An Efficient and Fine-Grained Big Data Access Control Scheme With Privacy-Preserving Policy

Kan Yang, Qi Han, Hui Li, *Member, IEEE*, Kan Zheng, *Senior Member, IEEE*, Zhou Su, *Member, IEEE*, and Xuemin Shen, *Fellow, IEEE*

***Abstract*—How to control the access of the 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 promising encryption technique that enables 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. Existing methods only partially hide the attribute values in the access policies, while the attribute names are still unprotected. In this paper, we propose an efficient and fine-grained big data access control scheme with privacy-preserving policy. Specifically, we hide the whole attribute (rather than only its values) in the access policies. To assist 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.**

***Index Terms*—Access control, attribute bloom filter (ABF), big data, linear secret-sharing scheme (LSSS) access structure, privacy-preserving policy.**

I. Introduction

IN THE era of big data, a huge amount of data can be generated quickly from various sources (e.g., smart phones, sensors, machines, social networks, etc.). Toward these big data, conventional computer systems are not competent to store and process these data. Due to the flexible and elastic computing resources, cloud computing is a natural fit for storing 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 design access control mechanisms according to the requirements of end-users.

When outsourcing data into the cloud, end-users lose the physical control of their data. Moreover, cloud service providers are not fully-trusted by end-users, which makes the access control more challenging. For example, 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.

Although 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 example, Alice encrypts her data to enable the "psychology doctor" to access. So, the access policy 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.

To prevent the privacy leakage from the access policy, a straightforward 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], and inner product encryption [11], [12]. Hiding the values of attributes can somehow protect user privacy, but the attribute

Manuscript received February 28, 2016; revised April 27, 2016; accepted May 15, 2016. Date of publication May 23, 2016; date of current version April 28, 2017. This work was supported in part by the NSERC Canada, and in part by the NSFC China under Grant U1401251 and Grant 61571286. *(Corresponding author: Qi Han.)*

K. Yang and X. Shen are with the Department of Electrical and Computer Engineering, University of Waterloo, Waterloo, ON N2J 3G1, Canada (e-mail: kan.yang@uwaterloo.ca; sshen@uwaterloo.ca).

Q. Han and H. Li are with the School of Communication Engineering, Xidian University, Xi'an 710071, China (e-mail: hanqiwildginger@gmail.com; lihui@mail.xidian.edu.cn).

K. Zheng is with the School of Information and Telecommunication Engineering, Beijing University of Posts and Telecommunications, Beijing, China (e-mail: kzheng@ieee.org).

Z. Su is with the School of Mechatronic Engineering and Automation, Shanghai University, Shanghai 200072, China (e-mail: zhousu@ieee.org).

Digital Object Identifier 10.1109/JIOT.2016.2571718

2327-4662 © 2016 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

name may also leak private information. Moreover, 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 $(\mathbb{M}, \rho)$ where $\mathbb{M}$ is a policy matrix and ρ matches each row M_i of the matrix $\mathbb{M}$ to an attribute [6], and hide the attributes by simply removing the attribute matching function ρ. Without the attribute matching function ρ, 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.

1) We propose an efficient and fine-gained big data access control scheme with privacy-preserving policy, where the whole attributes are hidden in the access policy rather than only the values of the attributes.
2) 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.
3) 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.

The remainder of this paper is organized as follows. We first describe the related work in Section II. In Section III, we introduce some preliminary knowledge. Section IV first defines the system model, and then defines our scheme and its security model. The detailed construction of our scheme is described in Section V. Section VI provides the security analysis and performance evaluation of our scheme. Finally, the conclusion is drawn in Section VII.

II. Related Work

In order to enable end-users to control the access of their own data stored on untrusted remote servers (e.g., cloud servers), encryption-based access control is an 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 wireless 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 proposed 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 constructions 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.

III. Preliminaries

A. Linear Secret-Sharing Schemes

Definition 1 (LSSS [6]): A secret sharing scheme Π over a set of parties $\mathbb{P}$ is called linear over $\mathbb{Z}_p$ (p is a prime) if:

1) the shares for each party form a vector over $\mathbb{Z}_p$;
2) there exists a matrix A called the share-generating matrix for Π. The matrix A has l rows and n columns. For $i = 1, \ldots, l$, the ith row of A is labeled by a party $\rho(i)[\rho$ is a function from $\{1, \ldots, l\}$ to $\mathbb{P}]$. When we consider the column vector $\vec{v} = (s, r_2, \ldots, r_n)$, where $s \in \mathbb{Z}_p$ is the secret to be shared and $r_2, \ldots, r_n \in \mathbb{Z}_p$ are randomly chosen, then $A\vec{v}$ is the vector of l shares of the secret s according to Π. The share $(A\vec{v})_i$ belongs to party $\rho(i)$.

It is shown in [22] that every linear secret-sharing scheme according to the above definition also enjoys the linear reconstruction property, defined as follows: Suppose that Π is an LSSS for access structure $\mathbb{A}$. Let $S \in \mathbb{A}$ be an authorized set, and let $I \subset \{1, 2, \ldots, l\}$ be defined as $I = \{i{:}\rho(i) \in S\}$. There exist constants $\{\omega_i \in \mathbb{Z}_p\}_{i\in I}$ such that if $\{\lambda_i\}$ are valid shares of any secret s according to Π, then $\sum_{i\in I} \omega_i\lambda_i = s$. Furthermore, these constants $\{\omega_i\}$ can be found in time polynomial in the size of the share-generating matrix A. For any unauthorized set, no such constants exist.

B. Bilinear Pairing

Let $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ be three multiplicative groups with the same prime order p. A bilinear mapping is a mapping $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ with the following properties.

1) *Bilinearity*: $\hat{e}(u^a, v^b) = \hat{e}(u, v)^{ab}$ for all $u \in \mathbb{G}_1$, $v \in \mathbb{G}_2$ and $a, b \in \mathbb{Z}_p$.
2) *Non-Degeneracy*: There exist $u \in \mathbb{G}_1$, $v \in \mathbb{G}_2$ such that $\hat{e}(u, v) \neq I$, where I is the identity element of $\mathbb{G}_T$.
3) *Computability*: $\hat{e}$ can be efficiently computed.

Such a bilinear mapping is called a bilinear pairing.

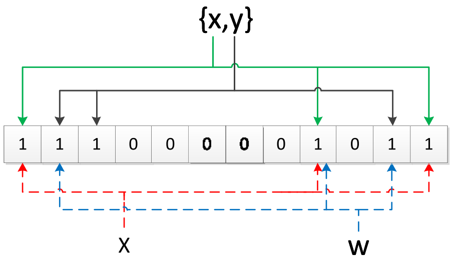


Fig. 1. Example BF for set $\{x, y\}$.

C. Bloom Filter

The concept of bloom filter (BF), proposed by Bloom [23] in 1970, 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: $h_i : \{0, 1\}^* \mapsto [1, m]$ for $1 \leq i \leq k$.

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 $\{h_i(e)\}_{i\in[1,k]}$ 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 $h_1(x)$, $h_2(x)$, $h_3(x)$, $h_1(y)$, $h_2(y)$, $h_3(y)$ are set to 1.

To check whether a given element x belongs to the set S, the BF query algorithm computes all the hash values $\{h_i(x)\}_{i\in[1,k]}$ 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 $x \notin S$, all of the bits at the corresponding positions of $h_i(x)$ 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 $h_i(w)$ are 1.

D. Decisional q-BDHE Assumption

The decisional q-bilinear Diffie-Hellman exponent (Decisional q-BDHE) problem is defined as follows.

Choose a group $\mathbb{G}$ of prime order p according to the security parameter λ. Let $a, s \in \mathbb{Z}_p^*$ be chosen at random and g be a generator of $\mathbb{G}$. Let g_i denote g^{a^i}. When given $\vec{y} = (g, g_1, \ldots, g_q, \ , g_{q+2}, \ldots, g_{2q}, g^s)$, the adversary must distinguish $\hat{e}(g, g)^{a^{q+1}s} \in \mathbb{G}_T$ from a random element R in $\mathbb{G}_T$.

An algorithm $\mathcal{B}$ has advantage ϵ in solving decisional q-BDHE problem in $\mathbb{G}$ if

$$\left|Pr\left[\mathcal{B}\left(\vec{y}, T = \hat{e}(g, g)^{a^{q+1}s}\right) = 0\right] - Pr\left[\mathcal{B}(\vec{y}, T = R) = 0\right]\right| \geq \epsilon.$$

Definition 2: We say that the Decisional q-BDHE assumption holds if no polynomial time algorithm has a non-negligible advantage in solving the q-BDHE problem.

IV. Definitions

In this section, we 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.

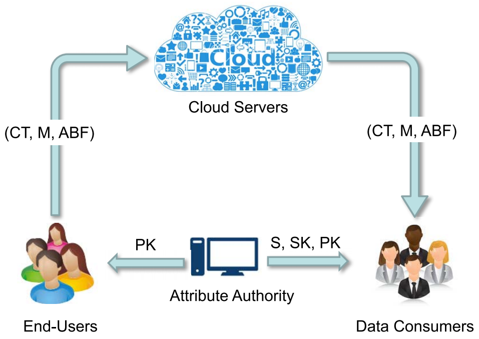


Fig. 2. System model.

A. Definition of System Model

We 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.

1) 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.

2) Attribute Authority: The attribute authority manages all the 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.

3) End-User: End-users are the data owners/producers who outsource 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.

4) 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.

B. Definition of Our Scheme

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

- *Setup(1^λ) $\to$ (PK, MSK):* The setup algorithm takes as input a security parameter λ. It outputs the PK and master secret key.
- *KeyGen(PK, MSK, S) $\to$ SK:* The key generation algorithm takes as inputs the PK, the master key MSK and a set of attribute S. It outputs the corresponding secret key SK.
- *Encrypt($PK, m, (\mathbb{M}, \rho)$) $\to$ (CT, ABF):* The data encryption algorithms contains: data encryption subroutine Enc and ABF building subroutine ABFBuild.

  - *Enc($PK, m, (\mathbb{M}, \rho)$) $\rightarrow$ CT:* The data encryption subroutine takes as inputs the PK, the message m and access structure $(\mathbb{M}, \rho)$. It outputs a ciphertext CT.
  - *ABFBuild($\mathbb{M}, \rho$) $\rightarrow$ ABF:* The ABF building subroutine takes as input the access policy $(\mathbb{M}, \rho)$. It outputs the ABF.
- *Decrypt($\mathbb{M}, ABF, PK, SK, CT$) $\rightarrow$ m:* The decryption algorithm consists of two subroutines: ABFQuery and Dec.
  - *ABFQuery(S, ABF, PK) $\rightarrow$ ρ':* The ABF query algorithm takes as inputs the attribute set S, the ABF and the PK. It outputs a reconstructed attribute mapping $\rho' = \{(\text{rownum}, \text{att})\}_S$, which shows the corresponding row number in the access matrix $\mathbb{M}$ for all the attributes att $\in S$.
  - *Dec($SK, CT, (\mathbb{M}, \rho')$) $\rightarrow$ m or $\perp$:* The data decryption algorithm takes as inputs the secret key SK, the ciphertext CT as well as the access matrix $\mathbb{M}$ and the reconstructed attribute mapping ρ'. If the attributes can satisfy the access policy, it outputs the message m. Otherwise, it outputs $\perp$.

C. Definition of Security Model

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

- *Init:* The adversary $\mathcal{A}$ chooses a challenge access structure $(\mathbb{M}^*, \rho^*)$, where $\mathbb{M}^*$ is an $l^* \times n^*$ matrix, and ρ^* maps each row of $\mathbb{M}^*$ to an attribute.
- *Setup:* The challenger runs the Setup algorithm and gives the public parameters PK to the adversary $\mathcal{A}$.
- *Phase 1:* In this phase, the adversary $\mathcal{A}$ issues queries for secret keys related to some attributes S_{att}.
  - If S_{att} satisfies $(\mathbb{M}^*, \rho^*)$, then abort.
  - Otherwise, the simulator generates a secret key related to S_{att} for the adversary $\mathcal{A}$.
- *Challenge:* The adversary $\mathcal{A}$ submits two equal length messages m_0 and m_1 to $\mathcal{B}$. The simulator $\mathcal{B}$ randomly chooses $b \in \{0, 1\}$ and encrypts m_b under the challenge access structure $(\mathbb{M}^*, \rho^*)$. 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 $\mathcal{A}$ in this game is defined as $\text{Adv}(\mathcal{A}) = |Pr[b' = b] - 1/2|$.

V. 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 preserving 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.

A. System Setup

During the system setup phase, the attribute authority runs the Setup algorithm. Let $\mathcal{U}$ denote the attribute space in the system. Let $\mathbb{G}$ and $\mathbb{G}_T$ be cyclic multiplicative groups of prime order p, and $\hat{e} : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$ 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 $g \in \mathbb{G}$, $\alpha, a \in \mathbb{Z}_p^*$, and $U = |\mathcal{U}|$ random group elements $h_1, h_2, \ldots, h_U \in \mathbb{G}$. It also generates k hash functions $H_1(), H_2(), \ldots, H_k()$ that maps an element to a position in the range of $[1, L_{\text{ABF}}]$.

The PK is published as

$$\begin{aligned}\text{PK} = \langle g, \hat{e}(g, g)^{\alpha}, g^a, L_{\text{att}}, L_{\text{rownum}}, L_{\text{ABF}}\\ h_1, h_2, \ldots, h_U,\ H_1(), H_2(), \ldots, H_k()\ \rangle.\end{aligned}$$

The master secret key is set as $\text{MSK} = g^{\alpha}$.

B. 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 $\mathcal{U}$[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.

1) *KeyGen(PK, MSK, S) $\rightarrow$ SK:* The algorithm takes as input the PK, the master key MSK and a set of attributes S. It computes

$$K = g^{\alpha} g^{at}, L = g^t, \left\{K_x = h_x^t\right\}_{x \in S}$$

where $t \in \mathbb{Z}_p^*$ is chosen at random. Finally, the secret key is set as

$$SK = \langle K, L, \{K_x\}_{x \in S}, S \rangle.$$

C. 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 ciphertexts under access policies expressed in LSSS structure. Other access structure, such as Boolean Formulas and Threshold Gates, can also be transformed into LSSS structure [24].

1) *Enc($PK, m, (\mathbb{M}, \rho)$) $\rightarrow$ CT:* The data encryption subroutine takes as inputs the PK, the message m and access structure $(\mathbb{M}, \rho)$. As shown in Fig. 3, $\mathbb{M}$ is an $l \times n$ access matrix and the injective function ρ maps rows of $\mathbb{M}$ to attributes. The algorithm first chooses an encryption secret $s \in \mathbb{Z}_p^*$ randomly and then selects a random vector $\vec{v} = (s, y_2, \ldots, y_n)$, where $y_2, \ldots, y_n$ are 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.

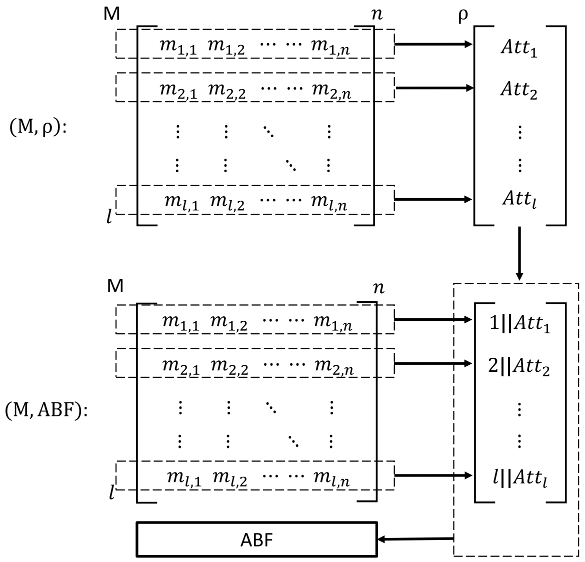

(M, ρ):
M
n
ρ
$m_{1,1}$ $m_{1,2}$ ⋯ ⋯ $m_{1,n}$
$m_{2,1}$ $m_{2,2}$ ⋯ ⋯ $m_{2,n}$
$m_{l,1}$ $m_{l,2}$ ⋯ ⋯ $m_{l,n}$
l
Att_1
Att_2
Att_l
(M, ABF):
M
n
$1||Att_1$
$2||Att_2$
$l||Att_l$
ABF

Fig. 3. LSSS access policy and ABF.

share the encryption secret s. For $i = 1, \ldots, l$, it calculates $\lambda_i = M_i \cdot \vec{v}$, where M_i is the vector corresponding to the ith row of $\mathbb{M}$. Then, it outputs the ciphertext as

$$CT = \Big\langle C = m\hat{e}(g, g)^{\alpha s}$$

$$C' = g^s, \left\{C_i = g^{a\lambda_i} h_{\rho(i)}^{-s}\right\}_{i=1,\ldots,l}\Big\rangle.$$

In traditional attribute-based encryption scheme, the access policy $(\mathbb{M}, \rho)$ 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 ρ. So, in order to prevent the privacy leakage, we remove this attribute mapping function ρ. However, when ρ 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. Moreover, due to the false positive property, traditional BF cannot be applied for the attribute localization. To this end, we employ a garbled BF [25] 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 λ-bit, where λ 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 order 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 $L_{\text{rownum}} + L_{\text{att}} = \lambda$.

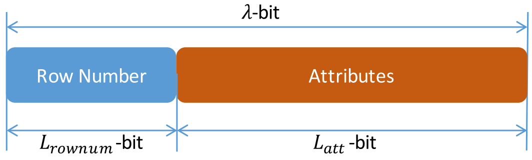


Fig. 4. λ-bit Element of ABF with L_{rownum}-bit row number string and $L_{\text{att}}(= \lambda - L_{\text{rownum}})$-bit attribute string.

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

1) *ABFBuild*$(\mathbb{M}, \rho) \rightarrow$ *ABF:* The ABF building subroutine takes as input the access policy $(\mathbb{M}, \rho)$. It first binds the attributes involved in the access policy and its corresponding row number in the access matrix $\mathbb{M}$ together and obtains a set of elements $S_e = \{i||\text{att}_e\}_{i\in[1,l]}$, where the i-th row of the access matrix maps to the attribute $\text{att}_e = \rho(i)$. 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 S_e as an input, the ABF can be constructed by calling the garbled BF Building algorithm in [25]. To add an element e in the set S_e to the ABF, the algorithm first shares the element e with (k, k) secret sharing scheme by randomly generating $k - 1$ λ-bit strings $r_{1,e}$, $r_{2,e}$, ..., $r_{k-1,e}$, and setting

$$r_{k,e} = r_{1,e} \oplus r_{2,e} \cdots \oplus r_{k-1,e} \oplus e.$$

Then, it hashes the attribute att_e associated with the element e with k independent and unified hash functions $H_1(), \ldots, H_k()$ and gets

$$H_1(\text{att}_e), H_2(\text{att}_e), \ldots, H_k(\text{att}_e)$$

where each $H_i(\text{att}_e)$ $(i \in [1, k])$ represents the position index of ABF. As shown in Fig. 5, it then stores the ith element share r_i to the position indexed by $H_i(\text{att}_e)$ in the ABF as

$$r_{1,e} \rightarrow H_1(\text{att}_e) \text{ position in ABF}$$
$$\vdots$$
$$r_{k,e} \rightarrow H_k(\text{att}_e) \text{ position in ABF.}$$

When we continue to add elements to the ABF, some location $j = H_i(e)$ 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 $H_j(\text{att}_{e_2})$ of element e_2 is the same as the position $H_i(\text{att}_{e_1})$ of element e_1. Considering that this position of the ABF has already been occupied by r_{i,e_1}, instead of randomly selecting a

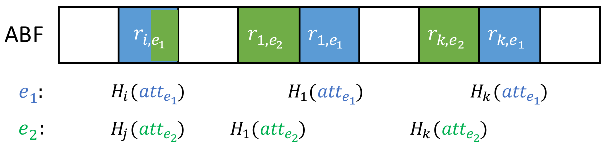


Fig. 5. Example of ABF.

Algorithm 1 ABFBuild

```
Input: An LSSS access policy $(\mathbb{M}, \rho)$, $\lambda$, $L_{ABF}$
Input: $k$ hash functions $\{H_1(), \cdots, H_k()\}$
Output: ABF
Generate an element set $S_e$ from the access policy $(\mathbb{M}, \rho)$
ABF = new $L_{ABF}$ element array of bit strings
for $i = 0$ to $L_{ABF} - 1$ do
    $ABF[i] = NULL$    ▷ Initialize the ABF with "NULL"
for each element $e = i||att_e \in S_e$ do
    $emptyPos = -1$, $finalShare = x$
    for $i = 0$ to $k - 1$ do
        $j = H_{i+1}(att_e)$    ▷ get the index of the position
        if $ABF[j] == NULL$ then
            if $emptyPos == -1$ then
                    ▷ reserve this position for the finalShare
                emptyPos = j
            else    ▷ generate a new share
                generate a random string $r_{j,e}$ with $\lambda$ bits
                $ABF[j] = r_{j,e}$
                $finalShare = finalShare \oplus ABF[j]$
        else    ▷ reuse an existing share
            $finalShare = finalShare \oplus ABF[j]$
    ABF[emptyPos] = finalShare
for $i = 0$ to $L_{ABF} - 1$ do
    if $ABF[i] == NULL$ then
            ▷ fill the empty position with random strings
        generate a random string $r_i$ with $\lambda$ bits
        $ABF[i] = r_i$
```

λ-bit string, we set $r_{j,e_2} = r_{i,e_1}$. If we change this position with another string, the previously inserted element cannot be recovered.

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

D. 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 $(\mathbb{M}, \rho)$ 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 ρ, so data consumers should first check which attributes they owned are in the access matrix by running the ABF query subroutine as follows.

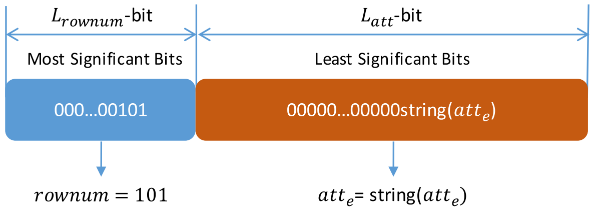


Fig. 6. String abstraction from the element.

1) *ABFQuery*$(S, \text{ABF}, PK) \rightarrow \rho'$*:* It takes as inputs the attribute set S, the ABF and the PK. For each attribute att $\in S$ owned by the data consumer, the algorithm first computes the position indices by feeding the attribute att with the k hash functions $H_1(), \ldots, H_k()$ and gets

$$H_1(\text{att}), H_2(\text{att}), \ldots, H_k(\text{att}).$$

Then, it fetches the corresponding strings from the positions indexed by $H_i(\text{att})$ $(i \in [1, k])$ in the ABF as follows:

$$H_1(\text{att}) \text{ position in ABF } \rightarrow r_{1,e}$$
$$\vdots$$
$$H_k(\text{att}) \text{ position in ABF } \rightarrow r_{k,e}.$$

After that, it reconstructs the element e as

$$\begin{aligned} e &= r_{1,e} \oplus r_{2,e} \oplus \cdots \oplus r_{k-1,e} \oplus r_{k,e} \\ &= r_{1,e} \oplus r_{2,e} \oplus \cdots \oplus r_{k-1,e} \oplus r_{1,e} \\ &\oplus r_{2,e} \oplus \cdots \oplus r_{k-1,e} \oplus e. \end{aligned}$$

Note that the element e is in the format of $e = i||\text{att}_e$ 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 corresponding 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

$$\rho' = \{(\text{rownum}, \text{att})\}_{\text{att} \in S}$$

which shows the corresponding row number in the access matrix $\mathbb{M}$. The ABF query algorithm is shown in Algorithm 2.

When obtaining the access policy $(\mathbb{M}, \rho)$, the data consumer can run the data decryption subroutine as in traditional attribute-based encryption systems.

1) *Dec*$(SK, CT, (\mathbb{M}, \rho')) \rightarrow m$ or $\perp$*:* The data decryption algorithm takes as inputs the secret key SK, the ciphertext CT as well as the access matrix $\mathbb{M}$ and the reconstructed attribute mapping ρ'. If the attributes can

Algorithm 2 ABFQuery

```
Input: An Attribute Bloom Filter $ABF$, a set of attributes $S$
Input: $k$ hash functions $\{H_1(), \cdots, H_k()\}$
Input: Maximum attribute string length $L_{att}$
Input: Maximum row number string length $L_{rownum}$
Output: $\rho' = \{(rownum, att)\}_{att \in S}$
for each $att \in S$ do
    $ReStr = \{0\}^\lambda$    ▷ initialize the reconstructed string
    for $i = 0$ to $k - 1$ do
        $j = H_{i+1}(att)$    ▷ get the index of the position
        $ReStr = ReStr \oplus ABF[j]$
    $att_eStr = LSB_{L_{att}}(ReStr)$
                              ▷ get $L_{att}$ least significant bits
    $att_e = RmLeadingZeroBits(att_eStr)$
                              ▷ remove all the leading zero bits
    if $att_e == att$ then
        $rownumStr = MSB_{L_{rownum}}(ReStr)$
                              ▷ get $L_{rownum}$ most significant bits
        $rownum$ = RmLeadingZeroBits(rownumStr)
                              ▷ remove all the leading zero bits
        Add $(rownum, att)$ into $\rho'$
```

satisfy the access policy, it can leverage the Lagrange Interpolation Formula to find coefficients $\{c_i | i \in I\}$ such that $\sum_{i \in I} c_i \lambda_i = s$, where $I = \{i : \rho'(i) \in S\} \subset \{1, 2, \ldots, l\}$. Then, the data consumer can compute

$$\frac{\hat{e}(C', K)}{\prod_{i \in I} \big(\hat{e}(C_i, L)\hat{e}(C', K_{\rho'(i)})\big)^{c_i}} = \hat{e}(g, g)^{\alpha s}$$

and recover the data as $m = C/\hat{e}(g, g)^{\alpha s}$. Otherwise, it outputs $\perp$ to denote that the decryption fails.

VI. Analysis of Our scheme

A. Security Analysis

Theorem 1: No polynomial time adversary can selectively break our big data access control scheme with an $l^* \times n^*$ ($n^* \leq q$) challenge access matrix, under the decisional q-BDHE assumption.

Proof: Our big data access control scheme is constructed on 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 adversary $\mathcal{A}$ with non-negligible advantage $\epsilon = Adv_{\mathcal{A}}$ in the selective security game (which is the same as the security game defined in Section IV-C), they can build a simulator $\mathcal{B}$ 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 $\mathcal{A}$ with non-negligible advantage $\epsilon = Adv_{\mathcal{A}}$ in the selective security game, we can build a simulator $\mathcal{B}'$ that also solves the decisional q-BDHE problem with non-negligible advantages. The construction of $\mathcal{B}'$ is similar to the simulator $\mathcal{B}$ in [6]. The Init phase in the $\mathcal{B}'$ is the same as the one in the $\mathcal{B}$. In the Setup phase, besides the steps from $\mathcal{B}$, $\mathcal{B}'$ also chooses some random oracles as the BF hash functions. The secret key query phases are also the same, which means that $\mathcal{B}'$.Phase1 = $\mathcal{B}$.Phase1 and $\mathcal{B}'$.Phase2 = $\mathcal{B}$.Phase2. The differences are in the Challenge phase: the encryption algorithm in $\mathcal{B}'$ consists of two subroutines. To simulate the ABF building subroutine, the simulator $\mathcal{B}'$ queries from the ABFBuild oracle. As for the data encryption subroutine, $\mathcal{B}'$.Enc = $\mathcal{B}$.Encrypt. Because the challenge matrix is selected by the adversary before the Init phase, so the constructed ABF is the same no matter which plaintext is selected for encryption, which means that the ABF will not increase the advantages of the adversary $\mathcal{A}$ in the security game. Similar to the proof in [6], we can show that $\mathcal{B}'$ 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 λ.

Proof: In our scheme, only the data consumers who hold the attributes can obtain the string of attribute from the attribute space $\mathcal{U}$. 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 $\mathbb{M}$ and the ABF.

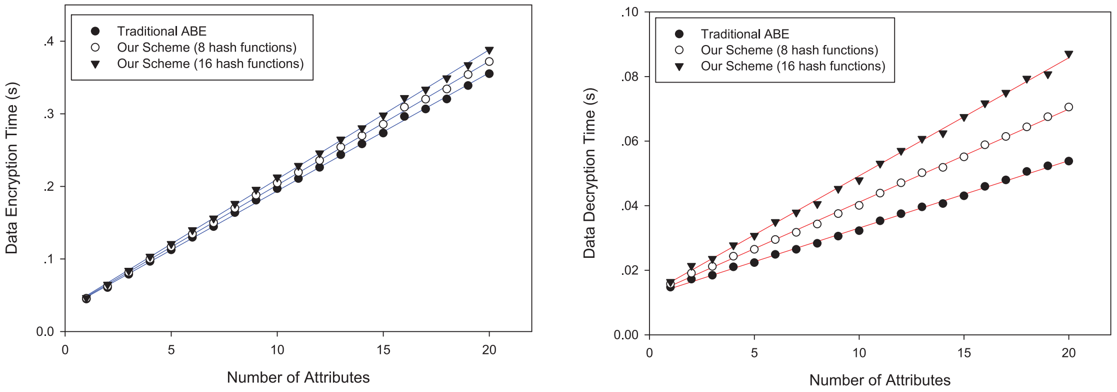
Data consumers are only allowed to check whether their owned attributes are in the access policy. Unless the data consumer has all the attributes of the attribute space or several data consumers collude together, they cannot check all the attributes from the attribute space in the system. Since the ABF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

B. 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 α-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

[2] [Online]. Available: https://sites.google.com/site/murmurhash/



(a) (b)

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

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.

VII. Conclusion

In this paper, we have proposed an efficient and fine-grained data access control scheme for big data, where the access policy 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 authority 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 *Applied Cryptography 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 disjunctions, polynomial equations, and inner products,” in *Advances in Cryptology–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,” *IEEE Wireless 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 encryption of mobile cloud networks: When QoE meets QoP,” *IEEE Wireless Commun.*, vol. 22, no. 4, pp. 74–80, Aug. 2015.

[17] H. Li *et al.*, “Enabling fine-grained multi-keyword search supporting 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.

[20] J. Lai, R. H. Deng, and Y. Li, "Expressive CP-ABE with partially hidden access structures," in *Proc. ASIACCS*, Seoul, South Korea, 2012, pp. 18–19.

[21] J. Hur, "Attribute-based secure data sharing with hidden policies in smart grid," *IEEE Trans. Parallel Distrib. Syst.*, vol. 24, no. 11, pp. 2171–2180, Nov. 2013.

[22] A. Beimel, "Secure schemes for secret sharing and key distribution," Ph.D. dissertation, Israel Inst. Technol. Technion, Haifa, Israel, 1996.

[23] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Commun. ACM*, vol. 13, no. 7, pp. 422–426, 1970.

[24] K. Yang, X. Jia, and K. Ren, "Secure and verifiable policy update outsourcing for big data access control in the cloud," *IEEE Trans. Parallel Distrib. Syst.*, vol. 26, no. 12, pp. 3461–3470, Dec. 2015.

[25] C. Dong, L. Chen, and Z. Wen, "When private set intersection meets big data: An efficient and scalable protocol," in *Proc. CCS*, Berlin, Germany, 2013, pp. 789–800.

Kan Yang received the B.Eng. degree in information security from the University of Science and Technology of China, Hefei, China, in 2008, and the Ph.D. degree in computer science from the City University of Hong Kong, Hong Kong, in 2013.

He is currently a Post-Doctoral Fellow of Broadband Communications Research Group with the Department of Electrical and Computer Engineering, University of Waterloo, Waterloo, ON, Canada. His current research interests include cloud security, big data security, mobile security, applied cryptography, and distributed systems.

Qi Han received the B.Sc. degree from Xidian University, Xi'an, China, in 2012, where he is currently pursuing the Ph.D. degree in cryptography.

He is a visiting student with the Department of Electrical and Computer Engineering, University of Waterloo, Waterloo, ON, Canada, in 2014. His current research interests include cryptography, privacy protection of healthcare, and security of cloud.

Hui Li (A'10–M'12) received the B.Sc. degree from Fudan University, Shanghai, China, in 1990, and the M.A.Sc. and Ph.D. degrees from Xidian University, Xi'an, China, in 1993 and 1998, respectively.

He was a Visiting Scholar with the Department of Electrical and Computer Engineering, University of Waterloo, Waterloo, ON, Canada, in 2009. Since 2005, he has been a Professor with the School of Telecommunications Engineering, Xidian University. He has coauthored two books. His current research interests include cryptography, security of cloud computing, wireless network security, information theory, and network coding.

Prof. Li served as a TPC Co-Chair of ISPEC 2009 and IAS 2009, and a General Co-Chair of e-forensic 2010, ProvSec 2011, and ISC 2011.

Kan Zheng (S'03–M'05–SM'09) received the B.S., M.S., and Ph.D. degrees from the Beijing University of Posts and Telecommunications (BUPT), Beijing, China, in 1996, 2000, and 2005, respectively.

He is currently a Full Professor with the BUPT. He has rich experiences in the research and standardization of new emerging technologies. He has authored over 200 journal articles and conference papers in the field of wireless networks and the Internet of Things.

Prof. Zheng holds Editorial Board positions for several journals. He has also served on the Organizing/TPC Committees for over ten conferences such as the IEEE PIMRC 2015 and the IEEE SmartGrid 2015. He is currently the Chair of the IEEE Computer Society STC Internet-of-Everything.

Zhou Su (S'02–M'03) received the B.E. and M.E. degrees from Xi'an Jiaotong University, Xi'an, China, in 1997 and 2000, respectively, and the Ph.D. degree from Waseda University, Tokyo, Japan, in 2003.

He was an exchange student between Waseda University and Xi'an Jiaotong University from 1999 to 2000. His current research interests include multimedia communication, Web performance, and network traffic.

Prof. Su was a recipient of the Best Paper Award of International Conference CHINACOM'08 and the Funai Information Technology Award for Young Researchers 2009. He is the Chair of interest group of IEEE Comsoc Society, Multimedia Communications Technical Committee, and MENIG. He also served as the Co-Chair of several international conferences including the IEEE CCNC'11-WIP track, WICON'11-Network track, and IWCMC'12-Security track. He has been the TPC Member of some flagship conferences such as the IEEE INFOCOM, the IEEE ICC, and the IEEE GLOEBECOM. He is a Member of the IEICE.

Xuemin (Sherman) Shen (M'97–SM'02–F'09) is a Professor and the University Research Chair with the Department of Electrical and Computer Engineering, University of Waterloo, Waterloo, ON, Canada. His current research interests include resource management in interconnected wireless/wired networks, wireless network security, social networks, smart grid, and vehicular *ad hoc* and sensor networks.

Prof. Shen served as the Technical Program Committee Chair/Co-Chair for the IEEE GLOBECOM'16, the IEEE INFOCOM'14, the IEEE VTC'10 Fall, the Symposia Chair for the IEEE ICC'10, the Tutorial Chair for the IEEE VTC'11 Spring and the IEEE ICC'08, and the Technical Program Committee Chair for the IEEE GLOBECOM'07. He also serves/served as an Editor-in-Chief for *IEEE Network, Peer-to-Peer Networking and Application*, and *IET Communications*. He is a Registered Professional Engineer of Ontario, Canada, an Engineering Institute of Canada Fellow, a Canadian Academy of Engineering Fellow, a Royal Society of Canada Fellow, and a Distinguished Lecturer of the IEEE Vehicular Technology and Communications Societies.