Updating Policies in CP-ABE-Based Access Control: An Optimized and Secure Service

Somchart Fugkeaw(✉) and Hiroyuki Sato

Department of Electrical Engineering and Information Systems,
The University of Tokyo, Tokyo, Japan
{somchart, schuko}@satolab.itc.u-tokyo.ac.jp

Abstract. Policy update management is one of the key problems in the ciphertext policy-attribute-based encryption (CP-ABE) supporting access control in data outsourcing scenario. The problem is that the policy is tightly coupled with the encryption itself. Hence, if the policy is updated, the data owner needs to re-encrypt files and sends them back to the cloud. This incurs overheads including computation, communication, and maintenance cost at data owner side. The computation and communication overheads are even more costly if there are frequent changes of access control elements such as users, attributes and access rules. In this paper, we extend the capability of our access control scheme: C-CP-ARBE to be capable to support secure and flexible policy updating in data outsourcing environment. We propose a policy updating method and exploit a very lightweight proxy re-encryption (VL-PRE) technique to enable policies to be dynamically and effectively updated in the cloud. Finally, we demonstrate the efficiency and performance of our proposed scheme through our evaluation and implementation.

1 Introduction

To consider adopting a cloud solution for storing large scales of highly value data, security and privacy are of paramount importance. Existing research works and cloud applications generally deploy encryption techniques and applicable access control model to satisfy the security requirement.

Access Control is among the most effective solutions for full-fledged network security control. Data access control for outsourced data should not only support the security but it should also provide a flexible and efficient management of the policy enforced over a large number of users as well as the optimized cost for handling the change of access control elements such as users, attributes, access policies. Importantly, the access control policy must be up-to-date to support the right and effective control and enforcement. In addition, access control supporting collaborative accesses across the data sources outsourced at the cloud servers is very important.

Attribute-based encryption (ABE) [6] is regarded as an effective solution for formulating a lightweight access control to outsourced data and unknown decrypting parties. To date, several works apply ciphertext attribute-based encryption (CP-ABE) [2–5, 8] for the access control solutions and generally concentrate on minimizing key management cost, reducing computing cost of interaction between data owner and

M. Aiello et al. (Eds.): ESOCC 2016, LNCS 9846, pp. 3–17, 2016.
DOI: 10.1007/978-3-319-44482-6_1

outsourced data storage, improving scalability and efficient revocation. However, these works have not addressed the policy evolution or policy updating problem in their proposed models.

In fact, policy updating is one of the critical administrative tasks to control the most up-to-date access policy enforcement. The policy update in CP-ABE renders the cost of policy update operation, cost of file re-encryption and communication cost for loading the file back to the cloud. All of these costs are usually occurred at data owner's side.

Therefore, in addition to the fine-grained and scalable access control model supporting data outsourced in the cloud, optimizing the policy update is also another grand challenge. For the operational point of view, the issues including correctness, security, and accountability of the subsequent update of policy are the requirements to be provided by CP-ABE policy updating scheme. These requirements are described as follows.

- Correctness: An updated policy must be syntactically correct and the policy updating must support any types of CP-ABE policy boolean. In addition, users who hold the keys containing a set of attributes satisfying the policy are able to decrypt the data encrypted by an updated policy.
- Security: A policy must be updated by the data owner or authorized administrator only in the secure manner and a new policy should not introduce problems for the existing access control.
- Accountability: All policy updating events must be traceable for auditing.

The remainder of the paper is organized as follows. Section 2 discusses related works. Section 3 presents detail of our proposed approach. Section 4 describes the policy updating method and presents concept of our proxy re-encryption scheme. Section 5 gives the evaluation and implementation detail. Finally, the conclusion and future work are depicted in Sect. 6.

2 Related Work

Ciphertext Policy Attribute Based Encryption (CP-ABE) was originally proposed in [7]. In CP-ABE, each user is given a set of attributes, which is embedded into the user's secret key, and a public key is defined for each user attribute. The ciphertext is associated with the access policy structure in which the encryptor can define the access policy by her own control. Users are able to decrypt a ciphertext if their attributes satisfy the ciphertext access structure.

However, policy update in ABE scheme has attracted less attention by existing research works. In [13], the authors introduced a ciphertext delegation method to update the policy of ciphertext in attribute-based access control. Their method aimed at solving user revocation based on a re-encryption delegation technique to protect newly encrypted data. Nevertheless, the performance on updating the ciphertext over the complex access policy was not examined by the authors.

Recently, Yang et al. [3, 9] proposed a method to outsource a policy updating to the cloud server. They proposed policy updating algorithms for adding and removing attributes in the AND, OR, and threshold gate of LSSS policy. The proposed scheme is

to update ciphertext in order to avoid file re-encryption. Cost for ciphertext update is also linear to the number of attributes updated over the access structure. Besides, the authors have not discussed how updated polices are maintained and how the security and accountability are supported when there is the policy update.

Proxy-based Re-encryption (PRE) was initially introduced by Mambo and Okamoto [11]. They proposed a technique that uses a concept of delegator to perform re-encryption of the ciphertext sent by the originator. In this scheme, the delegator learns neither the decryption keys nor original plaintext. Later, Ateniese et al. [12] introduced a proxy re-encryption scheme that improves security in preventing collusion attack over the bilinear map. They implemented the PRE to show its efficiency in a few PRE scenarios. This approach becomes adopted by several PRE-based scheme.

In 2014, Liang et al. [15] proposed a cloud-based revocable identity-based proxy re-encryption (CB-IB-PRE) scheme to support user revocation in the cloud data sharing systems. Hereafter, several works [e.g., 10, 14, 17, 19] have adopted PRE to optimize the revocation overhead, specifically the re-encryption cost in attribute-based access control.

In [16], the authors introduced adaptable CP-ABE scheme to handle policy changes in CP-ABE encryption for data outsourced in cloud computing. In this scheme, a trapdoor is generated from the central authority and it is used to transform a ciphertext under one access policy into ciphertexts under any other access policies. With this scheme, a data owner outsources ciphertext re-encryption task to the proxy and the proxy can not learn the content from the plaintext encrypted. However, the trapdoor generation is still the computation burden that the authority has to compute every time of all policy update events.

In [17], Yukata Kawai proposed a flexible CP-ABE proxy re-encryption scheme by combining key randomized and encrypted methodology and adaptive CP-ABE. The proposed scheme focuses on reducing the computation cost at client side by outsourcing the re-encryption key generation to cloud server. The universal re-encryption key (urk) is proposed to be used together with the decryption key (Sk_s) for generating the re-encryption key. The decryption key is concealed by randomized parameters and sent to the cloud for computing the re-encryption key. Importantly, Kawai's approach is the first attempt dealing with the outsourcing concept of re-encryption key generation in PRE setting. However, the author does not provide the performance evaluation to demonstrate the efficiency of the proposed scheme.

However, the proposed schemes [16, 17] only provide the security function while the implementation result and performance have not been provided. Hence, the efficiency of the proposed CP-ABE proxy re-encryption in handling the policy changes cannot be inferred.

In [19], Fugkeaw and Sato proposed PRE scheme that fully outsources re-encryption key generation to the proxy; the computation cost at data owner is minimized. However, if there are frequent revocation or policy update cases, the re-encryption key needs to be re-generated in every cases and data owners require to prepare and submit data package to the proxy for computing the re-encryption key.

To the best of our knowledge, existing normal PRE schemes are not practical for policy updating in large-scale data outsourcing environment where the access control elements are changed frequently. This is because cost for re-encryption key generation is unpredictable at the data owner side. However, offloading too much computation

cost to a proxy may introduce the delay for re-encryption task and thus cause efficiency problem. Besides, this strategy is also not advisable for the cloud model that the cloud provider charges the fee based on CPU usage. Thus optimizing both setup cost at data owner side and re-encryption cost at cloud side is a real challenge. Unfortunately, this computation optimization aspect has not been addressed by the existing PRE schemes. In this paper, we entail the practical solutions for handling policy evolution in the evolvable cloud environment with the consideration on computation and communication cost reduction in both data owner and cloud side.

3 Background

3.1 C-CP-ARBE Model

In this section, we give basic system definitions of our proposed access control called Collaborative-Ciphertext Policy-Attribute Role-based Encryption (C-CP-ARBE). The proposed access control model integrates role-based access control (RBAC) model into the CP-ABE. The model thus accommodates the benefits of RBAC feature with the attribute–based attribute encryption. RBAC provides more scalable management over a number of attributes [15]. Here, a set of attributes in CP-ABE is assigned to the specific roles and the privileges are included to compliment the expressiveness of access control mechanism. Definitions 1 and 2 show the complete set of our access control elements and access control policy (ACP).

Definition 1: User (U), Role (R), Attributes (attr), and Permission (P)

- User (U) is a subject who requests to access (read or write) the data outsourced by the data owner in the cloud. Each user is assigned the set of attributes with respect to his/her role by the attribute authority.
- Attributes (Attr) are a set of attributes used to characterize the user and associated to the particular attribute "role". A set of attributes is issued by attribute authority (AA).
- Role (R) is a super set of attribute where users and respective attributes are assigned to.
- Permission (P) is an action or privilege having value read (r) and write (w).

Definition 2: Access Control Policy (ACP)

ACP is a tree-based structure. Let ACP T is a tree represent the access structure in C-CP-ARBE. Each non-leaf node of the ACP tree represents the Role node and threshold gate where the Role node is a parent of threshold gate node. The threshold gate rule is the same as access tree of CP-ABE. We denote the parent of the children node x in the tree by parent(x). Thus, the parent of leaf node x is the pair of {Role node, threshold gate}. The function attr(x) is defined only x is in a leaf node of the tree.

To provide a fine-grained access control, we introduce special attribute "privilege" as an extended leaf (EL) node of the ACP T in order to identify the read or write privilege of the role. Figure 1 illustrates a sample access control policy used to enforce access rules to hospital staffs and patients in accessing disease diagnostic data.

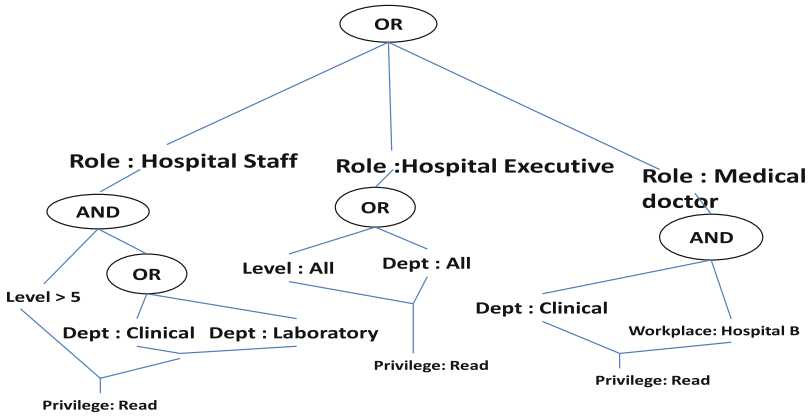

Fig. 1. Access control policy of disease diagnosis file

Figure 1 illustrates a sample access control policy used to enforce access rules to restrict the access of hospital staff and patients to the healthcare data. As seen from the figure, hospital staffs, hospital executives, and a specific group of medical doctor from another hospital is allowed to access the disease diagnostic data.

The policy is administered by the host hospital and it is able to be updated by authorized administrator. In reality, such a policy can be changed anytime. For example, the senior nurse may be allowed to access the diagnosis file for preparing the summarized report. In this case, the data owner needs to update the above policy tree by adding role "nurse" and its attributes with the logical rules specifying the authorized access to the diagnosis file. In addition to updating the policy, the file encrypted by the before-updated policy needs to be retrieved from the cloud and it will be decrypted and re-encrypted with a new policy. Then, it will be uploaded back to the cloud. This is a cumbersome task especially when there is a large amount of data as well as the high chance of policy changes. We will discuss how the policy change is securely and efficiently managed in Sect. 4.

3.2 C-CP-ARBE Constructs

Our proposed cryptographic process of C-CP-ARBE scheme [1] is a kind of Multi-Authority CP-ABE (MA-CP-ABE). We use attribute authority identification (a_{id}) to identify the authority who issues the attributes to users. Each user who is issued the attributes by the attribute authority is identified with uid.aid. Basically, bilinear map is a major construct in our user key generation protocol.

Definition 3: Bilinear Map [7]

Let G_1 and G_2 be two multiplicative cyclic groups of prime order p and e be a bilinear map, e: $G_1 \times G_1 \rightarrow G_2$. Let g be a generator of G_1. Let H: $\{0,1\}^* \rightarrow G_1$ be a hash function that is modeled in a random oracle.

The bilinear map e has the following properties:

1. Bilinearity: for all u, v $\in G_1$ and a, b $\in Z_p$, $e(u^a, v^b) = e(u, v)^{ab}$
2. Non-degeneracy: $e(g, g) \neq 1$.

The following table presents the notations and its description used in our proposed algorithms (Table 1).

Table 1. Notations used in the C-CP-ARBE

Notation	Description
$S_{uid.aid}$	Set of all attributes issued to user *uid* and managed by authority *aid*
SK_{aid}	a secret key which belongs to authority *aid*
PK_{aid}	Public key which belongs to authority *aid*
GSK_{uid}	A global secret key of a user *uid*. GSK is a private key issued by the certification authority CA
$Cert_{uid}$	A public key certificate containing user's public key issued by a certification authority CA
$UDK_{uid.aid}$	User Decryption key issued by authority *aid*
$EDK_{uid.aid}$	EDK is an encrypted form of a UDK which is encrypted by a user public key
GRP	Group role parameter is a seed numbers computed from a set of user members of the roles
SS	Secret seal is a symmetric key created from the AES algorithm together with the GRP
ACP	An access control policy used to encrypt the data files
SCT	A sealed ciphertext is a ciphertext encrypted with the SS

Here, we present our four major cryptographic algorithms including AA setup, user key generation, encryption, and decryption.

1. **AuthoritySetUp**

   Attribute Authority Setup (AA_k where each AA is identified with a_{id})
   Each AA_k (k $\in$ set of all authority S_A).
   Let S (a_K) be a set of attributes issued and managed by the authority AA_k.
   The AA setup (AA_k) chooses two random numbers $\alpha, \beta \in Z_p$.
   Then the Public Key AA_k(or PK_{aid}) = G_1, g, $h = g\beta_k$, $f = g^{1/\beta_k}$, $e(g,g)^{\alpha_k}$; and the Secret Key AA_k (or SK_{aid}) is (β_k, g^{α_k}).

2. **UserKeyGen**($S_{uid,aid}$, SK_{aid}, $Cert_{uid}$) $\rightarrow$ $EDK_{uid,aid}$, RDK_{aid}. The KeyGen algorithm takes continuous two steps as follows:
   (1) The algorithm takes input as set of attributes $S_{uid,aid}$, attribute authority's secret key SK_{aid}, then it returns the set of user decryption keys UDK.
   (2) A UDK is encrypted with the global public key of the user $Cert_{uid}$ and outputs an encrypted decryption key $EDK_{uid,aid}$. In addition to the UDK generated, the

system will also produce the root decryption key RDK_{aid} for further use in re-encryption key generation. It contains the data owner's ID attribute and digital signature attribute of the data owner. Thus, the RDK_{aid} is very small and it can be used to decrypt the files they created because these two attributes are bounded in the ACP as default attributes. RDK_{aid} is also encrypted by the data owner's public key.

3. **Enc**(PK_{aid}, [SS, GRP], M, ACP, $Cert_{uid}$) → SCT. The encryption algorithm performs two continuous steps as follows:
   (1) *Inner Layer:* the algorithm takes as inputs authority public key PK_{aid}, access control policy ACP, and data M. Then it returns a ciphertext CT.
   (2) *Outer Layer:* the algorithm takes group role parameter GRP which is randomly generated from a set of user members (i.e. Users' IDs) of all roles. GRP is used as a key together with AES algorithm to generate the session key referred as a secret seal SS. The SS is used to encrypt the ciphertext CT. Then, the algorithm returns sealed ciphertext SCT. Finally, a SS is encrypted with user's public key $Cert_{uid}$, and stored in the cloud server.
4. **Decrypt**(PK_{aid}, SCT, GSK_{uid}, EDK_{uid},) → M. The decryption algorithm performs two continuous steps as follows:
   (1) Decrypt the secret seal SS. The algorithm takes user's global secret key GSK_{uid} and then obtains the session key to decrypt the SCT and gets the CT.
   (2) Decrypt the encrypted decryption key (EDK_{uid}). The algorithm takes user's global secret key GSK_{uid} and then obtains the user decryption key UDK. Then, if the set of attribute S satisfies the ACP structure, the algorithm returns the original M.

4 Policy Updating Method

To complete the policy updating process, two tasks including policy updating and file re-encryption are required. To this end, we propose a policy updating algorithm and a proxy re-encryption technique called a very lightweight PRE (VL-PRE) to efficiently support the required tasks respectively.

4.1 Flexible and Secure Policy Update Management

Outsourcing policy update to the cloud enhances the service availability and reduces computing costs at data owner side.

In typical cloud-based access control systems, if there is a change to the policy, data owners apply a new policy to re-encrypt the files at their local side and send them back to the cloud server. Accordingly, policy update introduces the communication, computation, and maintenance cost at data owners.

Therefore, a flexible and secure policy update should be provided to allow data owners or administrators to manage the attributes (add, update, delete) in polices stored in a cloud server in a practical manner. We develop policy updating algorithm to

support access policy updating in the cloud. This reduces computation and communication cost and allows the data owners to update the policy anytime and anywhere.

```
Input newleafnode (Y')
Input currentleafnode (Y)
Input typeupdate
Input policyid_input
Set ACP = get policy where policyid = policyid_input
if (typeupdate  == delete)
        Set newpolicy ACP' = ACP.replace((Y), "")
else if (typeupdate  == edit)
        Set  ACP' = ACP.replace ((Y ), (Y'))
else if (typeupdate  == add)
        Set ACP' = ACP.insert (Y')
policy = ACP'
files_set = get fileid where policyid = policyid_input
      foreach (files_set as id)
              id.reencrypt(ACP')
      end foreach
If(Verify_Policy_Syntax (ACP'))
      output ACP'
else
      output "policy syntax error"
```

Fig. 2. Policy updating algorithm

```
Input policy ACP
If (policy.contain(comparison between an attribute name and
a negative integer))
      return false
else If (policy.contain(comparison between an attribute
name and an attribute name))
      return false
else if (policy.contain(attribute name start with integer))
      return false
else if (policy.contain ("and" | "or" using in attribute))
      return false
else if(!policy.contain(operator is 'and' , 'or' , 'of' ,
'>' , '<' , '>=' , '<=' , '=' only))
      return false
else if(policy.contain(threshold gate operator as 'K of
(P1, P2, ... PN)' && K is negative integer))
      return false
else
return true
```

Fig. 3. Policy update syntax validation

Figures 2 and 3 illustrate the policy updating process and policy updating syntax validation. The policy updating undertakes the updating operations including add, update, and delete of the attributes contained in the policy together the syntax checking. For the syntax checking, the algorithm checks the possible operands taken on the attribute type and attribute value. This guarantees that the updated policy is syntactically correct. In our scheme, after the policy updating is done, the proxy will automatically take the updated policy to re-encrypt all files encrypted by the before-updated policy.

4.2 Very Lightweight Proxy Re-Encryption (VL-PRE)

VL-PRE is an extended PRE model that is specifically designed to deliver a very lightweight PRE operation in supporting attribute revocation or policy update in CP-ABE based access control. The process of VL-PRE is divided into three phases: Generate re-encryption key, Update re-encryption key, and Renew re-encryption key. Generally, the proposed three-phase PRE is triggered when there is a case of attribute revocation or policy update. Basically, the proxy transforms ciphertext CT_{k1} to CT_{k2} with a re-encryption key $RK(rk_{s1 \rightarrow s2})$ where RK is generated by a proxy server.

Phase 1: Generate Re-encryption Key:

For the initial phase, it consists of Pre-process, ReKeyGen and ReEnc algorithms which are described as follows.

1. **Pre-process:** Data owner (1) chooses random seeds and generates secure random number R and applies random number R_{vn} (tagged with the current version number vn) to encrypt the root decryption key RDK_{aid} generated since the key generation phase. (2) applies R_{vn} to append the attributes in the leaf node of the updated version of access control policy ACP_{vn}, and gets the $ACP_{vn}^{R_{vn}}$. Then, data owner submits encrypted RDK_{aid} and $ACP_{v_n}^{R_{vn}}$ as parts of re-encryption key to the cloud proxy.
2. **ReKeyGen** (param; SS, $R_{vn}(RDK_{aid})$, $(ACP_{v_n}^{R_{vn}})$, ExpireTime) $\rightarrow$ $rk_{s2 \rightarrow (M',\ ACP')}$. The algorithm takes input param, secret seal SS, root decryption key encrypted by the Random R_{vn}, $R_{vn}(RDK_{aid})$, a new access policy embedded with Random R_{vn}, $ACP_{v_n}^{R_{vn}}$, and Expire_time. First, the SS is used to decrypt the sealed ciphertext (SCT) and the original ciphertext (CT) is derived. The Expire_time is used to indicate the validity of re-encryption key rk_{s2}. Hence, if the key expires, the owner needs to initiate re-key generation with a new random R_{vn}.

Then, the algorithm outputs a re-encryption key $rk_{s2 \rightarrow (M',\ ACP')}$ that can be used to transform a ciphertext under (M, ACP) to another ciphertext under (M′, ACP′).

- **ReEnc**(param; $rk_{s2 \rightarrow (M',\ ACP')}$, CMR function, CT(M, ACP)) $\rightarrow$ CT_{k2}: The algorithm takes input param, a re-encryption key $rk_{s2 \rightarrow (M',\ ACP')}$, CombineMatchRemove function CMR, and an original CT(M, ACP). It outputs a re-encrypted ciphertext $CT'(M', ACP')$.

According to the element of rk_{s2}, we embed the CombineMatchRemove (CMR) function to support the re-encryption process as follows:

(1) Combine pieces of R applied in leaf nodes of a new $ACP_{v_n}^{R_{vn}}$.
(2) Match R between R_{vn} (RDK_{aid}) and $ACP_{v_n}^{R_{vn}}$.
(3) Remove R from $R_{vn}(RDK_{aid})$.

Then, the RDK_{aid} is automatically used to decrypt the old ciphertext and the algorithm applies a new ACP′ to re-encrypt the data. Finally, the proxy takes SS to encrypt a new Ciphertext (CT_{k2}).

Phase 2: Update Re-encryption Key:

There are two algorithms for updating re-encryption key.

1. **UpdateACP(R_{vn},ACP_{vn+1})** $\rightarrow$ $ACP^{R_{vn}}_{v_{n+1}}$

Data owner applies current random number R_{vn} to encrypt the updated ACP, and the $ACP^{R_{vn}}_{v_{n+1}}$ is obtained and sent to the proxy.

2. **UpdateReEncKey($rk_{s2,vn}$,** $ACP^{R_{vn}}_{v_{n+1}}$**)** $\rightarrow$ **$rk_{s2,vn+1}$**

The proxy runs the algorithm by taking the updated ACP,$ACP^{R_{vn}}_{v_{n+1}}$ to update the current version of re-encryption key, $rk_{s2,vn}$. The new $rk_{s2,vn+1}$ is used to re-encrypt the existing ciphertext.

The algorithms help to reduce both computation and communication overhead at both data owner side and proxy since the RDK needs not to be encrypted every time and the information (only the updated ACP) sent out to the proxy is small. Besides, the proxy does not need to fully compute a new re-encryption key upon policy update, it only updates the key instead.

Phase 3: Renew Re-encryption Key

In this phase, if the current re-encryption key $rk_{s2,vn}$ expires, the algorithms in phase 1 will be run.

Here, the owner needs to initiate re-key generation with a new set of random seeds R_{vn+1} and updated ACP. Then, re-encryption key generation and ciphertext re-encryption are performed by the proxy.

However, re-encryption key renewal is not required to perform instantly when the key expires, it will be executed when there is the next policy update.

4.3 Security Model

Our C-CP-ARBE is secure under the random oracle model in the following security game.

1. **Initialization.** Adversary *A* outputs a challenge access policy ACP^C to Challenger C.
2. **Setup.** C runs CreateAttributeAuthority algorithm and gives a public keys *PK* to the adversary *A*. For corrupted authorities $S'_{A,}$ the challenger sends both the public keys and secret keys to adversary.
3. **Query Phase1:**
   (a) Private key extraction: *C* runs UserKeyGen on the attribute set *S* ($S_{uid,aid}$) of the corrupted AA and returns UDK to *A*.
   (b) Re-encryption key extraction oracle O_{rk} (*S,* ACP^C): With attribute set *S*, and an access control policy ACP^C, *C* returns reKeyGen(param; SS, R_{vn}(RDK_{aid}), (ACP^{CRvn})) $\rightarrow$ $rk_{s2,vn \rightarrow (M',\ ACP')}$ to *A*, where $rk_{s2,vn}$ is a generated re-encryption key and (*S*, SK_{aid}) $\rightarrow$ UDK.

(c) RE-encryption oracle O_{rk} (S, ACP^C, $CT_{(M,\ ACP)}$): With the input an attribute set S, an access control policy ACP^C, and an original ciphertext $CT_{(M,\ ACP)}$, C returns $rk_{s2 \rightarrow (M',\ ACP')}$,$CT_{(M,\ ACP)}$) $\rightarrow$ $CT^R_{(M\prime,ACP\prime)}$, where reKeyGen(param, SS, $R_{vn}(RDK_{aid})$, (ACP'^R)) $\rightarrow$ $rk_{s2 \rightarrow (M',\ ACP')}$, ($S$, SK_{aid}) $\rightarrow$ UDK and S| = ACP.

(d) Original ciphertext decryption oracle $O_{d2}(S,\ CT_{(M,\ ACP)})$. With the input an attribute set S and an original ciphertext $CT_{(M,\ ACP)}$,C returns Decrypt(S, UDK, $CT_{(M,\ ACP)}$) $\rightarrow$ M to A, where (S, SK_{aid}) $\rightarrow$ UDK and S| = ACP.

(e) Re-encrypted ciphertext decryption oracle $O_{d2}(S',\ CT^R_{(M\prime,ACP\prime)})$. With the input an attribute set S' and a re-encrypted ciphertext $CT^R_{(M\prime,ACP\prime)}$, C returns Decrypt (S', UDK′, $CT^R_{(M\prime,ACP\prime)}$) $\rightarrow$ M, where (S', SK_{aid}) $\rightarrow$ UDK′ and S′| = ACP′.

Note that if the ciphertexts queried to oracles O_{re}, O_{d2}, and O_{d1} are invalid, C simply outputs a $\perp$.

1. **Challenge.** A outputs two equal length messages M_0 and M_1 to C. C returns $CT^*(M^*,ACP^*)$ = Enc(ACP^*, M_b) to A, where b $\in$ {0,1}.
2. **Query Phase II:** A performs as it did in Phase 1.

Guess. A submits a guess bit b′ $\in$ {0,1}. If b′ = b, A wins. The advantage of A in this game is defined as $\Pr[b' = b|\mu = 0] = \frac{1}{2}$. □

In the security point of view of VL-PRE, we use random encryption to secure re-encryption key component while our core access control enforcement is based on CP-ABE. The detailed security proof is as presented in the original CP-ABE [7].

4.4 Policy Update Evaluation

We analyze and evaluate our policy update scheme based on the correctness, accountability, and security requirement.

Correctness: *An updated policy must be syntactically correct and users who hold the keys containing a set of attributes satisfying the policy are able to decrypt the data encrypted by an updated policy.*

Proof: The syntax of the updating is validated through the CP-ABE tree structure. Hence, attributes updated to AND, OR, K *out of* N is done at the policy structure. The policy checking for the update is controlled by our policy updating algorithm. The algorithm verifies the syntax of the threshold gates to ensure the correctness of grammar of tree-based model. Also, if the policy is updated with valid attributes (issued trusted AA with $PK_{x.aid}$) the users who hold sufficient attributes satisfying a new policy are able to decrypt the file encrypted by a new policy. This correctness is guaranteed by CP-ABE model.

Security: *A policy must be updated by the data owner or authorized administrator only in the secure manner and a new policy should not introduce problems for the existing access control.*

Proof: To enable the policies to be securely stored and managed in cloud, we make use a simple CP-ABE tree policy to encrypt the ACP. The policy encryption is simply formed by a set of identity attributes of the data owners and authorized users. Hence, only data owners and authorized users are allowed to access the policy and can use the policy to encrypt the data. Here, the data owner can selectively delegate the policy update function to the users. In addition, our scheme requires data owner's digital signature for executing and committing the update.

Accountability: *All policy updating events must be traceable.*

Proof: When the policy is updated, event log keeps the details of update including login users, update time, and update operations. In addition, the system requires digital signing of the authorized data owner or administrator to commit the update.

5 Evaluation

5.1 Comparison of Policy Update Cost

We analytically compare policy update features and update cost between the C-CP-ARBE, Yang et al. scheme [3], and Lai et al. scheme [16].

Table 2. Comparison of policy update feature and cost

Operation	Yang et al. [3]	Lai et al. [16]	Our C-CP-ARBE
Update key generation	At owner side	At owner/authority side	At cloud server
Policy storage outsourcing	No	No	Yes
Policy update method	Ciphtertext update	PRE	VL-PRE
Computation	$O(t_c)$	$O(1)$	$O(1)$

t_c = the total number of attributes in the updated ciphertext

From Table 2, according to Yang et al. scheme, data owner has to update key generation and to update the ciphertext to complete the policy updating process. For the ciphertext update, the data owner needs to compute ciphertext components for new attributes. The entire computation cost is subject to the number of attributes and the type of update operations (i.e. OR, AND) over the access structure. In Lai et al. scheme, PRE concept is used to convert the existing ciphertext according to the updated policy. In this scheme, the trapdoor or re-encryption key is generated at key generation authority or at data owner side. This limits the operation with the dependability on the availability of the authority or data owner. In contrast, we delegate the major cost of re-encryption key generation and file re-encryption to the delegated proxy in the cloud.

Our scheme has no limitation for update operations and number attributes involving in the policy. The computation cost for re-encryption is $O(1)$ as the re-encryption is performed once to complete the policy update. In addition, our scheme allows policies to be securely stored in the cloud. This enables flexible and efficient policy management in data outsourcing scenario.

5.2 Performance Evaluation

In our experiments, we implement the application service using PHP and Java language which are run on the Apache Sever. The service is run on Intel Xeon, E562 processor 240 GHz. with Ubuntu Linux. We use the Pairing-Based Cryptography library version 0.5.12 to simulate the cryptographic constructs of those two compared schemes. Our core cryptographic library is extended and developed from the CP-ABE programming library provided in [18]. On the client (data owner) end, a simulation was run on MacBook Pro Intel Core i5 Dual-core, 2.7 GHz.

In the experiment setting, we simulate KeyUpdate and CiphertextUpdate algorithms for Yang et al. scheme, while Trapdoor generation and policy update based on PRE are simulated for Lai et al. scheme. For our C-CP-ARBE scheme, the time used for executing policy updating algorithm and processing the VL-PRE are used to measure the total cost of policy update.

To demonstrate the performance improvement, we compare total time used for policy updating and re-encryption between these three approaches. We simulate the policy updating protocols of Yang et al. scheme by simulating the key update generation and ciphertext update while Lai et al. scheme and our C-CP-ARBE use the PRE strategy. To measure the performance, we vary the number of attributes updated in the given access policy. The access policy contains up to 120 attributes and it is used to encrypt 2-MB file. Then, we measure the total time for the policy update and file re-encryption or ciphertext update used by these three schemes.

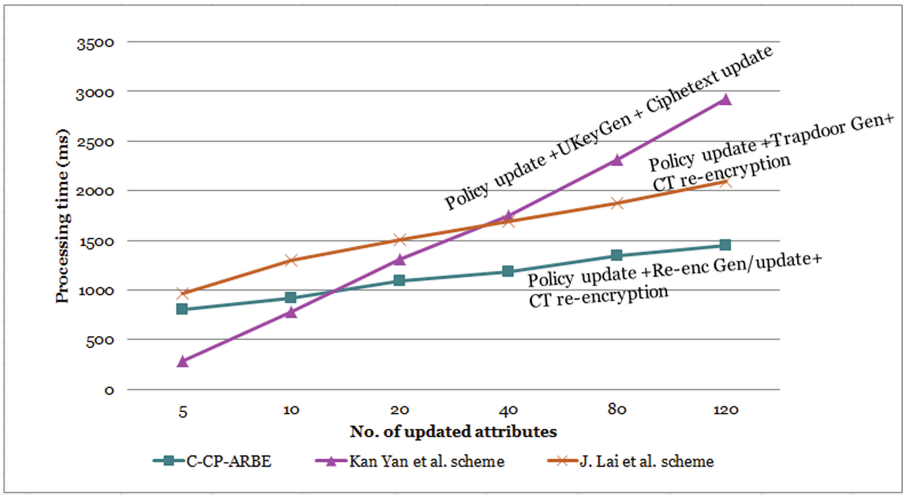

Fig. 4. Comparison of policy update cost

As of the Fig. 4, compared with Yang et al. scheme and Lai et al. scheme, our C-CP-ARBE fully outsources the PRE process to the proxy. Thus, the computation at data owner side is significantly reduced. With our scheme, the data owner only updates at her own machine, while the policy and the subsequent costs (re-encryption key generation and ciphertext re-encryption) are fully outsourced to the delegated proxy. With a small re-encryption key size and key update strategy of VL-PRE, the processing workload performed by the proxy at cloud is also optimized. In Lai et al. scheme, even though the PRE is used to transform the ciphertext, the data owner still needs to compute the trapdoor and update the policy before the proxy performs the re-encryption task. Obviously, both C-CP-ARBE and Lai et al. scheme do not get significant impact from the number of attributes changed or the operations used in the policy. In contrast, with the ciphertext update strategy of Yang et al. scheme, it is very practical to support a small number of updated attributes. However, when the number of updated attributes increases, the processing time sharply increases.

6 Conclusion

In this paper, we have presented a privacy-preserving CP-ABE based access control model with the capability of policy change management in the data outsourcing scenario. Our core access control policy contains roles, attributes, and privileges logically modeled in a tree-based structure. We introduce our proposed policy updating algorithm and VL-PRE to securely and economically support policy evolution in cloud data access control. VL-PRE uses a small package of re-encryption key generation and relies on key updates strategy instead of key generations. Therefore, it outperforms existing PRE schemes. Finally, we conduct the experiments to evaluate the performance of our proposed policy update scheme. The results reveal that our proposed scheme is efficient and promising for real deployment in supporting policy update for data outsourcing scenario.

For future work, we will conduct larger scale of experiments in real cloud environment and measure the scalability of the proposed system in serving concurrent updates of multiple policies.

References

1. Fugkeaw, S., Sato, H.: An extended CP-ABE based access control model for data outsourced in the cloud. In: Proceedings of the International Workshop on Middleware for Cyber Security, Cloud Computing and Internetworking (MidCCI 2015), pp. 73–78. IEEE (2015)
2. Wan, Z., Liu, J., Deng, H.R.: HASBE: a hierarchical attribute-based solution for flexible and scalable access control in cloud computing. IEEE Trans. Inf. Forensics Secur. **7**(2), 743–754 (2012)
3. Yang, K., Jia, X., Ren, K., Xie, R., Huang, L.: Enabling efficient access control with dynamic policy updating for big data in the cloud. In: Proceedings of the International Conference on Computer Communications (INFOCOM 2014), pp. 2013–2021. IEEE (2014)

4. Li, M., Yu, S., Zheng, Y., Ren, K., Lou, W.: Scalable and secure sharing of personal health records in cloud computing using attribute-based encryption. IEEE Trans. Parallel Distrib. Syst. **24**(1), 131–143 (2013)
5. Yang, K., Jia, X., Ren, K., Zhang, B., Xie, R.: Expressive, efficient, and revocable data access control for multi-authority cloud storage. IEEE Trans. Parallel Distrib. Syst. **25**(7), 1735–1744 (2014)
6. Goyal, V., Pandey, O., Sahai, A., Waters, B.: Attribute-based encryption for fine-grained access control of encrypted data. In: Proceedings of the International Conference on Computer and Communications Security (CCS 2006), pp. 89–98. ACM (2006)
7. Bethencourt, J., Sahai, A., Waters, B.: Ciphertext-policy attribute-based encryption. In: Proceedings of IEEE Symposium of Security and Privacy, pp. 321–334. IEEE (2007)
8. Fugkeaw, S.: Achieving privacy and security in multi-owner data outsourcing. In: Proceedings of the International Conference on Digital and Information Management (ICDIM 2012), pp. 239–244. IEEE (2012)
9. Yang, K., Jia, X., Ren, K.: Secure and verifiable policy update outsourcing for big data access control in the cloud. IEEE Trans. Parallel Distrib. Syst. (TPDS) **26**(12), 3461–3470 (2015). IEEE
10. Tysowski, P.K., Hasan, M.A.: Hybrid attribute- and re-encryption-based key management for secure and scalable mobile applications in clouds. IEEE Trans. Cloud Comput. **1**(2), 172–186 (2013). IEEE
11. Mambo, M., Okamoto, E.: Proxy cryptosystems: delegation of the power to decrypt ciphertexts. IEICE Trans. **E80-A**(1), 54–63 (1997)
12. Ateniese, G., Fu, K., Green, M., Hohenberger, S.: Improved proxy re-encryption schemes with applications to secure distributed storage. ACM Trans. Inf. Syst. Secur. **9**, 1–30 (2006). ACM
13. Sahai, A., Seyalioglu, H., Waters, B.: Dynamic credentials and ciphertext delegation for attribute-based encryption. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO 2012. LNCS, vol. 7417, pp. 199–217. Springer, Heidelberg (2012)
14. Liang, X., Cao, Z., Lin, H., Shao, J.: Attribute based proxy re-encryption with delegating capabilities. In: Li, W., Susilo, W., Tupakula, K.U., Safavi-Naini, R., Varadharajan, V. (eds.) ASIACCS, pp. 276–286. ACM, New York (2009)
15. Liang, K., Liu, J.K., Wong, D.S., Susilo, W.: An efficient cloud-based revocable identity-based proxy re-encryption scheme for public clouds data sharing. In: Kutyłowski, M., Vaidya, J. (eds.) ICAIS 2014, Part I. LNCS, vol. 8712, pp. 257–272. Springer, Heidelberg (2014)
16. Lai, J., Deng, R.H., Yang, Y., Weng, J.: Adaptable ciphertext-policy attribute-based encryption. In: Cao, Z., Zhang, F. (eds.) Pairing 2013. LNCS, vol. 8365, pp. 199–214. Springer, Heidelberg (2014)
17. Kawai, Y.: Outsourcing the re-encryption key generation: flexible ciphertext-policy attribute-based proxy re-encryption. In: Lopez, J., Wu, Y. (eds.) ISPEC 2015. LNCS, vol. 9065, pp. 301–315. Springer, Heidelberg (2015)
18. CP-ABE Library. http://acsc.cs.utexas.edu/cpabe/
19. Fugkeaw, S., Sato, H.: Embeding lightweight proxy re-encryption for efficient attribute revocation in cloud computing. Int. J. High Perform. Comput. Netw. (in press)