ISSN 2395-695X (Print)
ISSN 2395-695X (Online)
International Journal of Advanced Research in Biology Engineering Science and Technology (IJARBEST)
Vol. 2, Special Issue 10, March 2016

Secure and Supportable Policy Renew Subcontract for Big Data Access Control in theCloud

A.Sangeetha, *Member,* S.Tharani, *Member,* R.Vennila, *Member* Guided by A.Priya,AP/CSE Staff *Member,*

Abstract— They focus on solving the policy updating problem in AES systems, and propose a secure and verifiable policy updating outsourcing method. Instead of retrieving and re-encrypting the data, data owners only send policy updating queries to cloud server, and let cloud server update the policies of encrypted data directly, which means that cloud server does not need to decrypt the data before/during the policy updating. Our scheme can not only satisfy all the above requirements, but also avoid the transfer of encrypted data back and forth and minimize the computation work of data owners by making full use of the previously encrypted data under old access policies in the cloud. The contributions of this paper include: 1) we formulate the policy updating problem in AES systems and develop a new method to outsource the policy updating to the server. 2) We propose an expressive and efficient data access control scheme for big data, which enables efficient dynamic policy updating. 3) We design policy updating algorithms for different types of access policies, e.g., Boolean Formulas, LSSS Structure and Access Tree. Compared to the conference version, we also propose an efficient and secure policy checking method that enables data owners to check whether the cipher texts have been updated correctly by cloud server. In this method, we do not require any help of data users, and data owners can check the correctness of the cipher text updating by their own secret keys and checking keys issued by each authority. Our method can also guarantee data owners cannot use their secret keys to decrypt any cipher texts encrypted by other data owners, although their secret keys contain the components associated with all the attributes. Moreover, we discuss some key features of the attribute-based access control scheme and show how it is suitable for big data access control in the cloud. What's more, we also add more performance evaluation on policy updating algorithms and the policy checking method.

Index Terms— Dynamic key generation, Access Control, ABAC, AES, Big Data, Cloud

1 INTRODUCTION

Big data refers to high volume, high velocity, and/or high variety information assets that require new forms of processing to enable enhanced decision making, insight discovery and process optimization. Due to its high volume and complexity, it becomes difficult to process big data using on-hand database management tools. An effective option is to store big data in the cloud, as the cloud has capabilities of storing big data and processing high volume of user access requests in an efficient way. When hosting big data into the cloud, the data security becomes a major concern as cloud servers cannot be fully trusted by data owners.

Attribute-Based Encryption (ABE) [1]–[5] has emerged as a promising technique to ensure the end-to-end data security in cloud storage system. It allows data owners to define access policies and encrypt the data under the policies, such that only users whose attributes satisfying these access policies can decrypt the data. When more and more organizations and enterprises outsource data into the cloud, the policy updating becomes a significant issue as data access policies may be changed dynamically and frequently by data owners. However, this policy updating issue has not been considered in existing attribute-based access control schemes [6]–[9].

The policy updating is a difficult issue in attribute-based access control systems, because once the data owner outsourced data into the cloud, it would not keep a copy in local systems. When the data owner wants to change the access policy, it has to transfer the data back to the local site from the cloud, re-encrypt the data under the new access policy, and then move it back to the cloud server. By doing so, it incurs a high communication overhead and heavy computation burden on data owners. This motivates us to develop a new method to outsource the task of policy updating to cloud server.

The grand challenge of outsourcing policy updating to the cloud is to guarantee the following requirements:

1) *Correctness*: Users who possess sufficient attributes should still be able to decrypt the data encrypted under new access policy by running the original decryption algorithm.
2) *Completeness*: The policy updating method should be able to update any type of access policy.
3) *Security*: The policy updating should not break the security of the access control system or introduce any new security problems.

The policy updating problem has been discussed in key-policy structure [1] and ciphertext-policy structure [10]. However, these methods cannot satisfy the completeness requirement, because they can only delegate key/ciphertext with a new access policy that should be more restrictive than the previous policy. Furthermore, they cannot satisfy the security requirement either. For example, when a new attribute is added into a threshold gate and the threshold gate is changed from (t, n) to a $(t + 1, n + 1)$, both methods will set the share of the new attribute to be 0. In this case, users who only holds t attributes (excluding the new attribute) can satisfy the new $(t + 1, n + 1)$-gate.

In this paper, we focus on solving the policy updating problem in ABE systems, and propose a secure and verifiable policy updating outsourcing method. Instead of retrieving and re-encrypting the data, data owners only send policy updating

Kan Yang is with the Department of Electrical and Computer Engineering, University of Waterloo, Waterloo, ON N2L 3G1, Canada (email: kan.yang@uwaterloo.ca)
Xiaohua Jia is with the Department of Computer Science, City University of Hong Kong, Kowloon, Hong Kong S.A.R. (email: csjia@cityu.edu.hk)
Kui Ren is with the Department of Computer Science and Engineering, Sate University of New York at Buffalo, Buffalo, NY, 14260, USA (email: kuiren@buffalo.edu)

queries to cloud server, and let cloud server update the policies of encrypted data directly, which means that cloud server does not need to decrypt the data before/during the policy updating. Our scheme can not only satisfy all the above requirements, but also avoid the transfer of encrypted data back and forth and minimize the computation work of data owners by making full use of the previously encrypted data under old access policies in the cloud.

The contributions of this paper include:

1) We formulate the policy updating problem in ABE sytems and develop a new method to outsource the policy updating to the server.
2) We propose an expressive and efficient data access control scheme for big data, which enables efficient dynamic policy updating.
3) We design policy updating algorithms for different types of access policies, e.g., Boolean Formulas, LSSS Structure and Access Tree.

Compared to the conference version, we also propose an efficient and secure policy checking method that enables data owners to check whether the ciphertexts have been updated correctly by cloud server. In this method, we do not require any help of data users, and data owners can check the correctness of the ciphertext updating by their own secret keys and checking keys issued by each authority. Our method can also guarantee data owners cannot use their secret keys to decrypt any ciphertexts encrypted by other data owners, although their secret keys contain the components associated with all the attributes. Moreover, we discuss some key features of the attribute-based access control scheme and show how it is suitable for big data access control in the cloud. What's more, we also add more performance evaluation on policy updating algorithms and the policy checking method.

The remaining of this paper is organized as follows. In Section 2, we define system model, framework and security model. Section 3 describes an attribute-based access control scheme for big data based on an adapted mutli-authority CP-ABE method in [5]. In Section 4, we propose several policy updating algorithms for different types of access policies. In Section 5, we design a method that enables the data owner to check whether the ciphertexts have been updated correctly by cloud server. In Section 6, we give a comprehensive analysis of our scheme in terms of correctness, completeness, security and performance. The related work is given in Section 7. Finally, this paper is summarized in Section 8. In the Supplemental File, we describe the definition of access structures in ABE systems, as well as two types of access structures that are well utilized in constructing ABE schemes.

2 System and Security Model

System Model

We consider a cloud storage system with multiple authorities, as shown in Fig.1. The system model consists of the following entities: authorities (*AA*), cloud server (server), data owners (owners) and data consumers (users).

Authority. Every authority is independent with each other and is responsible for managing attributes of users in its domain. It also generates a secret/public key pair for each attribute in its domain, and generates a secret key for each user according to his/her attributes.

Server. The cloud server stores the data for data owners and provides data access service to users. The server is also responsible for updating ciphertexts from old access policies to new access policies.

Owner. The data owners define access policies and encrypt data under these policies before hosting them in the cloud. They also ask the server to update access policies of the encrypted data stored in the cloud. After that, they will check whether the server has updated the policies correctly.

User. Each user is assigned with a global user identity and can freely get the ciphertexts from the server. The user can decrypt the ciphertext, only when its attributes satisfy the access policy defined in the ciphertext.

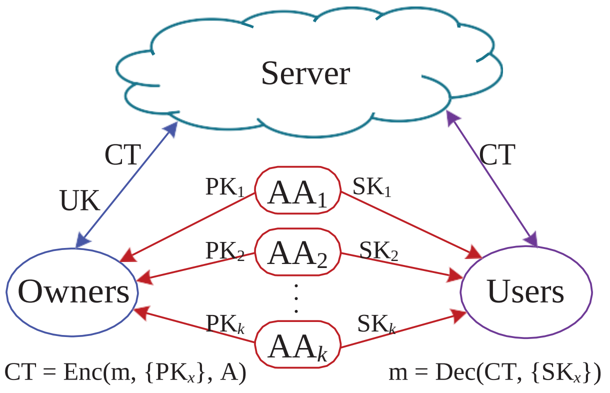

Fig. 1. System Model

Framework

To meet all the requirements of policy updating, we define the framework of our access control scheme as follows.

Definition 1 (Framework). *Our dynamic policy access control scheme is a collection of the following algorithms:* GlobalSetup, AuthoritySetup, SKeyGen, Encrypt, Decrypt, UKeyGen *and* CTUpdate.

- **GlobalSetup**(λ) $\rightarrow$ GP. The global setup algorithm takes no input other than the implicit security parameter λ. It outputs the global parameter GP for the system.
- **AuthoritySetup**(GP, *AID*) $\rightarrow$ (SK, PK). The authority setup algorithm is run by each authority *AID* with GP and the authority identity *AID* as inputs and its secret/public key pair (SK_{AID}, PK_{AID}) as outputs.
- **SKeyGen**(*GID*, GP, $S_{GID,AID}$, SK_{AID}) $\rightarrow$ $SK_{GID,AID}$. Each authority *AID* runs the secret key generation algorithm to generate a secret key $SK_{GID,AID}$ for user *GID*. It takes as inputs the global identity *GID*, the global parameter GP, a set of attributes $S_{GID,AID}$ issued by this authority *AID* and the secret key SK_{AID} of this authority. It outputs a secret key $SK_{GID,AID}$ for this user *GID*.
- **Encrypt**({PK}, GP, *m*, A) $\rightarrow$ CT. The encryption algo-

rithm takes as inputs a set of public keys {PK} of relevant

authorities, the global parameter GP, the message m and an access policy A. It outputs a ciphertext CT.

_ Decrypt(CT;GP;fSKGID;AIDg)!m. The decryption algorithm takes as inputs the ciphertext, the global parameter GP and a collection of secret keys from relevant authorities for user GID. It outputs the message m when the user's attributes satisfy the access policy associated with the ciphertext. Otherwise, the decryption fails.

_ UKeyGen(fPKg;EnInfo(m);A;A0) ! UKm. The update key generation algorithm is run by the data owner. It takes as inputs the relevant public keys, the encryption information EnInfo(m) of the message m, the previous access policy A and the new access policy A0. It outputs the update key UKm of m used to update the ciphertextCT from the previous access policy to the new one.

_ CTUpdate(CT;UKm) ! CT0. The ciphertext updating algorithm is run by cloud server. It takes as inputs the previous ciphertext CT and the update key UKm. It outputs a new ciphertext CT0 corresponding to the new access policy A0.

2.3 Security Model

The cloud server is curious about the stored data and messages it received during the services. But it is assumed that the cloud server will not collude with users, i.e., it will not send theciphertexts under previous policies to users, whose attributes can satisfy previous access policies but fail to satisfy newaccess policies. Data owners are assumed to be fully trusted. The users are assumed to be dishonest, i.e., they may collude to access unauthorized data. The authorities can be corrupted or compromised by the attackers. We assume that the adversary can corrupt authorities only statically, but key queries can be made adaptively. We now describe the security model of our system by the following game between a challenger and an adversary: Setup. The global setup algorithm is run. The adversary specifies a set S0 A _SA of corrupted authorities. The challenger generates secret/public key pairs by running the authority setup algorithm. For uncorrupted authorities in SA S0 A, the challenger sends only public keys to the adversary.For corrupted authorities in S0 A, the challenger sends both public keys and secret keys to the adversary.

Phase 1. The adversary makes secret key queries by submitting pairs (GID;SGID;AID) to the challenger, where GID is an identity and SGID;AID is a set of attributes belonging to an uncorrupted authority AID. The challenger gives the corresponding secret keys SKGID;AID to the adversary.

Challenge. The adversary submits two equal length messages m0 and m1. In addition, the adversary gives a set of challenge access structure f(M_ (1);r_ 1); _ _ _ ; (M_ (q);r_ q)g which must satisfy the constraint that the adversary cannot ask for a set of keys that allow decryption, in combination with any keys that can be obtained from corrupted authorities. The challenger then flips a random coin b, and encrypts mb under all access structures f(M_ (1);r_ 1); _ _ _ ; (M_ (q);r_ q)g. Then, the ciphertext fCT_ 1; _ _ _ ;CT_ qg are given to the adversary. Phase 2. The adversary may query more secret keys, as long as they do not violate the constraints on the challenge access structures. The adversary can also makes update key queries by submitting the pair (M_ (i);r_
i); (M_
(j);r_j),
the simulator returns the update key UKmb to the adversary.

Guess. The adversary outputs a guess b0 of b. The advantage of an adversary A in this game is defined as
Pr[b0 = b] 12
.

Definition 2. Our scheme is secure against static corruption of authorities if all polynomial time adversaries have at most a negligible advantage in the above security game.

3 ATTRIBUTE-BASED ACCESS CONTROL WITH DYNAMIC POLICY UPDATING FOR BIG DATA

We construct our dynamic-policy access control scheme based on an adapted CP-ABE method in [5]. Our scheme consists of five phases: System Initialization, Key Generation, Data Encryption, Data Decryption and Policy Updating.

3.1 System Initialization

The system initialization includes two phases: global setup and authority setup.

3.1.1 Global Setup

During the global setup, two multiplicative groups G and GT are chosen with the same prime order p and the bilinear map : G_G ! GT between them. A random oracle H maps global identities GID to elements of G. Let g be a generator of G, the global parameter GP is set to be
GP = (p; g; H):

3.1.2 Authority Setup

Each authority AID runs the authority setup algorithm AuthoritySetup to generate its secret/public key pair. Let SAID denote the set of all the attributes managed by the authority AID. For each attribute x 2 SAID, the authority chooses two random exponents ax;bx 2 Zp and publishes its public key as PKAID = f e(g;g)ax ; gbx g8x2SAID: It keeps SKAID = fax;bxg8x2SAID.

3.2 Key Generation

For each user GID, each authority AID will first assign a set of attributes SGID;AID to this user. It then runs the secret key generation algorithm SKGen to generate a set of secret keys as
SKGID;AID = fKx;GID = gaxH(GID)bxg8x2SGID;AID:

3.3 Data Encryption

The owner first encrypts the data m by running the encryption algorithm Encrypt. The algorithm takes as inputs a set of public keys fPKg for relevant authorities, the global parameters, the data m and an n_l access matrix M with r mapping its rows to attributes. It chooses a random encryption exponent (x_j)

$\rho(x_j)$ $\rho(x_{n+1})$

AND

Attr2AND

AttrRmAND

$\rho(x_j)$

$\rho(x_j)$ $\rho(x_{n+1})$

Fig. 2. Operations of Boolean Formula

$C_{n+1} = (C_{1;n+1};C_{2;n+1};C_{3;n+1})$ for the new attribute x_{n+1} from the component C_j corresponding to the existing attribute x_j.

To achieve this Attr2OR operation on data m, the update key generation algorithm UKGen takes the encryption information EnInfo(m) of the data m and the public keys. It chooses random a_m; r_{n+1} 2 Z_p and generates the update key as

UK_m = (a_m; $UK_{1;m}$ =

e(g;g)$_{a_{x_{n+1}} r_{n+1}}$

e(g;g)

$a_{x_j} r_j a_m$;

$UK_{2;m}$ = $g^{r_{n+1} r_j}$; $UK_{3;m}$ =

$g^{b_{x_{n+1}} r_{n+1}}$

$g^{b_{x_j} r_j a_m}$

)

Then, the data owner will send the tuple (Attr2OR; UK_m) to
the server and ask it to update the ciphertext CT corresponding
to m. The ciphertext updating algorithm CTUpdate constructs
the new ciphertext component $C_{x_{n+1}}$ as follows.
$C_{1;n+1} = (C_{1;j})^{a_m} \_ UK_{1;m} = e(g;g)^{l_{n+1}} \_ e(g;g)^{a_{x_{n+1}} r_{n+1}}$;
$C_{2;n+1} = C_{2;j} \_ UK_{2;m} = g^{r_{n+1}}$;
$C_{3;n+1} = (C_{3;j})^{a_m} \_ UK_{3;m} = g^{b_{x_{n+1}} r_{n+1}} \_ g^{w_{n+1}}$;
where $l_{n+1} = a_m \_ l_j$ and $w_{n+1} = a_m \_ w_j$.
4.1.2 Converting an attribute to an AND gate (Attr2AND)
This Attr2AND operation involves converting an existing attribute x_j(j 2 [1;n]) to an AND gate (x_j ^x_{n+1}) by adding a new attribute x_{n+1}. In this case, the combination of the new
C_j in the ciphertext.
4.1.4 Removing an attribute from an AND gate (AttrRmAND)
To remove an attribute from an AND gate, all the shares should
be re-randomized, such that the correctness requirement can
be satisfied. This can be easily achieved by using the method
of converting a (t; t)-gate to a (t □1; t)-gate which will be described later.

4.2 Updating a LSSS Structure

Access policies can also be expressed in LSSS structure as in
our access control scheme. To convert a LSSS structure (M;r)
to a new LSSS structure (M_0;r_0), it is too costly to choose a
new encryption secret s_0 and re-encrypt the data under the new
access policy. In order to save the communication cost and the
computation cost on data owners, in our method, we do not
change the encryption secret s, such that we can make full
use of the previous ciphertext encrypted under the old policy
(M;r).
To enable the data owner to re-randomize the encryption secret s, the encryption information EnInfo(m) of the data m should also contain two random vectors ~v and ~w, and the
public key of each attribute x is known to the data owner as
(g^{a_x};g^{b_x}). The data owner will run the update key generation
algorithm to construct the update keys and send them to the
cloud server. Upon receiving update keys, the cloud server will
run the ciphertext update algorithm to update ciphertext from
the previous access policy to the new policy. The update key
algorithm and the ciphertext update algorithm are designed as follows.
4.2.1 Update Key Generation
The update key generation algorithm UKGen takes as inputs public keys, the encryption information of data m, and the previous access policy (M;r) and the new one (M_0;r_0).
Suppose the new access policy is described as an $n_0 \_ l_0$ access matrix M_0 with r_0 mapping its rows to attributes. Since the mapping functions r and r_0 are non-injective, we let $num_{r(i);M}$ and $num_{r(i);M_0}$ denote the number of attribute r(i) in M and M_0 respectively.
It first calls the policy comparing algorithm PolicyCompare to compare the new access policy (M_0;r_0) with the previous one (M;r), and outputs three sets of row
indexes

5 Checking on Policy Updating

Focus on solving the policy updating problem in AES systems, and propose a secure and verifiable policy updating outsourcing method. Instead of retrieving and re-encrypting the data, data owners only send policy updating queries to cloud server, and let cloud server update the policies of encrypted data directly, which means that cloud server does not need to decrypt the data before/during the policy updating. Our scheme can not only satisfy all the above requirements, but also avoid the transfer of encrypted data back and forth and minimize the computation work of data owners by making full use of the previously encrypted data under old access policies in the cloud. The contributions of this paper include: 1) We formulate the policy updating problem in AES systems and develop a new method to outsource the policy updating to the server. 2) We propose an expressive and efficient data access control scheme for big data, which enables efficient dynamic policy updating. 3) We design policy updating algorithms for different types of access policies, e.g., Boolean Formulas, LSSS Structure and Access Tree. Compared to the conference version, we also propose an efficient and secure policy checking method that enables data owners to check whether the cipher texts have been updated correctly by cloud server. In this method, we do not require any help of data users, and data owners can check the correctness of the cipher text updating by their own secret keys and checking keys issued by each authority. Our method can also guarantee data owners cannot use their secret keys to decrypt any cipher texts encrypted by other data owners, although their secret keys contain the components associated with all the attributes. Moreover, we discuss some key features of the attribute-based access control scheme and show how it is suitable for big data access control in the cloud. What's more, we also add more performance evaluation on policy updating algorithms and the policy checking method.

Cloud computing is a revolutionary computing paradigm which enables flexible, on-demand and low-cost usage of computing resources. Those advantages, ironically, are the causes of security and privacy problems, which emerge because the data owned by different users are stored in some cloud servers instead of under their own control. To deal with security problems, various schemes based on the Attribute-Based Encryption have been proposed recently. However, the privacy problem of cloud computing is yet to be solved. This paper presents an anonymous privilege control scheme AnonyControl to address not only the data privacy problem in cloud storage, but also the user identity privacy issues in existing access control schemes. By using multiple authorities in cloud computing system, our proposed scheme achieves anonymous cloud data access and fine-grained privilege control. Our security proof and performance analysis shows that AnonyControl is both secure and efficient for cloud computing environment.

The main contributions of this existing are: 1) The proposed scheme is able to protect user's privacy against each single authority. 2) The proposed scheme is tolerant against authority compromise, and compromising of up to (N − 2) authorities does not bring the whole system down. 3) We provide detailed analysis on security and performance to show feasibility of our scheme. 4) We first implement the real toolkit of multi-authority based encryption scheme.

Chase introduced a multi-authority system, where each user has an ID and they can interact with each key generator (authority) using different pseudonyms. One user's different pseudonyms are tied to his private key, but key generators never know about the private keys, and thus they are not able to link multiple pseudonyms belonging to the same user. In fact they are even not able to distinguish the same user in different transactions. Also, the whole attributes set is divided into N disjoint sets and managed by N attributes authorities. That is, an attribute authority will only issue key components which it is in charge of. In this setting, even if an authority successfully guesses a user's ID, it knows only parts of the user's attributes, which are not enough to figure out the user's identity. However, the scheme proposed by Chase et ill-considered the basic threshold-based ABE, Chase introduced a multi-authority system, where each user has an ID and they can interact with each key generator (authority) using different pseudonyms. One user's different pseudonyms are tied to his private key, but key generators never know about the private keys, and thus they are not able to link multiple pseudonyms belonging to the same user. In fact they are even not able to distinguish the same user in different transactions. Also, the whole attributes set is divided into N disjoint sets and managed by N attributes authorities. That is, an attribute authority will only issue key components which it is in charge of. In this setting, even if an authority successfully guesses a user's ID, it knows only parts of the user's attributes, which are not enough to figure out the user's identity. However, the scheme proposed by Chase et al. Considered the basic threshold-based ABE

6 Analysis of Our Scheme

In this paper, we focus on solving the policy updating problem in AES systems, and propose a secure and verifiable policy updating outsourcing method. Instead of retrieving and re-encrypting the data, data owners only send policy updating queries to cloud server, and let cloud server update the policies of encrypted data directly, which means that cloud server does not need to decrypt the data before/during the policy updating. Our scheme can not only satisfy all the above requirements, but also avoid the transfer of encrypted data back and forth and minimize the computation work of data owners by making full use of the previously encrypted data under old access policies in the cloud. The contributions of this paper include: 1) We formulate the policy updating problem in AES systems and develop a new method to outsource the policy updating to the server. 2) We propose an expressive and efficient data access control scheme for big data, which enables efficient dynamic policy updating. 3) We design policy updating algorithms for different types of access policies, e.g., Boolean Formulas, LSSS Structure and Access Tree. Compared to the conference version, we also propose an efficient and secure policy checking method that enables data owners to check whether the cipher texts have been updated correctly by cloud server. In this method, we do not require any help of data users, and data owners can check the correctness of the cipher text updating by their own secret keys and checking keys issued by each authority. Our method can also guarantee data owners cannot use their secret keys to decrypt any cipher texts encrypted by other data owners, although their secret keys contain the components associated with all the attributes. Moreover, we discuss some key features of the attribute-based access control scheme and show how it is suitable for big data access control in the cloud. What's more, we also add more performance evaluation on policy updating algorithms and the policy checking method.

7 MODULE

- Big Data populate to Cloud Server
- ABE Security Enhancement
- User Interface Designing
- Cloud Storage Enhancement
- Accessory Verify Standards & Get BigData

7.1 BigData populate to CloudServer

The application admin can upload the data contained files to cloud server. The server act as interface between an cloud and ABE processing. The data which is uploading by the admin is to stored cloud server interconnected to database server.

7.2 ABE Security Enhancement

Attribute encryption standard plays vital role in which is populated by the admin is verified and converted to an encrypted format. ABE converts the Big Data to an unsigned format. The un-authorized used not able to view the ABE format. In this module ABE can be implemented.

7.3 User Interface Designing

By using this module End-user can register to access the application .The user profile are maintained by the Database server. DB server verifies the end-user details then after the control allows the user to access the application.

7.4 Cloud Storage Enchancement

In this module the purpose is to identify the performance of cloud storage. We can see the Big data content and the memory occupied by the data. We designed sample demo app to decrypt the data which the data is previously encrypted by the admin.

7.5 Accessor Verify Standards&GetBigData

We are designing this module for End-to-End client-server architecture. Client gives the request to server, and server response back to the client. Client receives the encrypted content

7 CONCLUSION

Focus on solving the policy updating problem in AES systems, and propose a secure and verifiable policy updating outsourcing method. Instead of retrieving and re-encrypting the data, data owners only send policy updating queries to cloud server, and let cloud server update the policies of encrypted data directly, which means that cloud server does not need to decrypt the data before/during the policy updating. Our scheme can not only satisfy all the above requirements, but also avoid the transfer of encrypted data back and forth and minimize the computation work of data owners by making full use of the previously encrypted data under old access policies in the cloud. The contributions of this paper include: 1) We formulate the policy updating problem in AES systems and develop a new method to outsource the policy updating to the server. 2) We propose an expressive and efficient data access control scheme for big data, which enables efficient dynamic policy updating. 3) We design policy updating algorithms for different types of access policies, e.g., Boolean Formulas, LSSS Structure and Access Tree. Compared to the conference version, we also propose an efficient and secure policy checking method that enables data owners to check whether the cipher texts have been updated correctly by cloud server. In this method, we do not require any help of data users, and data owners can check the correctness of the cipher text updating by their own secret keys and checking keys issued by each authority. Our method can also guarantee data owners cannot use their secret keys to decrypt any cipher texts encrypted by other data owners, although their secret keys contain the components associated with all the attributes. Moreover, we discuss some key features of the attribute-based access control scheme and show how it is suitable for big data access control in the cloud. What's more, we also add more performance evaluation on policy updating algorithms and the policy checking method.

REFERENCES

[1] Kan Yong l, O. Pandey, A. Sahai, and B. Waters, "Secure and demonstrable policy update subcontract for big data access control in the cloud" in CCS'06. ACM, 2015, pp. 140–170.
[2] J. Bethencourt, A. Sahai, and B. Waters, "Ciphertext-policy attributebased
encryption," in S&P'07. IEEE, 2007, pp. 321–334.
[3] B. Waters, "Ciphertext-policy attribute-based encryption: An expressive,
efficient, and provably secure realization," in PKC'11. Springer, 2011, pp. 53–70.
[4] A. B. Lewko, T. Okamoto, A. Sahai, K. Takashima, and B. Waters, "Fully secure functional encryption: Attribute-based encryption and (hierarchical) inner product encryption," in EUROCRYPT'10. Springer, 2010, pp. 62–91.
[5] A. B. Lewko and B. Waters, "Decentralizing attribute-based encryption,"
in EUROCRYPT'11. Springer, 2011, pp. 568–588.
[6] S. Yu, C. Wang, K. Ren, and W. Lou, "Achieving secure, scalable, and
fine-grained data access control in cloud computing," in INFOCOM'10. IEEE, 2010, pp. 534–542.
[7] K. Yang, X. Jia, and K. Ren, "Attribute-based fine-grained access control with efficient revocation in cloud storage systems," in AsiaCCS'13. ACM, 2013, pp. 523–528.
[8] K. Yang, X. Jia, K. Ren, B. Zhang, and R. Xie, "DAC-MACS: Effective Data Access Control for Multiauthority Cloud Storage Systems," IEEE Trans. Info. Forensics Security, vol. 8, no. 11, pp. 1790–1801, 2013.
[9] K. Yang and X. Jia, "Expressive, efficient, and revocable data access control for multi-authority cloud storage," IEEE Trans. Parallel Distrib. Syst., vol. 25, no. 7, pp. 1735–1744, July 2014.
[10] A. Sahai, H. Seyalioglu, and B. Waters, "Dynamic credentials and ciphertext delegation for attribute-based encryption," in CRYPTO'12. Springer, 2012, pp. 199–217.
[11] J. C. Benaloh and J. Leichter, "Generalized secret sharing and monotone
functions," in CRYPTO'88, 1988, pp. 27–35.
[12] A. Beimel, "Secure schemes for secret sharing and key distribution," DSc dissertation, 1996.
[13] D. Boneh, X. Boyen, and E.-J. Goh, "Hierarchical identity based encryption with constant size ciphertext," in EUROCRYPT'05, 2005, pp. 440–456