2011 International Joint Conference of IEEE TrustCom-11/IEEE ICESS-11/FCST-11

DACC: Distributed Access Control in Clouds

Sushmita Ruj, Amiya Nayak and Ivan Stojmenovic
SEECS, University of Ottawa,
Ottawa K1N 6N5, Canada
Email: {sruj, anayak, ivan}@site.uottawa.ca

***Abstract*—We propose a new model for data storage and access in clouds. Our scheme avoids storing multiple encrypted copies of same data. In our framework for secure data storage, cloud stores encrypted data (without being able to decrypt them). The main novelty of our model is addition of key distribution centers (KDCs). We propose DACC (Distributed Access Control in Clouds) algorithm, where one or more KDCs distribute keys to data owners and users. KDC may provide access to particular fields in all records. Thus, a single key replaces separate keys from owners. Owners and users are assigned certain set of attributes. Owner encrypts the data with the attributes it has and stores them in the cloud. The users with matching set of attributes can retrieve the data from the cloud. We apply attribute-based encryption based on bilinear pairings on elliptic curves. The scheme is collusion secure; two users cannot together decode any data that none of them has individual right to access. DACC also supports revocation of users, without redistributing keys to all the users of cloud services. We show that our approach results in lower communication, computation and storage overheads, compared to existing models and schemes.**

Keywords: Access control, Decentralized attribute-based encryption, Bilinear maps, Storage in Clouds.

I. Introduction

Cloud computing is an emerging technology, in which a user can rent the storage and computing resources of a server (also known as cloud) provided by a company. Users only require a terminal, a smart phone or tablet connected to the Internet. The applications run in the cloud instead of the user's machines. Clouds can store huge amount of data, so that mobile users do not have to carry their data. Some clouds provide application services (e.g., Google Apps [1], Microsoft online [2]), some provide infrastructural support (e.g., Amazon's EC2 [3], Eucalyptus [4], Nimbus [5]), or platform, to help developers to write applications that will run on the cloud (e.g., Amazon's S3 [6], Windows Azure [7]).

It is important to preserve the security of data and privacy of users. Cloud should ensure that the users trying to access data and services are authorized users. Authentication of users can be achieved using public key cryptographic techniques [8]. Users should ensure that the cloud is not tampering with their data and computational results. It might also be important to hide the users identity for privacy reasons. For example, while storing medical records, the cloud should not be able to access records of a particular patient, given the identity.

Users should also ensure that the cloud is able to perform some computations on the data, without knowing the actual data values. One way to hide the data from the clouds, but carry on computation on the data, is by the use of homomorphic encryption techniques [9], [10]. User sends homomorphicly encrypted messages, while the cloud (without knowing the actual data) performs computations on these encrypted messages and returns the results to the user. Code obfuscation techniques [11] can also be used so that the user wishing to execute its program in the cloud will not be able to understand the code, but only be provided with the results. An important problem is to anonymously search data stored in clouds, with different types of queries (e.g., range, multidimensional, or complex queries). This is achieved by means of searchable encryption [12], [13], which takes keywords in encrypted form, searches the databases by checking the indices of the encrypted keywords, and returns results, without knowing the keyword or the retrieved records. Efficient and secure keyword searches are required in clouds, which maintain large amounts of sensitive data [14], [15].

Consider now the following scenarios. Patients store their medical records in the cloud. Different users can access different data fields. The same data fields might be accessed by a selective group of people (authorized set). For example the patient's medical history and drug administration can be accessed by doctors and nurses, but not by hospital management staff.

In online social networking scenario, owners are members of the networking site, who keep their personal details, pictures, music videos in the cloud and other members can view them depending upon their access rights. A member can post a message or upload a picture, which will be visible only to the friends and certain selected communities (that she belongs to) but not accessible to the rest. It is desirable to also protect privacy of these data from the cloud.

Granting access rights to certain users and forbidding other users to access the data, is called *access control*. One way to achieve this is to attach a list of all valid users to the data. In cloud computing scenario, such lists can be extremely long and often dynamic, which will make handling such lists extremely difficult. Each time the list has to be checked to see if the user is valid. This results in a huge computation and storage costs. Another way to encrypt data is by using public keys of valid users, so that only they are able to decrypt data using their secret keys. However the same data then must be encrypted several times (individually for each user), which may result in huge storage costs. Hence we use the cryptographic technique called Attribute Based Encryption (ABE) to achieve access control in clouds. Using ABE, owners encrypt data with attributes that they possess and store the

978-0-7695-4600-1/11 $26.00 © 2011 Crown Copyright
DOI 10.1109/TrustCom.2011.15

information in the cloud. The cloud is unable to decode stored data. Users are given attributes and secret keys by a key distribution center (KDC). Those with matching set of attributes are able to decrypt the information. For example, consider a public health records repository [16]. The medical records contain history of the patients and might be accessed either by medical professionals (doctors and nurses), researchers and academicians or management authorities (insurance companies and government policy makers). Different people are entitled to access different records. Each user is given attributes such as the affiliation (hospital names), designation (occupation and specialization) etc. For example, only a psychiatrist or neurologist in Hospital A or B will be able to decipher the record the medical history of a bipolar person. Any other user, for example a hospital staff of hospital A or B, or a neurologist from Hospital C, will not be able to access the record.

Earlier work (by Yu *et al* [17]) assumed that owners encrypt data with attributes that they possess and send it to the cloud to securely store them. The owners give not only access rights to users, but also give them attributes and corresponding secret keys. Thus, KDC is not required. The problem with this technique is that users receive different secret keys from different owners, for the same attribute. This increases the total number of secret keys given to the users, which increases storage and communication overhead. A user might have attributes that enable it to access data belonging to several owners. A typical example is the patient records of particular hospital. In our approach, KDC is able to provide such a key.

A. Our contribution

We propose DACC: Distributed Access Control in Clouds, a new access control mechanism, where owners decide on attributes that users should have and users receive decryption keys which enable them to access records which they are authorized to access. The novelty of our scheme is the addition of key distribution centers (KDCs), which distribute secret keys to users. If there is only one KDC, then compromising it may render the whole system ineffective. To prevent such single point of failure, we have multiple KDCs (which act as attribute authorities). The cloud stores only ciphertexts (and is unable to decrypt them). Only one copy of ciphertext per record is stored, instead of multiple copies. Each attribute authority has a disjoint set of attributes. Owners decide the attributes that users should have, in order to access information. Each owner has an access policy (in the form of an access tree). Owners encrypt their information using the public keys belonging to the attributes in the policy and stores this encrypted information in the cloud. The selection of KDCs depends on the application. For example, to store health records, there can be several KDCs. A user can receive its affiliation from the government and its designation from head of medical research institute etc. Our access control mechanism also ensures that, even if two users collude, they cannot decrypt any information, that they are not individually authorized to obtain.

The access policy is in the form of a tree, with attributes as leaf nodes and Boolean functions AND and OR as internal nodes. Threshold gates t-out-of-n are included in this representation, because they are constructed from AND and OR gates when $t = n$ and $t = 1$, respectively.

We apply Lewko and Water's [18] technique to achieve access control in general settings. We tailor it for use in the cloud scenario. DACC has an extra feature to allow revocation of users, without redistributing decryption keys to all the users in the network, which was not discussed in [18]. Moreover, we show that it performs better than other proposed access control schemes for clouds.

B. Organization

The paper is organized in the following way. We present related work on cloud security in Section II. In Section III, we describe mathematical tools, network and attack models used in our access control mechanism. We discuss our scheme in details in Section IV. Sections V and VI analyze the security and performance of our scheme, respectively. We conclude the paper with open problems in Section VII.

II. Related Work

We will discuss several attribute based encryption schemes. This will help us to better appreciate why Lewko and Water's scheme [18] is best suited for data access control in clouds.

ABE was proposed by Sahai and Waters [19]. In ABE, a user has a set of attributes in addition to its unique ID. Identity-based encryption (IBE) was proposed by Shamir [20] and has been extensively studied. Each user in an IBE scheme has a unique identity, and the public key is the unique information about the user. IBE is a special case of ABE. There are two classes of ABE. In Key-policy ABE or KP-ABE (Goyal *et al*[21]), the sender has an access policy to encrypt data. The receiver receives attributes and secret keys from the attribute authority and is able to decrypt information if it has matching attributes. In Ciphertext-policy, CP-ABE (Bethencourt et al. [22]), the receiver has the access policy in the form of a tree, with attributes as leaves and monotonic access structure with AND, OR and other threshold gates.

The attribute authority (KDC) in all mentioned protocols is assumed to be honest. However, this may not hold, because in a distributed system, authorities can fail or be corrupt. Chase [23] proposed a multi-authority ABE, in which there are several KDC authorities (coordinated by a trusted authority) which distribute attributes and secret keys to users. Chase and Chow [24] devised a multi-authority ABE protocol which required no trusted authority. However, the main problem was that a user required at least one attribute from each of the authorities, which might not be practical. Recently, Lewko and Waters [18] proposed a fully decentralized ABE, where users could have zero or more attributes from each authority and did not require a trusted server. Their protocol has been recently applied to achieve access control in intelligent transport system [25]. This enables vehicles to transmit messages, such that only authorized vehicles can receive them.

Yu *et al* [17] scheme used ABE for access control. However, as already discussed, it has a few limitations. Li *et al* [16] proposed a secure cloud storage mechanism for health records. ABE was used for access control. They used cryptographic protocol [24], with multiple authorities to distribute attributes for access control. A patient's health record can be accessed (read and modified) by different people (friends and family, researchers, doctors and nurses or hospital staff and insurance company). Users are divided into private domain users (friends and family) and public domain users (the rest). Public domain users can be of the following types: education domain (consisting of researchers), health-care domain (consisting of doctors, nurses etc), insurance domain (insurance company, actuary health insurance). Each of the domains has multiple authorities, which generate secret keys and attributes from the users. The problem with this approach is that a user should have attributes from each KDC in its domain, which makes the access structure restrictive. It is also not clear who these KDCs are in reality.

Wang *et al* [26] proposed a hierarchical access control mechanism for cloud storage. It relies on Bethencourt *et al* [22] CP-ABE and Hierarchical IBE [27]. The attribute authorities are arranged in a hierarchical way, with a root master (playing the role of a trusted KDC), and several domain masters who generate keys for the domains and distribute them to the users. This scheme still relies on the trusted authority (root master) and fails, if the later is compromised. Ruj *et al.* [28] used attribute based encryption for access control in sensor networks.

III. Background

In this section we present our cloud storage model and the assumptions we have used in the paper. Table I presents the notations used throughout the paper. We also describe mathematical background used in our proposed solution.

TABLE I
NOTATIONS

Symbols	Meanings
U_u	u-th User/Owner
A_j	j-th KDC
$\mathcal{A}$	Set of KDCs
$\mathcal{W}$	Set of attributes
$w = \|\mathcal{W}\|$	Number of attributes
L_j	Set of attributes that KDC A_j possesses
$l_j = \|L_j\|$	Number of attributes that KDC A_j possesses
$I[j,u]$	Set of attributes that A_j gives to user U_u
I_i	Set of attributes that user n_i possesses
$PK[j]$	Public key of KDC A_j
$SK[j]$	Secret key of KDC A_j
$sk_{i,u}$	Secret key given by A_j corresponding to attribute i given to user U_u
S	Boolean access structure
R	Access matrix of dimension $n \times h$
$\|G\|$	Order of group G
M	Message
C	Ciphertext
H	Hash function, example SHA-1

A. Model and assumptions

We consider a network in which owners want to store their information in the cloud while users want to access the same information from the cloud. We do not consider computations in cloud. In a health-care scenario, owners can be the patients who store their records in the cloud, and doctors, nurses, researchers, insurance companies can retrieve them. There are KDCs which may be even servers scattered in different countries, that generate secret keys for the users. KDCs can be government organizations which give different credentials to users. These servers can be maintained by separate companies, so that they do not collude with each other. This differs from the traditional concept of cloud. A particular cloud is maintained by one company, thus if KDCs are a part of the cloud then they can collude and find the secret keys of users. Fig. 1 shows the overall model of our cloud environment. Users and owners are denoted by n_i, KDCs are servers which distribute attributes and secret keys SK to users and owners. KDCs are not part of the cloud. The owner encrypts message and stores the ciphertext C in the cloud.

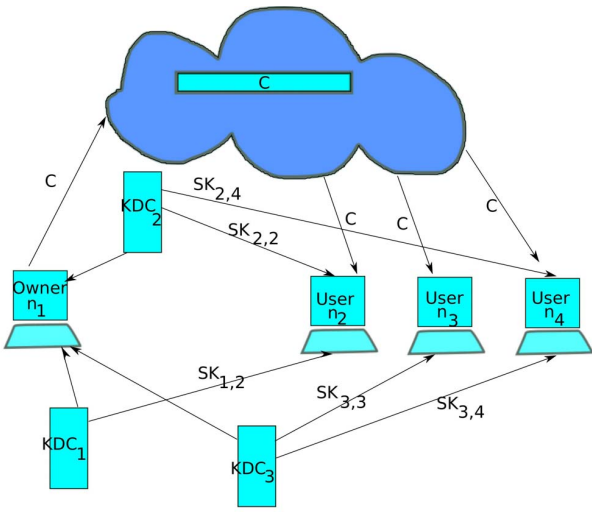

Fig. 1. Proposed cloud model

We assume that the cloud is honest but curious. It will not tamper with the data which is stored in the cloud, but will try to calculate the secret key from user inputs. It might collude with users and try to calculate the secret key of other users. Users can also attempt to collude and gain access of data that they are not individually authorized to.

B. Formats of access policies

Access policies can be in either formats 1) Boolean functions of attributes or 2) Linear Secret Sharing Scheme (LSSS) matrix. Any access structure can be converted into a Boolean function [18]. An example of a boolean function is $((a_1 \wedge a_2 \wedge a_3) \vee (a_4 \wedge a_5)) \wedge (a_6 \vee a_7))$, where $a_1, a_2, \ldots, a_7$ are attributes. Boolean functions can also be represented by access tree, with attributes at the leaves and $AND(\wedge)$ and $OR(\vee)$

as the intermediate nodes and root. Our pseudo-code of an algorithm that converts a Boolean function (in the form of access tree) to a LSSS matrix is given in the Appendix. The algorithm is described in [18] as follows. Root has vector (1). Let $v[x]$ be parent's vector. If node x=AND, then the left child is $(v[x]|1)$, and the right child is $(0,\ldots,-1)$. If x=OR, then both children also have unchanged vector $v[x]$. Finally, pad with 0s at end, such that all vectors are of equal length. The proof of validity of the algorithm is given in [29]. Fig. 2 shows an access tree with initial vectors. The rows of R are the required vectors.

C. Mathematical background

We will use bilinear pairings on elliptic curves. Let G be a cyclic group of prime order q generated by g. Let G_T be a group of order q. We can define the map $e: G \times G \rightarrow G_T$. The map satisfies the following properties:

1) $e(aP, bQ) = e(P, Q)^{ab}$ for all $P, Q \in G$ and $a, b \in \mathbb{Z}_q$, $\mathbb{Z}_q = \{0, 1, 2, \ldots, q-1\}$.
2) Non-degenerate: $e(g, g) \neq 1$.

We use bilinear pairing on elliptic curves groups. We do not discuss the pairing functions which mainly use Weil and Tate pairings [30] and computed using Miller's algorithm [31]. The choice of curve is an important consideration, because it determine the complexity of pairing operations. A survey on pairing friendly curves can be found in [32].

PCB library (Pairing Based Cryptography) [30] is a C library which is built above GNU GMP (GNU Math Precision) library and contains functions to implement elliptic curves and pairing operations. The curves chosen are either MNT curves or supersingular curves. Considering the requirements, elliptic curve group of size 159, with an embedding degree 6 (type d curves of PBC [30]) can be used. Pairing takes 14 ms on Intel Pentium D, 3.0 GHz CPU [33]. Such operations are very suitable for a cloud computing environment.

D. Lewko-Waters ABE scheme

Lewko-Waters [18] scheme consists of four steps: 1) System Initialization, 2) Key and attribute distribution to users By KDCs 3) Encryption of message by sender 4) Decryption by receiver.

1) System Initialization: Select a prime q, generator g of G, groups G and G_T of order q, a map $e: G \times G \rightarrow G_T$, and a hash function $H: \{0,1\}^* \rightarrow G$ which maps the identities of users to G. The hash function used here is SHA-1 [34]. Each KDC $A_j \in \mathcal{A}$ has a set of attributes L_j. The attributes disjoint ($L_i \bigcap L_j = \phi$ for $i \neq j$). Each KDC also chooses two random exponents $\alpha_i, y_i \in \mathbb{Z}_q$. The secret key of KDC A_j is

$$SK[j] = \{\alpha_i, y_i, i \in L_j\}. \tag{1}$$

The public key of KDC A_j is published:

$$PK[j] = \{e(g,g)^{\alpha_i}, g^{y_i}, i \in L_j\}. \tag{2}$$

2) Key generation and distribution by KDCs: User U_u receives a set of attributes $I[j,u]$ from KDC A_j, and corresponding secret key $sk_{i,u}$ for each $i \in I[j,u]$

$$sk_{i,u} = g^{\alpha_i} H(u)^{y_i}, \tag{3}$$

where $\alpha_i, y_i \in SK[j]$. Note that all keys are delivered to the user securely using the user's public key, such that only that user can decrypt it using its secret key.

3) Encryption by sender: Sender decides about the access tree. LSSS matrix R can be derived as described in III-B. Sender encrypts message M as follows:

1) Choose a random seed $s \in \mathbb{Z}_q$ and a random vector $v \in \mathbb{Z}_q^h$, with s as its first entry; h is the number of leaves in the access tree (equal to the number of rows in the corresponding matrix R).
2) Calculate $\lambda_x = R_x \cdot v$, where R_x is a row of R.
3) Choose a random vector $w \in \mathbb{Z}_q^h$ with 0 as the first entry.
4) Calculate $\omega_x = R_x \cdot w$.
5) For each row R_x of R, choose a random $\rho_x \in \mathbb{Z}_q$.
6) The following parameters are calculated:

$$\begin{aligned} C_0 &= M e(g,g)^s \\ C_{1,x} &= e(g,g)^{\lambda_x} e(g,g)^{\alpha_{\pi(x)}\rho_x}, \forall x \\ C_{2,x} &= g^{\rho_x} \forall x \\ C_{3,x} &= g^{y_{\pi(x)}\rho_x} g^{\omega_x} \forall x, \end{aligned} \tag{4}$$

where $\pi(x)$ is mapping from R_x to the attribute i that is located at the corresponding leaf of the access tree.

7) The ciphertext C is sent by the sender (it also includes the access tree via R matrix):

$$C = \langle R, \pi, C_0, \{C_{1,x}, C_{2,x}, C_{3,x}, \forall x\}\rangle. \tag{5}$$

4) Decryption by receiver: Receiver U_u takes as input ciphertext C, secret keys $\{sk_{i,u}\}$, group G, and outputs message M. It obtains the access matrix R and mapping π from C. It then executes the following steps:

1) U_u calculates the set of attributes $\{\pi(x) : x \in X\} \bigcap I_i$ that are common to itself and the access matrix. X is the set of rows of R.
2) For each of these attributes, it checks if there is a subset X' of rows of R, such that the vector $(1, 0 \ldots, 0)$ is their linear combination. If not, decryption is impossible. If yes, it calculates constants $c_x \in \mathbb{Z}_q$, such that $\sum_{x \in X'} c_x R_x = (1, 0, \ldots, 0)$.
3) Decryption proceeds as follows:
   a) For each $x \in X'$, $dec(x) = \frac{C_{1,x} e(H(u), C_{3,x})}{e(sk_{\pi(x),u}, C_{2,x})}$
   b) U_u computes $M = C_0 / \Pi_{x \in X'} dec(x)$.

IV. Proposed scheme: DACC

We now discuss our scheme: Distributed Access Control in Clouds (DACC). We first give a sketch of our scheme and then work out the mathematical details. A practical example is then presented to demonstrate how the scheme works.

A. Sketch of DACC

Initially the parameters of the scheme and the size of group are decided. The size of the group is chosen to be quite high, for example $2^{32}+1$. Set of attributes are selected by the KDCs. KDC A_j selects the set of attributes L_j. An owner U_u who wants to store information in the cloud, chooses a set of attributes I_u which are specific to the data it wants to encrypt. These attributes may belong to different KDCs. It then decides on the access structure, and converts the access tree to a matrix R, using Algorithm given in Section III-B. Depending upon the attributes it possesses and the keys it receives from the KDC, it encrypts and sends the data, and the access matrix. Each user is given a set of attributes when it registers for services from owners. The attributes are not given by the cloud, but by the KDCs. A ssh protocol (Secure shell protocol [35]) is used to securely transfer the attribute information. KDCs give secret keys to users. When a user wants to access some information, it asks the cloud for the data record. The cloud gives it an encrypted copy of the data. If a user has a valid set of attributes then it calculates the data using the secret key that it possesses.

B. Details of DACC

1) Data Encryption: Encryption proceeds in two steps. The Boolean access tree is first converted to LSSS matrix. In the second step the message is encrypted and sent to the cloud along with the LSSS matrix. A secure channel like ssh can be used for the transmission.

Suppose an owner U_u wants to store a record M. U_u defines the access structure S, which helps it to decide the authorized set of users, who can access the record M. It then creates a $m \times h$ matrix R (m is the number of attributes in the access structure) and defines a mapping function π of its rows with the attributes (using Algorithm in Section III-B). π is a permutation, such that $\pi : \{1, 2, \ldots, m\} \rightarrow \mathcal{W}$. The encryption algorithm takes as input the data M that needs to be encrypted, the group G, the LSSS matrix R, the permutation function π, which maps the attributes in the LSSS to the actual set of attributes. For each message M, the ciphertext C is calculated as per the Equations (4) and (5). Ciphertext C is then stored in the cloud.

When a user U_u requests a ciphertext from the cloud, the cloud transfers the requested ciphertext C using ssh protocol. The decryption algorithm proceeds as in Section III-D4, and returns plaintext message M, if the user has valid set of attributes.

C. An Example

Consider a professional network, where members can post their resume and look for jobs or possible employees. A person can belong to several professional groups located in different parts of the world. Suppose a member is looking for a job and uploads her resume, so that only particular communities can view it. For example, the communities that can view are "Jobs in Engineering" or "CS Research positions in Canada" or "Faculty positions in CS in US". Let there be two attribute centers 1) Types of jobs: J_1 (Engineering), J_2 (CS Research), J_3 (Faculty positions) and 2) Locations : P_1 (Canada), P_2 (US). Then the access tree is given in Figure 2.

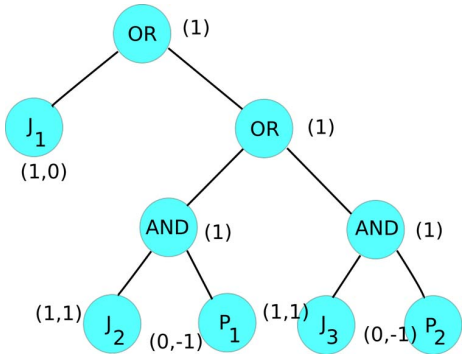

Fig. 2. Access tree for the job search example; 0 should be padded to change (1) to $(1, 0)$

The access matrix R can be constructed using Algorithm in Section III-B. Thus,

$$R = \begin{pmatrix} 1 & 0 \\ 1 & 1 \\ 0 & -1 \\ 1 & 1 \\ 0 & -1 \end{pmatrix}.$$

A group which discusses jobs related to CS positions in US and Canada will be able to access the resume. However, if there is a community which discusses social works jobs in Canada, then it will not be able to access the resume.

Let there be two KDCs A_1 and A_2. The set of attributes of A_1 and A_2 are $L_1 = \{J_1, J_2, J_3, \ldots\}$ and $L_2 = \{P_1, P_2, \ldots\}$. The owner has the access tree as shown in Fig. 2. Let π be denoted as

x	1	2	3	4	5
$\pi(x)$	J_1	J_2	P_1	J_3	P_2

.

Suppose there is a community (user $u = 2$) which advertises CS research positions and Faculty positions in Canada. Then it is given the attributes J_2, J_3 and P_1. Thus, $I[1, 2] = \{J_2, J_3\}$ and $I[2, 2] = \{P_1\}$. Next the user is given secret keys $sk_{2,1}, sk_{3,2}$ from A_1 and $sk_{1,2}$ from A_2.

During encryption, the owner sends to the cloud the information $C = \langle R, \pi, C_0, \{C_{1,x}, C_{2,x}, C_{3,x}\}_{x\in\{1,2,3,4,5\}}\rangle$. $C_0 = Me(g, g)^s$, where s is chosen at random from $\mathbb{Z}_q$.

When user 2 wants to access the above information C, the cloud transfers it securely, using ssh (an inbuilt secure shell standard protocol). The user first finds out the attributes that are present from π. It also finds that it has the attributes J_2, J_3, P_1 in common to the owner. From the matrix R it then finds that there are two rows corresponding to J_2 and P_1, such that $(1, -1) + (0, 1) = (1, 0)$ (linear combination of rows 2 and 3 of R gives $(1, 0)$).

It can thus calculate $e(g, g)^s$ according to Step 4 of the decryption mechanism. Once $e(g, g)^s$ is calculated, M can be

obtained. The cloud does not have the secret keys, so cannot decrypt the message.

D. User revocation

Users might be revoked. Revoked users must not have the ability to access data, even if they posses matching set of attributes. For this reason, the owners should change the stored data and send updated information to other users. The set of attributes I_u possessed by the revoked user U_u is noted and all users change their stored data that have attributes $i \in I_u$. In [17], revocation involved changing the public and secret keys of the minimal set of attributes which are required to decrypt the data. We do not consider this approach because here different data are encrypted by the same set of attributes, so such a minimal set of attributes is different for different users. Therefore, this does not apply to our model. Once the attributes I_u are identified, all data that possess the attributes are collected. For each such data record, the following steps are then carried out:

1) A new value of s, $s_{new} \in \mathbb{Z}_q$ is selected.
2) The first entry of vector v_{new} is changed to new s_{new}.
3) $\lambda_x = R_x v_{new}$ is calculated, for each row x corresponding to leaf attributes in I_u.
4) $C_{1,x}$ is recalculated for x.
5) New value of $C_{1,x}$ is securely transmitted to the cloud.
6) New $C_0 = Me(g,g)^{s_{new}}$ is calculated and stored in the cloud
7) New value of $C_{1,x}$ is not stored with the data, but is transmitted to users, who wish to decrypt the data.

We note here that the new value of $C_{1,x}$ is not stored in the cloud but transmitted to the non-revoked users who have attribute corresponding to x. This prevents a revoked user to decrypt the new value of C_0 and get back the message.

V. Security of DACC

We will show that only authorized users can decrypt the data stored in clouds. The cloud will not be able to read or modify any data stored. Even if it colludes with a user, it cannot decrypt any data that the user cannot decrypt alone. Two or more users also cannot collude and decrypt any information, they are not able to decrypt alone.

Theorem 1: Our access control scheme is secure (no outsider or cloud can decrypt ciphertexts), collusion resistant and allows access only to authorized users.

Proof: We first show that no unauthorized user can access data from the cloud. We will first prove the validity of our scheme. A user can decrypt data if and only if it has a matching set of attributes. This follows from the fact that access structure S (and hence matrix R) is constructed if and only if there exists a set of rows X' in R, and linear linear constants $c_x \in \mathbb{Z}_q$, such that $\sum_{x \in X'} c_x R_x = (1, 0, \ldots, 0)$. A proof of this appear in [29, Chapter 4].

We note that

$$dec(x) = \frac{C_{1,x} e(H(u), C_{3,x})}{e(sk_{\pi(x),u}, C_{2,x})} = e(g,g)^{\lambda_x} e(H(u), g)^{\omega_x} \quad (6)$$

Thus,

$$\begin{aligned} &\Pi_{x \in X'} dec(x) \\ &= \Pi_{x \in X'} (e(g,g)^{\lambda_x} e(H(u), g)^{\omega_x})^{c_x} \\ &= e(g,g)^s \end{aligned} \quad (7)$$

Equation (7) above holds because $\lambda_x = R_x \cdot v$ and $\omega_x = R_x \cdot w$, where $v \cdot (1, 0, \ldots, 0) = r$ and $\omega \cdot (1, 0, \ldots, 0) = 0$. $C_0 / \Pi_{x \in X'} dec(x) = C_0 / e(g,g)^s = M$.

For an invalid user, there does not exists attributes corresponding to rows x, such that $\sum_{x \in X'} c_x R_x = (1, 0, \ldots, 0)$. Thus $e(g,g)^s$ cannot be calculated.

We next show that two or more users cannot collude and gain access to data that they are not individually supposed to access. Suppose that there exist attributes $\pi(x)$ from the colluders, such that $\sum_{x \in X} c_x R_x = (1, 0, \ldots, 0)$. However, $e(H(u), g)^{\omega_x}$ needs to be calculated according to Eq. (7). Since different users have different values of $e(H(u), g)$, even if they combine their attributes, they cannot decrypt the message.

We next observe that the cloud cannot decode stored data. This is because it does not posses the secret keys $sk_{i,u}$ (by Eq.(3)). Even if it colludes with other users, it cannot decrypt data which the users cannot themselves decrypt, because of the above reason (same as collusion of users). The KDCs are located in different servers and are not owned by the cloud. For this reason, even if some (but not all) KDCs are compromised, the cloud cannot decode data. An important aspect of our design is that cloud does not know the attributes of the users. ■

VI. Performance

We calculate the computation and communication overhead of DACC scheme and DACC with revocation. Time taken to compute R from S is $O(m)$, where m is the number of attributes in the access structure. To check if there exists a set of rows in R (such that step (2) of decryption holds), is equivalent to solving the equation $CR = (1, 0, \ldots, 0)$, for non-zero row vector C. This takes $O(mh)$.

Pairing are the most expensive operations in DACC. During encryption, each user U_u performs only one pairing operation (to calculate $e(g,g)$). For each row x corresponding to attribute, it also performs two scalar multiplications to calculate $C_{1,x}$, one scalar multiplication to calculate $C_{2,x}$ and one to calculate $C_{3,x}$. Thus there are a total of $4m$ scalar multiplications. During decryption, there are two pairing operations, one for $e(H(u), C_{3,x})$ and the other for $e(sk_{i,u}, C_{2,x})$, for each x. The number of pairing operations is thus $2m$ to calculate $e(H(u), C_{3,x})$. There are also at most m scalar multiplications to calculate $(e(g,g)^{\lambda_x} e(H(u), g)^{\omega_x})^{\sigma_x}$. Therefore, the computation time is $(2m+1)T_p + 5mT_m$, where T_p and T_m are the time taken to perform pairing and scalar multiplication.

Information to be sent from owner to cloud and cloud to user require $m \log |G_T| + 2m \log |G| + m^2 + |Data|$ bits, where $|Data|$ is the size of the data. m^2 bits are needed to transfer the matrix R, and $m(|G_T| + 2|G|) + |G_T|$ to transfer C_0, $C_{1,x}$, $C_{2,x}$ and $C_{3,x}$ and $\log w$, to send π. Thus, the communication overhead is $m^2 + m(|G_T| + 2|G|) + |G_T| + \log w + |Data|$.

TABLE II
COMPARISON OF DACC WITH [17] AND [16]

Schemes	Security	Access policy	Means of Revocation	Size of secret keys	Size of Ciphertext	Computation by user
[17]	No distinct KDC	Any monotonic boolean function	Access control lists	Proportional to number of users	$m \log \lvert G \rvert + \lvert G_T \rvert$	$O(mT_p)$
[16]	$\lvert \mathcal{A} \rvert - 2$	CNF boolean function	Attribute revocation	Independent of number of users	$m \log \lvert G \rvert + \lvert G_T \rvert + m \log m$	$O(mT_p)$
DACC	$\lvert \mathcal{A} \rvert - 1$	Any monotonic boolean function	Attribute revocation	Independent of number of users	$m(2 \log \lvert G \rvert + \lvert G_T \rvert) + m^2$	$O(mT_p)$

When revocation is required, C_0 needs to be recalculated. $e(g, g)$ is previously calculated. So, only one scalar multiplication is needed. If the user revoked is U_u, then for each x, $C_{1,x}$ has to be recomputed. $e(g, g)$ is already computed. Thus, only two scalar multiplication needs to be done, for each x. So a total of $2m' + 1$ scalar multiplications are done by the cloud, where m' is the number of attributes belonging to all revoked users. Users need not compute any scalar multiplication or pairing operations. Additional communication overhead is $O((m' + 1)|G_T|)$.

A. Comparison with other access control schemes

In this section we compare our scheme with [17] and [16]. We study the security of the keys when KDCs are compromised, the type of access policy supported by the schemes, means of revocation, size of ciphertext and secret keys given to users and the computation overheads. We have already proved that ciphertexts cannot be decrypted by the cloud. The scheme is also secure unless all the KDCs get corrupted. From Table II, we see that the main drawback of [17] is that the size of the secret key is very large. The main drawback of [16] is that the access policies are restrictive.

VII. CONCLUSION AND FUTURE WORK

In DACC, the cloud is assumed to be honest. If this requirement is not possible to satisfy, then care should be taken, that the cloud does not modify the data that it contains. Under such circumstances, the authenticity of the data must be verified by the users. Also, it may be very important to hide the identity of the users and owners, at the same time provide their authentication. This can be achieved by means of attribute based signature scheme [36], which we leave as a future work. In DACC, the cloud learns the access structure used by the owner (because the corresponding matrix R is transmitted as part of message C without encoding), and the attributes of the users. In future we would like to hide the access structure from the cloud, by scrambling the matrix in some way. Another challenging problem would be to hide the user attributes from the cloud.

REFERENCES

[1] "Google apps," http://www.google.com/apps.
[2] "Microsoft online," http://www.microsoft.com/online.
[3] "Amazon ec2," http://aws.amazon.com/ec2/.
[4] "Eucalyptus," http://www.eucalyptus.com/.
[5] "Nimbus," http://www.nimbusproject.org/.
[6] "Amazon s3," http://aws.amazon.com/s3/.
[7] "Windows azure," http://www.microsoft.com/windowsazure/.
[8] H. Li, Y. Dai, L. Tian, and H. Yang, "Identity-based authentication for cloud computing," in *CloudCom*, ser. Lecture Notes in Computer Science, M. G. Jaatun, G. Zhao, and C. Rong, Eds., vol. 5931. Springer, 2009, pp. 157–166.
[9] A.-R. Sadeghi, T. Schneider, and M. Winandy, "Token-based cloud computing," in *TRUST*, ser. Lecture Notes in Computer Science, A. Acquisti, S. W. Smith, and A.-R. Sadeghi, Eds., vol. 6101. Springer, 2010, pp. 417–429.
[10] C. Gentry, "A fully homomorphic encryption scheme," Ph.D. dissertation, Stanford University, 2009, http://www.crypto.stanford.edu/craig.
[11] G. Wroblewski, "General method of program code obfuscation," Ph.D. dissertation, Wroclaw University of Technology, 2002, http://www.ouah.org/wobfuscation.pdf.
[12] D. X. Song, D. Wagner, and A. Perrig, "Practical techniques for searches on encrypted data," in *IEEE Symposium on Security and Privacy*, 2000, pp. 44–55.
[13] R. Curtmola, J. A. Garay, S. Kamara, and R. Ostrovsky, "Searchable symmetric encryption: improved definitions and efficient constructions," in *ACM Conference on Computer and Communications Security*, A. Juels, R. N. Wright, and S. D. C. di Vimercati, Eds. ACM, 2006, pp. 79–88.
[14] J. Li, Q. Wang, C. Wang, N. Cao, K. Ren, and W. Lou, "Fuzzy keyword search over encrypted data in cloud computing," in *INFOCOM*. IEEE, 2010, pp. 441–445.
[15] S. Kamara and K. Lauter, "Cryptographic cloud storage," in *Financial Cryptography Workshops*, ser. Lecture Notes in Computer Science, R. Sion, R. Curtmola, S. Dietrich, A. Kiayias, J. M. Miret, K. Sako, and F. Sebé, Eds., vol. 6054. Springer, 2010, pp. 136–149.
[16] M. Li, S. Yu, K. Ren, and W. Lou, "Securing personal health records in cloud computing: Patient-centric and fine-grained data access control in multi-owner settings," in *SecureComm*, ser. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, S. Jajodia and J. Zhou, Eds., vol. 50. Springer, 2010, pp. 89–106.
[17] S. Yu, C. Wang, K. Ren, and W. Lou, "Attribute based data sharing with attribute revocation," in *ASIACCS*, D. Feng, D. A. Basin, and P. Liu, Eds. ACM, 2010, pp. 261–270.
[18] A. B. Lewko and B. Waters, "Decentralizing attribute-based encryption," in *EUROCRYPT*, ser. Lecture Notes in Computer Science, K. G. Paterson, Ed., vol. 6632. Springer, 2011, pp. 568–588.
[19] A. Sahai and B. Waters, "Fuzzy identity-based encryption," in *EUROCRYPT*, ser. Lecture Notes in Computer Science, R. Cramer, Ed., vol. 3494. Springer, 2005, pp. 457–473.
[20] A. Shamir, "Identity-based cryptosystems and signature schemes," in *CRYPTO*, 1984, pp. 47–53.
[21] V. Goyal, O. Pandey, A. Sahai, and B. Waters, "Attribute-based encryption for fine-grained access control of encrypted data," in *ACM Conference on Computer and Communications Security*, 2006, pp. 89–98.
[22] J. Bethencourt, A. Sahai, and B. Waters, "Ciphertext-policy attribute-based encryption," in *IEEE Symposium on Security and Privacy*. IEEE Computer Society, 2007, pp. 321–334.
[23] M. Chase, "Multi-authority attribute based encryption," in *TCC*, ser. Lecture Notes in Computer Science, vol. 4392. Springer, 2007, pp. 515–534.

[24] M. Chase and S. S. M. Chow, "Improving privacy and security in multi-authority attribute-based encryption," in *ACM Conference on Computer and Communications Security*, 2009, pp. 121–130.

[25] S. Ruj, A. Nayak, and I. Stojmenovic, "Improved access control mechanism in vehicular ad hoc networks," in *ADHOC-NOW*, ser. Lecture Notes in Computer Science, H. Frey, X. Li, and S. Rührup, Eds., vol. 6811. Springer, 2011, pp. 191–205.

[26] G. Wang, Q. Liu, and J. Wu, "Hierarchical attribute-based encryption for fine-grained access control in cloud storage services," in *ACM Conference on Computer and Communications Security*, E. Al-Shaer, A. D. Keromytis, and V. Shmatikov, Eds. ACM, 2010, pp. 735–737.

[27] J. Horwitz and B. Lynn, "Toward hierarchical identity-based encryption," in *EUROCRYPT*, ser. Lecture Notes in Computer Science, L. R. Knudsen, Ed., vol. 2332. Springer, 2002, pp. 466–481.

[28] S. Ruj, A. Nayak, and I. Stojmenovic, "Distributed fine-grained access control in wireless sensor networks," in *IPDPS*. IEEE, 2011, pp. 352–362.

[29] A. Beimel, *Secure Schemes for Secret Sharing and Key Distribution. Ph D Thesis*. Technion, Haifa, 1996.

[30] "http://crypto.stanford.edu/pbc/."

[31] V. S. Miller, "http://crypto.stanford.edu/miller/miller.pdf."

[32] D. Freeman, M. Scott, and E. Teske, "A taxonomy of pairing-friendly elliptic curves," *J. Cryptology*, vol. 23, no. 2, pp. 224–280, 2010.

[33] S. Yu, K. Ren, and W. Lou, "FDAC: Toward fine-grained distributed data access control in wireless sensor networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 22, no. 4, pp. 673–686, 2011.

[34] D. R. Stinson, *Cryptography: Theory and Practice, Third Edition*. CRC Press Inc., Boca Raton, 2006.

[35] "Secure shell protocol," http://tools.ietf.org/html/rfc4252.

[36] H. K. Maji, M. Prabhakaran, and M. Rosulek, "Attribute-based signatures," in *CT-RSA*, ser. Lecture Notes in Computer Science, A. Kiayias, Ed., vol. 6558. Springer, 2011, pp. 376–392.

ACKNOWLEDGEMENTS

This work was partially supported by NSERC CRDPJ 386874 - 09 (Reliable and secure QoS routing and transport protocols for mobile ad hoc networks), and NSERC STPSC 356913-2007 (Maintaining fault-tolerant networks of robots for supporting wireless sensor networks).

Ivan Stojmenovic is also associated with the Department of Electronic, Energetics and Telecommunications, FTN, University of Novi Sad, Serbia. This research is also supported by the following grant: "Innovative electronic components and systems based on inorganic and organic technologies embedded in consumer goods and products," TR32016, Serbian Ministry of Science and Education.

APPENDIX

```
Subroutine vector(T: tree);

Input: tree T, a Boolean formula
in the form of a binary tree
(each node has 0 or 2 children)
with AND and OR as nodes

Output: Returns a vector v(x) for each node
x of T, with all vectors of same length

Notes: In implementation,
T is a record containing T.node
(possible values are: AND, OR),
T.left, T.right, T.L, T.vector; so v(x)=T.vector
{
L=0  /* length of vector
maxL=0  /*maximal length of vector
IF T is a leaf  /*T.left=NIL; x=T;
THEN RETURN T.vector= empty
/* v(x)=T.vector
ELSE
CHILDV(T, maxL);
Padding(T, maxL)
}

Function Padding(T, maxL) {
IF L< maxL
{ add maxL-L 0's at the end of v(x); L=maxL};
IF x is not a leaf
{
Padding(T.left, maxL);
Padding(T.right, maxL)
}
}

Function CHILDV(T, maxL)  {
IF T.node=OR
{
ORCHILD(T.left, T.vector, maxL);
ORCHILD(T.right, T.vector, maxL)
}
IF T.node=AND
{
ANDLEFTCHILD(T.left, T.vector, maxL);
ANDRIGHTCHILD(T.left, T.vector, maxL)
}
}

Function ORCHILD(X:tree, Y, maxL)  {
X.vector=Y /* returns same vector before
padding for the root node
CHILDV(X, maxL)
}

Function ANDCHILDLEFT(X:tree, Y, maxL) {
X.L= X.L+1  /* length of v(x) increases by 1
IF X.L> maxL
{
maxL=X.L  /* increases maxL if needed
X.vector = Y | 1  /* adds 1 at
the end of vector and returns it
}
CHILDV(X, maxL)
}

Function ANDCHILDRIGHT(X;tree, Y, maxL) {
X.L= X.L+1;
IF X.L> maxL
maxL=X.L;
X.vector=(0^{X.L-1},-1)/*X.L-1 0's before -1
CHILDV(X, maxL)
}
```