Available online at http://www.journalijdr.com

IJDR

ISSN: 2230-9926

International Journal of Development Research
Vol. 08, Issue, 09, pp.22919-22926, September, 2018

ORIGINAL RESEARCH ARTICLE

OPEN ACCESS

ACCESS CONTROL SCHEME TO BIG DATA USING PRIVACY PRESERVING POLICY

***Sriraksha, N., Surabhi, M., Amrutha, B.R., Rakshitha, M., Asha, R.N.**

Students, Dept of CSE, Global Academy of Technology
Asst Prof., Dept of CSE, Global Academy of Technology

ARTICLE INFO

Article History:

Received 20th June, 2018
Received in revised form
17th July, 2018
Accepted 27th August, 2018
Published online 30th September, 2018

Key Words:

Access control, Attribute bloom filter (ABF), Bigdata, Linear secret-sharing scheme (LSSS) Access structure, Privacy-preserving policy.

ABSTRACT

Controlling the access to a huge amount of big data becomes a very challenging issue, especially when big data are stored in the cloud. Ciphertext-policy attribute-based encryption (CP-ABE) is a encouraging encryption technique that helps end-users to encrypt their data under the access policies defined over some attributes of data consumers and only allows data consumers whose attributes satisfy the access policies to decrypt the data. In CP-ABE, the access policy is attached to the ciphertext in plaintext form, which may also leak some private information about end-users. The attribute values were partially hidden in the already existing systems, while the attribute names are still unprotected. In this paper, we propose access control scheme to big data using privacy preserving policy. Specifically, we hide the whole attribute (rather than only its values) in the access policies. To aid data decryption, we also design a novel attribute bloom filter to evaluate whether an attribute is in the access policy and locate the exact position in the access policy if it is in the access policy. Security analysis and performance evaluation show that our scheme can preserve the privacy from any linear secret-sharing schemes access policy without employing much overhead.

Citation: Sriraksha, N., Surabhi, M., Amrutha, B.R., Rakshitha, M., Asha, R.N. 2018. "Access control scheme to big data using privacy preserving policy", *International Journal of Development Research*, 8, (09), 22919-22926.

INTRODUCTION

In big data, an immense and voluminous amount of data can be generated quickly (e.g., social networks, sensors, machines, smart phones,, etc.)., Conventional computer systems are not competent to store and process these big data. As cloud computing is flexible and elastic computing resources .Cloud computing is a natural fit for and processing big data [1], [2]. With cloud computing, end- users store their data into the cloud, and rely on the cloud server to share their data to other users (data consumers). In order to only share end-users' data to authorized users, it is necessary to the requirements of end-users. When outsourcing data into the cloud, end-users lose the physical control of their data. Moreover, cloud service idersare not fully-trusted by end-users, which makes the access control more challenging. For an instance, if the traditional access control mechanisms (e.g., access control lists) are applied, the cloud server becomes the judge to evaluate the access policy and make access decision. Thus, end-users may worry that the cloud server may make wrong access decision intentionally or unintentionally, and disclose their data to some unauthorized users. In order to enable end-users to control the access of their own data, some attribute-based access control schemes [3]–[5] are proposed by leveraging attribute-based encryption [6], [7]. In attribute-based access control, end-users first define access policies for their data and encrypt the data under these access policies. Only the users whose attributes can satisfy the access policy are eligible to decrypt the data. The existing attribute-based access control schemes can deal with the attribute revocation problem [3]–[5], they all suffer from one problem: the access policy may leak privacy. This is because the access policy is associated with the encrypted data in plaintext form. From the plaintext of access policy, the adversaries may obtain some privacy information about the end-user. For an instance, Alice encrypts her data to enable the "psychology doctor" to access. So, the access pol-icy may contain the attributes "psychology" and "doctor." If anyone sees this data, although he/she may not be able to decrypt the data, he/she still can guess that Alice may suffer from some psychological problems, which leaks the privacy of Alice.

****Corresponding author:*** **Sriraksha, N.,**
Students, Dept of CSE, Global Academy of Technology

To prevent the privacy leakage from the access policy, a straight forward method is to hide the attributes in the access policy. However, when the attributes are hidden, not only the unauthorized users but also the authorized users cannot know which attributes are involved in the access policy, which makes the decryption a challenging problem. Due to this reason, existing methods [8]–[12] do not hide or anonymize the attributes. Instead, they only hide the values of each attribute by using wildcards [8], [9], hidden vector encryption [10].Hiding the values of attributes can somehow protect user privacy, but the attribute name may also leak private information. Most of these partially hidden policy schemes only support specific policy structures (e.g., AND-gates on multivalued attributes). In this paper, we aim to hide the whole attribute instead of only partially hiding the attribute values. Moreover, we do not restrict our method to some specific access structures. The basic idea is to express the access policy in linear secret- sharing scheme (LSSS) access structure (M, ñ) where M is a policy matrix and ñ matches each row Mi of the matrix M to an attribute [6], and hide the attributes by simply removing the attribute matching function ñ. Without the attribute matching function ñ, it is necessary to design an attribute localization algorithm to evaluate whether an attribute is in the access policy and if so find the correct position in the access policy. To this end, we further build a novel attribute bloom filter (ABF) to locate the attributes to the anonymous access policy, which can save a lot of storage overhead and computation cost especially for large attribute universe.

Our contributions are summarized as follows.

- We propose an access control scheme to big data using privacy preserving policy, where the whole attributes are hidden in the access policy rather than only the values of the attributes.
- We also design a novel ABF to evaluate whether an attribute is in the access policy and locate the exact position in the access policy if it is in the access policy.
- We further give the security proof and performance evaluation of our proposed scheme, which demonstrate that our scheme can preserve the privacy from any LSSS access policy without employing much overhead.

Related Work

In order to control the access of their own data stored on untrusted remote servers (e.g., cloud servers), the end-users have used encryption-based access control. It is a very effective method, where data are encrypted by end-users and only authorized users are given decryption keys. This can also prevent the data security during the transmission over wire-less networks which are vulnerable to many threats [13]–[15]. However, traditional public key (PK) encryption methods are not suitable for data encryption because it may produce multiple copies of ciphertext for the same data when there are many data consumers in the system. In order to cope with this issue, some attribute-based access control schemes [3], [5] are pro- posed by leveraging attribute-based encryption [6], which only produces one copy of ciphertext for each data and does not need to know how many intended data consumers during the data encryption. Moreover, once the cloud data are encrypted. Some searchable encryption algorithms [16], [17] are proposed to support search on encrypted cloud data.

Toward this problem, some works [8]–[12], [18]–[21] have been proposed to hide the access policy. In [8], two con-struct ions are proposed to partially hide the access policy. However, the access policy only supports AND-gates on multivalued attributes with wildcards. Li et al. [9] followed this paper and hided the attribute value by using a hash value to denote the value of an attribute. Considering that [8] and [9] are selectively secure, Lai *et al.* [12] proposed a fully secure ciphertext-policy attribute-based encryption (CP-ABE) scheme with partial hidden access policy. However, this scheme is only restricted to a specific access policy (i.e., AND-gates with multivalued attributes with wildcards) as in [8] and [9]. To support more expressive access policy, Lai *et al.* [20] also proposed a method to hide attribute values in access policy expressed in LSSS structure. Besides, there are also some policy hiding schemes using hidden vector encryption [10] and inner product encryption [11]. However, all of these existing schemes can only partially hide the access policy (i.e., hiding the values of the attributes). The attribute names are not hidden in the access policy.

Preliminaries

Linear Secret-Sharing Schemes

Definition 1 (LSSS [6]): A secret sharing schemeMover aset of parties P is called linear over Zp (p is a prime) if: the shares for each party form a vector over Zp; there exists a matrix A called the share-generating matrix for M. The matrix A has l rows and n columns. For =1,...,l, theith row ofAis labeled by a partyñ(i)[ñis a function from {1 ,...,}l to P]. When we consider the column vector v□=(s,r 2 , . . . , rn), where s∈Zpisthe secret to be shared and r 2 ,...,r∈nZp are randomlychosen, then Av is the vector of l shares of the secret s according to M. The share (Av)i belongs to party ñ(i). It is shown in [22] that every linear secret-sharing scheme according to the above definition also enjoys the linear recon-structing property, defined as follows: Suppose that M is an LSSS for access structure A. Let S∈A be an authorized set, and let I⊂{1,2 ,...,l} be defined as I= {i:ñ(i)∈S}. There exist constants {ùi∈Zp}i∈I such that if {ëi} are valid any secret s according to M, then ù ës.

Furtherrmore,∈ these constants ùi can be =found in time polynomial in the shares of *i Ii {}* size of the share{-}generating matrix A. For any unauthorized set, no such constants exist.

Bilinear Pairing

Let G1, G2 and GT be three multiplicative groups with the same prime order p. A bilinear mapping is a mapping eˆ :G1× G2→GTwith the following properties.

Bilinearity: eˆ(ua,vb)=ˆe(u, v)abfor all u ∈G1, v ∈G2and a,b∈Zp.

Non-Degeneracy: There exist u ∈G1, v ∈G2such that eˆ(u, v)≠I where I is the identity element ofGT.

Computability: it can be computed efficiently.

Bloom Filter: The bloom filter (BF) concept is a space-efficient probabilistic data structure, which is used to test whether an element is a member of a set. Specifically, a BF consists of a bit array of m bits and k independent hash functions defined as follows:

hi : {0,1} *›→[1,m]

for 1≤ I ≤ k

Initially, all the positions of the array are set to 0. To add an element e to the set, the BF building algorithm computes all the position indices as $h\{i(e)i\}[1\in,k]$ and sets the values at the corresponding positions in the bit array to 1. Fig. 1 gives an example of BF for set x,{y ,} where the values at positions indexed by $h1(x)$, $h2(x)$, $h3(x)$, $h1(y)$, $h2(y)$, $h3(y)$ are set to 1.

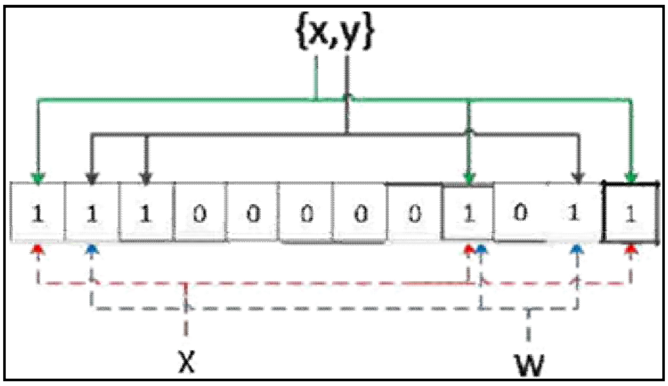

Fig. 1. Example BF for set {*x,y*}

To check whether a given element x belongs to the set S, the BF query algorithm computes all the hash values $h\{i(x)i\}[\in1,k]$ to get k array positions. If any of the bits at these positions are 0, the element x is definitely not in the set. However, if all of the bits are 1, we can say the element x is probably belong to the set S. There is a possibility for some x/S, all of the bits at the corresponding positions of $hi(x)$ are 1, which is called the false positive. For example, the element w in Fig. 1 is not in the set x,y but all the corresponding positions of $hi(w)$ are 1.

Decisional q-BDHE Assumption

The decisional q-bilinear Diffie-Hellman exponent (Decisional q-BDHE) problem is defined as follows. Choose a group G of prime order p according to the security parameter $ë$. Let $a,s\in Z_p$ be chosen at random andg be a generator of G. Let gi denote g^{ai} . When given y=(g,g1, . . . ,gq, ,gq+2, . . . ,g2q,g^s), the adversary mustdistinguish $e\hat{}(g,g)^{aq+1s}\in G_T$ from a random element R in G_T .

An algorithm B has advantage ‹ in solving decisional q- BDHE problem inGif

$_{Pr}Ó_B\ .y,T = e\hat{}(g,g)^{aq+1s}\ Ó = 0Ó- PrB^{Ó},T = R) = 0 \geq o$

Definitions

We will first describe the system model of big data storage and sharing. Then, we define our proposed big data access control scheme and its security model.

Definition of System Architecture: Consider the big data access control system, as shown in Fig. 2. The system consists of five entities, namely cloud servers, attribute authority, end-users, and data consumers.

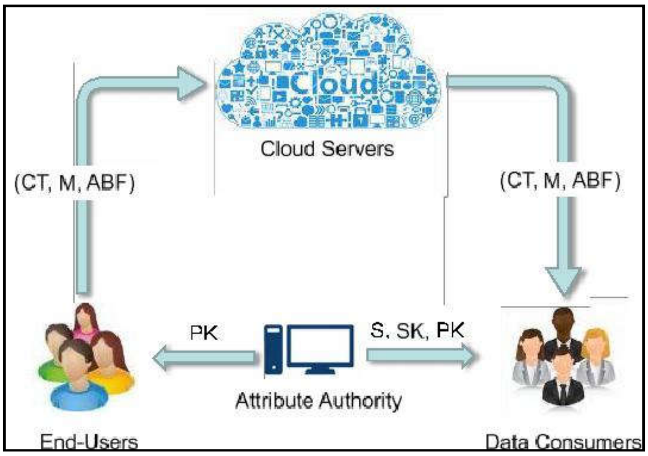

Fig. 2. System Architecture

Cloud Servers: Cloud Servers are employed to store, share and process big data in the system. The cloud servers are managed by cloud service providers, who are not in the same trust domain as end-users. Thus, cloud servers cannot be trusted by end-users to enforce the access policy and make access decisions. We also assume that the cloud server cannot collude with any end-users or data consumers.

Attribute Authority: The attribute authority manages allthe attributes in the system and assigns attributes chosen from the attribute space to end-users. It is also a key generation center, where the public parameters are generated. It also grants different access privileges to end-users by issuing secret keys according to their attributes. The attribute authority is assumed to be fully trusted in the system.

End-User: End-users are the data owners/producers whooutsource their data into the cloud. They also would like to control the access of their data by encrypting the data with CP-ABE. End-users are assumed to be honest in the system.

Data Consumers: Data consumers request the data from cloud servers. Only when their attributes can satisfy the access policies of the data, data consumers can decrypt the data. However, data consumers may try to collude together to access some data that are not accessible individually.

Definition of Our Scheme

Definition 3: Our big data access control scheme consists of the following algorithms: Setup, KeyGen, Encrypt, and Decrypt.

- *Setup($1^ë$) → (PK, MSK):* The setup algorithm takes asinput a security parameter $ë$. It outputs the PK and master secret key.
- *KeyGen(PK, MSK, S) → SK:* The key generationalgorithm takes as inputs the PK, the master keyMSK and a set of attribute S. It outputs the corresponding secret key SK.
- *Encrypt(PK, m, (*M*,ñ)) →(CT, ABF):* The data encryption algorithms contains: data encryptionsubroutine Enc and ABF building subroutine ABFBuild.
- *Enc(PK,m,(*M*,ñ→)) CT:* The data encryption subroutine takes as inputs the PK, the message m and access structure *(*M*,ñ)*. It outputs a cipher-text

*CT.ABFBuild(*M*,ñ)→ABF:* The ABF building sub-routine takes as input the access policy *(*M*, ñ)*. It outputs the ABF.

- *Decrypt(*M*, ABF, PK, SK, CT) → m:* The decryption algorithm consists of two subroutines: ABFQuery and Dec.
- *ABFQuery(S,ABF,PK)→ñ:*The' ABF query algorithm takes as inputs the attribute set *S*, the ABF and the PK. It outputs a reconstructed attribute mapping *ñ*'=(r{ownum,att)}*S*, which shows the corresponding row number in the access matrix M for all the attributes att∈*S*.
- *Dec(SK,CT,(*M*,ñ*'*))→m or⊥:* The data decryp-tion algorithm takes as inputs the secret key *SK*, the ciphertext *CT* as well as the access matrix M and the reconstructed attribute mapping *ñ*'. If the attributes can satisfy the access policy, it outputs the message *m*. Otherwise, it outputs⊥.

Security Model definetion

We consider the indistinguishability against selectively chosen plaintext attacks. It is based on the following game between an adversary *A* and a simulator *B*.

- *Init:* The adversary c*A*hooses a challenge access structure*(*M□*, ñ*□*)*, whereM□is an*l*□*n*□matr×ix, and *ñ*□maps eachrow of M□ to an attribute.
- *Setup:* The challenger runs the Setup algorithm and givesthe public parameters PK to the adversary *A*.
- *Phase 1:* In this phase, the adversary*A*issues queries forsecret keys related to some attributes *S*att.

If *S*att satisfies *(*M∗*, ñ*∗*)*, then abort.

Otherwise, the simulator generates a secret key related to *S*att for the adversary *A*.

Challenge: The adversary*A*submits two equal lengthmessages *m*0 and *m*1 to*B*. The simulator *B* randomly chooses *b*∈0{, 1 }and encrypts *mb* under the challenge access structure *(*M□*,ñ*□*)*. Finally it sends the generated challenge ciphertext*CT*□ to the adversary.

- *Phase 2:* Phase 2 is the same as Phase 1.
- *Guess:* The adversary outputs a guess*b*'of *b*.
- The advantage of *A* in this game is defined as Adv*(A)*=|*Pr*[*b*' = *b*] − 1/2|.

Construction of the Proposed Scheme

The construction of our big data access control is based on the CP-ABE in [6]. However, our access policy privacy pre-serving method can also be applied for any CP-ABE methods with LSSS structured access policies. According to the definition in Section IV-B, our big data access control scheme consists of four phases: 1) system setup; 2) key generation; 3) data encryption, and 4) data decryption.

System Setup

During the system setup phase, the attribute authority runs the Setup algorithm. Let *U* denote the attribute space in the system. Let G and G*T* be cyclic multiplicative groups of prime order*p*, andˆ*e* : G×G→G*T* be a bilinear map. Let *L*att be the maximum bit length of attributes in the system. Let *L*rownum be the maximum bit length of the row numbers of access matrix. Let *L*ABF be the size of bit array of the ABF. Let *k* be the number of hash functions associated with the ABF. The attribute authority randomly chooses a generator *g*∈ G,*á,a*∈ Z□, and *U*= |*U* | random group elements *h*1*, h* 2*, . . . , h U* ∈G. It also generates *k* hash functions*H*1*(), H*2*(), . . . ,Hk()*that maps an element to a position inthe range of [1*,L*ABF].

The PK is published as

PK = (*g,e*ˆ*(g,g)*^{á}*, g*^{a}*, L*att*, L*rownum*, L*ABF *h*1*, h* 2*, . . . ,hU, H*1*(), H*2*(),. ..,Hk()*•.

The master secret key is set as MSK =*g*^{á}.

Key Generation: Each data consumer should register and authenticate to the attribute authority. If the data consumer is not legal, it aborts. Otherwise, the attribute authority will evaluate the role of the data consumer in the system and assign a set of attributes *S* chosen from the attribute space*U*[1]to this data consumer. Together with these attributes, the authority also generates a corresponding secret key for this data consumer by running the following algorithm.

KeyGen(PK, MSK, S) → SK: The algorithm takes asinput the PK, the master key MSK and a set of attributes *S*. It computes

.   Σ

K =$g^{á}g^{at}$, *L* =g^{t},   *Kx*=h^{t}_{x}*x*∈*S*

where*t*∈Z_p□ is chosen at random. Finally, the secret key is set as

SK = (*K, L,*{*Kx*}*x*∈*S, S*•.

Data Encryption

Before outsourcing data into the cloud, end-users encrypt the data by running the Encrypt algorithm. It first calls the data encryption subroutine to encrypt the data into cipher-texts under access policies expressed in LSSS structure. Other access structure, such as Boolean Formulas and Threshold Gates, can also be transformed into LSSS structure.

1) Enc(PK, m, (M,ñ)) → CT: The data encryption sub-encryption secret s ∈ Z∗ randomly and then selects A random vector í = (s, yp y), where y, . . . ,y are ∗ 2, . . . , n2n routine takes as inputs the PK, the message *m* and access structure *(*M*,ñ)*. As shown in Fig. 3, M is an×*l n* access matrix and the injective function *ñ* maps rows of M to attributes. The algorithm first chooses an used to [1]The attribute space should be large such that it would be time- consuming for cloud servers to exhaustively search the attribute space.

Share the encryption secret *s*. For *i*=1 ,...,*l*, it calcu-lates*ëi*=*Mií*·,□where *Mi* is the vector corresponding to the *i*th row of M. Then, it outputs the ciphertext as

In traditional attribute-based encryption scheme, the access policy *(*M*,ñ)* will be attached to the ciphertext CT. However, the access policy is in plaintext, which may leak some private information about the end-users. Based on our observation, the attributes are leaked from the attribute mapping function *ñ*. So, in order to prevent the privacy leakage, we remove this attribute mapping function ñ. However, when ñ is removed, it becomes difficult for data consumers to decrypt the data, as they do not know which attributes are involved in the access policy. To cope with this problem, we propose an efficient attribute localization algorithm by utilizing the BF. However, traditional BF only provides the membership query for a large set, while our purpose goes further: we not only need to evaluate whether an attribute is in the access policy, but also need to locate the attribute to the precise row number in the access matrix.

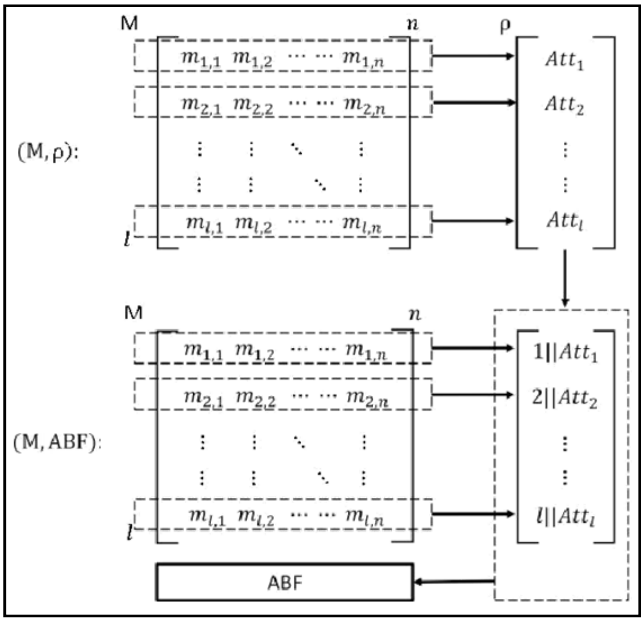

Fig. 3. LSSS access policy and ABF

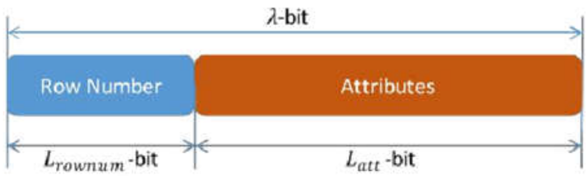

Fig. 4.ë-bit Element of ABF with Lrownum-bit row number string and Latt(=ë−Lrownum)-bit attribute string

Moreover, due to the false positive property, traditional BF cannot be applied for the attribute localization. To this end, we employ a garbled BF [23] as the building block of our attribute localization algorithm (ABF). Instead of using an array of bits in traditional BF, the garbled BF uses an array of ë-bit, where ë is the security parameter. Different from the traditional BF, the false positive probability is much lower, because it not only depends on the collision probability of hash functions, but also depends on the probability of string matching. Although the garbled BF achieves much lower false positive, it is still designed for membership query only. In to precisely locate attributes to the corresponding row number in the access matrix, we employ a specific string as the element of the garbled BF. As shown in Fig. 4, the element is a concatenation of two fixed length strings: one string represents the row number with *L*rownum-bit, and the other string represents the attribute with the bit length of *L*att-bit, where

*L*rownum*L*+att*ë*.=

When the data encryption is finished, the end-users then build the ABF by running the following subroutine.

1) *ABFBuild(*M,*ñ)→ABF:* The ABF building subroutinetakes as input the access policy *(*M,*ñ)*. It first binds the attributes involved in the access policy and its corresponding row number in the access matrix M together and obtains a set of elem=en{t|s|*Se*}∈*i*atte *i* [1,*l*], where the *i*-th row of the access matrix maps to the attrib=uteatte*ñ(i)*. Both of the row number *i* and the attribute att*e* are expanded to the maximum bit length by filling with zeros on the left of the bit strings. By taking the set of elements *Se* as an input, the ABF can be constructed by calling the garbled BF Building algorithm in [23]. To add an element *e* in the set *Se* to the ABF, the algo- rithm first shares the element *e* with *(k,k)* secret sharing scheme by randomly generating *k*− 1 *ë*-bit strings *r*1,*e*, *r*2,*e*,. . . , *rk*−1,*e*, and setting *rk,e*= *r*1,*e* □ *r*2,*e* ··· □ *rk*−1,*e* □ *e.*

Then, it hashes the attribute att*e* associated with the element *e* with *k* independent and unified hash functions *H*1*(), . . . ,Hk()*and gets *H*1*(*att*e), H*2*(*att*e),...,Hk(*att*e)* ‘*r*1,*e*→ *H*1*(*att*e)*position in ABF. *rk,e→Hk(*att*e)*position in ABF.

When we continue to add elements to the ABF, some location *j=Hi(e)* may have been occupied by a previously added element. If such situation happens, we reuse this existing share as one share of the new element. For example, as shown in Fig. 5, the position *Hj(*att*e*$_2$*)*of element *e*2is the same as the position *Hi(*att$_{e1}$*)*of element *e*1. Considering that this position of the ABF has already been occupied by *ri,e*$_1$, instead of *ë*-bit string, we set*rj,e*$_2$*r*=*i,e*$_1$. If we change this positionwith another string, the previously inserted element cannot be recovered.

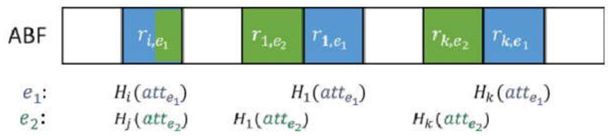

Fig. 5. Example of ABF

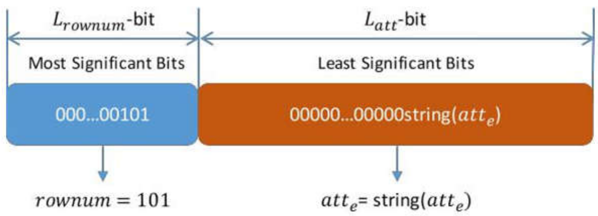

Fig. 6. String abstraction from the element

Algorithm 1 ABFBuild

Input: An LSSS access policy*(*M, *ñ),ë,LABF*
Input: *k*hash functions{*H*1*(),* ··· ,}*Hk()*
Output: ABF

1:Generate an element set Se from the access policy (M, ñ)
2:ABF = new LABF element array of bit strings
3:for i=0 toLABF−1 do
4:ABF[i]=NULLdInitialize the ABF with“NULL”
5:for each elemente=i||atte∈Se do
6: emptyPos= −1,finalShare= x

for i=0 tok−1 do
j = Hi+1(atte) d get the index of the position
if ABF[j]==NULL then
if emptyPos==− 1 then
d reserve this position for the finalShare
emptyPos = j d generate a new share
else
generate a random string rj,e with ë bits
ABF[j]=rj,e
finalShare=finalShare⊕ ABF[j]
else d reuse an existing share
finalShare=finalShare⊕ ABF[j]
ABF[emptyPos] = finalShare
fori= 0 to LABF 1−do
if ABF[i]==NULL then
d fill the empty position with random strings
generate a random string ri with ë bits

The entire ABF building algorithm is shown in Algorithm 1. Finally, the end-users will outsource the data in the form of *(CT,* M, ABF*)* to cloud servers.

Data Decryption

When accessing the data stored in the cloud, data consumers can download the encrypted data according to their interests. However, the access control happens during the decryption, which means that data consumers can decrypt the data only when their attributes can satisfy the access policies used to encrypt the data. In traditional ABE systems, the access policy *(*M*,ñ)* is attached to the ciphertext. So, the data consumers can easily check whether their attributes can satisfy the access policy. However, in our scheme, we hide the attributes mapping function *ñ*, so data consumers should first check which attributes they owned are in the access matrix by running the ABF query subroutine as follows.

1) *ABFQuery(S,*ABF*,PK)* $\rightarrow \tilde{n}^j$*:* It takes as inputs theattribute set *S*, the ABF and the PK. For each attribute att∈*S* owned by the data consumer, the algorithm first computes the position indices by feeding the attribute att with the *k* hash functions *H*1 *(),...,Hk()* and gets *H*1*(*att*), H*2*(*att*),...,Hk(*att*)*.

Then, it fetches the corresponding strings from theposi-tions indexed by *H(*att*) (i*∈ [1*,k*]*)* in the ABF as *I* follows:

*H*1*(*att*)*position in ABF→ *r*1*,e*

*Hk(*att*)*position in ABF→*rk,e.*

After that, it reconstructs the element *e* as *e* = r1,e⊕r2,e⊕ · · · ⊕rk−1,e⊕

$rk,e = r1,e \oplus r2,e \oplus \cdots \oplus rk-1,e \oplus r1,e \oplus r2,e \oplus \cdots \oplus rk-1,e \oplus e.$

Note that the element *e* is in the format of *e*= |*i*| att*e*as shown in Fig. 4. Then, it takes the last *L*att bits from the string *e*, and removes all the zero bits on the left of the string to obtain the string att*e*. As shown in Fig. 6, if att*e* is the same as the attribute att, we say that this attribute att is in the access matrix. Then, it obtains the first *L*rownum bits from the string *e* to obtain the corre-sponding row number by removing all the zero bits at the left as well. Otherwise, att*e* is not the same as the attribute att, it means that the attribute att does not exist in the access policy. Finally, it outputs the reconstructed attribute mapping as which shows the corresponding row number in the access matrix M. The ABF query algorithm is shown in Algorithm 2.

$\tilde{n}^j$= {*(*rownum*,* att*)*}att∈*S*

When obtaining the access policy *(*M*, ñ)*, the data consumer can run the data decryption subroutine as in traditional attribute-based encryption systems.

*Dec(SK, CT, (*M*,*$\tilde{n}^j$*))* → *m* or⊥*:* The data decryptionalgorithm takes as inputs the secret key *SK*, the ciphertext*CT* as well as the access matrix M and the reconstructed attribute mapping $\tilde{n}^j$. If the attributes can satisfy the access policy, it can leverage the Lagrange Interpolation Formula to find coefficients {*ci*|*i*∈*I*} such that_*i*∈*Iciλi*= *s,* where *I* = {*i* : *ρ(i)* ∈*S*} ⊂ {1, 2, . . . ,*l*}. recover the data as *m* = *C/`e(g, g)αs.* Otherwise, it outputs⊥to denote that the decryption fails.

Algorithm 2 ABFQuery

Input: An Attribute Bloom FilterABF, a set of attributesS
Input: khash functions{H1(),•••,Hk()}
Input: Maximum attribute string lengthLatt
Input: Maximum row number string lengthLrownum
Output: ñj= {(rownum,att)}att∈S

a) for eachatt∈S do
b) ReStr= {0}ë d initialize the reconstructed string
c) for i=0 tok−1 do
d) j= Hi+1(att)d get the index of the position
e) ReStr=ReStr⊕ ABF[j]
f) atteStr=LSBLatt(ReStr)
g) d get Lattleast significant bits
h) atte=RmLeadingZeroBits(atteStr)
i) d remove all the leading zero bits
j) if atte==att then
k) rownumStr= MSBLrownum(ReStr)
l) get Lrownum most signdificant bits
m) rownum= RmLeadingZeroBits(rownumStr)
n) remove all the leadindg zero bits
o) Add(rownum, att)intoñ j

Analysis of Our Scheme

Security Analysis

Theorem 1: No polynomial time adversary can selectivelybreak our big data access control scheme with an *l*n×*(n*q)*ch≤allenge access matrix, under the decisional *q*-BDHEassumption.

Proof: Our big data access control scheme is constructedon top of the attribute-based encryption scheme in [6], which is proved to be selective secure against the chosen plaintext attacks under the decisional *q*-BDHE assumption. It is shown in [6] that if there is an adv *A*ersary with non-negligible advanta=ge‹*Adv* in the selective security game (which is the *A* same as the security game defined in Section IV-C), they can build a simulator *B* that solves the decisional *q*-BDHE problem with non-negligible advantages. Similarly, to prove the security of our big data access control scheme, we show that if there is an adversary *A*y with non-negligible advantage‹=*Adv*$_A$in the selective security game, we can build a simulator ^{j}tBhat also solves the decisional q-BDHE problem with non-negligible advantages. The construction of B^j is similar to the simulator B in [6]. The Init phase in the B^j is the same as the one in the B. In the Setup j phase, besides the steps from B, The secret key query phases569 are also the same, which means that B^j.Phase1 =B.Phase1 and B^j.Phase2 =B.Phase2. The differences are in the Challenge phase: the encryption algorithm in B^j consists of two subroutines. To simulate the ABF building subroutine, the simulator B^j queries from the ABF Build oracle. As for the data encryption subroutine, ^{Bj}.Enc${}^{=B}$.Encrypt. Because the challenge matrix is selected by the adversary before the Initphase, so the constructed ABF is the same no matterwhich plaintext is selected for encryption, which means that the ABF will not increase the advantages of the adversary iAn the security game. Similar to the proof in [6], we can show that B^j plays the q-BDHE problem with non-negligible advantages.

Theorem 2: Our big data access control scheme is privacy-preserving against the adversaries with polynomial time in the security parameter *ë*.

Proof: In our scheme, only the data consumers whohold the attributes can obtain the string of attribute from the attribute space *U*. Adversaries who have no knowledge about the attribute string cannot launch the brute force attack to guess the attribute string within polynomial time. So, they cannot obtain the private information from the access policy consisting of the matrix M and the ABF.

Data consumers are only allowed to check whether their owned attributes are in the access policy. Unless the data consumer has all the attributes of the attribute space or several data consumers collude together, they cannot check all the attributes from the attribute space in the system. Since the ABF is constructed with a garbled BF where *ë*-bit strings are embedded into the BF, the false positive probability of the *ë*.
ABF can be reduced to *(1/2)*.

Performance Analysis: To resist the privacy leakage from the access policy, we employ an ABF to enable data consumers to locate the position of attributes in the access policy. Specifically, the ABF building algorithm is added during the data encryption and the ABF query algorithm is added during the data decryption. In order to show how much computation overhead incurred by the ABF, we do the experiment on a Unix system with an Intel Core i5 CPU at 2.4 GHz and 8.00 GB RAM. The code uses the pairing-based cryptography library version 0.5.12, and a symmetric elliptic curve *á*-curve, where the base field size is 512-bit and the embedding degree is 2, such that the security parameter is equal to 1024-bit. To implement the ABF, we employ the MurmurHash created by Austin Appleby in 2008.[2] All the experimental results are the mean of 20 trials. Fig. 7(a) shows the encryption time versus the number of attributes involved in the access policy. The traditional ABE line in Fig. 7(a) is the implementation of the ABE without privacy-preserving policy from the [6]. The encryption time in our scheme consists of both ABF building and data encryption. The lines of our scheme in this figure apply eight hash functions and 16 hash functions to build ABF, respectively. Fig. 7(b) shows the decryption time versus the number of attributes involved in the decryption. The decryption time in our scheme consists of both the ABF query time and data decryption time. The attribute number here also means how many attributes are tested by running the ABF query algorithm. Therefore, our scheme can preserve the privacy of the access policy without increasing much computation overhead for both data encryption on end-users and data decryption on data consumers.

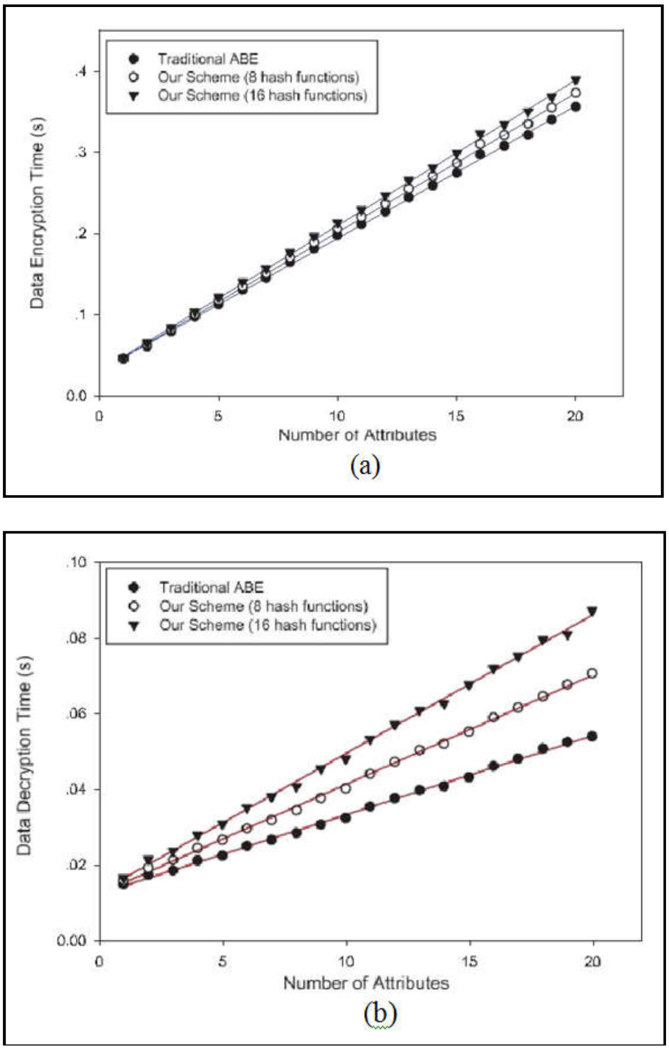

Fig. 7. Computation time comparison between the ABE in [6] and our scheme (data size: 1 KB, security parameter: 1024). (a) Data encryption. (b) Datadecryption.

Conclusion

In this paper, an efficient and fine-grained data access control scheme for big data, where the access pol- icy will not leak any privacy information. Different from the existing methods which only partially hide the attribute values in the access policies, our method can hide the whole attribute (rather than only its values) in the access policies. However, this may lead to great challenges and difficulties for legal data consumers to decrypt data. To cope with this problem, we have also designed an attribute localization algorithm to evaluate whether an attribute is in the access policy. In order to improve the efficiency, a novel ABF has been designed to locate the precise row numbers of attributes in the access matrix. We have also demonstrated that our scheme is selectively secure against chosen plaintext attacks. Moreover, we have implemented the ABF by using MurmurHash and the access control scheme to show that our scheme can preserve the privacy from any LSSS access policy without employing much overhead. In our future work, we will focus on how to deal with the offline attribute guessing attack that check the guessing "attribute strings" by continually querying the ABF.

REFERENCES

[1] P. Mell and T. Grance, "The NIST definition of cloud computing," Recommendations Nat. Inst. Standards Technol., NIST, Washington, DC, USA, Tech. Rep. 800-145, 2011.

[2] R. Lu, H. Zhu, X. Liu, J. K. Liu, and J. Shao, "Toward efficient and privacy-preserving computing in big data era," *IEEE Netw.*, vol. 28, no. 4, pp. 46–50, Jul./Aug. 2014.

[3] 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, Jul. 2014.

[4] H. Li, D. Liu, K. Alharbi, S. Zhang, and X. Lin, "Enabling fine-grained access control with efficient attribute revocation and policy updating in smart grid," *KSII Trans. Internet Inf. Syst.*, vol. 9, no. 4, pp. 1404–1423, 2015.

[5] K. Yang, Z. Liu, X. Jia, and X. S. Shen, "Time-domain attribute-based access control for cloud-based video content sharing: A cryptographic approach," *IEEE Trans. Multimedia*, vol. 18, no. 5, pp. 940–950, May 2016.

[6] B. Waters, "Ciphertext-policy attribute-based encryption: An expressive, efficient, and provably secure realization," in *Proc. PKC*, Taormina, Italy, 2011, pp. 53–70.

[7] H. Lin, Z. Cao, X. Liang, and J. Shao, "Secure threshold multi author-ity attribute based encryption without a central authority," in *Proc.INDOCRYPT*, Kharagpur, India, 2008, pp. 426–436.

[8] T. Nishide, K. Yoneyama, and K. Ohta, "Attribute-based encryption with partially hidden encryptor-specified access structures," in *AppliedCryptography and Network Security*. Heidelberg, Germany: Springer,2008, pp. 111–129.

[9] J. Li, K. Ren, B. Zhu, and Z. Wan, "Privacy-aware attribute-based encryption with user accountability," in *Proc. Inf. Security*, Pisa, Italy, 2009, pp. 347–362.

[10] D. Boneh and B. Waters, "Conjunctive, subset, and range queries on encrypted data," in *Theory of Cryptography*. Heidelberg, Germany: Springer, 2007, pp. 535–554.

[11] J. Katz, A. Sahai, and B. Waters, "Predicate encryption supporting dis-junctions, polynomial equations, and inner products," in *Advances inCryptology–EUROCRYPT 2008*. Heidelberg, Germany: Springer, 2008,pp. 146–162.

[12] J. Lai, R. H. Deng, and Y. Li, "Fully secure cipertext-policy hiding CP-ABE," in *Information Security Practice and Experience*. Heidelberg, Germany: Springer, 2011, pp. 24–39.

[13] L. Lei, Z. Zhong, K. Zheng, J. Chen, and H. Meng, "Challenges on wireless heterogeneous networks for mobile cloud computing," *IEEEWireless Commun.*, vol. 20, no. 3, pp. 34–44, Jun. 2013.

[14] K. Zheng*et al.*, "Big data-driven optimization for mobile networks toward 5G," *IEEE Netw.*, vol. 30, no. 1, pp. 44–51, Jan./Feb. 2016.

[15] Z. Su, Q. Xu, and Q. Qi, "Big data in mobile social networks: A QoE-oriented framework," *IEEE Netw.*, vol. 30, no. 1, pp. 52–57, Jan./Feb. 2016.

[16] H. Li, D. Liu, Y. Dai, and T. H. Luan, "Engineering searchable encryp-tion of mobile cloud networks: When QoE meets QoP," *IEEE WirelessCommun.*, vol. 22, no. 4, pp. 74–80, Aug. 2015.

[17] H. Li *et al.*, "Enabling fine-grained multi-keyword search support-ing classified sub-dictionaries over encrypted cloud data," *IEEE Trans.Depend. Secure Comput.*, vol. 13, no. 3, pp. 312–325, May/Jun. 2016,doi: 10.1109/TDSC.2015.2406704.

[18] K. Frikken, 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, Oct. 2006.

[19] S. Yu, K. Ren, and W. Lou, "Attribute-based content distribution with hidden policy," in *Proc. Secure Netw. Protocols (NPSec)*, Orlando, FL, USA, 2008, pp. 39–44

****