Computer Networks 153 (2019) 1–10

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

Fine-grained data access control with attribute-hiding policy for cloud-based IoT

Jialu Hao[a,*], Cheng Huang[b], Jianbing Ni[b], Hong Rong[a], Ming Xian[a], Xuemin (Sherman) Shen[b]

[a] College of Electronic Science and Technology, National University of Defense Technology, Changsha, 410073, PR China
[b] Department of Electrical and Computer Engineering, University of Waterloo, Waterloo, ON N2L 3G1, Canada

ARTICLE INFO

Article history:
Received 30 June 2018
Revised 6 December 2018
Accepted 14 February 2019
Available online 15 February 2019

Keywords:
Access control
Attribute-based encryption
Attribute hiding
Policy privacy
Cloud computing
Internet of Things

ABSTRACT

Ciphertext-policy attribute-based encryption (CP-ABE) is a promising approach to achieve fine-grained access control over the outsourced data in Internet of Things (IoT). However, in the existing CP-ABE schemes, the access policy is either appended to the ciphertext explicitly or only partially hidden against public visibility, which results in privacy leakage of the underlying ciphertext and potential recipients. In this paper, we propose a fine-grained data access control scheme supporting expressive access policy with fully attribute hidden for cloud-based IoT. Specifically, the attribute information is fully hidden in access policy by using randomizable technique, and a fuzzy attribute positioning mechanism based on garbled Bloom filter is developed to help the authorized recipients locate their attributes efficiently and decrypt the ciphertext successfully. Security analysis and performance evaluation demonstrate that the proposed scheme achieves effective policy privacy preservation with low storage and computation overhead. As a result, no valuable attribute information in the access policy will be disclosed to the unauthorized recipients.

1. Introduction

Internet of Things (IoT) is the network of physical "things", such as smart phones, sensors, and wearable devices, that enables these things to connect and exchange data, creating opportunities to improve our daily lives in different domains including electronic healthcare, smart home and transportation [1–3]. Almost 50 billion IoT devices will be connected together by 2020 [4], and they will continuously produce large amounts of data which should be stored and processed in a well-organized way. In such situation, the traditional local data management is not scalable for large data volume [5]. The cloud computing technology, which provides plentiful storage and computation resources, has been widely used to maintain and manage these IoT-driven data [6,7].

However, the data owners lose the physical control over their data after outsourcing them to the cloud [8]. The frequent data leakage incidents [9–11] undermine trust in the cloud service provider, which make data privacy and security be a serious concern for data owners [12,13]. Although traditional encryption technology can be used to protect data confidentiality, it is relatively inefficient to serve the needs of flexible data sharing. Thus, the novel attribute-based encryption (ABE) [14] is applied to achieve fine-grained access control and preserve data confidentiality simultaneously. Especially, ciphertext-policy attribute-based encryption (CP-ABE) [15] enables the data owners to encrypt their data under specified access policy over a set of attributes, and the data recipients are allowed to decrypt the ciphertext only if their attributes satisfy the access policy associated with the ciphertext. However, in the conventional CP-ABE schemes [15,16], the access policy is explicitly appended to the data ciphertext, thus anyone who obtains the ciphertext, including the cloud service provider, may be able to infer some secret information about the data content or the privileged data recipients from the policy. For example, to share the medical records with the doctors or nurses in the *Cardiology Department* of *Hospital A or B*, a patient encrypts them under the access policy {[**Occupation**: ("*Doctor*" OR "*Nurse*")] AND [**Department**: ("*Cardiology*")] AND [**Hospital**: ("*A*" OR "*B*")]}, and uploads the encrypted data including the policy explicitly to the cloud server. In such situation, anyone who obtains the ciphertext infers that the data owner may suffer from a heart problem, although they do not obtain the plaintext. Even worse,

* Corresponding author.
E-mail addresses: jialu.hao@uwaterloo.ca, j35hao@uwaterloo.ca (J. Hao), cheng.huang@uwaterloo.ca (C. Huang), jianbing.ni@uwaterloo.ca (J. Ni), qwertmingx@sina.com (M. Xian), sshen@uwaterloo.ca (Xuemin (Sherman) Shen).

https://doi.org/10.1016/j.comnet.2019.02.008
1389-1286/

the cloud service provider may conclude that the users who request these data regularly are working at the *Cardiology Department* of *Hospital A or B*. Obviously, such information disclosures are not expected by both the data owners and recipients, which makes it necessary to preserve the privacy of access policy in certain applications, just like this sensitive electronic healthcare system.

To solve the privacy leakage problem caused by the public access policy, a direct solution is to hide the attribute information in the policy. However, the simple approach makes the decryption infeasible for the authorized recipients, since they do not know which attributes should be used for decryption. Some proposed schemes [17–22] consider a trade-off between the policy privacy and the feasibility, in which the attribute is split into two parts: name and value. Instead of hiding the whole attribute, only the attribute value is concealed in the access policy. Though these schemes can protect the policy privacy to some extent, the attribute name itself could still reveal some valuable information. On the other hand, inner-product predicate encryption (IPE) [23] can be applied to construct a CP-ABE scheme with fully hidden policy, but the blow up in size caused by the access structure transformation makes it extremely inefficient [21].

Recently, Yang et al. [24] put forward an innovative idea of removing the attribute mapping function ρ from the access policy (M, ρ), which is in the form of linear secret sharing scheme (LSSS). Without sending ρ directly, they utilize a Bloom filter structure [25,26] to help the recipients to locate their attributes to the access matrix M precisely. However, their scheme is not secure against the dictionary attacks, which means anyone can query any attribute from the Bloom filter to confirm whether it is in the access policy, and further recover the whole access policy through multiple trials.

In this paper, we handle the above issue of policy privacy preservation by hiding the whole attribute. Based on the observation in [24], since the attribute mapping function ρ reveals the relationship between the row and attribute in the expressive LSSS-based access policy (M, ρ), removing it can effectively hide the attribute information. However, how to recover the relationship between their attributes and the access matrix M for the authorized recipients, while resisting dictionary attacks, is a challenging problem. In our scheme, we propose a fuzzy attribute positioning mechanism based on garbled Bloom filter to help the recipients query the row numbers for their attributes, in which only authorized recipients are allowed to verify the validity of the results through successful decryption, while for unauthorized recipients no valuable attribute privacy can be compromised. Thus, we can realize fine-grained data access control on the outsourced data, and protect both data confidentiality and policy privacy.

Our contributions are summarized as follows.

1. We propose a fine-grained attribute-based data access control scheme with attribute-hiding policy for cloud-based IoT. Different from existing schemes, our scheme supports expressive access policy and the attribute information is fully hidden.
2. We design a fuzzy attribute positioning mechanism based on garbled Bloom filter to assist the authorized recipients to locate the attributes effectively and decrypt the ciphertext successfully, and prevent the unauthorized recipients deducing any valuable attribute information from the ciphertext.
3. We analyze the security and efficiency of our proposed scheme, and the further simulations demonstrate that the scheme can achieve effective policy privacy preservation with low storage and computation overhead.

The remainder of this paper is organized as follows. We first introduce some related work in Section 2 and review several preliminary concepts in Section 3. The system model, security model and design goals are presented in Section 4, followed by the detailed construction of our proposed scheme in Section 5. Finally, we analyze the security and evaluate the performance in Section 6 and give the conclusion in Section 7.

2. Related work

The notion of attribute-based encryption (ABE) was first introduced by Sahai and Waters [14], which later develops into two forms: ciphertext-policy ABE (CP-ABE) [15] and key-policy ABE (KP-ABE) [27]. Since CP-ABE enables the data owners to specify fine-grained access policy for their data, it soon became popular in the outsourced data access control systems. In a CP-ABE system, the data owners encrypt the data under the access policy on the system attribute universe, and the data recipients request the secret key associated with their attributes from the attribute authority. If and only if the access policy of the ciphertext is satisfied by the recipient's attribute set, can it be decrypted successfully. Generally, according to the expression form, the access policy is divided into three categories: AND-based [17], tree-based [27] and LSSS-based [15]. The AND-based policy is limited in expressiveness and the tree-based policy is a more expressive one that supports the gates of *AND, OR*, and *m of n* threshold. In addition, an LSSS-based access policy is often considered as the most expressive representation, since any monotonic boolean formula can be converted into this type[15].

Currently, many ABE schemes with some new promising functionalities which make them more practical have been proposed, such as revocable ABE [28], lightweight ABE [29], outsourcing ABE [30] and large universe ABE [16]. However, most of the schemes expose the access policy in clear text, which may incur privacy leakage, thus the research on anonymity of ABE is also necessary. In an anonymous ABE, the access policy is hidden such that the unauthorized recipients cannot presume what access policy is formulated by the data owners. The concept of partially hidden access policy was introduced into ABE by Nishide et al. [17] to achieve anonymity, in which the attribute is split into an attribute name and multiple attribute values, and only the attribute values are concealed. Based on the scheme in [17], some works [18–20] improved the construction in terms of efficiency and security, but they are still restricted with the less expressive AND-based access policy. Later, Lai et al. [21] put forward an anonymous CP-ABE scheme in the composite order groups, which partially hides the LSSS-based access policy. With the same form of the access policy, Cui et al. [22,31] proposed a more efficient scheme in the prime order groups on the basis of the large universe construction in [16], where opportunistic decryption tests are required for the authorized recipients. It might also be noted that all the above schemes focus on the partially hidden access policy, but the public attribute names may also lead to the issue of privacy leakage. Some other schemes [23,32] based on the inner-product predicate encryption and hidden vector encryption are proposed to protect the policy privacy, but the efficiency and expressiveness are restricted. Table 1 shows the comparisons of some existing schemes in CP-ABE to preserve the policy privacy.

Recently, Yang et al. [24] proposed a creative scheme to fully hide the attribute information by removing the attribute mapping function ρ from the access policy, but their scheme is vulnerable against the dictionary attacks. In their scheme, anyone is allowed to query any attribute from the attribute Bloom filter to reveal whether it is in the access matrix and further recover the whole access policy through multiple tests. To resist this dictionary attack, we design a fuzzy attribute positioning mechanism, in which only authorized recipients can obtain the attribute information by successful decryption.

Table 1
Comparisons of CP-ABE schemes with policy hidden.

Schemes	Policy hidden	Access policy	Group order	Decryption test
Basic CP-ABE [15]	no	LSSS	prime	N/A
Nishide et al. [17]	yes (disclosed attribute name)	AND gates with multi-values	prime	deterministic[a]
Li et al. [18]	yes (disclosed attribute name)	AND gates with multi-values	prime	deterministic
Lai et al. [19]	yes (disclosed attribute name)	AND gates with multi-values	composite	deterministic
Zhang et al. [20]	yes (disclosed attribute name)	AND gates with multi-values	prime	deterministic
Lai et al. [21]	yes (disclosed attribute name)	LSSS with multi-values	composite	opportunistic[b]
Cui et al. [22]	yes (disclosed attribute name)	LSSS with multi-values	prime	opportunistic
Michalevsky et al. [23]	yes (whole policy hidden)	Inner product predicates	prime	opportunistic
Khan et al. [32]	yes (whole policy hidden)	LSSS with hidden vectors	prime	opportunistic
Yang et al. [24]	no	LSSS	prime	N/A
Ours	yes (whole attribute hidden)	LSSS	prime	opportunistic

[a] "deterministic" means that the number of decryption test is fixed, usually is one.
[b] "opportunistic" means that multiple tests may be required before finding the attributes for successful decryption.

3. Preliminaries

In this section, we review some technical preliminaries related to our work.

3.1. Access structure

Definition 1 (Access Structure [16])**.** An access structure on an attribute universe U is a collection $\mathbb{A}$ of non-empty sets of attributes. The sets in $\mathbb{A}$ are called the authorized sets. In addition, an access structure which satisfies the following requirement is called monotone: if $B \in \mathbb{A}$ and B⊆C, then $C \in \mathbb{A}$.

In the CP-ABE scheme, only the user who has an authorized attribute set is allowed to decrypt the ciphertext. In this paper, we only consider the monotone access structure, and the concept of access structure is also referred to as access policy in our context.

3.2. Linear secret sharing scheme

We apply the linear secret sharing scheme to represent the access policy in our scheme.

Definition 2 (LSSS [16])**.** Let p be a prime. A linear secret sharing scheme $\prod$ with a secret in Z_p according to the access policy over an attribute universe U is called linear if:

1. The shares of a secret $s \in Z_p$ assigned to each attribute constitute a vector over Z_p.
2. For an access policy over U, there exist an $l \times n$ share-generating matrix, and an attribute mapping function ρ labeling each row in M with an attribute in U, which satisfy that:
   With a column vector $\vec{z} = (s, z_2, z_3, \ldots, z_n)$, where $z_2, z_3, \ldots, z_n$ are random values in Z_p, $M\vec{z}$ is the vector formed by the l shares of the secret s based on $\prod$, and $(M\vec{z})_j$ is the share assigned to the attribute $\rho(j)$. The pair of (M, ρ) is referred to as access policy.

The linear secret sharing scheme satisfies reconstruction and security requirements. Specifically, if S is an authorized set for the policy (M, ρ), there exist constants $\{\omega_i \in Z_p\}_{i \in I}$ such that $\sum_{i \in I}(\omega_i M_i) = (1, 0, \ldots, 0)$, where I denotes the set of rows for which the corresponding attributes belong to S, i.e. $I = \{i | \rho(i) \in S \cap i \in [l]\}$[1] Obviously, the secret s can be recovered through $\sum_{i \in I}(\omega_i \lambda_i) = s$. However, no such constant exists for any unauthorized set.

[1] For simplicity, in our context we define $[n] \stackrel{def.}{=} \{1, 2, \ldots, n\}$ for $n \in \mathbb{N}$.

3.3. Bloom filter

Bloom filter [25] is a space-efficient data structure for probabilistic set membership querying. A Bloom filter includes an m-bit array to encode a set A including at most n elements, and a set of independent hash functions H, where each $h_i \in H$ maps an element to a position index in $[m]$ uniformly. In general, $(m, n, k, H) - BF$ is used to represent a Bloom filter with parameters (m, n, k, H), BF_A denotes a Bloom filter encoding the set A, and $BF_A[i]$ denotes the value in the ith position of BF_A.

At first, each bit in the array is 0. To add an element $x \in A$ to the filter, x is hashed by k hash functions respectively to generate k position indexes. Then, for each $i \in [k]$, set $BF_A[h_i(x)] = 1$. To query whether an element y belongs to the set A, y is also hashed by the hash functions, and if there exists $BF_A[h_j(y)] = 0$, then $y \notin A$. Otherwise, $y \in A$ with a high probability. A false positive exists in the Bloom filter, which means it is possible that $y \notin A$ but all $BF_A[h_j(y)]$ equal to 1. Given the size of A, the probability of false positive can be adequately small by selecting m and k optimally.

The garbled Bloom filter is proposed by Dong et al. [26] to deal with the issue of private set intersection. Instead of using an array of bits, an array of η-bit strings is applied in the garbled Bloom filter. To add an element $x \in A$ to the filter, it is split into k shares which will be stored at the positions $\{h_i(x)\}_{i \in [k]}$. To query an element y, if the value recovered from the shares of the k positions $\{h_i(y)\}_{i \in [k]}$ is equal to y, then $y \in A$, otherwise y is not in A.

4. System model and design goals

4.1. System model

Four entities are included in our system, namely data owners, data recipients, attribute authority and cloud server, as shown in Fig. 1.

- **Data owners** To save the local storage and computing cost, the data owners would like to outsource the data generated by the IoT devices to the cloud. Meanwhile, fine-grained access control over the outsourced data is desired by the owners, thus they will use the CP-ABE scheme to encrypt the data before uploading them to the cloud.
- **Data recipients** The data recipients originate data requests to the cloud server and receive the ciphertext. Only authorized recipients possessing attributes that satisfy the access policy of the data, can decrypt the ciphertext successfully. While for unauthorized recipients, they can neither recover the plaintext, nor guess the attributes involved in the access policy.
- **Attribute authority** The attribute authority manages the system attribute universe and distributes the attributes and corresponding private keys to the recipients according to their roles

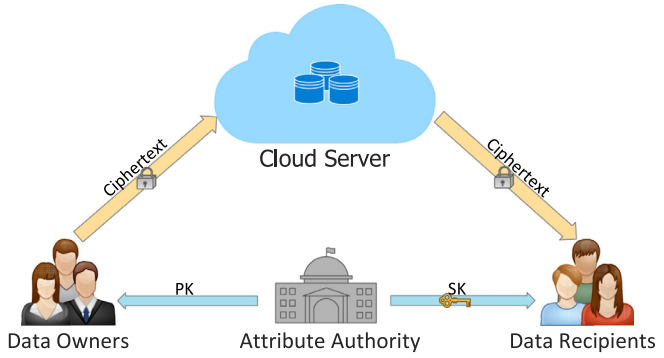


Fig. 1. System model.

or credentials. In addition, the system public key is also generated and published by the attribute authority.

- **Cloud server** The cloud server is considered to have powerful storage and computing resources and is always online to provide services. It helps the data owners store and process their data, responses the requests from the recipients, and distributes the corresponding data to them. Note that, in our system, the data access control is embedded into the decryption, but not implemented by the cloud server.

4.2. Security model

In our system, the attribute authority is regarded as a entirely credible party and the data owners are honest as well. Since the cloud server is in different trust domain with the data owners, it is assumed to be semi-honest, which means it is interested in the data privacy and is not reliable to make the access decisions of the data, but will execute the operations requested by the system users faithfully. The recipients are divided into two kinds: authorized and unauthorized. The authorized recipients are allowed to obtain the data content, and we assume that they will not leak the data information actively. The unauthorized recipients are the potential attackers of the system. They may collude with each other to attempt to decrypt the ciphertext which cannot be accessed individually, also they are interested in the policy privacy of the ciphertext.

Note that, the dictionary attack is considered in our scheme, which means the attribute universe is public, such that the unauthorized recipients, even the cloud server, may conspire to compromise the hidden attribute information of the access policy by testing all the system attributes.

4.3. Design goals

Considering the requirement mentioned in the system and security model, our goal is to design a fine-grained and privacy preserving data access control scheme supporting expressive access policy with fully hidden attributes. Concretely, the following goals should be fulfilled.

- *Fine-grained access control.* The recipients whose attributes satisfy the access policy can decrypt the ciphertext to obtain the data content, while those unauthorized recipients cannot even through colluding.
- *Privacy preservation of expressive policy.* The expressive LSSS-based access policy should be supported. Meanwhile, the unauthorized recipients, include the cloud server, cannot compromise the attribute privacy of the access policy.
- *Practical implementation.* The underlying system operations, such as encryption and decryption, should be completed by the corresponding entities effectively and efficiently.

5. Our proposed scheme

In this section, we first give an overview of the proposed scheme, and then describe the construction in detail of four phases: 1) system setup; 2) key generation; 3) data encryption, and 4) data decryption. Our construction is on the basis of the CP-ABE scheme in [15], and the idea of the attribute-hiding policy can also be used in other ABE schemes with LSSS-based access policy. Table 2 presents some notations used in our scheme.

5.1. Scheme overview

We propose a fine-grained and privacy preserving data access control scheme supporting expressive access policy with fully hidden attributes for cloud-based IoT. In our scheme, we apply the basic CP-ABE primitive to achieve flexible access control, and remove the attribute mapping function ρ from the access policy (M, ρ) to hide the attribute information. To help the authorized recipients locate their attributes to the access matrix, a fuzzy attribute positioning mechanism is designed based on a modified garbled Bloom filter, which is referred to as attribute Bloom filter in our context.

As shown in Table 3, to add an attribute att_x to the original garbled Bloom filter [26], the value att_x itself is inserted. While in [24], a unique value $rownum_x||att_x$ associated with the attribute att_x is inserted, where $rownum_x$ is used to help the recipients to precisely recover the corresponding row number in the access matrix M of attribute att_x. However, since the attribute universe U may be public, anyone including the cloud server can launch the dictionary attack, which means they are able to query any attribute from the filter to make sure whether it is in the access policy, thus the attribute privacy is still revealed. Different from their schemes, to add an attribute to the filter, a unique value binding with the corresponding row number is inserted in our scheme. When the recipients look up the filter, a correct row number can be recovered for those attributes belonging to the policy, but a random row number for others. In addition, only authorized recipients can verify the validity of the row numbers for the attributes through successful decryption, thus the attribute privacy can be preserved effectively.

Generally, the following four algorithms are included in our scheme.

- $Setup(U) \rightarrow (PK, MSK)$ This algorithm takes as input the attribute universe U, and outputs the public key PK and the system master secret key MSK.
- $KeyGen(PK, MSK, S) \rightarrow SK_S$ This algorithm takes as input PK, MSK and a attribute set S, and generates the secret key SK_S associated with S.
- $Encrypt(PK, msg, (M, \rho)) \rightarrow CT$ This algorithm takes as input PK, a message msg and an access policy (M, ρ), and outputs the ciphertext CT, where only M is included in the ciphertext. More specifically, two functions are included in the *Encrypt* algorithm: *CTGen* and *ABFBuild*.
  - *CTGen*: This function encrypts the data under the access policy, which can be seen as the encryption algorithm in the basic CP-ABE scheme.
  - *ABFBuild*: This function constructs an attribute Bloom filter to hide the attribute information from the access policy.
- $Decrypt(CT, SK_S) \rightarrow msg / \perp$ This algorithm takes as input the ciphertext CT and SK_S associated with S, and returns the message msg if the attribute set S satisfies the access policy embedded in CT. Otherwise, it returns $\perp$ with a overwhelming probability. It contains three functions: *ABFQuery, MapRecover* and *DecTest*.
  - *ABFQuery*: This function is used to query a row number from the attribute Bloom filter for each attribute in the recipient's attribute set.

Table 2
Notations used in our scheme.

Notations	Descriptions
PK, MSK	system public key and master secret key
$U = \{att_1, \ldots, att_{\|U\|}\}$	system attribute universe
h_{att_x}	public key components for attribute att_x in U
$S = \{att_j^*\}$	attribute set of the data recipient, $S \subset U$
SK_S	secret key associated with attribute set S
M	an $l \times n$ matrix in the access policy
ρ	an attribute mapping function in the access policy
msg	data file to be uploaded
CT	final ciphertext uploaded to the cloud
(m, n, k, H, η)-*T*	attribute Bloom filter T with parameters (m, n, k, H, η)
Θ	a mapping function from attribute set S to a set of rows $J \subseteq [l]$
M_J	a submatrix of M including the rows belonging to J
$\mathcal{I}$	a set of minimum subsets of J such that for each $I \in \mathcal{I}$ there exists $\sum_{i \in I} w_i M_i = (1, 0, \ldots, 0)$

Table 3
Comparisons of value inserting.

Schemes	Added attribute	Inserted value
Dong et al. [26]	att_x	att_x
Yang et al. [24]	att_x	$rownum_x \| att_x$
Ours	att_x	$\xi l + rownum_x$

- *MapRecover*: This function helps to recover a set of the possible attribute mapping functions.
- *DecTest*: This function is designed to test whether the *Decrypt* algorithm is successful, and returns the final result.

5.2. Scheme description

5.2.1. System setup

The attribute authority first executes the *Setup* algorithm with the input of the attribute universe $U = \{att_1, \ldots, att_{|U|}\}$. Let $\mathbb{G}$ and $\mathbb{G}_T$ be multiplicative cyclic groups of prime order p, g be a generator of $\mathbb{G}$, and $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ be a bilinear map. The attribute authority randomly chooses $\alpha, \beta \in Z_p^*$ and group elements $h_{att_1}, \ldots, h_{att_{|U|}} \in \mathbb{G}$ for all the attributes in U. The public key PK is published as

$$PK = \langle e(g,g)^{\alpha}, g, g^{\beta}, h_{att_1}, \ldots, h_{att_{|U|}} \rangle$$

The attribute authority sets $MSK = g^{\alpha}$ as the system master secret key.

5.2.2. Key generation

When the data recipient joins the system, the authority will assign an attribute set $S \subset U$ to him according to his roles or credentials, and run the *KenGen* algorithm to generate the corresponding secret keys.

- $KeyGen(PK, MSK, S) \to SK_S$

This algorithm takes as input *PK, MSK* and an attribute set *S*. It chooses a random number $t \in Z_p^*$, and computes

$$D = g^{\alpha} g^{\beta t}, \quad D' = g^t, \quad \forall att_x \in S \quad D_{att_x} = h_{att_x}^t$$

Finally, the secret key is distributed to the recipient as

$$SK_S = \langle S, D, D', \{D_{att_x}\}_{att_x \in S} \rangle$$

5.2.3. Data encryption

To achieve data confidentiality and fine-grained access control simultaneously, the data owner first specifies an access policy (M, ρ) over U for the data *msg*. Then, it executes the $Encrypt(PK, msg, (M, \rho))$ algorithm to produce the ciphertext *CT* which will be uploaded to the cloud server.

The *Encrypt* algorithm in our scheme includes two functions: *CTGen* and *ABFBuild*. The function *CTGen* is used to produce the real ciphertext components and the function *ABFBuild* is designed to help the recipients locate their attributes to the access matrix M. Note that the second one is indispensable since the attribute mapping function ρ is removed from the final ciphertext *CT* to prevent the disclosure of attribute privacy in the access policy.

1. $CTGen(PK, msg, (M, \rho)) \to CT_0$

The function takes as input the public key *PK*, the data *msg* and the access policy (M, ρ), where M is an $l \times n$ access matrix and ρ is an injective function that maps each row in M to a unique attribute in U. It first selects random numbers $s, z_2, \ldots, z_n \in Z_p$, and constructs a vector $\vec{z} = (s, z_2, \ldots, z_n)$. It calculates $\lambda_i = M_i \cdot \vec{z}$ for each $i \in [l]$, where M_i means the ith row of M. Here λ_i can be seen as the secret share that is assigned to the attribute $\rho(i)$. Then the ciphertext CT_0 is produced as

$$CT_0 = \langle C = msg \cdot e(g,g)^{\alpha s}, C_0 = g^s, \{C_i = g^{a\lambda_i} h_{\rho(i)}^{-s}\}_{i \in [l]} \rangle$$

Remarks. In order to allow the recipient to test whether the decryption succeeds, we can adopt the technique introduced in [33], in which two independent and uniform δ-bit symmetric keys (key_1, key_2) are generated from a randomly selected value $key \in \mathbb{G}_T$. Then *key* is encrypted by running $CTGen(PK, key, (M, \rho)) = CT_0$. In addition, the data *msg* is encrypted under key_1 with the symmetric encryption $\mathrm{SE}_{key_1}(msg)$. Finally, the ciphertext is in the form $CT' = (CT_0, key_2, \mathrm{SE}_{key_1}(msg))$. After decrypting *key* from CT_0, the recipient first uses key_2 in CT' to verify whether *key* is decrypted successfully, where the false positive probability (approximately $1/2^{\delta}$) can be ignored with a long enough δ. If successful, the recipient can decrypt *msg* from the symmetric ciphertext $\mathrm{SE}_{key_1}(msg)$ through key_1 derived from *key*.

2. $ABFBuild(M, \rho) \to T$

The function first defines the parameters (m, n, k, H, η) of the attribute Bloom filter T, where m means the size of the filter, n is the number of attributes to be added, k means the number of the hash functions in H, and η represents the bit length of the inserted value. In our scheme, n is set as the number of rows in M, which also means the number of attributes in the policy. Then, m and k can be selected optimally according to n, η should be longer than the bit length of l, and $H = \{h_j\}_{j \in [k]}$ are k independent hash functions that hash each attribute to $[m]$ uniformly.

To add an attribute $\rho(i)$ to the filter, a unique value $v_i = \xi_i l + i$ binding with the row number i will be inserted, where ξ_i is a random number and $v_i < 2^{\eta}$. More specifically, v_i is split into k shares $\{r_i^j\}_{j \in [k]}$ with the (k, k) secret sharing scheme based on *XOR* operations [34], and the share r_i^j is put at the position $h_j(\rho(i))$.

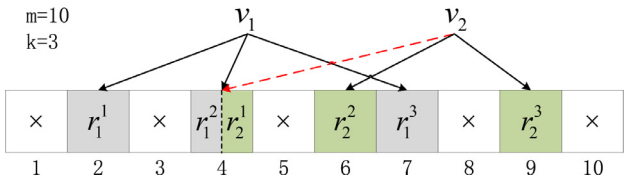


Fig. 2. Example of inserting values into the attribute Bloom filter.

Function 1 ABFBuild.

Input: (M, ρ)
Output: (m, n, k, H, η)-T

1: $n = l$, Select m, k, H, η optimally
2: $T =$ **new** m-element array of η-bit strings
3: **for** $i = 1$ to m **do**
4: $T[i] = null$
5: **end for**
6: **for** $i = 1$ to l **do**
7: Select a random number ξ_i, such that $\xi_i l + i < 2^\eta$
8: $EmptyPos = 0$, $FinalShare = \xi_i l + i$
9: **for** $j = 1$ to k **do**
10: $pos = h_j(\rho(i))$
11: **if** $T[pos] == null$ **then**
12: **if** $EmptyPos == 0$ **then**
13: $EmptyPos = pos$
14: **else**
15: Select a random number v from $\{0,1\}^\eta$
16: $T[pos] = v$
17: $FinalShare = FinalShare \oplus T[pos]$
18: **end if**
19: **else**
20: $FinalShare = FinalShare \oplus T[pos]$
21: **end if**
22: **end for**
23: $T[EmptyPos] = FinalShare$
24: **end for**
25: **for** $i = 1$ to m **do**
26: **if** $T[i] == null$ **then**
27: Select a random number v from $\{0,1\}^\eta$
28: $T[i] = v$
29: **end if**
30: **end for**

The k shares of v_i are computed as follows: it first chooses $k-1$ random number $r_i^1, r_i^2, \ldots, r_i^{k-1}$ with η bits, and computes $r_i^k = r_i^1 \oplus r_i^2 \oplus \cdots \oplus r_i^{k-1} \oplus v_i$.

Note that during the inserting process, some location $pos = h_j(\rho(i))$ could have been occupied by a previous inserted value. In such a case, the existing $T[pos]$ will not be overwritten, which means r_i^j is set as $T[pos]$ and used to compute the final share. For example, as shown in Fig. 2, a value v_1 is inserted into the filter first, and the corresponding positions 2, 4, 7 have be filled with the shares of v_1. Then for the value v_2, position 4 have already be occupied by the share r_1^2 of v_1. So in order to guarantee that the previous inserted value v_1 can be recovered, r_1^2 will be reused, which means that instead of randomly choosing a new share, we set $r_2^1 = r_1^2$. Following Function 1 shows the detailed implement process of *ABFBuild*.

Remarks. Note that the false positive error leads to an empty value of the variable *EmptyPos* after the inner loop (line 9–22), which will cause an illegal index position of the attribute Bloom filter T in line 23. In such a situation, the *ABFBuild* function will be failed and the encryption process will be interrupted, so it will not degrade the security of our scheme. In addition, as proved in [26], the upper bound of the failure rate of the *ABFBuild* function is $p^k(O(\frac{k}{p}\sqrt{\frac{\ln m - k \ln p}{m}}) + 1)$, where $p = 1 - (1 - 1/m)^{(n-1)k}$, which can be ignored with optimally selected m and k. Different from [24], our scheme enables the data owners to select the parameters of the attribute Bloom filter, such that even if the false positive error occurs, the *ABFBuild* function can be efficiently re-executed with new parameters (less than 5ms for 50 attributes). Thus, the effects on the efficiency of the proposed scheme caused by false positive property is limited and acceptable.

After calling the two functions, the data owner uploads the final ciphertext $CT = \langle M, CT', (m, n, k, H, \eta)\text{-}T \rangle$ to the cloud server.

5.2.4. Data decryption

The recipients are allowed to download the ciphertext from the cloud server depending on their interests. When they obtain the ciphertext *CT*, they can run the *Decrypt*(*CT*, SK_S) algorithm to recover the plaintext, if their attribute sets satisfy the access policy. In our scheme, the *Decrypt* algorithm consists of three functions: *ABFQuery, MapRecover* and *DecTest*. The first function is used to query a row number for each attribute in S, the second one is to recover a set of possible attribute mapping functions, and the last one is to test whether the decryption succeeds.

1. *ABFQuery*$(S, T) \rightarrow \Theta$

The function takes as input the attribute set S and the attribute Bloom filter T, where the parameters of T are include implicitly. For each attribute $att_x \in S$, it first computes the k positions $\{h_j(att_x)\}_{j\in[k]}$ and obtains the shares $\{r_x^j = T[h_j(x)]\}_{j\in[k]}$. Then the corresponding row number of the attribute att_x is calculated as $rownum_x = (r_x^1 \oplus r_x^2 \oplus \cdots \oplus r_x^k) \mod l$. As shown in Function 2, a

Function 2 ABFQuery.

Input: S, T
Output: $\Theta : S \rightarrow J$

1: $J = \varnothing$, $\Theta = \varnothing$
2: **for** $att_x \in S$ **do**
3: $temp = 0$
4: **for** $j = 1$ to k **do**
5: $pos = h_j(att_x)$
6: $temp = temp \oplus T[pos]$
7: **end for**
8: $rownum_x = temp \mod l$
9: **if** $rownum_x == 0$ **then**
10: $rownum_x = l$
11: **end if**
12: **if** $rownum_x \notin J$ **then**
13: add $rownum_x$ into J
14: **end if**
15: Add $att_x \rightarrow rownum_x$ into Θ
16: **end for**

mapping function Θ: $S \rightarrow J$ from the attribute set S to a set of rows $J \subseteq [l]$ will be generated after calling the *ABFQuery* function.

Remarks. Note that for those attributes existed in the access policy, the row numbers recovered from T are valuable, but for others are just random numbers in $[l]$. In addition, it is possible that some different attributes may recover the same row number, and the possibility is influenced by the number of attributes in the attribute set and access policy.

After generating Θ, the *Decrypt* algorithm calls the following *MapRecover* function.

2. $MapRecover(\Theta) \rightarrow \mathcal{P}$

This function takes as input Θ, and produces a set of attribute mapping functions $\mathcal{P}$ by choosing only one attribute for each row in J, such that each $\tilde{\rho} \in \mathcal{P}$ is an injective function which maps J to an attribute set $\tilde{S} \subseteq S$. As shown in Function 3, it first generates

Function 3 MapRecover.

Input: $\Theta : S \rightarrow J$
Output: $\mathcal{P}$
1: $Num = 1$
2: **for** each $rownum \in J$ **do**
3: $atts_{rownum} = \Theta^{-1}(rownum)$
4: $Num = Num * \textbf{length}(atts_{rownum})$
5: **end for**
6: **for** $i = 0$ to $(Num - 1)$ **do**
7: $step = Num$, $\tilde{\rho}_i = \varnothing$, $\tilde{S}_i = \varnothing$
8: **for** each $rownum \in J$ **do**
9: $len = \textbf{length}(atts_{rownum})$
10: $step = step/len$
11: $attIndex = (i/step) \mod len$
12: $att = atts_{rownum}[attIndex]$
13: Add att to $\tilde{S}_i$
14: Add $rownum \rightarrow att$ into $\tilde{\rho}_i : J \rightarrow \tilde{S}_i$
15: **end for**
16: Add $\tilde{\rho}_i$ into $\mathcal{P}$
17: **end for**

the attributes associated with each row in J and calculates the total number of the mapping functions in $\mathcal{P}$ (line 1–5). Then for each $\tilde{\rho}_i$, it chooses only one attribute for each row in J to compose an attribute set $\tilde{S}_i$, with the condition that $i \neq j \Rightarrow \tilde{S}_i \neq \tilde{S}_j$ (line 8–15). Finally, it adds all $\tilde{\rho}_i$ to the set $\mathcal{P}$ (line 16).

At last, for each $\tilde{\rho}_i \in \mathcal{P}$ and corresponding secret key $SK_{\tilde{S}_i} \subseteq SK_S$, the *Decrypt* algorithm calls the following *DecTest* function. If any of them outputs msg, then the decryption completes successfully. Otherwise, the *Decrypt* algorithm outputs $\perp$, which represents that the recipient's attributes do not satisfy the access policy.

3. $DecTest(CT', (M_J, \tilde{\rho}), SK_{\tilde{S}}) \rightarrow msg/\perp$

Here M_J denotes the matrix composed of the rows belonging to J. Similarly with [21] and [22], this function first computes a set $\mathcal{I}$ from M_J, where $\mathcal{I}$ denotes the set of minimum subsets of J such that for each $I \in \mathcal{I}$ there exists $\sum_{i \in I} w_i M_i = (1, 0, \ldots, 0)$. Then, for each $I \in \mathcal{I}$, it calculates

$$B = \frac{e(C_0, D)}{\prod_{i \in I}(e(C_0, D_{\tilde{\rho}(i)})e(C_i, D'))^{w_i}}$$

For authorized recipients with the right choice of $\tilde{\rho}$ and I, C_i and $D_{\tilde{\rho}(i)}$ are matched which means they are generated from the same public key component $h_{\tilde{\rho}(i)}$, such that

$$B = \frac{e(g,g)^{\alpha s} e(g,g)^{\beta t s}}{\prod_{i \in I} e(g,g)^{\beta t w_i \lambda_i}} = e(g,g)^{\alpha s}, \quad \frac{C}{B} = key$$

Then, it generates key_1 and key_2 from key. After verifying that key_2 is correct, it can recover msg with the symmetric decryption algorithm under key_1.

Otherwise, B is a random value in $\mathbb{G}_T$, so a random key_2 will be derived, which cannot pass the validation process with a overwhelming probability. Thus the function outputs $\perp$.

Remarks. Since M_J is fixed during the *DecTest* phase, $\mathcal{I}$ only needs to be calculated once. In addition, the paring results can be reused in different tests.

6. Security analysis and performance evaluation

6.1. Security analysis

We analyze the security features of the proposed scheme from the perspectives of data confidentiality and policy privacy.

- **Data confidentiality** The proposed construction is based on the underlying CP-ABE primitive in [15], which has been proved selectively CPA-secure on the basis of the decisional q-BDHE assumption. Following we will demonstrate that the modifications in our scheme do not affect the data confidentiality. It can be seen the *Setup* and *KeyGen* algorithms in our proposed scheme are the same with that in [15]. In addition, the ciphertext generated from the *Encrypt* algorithm in our scheme has the similar structure with [15], except that an attribute Bloom filter T is included. Note that, the attribute Bloom filter is derived only from the access policy (M, ρ), which is public in [15]. Since no more information in our scheme has been disclosed to the adversary compared with [15], the advantages of the adversary to break the data confidentiality in our scheme is no more than that in [15]. Thus, based on the same security assumption with [15], we can conclude that our scheme is able to guarantee the data confidentiality.
- **Policy privacy** In our scheme, the attribute mapping function ρ is removed from the access policy to prevent the leakage of attribute information. Additionally, a fuzzy attribute positioning algorithm is designed to help the authorized recipients to decrypt the ciphertext. Following we will show that the adversary (unauthorized recipients) cannot recover valuable attribute information from the fuzzy attribute positioning algorithm. In the fuzzy attribute positioning algorithm, the row number corresponding to each attribute in the access policy is inserted into the attribute Bloom filter through the *ABFBuild* function. By calling the *ABFQuery* function, the adversary is allowed to query a row number for every attribute in his attribute set. Furthermore, considering the dictionary attack, the adversary can check all the system attributes. However, correct row numbers can be recovered only for those attributes belonging to the access policy, while for others, a random row number is returned. Note that, the validity of the row number for an attribute can only be verified through successful decryption. Since the adversary cannot break the data confidentially, even through colluding, nor can it identify the attribute mapping relationship. Thus, our scheme can protect the policy privacy through hiding the attribute information in the access policy.

6.2. Performance evaluation

We first give a comparison of our scheme and some other CP-ABE schemes with LSSS-based access policy in the literature [15,21,22,24], with respect to storage overhead and computation cost. Some related notions are clarified as follows.

- $|\mathbb{G}|, |\mathbb{G}_T|, |Z_p|$: The bit-length of element in $\mathbb{G}$, $\mathbb{G}_T$ and Z_p, respectively.
- $|U|$, $|S|$, l: The number of attributes in the system attribute universe, recipient attribute set and access policy, respectively.
- $|M|$, $|(M, \rho)|$, $|h|$, m: The bit-length of access matrix, access policy and hash function, respectively.
- L_l, L_{att}: The bit-length of the value l and attribute string, respectively.
- k: The number of hash functions for the attribute Bloom filter.
- m: The size of the attribute Bloom filter.
- $X_{M,1}$: The number of elements in $\mathcal{I}_M = \{I_1, \ldots, I_{X_{M,1}}\}$, where $\mathcal{I}_M$ means the set of minimum subsets satisfying the access matrix M.

Table 4
Comparisons of storage overhead.

Schemes	Public key	Master secret key	Recipient secret key	Ciphertext
Basic CP-ABE [15]	$(2+\|U\|)\|\mathbb{G}\|+\|\mathbb{G}_T\|$	$\|\mathbb{G}\|$	$(2+\|S\|)\|\mathbb{G}\|$	$\|\mathbb{G}_T\|+(1+l)\|\mathbb{G}\|+\|(M,\rho)\|$
Lai et al. [21]	$(4+\|U\|)\|\mathbb{G}\|+\|\mathbb{G}_T\|$	$\|U\|\|\mathbb{G}\|+\|\mathbb{Z}_p\|$	$(2+\|U\|)\|\mathbb{G}\|$	$2\|\mathbb{G}_T\|+4l\|\mathbb{G}\|+\|(M,\rho)\|$
Cui et al. [22]	$9\|\mathbb{G}\|+\|\mathbb{G}_T\|+\|h\|$	$\|\mathbb{G}\|+4\|\mathbb{Z}_p\|$	$(2+5\|S\|)\|\mathbb{G}\|$	$(3+6l)\|\mathbb{G}\|+\|(M,\rho)\|$
Yang et al. [24]	$(2+\|U\|)\|\mathbb{G}\|+\|\mathbb{G}_T\|+k\|h\|$	$\|\mathbb{G}\|$	$(2+\|S\|)\|\mathbb{G}\|$	$\|\mathbb{G}_T\|+(l+1)\|\mathbb{G}\|+\|M\|+m(L_l+L_{att})$
Ours	$(2+\|U\|)\|\mathbb{G}\|+\|\mathbb{G}_T\|$	$\|\mathbb{G}\|$	$(2+\|S\|)\|\mathbb{G}\|$	$\|\mathbb{G}_T\|+(1+l)\|\mathbb{G}\|+\|M\|+mL_l+k\|h\|$

Table 5
Comparisons of computation cost.

Schemes	Operations							
	Encryption				Decryption			
	Multi	Expo	Pairing	Hash	Multi	Expo	Pairing	Hash
Basic CP-ABE [15]	$l+1$	$2l+2$	0	0	$\|I\|+2$	$\|I\|$	$2\|I\|+1$	0
Lai et al. [21]	$6l+2$	$8l+4$	0	0	$\le X_{M,2}+X_{M,1}+\|I\|+2$	$\le \|I\|+X_{M,2}$	$\le 2(1+\|I\|+\|S\|)$	0
Cui et al. [22]	$2l+1$	$8l+4$	0	1	$\le 5X_{M,2}+2X_{M,1}$	$\le X_{M,2}+2X_{M,1}$	$\le 6\|S\|+1$	$\le X_{M,1}$
Yang et al. [24]	$l+1$	$2l+2$	0	k	$\|I\|+2$	$\|I\|$	$2\|I\|+1$	k
Ours	$l+1$	$2l+2$	0	k	$\le \|S\|\|J\|+2\|\mathcal{P}\|X_{M_J,1}$	$\le \|\mathcal{P}\|X_{M_J,2}$	$\le \|S\|+\|J\|+1$	k

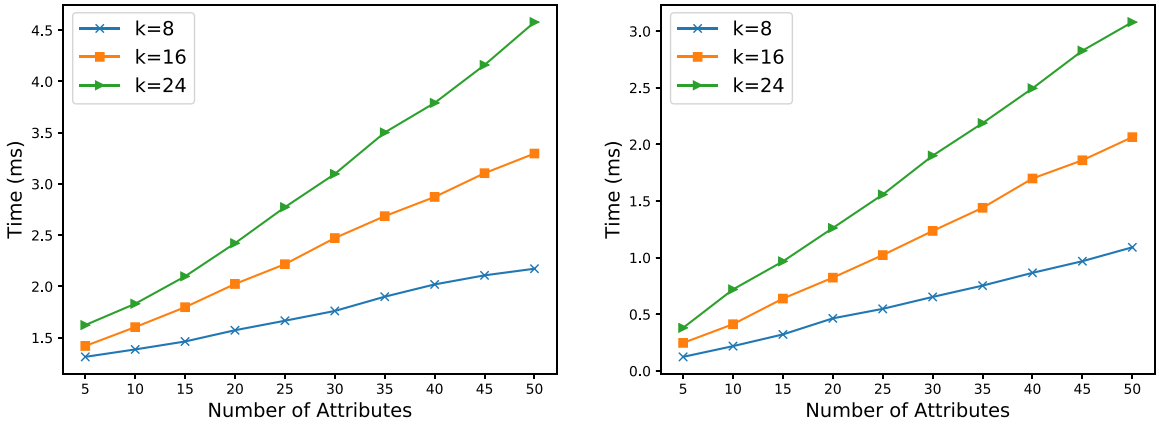


Fig. 3. Computation time for *ABFBuild* and *ABFQuery* functions.

- $X_{M,2}$: The total number of attributes in all the subsets of $\mathcal{I}_M$, i.e., $|I_1|+\cdots+|I_{X_{M,1}}|$.
- $|\mathcal{P}|$: The number of elements in the set of attribute mapping functions $\mathcal{P}$.
- $|J|$: The number of rows used in the decryption.
- $|I|$: The number of attributes used during the final successful decryption.

Table 4 presents the sizes of the public key, master secret key, recipient secret key and ciphertext (i.e., storage overhead). It demonstrates that our scheme achieves attribute hiding only with few ciphertext storage overhead caused by the attribute Bloom filter compared with the underlying CP-ABE scheme [15]. Due to the small size of the value inserted into the Bloom filter, our scheme has a better performance than [24]. Compared with [21] and [22] which apply the LSSS-based access policy with multi-valued attributes, the size of the recipient secret key in our scheme is much smaller. In addition, since the bit-length of the group element is much longer than that of l, our scheme can also save some ciphertext storage space.

Table 5 shows the computing operations involved in the encryption and decryption processes, in which only some time-consuming operations are considered, such as pairing, hashing and multiplication and exponentiation on groups. Considering the encryption process, our scheme only has some additional hashing operations compared with the underlying CP-ABE scheme. Our scheme also has a better performance compared with [21] and [22] in terms of the multiplication and exponentiation operations. With regard to the decryption process, since opportunistic decryption test is required in [21], [22] and our scheme, we give the results in a worst-case scenario. For [21] and [22], a set of minimum subsets of attributes $\mathcal{I}_M$ from M needs to be calculated before the decryption test. While in our scheme, we compute $\mathcal{I}_{M_J}$ from M_J, thus $X_{M_J,1} \le X_{M,1}$ and $X_{M_J,2} \le X_{M,2}$. In addition, for the most time-consuming pairing operation, our scheme has a significantly better performance. For the multiplication and exponentiation operations, although the cost in our scheme is affected by $|\mathcal{P}|$, the smaller $X_{M_J,1}$ and $X_{M_J,2}$ make them be completed efficiently.

Generally, taking into a comprehensive consideration of storage overhead, computation cost and policy privacy, our scheme can achieve more effective privacy preservation with a better overall performance.

We simulate our scheme with python 3.5 on a notebook with an Intel Core i7-7600U CPU at 2.80GHz and 16GB RAM running Ubuntu 18.04. Charm framework (v0.5) is applied to implement the cryptographic operations from the supporting of the PBC library (v0.5.14) and the OpenSSL library (v1.0.2). We use the double hashing technology [35] based on the 128-bit MurmurHash and SpookyHash to construct the k hash functions of the attribute

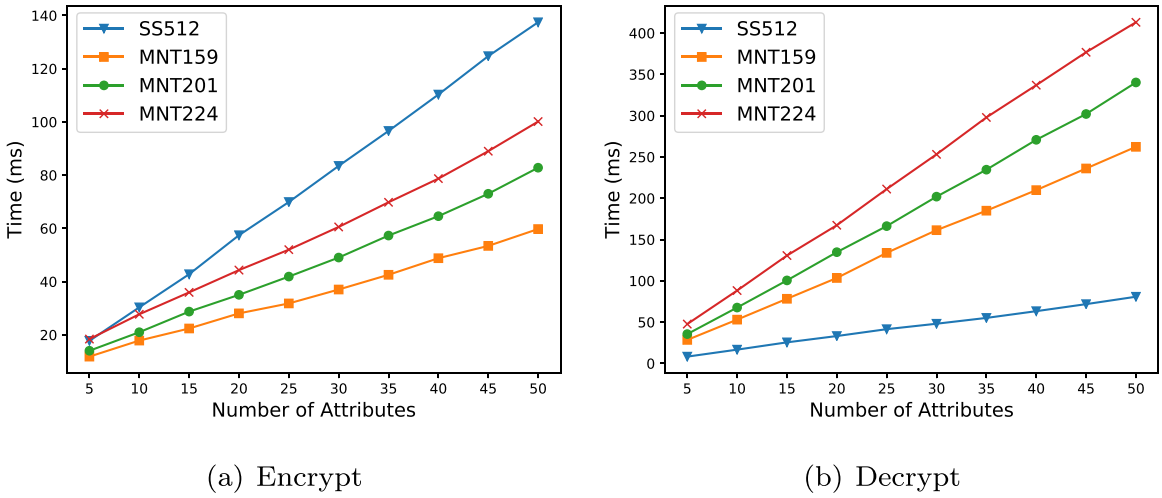


Fig. 4. Computation time for *Encrypt* and *Decrypt* algorithms.

Bloom filter. The numbers of attributes in the access policy and recipient attribute set are both from 5 to 50. All the results are average running time in milliseconds of 50 trials.

Fig. 3 shows the running time of the *ABFBuild* and *ABFQuery* functions with $k = 8$, 16, 24, where $m = 1024$, $\eta = 8$, and n is the number of attributes. In Fig. 3(a), 50 attributes can be inserted into the filter in less than $5ms$, thus in a worst case that the *ABFBuild* function fails, it can be completed with the new parameters soon. In addition, it can be calculated that the *ABFBuild* function has a negligible failure rate of 10^{-5} with $k = 16$, so we adopt $k = 16$ in the subsequent simulations.

We simulate the encryption and decryption algorithms in our scheme on the basis of four elliptic curves: SS512, MNT159, MNT201 and MNT224[2], which provide different security levels. Fig. 4 shows that the execution time of encryption and decryption increases linearly with the number of attributes. The running time for building and querying the attribute Bloom filter is only a tiny fraction of the total time for the encryption and decryption processes. In addition, in our experiment, the decryption of a ciphertext containing 50 attributes can be completed in less than $400ms$ with a secret key of 50 attributes, and the results are acceptable even with multiple decryption tests in practical applications.

7. Conclusion

In this paper, we have proposed a fine-grained data access control scheme supporting expressive access policy with fully attribute hidden for cloud-based IoT. We have designed a fuzzy attribute positioning mechanism based on garbled Bloom filter such that the authorized recipients are able to locate their attributes to the access matrix and decrypt the ciphertext efficiently, while for unauthorized recipients no valuable attribute information can be presumed. Our scheme can achieve both data confidentiality and policy privacy preservation on the basis of the underlying CP-ABE scheme. Numerical analysis and simulation results demonstrate that our scheme can achieve effective policy privacy preservation with low storage and computation overhead. In the future work, we will focus on how to decrease the number of decryption tests for the authorized recipients with more redundant attributes which are not in the ciphertext access policy.

[2] "SS" means the super singular curves (symmetric) and "MNT" is Miyaji, Nakabayashi, Takano curves (asymmetric). The number after the type of the curve means the bit size of the base field [16].

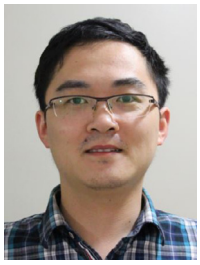
References

[1] J. Ni, K. Zhang, X. Lin, X. Shen, Securing fog computing for Internet of Things applications: challenges and solutions, IEEE Commun. Surv. Tutor. 20 (1) (2018) 601–628, doi:10.1109/COMST.2017.2762345.
[2] Internet of Things security: a top-down survey, Comput. Netw. 141 (2018) 199–221, doi:10.1016/j.comnet.2018.03.012.
[3] K. Zhang, J. Ni, K. Yang, X. Liang, J. Ren, X. Shen, Security and privacy in smart city applications: challenges and solutions, IEEE Commun. Mag. 55 (1) (2017), doi:10.1109/MCOM.2017.1600267CM.
[4] Popular internet of things forecast of 50 billion devices by 2020 is outdated, (Available at: https://spectrum.ieee.org/tech-talk/telecom/internet/popular-internet-of-things-forecast-of-50-billion-devices-by-2020-is-outdated). Accessed June 1, 2018.
[5] A. Ouaddah, H. Mousannif, A.A. Elkalam, A.A. Ouahman, Access control in the Internet of Things: big challenges and new opportunities, Comput. Netw. 112 (2017) 237–262, doi:10.1016/j.comnet.2016.11.007.
[6] X. Liu, R. Deng, K.-K.R. Choo, Y. Yang, H. Pang, Privacy-preserving outsourced calculation toolkit in the cloud, IEEE T. Depend. Secure (2018), doi:10.1109/TDSC.2018.2816656.
[7] K. Yang, Z. Liu, X. Jia, X. Shen, Time-domain attribute-based access control for cloud-based video content sharing: a cryptographic approach, IEEE T. Multimedia 18 (5) (2016), doi:10.1109/TMM.2016.2535728.
[8] J. Shu, X. Liu, X. Jia, K. Yang, R. Deng, Anonymous privacy-preserving task matching in crowdsourcing, IEEE Internet Things J. 5 (4) (2018) 3068–3078, doi:10.1109/JIOT.2018.2830784.
[9] 47gb of Medical Records and Test Results Found in Unsecured Amazon s3 Bucket, 2018a, (Available at: https://www.hipaajournal.com/47gb-medical-records-unsecured-amazon-s3-bucket). Accessed June 1.
[10] 7 Most Infamous Cloud Security Breaches, 2018b(Available at: https://blog.storagecraft.com/7-infamous-cloud-security-breaches). Accessed June 1.
[11] Top 5 largest data leaks of 2017 - so far, 2018c(Available at: https://www.kaspersky.com/blog/data-leaks-2017/19723), Accessed June 1.
[12] 8 public Cloud Security Threats to Enterprises in 2018, a, (Available at: https://www.comparethecloud.net/articles/8-public-cloud-security-threats-to-enterprises-in-2017) Accessed June 1, 2018a.
[13] The Dirty Dozen: 12 Top Cloud Security Threats for 2018, 2018b, (Available at: https://www.csoonline.com/article/3043030/security/12-top-cloud-security-threats-for-2018.html). Accessed June 1.
[14] A. Sahai, B. Waters, Fuzzy identity-based encryption, in: Proc. of EUROCRYPT'05, 2005, pp. 457–473, doi:10.1007/11426639_27.
[15] B. Waters, Ciphertext-policy attribute-based encryption: an expressive, efficient, and provably secure realization, in: Proc. of PKC'11, 2011, pp. 53–70, doi:10.1007/978-3-642-19379-8_4.
[16] Y. Rouselakis, B. Waters, Practical constructions and new proof methods for large universe attribute-based encryption, in: Proc. of CCS'13, 2013, pp. 463–474, doi:10.1145/2508859.2516672.
[17] T. Nishide, K. Yoneyama, K. Ohta, Attribute-based encryption with partially hidden encryptor-specified access structures, in: Proc. of ACNS'08, 2008, pp. 111–129, doi:10.1007/978-3-540-68914-0_7.
[18] J. Li, K. Ren, B. Zhu, Z. Wan, Privacy-aware attribute-based encryption with user accountability, in: Proc. of ISC'09, 2009, pp. 347–362, doi:10.1007/978-3-642-04474-8_28.
[19] J. Lai, R.H. Deng, Y. Li, Fully secure cipertext-policy hiding CP-ABE, in: Proc. of ISPEC'11, 2011, pp. 24–39, doi:10.1007/978-3-642-21031-0_3.
[20] Ensuring attribute privacy protection and fast decryption for outsourced data security in mobile cloud computing, Inf. Sci. 379 (2017) 42–61, doi:10.1016/j.ins.2016.04.015.

[21] J. Lai, R.H. Deng, Y. Li, Expressive CP-ABE with partially hidden access structures, in: Proc. of ASIACCS'12, 2012, pp. 18–19, doi:10.1145/2414456.2414465.
[22] H. Cui, R.H. Deng, G. Wu, J. Lai, An efficient and expressive ciphertext-policy attribute-based encryption scheme with partially hidden access structures, in: Proc. of ProvSec'16, 2016, pp. 19–38, doi:10.1007/978-3-319-47422-9_2.
[23] Y. Michalevsky, M. Joye, in: Decentralized Policy-Hiding ABE with Receiver Privacy, Springer, 2018, pp. 548–567, doi:10.1007/978-3-319-98989-1_27.
[24] K. Yang, Q. Han, H. Li, K. Zheng, Z. Su, X. Shen, An efficient and fine-grained Big Data access control scheme with privacy-preserving policy, IEEE Internet Things J. 4 (2) (2017) 563–571, doi:10.1109/JIOT.2016.2571718.
[25] B.H. Bloom, Space/time trade-offs in hash coding with allowable errors, Commun. ACM 13 (7) (1970) 422–426, doi:10.1145/362686.362692.
[26] C. Dong, L. Chen, Z. Wen, When private set intersection meets big data: an efficient and scalable protocol, in: Proc. of CCS'13, 2013, pp. 789–800, doi:10.1145/2508859.2516701.
[27] V. Goyal, O. Pandey, A. Sahai, B. Waters, Attribute-based encryption for fine-grained access control of encrypted data, in: Proc. of CCS'06, 2006, pp. 89–98, doi:10.1145/1180405.1180418.
[28] J. Hao, C. Huang, J. Liu, M. Xian, X. Shen, Efficient outsourced data access control with user revocation for cloud-based IoT, in: Proc. of GlobeCom'18, 2018, pp. 1–6, doi:10.3233/MGS-180297.
[29] Q. He, N. Zhang, Y. Wei, Y. Zhang, Lightweight attribute based encryption scheme for mobile cloud assisted cyber-physical systems, Comput. Netw. 140 (2018) 163–173, doi:10.1016/j.comnet.2018.01.038.
[30] Z. Yu, M.H. Au, R. Yang, J. Lai, Q. Xu, Achieving flexibility for ABE with outsourcing via proxy re-encryption, in: Proc. of ASIACCS'18, 2018, pp. 659–672, doi:10.1145/3196494.3196557.
[31] H. Cui, R.H. Deng, J. Lai, X. Yi, S. Nepal, An efficient and expressive ciphertext-policy attribute-based encryption scheme with partially hidden access structures, Comput. Netw. 133 (2018) 157–165, doi:10.1016/j.comnet.2018.01.034.
[32] F. Khan, H. Li, L. Zhang, J. Shen, An expressive hidden access policy CP-ABE, in: Proc. of DSC'17, 2017, pp. 178–186, doi:10.1109/DSC.2017.29.
[33] D. Boneh, B. Waters, Conjunctive, subset, and range queries on encrypted data, in: Proc. of TCC'07, 2007, pp. 535–554, doi:10.1007/978-3-540-70936-7_29.
[34] B. Schneier, Applied Cryptography: Protocols, Algorithms, and Source Code in C, 1996, doi:10.1016/S0740-624X(96)90083-0.
[35] P.G. Bradford, M.N. Katehakis, A probabilistic study on combinatorial expanders and hashing, SIAM J. Comput. 37 (1) (2007) 83–111, doi:10.1137/S009753970444630X.

Jialu Hao received the B.E. degree from Shandong University, Jinan, Shandong, in 2013, and the M.S. degree from National University of Defense Technology, Changsha, Hunan, in 2015, where he is currently pursuing the Ph.D. degree. In 2018, he joined the Department of Electrical and Computer Engineering, University of Waterloo as a visiting student. His research interests are in the areas of wireless network security, cloud security and applied cryptography.

Cheng Huang received the B.E. degree and M.E. degree from Xidian University, China, in 2013 and 2016 respectively, and was a Project Officer with the INFINITUS laboratory at the School of Electrical and Electronic Engineering, Nanyang Technological University till July 2016. Since September 2016, he has been a Ph.D. Candidate with the Department of Electrical and Computer Engineering, University of Waterloo, ON, Canada. His research interests are in the areas of applied cryptography, cyber security and privacy.



Jianbing Ni received the B.E. degree and the M.S. degree from the University of Electronic Science and Technology of China, Chengdu, China, in 2011 and 2014, respectively. He is currently working toward the Ph.D. degree with the Department of Electrical and Computer Engineering, University of Waterloo, Waterloo, ON, Canada. His research interests are applied cryptography and information security, with current focus on cloud computing, 5G network, mobile crowdsensing and Internet of Things.

Hong Rong received the B.E. degree from the South China University of Technology, in 2011, and M.S. degree from the National University of Defense Technology, in 2013, where he is currently pursuing the Ph.D. degree. His research interests include privacy preservation of big data, cloud computing, and virtualization.

Ming Xian received the B.E., M.S., and Ph.D. degrees from the National University of Defense Technology, in 1991, 1995, and 1998, respectively. He is currently a Professor in College of Electronic Science, National University of Defense Technology. His research focus on cryptography and information security, cloud computing, and system modeling, and simulation and evaluation.

Xuemin (Sherman) Shen received Ph.D. degree from Rutgers University, New Jersey (USA) in electrical engineering, 1990. Dr. Shen is a University Professor, Department of Electrical and Computer Engineering, University of Waterloo, Canada. His research focuses on resource management in interconnected wireless/wired networks, wireless network security, social networks, smart grid, and vehicular ad hoc and sensor networks. Dr. Shen is a registered Professional Engineer of Ontario, Canada, an IEEE Fellow, an Engineering Institute of Canada Fellow, a Canadian Academy of Engineering Fellow, a Royal Society of Canada Fellow, and a Distinguished Lecturer of IEEE Vehicular Technology Society and Communications Society. Dr. Shen is the Editor-in-Chief for IEEE Internet of Thing Journal and the vice president on publications of IEEE Communications Society. He received the Joseph LoCicero Award in 2015 and the Education Award in 2017 from the IEEE Communications Society. He has also received the Excellent Graduate Supervision Award in 2006, and the Outstanding Performance Award in 2004, 2007, 2010, and 2014 from the University of Waterloo, the Premier's Research Excellence Award (PREA) in 2003 from the Province of Ontario, Canada. Dr. Shen served as the Technical Program Committee Chair/Co-Chair for IEEE Globecom' 16, IEEE Infocom'14, IEEE VTC'10 Fall, the Symposia Chair for IEEE ICC'10, the Tutorial Chair for IEEE VTC'11 Spring, the Chair for IEEE Communications Society Technical Committee on Wireless Communications, and P2P Communications and Networking.