Hindawi
Wireless Communications and Mobile Computing
Volume 2021, Article ID 6686675, 15 pages
https://doi.org/10.1155/2021/6686675

Research Article

A Privacy-Preserving Attribute-Based Encryption System for Data Sharing in Smart Cities

Xieyang Shen,[1,2] Chuanhe Huang,[1,2] Danxin Wang,[1,2] and Jiaoli Shi[3]

[1]*School of Computer Science, Wuhan University, China*
[2]*Collaborative Innovation Center of Geospatial Technology, China*
[3]*School of Information Science and Technology, Jiujiang University, Jiujiang, China*

Correspondence should be addressed to Chuanhe Huang; huangch@whu.edu.cn

Received 28 June 2021; Accepted 8 September 2021; Published 8 October 2021

Academic Editor: Yushu Zhang

Information leakage and efficiency are the two main concerns of data sharing in cloud-aided IoT. The main problem is that smart devices cannot afford both energy and computation costs and tend to outsource data to a cloud server. Furthermore, most schemes focus on preserving the data stored in the cloud but omitting the access policy is typically stored in unencrypted form. In this paper, we proposed a fine-grained data access control scheme based on CP-ABE to implement access policies with a greater degree of expressiveness as well as hidden policies from curious cloud service providers. Moreover, to mitigate the extra computation cost generated by complex policies, an outsourcing service for decryption can be used by data users. Further experiments and extensive analysis show that we significantly decrease the communication and computation overhead while providing a high-level security scheme compared with the existing schemes.

1. Introduction

The smart city is considered to be one of the promising urban environments to achieve the goal of intelligent and effective data sharing via different sectors of society [1]. Government, organizations, and companies orchestrate a wide range of services through sensors or smart devices. These associations build a multiauthority environment producing massive data every second. The huge exchange causes concerns about security and privacy issues in data sharing. Such data loss may cause financial and physical problems for citizens, so data must be transformed and stored in an encrypted form.

Ciphertext attribute-based encryption (CP-ABE) is one of the possible solutions for those problems. However, existing CP-ABE schemes cannot fit the smart city environment because they still have some demerits on both expressive policy and privacy preservation. For instance, a temperature sensor encrypted the data with an access policy as $\{\text{Date} = \text{September 11th AND Distance} \leq 1000 \text{ meters, Identity} = (\text{administrators}\,;\,\text{subscribers})\}$. So, when a legal user with the attribute $\{(\text{Date} = \text{September 11th}), (\text{Distance} = 600 \text{ meters}), \text{subscribers}\}$ tries to get the temperature data, sensors must transform its policy about distance into $\{\text{Distance} = 1 \text{ meter OR Distance} = 2 \text{ meters} \ldots \text{OR Distance} = 999 \text{ meters}\}$ so that the legal user can access the data. Moreover, this kind of method of enumeration cannot be used when numbers in attributes are noninteger. Thus, we need a more flexible access control scheme to meet this challenge of unequal comparison.

As a consequence of this, a much more detailed policy also brings more privacy concerns. Recent smart city researches [2, 3] show that smart nodes like sensors or wearable devices are more likely to attract malicious attackers. Furthermore, fine-grained policies will reveal privacy information as most of the policies are stored on the cloud without hiding. There exist several methods of policy hidden in ABE, but they cannot support those point-in-interval policies. For this reason, compared to traditional cloud computing, it is much more urgent to deploy a method to hide access policies in smart city. On the other hand, a hidden access policy also brings a problem: it is tougher for users to know whether they can satisfy the access policy. This situation will take more calculation cost, especially for those lightweight devices. The existing access control scheme based on CP-ABE cannot satisfy both requirements properly. Some of the schemes [4–6] provide policy hiding method with traditional CP-ABE but cannot be used for unequal comparison policy while some others [7, 8] build just the reverse scheme.

The research goal of this paper is to construct an access control scheme with flexible policies to realize unequal attributes' comparison and further hide the policy in the ciphertext. Besides, to reduce the computation cost on smart devices, decryption for ciphertext can be outsourced to the cloud or other computing devices without leaking any data information. The practical significance of our work is that (1) the proposed scheme can implement fine-grained access control for unequal attribute comparison. (2) It can hide the access policy by transferring it into hidden values in the ciphertext. (3) Our scheme also provides outsourced decryption for energy- or computation-limited devices.

In order to present an ideal solution, we propose a new access control scheme that can implement expressive policy with attribute comparison by using 0-encoding and 1-encoding. At the same time, we hide the detailed access policy within a secret to avoid personal information leakage. At last, we implement an outsourcing decryption method to keep the safety of data as well as reduce the computation cost on the node side. By making an efficient solution for addressing the above issues, we try to decrease both storage and computation overheads.

In summary, our main contributions are threefold:

(i) We propose a more efficient access control method based on 0-encoding and 1-encoding so that the attributes of users and attributes in access policy can be compared with "≥" or "≤". With more flexible access control and less encryption/decryption time, our scheme only pays the price of affordable additional cost on the secret key size

(ii) We present a policy hidden method that can be implemented on a policy with attribute comparison. The attributes of the access structure are replaced with hidden values to achieve a complete access policy hidden method

(iii) We construct an optional lightweight outsourcing decryption method to significantly reduce the user-side computation overhead for only one pairing operation. Besides, the ciphertext length can be reduced to constant size when the access policy has no unequal attributes

The remainder of this paper is as follows. We first depict some related work by other researchers in Section 2. In Section 3, we list the main preliminaries and definitions of our scheme. Section 4 describes the details of our access control scheme in smart cities and introduces the security game based on it. Section 5 firstly analyzes the security of our scheme and gives the proof of the security game above and then shows the result of simulation and experiments. At last, in Section 6, we conclude our work and discuss the merits and demerits.

2. Related Work

Smart cities have changed the way of traditional computing services and need to be aware of security and privacy threats [9]. It unifies servers, clients, and smart devices to cooperate for offering services. In schemes proposed by Yang et al. [10] and Wang et al. [11], applications are distributed and widespread, supporting different nodes providing services quietly with nearly zero requirements for latency or price. To guarantee the safety and privacy of these nodes and data they generated, it is critical to survey the security and privacy issues with its nodes. The accessibility and Internet association of nodes have given more challenges like privacy preserving in [12], energy saving in [13], and anonymous authentication in [14]. In a word, we not only need to focus on security issues but also take the factor of smart cities into consideration.

To deal with the inequalities' comparison between attributes, some researchers have used the sketch proposed in [15], which presents a range as a predefined attribute using AND, OR to build the policy. For example, Shi et al. [16] proposed a predicate encryption scheme to set a point that is associated with ciphertext and a multidimensional range that is related to the key so that the decryption process works if the point is in the range. However, their scheme can only support access policy with AND gate. Later on, Gay et al. [17] developed a lattice-based variant of the previous scheme [16]. Furthermore, by introducing range attributes and range compare, Attrapadung et al. [7] have greatly improved the efficiency of numeric comparison. After that, they have proposed a generic method to convert traditional ABE into range attribute ones in [18]. Other methods like using 0-encoding and 1-encoding presented by Xue et al. [8] significantly reduced the computation overhead from $O(\log N)$ to $O(1)$. However, in this scheme, data owners must define their access policies first. Then, attribute authorities can generate private keys for specific users unsuitable for smart cities' environments. Furthermore, all the above schemes stored their access policies in plaintext to expose more information to the cloud. From the above, most of the previous work on unequal attribute comparison normally cannot hide the access policy.

Information leakage from access policy is also a significant security problem in data access control [19]. Despite the privacy-preserving issues of access policy designers, attackers could infer the information of data users by whether they can decrypt or not. To implement the hidden policy, Ying et al. [5] separated the attributes into attribute name and attribute value while hiding more sensitive attribute value instead of the attribute name. This partially hidden scheme can improve efficiency significantly. In the scheme proposed by Helil and Rahman [20], they presented an attribute hidden scheme by using sensitive data set to limit user access control by partially changing the data set constraint structure. They bring in a new entity named constraint detect server to check the user legitimacy before access operations. Wang et al. [21] defined a model under multiple data owner scenes with searchable and revocable policy hidden methods. By using keyword index, the accuracy of searching ciphertext can be significantly improved and correctly decrypted the ciphertext. By dividing users into different security domains, their scheme can manage keys for data owners as well as users in batch. Zhang et al. [6] proposed an access policy which is totally hidden to improve the privacy-preserving issues, but their access structure can only use "AND" gate. Fan et al. [22] proposed a policy hidden method with constant ciphertext length; they further introduced central authority (CA) to promulgate certification and keys in the multiauthority environment. As we can see from the above cases, most current work on policy hidden method cannot deal with inequalities' comparison between attributes due to the high computation cost.

The most significant drawback of ABE in smart cities is the computation overhead on the user side. In the scheme proposed by Rasori et al. [3], the computation cost on the user side is linear with the complexity of policy which cannot afford for most of the devices in smart cities. So, there are amounts of schemes eager to improve the efficiency of decryption, such as the scheme of Shi et al. [23] which used cloud-aided revocation and attribute scalability devised by Wang et al. [24]. At this point, Green et al. [25] introduced the ABE with outsourced decryption (OD-ABE) to outsource a large amount of computation to third-party entities such as cloud service providers. As a result, the computational cost of decryption can be decreased significantly. Besides, despite the similar problem of security issues in cloud computing, smart devices also face the problem of energy constraint. Thus, outsourcing their data and computation overhead to cloud service providers (CSP) seems to be a better solution. The team of Shi et al. [23, 26] proposed several schemes, exploring the way of cloud-aided decryption and revocation. Odelu et al. [27] seek another form of reducing the storage and communication overhead by using constant-size ciphertext. All of the schemes above tried to reduce the computation overhead differently. However, the data access control in smart city needs a more convenient way like outsourcing to decrease the overhead significantly.

From the above analysis of existing schemes, it is hard to find a balance between privacy-preserving and flexible access control because nodes and devices are enslaved to their computing power and capacity in smart cities. Most schemes only consider one or two aspects of unequal attribute comparison, policy hidden, and decryption outsourcing. In addition, we give the comparison of the related scheme in Table 1. As in our paper, we try to build a scheme with high-level security as well as do not increase the computation cost on smart nodes.

3. Preliminaries and Definitions

We will present some necessary definitions in this part. Firstly, we introduce the bilinear map and the decisional q-bilinear Diffie-Hellman exponent assumption. Secondly, we give 0-encoding and 1-encoding proposed by Lin et al. [30]. Finally, the generation of the extended attribute set first presented by [8] will be introduced, and we further improved the comparison method.

3.1. Bilinear Map. Let G and G_T be two multiplicative cyclic groups of prime p. Let g be the generator of group G, and then, we have $e : G \times G \longrightarrow G_T$ be a bilinear map that satisfies the following three properties.

(1) Bilinearity: if $\forall u, v \in G$ and $x, y \in Z_p$, then we have $e(u^x, v^y) = e(u, v)^{xy}$

(2) Nondegeneracy: $\exists g, h \in G, e(g, g) \neq 1$

(3) Computability: $\forall u, v \in G$, $e(u, v)$ is an admissible algorithm to compute the value of $e(g, h)$

3.2. Decisional Bilinear Diffie-Hellman Exponent Assumption. The decisional q-bilinear Diffie-Hellman exponent assumption is defined as follows. Let G be a group of prime order p with the generator g. Randomly choose a and s as $a, s \in Z_p^*$. Thus, an adversary must distinguish $e(g, g)^{a^{q+1}s} \in G_T$ from a random element in G_T when given $(g, g_1, \cdots, g_q, g_{q+2}, g_1)$. The advantage for the adversary is

$$\mathrm{Adv}^{q\text{-BDHE}} = \left|\Pr\left[B\left(\vec{y}, e(g, g)^{a^{q+1}s}\right) = 0\right] - \Pr\left[B\left(\vec{y}, R\right) = 0\right]\right|, \tag{1}$$

where R is the random element in G_T.

Definition 1. The q-BDHE assumption stands if and only if no probability polynomial time algorithm has a nonnegligible advantage to solve the decisional q-BDHE problem.

3.3. 0-Encoding and 1-Encoding. The attribute comparison of our scheme is based on an encoding method which was first proposed by [30].

The 0-encoding and 1-encoding are transferred from a binary string $S = S_n S_{n-1} \cdots S_1 \in \{0, 1\}^n$ with the length of n. The 0-encoding and 1-encoding of S are defined as follows:

$$\begin{aligned} S_s^0 &= \{s_n s_{n-1} \cdots s_{i+1} 1 \mid s_i = 0, \quad 1 \leq i \leq n\}, \\ S_s^1 &= \{s_n s_{n-1} \cdots s_i \mid s_i = 1, \quad 1 \leq i \leq n\}. \end{aligned} \tag{2}$$

These equations mean that 1-encoding is a set of all its substrings that contained "1" as the last number, and 0-encoding is a set of its substrings that contained "0" along with a "1" at the end of the set. The procedure of comparison is shown in Algorithm 1.

After encoding the binary string, we transfer two integers x and y into 1-encoding and 0-encoding as S_x^1 and S_y^0, respectively. Then, we use the formula to check the comparison as

$$x > y \Longleftrightarrow S_x^1 \cap S_y^0 \neq \varnothing. \tag{3}$$

From the formula, we can know that if and only if at least one same element in both S_x^1 and S_y^0 can we make the

Table 1: Comparison of existing schemes.

Methods	Basic algorithm	Unequal attribute comparison	Policy hidden	Outsourcing	Main contribution
[28]	CP-ABE and KP-ABE	No	No	No	The first sketch of ABE for range attributes
[18]	CP-ABE and KP-ABE	Yes	No	No	ABE for Boolean formula over range attributes
[16]	CP-ABE	Yes	Yes	No	Range attribute comparison for AND gates
[29]	CP-ABE	No	Partial policy hidden	Yes	Offline/online attribute-based encryption algorithms
[8]	CP-ABE and KP-ABE	Yes	No	No	Comparable attribute-based encryption using encoding
Proposed	CP-ABE	Yes	Yes	Yes	Attribute comparison and policy hiding

```
Input: attribute set $S$ with name and value
Output: encoding set for each unequal attributes $S^0$ or $S^1$
1: for any attribute $a \in S$ do
2:     Divide attribute into attribute name $a_n$ and attribute value $a_v$;
3:     if attribute $a$ is an unequal attribute then
4:       Transfer $a_v$ into binary string $a_{v2}$
5:       if the access policy needs $>$ or $\geq$ for attribute $a$ then
6:         for any digit $i$ in $a_{v2}$ do
7:           if $i = 1$ then
8:             Set the first $i$ digits $S_a$ as a substring of $a_{v2}$
9:             Add $S_a$ into $S_a^1$
10:          end if
11:        end for
12:      end if
13:      if the access policy needs $<$ or $\leq$ for attribute $a$ then
14:        for any digit $i$ in $a_{v2}$ do
15:          if $i = 0$ then
16:            Set the first $i - 1$ digits $S_a$ as a substring of $a_{v2}$
17:            Add 1 at the end of the substring $S_a$
18:            Add $S_a$ into $S_a^0$
19:          end if
20:        end for
21:      end if
22:      Combine the attribute name $a_n$ with $S_a^0$ or $S_a^1$ for unequal attributes
23:    end if
24: end for
```

Algorithm 1: 0- and 1-encoding generation.

conclusion of $x > y$. Let us take $x = 10$ and $y = 8$ as an example. We could see from Table 2, 1-encoding of x and 0-encoding of y have the same element 101 (with bold font), which means that $x > y$ holds. On the contrary, 0-encoding of x and 1-encoding of y do not have any same part, so the inequality $y > x$ is not tenable.

After transferring unequal attributes into 0- and 1-encoding, we will further attach them into the access structure. For example, we try to change the attribute Atr2 = 8 into Atr2 > 8 in the access policy tree in Figure 1. First of all, we turned the attribute into an access policy tree where the root node is an OR gate, and the leaf nodes are corresponding encoded binary strings. Then, we attach the subtree to the leaf node of the original access tree. As we mentioned before, if and only if at least one common element in both two sets (encoded by 0-encoding and 1-encoding, respectively) can the OR gate output the true result. So the OR gate can perfectly fit both encoding and the access policy. At last, when a user tries to compare its attribute Atr2 = 10, encoding the value into the 1-encoding format. Any binary string match with the subtree leaf node means that the attribute Atr2 of the user satisfied the access policy. Thus, the data user who wants to access data can use the corresponding encoded attribute to match the OR gate subtree. More details of unequal attribute comparison will be given in the following section.

3.4. Unequal Attribute Set. To realize unequal attributes comparing with ">," "<," "≤," and "≥," we developed the

Table 2: Comparison of 0-encoding and 1-encoding.

	0-encoding	1-encoding
$x = 1010_2$	11	1
	1011	**101**
$y = 1000_2$	11	1
	101	
	1001	

attribute set used by an access policy designer with three different types. Let S_0 be the original attribute set of unequal attributes. For those attributes that the policy only needs ">," we replace them with 0-encoding, and for those that only need "<," we replace them with 1-encoding. For those that both need ">" and "<," we replace them with both 0-encoding and 1-encoding. Furthermore, for comparing "≤" and "≥," check the equality first and then try the encoding way. Then, we denote the unequal attribute set with a new expression as S_1, which data owners use to define the unequal attributes in the access policy tree. For example, an unequal attribute in the access policy is "Distance <10"; we first transmit 10 to 1-encoding: "1" and "101" (as the binary number of 10 is "1010"). Then, the attribute set of "Distance < 10" is extended to $\text{Distance}^1 = 1$ and $\text{Distance}^1 = 101$.

As for the user side, they also need an unequal attribute set to compare their attributes with those in the access policy. However, as the access policy is hidden by data owners, users cannot tell whether the unequal attributes in the access policy need ">," "<," "≤," or "≥." So we replace those unequal attributes with both 0-encoding and 1-encoding and further denote the new set as S_2. In particular, as the 1-encoding on the user side is used to compare with the 0-encoding on access policy, to make it easier for comparison, we save 1-encoding in S_2 as 0-encoding form and vice versa. For instance, for a user with the unequal attribute "Distance" and the value is "Distance =10", we turn 10 both into 0-encoding and 1-encoding and save them as $\text{Distance}^1 = \{11, 1011\}$ (which actually is 0-encoding) and $\text{Distance}^0 = \{1,101\}$ (which actually is 1-encoding).

4. System Structure and Security Model

The data access control in smart cities has a strong demand on both user privacy and policy flexibility. However, we could learn from previous work that CP-ABE has the demerit to realize both of them. The main reason for this problem is that a fine-grained access policy will inevitably reveal more information about the data owner. Out of this consideration, we try to build our system structure to strike a balance between flexible policy and privacy preservation.

This section will briefly present the system model and the security game based on the above problems. Then, we will give the detailed process of our scheme. The list of notations can be found in Table 3.

4.1. System Model.

We present the access control system in the smart cities in this subsection. The system model is formed of five types of entities: cloud service providers (servers in the cloud), attribute authorities (AAs), smart devices while providing storing or collecting services, data owners, and data users. Our system model is shown in Figure 2, and we will introduce them one by one.

Cloud servers play the role of storing data and execute computation steps in the outsourced computing part. Typically, we consider those cloud servers are curious but honest, which means that cloud service providers would give their best to get the encrypted data stored on their servers but treat correctly doing what data owners want to do as a prerequisite.

Attribute authority (AA for short) manages attribute keys and entitles them to different users in the whole system. For the sake of authenticity, we defined that each AA is independent and can manage an arbitrary number of attributes. However, each attribute can only be managed, granted, or revoked by one AA. In other words, any attribute can only be authorized by one authority (which may suit most of the situation in reality). Besides, AA will define what kind of attributes are unequal attributes.

Smart devices are the infrastructure of our system; they can collect, store, and process data unconsciously. Besides, these devices may belong to different owners like attribute authorities, service providers, or even data owners. For other kinds of devices, they may have different functions like storing data or providing computing power.

Data owners make the definition of access policy before the data encryption on their side. However, the data owners can also be some intelligence devices or sensors; they can collect and store data by themselves. Besides, to decrypt the ciphertext, all the unequal attributes along with other attributes defined by owners in the access policy must be satisfied by access entities.

In smart city systems, data users usually use lightweight devices so that they cannot afford heavy computation overhead. As we proposed an entirely hidden access policy in our scheme, it is hard for data users to tell whether they can fit the access policy, so they might try to decrypt the data they did not get through. So, it is more suitable for users to mitigate their decryption computation overhead to cloud service providers.

4.2. Access Structure.

The construction of the access structure is based on (t-n)-Shamir Secret Sharing Scheme. Let us denote T as an access control policy tree, and the generation of T is divided into two parts: normal attributes and unequal attributes. We built T with the following two steps.

The first step is to assign secret s in the access policy tree. As we all know, the nonleaf node of T represents the threshold gate and the leaf node of T stands for attributes. At this point, we assigned the secret s to the root node and then assigned the rest of the nodes from top to bottom. For every unassigned nonleaf node, run the following steps recursively:

(1) If the unassigned node represents a threshold gate, divide the corresponding secret S_x with (t-n)-Shamir Secret Sharing Scheme where n is the number of its child nodes and t is the number to recover secret

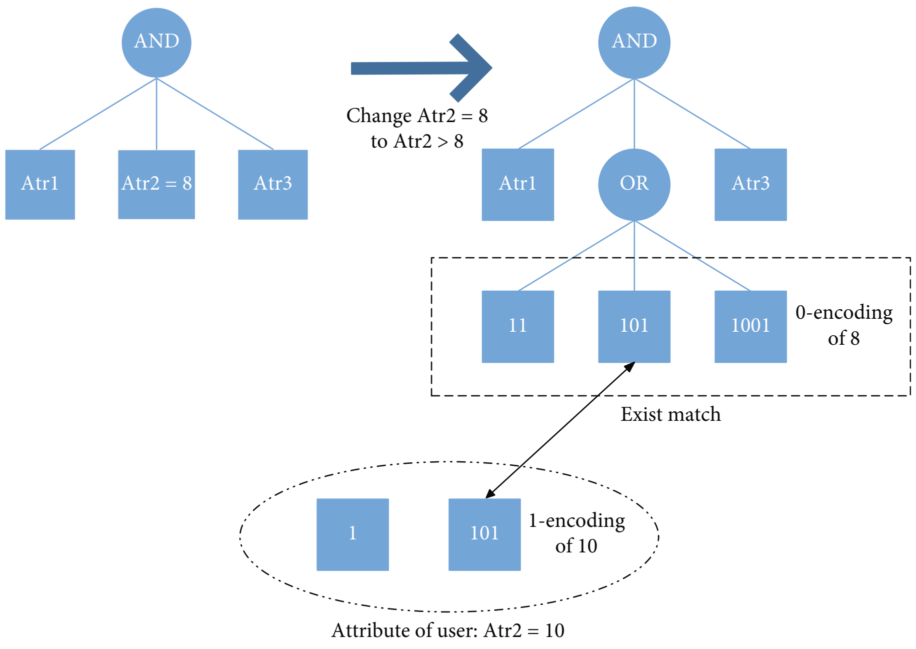

FIGURE 1: Model of comparison and matching for unequal attribute.

TABLE 3: List of notations.

Notations	Explanations
T	Access control policy tree
AID	Attribute authority index
U_d	Attribute set authorized by AA_d
U	Universal attribute set s
CT	Ciphertext of access control policy
gid	Global identity
APK	Attribute authority public key
ASK	Attribute authority secret key
$SK_{gid,i}$	Secret key of attribute i for user with gid
S_n	Extended attribute set with index n
S_x^0	A binary string with 0-encoding transformed by attribute x
S_x^1	A binary string with 1-encoding transformed by attribute x

(2) If the unassigned node represents an "AND" gate, use (t-n)-Shamir Secret Sharing Scheme the same as above where set $t = n$

(3) If the unassigned node represents an "OR" gate, use (t-n)-Shamir Secret Sharing Scheme the same as above where set $t = 1$

The second step is to attach unequal attributes. After all nonleaf nodes are assigned, replace the unequal attributes in the leaf nodes as the one-layer subtree, which is showed in Figure1. As these one-layer subtrees are all with "OR" gate, assign all of its leaf nodes with point 3 as we listed above.

4.3. Architecture Framework. We denote $\text{AID} = \{1, 2, \cdots\}$ be the index set of the AA and different attribute authorities can be listed as AA_1, AA_2. Besides, we use $d(d \in \text{AID})$ as the index of attribute authority AA_d and U_d is the set of attributes managed by AA_d. The attribute universe of the whole system as $U = \bigcup_{d \in \text{AID}} U_d$. Each AA does not share their attributes with other authorities and further does not need to know the existence of each other as we mentioned before, which means that $U_i \cap U_j = \varnothing$ for all $i \neq j \in \text{AID}$. We defined our architecture framework below:

(i) $\text{GlobalSetup}(\lambda) \longrightarrow \text{GPP}, \text{gid}$: the input of this phase is the security parameters λ and the output are the global public parameters GPP which will be used in other phases later. Besides, each user will get an identifier gid in this phase

(ii) $\text{AuthoritySetup}(\text{GPP}, U_d) \longrightarrow (\text{APK}, \text{ASK})$: each AA must run the setup algorithm before the whole system runs. It takes the inputs as GPP, which outputs in GlobalSetup phase, the attribute domain U_d of the authority itself. The output of this phase is the authority secret key ASK which is used for the authority itself and the public key APK, which sends to users for attribute authentication

(iii) $\text{SKeyGen}(\text{gid}, \text{GPP}, \text{ASK}) \longrightarrow (SK_{gid})$: the secret key generation outputs the secret key SK_{gid} for users. At the same time, the inputs are the users' global identity gid, the global public parameters GPP, and the secret key ASK. After generating

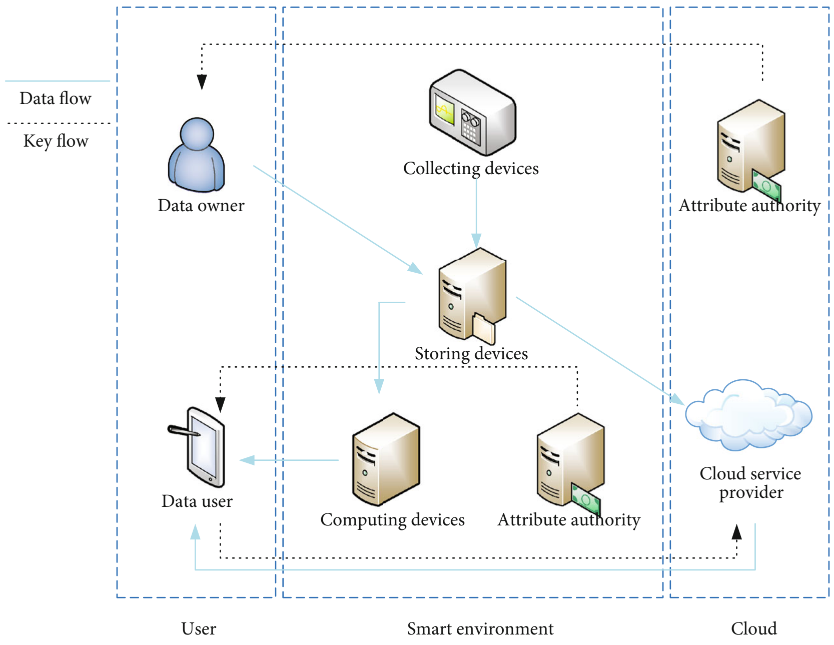

FIGURE 2: System model in smart cities.

the secret key, the authority will send it to the corresponding user in a secure channel

(iv) $\mathrm{Encrypt}(M, T, \mathrm{GPP}, \mathrm{APK}) \longrightarrow (\mathrm{CT})$: encryption phase including the input part of message M, access control policy T, the global public parameters GPP, and the public keys APK generated by different authorities. The outputs part only contain the ciphertext CT

(v) $\mathrm{Decrypt}(\mathrm{CT}, \mathrm{GPP}, \mathrm{ASK}) \longrightarrow (M) \mathrm{or} \perp$: the phase is run on the user side when they decided to decrypt the ciphertext by themselves. User inputs ciphertext download from other entities, their secret key ASK, and the global public parameters GPP. If the attribute satisfied the access policy (which is hidden in the ciphertext), the user could get the output as the message M. Otherwise, the output is a reject symbol $\perp$ implying decryption failed

(vi) $\mathrm{OutKeyGen}(\mathrm{SK}) \longrightarrow (\mathrm{OK}, \mathrm{RK})$: the outsourced key generation phase is run by the data user when the devices of user cannot afford the computation or energy cost on decryption. It takes inputs as the user's secret key and outputs a key pair OK (outsourced key) and RK (recovery key) used for outsourced decryption and recover the true plaintext

(vii) $\mathrm{OutDecrypt}(\mathrm{CT}, \mathrm{OK}) \longrightarrow \mathrm{CT}'$: this phase is run by cloud server or other computing devices. It takes the input of the previous ciphertext CT and the outsourced key from data user. It outputs the new ciphertext for user to decrypt

(viii) $\mathrm{UserDecrypt}(\mathrm{GPP}, \mathrm{RK}, \mathrm{CT}') \longrightarrow (M) \mathrm{or} \perp$: user decryption phase runs after receiving CT' from the outsourcing entities, and the user uses the recovery key and the public key to recover M from CT'

4.4. Security Model. In our schemes, we take these points into consideration. (1) The cloud service providers are honest but curious about the data and access policy; they will try their best to get them. (2) Cloud servers may send the data (in the form of ciphertext) to unauthorized users. (3) Users and cloud servers may collude with each other. Under such a presupposition, we defined our security game which runs between an adversary A and a challenger B with five steps.

(i) Setup: (1) the adversary A randomly chooses an access structure to challenge. After deciding the structure, A sends it to challenger B in a secure channel. (2) B runs GlobalSetup and publishes the global public parameters GPP. (3) B runs AuthoritySetup and asks all authorities to send their public key APK to A

(ii) Key query phase 1: A first generates an attribute set that cannot access the data through the access structure generated in Setup phase and sends the attribute set to B along with gid. Then, B runs SKeyGen to generate a secret key and OutKeyGen to generate an outsourced key for A, respectively

(iii) Challenge phase: A submits two equal length plaintext M_0 and M_1 to B. After this, the adversary should give the public key APK of all authorities whose attributes appear in the access policy to the challenger. Then, challenger throws a coin $\beta \in \{0, 1\}$ and sends it to A the encrypted M_β

(iv) Key query phase 2: A can makes as many queries as he wants according to phase 1

(v) Guess: A submits a guess β' for β. The advantage of A is defined as $|\Pr[\beta = \beta'] - 1/2|$

Definition 2. Our scheme is secure if any adversary cannot win the game above in any polynomial time with a nonnegligible advantage.

4.5. Access Control Scheme. In this section, we will explain the detail of our access control scheme step by step. Based on the architecture framework, our scheme contains four primary phases: system initialization (ran by authorities), key generation (ran by AA), data encryption (ran by owner), and data decryption (ran by user). Besides, the execution of the other three outsourcing phases depends on the requirements of users.

4.5.1. Phase 1: System Initialization. The system initialization phase runs before the whole system starts. In this phase, authorities generate essential parameters and publish the attribute public key. The first step is global setup and the second is authority setup.

(1) Global Setup. At the beginning of this phase, let a bilinear group G and the corresponding map $e : G \times G \longrightarrow G_T$ with the order p, g is a generator of G. The global parameter GPP which is used in other phases is published as $\text{GPP} = (p, g, e, H, G, G_T)$, where H is a hash function that maps a binary string of any length to an element of group G as $H : \{0, 1\}^* \longrightarrow G$. Besides, all users will be granted a global identifier gid in this phase.

(2) Authority Setup. Authority setup runs after GPP is generated. Each authority AA_d manages their own attribute universe U_d with n_d different attributes $\text{att}_i (i \in [1, n_d])$. We assume that each attribute only belongs to a specific authority. Thus, different attribute keys generated by different authorities will not have data conflicts. This kind of assumption is practicable as different authorities may belong to different associations or enterprises in reality.

AA_d chooses $a_i, b_i \in Z_p^* (i \in [1, n_d])$ for each attribute att_i (both normal attributes and unequal attributes). Authorities save the set of a_i and b_i as the authority secret key ASK, namely,

$$\text{ASK} = \{<a_i, b_i>\}. \tag{4}$$

After this, AA_d calculates x_i, y_i as follows:

$$\begin{aligned} x_i &= g^{-a_i}, \\ y_i &= e(g, g)^{b_i}. \end{aligned} \tag{5}$$

Then, AA_d publishes the combination of x_i, y_i for every attribute as the public key and can add a signature for integrity if needed. So the attribute authority public key is composed of

$$\text{APK} = \{<x_i, y_i>\}. \tag{6}$$

4.5.2. Phase 2: Key Generation. The key generation algorithm runs by authority and takes the input as the authority secret key ASK, user's global identifier gid, and corresponding attribute set S_2. This phase outputs the secret key $\text{SK}_{\text{gid},i}$ which associates with the user's global identity and the corresponding attribute.

In this phase, each AA first generates users' unequal attribute set from S_0 to transfer unequal attributes to 0-encoding and 1-encoding form. After that, authority randomly selects a security parameter related to gid as u_d and a set of security parameters for every unequal attribute in S_2 as $\{r_j \mid j \in S_2\}$. Then, user will get the secret from one authority as $D_j = g^{b_j + u_d} \cdot H(j)^{r_j}$ and $D'_j = g^{r_j}$ where j is the unequal attribute with the index of i. At last, the secret for unequal attributes will be combined by user as

$$D = \left\{ D_j = g^{u_d} \cdot H(j)^{r_j}, D'_j = g^{r_j} | \forall j \in S_2 \right\}, \tag{7}$$

where S_2 stands for the unequal attribute set. The attribute authority AA_d calculates the secret key for the rest of the attributes as

$$\text{SK}_{\text{gid},i} = g^{b_i + u_d} H(\text{gid})^{a_i}. \tag{8}$$

Then, the user can combine the secret key from each authority to set his own as follows:

$$\text{SK}_{\text{gid}} = \left\{ <\text{SK}_{\text{gid},i} = g^{b_i + u_d} H(\text{gid})^{a_i}>, D \right\}. \tag{9}$$

4.5.3. Phase 3: Data Encryption by Owners. In this phase, we can divide the operation of data owners into two steps. The first one is to build the access tree with unequal attributes. The data owner selects a random secret $s \in Z_p{}^*$ and assigns it to the root node. As we mentioned in Access Structure parts, we use $(t\text{-}n)$-Shamir Secret Sharing Scheme to set the secret s. For every unassigned node, owner selects a random polynomial q_x with the degree $d_x = k_x - 1$, where k_x is the threshold value. Then, the secret assigned by this node s_x is the constant term of q_x. Let us take threshold $(2, 3)$ as an example; then, we can generate the polynomial like $q_x = a_1 x + s_x$ and assign the secret $q_x(0) = s_x, q_x(1) = a_1 + s_x, q_x(2) = 2a_1 + s_x$ to each child node, respectively.

After building the access tree, each attribute in the access policy will be represented by a leaf node x for normal attributes or a one-layer subtree for unequal attributes. For all the one-layer subtree that has the "OR" gate, we can assign the corresponding secret s_x to the leaf node of the subtree. Then, we calculate C as

$$C = \left\{ C_j = g^{s_x}, C'_j = H(j)^{s_x} | \forall j \in S_1 \right\}. \quad (10)$$

The second step is to encrypt the data. For the sake of security and computation efficiency, we use symmetric encryption to encrypt the plaintext at first. The owner sets $C_0 = E_K(M)$ where K is the key of symmetric encryption. Then, the data owner selects a random secret $s \in Z_p{}^*$ and calculates

$$\begin{aligned} C_1 &= \prod g^{s_i} = g^s, \\ C_2 &= \left(\prod g^{-a_i}\right)^s, \\ C_3 &= K\left(\prod e(g, g)^{b_i}\right)^s. \end{aligned} \quad (11)$$

Same as the secret key, CT is also composed of two parts: C stands for the unequal attributes while C_1, C_2, C_3 is used for decryption. Thus, data owners can set the ciphertext CT as

$$\mathrm{CT} = \{C_0, C_1, C_2, C_3, C\}. \quad (12)$$

After the encryption phase, the access policy information will be transformed to s and hidden in the ciphertext CT but still have C for unequal attribute comparison. By doing this, we can hide the access policy from third-party entities (cloud or smart devices) and data users as well as providing a numerical comparison.

4.5.4. Phase 4: Data Decryption by Users. As we mentioned before, there are two kinds of scenes for user decryption. When the user wants to execute a data processing operation, devices can decrypt by themselves or not, depending on user settings or electric quantity. Moreover, if a user is far away from storing devices or the cloud, it can also call the outsourcing phase. Users can choose either phase 4 or phase 5 for decryption. This phase started on the user side when the user decided to decrypt the ciphertext.

Like the encryption phase, the decryption phase also needs to deal with the unequal attribute first either. As every unequal attribute in the access tree is transmitted to C in CT, the user first checks the attribute in C and finds the corresponding D in SK. Let the corresponding attribute be j, and then, the user calculates as follows:

$$F_j = \frac{e(C_j, D_j)}{e\left(D'_j, C'_j\right)} = \frac{e\left(g^{u_d} \cdot H(j)^{r_j}, g^s\right)}{e\left(g^{r_j}, H(j)^s\right)} = e(g, g)^{u_d s}. \quad (13)$$

Here, F_j is built to check whether the user's unequal attributes can meet the demand of access policy. After calculating F_j for all unequal attributes in the access policy, the user can get a secret for all unequal attributes as

$$S = \prod e(g, g)^{u_d s} = e(g, g)^{us} \left(u = \sum_{d=0}^{\mathrm{AID}} u_d \right). \quad (14)$$

Then, the plaintext can be recovered by

$$\begin{aligned} \mathrm{De} &= \frac{C_3 \cdot S}{e(H(\mathrm{gid}), C_2)e\left(\mathrm{SK}_{\mathrm{gid},i}, C_1\right)} \\ &= \frac{K\left(\prod e(g, g)^{b_i}\right)^s e(g, g)^{us}}{e(H(\mathrm{gid}), (\prod g^{-a_i})^s)e(\prod g^{b_i+u}H(\mathrm{gid})^{a_i}, g^s)} \\ &= \frac{K\left(\prod e(g, g)^{b_i+u}\right)^s}{e(H(\mathrm{gid}), (\prod g^{-a_i})^s)e(\prod g^{b_i+u}, g^s)e(\prod H(\mathrm{gid})^{a_i}, g^s)} \\ &= K. \end{aligned} \quad (15)$$

4.5.5. Phase 5: Data Decryption with Outsourcing. When a user needs a smart node to get data from a remote node or does not want to afford computation cost, this phase will run instead of phase 4. This phase contains three steps: OutKeyGen, OutDecrypt, and UserDecrypt. We will introduce these three steps together for brevity.

User first generates the outsourced key and recover key for OutKeyGen phase. The user generates the outsourced key as

$$\mathrm{OK}_{\mathrm{gid}} = \left\{ <\mathrm{OK}_{\mathrm{gid},i} = g^{\frac{(b_i+u_d)}{\sigma}} H(\mathrm{gid})^{a_i}>, D, H(\mathrm{gid}) \right\}. \quad (16)$$

Here, $\sigma \in Z^*$ as the recover key RK is selected by the data user.

The second step OutDecrypt is run by CSP or computing devices. The trustee first downloads the ciphertext from storing devices after receiving OK. The same as in decryption phase on users, unequal attribute key is calculated the same as in Equation (13). Then, the corresponding S of all unequal attributes is also generated. The ciphertext will be partially decrypted by

$$\begin{aligned} \mathrm{CT}' &= \frac{C_3 \cdot S}{e(H(\mathrm{gid}), C_2)e\left(\mathrm{OK}_{\mathrm{gid},i}, C_1\right)} \\ &= \frac{K\left(\prod e(g, g)^{b_i}\right)^s e(g, g)^{us}}{e(H(\mathrm{gid}), (\prod g^{-a_i})^s)e(\prod g^{b_i+(u_d/\sigma)}H(\mathrm{gid})^{a_i}, g^s)} \\ &= \frac{K\left(\prod e(g, g)^{(b_i+u)}\right)^s}{e(H(\mathrm{gid}), (\prod g^{-a_i})^s)e(\prod g^{(b_i+u_d)/\sigma}, g^s)e(\prod H(\mathrm{gid})^{a_i}, g^s)} \\ &= K\left(e(g, g)^{1/\sigma}\right). \end{aligned} \quad (17)$$

The last phase is UserDecrypt which is executed on the user side. After the user gets CT$'$ from outsourcing entities, retrieve the plaintext by using RK:

$$\mathrm{De} = \mathrm{CT}' e(g,g)^{-1/\sigma} = \left(K e(g,g)^{1/\sigma}\right) e(g,g)^{-1/\sigma} = K. \quad (18)$$

5. Security and Performance Evaluation

In this phase, we will first give the security proof according to the security model and then analyze the security properties from different aspects. Besides, performance evaluations on computation cost and storage cost will be given.

5.1. Security Proof

Theorem 3. *If the q-BDHE assumption holds, any adversary cannot break our scheme in polynomial time with a nonnegligible advantage.*

Proof. We first suppose that there exists an adversary A which can break our scheme with advantage ε. Then, there will be a simulator B which can play the q-BDHE game with advantage $\varepsilon/2$ according to Theorem 3. □

Our q-BDHE build on a group $(g, h = g^s, \overrightarrow{y}_{g,\alpha,n}, T)$, and then, the simulator can generate $\overrightarrow{y}_{g,\alpha,n} = (g, g_1, \cdots, g_{2n-1}, g_{2n})$. B randomly chooses $\tau \in \{0,1\}$ and can get T as

$$\begin{cases} T = e(g_{n+1}, h), \tau = 0, \\ T \in G_T, \tau = 1. \end{cases} \quad (19)$$

The simulation of our security game goes in five steps.

(1) Setup: the adversary first chose an access structure and built an attribute set containing normal and unequal attributes that do not satisfy the structure. After receiving the access structure, B randomly chooses $i^* \in I$ and $a_i, b_i, c_i \in Z_p$ where I represents the number of attributes. B calculates public keys as follows.

For $i = i^*$, B first checks whether the corresponding attributes belong to the access structure which is sent by A. If so, the public key is generated as

$$(x_i, y_i) = \left(g^{a_i} \prod g_{n+1-i}, e(g,g)^{c_i} e(g,g)^{\alpha^{n+1}}\right). \quad (20)$$

Otherwise,

$$(x_i, y_i) = \left(g^{-a_i}, e(g,g)^{b_i}\right). \quad (21)$$

For $i \in I - (i^*)$, B checks the attributes as above operations and generates (x_i, y_i).

If the attribute belongs to the access structure,

$$(x_{i^*}, y_{i^*}) = (g^{a_i} g_{n+1-i}, e(g,g)^{c_i}). \quad (22)$$

Otherwise, B calculates the public key as in Equation (21).

(2) Key query: the first time key query begins when A sent the attribute set and gid to B. Along with attribute, A also needs to send a token to B to ask for secret key SK or outsourced key OK. Before generating the secret key for this attribute set, B generates the hash function (used in secret key) as $H(\mathrm{gid}) = g \cdot g^z$ where $z \in Z_p$ and randomly selected $\sigma \in Z^*$. Furthermore, B selects an attribute att_k which does not belong to the access structure. Then, for the situation in Equation (20), B calculates the private key as

$$\mathrm{SK}_{\mathrm{gid},i} = \mathrm{OK}_{\mathrm{gid},i} = (g_k)^{-a_i} g^{c_i} \left(\prod_{i \in I-(i^*)} g_{n+1-i+k}^{-1} \right) (y_i)^{-z}. \quad (23)$$

For the situation in Equation (21), set

$$\mathrm{SK}_{\mathrm{gid},i} = \mathrm{OK}_{\mathrm{gid},i} = (g_k)^{-a_i} g^{c_i} g_{q+1-i+k} (y_i)^{-z}. \quad (24)$$

For the situation in Equation (22), set

$$\begin{aligned} \mathrm{SK}_{\mathrm{gid},i} &= H(\mathrm{gid}) g^{b_i+u}, \\ \mathrm{OK}_{\mathrm{gid},i} &= H(\mathrm{gid}) g^{(b_i+u)/\sigma}, \end{aligned} \quad (25)$$

where u is randomly chosen by B which is associated with gid. However, B cannot send both $\mathrm{SK}_{\mathrm{gid},i}$ and $\mathrm{OK}_{\mathrm{gid},i}$ with the same gid and i to A. Then, B returns the corresponding key to A depending on the token of whether using the outsourced key or not.

(3) Challenge: A randomly generates two equal length messages M_0 and M_1 for B to pick. After receiving the message, B flips a secure coin $v \in \{0,1\}$ to make sure A does not have any information about the flipping. For each a_i and c_i, we use $a_I = \sum a_i$ and $c_I = \sum c_i$ to define the ciphertext which is generated by B: sourced key or not.

$$\begin{aligned} C_1' &= h, \\ C_2' &= (\prod g^{-a_i})^s = \left(g^{-a_{i^*}} \prod_{l \in I-(i^*)} g_{n+1-l} \prod_{l \in I-(i^*)} g_{n+1-k}^{-1} \right)^s = h^{-a_I}, \\ C_3' &= M_v T e(g,h)^{c_I}. \end{aligned} \quad (26)$$

As we mentioned before in Equation (19), for $\tau = 0$ and $T = e(g_{n+1}, h)$, we can get

$$\begin{aligned}C_3{}' &= M_v T e(g,h)^{c_I} = M_v e(g_{n+1},h) e(g,h)^{c_I} \\ &= M_v e(g_{n+1},h) \prod_{i\in I} e(g,h)^{c_i} = M_v e(g,h)^{c_{i^*}+\alpha^{n+1}} \prod_{l\in I-(i^*)} e(g,g)^{c_l} \\ &= M_v \left(\prod e(g,g)^{b_i}\right)^s. \end{aligned} \tag{27}$$

Thus, $\mathrm{CT}' = \{C_0, C_1{}', C_2{}', C_3{}', C\}$ be the right ciphertext of M_v. Otherwise, when $\tau = 1$, T is a random value on group G_T; CT' is also a random value ciphertext.

(4) Key query: A can make as many queries as he wants according to phase 2 and B must return corresponding results. Here, we must emphasize that for the same attributes and gid, A cannot query both $\mathrm{SK}_{\mathrm{gid},i}$ and $\mathrm{OK}_{\mathrm{gid},i}$.

(5) Guess: A outputs a guess for v as v' according to the phase above. Then, B checks the result and outputs τ. For $v' \neq v$ outputs $\tau = 1$ and for $v' = v$ outputs $\tau = 0$.

The values of v and τ in our game are independent of each other so that A cannot obtain any information about v. When $\tau' = 0$, the ciphertext is valid and A can guess v with a nonnegligible advantage of $\Pr[v = v' | \tau' = 0] = (1/2) + \varepsilon$. However, when $\tau' = 1$, T is a random value and CT' cannot be identified either. So the probability of guessing v by A is $\Pr[v \neq v' \mid \tau' = 1] = 1/2$.

In conclusion, the advantage of B in q-BDHE game can be summarized as

$$\begin{aligned}\mathrm{Adv} &= \frac{1}{2}\Pr\left[v = v' \mid \tau' = 0\right] + \frac{1}{2}\Pr\left[v \neq v' \mid \tau' = 1\right] - \frac{1}{2} \\ &= \frac{1}{2}\left(\frac{1}{2}+\varepsilon\right) + \frac{1}{2}\frac{1}{2} - \frac{1}{2} = \frac{\varepsilon}{2}.\end{aligned} \tag{28}$$

Based on the above equation, we can know that if there does exist an adversary that can break our scheme, there must be a simulator that can play q-BDHE game with the advantage of $\varepsilon/2$. Relatively, no one can break our scheme under the q-BDHE construction in the security game proposed before.

5.2. Security Analysis. In CP-ABE, the access policy is attached to the ciphertext by transforming it into tree form. However, although this kind of method does provide convenience to access control, it also leaks the privacy of the access policy. Malicious users may infer the identity of data owners by different access policies they made. Consequently, the policy hiding scheme offers high-level security but introduces concerns about the correctness of the access policy. In our scheme, these concerns can be divided into two parts: policy confidentiality and unequal attribute comparison. Besides, we also take collusion attacks into consideration. Analysis of these security issues is listed below.

5.2.1. Policy Confidentiality. The access policy tree in our scheme is hidden through the secret s. By replacing each node of the access tree with a secret value associated with s, we can shift the operation of verifying access policy to recover the secret s. In particular, unequal attributes can be hidden through the OR gate subtree by being transformed into 0-encoding and 1-encoding ways. So the decryption result cannot leak any information about attributes or data. Furthermore, our scheme can prevent attacks like guess access structure through multiple access applications by hiding the access tree through the secret.

5.2.2. Unequal Attributes. The correctness of 0-encoding and 1-encoding can be found in [30]. Furthermore, if the attribute set of a user meets the demand of access policy, there must be an encoding that belongs to the extended attribute set. On the opposite, if the user's attribute does not satisfy the access policy, the match between the encoding set will be failed. Our extended attribute set contains not only attribute name but also attribute value which can also resist malicious users misusing encoding or fake attributes.

5.2.3. Collusion Attacks. There may exist several kinds of collude operations in our model, and we will dissect them one by one. The first one is collusion between different users. For those users who cannot satisfy the access control policy, we assume a scene that they want to combine their secret key to get a legal one. Since the prime order of their secret key is randomly chosen by authorities, respectively, no matter what kinds of attribute set they ever had, they cannot get a proper key in any case.

Another case is the collusion between an unauthorized user with gid_1 and a revoked user with gid_2. User with gid_1 may want the secret key gid_2 ever had to get a combination of attributes and his own gid_1. As mentioned before, this kind of collusion cannot exist either because of the random oracle and multiple authorities.

The last situation is about different users colluding with each other to speculate the access structure, especially the unequal range. Illegal users who cannot satisfy the access policy may want to construct a new key by corrupting with other illegal users. But the only feedback of a failure decryption operation is nothing but a fail symbol $\perp$; thus, illegal users cannot know exactly which attributes did not satisfy the policy. Moreover, every user has a specific secret key as gid of each user is different. So, a part of the secret key $H(\mathrm{gid})$ is totally different either. All in all, malicious users cannot combine their keys or guess any helpful information for decryption.

5.3. Performance Evaluation. The simulation platform of our scheme is on an Ubuntu 14.04 system with an Intel Core(TM) i5-5600 U at 2.6 GHz and 4 GB RAM. The implementation is based on Java Pairing-Based Cryptography library (JPBC ver2.0.0) and adopts a 160-bit group order on the curve $y^2 = x^3 + x$. Besides, we add one unequal attribute in every three attributes and set their value randomly. All the times shown in the below figures are the mean of 1000 times for the purpose of accuracy. We will analyze our

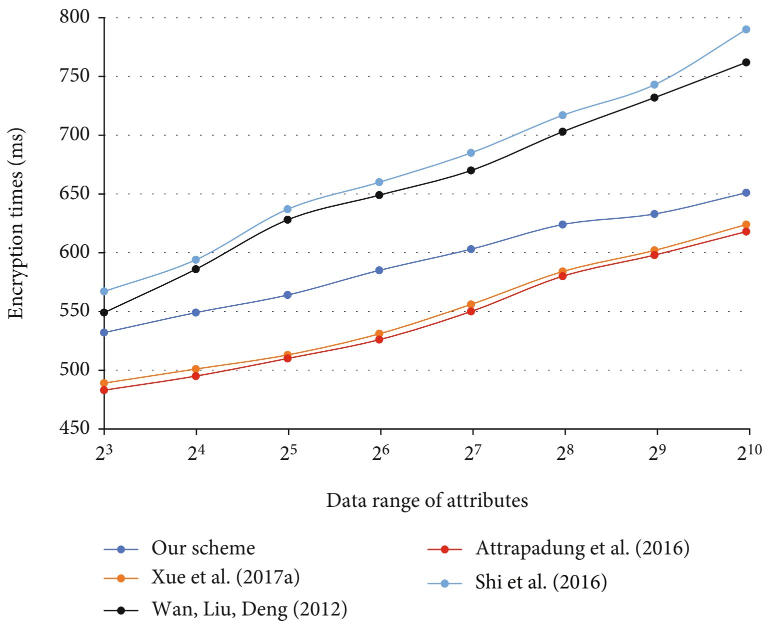

Figure 3: Encryption time on nodes.

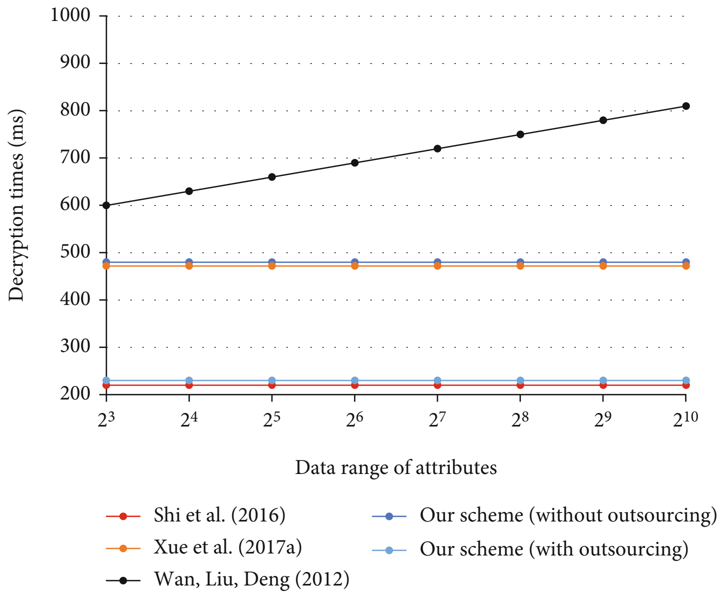

Figure 4: Decryption time on users.

scheme on both computation overhead and storage overhead. For computation overhead, we mainly focus on the cost of encryption and decryption phase; as for storage overhead, the size of ciphertext and secret key will be taken into consideration.

5.3.1. Computation Overhead. As shown in Figures 3 and 4, the computation cost of our scheme grows linearly on encryption and almost stays the same on decryption with the increasing of attribute value space. For encryption time, compared with the scheme revised by Wan et al. [31], our method does not need to generate another structure for policy and attributes, which can fit the environment of power-limited devices better. For the scheme in [8], as we all used 0-encoding and 1-encoding for attribute comparison, our scheme takes more time because of the secret generation for policy hiding. Our computation cost on encryption exceeded 650 ms when the attribute number reaches 2^{10}. It is affordable in consideration of data owner does not need to publish their data very often.

Decryption time in our scheme (without outsourcing) and scheme in [8] almost stay the same as the attribute number grows. When the user tries to use outsourcing for decryption, computation costs will be largely decreased. For all schemes we compared with, decryption time grows very slowly as the range of attributes increases.

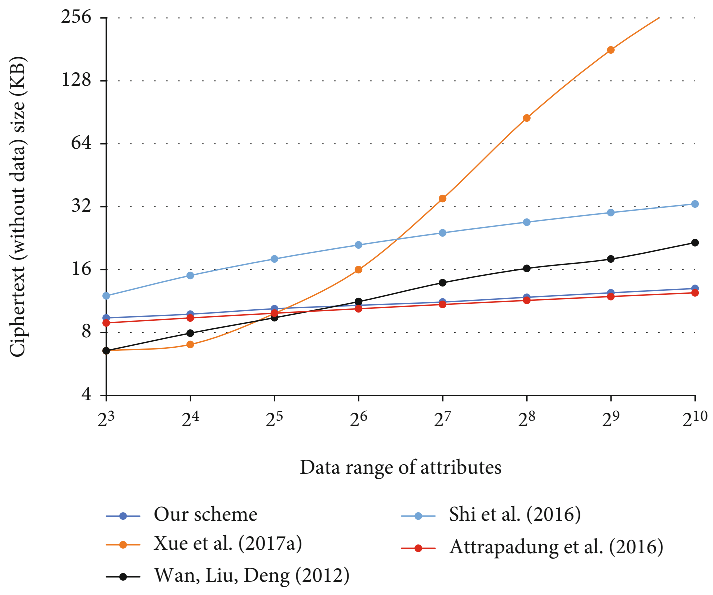

Figure 5: Comparison of ciphertext size.

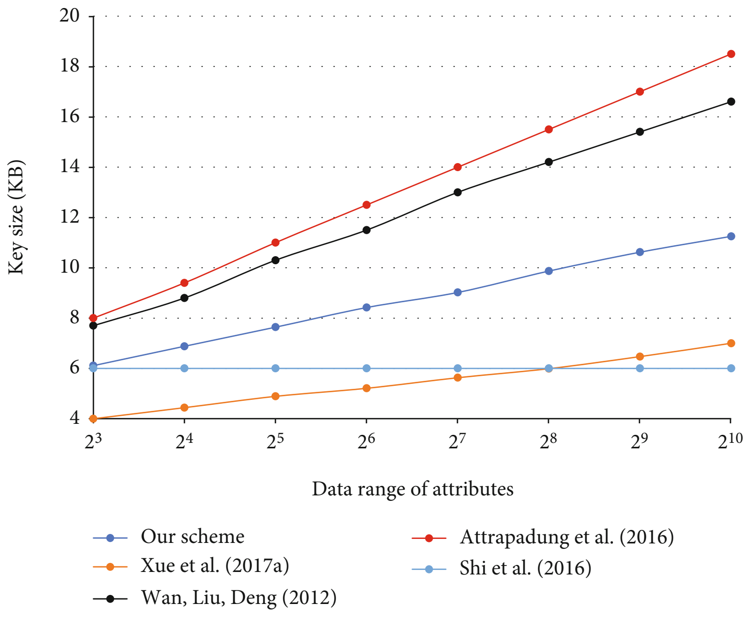

Figure 6: Comparison of secret key size.

The experiments on computation overhead show that our scheme takes a bit more time on both encryption and decryption compared to the existing methods. But as our scheme can provide policy hidden method which is not proposed by other schemes, we could say that we proposed a more cost-efficient solution in the smart city environment.

5.3.2. Storage Overhead. Figure 5 shows the comparison of ciphertext size growing trends through the increase of attribute number in the attribute universe. To clearly show the transformation of the ciphertext size, we do not take the symmetric encryption key and the data size into consideration. This means that we delete the data part in all schemes, while in our method, it is C_0 with symmetric encryption key K. So the ciphertext in our experiments is $\{C_1, C_2, C_3, C\}$. Due to the unequal attributes being a quarter of all attributes, the ciphertext size could be huge when the number of attributes is low. This is because we must generate C part in ciphertext according to Equations (10) and (12). However, as the number of attributes grows, ciphertext size in our scheme increases slowly and will have better performance in complicated real-world scenarios. Furthermore, the ciphertext size will greatly influence the performance of the whole system as it will introduce high communication costs and heavy storage overhead on different entities (including resource-constrained smart devices).

There is another advantage of the scheme on ciphertext size, which cannot directly be shown in Figure 5. Suppose the data owner defined an access policy without unequal attributes. In that case, the unequal attribute comparison part C is no longer needed, and the length of our ciphertext decreased to $2|G| + |G_T|$, where $|G|$ and $|G_T|$ are the length of elements in group G and G_T, respectively. From this aspect, we can see that the main cost of ciphertext depends on the unequal attributes as the number of attributes grows.

Similar to the ciphertext size, our secret key size is also larger than the scheme in [8] as shown in Figure 6. As we can see from Equations (7), (8), and (9), our secret key has two components: $\mathrm{SK}_{\mathrm{gid},i}$ and D. For the first part, with the same gid and corresponding a, b, u, the size of $\mathrm{SK}_{\mathrm{gid},i}$ stays the same as a constant size. As for D in Equation (6), its size is associated with the number of unequal attributes and their encoding binary strings. As the proportion of unequal attributes of our experiment is the exact, storage cost on D determines the number of its binary strings of 0-encoding and 1-encoding. With the combination of $\mathrm{SK}_{\mathrm{gid},i}$ and D, the size of the secret key is $\mathrm{Log}_2 N + C$ where N is the value space of attributes and C is a constant number. For the unequal attributes that have the same proportion in our experiment, key size grows steady as the attribute number grows. In this case, we could say that the additional overhead is affordable, because with more than one thousand attributes, our scheme only needs an extra 4 KB for key size.

Based on performance evaluations on both computation and storage costs, our scheme has advantages on decryption overhead at the expense of affordable extra secret key size. Furthermore, with the increase of the attribute universe, both computation overhead and storage overhead do not grow significantly. At this point, our scheme is suitable to be applied in smart city environments and power-constrained devices.

6. Conclusion

In this paper, we focus on how to provide both unequal attribute comparison and policy hiding method in smart city access control environment. Our scheme constructed an efficient solution by attaching an encoding subtree to the unequal attribute leaf node and transferred access policy into corresponding values to implement the above properties, respectively. Both theoretical analysis and experimental results show that compared to the existing works, our scheme implements better encryption and decryption speed as well as reduces storage overhead for ciphertext to some extent. Although our method takes extra space to store the secret keys, the rate of extra cost grows slowly enough as we only need an extra 4 KB for every thousand attributes. In this case, even the lightweight devices can afford the additional cost in exchange for protecting the privacy of access control policy.

Therefore, we believe that our scheme provides a more efficient method for ABE to overcome the weakness of high computational overhead and inflexible access control. However, the unequal subtree still takes more storage and communication cost and is hard to update or revoke. Accordingly, future research lies in unequal attribute updating and revocation as these operations will influence a large number of users and raise computation costs. For security and privacy parts, the problem of unequal attribute information leakage will be taken into consideration either.

Data Availability

The data used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This work was supported by the National Science Foundation of China under Grant Number 61772385 and Number 61572370.

References

[1] L. Cui, G. Xie, Y. Qu, L. Gao, and Y. Yang, "Security and privacy in smart cities: challenges and opportunities," *IEEE Access*, vol. 6, pp. 46134–46145, 2018.

[2] S. Belguith, N. Kaaniche, M. Laurent, A. Jemai, and R. Attia, "PHOABE: securely outsourcing multi-authority attribute based encryption with policy hidden for cloud assisted IoT," *Computer Networks*, vol. 133, pp. 141–156, 2018.

[3] M. Rasori, P. Perazzo, and G. Dini, "A lightweight and scalable attribute-based encryption system for smart cities," *Computer Communications*, vol. 149, pp. 78–89, 2020.

[4] J. Hao, C. Huang, J. Ni, H. Rong, M. Xian, and X. S. Shen, "Fine-grained data access control with attribute-hiding policy for cloud-based IoT," *Computer Networks*, vol. 153, pp. 1–10, 2019.

[5] Z. B. Ying, M. A. Jian-Feng, and J. T. Cui, "Partially policy hidden CP-ABE supporting dynamic policy updating," *Journal on Communications*, 2015.

[6] L. Zhang, Y. Cui, and Y. Mu, *Improving Privacy-Preserving CP-ABE with Hidden Access Policy*, Springer, 2018.

[7] N. Attrapadung, G. Hanaoka, K. Ogawa, G. Ohtake, H. Watanabe, and S. Yamada, "Attribute-based encryption for range attributes," in *International Conference on Security & Cryptography for Networks*, Springer-Verlag New York, Inc, 2016.

[8] K. P. Xue, J. Hong, Y. Xue, D. S. L. Wei, N. Yu, and P. Hong, "CABE: a new comparable attribute-based encryption construction with 0-encoding and 1-encoding," *IEEE Transactions on Computers*, vol. 66, no. 9, pp. 1491–1503, 2017.

[9] Y. Qu, M. R. Nosouhi, L. Cui et al., "Privacy preservation in smart cities," in *Smart Cities Cybersecurity and Privacy*, pp. 75–88, Elsevier, 2019.

[10] K. P. Xue, Y. Xue, J. Hong et al., "RAAC: robust and auditable access control with multiple attribute authorities for public cloud storage," *IEEE Transactions on Information Forensics and Security*, vol. 12, no. 4, pp. 953–967, 2017.

[11] W. Jing, C. Huang, and J. Wang, *Scalable Access Policy for Attribute Based Encryption in Cloud Storage*, Springer, 2015.

[12] S. Liu, J. Yu, C. Hu, and M. Li, “Traceable multiauthority attribute-based encryption with outsourced decryption and hidden policy for CIoT,” *Wireless Communications and Mobile Computing*, vol. 2021, Article ID 6682580, 16 pages, 2021.

[13] A. K. Junejo and N. Komninos, “A lightweight attribute-based security scheme for fog-enabled cyber physical systems,” *Wireless Communications and Mobile Computing*, vol. 2020, Article ID 2145829, 18 pages, 2020.

[14] A. Arfaoui, O. R. M. Boudia, A. Kribeche, S. M. Senouci, and M. Hamdi, “Context-aware access control and anonymous authentication in WBAN,” *Computers & Security*, vol. 88, p. 101496, 2020.

[15] J. Bethencourt, A. Sahai, and B. Waters, “Ciphertext-policy attribute-based encryption,” in *2007 IEEE Symposium on Security and Privacy (SP '07)*, Berkeley, CA, USA, May 2007.

[16] E. Shi, J. Bethencourt, H. Chan, D. Song, and A. Perrig, “Multi-dimensional range query over encrypted data,” in *2007 IEEE Symposium on Security and Privacy (SP '07)*, p. 350364, Berkeley, CA, USA, May 2007.

[17] R. Gay, P. Meaux, and H. Wee, “Predicate encryption for multi-dimensional range queries from lattices,” in *PKC*, J. Katz, Ed., vol. 9020, p. 752776, LNCS, 2015.

[18] N. Attrapadung, G. Hanaoka, K. Ogawa, G. Ohtake, H. Watanabe, and S. Yamada, “Attribute-based encryption for range attributes,” *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, vol. 101-A, no. 9, pp. 1440–1455, 2018.

[19] L. Gao, T. H. Luan, B. Gu et al., *Privacy-Preserving in Edge Computing*, 2021.

[20] N. Helil and K. Rahman, *CP-ABE Access Control Scheme for Sensitive Data Set Constraint with Hidden Access Policy and Constraint Policy*, Security and Communication Networks, 2017.

[21] S. P. Wang, T. Gao, and Y. Zhang, “Searchable and revocable multi-data owner attribute-based encryption scheme with hidden policy in cloud storage,” *PLoS One*, vol. 13, no. 11, 2018.

[22] Y. D. Fan, X. P. Wu, and J. S. Wang, “Multi-authority attribute-based encryption access control scheme with hidden policy and constant length ciphertext for cloud storage,” in *2017 IEEE Second International Conference on Data Science in Cyberspace (DSC)*, pp. 205–212, Shenzhen, China, June 2017.

[23] J. L. Shi, C. H. Huang, J. Wang, K. He, and J. H. Wang, “An access control scheme with direct cloud-aided attribute revocation using version key,” *Algorithms and Architectures for Parallel Processing, Ica3pp 2014, Pt I*, vol. 8630, pp. 429–442, 2014.

[24] J. Wang, C. Huang, N. N. Xiong, and J. Wang, “Blocked linear secret sharing scheme for scalable attribute based encryption in manageable cloud storage system,” *Information Sciences*, vol. 424, pp. 1–26, 2018.

[25] M. Green, S. Hohenberger, and B. Waters, “Outsourcing the decryption of ABE ciphertexts,” *Proceedings of the 20th USENIX Conference on Security*, USENIX Association, 2011.

[26] J. L. Shi, C. Huang, J. Wang, K. He, and X. Shen, “An access control scheme with dynamic user management and cloud-aided decryption,” *Security and Communication Networks*, vol. 9, no. 18, 5672 pages, 2016.

[27] V. Odelu, A. K. Das, Y. S. Rao, S. Kumari, M. K. Khan, and K. K. R. Choo, “Pairing-based CP-ABE with constant-size ciphertexts and secret keys for cloud environment,” *Computer Standards & Interfaces*, vol. 54, pp. 3–9, 2017.

[28] A. Sahai and B. Waters, “Fuzzy identity-based encryption,” *Advances in Cryptology – EUROCRYPT 2005*, vol. 3494, pp. 457–473, 2005.

[29] X. Yan, G. He, J. Yu, Y. Tang, and M. Zhao, “Offline/online outsourced attribute-based encryption with partial policy hidden for the internet of things,” *Journal of Sensors*, vol. 2020, Article ID 8861114, 11 pages, 2020.

[30] H. Y. Lin and W. G. Tzeng, *An Efficient Solution to the Million-Aires Problem Based on Homomorphic Encryption*, 2005.

[31] Z. Wan, J. Liu, and R. H. Deng, “Hasbe: a hierarchical attribute-based solution for flexible and scalable access control in cloud computing,” *IEEE Transactions on Information Forensics and Security*, vol. 7, no. 2, pp. 743–754, 2012.