Privacy Preserving and Delegated Access Control for Cloud Applications

Xinfeng Ye*

Abstract: In cloud computing applications, users’ data and applications are hosted by cloud providers. This paper proposed an access control scheme that uses a combination of discretionary access control and cryptographic techniques to secure users’ data and applications hosted by cloud providers. Many cloud applications require users to share their data and applications hosted by cloud providers. To facilitate resource sharing, the proposed scheme allows cloud users to delegate their access permissions to other users easily. Using the access control policies that guard the access to resources and the credentials submitted by users, a third party can infer information about the cloud users. The proposed scheme uses cryptographic techniques to obscure the access control policies and users’ credentials to ensure the privacy of the cloud users. Data encryption is used to guarantee the confidentiality of data. Compared with existing schemes, the proposed scheme is more flexible and easy to use. Experiments showed that the proposed scheme is also efficient.

Key words: cloud computing; access control; security

1 Introduction

In cloud computing, cloud providers host the data and applications for cloud users. Data encryption has been used\[^{[1, 2]}\] to ensure the confidentiality of the data stored on cloud providers. However, a cloud provider not only stores users’ data, it also hosts the applications that its users execute. Thus, using mechanisms to control the access to these applications and data is another approach of securing users’ assets hosted by cloud providers. In this paper, data and applications are called digital assets or assets in short.

Fine-grained access control has been used in cloud computing\[^{[3, 4]}\]. In fine-grained access control schemes, an access control policy is created for each data item. When a data item is accessed, the cloud providers carry out policy enforcement according to the access control policy of the data item. A user can only access a data item if the user’s credentials satisfy the data item’s access policy.

Many cloud applications, e.g., cloud manufacturing\[^{[5]}\], require close collaboration of the users. This means that a user’s assets hosted by a cloud provider need to be accessed by other users. For this type of applications, the access control scheme should allow the delegation of access permission. Access permission delegation means a user, say Alice, delegates her access permission on a data item or an application to another user, say Bob. This allows Bob to carry out operations on the data item or the application on behalf of Alice.

Xu et al.\[^{[6]}\] and Liu and Zic\[^{[7]}\] proposed schemes that allow access permission delegation. However, both schemes do not consider delegating access permissions on the data items that are jointly owned by multiple users. In practice, joint ownership frequently occurs. For example, when two companies jointly develop a product, the data concerning the product is jointly owned. This means that a contractor working on the product must be vetted by both companies. Therefore, it is important to develop an access control scheme

---

* Xinfeng Ye is with the Department of Computer Science, The University of Auckland, Auckland 1142, New Zealand. E-mail: xinfeng@cs.auckland.ac.nz.
* To whom correspondence should be addressed.
Manuscript received: 2015-09-01; accepted: 2015-10-13
that allows the delegation of access permissions in the presence of joint ownership on data items or applications.

An issue with many existing access control schemes, e.g., Refs. [8, 9], etc., is that they do not hide the policies or the credentials that are used in access control. Ye and Khousainov[10] showed that the access control policies and users’ credentials can reveal some secrets of the data’s owners and the holders of the credentials. Using meaningless names for the attributes in policies and credentials does not always solve the problem. This is because, with sufficient amount of access control rules and credentials, it is possible to infer the meaning of the attributes[11].

Much research has been carried out to ensure the confidentiality of data and the privacy of their access control policies[2,12–14]. However, none of these schemes allow access permission delegation; and most of these schemes are not practical due to usability limitations[15].

This paper proposed an access control scheme for cloud applications. The scheme uses a fine-grained attribute-based access control approach and allows the delegation of access permissions to be carried out easily in the presence of joint ownership of digital assets. The scheme guarantees both the confidentiality of data and the privacy of the access control policies and the credentials used for access control. Compared with existing schemes, the proposed scheme is more flexible and practical to use.

This paper is organized as below. Section 2 describes some concepts of cryptography. Section 3 shows the details of the proposed scheme. Section 4 measures the execution time of the scheme. Comparisons with existing works and conclusions are given in Sections 5 and 6, respectively.

2 The Basics of Cryptography

This paper uses the cryptosystem developed in our earlier work[10] for hiding the contents of the access control rules and the credentials. The study in Ref. [10] is based on the RSA algorithm[16] and the scheme by Ray et al.[17] This section introduces some concepts of cryptography. The proof of Theorem 1 can be found in Ref. [10].

Definition 1 Two integers, \(a\) and \(b\), are relatively prime if their greatest common divisor is 1. That is, \(gcd(a, b) = 1\).

Definition 2 Euler’s totient function \(\varphi(N)\) is defined as:

\[
\begin{cases}
\text{if } N \text{ is prime,} & \\
\varphi(N) = N - 1; & \\
\text{if } N = N_1 N_2 \cdots N_k \text{ and} & \\
\forall i, j : [1..k], N_i \text{ and } N_j \text{ are relatively prime,} & \\
\varphi(N) = \varphi(N_1) \varphi(N_2) \cdots \varphi(N_k). & 
\end{cases}
\]

Definition 3 A key \(K\) is a pair \(\langle e, N \rangle\), where \(N\) is a product of distinct primes and \(e\) is relatively prime to \(\varphi(N)\); \(e\) is the exponent and \(N\) is the base of key \(K\).

Definition 4 The encryption of a message \(m\) with key \(K = \langle e, N \rangle\), denoted as \([m, K]\), is defined as

\[m, \langle e, N \rangle\rangle = m^e \mod N.
\]

Definition 5 The matching key of key \(K = \langle e, N \rangle\), denoted as \(K^{-1}\), is a pair \(\langle d, N \rangle\), satisfying \(ed \equiv 1 \mod \varphi(N)\) where “\(\equiv\)” is the congruence modulo relation. \(K\) can decrypt the message encrypted using \(K^{-1}\), and vice versa. That is,

\[[m, K], K^{-1}] = [m, K^{-1}], K = m.
\]

In the RSA cryptosystem, a pair of matching keys is called a public/private key pair.

Definition 6 Two keys \(K_1 = \langle e_1, N_1 \rangle\) and \(K_2 = \langle e_2, N_2 \rangle\) are compatible if \(e_1 = e_2\) and \(N_1\) and \(N_2\) are relatively prime.

Definition 7 If two keys \(K_1 = \langle e, N_1 \rangle\) and \(K_2 = \langle e, N_2 \rangle\) are compatible, then the product key, \(K_1 \times K_2\), is defined as \(\langle e, N_1 N_2 \rangle\).

Theorem 1 Let \(K_i = \langle e, N_i \rangle\) where \(1 \leq i \leq n\) be compatible keys. For any message \(m\) such that \(m \leq N_i\),

\[[m, K_1 \times K_2 \times \cdots \times K_n], K^{-1} = m,
\]

where \(K^{-1}\) is the matching key of key \(K\) and \(K = K_{x_1} \times K_{x_2} \times \cdots \times K_{x_p}\) such that \(1 \leq x_i \leq n, 1 \leq i \leq p,\) and \(x_i \neq x_j\) if \(i \neq j\).

In Theorem 1, \(K\) is a key that is formed by a subset of the keys in set \(K_1, K_2, \cdots, K_n\). Theorem 1 states that, if a message is encrypted using a product key that is formed with all the keys in \(K_1, K_2, \cdots, K_n\), then the matching key of \(K\), i.e., \(K^{-1}\), can be used to decrypt the encrypted message. For example, assume that \(K_1, K_2,\) and \(K_3\) are compatible keys, a message encrypted with product key \(K_1 \times K_2 \times K_3\) can be decrypted using any one of the keys in set:

\[
\left\{ \frac{1}{K_1}, \frac{1}{K_2}, \frac{1}{K_3}, (K_1 \times K_2)^{-1}, (K_1 \times K_3)^{-1}, \right\}
\]
3 The Scheme

3.1 An overview of the scheme

The proposed scheme uses a fine-grained attribute-based access control approach. In fine-grained attribute-based access control, a set of access control rules specifies the conditions under which access to a digital asset is granted. The rules are defined in terms of the attributes that a user might possess, e.g., radar designer, etc.

Cloud providers store their users’ digital assets. Each asset has an access control list containing rules that allow users to access the asset based on the attributes possessed by the users. An access control list specifies three access modes: read, write, and execute. Each mode has a set of access control rules. A user may access an asset in a given mode if the user satisfies the access control rule for that access mode.

The owner of an asset delegates the access permission of the asset to other users by setting the access control rule for each of the access modes of the asset. A user that satisfies the access control rule of an asset is called a delegate. For each access mode, the owner also specifies whether a delegate can further delegate her access permission to other users. If further delegation is allowed, the delegates can delegate their permissions to other users by specifying their own access control rules. Thus, a chain of delegation can be formed for each access mode of the asset. The users high up in the chain can revoke the delegations to the users lower down in the chain. For example, Alice specified access control rules for read and write operations on her asset respectively. Alice also indicated that the read operation can be further delegated while the write operation cannot. Assume that Bob satisfies both Alice’s access control rules. Bob can delegate his read permission on Alice’s asset by specifying his own access control rule while he cannot delegate his write permission on Alice’s asset.

The users that have access permission on an asset are called the authorized users. Each authorized user defines her own set of attributes, and assigns the attributes to the users that she wants to delegate access permission. An authorized user issues credential certificate to her delegates. A credential certificate states the attributes that an authorized user assigned to her delegate. A user can acquire attributes from multiple authorized users. As a result, the user will be issued multiple credential certificates. An authorized user stores her delegates’ credential certificates on cloud providers.

It is assumed that the cloud providers are honest but curious. They honestly execute the access control scheme. When a user, say Bob, wants to operate on an asset hosted by a cloud provider, say cp, cp decides whether the operation can be carried out by checking Bob’s credentials (i.e., attributes) against the access control rules of the asset for the given operation.

To prevent a third party from inferring information from the access control rules and credentials, access control rules and credentials should not be stored in clear text. In order to obscure the access control rules and credentials, the cryptographic system developed in Ref. [10] is used. A pair of matching keys is used to encrypt and decrypt information for the purpose of access control. The access control rules of an asset are converted to a set of decryption keys called rule keys. The credentials of a user are converted to an encryption key called credential key. If and only if the credentials satisfy the access control rules, the credential key and one of the rule keys form a matching key pair. That is, the information encrypted by the credential key can be decrypted by the rule key. Thus, when a cloud provider, say cp, checks whether the credentials of a user, say Bob, satisfies the access control rule of a digital asset, say DA, cp uses Bob’s credential key to encrypt a random string. If the encrypted string can be decrypted correctly using DA’s rule key, it means that the access control rules, the credential key and the rule key (i.e., the access control rule of DA) form a matching key pair. That is, Bob’s credentials satisfy DA’s access control rule. Otherwise, it means the two keys do not form a matching key pair. That is, Bob’s credentials do not satisfy DA’s access rule. Since the keys are sequences of bytes, even if the cloud provider knows the keys, the provider does not understand the contents of the rule or the credentials. Thus, the privacy of the access control rule and the credentials is ensured.

The conversions of the rules and credentials to keys are carried out by authorized users when they create their access control rules and assign attributes to their delegates. Thus, only the authorized users know (a) the mapping between the access control rules of the digital assets and the rule keys, and (b) the mapping between the users’ credentials and their credential keys.

To ensure the confidentiality of digital assets, data encryption and program obfuscation can be used. Obfuscation can make applications hard to understand.
by human\textsuperscript{[18]}. This paper only investigates using encryption to secure the data stored on the cloud provider.

Each user has a pair of public/private keys. The public keys are kept by the cloud provider and the users hold their private keys. A data owner encrypts a data item using a symmetric-key encryption algorithm before the data is stored on the cloud provider. The secret key used to encrypt/decrypt the data item is encrypted using the public keys of the users that have permissions to access the data. The encrypted secret keys are stored on the cloud providers. If a user satisfies an access control rule of a data item, the user will be given the encrypted data item and the encrypted secret key. The user decrypts the encrypted secret key using her private key, and obtains the data item by decrypting it using the secret key. Since the data and the secret keys are encrypted, the cloud providers cannot read the data or the secret keys in clear text. Thus, the confidentiality of the data is ensured.

A user needs to authenticate with the cloud provider before accessing an asset. It is assumed that a public key authentication scheme\textsuperscript{[19]} is used to establish the identity of each user.

3.2 Access control rules

The access control rule given by an authorized user is represented as a logic expression in disjunctive normal form. For example, if a rule states that an asset can only be read by users who have attributes $A$ and $B$ or attributes $A$ and $C$, the logic expression representing the rule is \((A \land B) \lor (A \land C)\). If the attributes possessed by a user satisfy one of the disjuncts in the expression, the user satisfies the rule.

The attributes used in the access control rules of an authorized user are defined by the authorized user. There is no need for the authorized users to agree on the meaning of the attributes defined by them. This is because the authorized users do not need to understand each other’s access control rules. In practice, people from different industries or disciplines might work together on a project. For example, a manufacture and a marketing company might set up a joint venture to produce and sell a product. The manufacturer and the marketing company are likely to use different terminologies. For the authorized users, the freedom of using their self-defined attributes in specifying the access control rules makes the scheme flexible and easy for the authorized users to use as they do not need to adopt attributes that they are not familiar with.

3.3 Joint ownership

An asset might be jointly owned by multiple users $u_i$ \((1 \leq i \leq n)\). For example, two companies are working together to make a product. The information and the data for the product are owned by both companies.

The co-owners of an asset set their access control rules for the asset independently. The co-owners must agree on the number of co-owners’ access control rules that a user must satisfy in order to be granted access permission of the asset. For example, if Alice, Bob, and Carol jointly own an asset, they might specify that a user must satisfy the rules of at least two co-owners in order to be granted access permission of the asset. An $m$-out-of-$n$ access control rule can be represented as:

\[
\bigwedge_{i=1}^{m} \left( S \in \mathcal{P}(\cup_{1 \leq i \leq n} \{r_i\}) \land (|S|=m) \right), \quad r_i \in S
\]

In the formula above, $r_i$ is the access control rule given by co-owner $u_i$, $\mathcal{P}(\cup_{1 \leq i \leq n} \{r_i\})$ represents the set of the rules given by all the co-owners, $|S|$ denotes the cardinality of set $S$.

The $m$-out-of-$n$ access control rule has two special cases. They are (a) the access control rules of all co-owners must be satisfied (i.e., $m = n$), and (b) only one co-owner’s access control rule needs to be satisfied (i.e., $m = 1$). For the two special cases, the degenerate forms of the formula are $\bigwedge_i r_i$ (i.e., $m = n$) and $\bigvee_i r_i$ (i.e., $m = 1$), respectively.

3.4 Delegation

The owner of an asset specifies the original access control rule for various access modes of the asset. The users that satisfy the access control rules can carry out the corresponding operations on the asset. These users are the delegates of the owner as well as the authorized users as they satisfy the owner’s access control rules. If the owner of the asset allows the access permission to be delegated, the authorized users can delegate their permissions by specifying their own access control rules for the asset. An authorized user can also specify whether her delegates can delegate their access permission.

For example, assume that (a) Alice created a file and specified that users with attribute $A$ can read the file and delegate their read permission, (b) $A$ is an attribute issued by Alice, and (c) Bob and Carol both have been
given attribute $A$ by Alice. Thus, Bob and Carol can read the file. Bob might delegate his read permission by specifying a rule allowing users with attribute $B$ (issued by Bob) to read the file. Similarly, Carol might delegate her permission by stating a rule allowing the users with attribute $C$ (issued by Carol) to read the file. Apart from specifying their access control rules, Bob and Carol also indicate whether their delegates can further delegate their read permission. Assume that Bob allows his delegates to delegate their read permission while Carol does not. If Jimmy is given attribute $B$ by Bob and Susan is assigned attribute $C$ by Carol, Jimmy and Susan will be able to read the file. Jimmy can also delegate his read permission to other users by specifying his own access control rules. Susan is not able to delegate her read permission to others as Carol does not allow her delegates to do so.

In the presence of permission delegation, the access control rules for each access mode of an asset are organized as a delegation tree with the asset owner’s rule stored at the root of the tree. The children of a node are the rules of the delegates of the node. The rules of the users in the above example are stored in the tree shown in Fig. 1. The root of the tree stores the rules given by Alice (i.e., the owner of the file). Bob’s and Carol’s rules are stored as the children of the root of the tree since Bob and Carol need to satisfy the rules given by Alice. Bob and Carol delegate their permissions to Jimmy and Susan, respectively. Jimmy only needs to satisfy Bob’s rule. Hence, Jimmy’s rule is stored below Bob’s rule. For the same reason, Susan’s rule is stored below Carol’s rule. Jimmy can have child nodes representing Jimmy’s delegates. Susan must be a leaf node as Carol does not allow Susan to delegate her read permission.

For jointly owned assets, the root of the delegation tree stores the combined access control rule of the co-owners, i.e., $\bigvee \{S \in \mathcal{P}(\cup_{1 \leq i \leq n}(r_i)) : \mathcal{L}(S) = m \} \land (r_i \in S, r_j)$, where $r_i$ is the rule set by co-owner $u_i$ as explained in Section 3.3. For example, Fig. 2 shows the delegation tree of an access mode of a file that was jointly created by Alice and Ted. It is assumed that (a) Alice and Ted require a user must satisfy both Alice’s and Ted’s policy to gain read permission on the file, (b) Alice and Ted allow the read permission to be further delegated by their delegates, and (c) Bob and Carol both satisfy Alice’s and Ted’s policy. In Fig. 2, the root of the tree stores the logical conjunction of the rules given by Alice and Ted since both Alice’s and Ted’s rule must be satisfied.

The cloud provider is responsible for maintaining the delegation trees. It constructs the logic expression representing the $m$-out-of-$n$ access control rule according to the requirements given by the co-owners of an asset. It only allows a user to create her own access control rules if the parent of the user in the delegation tree indicates that the user can delegate her access permission.

### 3.5 Distributing public keys

Each user has a pair of public/private keys. The keys are used for authentication, access control, and ensuring the confidentiality of data items. Since not every user is willing to obtain a X.509 certificate from a certificate authority, the proposed scheme relies on a chain of trust to distribute users’ public keys.

When the owner of an asset, say Alice, signs a contract with a cloud provider for hosting her assets, Alice generates a pair of public/private keys. Alice gives her public key to the cloud provider. The cloud provider uses a public key authentication scheme to establish the identity of each user. Thus, Alice’s public key will be used by the cloud provider to authenticate Alice.

When Alice assigns an attribute to another user, say Bob, Bob generates a pair of public/private keys and gives his public key to Alice. Alice passes Bob’s public key to the cloud provider. As the cloud provider has Alice’s public key, Alice can be authenticated by the cloud provider using a public key authentication mechanism. Thus, the cloud provider is sure that the information received from Alice is sent by Alice. Hence, the cloud provider can be certain that the key given by Alice does belong to a person that Alice
regards as Bob. Since the cloud provider carries out access control on behalf of Alice, as long as Alice is satisfied that the public key belongs to Bob, it is sufficient for the cloud provider to use the public key to identify Bob in the public key authentication mechanism.

If Bob delegates his access permission to another user, say Carol, Carol gives her public key to Bob. Bob passes Carol’s public key to the cloud server. As the cloud provider has Bob’s public key, it can authenticate Bob. Thus, the provider can trust that the key given by Bob does belong to a person that Bob regards as Carol. Similarly, Carol can give her delegates’ public keys to the cloud provider. There is no limit on the depth of the access permission delegation. It can be seen that a chain of trust, i.e., “cloud provider → Alice → Bob → Carol” in the above example, is formed while the delegation of permission is carried out. Using this chain, the cloud provider can collect the public keys of all the users.

### 3.6 Obscuring credentials

When an authorized user, say Alice, delegates her access permission to other users, Alice needs to specify an access control rule in terms of the attributes that she defines. Alice assigns the attributes to the users to whom she wants to delegate her permission. Authorized users define their own attributes. They do not need to coordinate with each other in defining their attributes. This makes it easier for the users to use the system.

An authorized user might give multiple attributes to a delegate. The delegates credential should include all these attributes. An authorized user issues credential certificates to her delegates and gives all the credential certificates to the cloud provider. Since the certificates are sent by the delegator to the cloud provider directly, they cannot be tampered by the delegates. Thus, the cloud provider can be assured of their authenticity.

In order to obscure the credentials, for each attribute defined by an authorized user, the authorized user creates an attribute key that conforms to Definition 3 in Section 2. An attribute key is used to represent the attribute possessed by a user. For an authorized user, all the attribute keys generated by the authorized user have the same exponent, different bases, and the bases are relatively prime to each other. Thus, according to Definition 6 in Section 2, the attribute keys generated by one authorized user are compatible with each other.

Algorithm 1 is used by an authorized user to generate an attribute key. In Algorithm 1, since $p_1$ and $p_2$ are different from all the primes in uprimes, $p_1 \times p_2$ must be relatively prime to the product of any two primes in uprimes. Thus, according to Definition 6, the new key must be compatible with all the existing keys.

If a user, say Bob, is given a single attribute by an authorized user, say Alice, the attribute key that corresponds to Bob’s attribute is used as Bob’s credential. If Alice assigns several attributes to Bob, Bob’s credential is represented by the product key that is formed by the keys corresponding to each of Bob’s attributes. For example, if (a) Alice has assigned attributes $A_1, A_2, \cdots, A_n$ to Bob, and (b) $K_1, K_2, \cdots, K_n$ are the corresponding attribute keys of $A_1, A_2, \cdots, A_n$, respectively, Bob’s credential is represented as $K_1 \times K_2 \times \cdots \times K_n$. The product key is a pair $(e, N)$ where $N = N_1 N_2 \cdots N_n$ and $K_i = \langle e, N \rangle$ $(1 \leq i \leq n)$.

Only Alice knows how to map an attribute defined by her to its corresponding attribute key. Hence, the key representing the credential of a user cannot be easily linked back to any attribute by Bob or any third party, e.g., the cloud provider. That is, only Alice understands the meaning of the credentials that she assigns to other users. Hence, the meanings of the user’s credentials are kept secret.

### 3.7 Obscuring access control rules

As described in Section 3.2, the access control rule given by a user is represented in a disjunctive normal form, e.g., “$(A \land B) \lor (A \land C)$”. The rule is given to the cloud provider for enforcing access control on the
user’s asset. In order to make the access control rule incomprehensible to the cloud provider, each disjunct in a rule is mapped to a key, i.e., a sequence of bytes.

As described in Section 3.6, each attribute is mapped to an attribute key. Using the attribute keys, a product key corresponding to each disjunct in an access control rule can be obtained. The product key is generated using the keys of the attributes in the disjunct. For example, for “\((A \land B)\)”, A’s and B’s attribute keys (i.e., \(K_A\) and \(K_B\)) are used to generate the product key \(K_A \times K_B\). Once the product key is obtained, \((K_A \times K_B)\)’s matching key, i.e., \((K_A \times K_B)^{-1}\), is calculated. \((K_A \times K_B)^{-1}\) is called a rule key. A rule key is used to represent the corresponding disjunct in an access control rule. Thus, a rule will be converted into several rule keys. For example, rule “\((A \land B) \lor (A \land C)\)” is converted into keys \((K_A \times K_B)^{-1}\) and \((K_A \times K_C)^{-1}\).

Algorithm 2 describes how to convert a rule to a set of rule keys. \texttt{AttributeToKey} is a function that maps an attribute to its corresponding attribute key. Each authorized user uses Algorithm 2 to convert her access control rule to a set of rule keys. The keys are given to the cloud provider for rule enforcement.

### 3.8 Rule enforcement

Originally, an access control rule is a logic expression in disjunctive normal form. According to Section 3.7, each rule is converted to a set of rule keys. Each key corresponds to the conditions specified in a disjunct of the logic expression. Thus, as long as one of the keys can be used to determine that a user’s credential matches the corresponding disjunct evaluate to true, the access control rule is satisfied by the user. If an asset is jointly owned by several authorized users, the user must satisfy the access control rules of at least \(m\) authorized users as discussed in Section 3.3.

As explained in Sections 3.5 and 3.6, users’ public keys and credential certificates are stored on the cloud provider. A user that wants to carry out operations on an asset hosted by a cloud provider needs to use a public key authentication mechanism to prove his identity to the cloud provider. The cloud provider can verify a user’s identity using the public key of the user. After a user is authenticated, the cloud provider carries out some encryption and decryption operations using the user’s credentials (i.e., the credential keys contained in the user’s certificates) and the rule keys of the asset that the user wants to access. The outcomes of these operations determine whether the user satisfies the access control rules of the asset.

A rule enforcement example is given below. Assume that (a) a user, say Bob, has been given attributes \(A_1\) and \(A_2\) by Alice and attributes \(T_1\) and \(T_2\) by Ted, (b) Alice and Ted jointly own an asset, (c) “\((A_1 \land A_3) \lor A_2 \lor (A_3 \land A_4)\)” and “\((T_1 \land T_3) \lor (T_1 \land T_2)\)” are the access control rules set by Alice and Ted, respectively, (d) Alice and Ted require users to satisfy both their access control rules in order to access the asset, and (e) the key assigned to attributes \(A_1, A_2, A_3, A_4, T_1, T_2,\) and \(T_3\) are \(K_{A_1}, K_{A_2}, K_{A_3}, K_{A_4}, K_{T_1}, K_{T_2},\) and \(K_{T_3}\), respectively. To access the asset, Bob needs to satisfy both Alice’s and Ted’s rules. It can be seen that, in order to satisfy a disjunct in a rule, the attributes that appear in the disjunct must be a subset of the attributes possessed by Bob. According to the assumption, the set of attributes that Bob has been given by Alice and Ted is \(\{A_1, A_2, T_1, T_2\}\). The attributes appearing in

---

**Algorithm 2: Generating rule keys**

\[
\text{ObscureRule}(\text{Rule})
\]

\[
\begin{align*}
\text{Input: } & \text{Rule is the access control rule for an} \\
& \text{access mode of an asset} \\
\text{Output: } & \text{a set of rule keys} \\
\end{align*}
\]

```
// Keys is a set holding the converted rule keys
1. \text{Keys} \leftarrow \emptyset; \\
// an access control rule is in disjunctive normal form
2. \text{let } Rule = t_1 \lor t_2 \lor \cdots \lor t_n \\
// find the rule key for each disjunct in the rule
3. \text{for each } t_i \text{ where } 1 \leq i \leq n \text{ in Rule do } \\
\quad // each disjunct is a conjunction of one or more attributes
\quad // AttrKeys is a set that includes the attribute keys of all the attributes in \(t_i\)
\quad \text{let } t_i = r_1 \land r_2 \land \cdots \land r_m \text{ and }
\quad \text{AttrKeys} = \{\text{AttributeToKey}(r_j) | 1 \leq j \leq m\}
\quad // \(K\) is a product key that corresponds to \(t_i\)
\quad K \leftarrow K_1 \times K_2 \times \cdots \times K_m \\
\quad \text{where } K_i \in \text{AttrKeys} \\
\quad \text{and } K_i \neq K_j \text{ for } i \neq j \\
4. \text{let } \mathbf{K} = (\mathbf{e}, N) \\
// compute \(K\)’s matching key, i.e., the rule key that corresponds to disjunct \(t_i\)
5. \text{compute } K^{-1} \text{ such that }
\quad K^{-1} = (\mathbf{d}, N) \\
\quad \text{where } \mathbf{e} \cdot \mathbf{d} \equiv 1 \text{ mod } \varphi(N) \\
6. \text{let } \text{Keys} = \text{Keys} \cup \{K^{-1}\} \\
7. \text{// end of for each }
8. \text{return } \text{Keys}
```

---
disjunct “(A1 ∨ A3)” of Alice’s access control rule form set \{A_1, A_3\} which is not a subset of \{A_1, A_2, T_1, T_2\}. As Bob does not have attribute A_3, Bob cannot satisfy disjunct “(A1 ∨ A3)” in Alice’s rule. According to Section 3.6, the credential given to Bob by Alice is the product key \(K_{A_1} \times K_{A_2}\). From Section 3.7, \((K_{A_1} \times K_{A_3})^{-1}, K_{A_3}^{-1}\), and \((K_{A_1} \times K_{A_4})^{-1}\) are the rule keys corresponding to disjunct “(A1 ∨ A3)”, “A1”, and “(A3 ∨ A4)”, respectively. To check whether Bob’s credential satisfies Alice’s access control rule, the cloud provider checks whether one of the disjuncts in Alice’s access control rule can be satisfied by Bob’s credential. First, the cloud provider generates a random string \(T\) and encrypts \(T\) using key \(K_{A_1} \times K_{A_2}\) (i.e., Bob’s credential key given by Alice) to obtain ciphered text \(CT\) (i.e., \(CT = [T, K_{A_1} \times K_{A_2}]\)). To test whether Bob’s credential satisfies “(A1 ∨ A3)”, the cloud provider uses key \((K_{A_1} \times K_{A_3})^{-1}\) (i.e., the rule key representing disjunct “(A1 ∨ A3)” to decrypt \(CT\). Since “(K_{A_1} \times K_{A_3} ≠ K_{A_1} \times K_{A_3}”, \((K_{A_1} \times K_{A_3})^{-1}\) is not the matching key of \(K_{A_1} \times K_{A_2}\). As a result, \((K_{A_1} \times K_{A_3})^{-1}\) cannot decrypt \(CT\). Since the decryption fails, it is regarded as Bob does not satisfy the disjunct “(A1 ∨ A3)” that is represented by \((K_{A_1} \times K_{A_3})^{-1}\).

Similarly, when examining whether Bob’s credential satisfies “A2”, the scheme uses rule key \(K_{A_2}^{-1}\) (i.e., the rule key representing disjunct “A2” in Alice’s access control rule) to decrypt \(CT\). Let \(\tilde{K} = K_{A_2}\) in Theorem 1 in Section 2. Thus, “A2 = K_{A_2}^{-1}” holds. According to Theorem 1 in Section 2, “[T, K_{A_1} \times K_{A_3}], K_{A_2}^{-1}] = T” holds. As the decryption is successful, it is regarded that the credential provided by Bob satisfies the disjunct (i.e., “A2”) that is represented by rule key \(K_{A_2}^{-1}\). In a disjunctive normal form, if one of the disjuncts is satisfied, the whole logic expression evaluates to true. Thus, if Bob satisfies A_2, Bob satisfies Alice’s rule. The other disjunct (i.e., “(A3 ∨ A4)”) does not need to be checked. To access the asset, Bob also needs to satisfy Ted’s policy. The same method is used when checking whether Bob’s credential satisfies Ted’s access control rule.

Algorithm 3 checks whether a credential key satisfies the rule given by an authorized user. checknextdisjunct indicates whether there is a need to check the next disjunct in the access control rule expression (line 5). If one disjunct in the rule expression is true (i.e., the rule key can decrypt \(CT\)), the rule is satisfied. Thus, there is no need to check the rest of the disjuncts in the rule expression (lines 7 to 10).

Algorithm 4 defines the rule enforcement process that a cloud provider follows when it checks whether a user can be granted access permission to an asset. An asset might be jointly owned by several authorized users. Argument \(m\) specifies the minimum number of authorized users’ rules that a user needs to satisfy in order to be granted access permission. Rules is a set of rule keys representing all the rules given by various authorized users of an asset. Each element in Rules is the set of rule keys representing the access control rule of one authorized user. numOfSatisfiedRule records the number of access control rules that are satisfied by the user’s credential so far (line 1). The cloud provider checks a user’s credential against the access control rule of each authorized user (line 2). A user might have several credential certificates issued by various authorized users. cert.issuer is the ID of the issuer of a certificate. An access control rule given by an authorized user should be evaluated against the credential issued by the same authorized user. For example, if Bob wants to access an asset whose access control rule is set by Alice, Alice’s rule should be checked against the credential issued to Bob by Alice. Thus, the correct credential certificate needs to be checked against the credential issued to Bob by Alice.
Algorithm 4 Enforcing rule

\textbf{RuleEnforcement}(m, Rules, Certificates)

\begin{algorithmic}
\State \\textbf{Input:} \(m\) is the minimum number of access control rules that the user needs to satisfy.
\State \(Rules\) is a set of rule keys set representing the access control rules of authorized users.
\State \(Certificates\) is a set of credential certificates of the user whose credential is being checked.
\State \\textbf{Output:} \textit{grant} or \textit{deny}.
\State \texttt{// initialise numOfSatisfiedRule}
\State 1. \texttt{numOfSatisfiedRule} \leftarrow 0
\State 2. \texttt{for each rule in Rules do}
\State 3. \texttt{let rule be the access control rule set by user au}
\State 4. \texttt{let cert be a certificate in Certificates such that cert.issuer = au}
\State 5. \texttt{if (cert does not exist) then continue end-if}
\State 6. \texttt{extract credential key key from cert}
\State 7. \texttt{if CredSatisfiesRule(key, rule) then}
\State 8. \texttt{numOfSatisfiedRule \leftarrow numOfSatisfiedRule + 1}
\State 9. \texttt{if numOfSatisfiedRule \geq m then}
\State 10. \texttt{continue}
\State 11. \texttt{else}
\State 12. \texttt{return grant}
\State 13. \texttt{end-if}
\State 14. \texttt{end-if}
\State 15. \texttt{end-for-each}
\State 16. \texttt{return deny}
\end{algorithmic}

RuleEnforcement is invoked when an access control rule is checked (line 4). If the user has not been issued any credential by the authorized user that created the access control rule, the user cannot satisfy the rule. As the user is granted access permission as long as the user can satisfy \(m\) of all the access control rules, failing one rule does not mean that the user should be denied access permission. Thus, the next access control rule is checked (line 5). Otherwise, the credential key is retrieved from the certificate to check whether it satisfies the rule (lines 6 and 7). \texttt{numOfSatisfiedRule} is incremented when an access control rule is satisfied (line 8). Once the user has satisfied sufficient number of access control rules (i.e., \(numOfSatisfied \geq m\)), the user is granted access permission (lines 11 and 12). Otherwise, the access control rule of another authorized user is examined (lines 9 and 10). If the number of access control rules that are satisfied by the user cannot reach the required number (i.e., \(m\)) when all the access control rules have been checked, the access request is declined (line 16).

3.9 Ensuring data confidentiality

Encryption is used to ensure data confidentiality. The proposed scheme uses an encryption mechanism that is similar to the approach by Gonzalez-Manzano et al.\cite{20} The owner of a data item encrypts her data item using a symmetric-key encryption algorithm, e.g., AES. Each data item is encrypted with its own unique secret key. The secret key needs to be given to all the users that have permission to access the data. For each data item, the authorized user asks the cloud provider to find out all her delegates that satisfy her access control rule for the data item. For each delegate that satisfies the access control rule of a data item, the authorized user uses the public key of the delegate to encrypt the secret key that is used to encrypt the data item. The encrypted secret keys are stored with their corresponding data items on the cloud provider. When a user accesses a data item, if the user satisfies the access control rule of the data item, the cloud provider gives the encrypted data item as well as its encrypted secret key to the user. The user decrypts the encrypted secret key using her private key to obtain the secret key. Then, the secret key is used to obtain the plain text of the data item.

Jointly owned data only need to be encrypted by one of the co-owners since the purpose of encrypting a data item is to keep the cloud provider from knowing the clear text of the data. Thus, encrypting the data once would prevent the cloud provider from understanding the content of the data. The co-owners need to reach an agreement on the secret key being used to encrypt the data item. Thus, all the co-owners can decrypt the encrypted data when they need it.

Storing the encrypted secret keys on the cloud providers relieves the authorized users from distributing the encrypted keys to their delegates. As the secret keys are encrypted, the cloud provider cannot find out the clear text of the keys. Hence, it is not able to use the key to decrypt the encrypted data.

Cloud providers use Algorithm 5 to find the delegates that satisfy a user’s access control rule. Algorithm 5 is invoked when the access control rule for an asset is created or changed. It is also applied to each data item of an authorized user when the authorized user assigns a new attribute to her delegate. This is because these changes might result in some users satisfying the access.
Algorithm 5 Finding conforming delegates

\begin{algorithm}
\begin{algorithmic}
\State \textbf{Input:} \textit{au} is the ID of an authorized user
\hspace{2em} \textit{data} denotes a data item
\State \textbf{Output:} a set of users that satisfy \textit{au}'s access control rule for \textit{data}
\begin{enumerate}
\item let \textit{Rule} be the set of rule keys that corresponds to \textit{au}'s access control rule for \textit{data}
\item let \textit{Certificates} be a set of credential certificates issued by \textit{au}
\hspace{2em} // initialize \textit{conformDelegate}
\item \textit{conformDelegate} $\leftarrow \emptyset$
\item \textbf{for each} \textit{cert} in \textit{Certificates} \textbf{do}
\hspace{2em} \textbf{if} \textit{CredSatisfiesRule(}\textit{key, Rule}\textit{)} \textbf{then}
\hspace{4em} \textit{conformDelegate} $\leftarrow \textit{conformDelegate} \cup \{\textit{cert}'s\ holder\}$
\item \textbf{else}
\hspace{2em} remove the encrypted secret key for \textit{cert}'s\ holder
\item \textbf{end-if}
\item \textbf{end-for-each}
\item \textbf{return} \textit{conformDelegate}
\end{enumerate}
\end{algorithmic}
\end{algorithm}

rule of the authorized user.

As explained in Sections 3.6 and 3.7, an authorized user stores her access control rules and the credentials’ certificates that have been issued to other users on the cloud providers. A cloud provider can find out the access control rule set by authorized user \textit{au} (line 1) and all the credential certificates issued by \textit{au} (line 2). Set \textit{conformDelegate} contains the IDs of all the users that satisfy \textit{au}'s access control rule (line 3). Each credential given by \textit{au} is checked against \textit{au}'s access control rule for data (lines 4 and 5). Users satisfying \textit{au}'s access control rule are added to \textit{conformDelegate} (lines 6 and 7). If a user no longer satisfies \textit{au}'s rule, the user cannot access data. Thus, the encrypted secret key that \textit{au} generated for the user (if any) should be deleted (lines 8 and 9). After each of the credentials supplied by \textit{au} is checked, the set of IDs of the users that satisfy \textit{au}'s access control rule is returned to user \textit{au} (line 12).

3.10 Speeding up the access control process

The access control rules and the credentials of the users are stored on the cloud providers. Thus, a user’s permission for accessing an asset can be determined by the cloud providers before the user actually requests to access the asset. This means that, instead of carrying out the rule enforcement operation on the fly, the cloud providers can pre-compute a list of eligible users for each access mode of an asset. When a user wants to carry out an operation on an asset, the cloud provider first verifies the user’s identity using a public key authentication mechanism. If the identity of the user can be established, the cloud provider just needs to look up the list of eligible users to determine whether the user’s access request can be granted. Thus, at run time, the access control process becomes a simple lookup operation without involving the expensive encryption and decryption operations.

3.11 Changing access control rules and credentials

An authorized user, say Alice, might change her access control rule on an asset. After Alice’s rule is changed, it is necessary to check whether Alice’s delegates can satisfy Alice’s new rule. For the delegates that can no longer meet the conditions in Alice’s new rule, their encrypted secret keys for decrypting the data should be deleted by the cloud provider. For Alice’s delegates that have created their own rules on the asset, if a delegate cannot satisfy Alice’s new rule, all the rules that are in the sub-tree rooted at the delegate’s node in the delegation tree should be deleted. For example, in Fig. 1, after Alice changes her rule, it is necessary to check whether Bob and Carol can satisfy Alice’s new rule. If a user, say Bob, no longer satisfies Alice’s new rule, Bob cannot delegate any permission to Jimmy. As a result, Bob’s and Jimmy’s rules are deleted from the delegation tree. The same reasoning applies to Carol.

Algorithm 6 is executed by the cloud provider when a user, say \textit{au}, changes her access control rule. In the algorithm, \textit{c.creator} is the ID of the user that created the rule stored in node \textit{c} of the delegation tree.

An authorized user, say Alice, might change the credentials of her delegates (i.e., assign new attributes or revoke all or some attributes issued to delegates). If Alice revokes all the attributes of a delegate, say Bob, Alice would tell the cloud provider to delete Bob’s credential certificate issued by Alice. Otherwise, Alice would issue a new credential certificate to Bob, and gives the new certificate to the cloud provider.

After being notified by Alice, the cloud provider scans through all the assets for which Alice has set access control rules to find out whether Bob has used his old credential to set his own access control rules on these assets. If Bob’s new credential (if any) cannot satisfy Alice’s rule, Bob’s rule and all the rules below Bob’s node in the delegation tree are deleted. This is
Algorithm 6 Updating delegation tree
\[
\text{PolicyCleanse}(\text{asset}, \text{au})
\]
input: \text{asset} is the asset whose access control rule is changed
\text{au} is the authorized user that sets an access control rule on \text{asset}

1. let \( R \) be the node holding user \( au \)'s access control rule in \( asset \)'s delegation tree, and \( Children \) be the set of \( R \)'s child nodes

2. for each \( c \) in \( Children \) do
   // \( cert \) is the credential certificate that \( au \)
   // issued to delegate \( c.creator \)
   1. let \( cert \) be the credential certificate of user \( c.creator \) such that \( cert.issuer = au \)
   // \( key \) is the credential key of \( c.creator \)
   2. let \( key \) be the encrypted secret key for \( c.creator \)
   3. delete node \( c \)
   4. end-for-each

Algorithm 7 Revalidating delegate
\[
\text{CheckDelegate}(\text{au}, \text{del})
\]
input: \( au \) is an authorized user that has changed some attributes assigned to \( au \)'s delegate \( del \)

1. let \( DA \) be the set of assets that \( au \) has set access rules
2. let \( cert \) be \( del \)'s new credential certificate given by \( au \)
3. for each asset \( da \) in \( DA \) do
4. let \( r_{au} \) be the rule in \( da \)'s delegation tree
5. let \( r_{del} \) be a child node of \( r_{au} \) and \( r_{del} \) stores \( del \)'s access control rule
   // actions to be taken when \( del \)'s new
   // credential cannot satisfy \( au \)'s access
   // control rule
6. if \( (r_{del} \text{ exists}) \lor (\neg \text{CredSatisfiesRule}(cert.key, r_{au}.rule)) \)
   then
   // delete the sub-tree rooted at node \( c \) by
   // calling Algorithm 6
   7. delete \( del \)'s access control rule from \( r_{del} \)
   8. PolicyCleanse(\( da, del \))
   // If \( del \) cannot access the asset, its
   // encrypted secret key need to be
   // removed.
   9. delete the encrypted secret key for \( del \)
10. delete node \( r_{del} \)
11. end-if
12. end-for-each

because Bob can neither access the asset nor delegate any access permission to other users.

Algorithm 7 is used by the cloud provider after a user’s credential is changed. If a user, say \( del \), has not created any access control rule (i.e., if “\( r_{del} \text{ exists} \)” in line 6 of Algorithm 7 is false), the delegation tree is not affected by the change of \( del \)'s credential. If \( del \) has not been given a new certificate (i.e., “\( cert \text{ does not exist} \)” in line 6 is true), it means all the user’s attributes have been revoked. \( cert.key \) in line 6 is \( del \)'s new credential key.

When access control rules and user credentials are changed, the authorized users only need to generate the new rule keys and credential keys. The cloud provider is responsible for checking whether the changes would invalidate some access control rules. The secret keys for encrypting data items are not affected by the changes to rules and credentials. This is because the users affected by the changes still need to go through the rule enforcement process of the cloud provider to obtain the data and the secret keys for decrypting the data. If they can no longer meet the conditions specified in the access control rules of the data, they will be denied the access to the data and the secret key by the cloud provider.

4 Performance Evaluation

A prototype of the proposed scheme has been implemented using Java to evaluate the execution cost of the scheme. The evaluation was carried out on a Dell Latitude E6540 with a 2.7 GHz Intel Core i5-4310M processor, 8 GB memory, and 64-bit Windows 7. In all experiments, (a) the exponent of each key (i.e., \( e \)) is 65,537, and (b) the rule key has a single disjunct (i.e., “\( \text{Attr}_1 \land \text{Attr}_2 \land \cdots \land \text{Attr}_n \)” where \( 1 \leq n \leq 10 \)).

The first experiment measures the time for generating attribute keys using Algorithm 1 in Section 3.6. 10,000 keys were generated. The average time for generating one key of various length is shown in Fig. 3. It can be
seen that the time is comparable with generating a key of the same size in the RSA algorithm. Thus, attribute keys can be generated efficiently.

The second experiment evaluates (a) how the number of attributes possessed by a user affects the cost of creating a credential key, and (b) how the number of attributes appearing in a disjunct of an access rule influences the time for generating a rule key. Two sets of experiments were carried out. In one experiment, the size of each attribute key is set to 256 bits. The length of each attribute key is set to 512 bits in the other experiment. As shown in Fig. 4, the cost of generating a rule key is much higher than generating a credential key. This is because creating a rule key also requires applying the extended Euclidean algorithm to calculate the matching key. However, the longest time observed in the experiment (i.e., there are 10 attributes in a rule key and each attribute key is 512 bits) is only about 3500 μs. Hence, the credential keys and rule keys can be generated efficiently.

The last experiment measures the costs of rule enforcement. The enforcement involves encrypting a 10-byte string using the credential key and decrypting the encrypted string using a rule key. The speeds of the encryption and decryption are affected by the sizes of the keys. The size of a key depends on the number of attributes in a credential or a rule. According to Definition 7 in Section 2, the size of a product key is \( mn \) where \( m \) is the size of each attribute key and \( n \) is the number of attribute keys used to form the product key. In this experiment, the credential key is set to contain 10 attributes while the number of attributes in the rule key varies between 1 and 10.

In the experiments, the size of an attribute key is set to 256 and 512 bits respectively. According to Fig. 5, when the size of an attribute key is 256 bits, the longest time for the enforcement operation is 35 ms. This appears to be reasonable for any on-line application. In the worst case (i.e., there are 10 attributes in the credential key and rule key respectively, and the size of each attribute key is 512 bits), the enforcement time is about 249 ms. As discussed in Section 3.10, the cloud providers can carry out the rule enforcement operations in advance. That is, the cloud provider pre-computes a list of users that are eligible to access an asset. Hence, the rule enforcement operation can be carried out off-line. For an off-line operation, 249 ms seems to be reasonably efficient.

5 Related Work

Nabeel and Bertino[21] proposed a scheme for dividing the access control operations between the data owner and the cloud provider. The scheme divides the access control rules into two sets. One set is only visible to the data owner while the other set is given to the cloud provider. Thus, the access control policies are...
partially hidden from the cloud provider. Different from Ref. [21], the scheme here hides the access control policies from the cloud providers completely by representing the policies in a form that cannot be understood by the cloud providers. Unlike the scheme in Ref. [21], this paper also addressed the issue of delegating access permissions which is important for many cloud applications.

Many approaches that use cryptographic mechanisms to enforce access control rules have been proposed\cite{1, 13, 14}. They usually group data items based on access control rules and encrypt each group with a different symmetric key. Then, users can derive the keys only for the data items they are allowed to access. Like the scheme in Ref. [21], none of these schemes address delegating access permissions which was studied in this paper.

Ye and Khoussainov\cite{10} presented an access control scheme that allows the access control rules and credentials to be stored in an obscured form. Different from the scheme in this paper, Ref. [10] did not allow access permission delegation. Hence, Ref. [10] is not as flexible as the scheme in this paper.

Ray et al.\cite{17} proposed a scheme for controlling the access of files in a hierarchical organisation. The scheme requires the credential keys of the entities at the lower levels of the hierarchy contain the keys of the entities at higher levels of the hierarchy. Thus, a single entity is needed for generating the keys for all the users in the system. However, it might not be practical to find an entity that is trusted by everyone in the system. Hence, the scheme is not well suited for many cloud applications. Unlike Ref. [17], the scheme in this paper allows the users to manage their own keys’ generation. Thus, it is more flexible to use.

Carrying out access control with hidden credentials has been studied by many people\cite{11, 22}. These schemes are based on the identity-based encryption scheme\cite{23}. In these schemes, the access control policies are used as keys to encrypt the data. Only the people who meet the conditions specified in the policies are able to generate the decryption keys. Frikken et al.\cite{12} improved the performance of hidden credential schemes. Li and Li\cite{24} proposed a scheme for hiding the attributes of the identity during a trust negotiation. They used a topologically uniform circuit and a committed-integer based oblivious transfer protocol. All these schemes have very high running cost. For example, Frikken’s scheme needs $O(p^2mn)$ encryption operations and $O(p^2mn)$ communications where $m$ is the number of credentials, $n$ is the number of attributes in a policy, and $\rho$ is the number of bits used to represent the attributes. The cost of running the scheme in this paper is low as it only needs one encryption and one decryption operation, and, it does not require the server and the user to engage in multiple rounds of communications. Unlike the identity-based encryption schemes\cite{12, 24}, the scheme in this paper uses a different approach in generating the credential and rule keys. As a result, the keys used by the proposed scheme can be generated more efficiently.

Baden et al.\cite{25} used attribute-based encryption to ensure the confidentiality of data. They assumed that all the users share the same set of attributes. In practice, this requirement might be difficult to be met. For example, in cloud manufacturing, partners from different industries might need to collaborate on a project. These partners are likely to use different terminologies. Unlike Ref. [25], the scheme proposed in this paper allows users to use their own attributes when defining access control rule. Thus, the scheme in this paper is easier and more flexible for people to use.

She et al.\cite{8, 26} proposed several schemes for controlling the flow of information through a composite service. The schemes do not hide the contents of access control policies. Confidential policies remain on the policy’s creator and access control needs to be carried out by the creator. Different to She’s schemes, the scheme in this paper hides the contents of the policy and allows the policy to be checked by cloud providers. Thus, the proposed scheme incurs less communications between the users and the cloud providers. Hence, the proposed scheme is more efficient.

Trust negotiation has been studied in Refs. [9, 27], etc. To minimize the amount of information disclosed to partners, Winsborough’s scheme requires the partners exchange their credentials in several rounds. Squicciarini’s scheme uses substitution and generalization to minimize and blur the information exchanged between partners. These schemes did not intend to hide the policies or credentials. Different to these schemes, the scheme in this paper completely hides the contents of the policies and credentials. Thus, it provides a higher level of privacy.

The access delegation schemes by Liu and Zic\cite{7} and Xu et al.\cite{6} allow users to delegate their access permission without changing the system configuration.
Both schemes did not consider how to delegate permissions on jointly owned data items. Unlike Refs. [6, 7], the scheme in this paper used an attribute-based access control approach. By setting access control rules, permission delegation for jointly owned assets can be easily handled in this paper.

Gonzalez-Manzano et al. [28] proposed a model for handling access control policies of co-owners of jointly owned objects. Their focus is on decomposing objects and mediating the access control rules of the co-owners to ensure that the requirements of all the co-owners can be satisfied. Unlike the scheme proposed in this paper, the scheme in Ref. [28] does not consider the confidentiality of data nor the privacy of the access control policies.

6 Conclusions

The access control scheme in this paper allows access permission delegation to be carried out easily in a cloud computing environment. In the proposed scheme, cryptography is used to ensure the confidentiality of data, the privacy of the access control rules, and the credentials required for access control. The scheme is flexible and easy to use as it allows users to delegate their access permissions by (a) assigning user self-defined attributes to their delegates and (b) specifying access control rules in terms of the attributes. Access control enforcement and most of the tasks caused by changing access control rules and credentials are carried out by cloud providers. Users’ tasks are limited to generating credential key and policy keys. Compared with the existing schemes, the proposed scheme is simple and efficient to use. The experiments showed that (a) the time for generating an attribute key is comparable to generating an RSA key, and (b) when the size of each attribute key is 256 bits, with no more than ten attributes in the credential or the access control rule, the policy enforcement can be carried out in less than 35 ms. The scheme allows the efficiency of the access control process to be further improved by pre-computing a list of eligible users for each asset.

References

[1] M. Nabeel, N. Shang, and E. Bertino, Privacy preserving policy based content sharing in public clouds, IEEE Transactions on Knowledge and Data Engineering, vol. 25, no. 11, pp. 2602–2614, 2013.

[2] K. P. N. Puttaswamy, C. Kruegel, and B. Y. Zhao, Silverline: Toward data confidentiality in storage-intensive cloud applications, in Proceedings of the 2nd ACM Symposium on Cloud Computing, New York, NY, USA, 2011.

[3] D. Song, E. Shi, I. Fischer, and U. Shankar, Cloud data protection for the masses, Computer, vol. 45, no. 1, pp. 39–45, 2012.

[4] S. Yu, C. Wang, K. Ren, and W. Lou, Achieving secure, scalable, and fine-grained data access control in cloud computing, in Proceedings of the 29th Conference on Information Communications, Piscataway, NJ, USA, 2010, pp. 534–542.

[5] X. Xu, From cloud computing to cloud manufacturing, Robotics and Computer-Integrated Manufacturing, vol. 28, no. 1, pp. 75–86, 2012.

[6] Y. Xu, A. M. Dunn, O. S. Hofmann, M. Z. Lee, S. A. Mehdi, and E. Witchel, Application-defined decentralized access control, in Proceedings of the 2014 USENIX Conference, Berkeley, CA, USA, 2014, pp. 395–408.

[7] D. Liu and J. Zic, User-controlled identity provisioning for secure account sharing, in Proceedings of the 2014 IEEE International Conference on Cloud Computing, Washington, DC, USA, 2014, pp. 644–651.

[8] W. She, I. Yen, B. Thuraisingham, and E. Bertino, The SCIFIC model for information flow control in web service composition, in Proceedings of the 2009 IEEE International Conference on Web Services, Washington, DC, USA, 2009, pp. 1–8.

[9] A. C. Squicciarini, E. Bertino, E. Ferrari, and I. Ray, Achieving privacy in trust negotiations with an ontology-based approach, IEEE Trans. Dependable Secur. Comput., vol. 3, no. 1, pp. 13–30, 2006.

[10] X. Ye and B. Khossainov, Fine-grained access control for cloud computing, Int. J. Grid Util. Comput., vol. 4, no. 2/3, pp. 160–168, 2013.

[11] R. W. Bradshaw, J. E. Holt, and K. E. Seamons, Concealing complex policies with hidden credentials, in Proceedings of the 11th ACM Conference on Computer and Communications Security, New York, NY, USA, 2004, pp. 146–157.

[12] K. Friksen, M. Atallah, and J. Li, Attribute-based access control with hidden policies and hidden credentials, IEEE Trans. Comput., vol. 55, no. 10, pp. 1259–1270, 2006.

[13] M. Harbach, Towards privacy-preserving access control with hidden policies, hidden credentials and hidden decisions, in Proc. of the 10th Annual International Conference on Privacy, Security and Trust, Paris, France, 2012, pp. 17–24.

[14] R. Zhang, L. Liu, and X. Rui, Role-based and time-bound access and management of EHR data, Security and Communication Networks, vol. 7, no. 6, pp. 994–1015, 2014.

[15] H. Takabi, Privacy aware access control for data sharing in cloud computing environments, in Proceedings of the 2nd International Workshop on Security in Cloud Computing, New York, NY, USA, 2014, pp. 27–34.

[16] R. L. Rivest, A. Shamir, and L. Adleman, A method for obtaining digital signatures and public-key cryptosystems, Commun. ACM, vol. 21, no. 2, pp. 120–126, 1978.
Xinfeng Ye gained his BSc degree in computer science from Huaqiao University, China, in 1987, and MSc and PhD degrees in computer science from The University of Manchester, England, in 1988 and 1991, respectively. He is currently a senior lecturer in the Department of Computer Science at The University of Auckland, New Zealand. His current research interests include cloud computing and system security.