and Karthik Mahendra Raje Urs. Achieving Usable and Privacy-assured Similarity Search over Outsourced Cloud Data. IEEE INFOCOM 2012, Orlando, Florida, March 25-30, 2012

[11] Yu S., Wang C., Ren K., Lou W. Achieving Secure, Scalable, and Fine-grained Data Access Control in Cloud Computing. INFOCOM 2010, pp. 534-542, 2010

[12] Kan Yang, Xiaohua Jia, Kui Ren, Bo Zhang, Ruitao Xie: DAC- MACS: Effective Data Access Control for Multiauthority Cloud Storage Systems. IEEE Transactions on Information Forensics and Security, Vol. 8, No. 11, pp.1790-1801, 2013.

[13] Stehlé D, Steinfeld R. Faster fully homomorphic encryption. in: Proceedings of 16th International Conference on the Theory and Application of Cryptology and Information Security. Singapore: Springer press, pp.377-394, 2010.

[14] Junzuo Lai, Robert H. Deng ,Yingjiu Li ,et al. Fully secure key- policy attribute-based encryption with constant-size ciphertexts and fast decryption. In: Proceedings of the 9th ACM symposium on Information, Computer and Communications Security (ASIACCS), pp. 239-248, Jun. 2014.

[15] Bethencourt J, Sahai A, Waters B. Ciphertext-policy attribute based encryption. in: Proceedings of the 2007 IEEE Symposium on Security and Privacy (SP). Washington, USA: IEEE Computer Society, pp. 321-334, 2007.

[16] Liang Xiaohui, Cao Zhenfu, Lin Huang, et al. Attribute based proxy re-encryption with delegating capabilities. in: Proceedings of the 4th International Symposium on Information, Computer and Communications Security. New York, NY, USA: ACM press, pp. 276-286, 2009.

[17] Pirretti M, Traynor P, McDaniel P, et al. Secure atrribute-based systems. in: Proceedings of the 13th ACM Conference on Computer and Communications Security. New York, USA: ACM press, pp. 99-112, 2006.

[18] Yu S., Wang C., Ren K., et al. Attribute based data sharing with attribute revocation. in: Proceedings of the 5th International Symposium on Information, Computer and Communications Security (ASIACCS), New York, USA: ACM press pp. 261-270, 2010.

[19] Sandhu R S, Coyne E J, Feinstein H L, et al. Role-based access control models. Computer, 29(2): 38-47, 1996.

[20] Tian X X, Wang X L, Zhou A Y. DSP RE-Encryption: A flexible mechanism for access control enforcement management in DaaS. in: Proceedings of IEEE International Conference on Cloud Computing. IEEE press, pp.25-32, 2009

[21] Di Vimercati S D C, Foresti S, Jajodia S, et al. Over-encryption: management of access control evolution on outsourced data. in: Proceedings of the 33rd international conference on Very large data bases. Vienna, Austria: ACM, pp. 123-134, 2007.

[22] Kan Yang, Xiaohua Jia, Kui Ren, Ruitao Xie, Liusheng Huang: Enabling efficient access control with dynamic policy updating for big data in the cloud. INFOCOM 2014, pp.2013-2021, 2014.

[23] Jia W, Zhu H, Cao Z, et al. SDSM: a secure data service mechanism in mobile cloud computing. in: Proceedings of 30th IEEE International Conference on Computer Communications. Shanghai, China: IEEE, pp. 1060-1065, 2011.

[24] D. Huang, X. Zhang, M. Kang, and J. Luo. Mobicloud: A secure mobile cloud framework for pervasive mobile computing and communication. in: Proceedings of 5th IEEE International Symposium on Service-Oriented System Engineering. Nanjing, China: IEEE, pp. 90-98, 2010.

[25] Benjamin Livshits, Jaeyeon Jung. Automatic Mediation of Privacy-Sensitive Resource Access in Smartphone Applications. USENIX Security, pp.113-130, Aug. 2013.

[26] Zhou Z, Huang D. Efficient and secure data storage operations for mobile cloud computing. in: Proceedings of 8th International Conference on Network and Service Management (CNSM 2012), Las Vegas, USA: IEEE, pp. 37-45, 2012.

[27] P. K. Tysowski and M. A.Hasan. Hybrid attribute- and re- encryption-based key management for secure and scalable mobile applications in clouds. IEEE Transactions on Cloud Computing, vol. 1, no. 2, pp. 172-186, Nov. 2013.

[28] Boneh D, Franklin M. Identity-based encryption from the Weil pairing. in: Proceedings of the Advances in Cryptology. Berlin, Heidelberg: Springer-Verlag, pp. 213−229, 2001.

[29] Sahai A, Waters B. Fuzzy identity based encryption. in: Proceedings of the Advances in Cryptology. Aarhus, Denmark: Springer-Verlag, pp.457-473, 2005.

[30] Shamir A. How to share a secret. Communications of the ACM,1979, 22 (11): 612-613