IoT-Cloud Data Sharing and Access Control System with Efficient Policy Updating

Luqi Huang, Fuchun Guo, Willy Susilo, *Fellow, IEEE*, Li Wang

***Abstract*—The integration of the Internet of Things (IoT) with cloud computing has expanded the scope of IoT applications but also introduced challenges in dynamic data management. Attribute-Based Encryption (ABE) serves as a crucial technology for constructing access control systems in data sharing environments. ABE schemes with policy updating capabilities allow data owners to dynamically and frequently modify access policies. Existing ABE schemes typically outsource the task of policy updating to a cloud server; however, the size of the update keys remains linear with the number of updated attributes and depends on the types of updated gates. The resulting updating costs are not less than generating a new ciphertext for data owners, especially when updating multiple attributes and threshold gates. Therefore, in this paper, we propose a novel ABE scheme, termed Policy Updatable Ciphertext-Policy ABE (PU-CP-ABE), which enables efficient and flexible policy updating. Our construction ensures that the size of update keys depends only on the number of updating gates, independent of both the number of attributes and the gate types. Furthermore, PU-CP-ABE provides a unified update mechanism that supports AND, OR, and threshold gates without requiring distinct update keys for different gate types.**

***Index Terms*—Attribute-based encryption, Internet of things, Cloud data sharing, Policy updating, Dynamic data management**

I. Introduction

The Internet of Things (IoT) consists of numerous interconnected smart devices, but it often faces challenges such as limited storage, computational capacity, and energy efficiency, which can significantly affect service performance. Cloud computing offers a compelling solution to these limitations. This convergence has given rise to the IoT-cloud paradigm, an integrated framework that leverages the strengths of both technologies to enable more efficient, scalable, and intelligent services.

However, as many IoT devices offload data processing to the cloud, security concerns within IoT-cloud environments become increasingly critical. In particular, the highly sensitive nature of certain data types, such as medical records, raises serious concerns regarding data confidentiality and privacy. Attribute-Based Encryption (ABE) has emerged as a promising cryptographic primitive to address this challenge. ABE enables data owners to define fine-grained access policies and encrypt data accordingly, ensuring that only users whose attributes satisfy the access policy can decrypt the data.

In IoT-cloud systems, data owners are often required to dynamically adjust access control over their data due to frequent changes in the set of authorized users. These changes may arise from personnel turnover, evolving organizational structures, temporary collaborations, or shifts in user roles and responsibilities. However, once data is encrypted and outsourced, owners typically lack a local copy, requiring them to download, re-encrypt under a new policy, and re-upload the ciphertext, resulting in high communication and computational overhead. To mitigate this, servers are delegated to perform access policy updates on the ciphertext using update keys provided by data owners.

One typical scenario involves an Electronic Health Record (EHR) system within an IoT-cloud infrastructure, as illustrated in Fig. 1. Patients with chronic or severe illnesses are equipped with implantable or wearable sensors that continuously monitor physiological parameters. These health data are collected by the Data Collection Layer (step 1) and processed by third-party health IT providers such as Epic Systems, Cerner, and Allscripts. In the Data Processing Layer, sensitive data are encrypted using ABE under an access policy $\mathcal{T}_1$: "att_A = Senior Doctor" OR ("att_B = Cardiology Department" AND "att_C = Experience > 5 years"). The encrypted data (original ciphertext) are then transmitted to the cloud servers in the Application Service Layer (step 2).

In real-world healthcare systems, however, access policies often require complex updates involving multiple attributes and threshold gates. As clinical workflows have evolved, nurses have become key participants in patient care. To reflect this, third-party providers issue an update request (step 3): the original policy should be modified to add the attribute "att_D = Nurse Practitioner" and replace the strict AND gate with a (2,3) threshold gate. The cloud server in the Application Service Layer performs the policy update on the existing ciphertext (step 4) and returns the updated ciphertext to the hospital (step 5). The new ciphertext (step 6) now grants access to any user who satisfies any two of the following three conditions: "att_A = Senior Doctor," "att_D = Nurse Practitioner," and "att_B = Cardiology Department AND att_C = Experience > 5 years".

In real-world scenarios, most ABE schemes that support policy updating [1], [2], [3], [4], [7], [8], [9] are built upon

Willy Susilo was supported by the Australian Research Council (ARC) Laureate Fellowship under Grant FL230100033 and the ARC Discovery Project under Grant DP240100017. Fuchun Guo was supported by the ARC Future Fellow under Grant FT220100046. *(Corresponding author: Luqi Huang, lh852@uowmail.edu.au)*

W.Susilo, F.Guo and L.Huang are with the Institute of Cybersecurity and Cryptology, School of Computing and Information Technology, University of Wollongong, Wollongong, Australia.

Li Wang is with the School of Computer Science and Technology, Anhui University, Hefei, China.

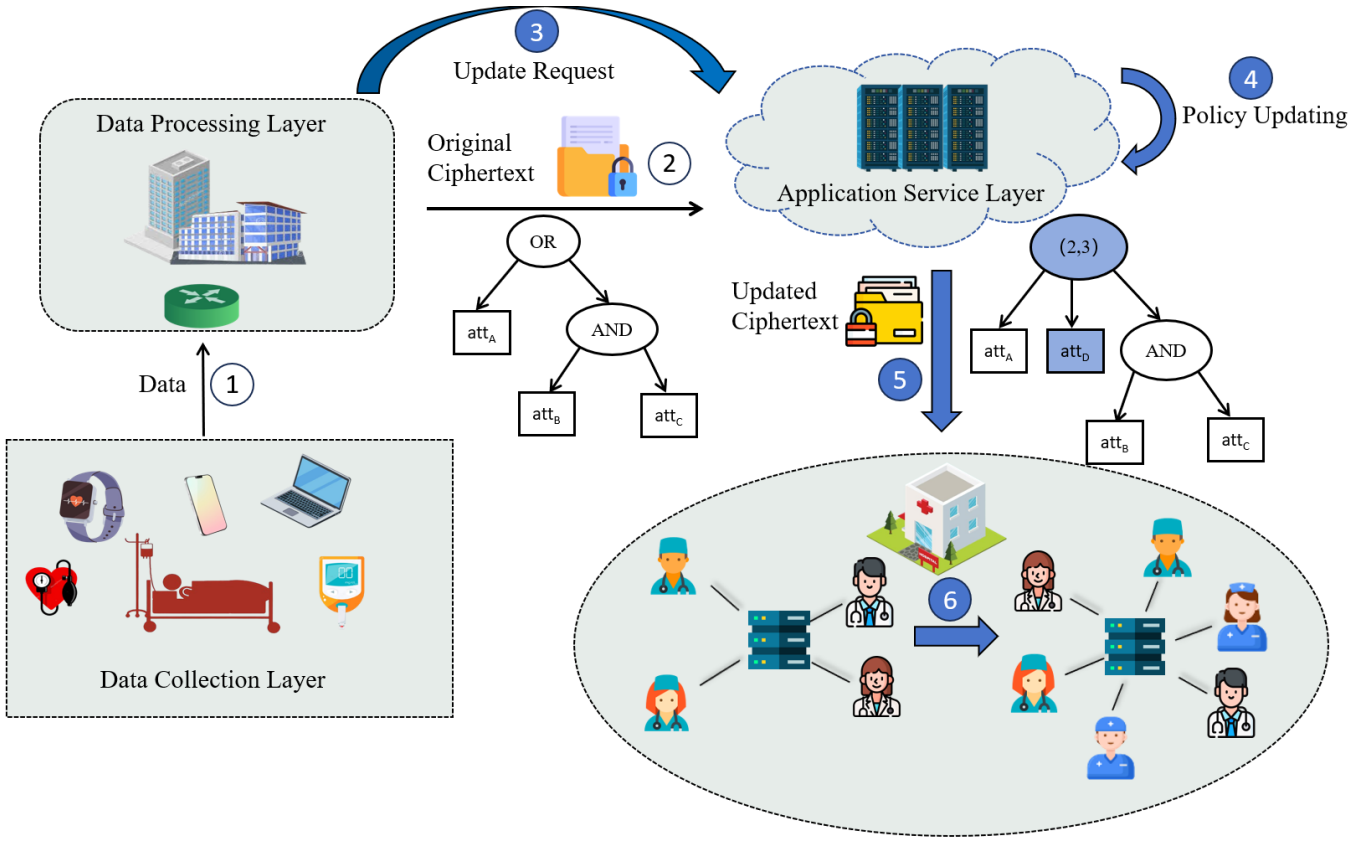


Fig. 1. Access policy updating of electronic health record IoT-cloud system

the Linear Secret Sharing Scheme (LSSS). However, LSSS can directly implement only OR and AND gates. Thus, transforming a typical (t, n)-threshold access structure into an LSSS matrix requires decomposing it into a combination of basic gates, which results in significant matrix expansion. Moreover, a single attribute often maps to multiple rows in the access matrix, meaning that even a minor modification is necessary for altering much of the matrix structure.

Consequently, complex updates involving multiple attributes or threshold gates demand extensive modifications to the LSSS matrix, leading to substantial computational and communication overheads for data owners during update-key generation. Although policy-updatable ABE schemes based on access trees [5], [6] offer better efficiency than their LSSS-based counterparts, the size of update keys still scales linearly with the number of updated attributes under complex updating requirements.

Therefore, achieving both efficiency and flexibility in policy updating remains a key challenge in ABE systems. To address this, we propose a new ABE scheme that enables efficient policy updating, specifically designed for real-world scenarios with dynamic and diverse updating needs.

A. Our Contribution

In this paper, we propose a new ABE scheme that supports efficient and flexible policy updating for encrypted data sharing in the IoT-cloud, called Policy Updatable CP-ABE (PU-CP-ABE). We first extend the basic threshold scheme in [10] to a CP-ABE scheme based on the access tree and then propose dynamic policy updating algorithms. Our PU-CP-ABE supports expressive policy updating for OR, AND, threshold gates and dramatically reduces the size of update keys, especially when complex policy updating. It minimizes the computational and communication overhead of policy updating for data owners. The primary contributions are detailed in this subsection:

- **Lightwight update key.** Our PU-CP-ABE scheme minimizes the size of update keys compared with most policy updating ABE schemes. The size of our update keys is linear with the number of updating gates rather than the number of attributes and the type of gates. We especially achieve constant size update keys when single gate updating is not limited by the number of attributes and the different gates, leaving extremely low overheads for data owners.
- **General policy update.** Our PU-CP-ABE scheme supports general access policy update algorithms of AND, OR and threshold without classification of different gates. Compare to existing schemes that rely on different update keys or specific algorithms for different types of gates, our approach introduces a single unified mechanism. This design removes structural restrictions and enables flexible policy updates that can adapt to diverse and dynamic access requirements in practical applications.

- **Security and implement.** We provide a formal security proof demonstrating that the scheme satisfies IND-sCPA security within a clearly defined model. Furthermore, we implement the proposed ABE scheme featuring policy updating and evaluate its performance through practical experiments. The results confirm that our approach supports fine-grained access control while maintaining compact ciphertexts and enabling adaptable policy updates.

B. Related Work

Shamir first introduced the concept of identity-based cryptography in 1984 [11]. Later in 2001, Boneh and Franklin [12] developed the first practical Identity-Based Encryption (IBE) scheme, which allowed encryption using identity information as public keys. This was followed by the introduction of fuzzy IBE and the proposal of ABE by Sahai and Waters [13].

ABE schemes are generally categorized into Ciphertext-Policy ABE (CP-ABE) and Key-Policy ABE (KP-ABE). In CP-ABE, the access policy is embedded within the ciphertext, whereas in KP-ABE, it is included in the user's key. The first implementations of KP-ABE and CP-ABE were presented by Goyal et al. [14] and Bethencourt et al. [15]. Since then, many enhanced ABE variants have been developed, including revocable ABE [16], [17], accountable ABE [18], [19], multi-authority ABE [20], [21], policy-hiding ABE [22], [23], and hierarchical ABE [24], [25].

Policy-updating ABE. Despite its widespread use, efficiently updating access policies in ABE systems remains a key issue. Goyal et al. introduced key delegation in [14], laying the foundation for update mechanisms. Sahai et al. later proposed a ciphertext delegation approach [26], although it required that updated policies remain subsets of the original ones.

Large update key. In 2015, Yang et al. [1] proposed an outsourced strategy update approach based on the LSSS. In this model, update keys are generated by data owners and uploaded to a cloud server, which modifies the related ciphertext components accordingly. The update process compares the original and new access policies and categorizes the attributes of the new policy. Ying et al. [2] improved the underlying security model but retained the same update mechanism. However, both schemes require re-encrypting the entire ciphertext regardless of the change in access policy, resulting in high overhead.

In a similar effort, Ying et al. [3] presented a reliable update framework using three distinct types of update keys, and Hao et al. [4] designed a large-universe ABE with update sizes that scale linearly with the number of policy rows. Fugkeaw et al. [5] incorporated proxy re-encryption, though this required data owners to upload encrypted versions of updated policies, increasing system complexity.

Limited policy update. In 2022, Wang et al. [7] proposed the ABE scheme capable of supporting fine-grained policy updates, excluding threshold gate modifications. Susilo et al. [8] introduced a modular access control extension that behaves like an OR-gate. Huang et al. [9] proposed both restrictive and non-restrictive policy extensions, mimicking AND and OR gate behavior. However, these schemes lack comprehensive flexibility, as they are limited to specific gate types and do not accommodate more diverse or real-world update scenarios.

IoT-cloud. In IoT-cloud environments, the rapid increase in connected devices and services has resulted in significant strain on centralized cloud infrastructures. To improve service delivery, Barcelo et al. [27] addressed the service distribution problem through a linear programming approach. Nonetheless, relying on a single service model often fails to meet the heterogeneous and evolving requirements of users. To resolve this limitation, Baker et al. [28] developed a service composition algorithm tailored for multi-cloud IoT scenarios.

As the volume of services and complexity of user needs continue to grow, the service optimization problem emerges. To address this, Xu et al. [29] proposed a domain-based optimization strategy utilizing artificial bee colony algorithms. In practice, IoT-cloud applications frequently require access to data distributed across different geographical locations. To maintain high-quality service and ensure user satisfaction, Chatterjee et al. [30] designed an optimal selection rule for choosing the best-fit data center, each hosting a single virtual machine, to fulfill user requests efficiently.

II. Preliminary

A. System description

This section begins by providing an overview of our model. These key players include the authority, cloud server, data owners, and data users.

- **Authority.** The key authority plays a fundamental role in initializing system parameters. It also handles the issuance of keys to users, assigning access permissions according to their associated attributes. The authority is considered completely trustworthy and is assumed not to collude with any parties.
- **Cloud server.** The cloud server stores encrypted data, manages ciphertext updates, and facilitates user access. It is assumed to be honest-but-curious: it follows the protocol correctly but may try to infer information from the data without colluding with outsiders.
- **Data owner.** Data owners definine access policies and encrypting data under these policies. Moreover, they entrust the cloud server with the task of updating access policies for the encrypted data. Data owners carry out their responsibilities by encrypting data, uploading the ciphertexts, and honestly submitting update requests.
- **Data user.** This role represents the individual attempting to access the protected data. The authority issues a private key to the user. When the user's attributes fulfill access conditions in the encrypted content, they are able to decrypt the ciphertext and obtain the original data.

B. Cryptographic background

1) Bilinear Maps: Our design is based on some facts about groups with efficiently computable bilinear maps.

Let $\mathbb{G}$ and $\mathbb{G}_T$ be two multiplicative cyclic groups of prime order p. Let g be a generator of $\mathbb{G}$. A bilinear map is an injective function $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$ with the following properties:

1. **Bilinearity:** For all $u, v \in \mathbb{G}$ and $a, b \in \mathbb{Z}_p$, we have $e(u^a, v^b) = e(u, v)^{ab}$.
2. **Non-degeneracy:** $e(u, v) \neq 1$.
3. **Computability:** There is an efficient algorithm to compute $e(u, v)$ for $\forall u, v \in \mathbb{G}$.

2) Access tree: Let $\mathcal{T}$ be a tree structure used to define an access policy, where each internal (non-leaf) node represents a threshold gate. For a node x, let num_x denote the number of its child nodes, and t_x its threshold. The condition $0 \leq t_x \leq num_x$ must hold. Each leaf node x is linked to a specific attribute. The child nodes of any given node are indexed from 1 to num.

Let $\mathcal{T}_x$ denote the subtree rooted at node x within $\mathcal{T}$. A set of attributes A satisfies the subtree $\mathcal{T}_x$ if $\mathcal{T}_x(A) = 1$. The function $\mathcal{T}_x(A)$ is evaluated recursively: if x is an internal node, compute $\mathcal{T}_{x'}(A)$ for each child x' of x. The result is 1 if at least t_x of these children return 1. For a leaf node x, $\mathcal{T}_x(A)$ equals 1 if the corresponding attribute t_x exists in A.

III. System and security definitions

A. System Definition

Based on the framework definition in [1], we show our PU-CP-ABE consisting of the following algorithms:

- **Setup**$(\lambda, U) \rightarrow (mpk, msk)$. The setup algorithm takes as input the security parameter λ, the attribute universe U. It outputs the master public key, master secret key (mpk, msk).
- **KeyGen**$(mpk, msk, A) \rightarrow sk_A$. The key generation algorithm takes as input the attribute set A and the master public key and secret key mpk, msk. It returns the private key sk_A.
- **Enc**$(M, mpk, \mathcal{T}) \rightarrow (ct_{\mathcal{T}}, \mathbb{E}_{\mathcal{T}})$. The encryption algorithm takes as input a message M, the master public key mpk, the access tree $\mathcal{T}$. It returns the ciphertext $ct_{\mathcal{T}}$ and random numbers $\mathbb{E}_{\mathcal{T}}$ used in $ct_{\mathcal{T}}$.
- **Dec**$(mpk, sk_A, ct_{\mathcal{T}}) \rightarrow M$. The decryption algorithm takes as input the master public key mpk, private key sk_A and the ciphertext $ct_{\mathcal{T}}$. It outputs the message if A satisfies $\mathcal{T}$.
- **UPKeyGen**$(mpk, \mathbb{E}_{\mathcal{T}}, \mathcal{T}, \mathcal{T}^{'}) \rightarrow tk_{\mathcal{T}\rightarrow\mathcal{T}'}$. The update key generation algorithm takes as input the master public key mpk, random numbers $\mathbb{E}_{\mathcal{T}}$ in $ct_{\mathcal{T}}$, and the original access tree $\mathcal{T}$, updated access tree $\mathcal{T}^{'}$. It outputs the update key $tk_{\mathcal{T}\rightarrow\mathcal{T}'}$.
- **CTUpdate**$(mpk, ct_{\mathcal{T}}, tk_{\mathcal{T}\rightarrow\mathcal{T}'}) \rightarrow ct_{\mathcal{T}'}$. The ciphertext update algorithm takes as input the master public key mpk, the ciphertext $ct_{\mathcal{T}}$ and the update key $tk_{\mathcal{T}\rightarrow\mathcal{T}'}$. It outputs the updated ciphertext $ct_{\mathcal{T}'}$.

Correctness: The correctness of our PU-CP-ABE are as following: For any master public and secret keys $(mpk, msk) \leftarrow$ Setup(λ, U) and the original ciphertext $ct_{\mathcal{T}} \leftarrow$ Enc$(M, mpk, \mathcal{T})$ or the updated ciphertext $ct_{\mathcal{T}'} \leftarrow$ CTUpdate$(mpk, ct_{\mathcal{T}}, tk_{\mathcal{T}\rightarrow\mathcal{T}'})$ output by CTUpdate algorithm, and the private key $sk_A \leftarrow$ KeyGen(mpk, msk, A), if the attribute set A satisfies the access tree $\mathcal{T}$, the decryption algorithm correctly recovers the original message M. Otherwise, the ciphertext $ct_{\mathcal{T}}$ cannot be decrypted using the private key sk_A.

B. Security Model

Following [1], we construct our security model based on semantic security under *Indistinguishability under Selective Chosen Plaintext Attacks* (IND-sCPA). The model is a game between challenger C and adversary $\mathcal{A}$, where $\mathcal{A}$ first selects a challenge access tree $\mathcal{T}^*$. For any attribute set A not satisfying $\mathcal{T}^*$, $\mathcal{A}$ cannot computationally distinguish the encrypted message in the challenge ciphertext using sk_A.

- **Initialization.** The adversary $\mathcal{A}$ selects a target access structure $\mathcal{T}^*$ that it intends to compromise and forwards it to the challenger C.
- **Setup.** The challenger C executes the setup procedure to produce the master public/secret key pair (mpk, msk) using a security parameter λ, and defines the universal attribute set U. It transmits mpk and U to $\mathcal{A}$ while retaining msk to handle future queries.
- **Query phase 1.** $\mathcal{A}$ can make the following queries: $\mathcal{A}$ queries private keys with attribute sets $A_1, \ldots, A_{q_1}$ that attribute sets $A_i, i \in \{1, \ldots, q_1\}$ could not satisfy the challenge access tree $\mathcal{T}^*$. C runs the private key generation algorithm to compute sk_{A_i} and then sends them to $\mathcal{A}$.
- **Challenge.** $\mathcal{A}$ outputs two equal length messages M_0, M_1. C flips a random coin b, and encrypts M_b. The ciphertext $ct_{\mathcal{T}^*}$ under the challenge access tree $\mathcal{T}^*$ are given to $\mathcal{A}$.
- **Query phase 2.** $\mathcal{A}$ can query more private keys same as in phase 1. $\mathcal{A}$ also make more update key queries by submitting access trees $(\mathcal{T}_j, \mathcal{T}_j^{'})$ under the restriction that query original access tree $\mathcal{T}_j$ cannot be the same as $\mathcal{T}^*$. In addition, $\mathcal{A}$ can also make ciphertext updating queries expect for $ct_{\mathcal{T}^*}$.
- **Guess.** $\mathcal{A}$ outputs a guess $b^{'}$ of b and wins the game if $b^{'} = b$.

Definition 1. *Our scheme is selectively secure under chosen plaintext attacks if, in the above security game, all polynomial-time adversaries $\mathcal{A}$ can succeed with at most a negligible advantage.*

IV. Constructions

A. Discussion and Overview

Discussion. Yang et al. [1] first introduced policy updating via the PolicyCompare algorithm, which classifies rows of the updated LSSS matrix into :$I_{1,\mathbb{M}'}$, rows that can directly reuse elements from the previous matrix; $I_{2,\mathbb{M}'}$, rows that correspond to existing attributes but require new elements; and $I_{3,\mathbb{M}'}$, entirely new attributes that necessitate the addition of fresh rows. Specifically, the size of the update keys grows linearly with the total number of rows. In complex updates involving multiple attribute modifications or changes to threshold gates, even small policy changes can cause large and redundant expansions of the index sets.

To address this, we propose a policy updating algorithm based on access trees, which provide a more efficient structure for modifying attributes or gates. Since basic tree-based ABE lacks update support, we extend the threshold ABE framework [10] to handle trees with multiple threshold gates.

Overview. Our scheme extends the basic threshold access policy scheme in [10] to support access trees. We first split a whole access tree $\mathcal{T}$ into K access subtrees $\mathcal{T}_j$ according to K non-leaf nodes, each ciphertext $ct_{\mathcal{T}_j}$ of a subtree $\mathcal{T}_j$ corresponding to the original threshold ciphertext in the scheme of [10]. We then set the different random number r in the private key sk_A with different attribute set A in order to prevent attribute collusion.

Suppose for each subtree $\mathcal{T}_j$ with leaf nodes set L_j and non-leaf nodes set N_j. First case: if $|L_j \cap A|$ is not less than the threshold t_j of $\mathcal{T}_j$, i.e. $\mathcal{T}_j(A) = 1$. Second case: if L_j is not enough to satisfy t_j and need non-leaf nodes of N_j (which are root nodes of subtrees that satisfy the attribute set A), assuming a set $S_j = \{s_k\} \subseteq N_j$, thus existing $|L_j \cap A| + |S_j| = t_j$. It also can be presented $\mathcal{T}_j(A) = 1$.

In the second case, a data user need to use leaf nodes L_j and satisfying subtrees S_j to meet the threshold t_j of $\mathcal{T}_j$. Specifically, each satisfying subtrees $s_k \in S_j$ can decrease the threshold value t_j to $t_j - |S_j|$ in $ct_{\mathcal{T}_j}$. Finally, L_j and S_j are enough to satisfy $\mathcal{T}_j$. This setting can extend for each subtree $\mathcal{T}_j$ and finally make up the access tree $\mathcal{T}$.

Tab. I. Notations used in our scheme

Notation	Description
A	Attribute set in the private key.
$\mathcal{T}$	Access tree in the ciphertext.
K	The number of subtrees in access tree $\mathcal{T}$.
$\mathcal{T}_j$	Access subtree in $\mathcal{T}$.
L_j	Leaf node set in subtree $\mathcal{T}_j$.
N_j	Non-leaf node set in subtree $\mathcal{T}_j$.
S_j	Root node set of satisfying subtrees of $\mathcal{T}_j$.
t_j	Threshold value in subtree $\mathcal{T}_j$.
s_j	Root node in subtree $\mathcal{T}_j$.
$\mathbb{E}_j$	Random numbers used in ciphertext $ct_{\mathcal{T}_j}$.
$ct_{\mathcal{T}_j}$	Ciphertext for each subtree $\mathcal{T}_j$ in access tree $\mathcal{T}$.
$\mathcal{T}(A) = 1$	A satisfies the access tree $\mathcal{T}$.
$\mathcal{T}_j(A) = 1$	A satisfies the subtree $\mathcal{T}_j$.
$\mathcal{T}_j(A) = 0$	A not satisfies the subtree $\mathcal{T}_j$.

B. Basic construction

Setup$(\lambda, U) \rightarrow (mpk, msk)$. The setup algorithm takes as input the security parameter λ and the attribute universe $U = \{att_1, att_2, \ldots, att_l\}$. We define $\rho(att_i) = i$ in the system. The setup algorithm also chooses other l default attributes $U' = \{att_{l+1}, \ldots, att_{2l}\} = \{l+1, \ldots, 2l\}$. It randomly chooses $g_2, v, w, h_0, h_1, \ldots, h_{2l} \in \mathbb{G}, \alpha, \beta \in \mathbb{Z}_p$ and sets $g_1 = g^\alpha, e(g_1, g_2) = e(v, w)$. The setup algorithm outputs as follows.

$$mpk = (g, g_2, v^\beta, \{h_i\}_{i=0,\ldots,2l}, e(g_1, g_2)), msk = (\alpha, w^{\frac{1}{\beta}}).$$

KeyGen$(mpk, msk, A) \rightarrow sk_A$. The key generation algorithm takes as input master key pair (mpk, msk) and the attribute set A. It randomly chooses $r \in \mathbb{Z}_p$ and a $l-1$ degree polynomial $q(.)$ with $q(0) = \alpha$. The keygen algorithm returns the private key $sk_A = \{w^{\frac{r-1}{\beta}}, sk_t\}_{t \in A \cup U'}$, and $sk_t =$

$$((g_2 h_0 h_t)^{rq(t)}, g^{rq(t)}, h_1^{rq(t)}, \ldots, h_{t-1}^{rq(t)}, h_{t+1}^{rq(t)}, \ldots, h_{2l}^{rq(t)}).$$

Enc$(mpk, \mathcal{T}, M) \rightarrow (ct_{\mathcal{T}}, \mathbb{E}_{\mathcal{T}})$. The encryption algorithm takes as input mpk, the access tree $\mathcal{T}$ and the message M. Firstly, we set the number of non-leaf nodes in access tree $\mathcal{T}$ is K. The encryption algorithm splits the access tree $\mathcal{T}$ into K subtrees $\mathcal{T}_j, j \in \{1, \ldots, K\}$ according to K non-leaf nodes. It also randomly chooses $s_j \in \mathbb{Z}_p$ for each subtrees $\mathcal{T}_j$. For simplicity, we define s_j is the root node of subtree $\mathcal{T}_j$. Specially, s_r is the root node of $\mathcal{T}$.
For each subtree $\mathcal{T}_j$, we make L_j and N_j are leaf nodes set and non-leaf nodes set, respectively. We define the corresponding threshold value t_j of the root node s_j in subtree $\mathcal{T}_j$.
The encryption algorithm to encrypt a message M as follows: For each subtree $\mathcal{T}_j$ with sets L_j, N_j and threshold t_j:

- If $L_j \neq \emptyset$ in the subtree $\mathcal{T}_j$, the encryption algorithm picks a default attribute subset $\Omega_j \subseteq U'$ with the first $l - t_j$ elements, $\Omega_j = \{l+1, l+2, \ldots, 2l - t_j\}$. The encryption algorithm computes the ciphertext $ct_{\mathcal{T}_j}$ as follows: $ct_{\mathcal{T}_{j,1}} = (h_0 \prod_{i \in L_j \cup \Omega_j} h_i)^{s_j}, ct_{\mathcal{T}_{j,2}} = g^{s_j}$.
- If $N_j \neq \emptyset$ in the subtree $\mathcal{T}_j$, for each non-leaf node $s_k \in N_j, k \neq j$, the encryption algorithm computes the ciphertext $ct_{\mathcal{T}_j, i+2} = h_{2l-t_j+i}^{s_j} g_2^{s_k}$, $i \in \{1, \ldots, |N_j|\}$ and add them to $ct_{\mathcal{T}_j}$.

The encryption algorithm stores the ramdom numbers $\mathbb{E}_{\mathcal{T}} = \{s_j\}, j \in \{1, \ldots, K\}$ locally. Specially, for subtree $\mathcal{T}_r$ with root node s_r of $\mathcal{T}$, the ciphertext $ct_{\mathcal{T}_r} = (ct_{\mathcal{T}_{r,1}}, ct_{\mathcal{T}_{r,2}}, ct_{\mathcal{T}_{r,3}} = (v^\beta)^{s_r}, \{ct_{\mathcal{T}_r, i+3}\}, i \in \{1, \ldots, |N_r|\})$. The encryption algorithm returns the ciphertext $ct_{\mathcal{T}}$ as follows.

$$ct_{\mathcal{T}} = (M \cdot e(g_1, g_2)^{s_r}, ct_{\mathcal{T}_j}, j \in \{1, \ldots, K\})$$

Dec$(mpk, ct_{\mathcal{T}}, sk_A) \rightarrow M$. The decryption algorithm takes as input the master public key mpk, private key sk_A and the

ciphertext $ct_{\mathcal{T}}$. The decryption algorithm decrypt as following for each subtree $\mathcal{T}_j$ in access tree $\mathcal{T}$:

- If $N_j = \emptyset$ in subtree $\mathcal{T}_j$, that means children nodes in subtree $\mathcal{T}_j$ are all leaf nodes L_j, the attribute set A in private key satisfies the threshold access structure, (i.e., there exists a subset $A_j^{'} \subseteq A \cap L_j$ with $|A_j^{'}| = t_j$) can use the private key $\{sk_t\}_{t \in A_j^{'} \cup U'}$ to decrypt as follows:
  (1) The decryption algorithm picks a default attribute subset $\Omega_j \subseteq U^{'}$ with the first $l - t_j$ elements, $\Omega_j = \{l+1, l+2, \ldots, 2l - t_j\}$.
  (2) After that, the decryption algorithm computes:

$$P_{j,1} = \prod_{i \in A_j^{'} \cup \Omega_j} ((g_2 h_0 \prod_{t \in L_j \cup \Omega_j} h_t)^{rq(i)})^{\Delta_{i, A_j^{'} \cup \Omega_j}(0)}$$
$$= (g_2 h_0 \prod_{t \in L_j \cup \Omega_j} h_t)^{r\alpha},$$
$$P_{j,2} = \prod_{i \in A_j^{'} \cup \Omega_j} (g^{rq(i)})^{\Delta_{i, A_j^{'} \cup \Omega_j}(0)} = g^{r\alpha},$$

  (3) Finally, the decryption algorithm computes:

$$D_{j,1} = \frac{e(P_{j,1}, g^{s_j})}{e((h_0 \prod_{i \in L_j \cup \Omega_j} h_i)^{s_j}, P_{j,2})} = e(g_2^{r\alpha}, g^{s_j}).$$

- If $N_j \neq \emptyset$ in subtree $\mathcal{T}_j$, that means children nodes in subtree $\mathcal{T}_j$ include non-leaf nodes. The attribute set A satisfies the threshold access structure of subtree $\mathcal{T}_j$, (i.e., there are two subsets $A_j^{'} \subseteq A \cap L_j$ and $S_j = \{s_k\} \subseteq N_j$, satisfying $|A_j^{'} \cup S_j| = t_j$). S_j is presented root nodes of attribute set A in private keys satisfying subtrees $\mathcal{T}_k$, for each $s_k \in S_j$, i.e., holding $D_{k,1} = e(g_2^{r\alpha}, g^{s_k})$.
  (1) The decryption algorithm picks a default attribute subset $\Omega_j \subseteq U^{'}$ with the first $l - t_j$ elements, $\Omega_j = \{l+1, l+2, \ldots, 2l - t_j\}$.
  (2) The decryption algorithm computes $D_{j,3,i}$ for $i \in \{1, \ldots, |S_j|\}$ with $s_k \in S_j$,

$$D_{j,3,i} = e(h_{2l-t_j+i}^{s_j} g_2^{s_k}, P_{k,2}) / D_{k,1} = e(h_{2l-t_j+i}^{s_j}, g^{r\alpha})$$

  (3) After that, the decryption algorithm computes as following: $P_{j,1}$

$$= \prod_{i \in A_j^{'} \cup S_j \cup \Omega_j} ((g_2 h_0 \prod_{t \in L_j \cup S_j \cup \Omega_j} h_t)^{rq(i)})^{\Delta_{i, A_j^{'} \cup S_j \cup \Omega_j}(0)},$$
$$= (g_2 h_0 \prod_{t \in L_j \cup S_j \cup \Omega_j} h_t)^{r\alpha}$$

$$P_{j,2} = \prod_{i \in A_j^{'} \cup S_j \cup \Omega_j} (g^{rq(i)})^{\Delta_{i, A_j^{'} \cup S_j \cup \Omega_j}(0)} = g^{r\alpha},$$

  Then the decryption algorithm computes $D_{j,1}$ as following:

$$\frac{e(P_{j,1}, g^{s_j})}{e((h_0 \prod_{i \in L_j \cup \Omega_j} h_i)^{s_j}, P_{j,2}) \cdot \prod_{i=1}^{|S_j|} D_{j,3,i}} = e(g_2^{r\alpha}, g^{s_j})$$

The decryption algorithm computes each satisfied subtrees $\mathcal{T}_j$ until $\mathcal{T}_r$ with $D_{r,1}$. At last, the algorithm now decrypts message by computing $M = M \cdot e(g_1, g_2)^{s_r} \cdot (D_{r,2}/D_{r,1})$.

Correctness: For attribute set A satisfied the access tree $\mathcal{T}$, each satisfied subtree $\mathcal{T}_j$ can compute $D_{j,1} = e(g_2^{r\alpha}, g^{s_j})$. Then for subtree $\mathcal{T}_r$ with the root node s_r, $D_{r,1} = e(g_2^{r\alpha}, g^{s_r})$, $D_{r,2} = e((v^{\beta})^{s_r}, w^{\frac{r-1}{\beta}}) = e(g_2^{r\alpha}, g^{s_r}) e(g_2^{\alpha}, g^{-s_r})$, and then $M = M \cdot e(g_1, g_2)^{s_r} \cdot (D_{r,2}/D_{r,1})$.

Example: For simplicity, we provide an example in the Fig. 2. The access policy in $ct_{\mathcal{T}}$ is $(att_A \ OR \ (2, (att_D, (att_B \ AND \ att_C), (2, (att_E, att_F, att_G)))))$ and the attribute set is $A = \{att_C, att_D, att_E, att_F\}$ in sk_A. Specially, the following is the decryption process of $\mathcal{T}_3, \mathcal{T}_1, \mathcal{T}_r$ in $\mathcal{T}$.

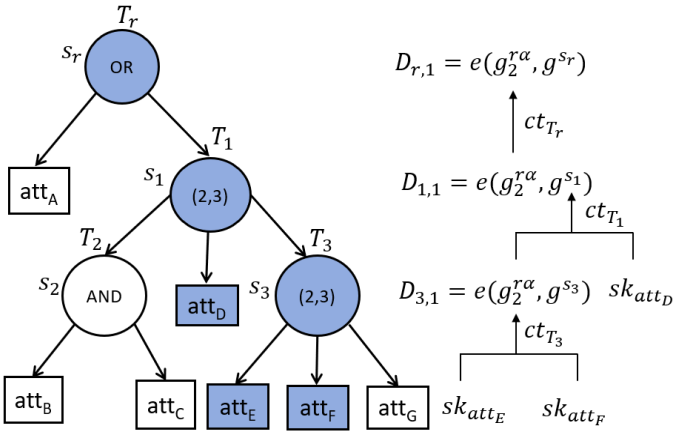


Fig. 2. The decryption of access tree $\mathcal{T}$

According to the nodes sets of each subtrees in Tab.2, firstly, it is clear that $\mathcal{T}_3(A) = 1$, since $|L_3 \cap A| = t_3$. Consequently, when $N_3 = \emptyset$, we can directly obtain the $D_{3,1} = e(g_2^{r\alpha}, g^{s_3})$ from sk_{att_E}, sk_{att_F} and $ct_{\mathcal{T}_3}$. The subtree $\mathcal{T}_3$ thus becomes the satisfying non-leaf node for $\mathcal{T}_1$, i.e., $S_1 = \{s_3\}$. Next, for $\mathcal{T}_1$, only one attribute att_D in A of sk_A satisfies the access policy. Therefore, the non-leaf node s_3 is required to help achieve the threshold t_1 in $ct_{\mathcal{T}_1}$, i.e. $|L_1 \cap A| + |S_1| = t_1$. As a result, $\mathcal{T}_1(A) = 1$, yielding $D_{1,1} = e(g_2^{r\alpha}, g^{s_1})$. Therefore, $\mathcal{T}_1$ becomes the satisfying non-leaf node for $\mathcal{T}_r$, i.e., $S_r = \{s_1\}$. Finally, based on the satisfying non-leaf node s_1 and $ct_{\mathcal{T}_r}$, we can compute $D_{r,1}$ for the root node s_r of the access tree $\mathcal{T}$. Thus $\mathcal{T}_r(A) = \mathcal{T}(A) = 1$, which indicates that the entire access tree is satisfied and the decryption process can successfully proceed.

Tab. II. Nodes set of the access tree $\mathcal{T}$

Subtree	Leaf nodes	Non-leaf nodes	Threshold	Satifying non-leaf nodes
$\mathcal{T}_r$	$L_r = \{att_A\}$	$N_r = \{s_1\}$	$t_r = 1$	$S_r = \{s_1\}$
$\mathcal{T}_1$	$L_1 = \{att_D\}$	$N_1 = \{s_2, s_3\}$	$t_1 = 2$	$S_1 = \{s_3\}$
$\mathcal{T}_2$	$L_2 = \{att_B, att_C\}$	$N_2 = \emptyset$	$t_2 = 2$	$S_2 = \emptyset$
$\mathcal{T}_3$	$L_3 = \{att_E, att_F, att_G\}$	$N_3 = \emptyset$	$t_3 = 2$	$S_3 = \emptyset$

C. Policy updating

The update key generation algorithm, UPKeyGen, and the ciphertext updating algorithm, CTUPdate, are related to the structural between the original access tree $\mathcal{T}$ and the new access tree $\mathcal{T}'$. In this context, we specifically focus on policy updating for a single gate, with no constraints on the number of updated attributes or different gate types.

On the one hand, different updated attributes for a threshold gate are compacted in one updating component thus the size of update keys is constant for a gate updating. On the other hand, we provide policy updating but without classification of gates because the differences between AND, OR and threshold gates are only in the setting of threshold value t_j of Ω_j in the subtree $\mathcal{T}_j$. Therefore we only classify two different types of updating operation, Updating attributes to an existing gate (Attributes2Existing) and Updating attributes to a new gate (Attributes2New).

In Attributes2Existing case, the update key $tk_{\mathcal{T}\to\mathcal{T}'}$ consists of $tk_{\mathcal{T}'_{j,1}}$ used to update access policy and one random number t'_j. In Attributes2New case, besides $tk_{\mathcal{T}'_{j,1}}$ and t'_j, the update key $tk_{\mathcal{T}\to\mathcal{T}'}$ includes a new subtree ciphertext $ct_{\mathcal{T}_t}$ and a new leaf-node component $tk_{\mathcal{T}'_{j,(|N_j|+1)+2}}$ for $\mathcal{T}_j$. The update key still keeps constant size even including a new subtree ciphertext $ct_{\mathcal{T}_t}$ due to only two components in the ciphertext for a subtree $\mathcal{T}_t$.

UPKeyGen$(mpk, \mathbb{E}_{\mathcal{T}}, \mathcal{T}, \mathcal{T}') \to tk_{\mathcal{T}\to\mathcal{T}'}$. The update key generation algorithm takes as input mpk, random numbers $\mathbb{E}_{\mathcal{T}}$ used in $ct_{\mathcal{T}}$, the original access tree $\mathcal{T}$ and the updated access tree $\mathcal{T}'$. Let T'_j, Ω'_j are new attributes sets in the updated subtree $\mathcal{T}'_j$. The update algorithm first randomly chooses $t'_j \in \mathbb{Z}_p$ and then generates update keys as follows.

- **Updating attributes to an existing gate.** The updating operation involves converting attributes in the subtree $\mathcal{T}_j$ to an existing AND or OR or threshold gate with adding or removing attributes sets T'_j and Ω'_j. As shown in Fig. 3, these new attributes (att_D, att_E) play sibling nodes as existing attributes (att_B, att_C) in the new access policy. Therefore, we can modify the previous ciphertext $ct_{\mathcal{T}_j}$ by using update key $tk_{\mathcal{T}\to\mathcal{T}'} = (tk_{\mathcal{T}'_{j,1}}, t'_j)$ as following:

$$tk_{\mathcal{T}'_{j,1}} = (h_0 \prod_{t\in T'_j\cup\Omega'_j} h_t)^{s_j} (h_0 \prod_{i\in L_j\cup\Omega_j} h_i)^{-s_j t'_j}$$

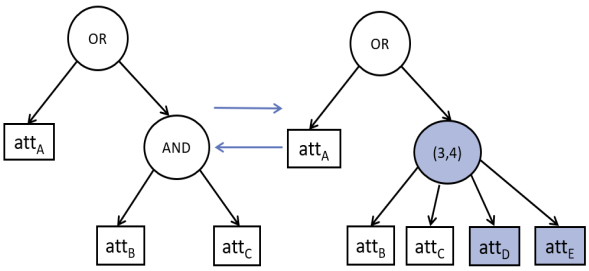


Fig. 3. Updating attributes to an existing gate.

- **Updating attributes to a new gate.** The updating operation involves converting attributes in the subtree $\mathcal{T}_j$ to a new AND or OR or threshold gate with adding or removing attributes sets T'_j and Ω'_j. In the new access policy, as shown in the Fig. 4, new attributes att_D, att_E are sibling nodes as the existing attribute att_A and the new $(2,3)$ threshold gate takes the place of attribute att_A denoted as $s_t \in \mathbb{Z}_p$. The new ciphertext $ct_{\mathcal{T}_t}$ is generated for the subtree $(2,(att_A, att_D, att_E))$. Therefore we can modify the previous ciphertext $ct_{\mathcal{T}_j}$ by using update key $tk_{\mathcal{T}\to\mathcal{T}'} = (tk_{\mathcal{T}'_{j,1}}, tk_{\mathcal{T}'_{j,(|N_j|+1)+2}}, ct_{\mathcal{T}_t}, t'_j)$ as following:

$$tk_{\mathcal{T}'_{j,1}} = (h_0 \prod_{t\in T'_j\cup\Omega'_j} h_t)^{s_j} (h_0 \prod_{i\in L_j\cup\Omega_j} h_i)^{-s_j t'_j},$$
$$tk_{\mathcal{T}'_j,(|N_j|+1)+2} = h_{2l-t_j+|N_j|+1}^{s_j} g_2^{s_t}$$

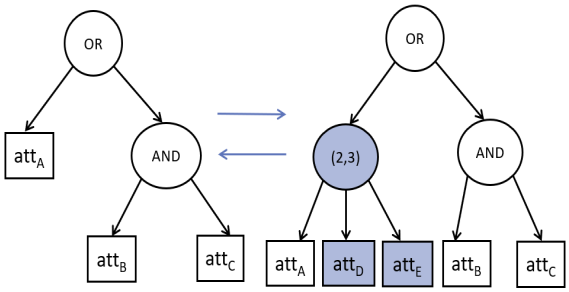


Fig. 4. Updating attributes to a new gate.

One specific option for updating attributes with a new gate is to extend the original access policy by attaching a new policy to the root node. This operation transforms the access tree $\mathcal{T}$ into a new AND, OR, or threshold gate with added or removed attributes, denoted as RootEtOR, RootEtAND, and RootEtThreshold, respectively in [7]. For example, in the RootEtAND case illustrated in Fig. 5, the new AND gate replaces the original root node, and the data owner selects a new root secret $s_{r'} \in \mathbb{Z}_p$. The previous ciphertext $ct_{\mathcal{T}}$ can then be modified using the update key $tk_{\mathcal{T}\to\mathcal{T}'} = (e(g_1, g_2)^{s_{r'}-s_r}, ct_{\mathcal{T}_{r'}})$.

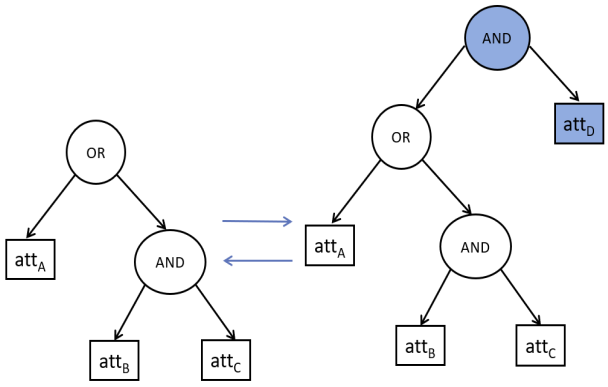


Fig. 5. Updating attributes to the root gate.

CTUpdate$(mpk, ct_{\mathcal{T}}, tk_{\mathcal{T}\to\mathcal{T}'}) \to ct_{\mathcal{T}'}$. The ciphertext update algorithm takes as input the master public key mpk, the ciphertext $ct_{\mathcal{T}}$ and the update key $tk_{\mathcal{T}\to\mathcal{T}'}$. If the ciphertext updating is updating attributes to an existing gate, the updated subtrees' ciphertext as follows,

$$ct_{\mathcal{T}_j'} = (ct_{\mathcal{T}_{j,1}}^{t_j'} \cdot tk_{\mathcal{T}_{j,1}'}, ct_{\mathcal{T}_{j,2}}, \{ct_{\mathcal{T}_{j,i+2}}\}, i \in \{1, \ldots, |N_j|\})$$

If the ciphertext updating is updating attributes to a new gate, update keys in this case include the new ciphertext component $ct_{\mathcal{T}_t}$, the updated subtrees' ciphertext $ct_{\mathcal{T}_j'}$ as following,

$$(ct_{\mathcal{T}_{j,1}}^{t_j'} \cdot tk_{\mathcal{T}_{j,1}'}, ct_{\mathcal{T}_{j,2}}, \{ct_{\mathcal{T}_{j,i+2}}, tk_{\mathcal{T}_j', (|N_j|+1)+2}\}, i \in \{1, \ldots, |N_j|\})$$

If the ciphertext updating is RootEtOR or RootEtAND or RootEtThreshold, update keys in this case specially update $M \cdot e(g_1, g_2)^{s_r} \cdot e(g_1, g_2)^{s_{r'}-s_r} = M \cdot e(g_1, g_2)^{s_{r'}}$ and then add $ct_{\mathcal{T}_r'}$ in the new ciphertext $ct_{\mathcal{T}'}$.

V. Security Proof

A. Complexity Assumption

The security of our scheme is based on the decisional q-Bilinear Diffle-Hellman Exponent (BDHE) problem in [31], which is defined as follows: denote $\overrightarrow{y} = (g_1, g_2, \ldots, g_q, g_{q+2}, \ldots, g_{2q})$ where $g_i = g^{\alpha^i}$ for some unknown $\alpha \in \mathbb{Z}_p$. An algorithm $\mathcal{B}$ that outputs $b \in \{0, 1\}$ has advantage ϵ in solving the decisional $q - BDHE$ problem if $|\Pr[\mathcal{B}(g, h, \overrightarrow{y}, Z = e(g^{\alpha^{q+1}}, h)) = 1] - \Pr[\mathcal{B}(g, h, \overrightarrow{y}, Z) = 1]| \geq \epsilon$. where the probability is over the random choice of generators $g, h \in \mathbb{G}$, the random choice of $\alpha \in \mathbb{Z}_p$, the random choice of $Z \in \mathbb{G}_T$.

B. Proof

Theorem 1. *If the q-BDHE problem holds then no polynomial adversary can selectively break our scheme with a challenge access tree $\mathcal{T}^*$.*

Proof. Suppose there exists an adversary $\mathcal{A}$ that undermines the proposed scheme with an advantage $\mathsf{Adv}^{\mathsf{IND\text{-}sCPA}}_{\mathsf{PU\text{-}CP\text{-}ABE}}(\mathcal{A}) \geq \epsilon$. We construct a simulator $\mathcal{B}$ with an advantage ϵ in solving the q-BDHE assumption. Given an instance of the underlying assumption, $\mathcal{B}$ simulates the role of the challenger C in the security game and interacts with the adversary $\mathcal{A}$ as described below.

Initialization. Selective game begins with $\mathcal{A}$ first providing a challenge access tree $\mathcal{T}^*$ that it intends to attack. For simplicity, we let there are K^* challenge subtrees $\mathcal{T}_j^*$ in $\mathcal{T}^*$ and corresponding challenge threshold values t_j^*. We assume that each attribute belonging to U only appears once in $\mathcal{T}^*$.

Setup. $\mathcal{B}$ first defines the universe of attributes as $U = \{1, \ldots, l\}$ and chooses l default attributes $U' = \{l+1, \ldots, 2l\}$. We let $2l + 1 \ll q$ in the system. For subtrees $\mathcal{T}_j^*, j \in \{1, \ldots, K^*\}$ in challenge access tree $\mathcal{T}^*$, $\mathcal{B}$ sets attributes sets $\Omega_j^* = \{l+1, \ldots, 2l - t_j^*\}$ according to each threshold value t_j^* in $\mathcal{T}_j^*$. Specially, $\mathcal{B}$ sets the attribute set $\Lambda = \{2l - t_1^* + i\}$ for $i \in \{1, \ldots, |N_1^*|\}$.

Then $\mathcal{B}$ randomly chooses $\gamma_t, \theta_t \in \mathbb{Z}_p$, $t \in \{1, \ldots, 2l\}$ and $\gamma_0^{(j)}, \tau_j$ from $\mathbb{Z}_p$, $j \in \{1, \ldots, K^*\}$. $\mathcal{B}$ first generates $g_t = g^{\gamma_t} g^{a^{q-t+1}}, t \in \{1, \ldots, 2l\}$ and then produces $h_0^{(j)} = g^{\gamma_0^{(j)}} \prod_{i \in (L_j^* \cup \Omega_j^*)} g_i^{-\tau_j}, j \in \{1, \ldots, K^*\}$. Finally, $\mathcal{B}$ makes $h_0 = \prod_{j=1}^{K^*} h_0^{(j)}$.

Furthermore, $\mathcal{B}$ generates attribute sets $\Phi_k = \{\tau_j | k \in (L_j^* \cup \Omega_j^*)\}$ for $k \in (U \cup U')$. For $t \in \{1, \ldots, 2l\}$, $\mathcal{B}$ generates $h_t = g^{\theta_t}$ when $t \in \Lambda$ and $h_t = g_t^{\sum_{j=1}^{|\Phi_t|} \tau_j}$ when $t \notin \Lambda$.

Finally, $\mathcal{B}$ randomly chooses $\beta, b_w \in \mathbb{Z}_p$, and implicitly sets $\alpha = a^q$ by letting $g_1 = g^{a^q}$, $g_2 = g^a$. At last, $\mathcal{B}$ sets $L_1^* \cup \Omega_1^* = \cup_{j=1}^{K^*}(L_j^* \cup \Omega_j^*)$. $\mathcal{B}$ then generates $v = g^{\frac{1}{b_w}}$ and $w = g^{a^{q+1} b_w}$ with $e(g_1, g_2) = e(v, w)$ and provides $\mathcal{A}$ the public parameters $(g, g_2, v^\beta, Z, h_0, h_1, \ldots, h_{2l}), Z = e(g_1, g_2) = e(g^{a^q}, g^a)$.

Query phase 1. During this phase, $\mathcal{B}$ responds to private key queries from $\mathcal{A}$. Assuming that $\mathcal{A}$ submits a maximum of q_1 queries for the private key of attribute A, with the constraint that A does not satisfy the challenge access tree $\mathcal{T}^*$, specially, $\mathcal{T}_j^*(A) = 0$, $\mathcal{B}$ begins by selecting the unsatisfied subtrees $\mathcal{T}_j^*$ in the challenge access tree $\mathcal{T}^*$. Subsequently, $\mathcal{B}$ defines three sets, T_j, T_j', T_j'', in the following manner. For each unsatisfied subtree $\mathcal{T}_j^*$, $T_j = (A \cap L_j^*) \cup \Omega_j^*$, $T_j \subseteq T_j' \subseteq (L_j^* \cup \Omega_j^*)$ and $|T_j'| = l - 1$, $T_j'' = T_j' \cup \{0\}$. For each attribute $i \in A \cup U'$, $\mathcal{B}$ randomly chooses an $l - 1$ degree polynomial $q(.)$ such that $q(0) = a^{q-1}$. The private key sk_i is computed as follows for each attribute $i \in A \cup U'$:

(1) For i that belong to satisfied subtrees or T_j' of unsatisfied subtree, i.e., $i \in (L_j^* \cup \Omega_j^*)$. $\mathcal{B}$ randomly chooses $\widehat{r}, t_i \in \mathbb{Z}_p$ and sets $r = \widehat{r} + a + 1, q(i) = a^{i-1} + t_i$, then computes $(w^{\frac{r-1}{\beta}} = (w^{\frac{\widehat{r}}{\beta}} \cdot g^{\frac{a^{q+2} b_w}{\beta}})$ and $sk_i = (g_2 h_0 h_i)^{rq(i)}, g^{rq(i)}, h_1^{rq(i)}, \ldots, h_{i-1}^{rq(i)}, h_{i+1}^{rq(i)}, \ldots, h_{2l}^{rq(i)})$,

$$(g_2 h_0 h_i)^{rq(i)} = (g_2 \cdot \prod_{j=1}^{K^*} g^{\gamma_0^{(j)}} \prod_{i \in (L_j^* \cup \Omega_j^*), i \neq j} g_i^{-\tau_j})^{(\widehat{r}+a+1)(a^{i-1}+t_i)}$$

(2) For $i \notin T_j'$, i.e., $i \notin (L_j^* \cup \Omega_j^*)$. C randomly selects $r = \widetilde{r} + a + 1, \widetilde{r} \in \mathbb{Z}_p$ and sets $q(i) = \Delta_{0,T_j''}(i)(q(0) - a^{i-1}) + \sum_{t \in T_j'}(i) q(t)$. C computes the private key $sk_i = (w^{\frac{r-1}{\beta}}, (g_2 h_0 h_i)^{rq(i)}, g^{rq(i)}, h_1^{rq(i)}, \ldots, h_{i-1}^{rq(i)}, h_{i+1}^{rq(i)}, \ldots, h_{2l}^{rq(i)})$ and specially $(g_2 h_0 h_i)^{rq(i)} =$

$$(g_2 \prod_{j=1}^{K^*} g^{\gamma_0^{(j)}} \prod_{i \in (L_j^* \cup \Omega_j^*)} g_i^{-\tau_j})^{(\widetilde{r}+a+1)(\Delta_{0,T_j''}(i)(q(0)-a^{i-1}) + \sum_{t \in T_j'}(i) q(t))}$$

$$= \Gamma \cdot (g^a)^{a(\Delta_{0,T_j''}(i) q(0))} \cdot h_i^{a(-\Delta_{0,T_j''}(i) a^{i-1})}$$

Note that when $i \in (L_j^* \cup \Omega_j^*)$ and $i \in \Lambda$, $\mathcal{B}$ sets $q(i) = t_i$ and generates sk_i same as in the scheme.

Challenge. $\mathcal{A}$ submits two challenge messages, denoted as M_0 and M_1 both of equal length to $\mathcal{B}$. Subsequently, $\mathcal{B}$ randomly flips a fair binary coin, denoted as b, and provides an encryption of the selected message M_b in response. The ciphertext $ct_{\mathcal{T}^*}$ is output as follow: $(M_b \cdot Z, ct_{\mathcal{T}_j^*}, j \in \{1, \ldots, K^*\})$, and specially, the ciphertext for the root s_r is $ct_{\mathcal{T}_1^*} =$

$$(h^{\sum_{j=1}^{K^*} \gamma_0^{(j)}}, h, h^{\frac{\beta}{b_w}}, \{h^{\theta_i} g_2^{s_k}\}, i \in \{1, \ldots, |N_1^*|\})$$

Query phase 2. $\mathcal{A}$ continues to make private key queries as in phase 1. $\mathcal{A}$ can make update key queries by submitting access trees $(\mathcal{T}_j, \mathcal{T}_j')$ under the restriction that query original access tree $\mathcal{T}_j$ cannot be the same as $\mathcal{T}^*$. In addition, $\mathcal{A}$ can also make ciphertext updating queries expect for $ct_{\mathcal{T}^*}$.

Guess. After completing phase 2 of the query process, the adversary $\mathcal{A}$ outputs a guess b' for the challenge bit b. If $b' = b$, then $\mathcal{B}$ outputs 0, indicating that the challenge element Z is of the form $e(g^{\alpha^{q+1}}, h)$. Otherwise, it outputs 1, asserting that Z is a random element in $\mathbb{G}_T$.

When $\mathcal{A}$ can break the encryption and then $\mathcal{A}$ played the proper security game, because of $M_b \cdot Z$ under $Z = e(g^{\alpha^{q+1}}, h)$. The view of $\mathcal{A}$ is identical to its view in a real attack game, and therefore $\mathcal{A}$ satisfies $\Pr[b' = b] - \frac{1}{2} \geq \epsilon$. When Z is random from $\mathbb{G}_T$ then $\Pr[b' = b] = \Pr[\mathcal{B}(g, h, \overrightarrow{y}, Z = e(g^{\alpha^{q+1}}, h)) = 1] = \frac{1}{2}$. We have that $\Pr[\mathcal{B}(g, a, s, \overrightarrow{b}, Z = e(g^{\alpha^{q+1}}, h)) = 1] = \frac{1}{2} + \mathsf{Adv}^{\text{IND-sCPA}}_{\text{PU-CP-ABE}}(\mathcal{A}) \geq \frac{1}{2} + \epsilon$. Therefore $\mathcal{B}$ has the advantage at least ϵ in solving the $q-$BDHE assumption. This concludes the proof of Theorem 1, thereby demonstrating the security of our proposed scheme under the defined model.

VI. Performance Analysis

A. Theoretical Analysis

In this section, we conduct a comprehensive comparison between our scheme with the most related works [1], [3], [4], [5]. Tab. 3 provides an empirical evaluation of performance, considering various updating types, including Converting an attribute to an OR gate (Att2OR), Converting an attribute to an AND gate (Att2AND), Attributes2Existing, and Attributes2New. The comparison encompasses metrics in ciphertext size and update key size.

Efficient ciphertext. We begin by demonstrating that our ciphertext size is much smaller than that of existing CP-ABE schemes. Tab. 3 reveals that the ciphertext sizes in [1], [3], [4] and [5] are linearity with n (the row of the access matrix $\mathbb{M}$) and $|Y|$ (the number of leaf nodes in the access tree), respectively. Different with them, our ciphertext size increases only linearly with the number of gates K (the number of non-leaf nodes in the access tree).

To enable a fair comparison, we evaluate ciphertext sizes based on a general threshold-gate access tree with K (t_i, n_i) gates. Converting this structure into an access matrix $(\mathbb{M}, \rho)$ yields a row count of $n = \sum_{i=1}^{K} n_i C_{n_i-1}^{t_i-1} - K$, while the total number of leaf nodes is $|Y| = \sum_{i=1}^{K} n_i$. The key advantage of our approach is evident: since the number of gates K is consistently smaller than both n and $|Y|$, our ciphertext size is substantially smaller than that of prior CP-ABE schemes. Furthermore, as illustrated in Fig. 6, which shows ciphertext sizes (in KB) under different K values with a 512-bit security parameter, our scheme consistently produces the smallest ciphertexts with minimal growth.

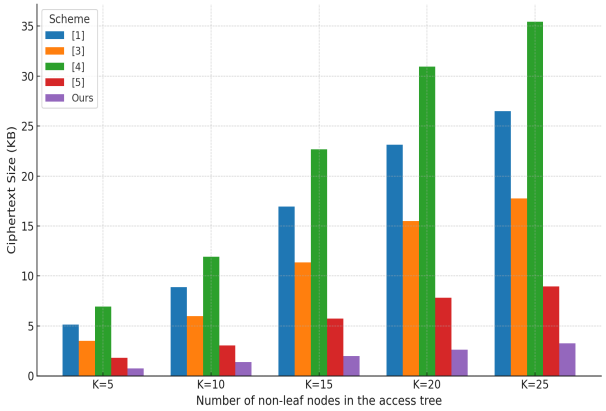


Fig. 6. Comparison of ciphertext size

Lightweight update keys. Our update key size is much smaller than in existing policy-updatable ABE schemes. As shown in Tab. 3, the schemes in [1], [3] grow linearly with the number of updated attributes when multiple attributes are modified within a gate. In [4], ciphertext components are uploaded instead of update keys, but the size still scales with the number of affected rows. In [5], data owners must submit re-encryption keys together with encrypted root keys and policies.

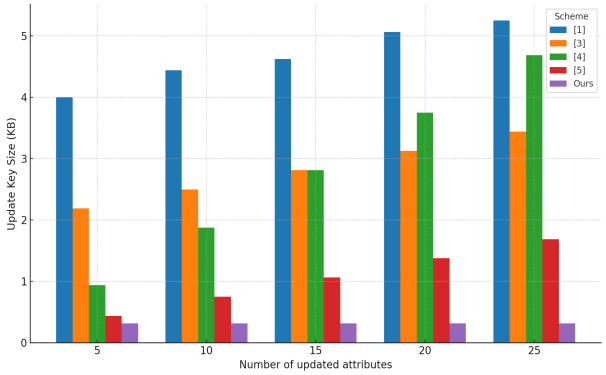


Fig. 7. Comparison of update key size

In contrast, our scheme generates update keys of constant size, regardless of the number of updated attributes or gate types. As shown in Fig. 7, which compares the update key sizes (in KB) under various numbers of updated attributes, our scheme consistently maintains the smallest update key size. This efficiency is particularly beneficial in scenarios involving

Tab. III. Performance comparison

	Ciphertext size	Attr2OR	Attr2AND	Attributes2Exisiting	Attributes2New
[1]	$3n\|G\| + \|G_T\|$	$3\|G\| + \|Z_p\| + \|G_T\|$	$3\|G\| + 2\|G_T\|$	$(2I_{1,\mathbb{M}'} + 2I_{2,\mathbb{M}'} + 3I_{3,\mathbb{M}'})\|G\| + \|Z_p\|$	
[3]	$(2n+1)\|G\| + \|G_T\|$	-	-	$I_1\|Z_p\| + 2I_2\|G\|$	
[4]	$(4n+2)\|G\| + \|G_T\|$	-	-	$3n^{'}\|G\|$	
[5]	$(\|Y\|+1)\|G\| + \|G_T\|$	-	-	$(\|Y\|+2)\|G\|$	
Ours	$(2K+1)\|G\| + \|G_T\|$	$\|Z_p\| + 4\|G\|$	$\|Z_p\| + 4\|G\|$	$\|Z_p\| + \|G\|$	$\|Z_p\| + 4\|G\|$

Tab. IV. Trade-off comparison

	Private key size	Access policy	Decryption overhead	Update key size
[7]	$3(\|S\|+1)\|G\|$	LSSS	$(2\|I\|+1)Exp + 6e$	$2(\|\mathcal{K}\|+1)\|G\| + 4\|Z_p\|$
[8]	$(\|S\|+3)\|G\|$	LSSS	$2\|I\|Exp + 2(\|I\|+1)e$	$(\|S\| + \|S^{'}\| + 6)\|G\|$
Ours	$(\|S\|+l)(2l+1)\|G\|$	Access Tree	$2K_1\|A^{'}_{\omega}\|Exp + 2K_1 e +$ $2K_2(\|A^{'}_{\omega}\| + \|S_k\|)Exp + (\|S_k\|+2)e$	$K^{'}(\|G\| + \|Z_p\|)$ / $K^{'}(4\|G\| + \|Z_p\|)$

frequent policy updates, significantly reducing the burden on data owners in both computation and transmission.

Tab. V. Notations used in our comparison

Notation	Description
$\|G\|$	Size of a prime-order group G.
$\|Z_p\|$	Size of a element in Z_p.
$\|G_T\|$	Size of group G_T.
$\|S\|$	Number of attribute set in private keys.
n	Number of the row in original access policy $\mathbb{M}$.
$n^{'}$	Number of the row in update access policy $\mathbb{M}^{'}$.
$I_{1/2/3,\mathbb{M}'}$	Number of index sets of different update keys in [1].
$I_{1/2}$	Nnumber of altered and inserted attributes in [3].
$\|Y\|$	Number of leaf nodes of the access tree in [5].
l	Number of attribute universal set.
$\|K\|$	Number of subtrees in access tree $\mathcal{T}$.
$\|I\|$	Number of attributes in private keys satisfying $\mathbb{M}$.
Exp	Exponential operation.
e	Pairing operation.
$\|\mathcal{K}\|$	Number of same attributes in $\mathbb{M}^{'}$ with in $\mathbb{M}$.
$\|S^{'}\|$	Number of the attributes satisfying $\mathbb{M}^{'}$.
K_1	Number of subtrees in access tree $\mathcal{T}$ with $N_j = \emptyset$.
$\|A^{'}_{\omega}\|$	Size of $\|A^{'}_{j} \cup \Omega_j\|$.
K_2	Number of subtrees in access tree $\mathcal{T}$ with $N_j \neq \emptyset$.
$\|S_k\|$	Number of root nodes that satifying $\mathbb{M}$.
$K^{'}$	Number of updated gates.

B. Experiment Analysis

In this section, we evaluate the time performance of our scheme, focusing on encryption and gate updates (OR, AND, and threshold), with respect to the number of attributes. Experiments were conducted using C++ on a Windows 10 (64-bit) system with an Intel Core i7-8700 CPU and 16GB RAM, leveraging GMP [32] and PBC [33]. The experiment involved a 160-bit elliptic curve group constructed on a curve defined by $y^2 = x^3 + x$ over a 512-bit field $\mathbb{F}_q$. We compare our scheme with [1], [5], excluding [3], [4] due to their structural similarity to [1].

Efficient ciphertext. Fig. 8(a) shows the encryption cost of our scheme compared with [1], [5] as the number of attributes increases. In [1], the cost grows rapidly due to the linear on the number of rows in the access matrix, which expands quickly during gate conversion. In [5], the cost scales linearly with the number of attributes, reflecting the number of leaf nodes in the access tree. By contrast, our ciphertext size grows linearly with the number of gates, resulting in a much lower encryption cost than both [1] and [5].

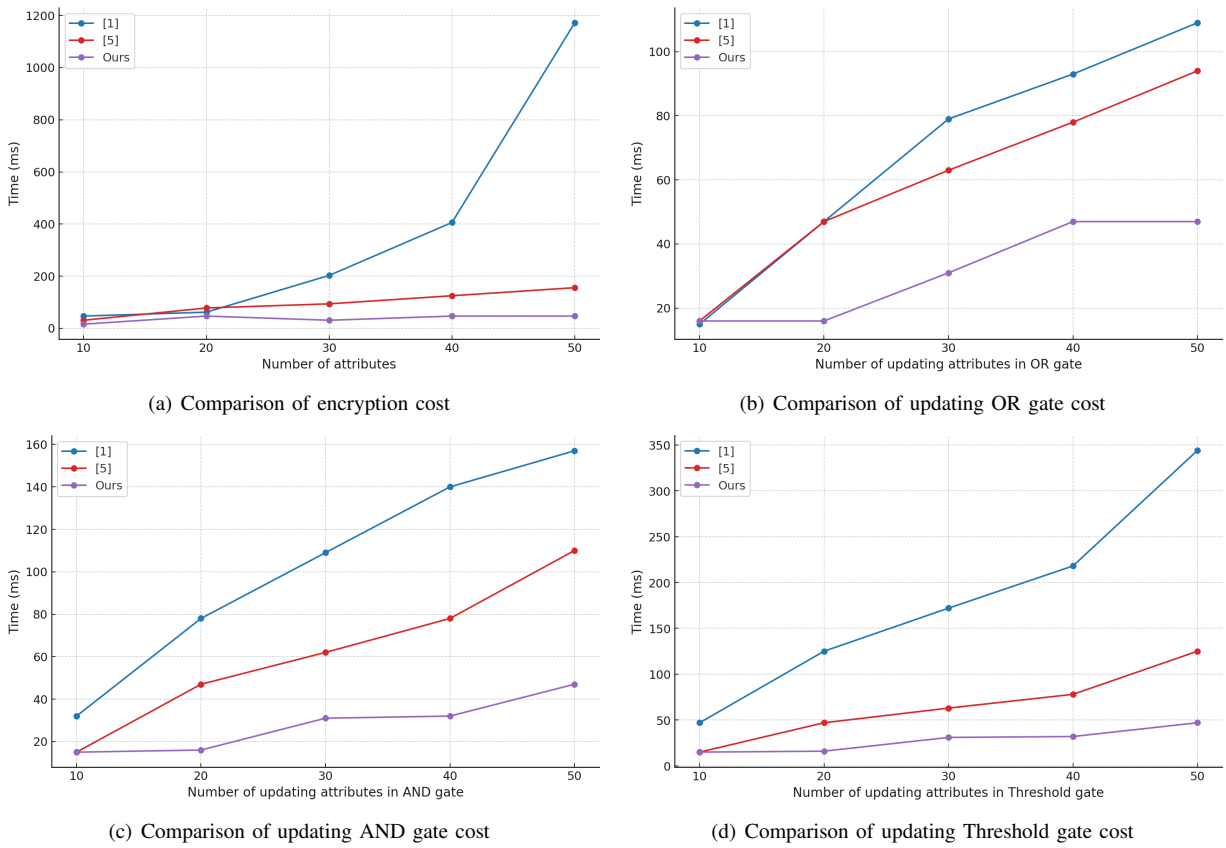
Lightweight update keys. Fig. 8(b)–(d) compare the computational costs of OR, AND, and threshold updates in [1], [5], and our scheme. While all schemes incur update key generation costs that grow linearly with the number of updated attributes, our scheme consistently achieves the lowest cost. The higher costs in [1] and [5] arise from extensive structural modifications required for multi-attribute or threshold updates. By contrast, our update key size depends only on the number of updated gates in the access tree, regardless of attribute count or gate type, enabling substantially faster updates as the number of attributes increases.

C. Trade-off Analysis

To support flexible and efficient policy updates, our scheme introduces a novel access tree structure, which impacts private key organization and adds decryption overhead. We analyze these trade-offs in terms of private key size and decryption cost, respectively.

Private key sizes. To realize flexible and lightweight updates for threshold access policies, our construction builds on the scheme in [10], which achieves constant-size ciphertexts for threshold access structures. However, to guarantee successful decryption under this framework, the private key structure necessarily inherits the construction of [10], where the size is tied to the attribute universe. This dependency is thus unavoidable: the larger private key size is the trade-off for maintaining compact ciphertexts and enabling efficient policy updates. Moreover, our construction supports update key generation directly at the gate level, eliminating the need for global structural modifications. As reflected in Tab. 4, this design optimizes the size of update keys, even though it requires larger private keys.

Decryption overheads. The decryption cost of our scheme is not directly comparable with that of LSSS-based approaches, primarily owing to their fundamentally different



(a) Comparison of encryption cost

(b) Comparison of updating OR gate cost

(c) Comparison of updating AND gate cost

(d) Comparison of updating Threshold gate cost

Fig. 8. Performance evaluation

structures. LSSS-based schemes typically incur heavy overhead when handling threshold or multi-attribute updates, as these operations matrix expansion and modifications to multiple rows. In contrast, our tree-based construction naturally accommodates threshold gates and complex updates without requiring large-scale structural changes, making it inherently more suitable for dynamic policy updating. Although our scheme involves a more complex decryption process, this trade-off enables enhanced adaptability and efficiency in policy updating, which is difficult to achieve with LSSS-based methods.

VII. Conclusion

In this paper, we propose a novel CP-ABE scheme with policy updating. The scheme enables efficient and flexible policy updates within a secure IoT-cloud access control system. Notably, it reduces the size of update keys to a constant when updating a gate associated with multiple attributes, regardless of gate type. This offers a significant improvement over existing schemes, where the update key size grows linearly with the number of attributes. Furthermore, our scheme supports a flexible updating mechanism, allowing for adjustments in the access policy to be more or less restrictive based on varied real-life requirements. While the scheme demonstrates several promising features, future work remains to explore shorter private keys, efficient decryption algorithms and applicability in broader practical scenarios.

References

[1] Yang K, Jia X, Ren K. Secure and verifiable policy update outsourcing for big data access control in the cloud[J]. IEEE Transactions on Parallel and Distributed Systems, 2014, 26(12): 3461-3470.

[2] Ying Z, Li H, Ma J, et al. Adaptively secure ciphertext-policy attribute-based encryption with dynamic policy updating[J]. Sci. China Inf. Sci, 2016, 59(4): 1-16.

[3] Ying Z, Jiang W, Liu X, et al. Reliable policy updating under efficient policy hidden fine-grained access control framework for cloud data sharing[J]. IEEE Transactions on Services Computing, 2021, 15(6): 3485-3498.

[4] Hao J, Liu J, Rong H, et al. OE-CP-ABE: over-encryption based cp-abe scheme for efficient policy updating[C]//Network and System Security: 11th International Conference, 2017: 499-509.

[5] Fugkeaw S, Sato H. Scalable and secure access control policy update for outsourced big data[J]. Future Generation Computer Systems, 2018, 79: 364-373.

[6] Zhang P, Chen Z, Liu J K, et al. An efficient access control scheme with outsourcing capability and attribute update for fog computing[J]. Future Generation Computer Systems, 2018, 78: 753-762.

[7] Wang T, Zhou Y, Ma H, et al. Flexible and Controllable Access Policy Update for Encrypted Data Sharing in the Cloud[J]. The Computer Journal, 2023, 66(6): 1507-1524.

[8] Susilo W, Jiang P, Guo F, et al. EACSIP: Extendable access control system with integrity protection for enhancing collaboration in the cloud[J]. IEEE Transactions on Information Forensics and Security, 2017, 12(12): 3110-3122.

[9] Huang Q, Li N, Yang Y. DACSC: Dynamic and fine-grained access control for secure data collaboration in cloud computing[C]//2018 IEEE Global Communications Conference (GLOBECOM). IEEE, 2018: 1-7.

[10] Ge A, Zhang R, Chen C, et al. Threshold ciphertext policy attribute-based encryption with constant size ciphertexts[C]//Information Security and Privacy: 17th Australasian Conference, ACISP 2012, Wollongong, NSW, Australia, July 9-11, 2012. Proceedings 17. Springer Berlin Heidelberg, 2012: 336-349.
[11] Shamir A. Identity-based cryptosystems and signature schemes[C]//Workshop on the theory and application of cryptographic techniques. Springer, Berlin, Heidelberg, 1984: 47-53.
[12] Boneh D, Franklin M. Identity-based encryption from the Weil pairing[C]//Annual international cryptology conference. Springer, Berlin, Heidelberg, 2001: 213-229.
[13] Sahai A, Waters B. Fuzzy identity-based encryption[C]//Advances in Cryptology–EUROCRYPT 2005: 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005. Proceedings 24. Springer Berlin Heidelberg, 2005: 457-473.
[14] Goyal V, Pandey O, Sahai A, et al. Attribute-based encryption for fine-grained access control of encrypted data[C]. The 13th ACM conference on Computer and communications security. 2006: 89-98.
[15] Bethencourt J, Sahai A, Waters B. Ciphertext-policy attribute-based encryption[C]//2007 IEEE symposium on security and privacy (SP'07). IEEE, 2007: 321-334.
[16] Ge C, Susilo W, Baek J, et al. Revocable attribute-based encryption with data integrity in clouds[J]. IEEE Transactions on Dependable and Secure Computing, 2021, 19(5): 2864-2872.
[17] Han D, Pan N, Li K C. A traceable and revocable ciphertext-policy attribute-based encryption scheme based on privacy protection[J]. IEEE Transactions on Dependable and Secure Computing, 2020, 19(1): 316-327.
[18] Li J, Zhang Y, Ning J, et al. Attribute based encryption with privacy protection and accountability for CloudIoT[J]. IEEE Transactions on Cloud Computing, 2020, 10(2): 762-773.
[19] Li J, Zhang Y, Ning J, et al. Attribute based encryption with privacy protection and accountability for CloudIoT[J]. IEEE Transactions on Cloud Computing, 2020, 10(2): 762-773.
[20] Chase M. Multi-authority attribute based encryption[C]//Theory of Cryptography: 4th Theory of Cryptography Conference, TCC 2007, Amsterdam, The Netherlands, February 21-24, 2007. Proceedings 4. Springer Berlin Heidelberg, 2007: 515-534.
[21] Lewko A, Waters B. Decentralizing attribute-based encryption[C]//Annual international conference on the theory and applications of cryptographic techniques. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011: 568-588.
[22] Phuong T V X, Yang G, Susilo W. Hidden ciphertext policy attribute-based encryption under standard assumptions[J]. IEEE transactions on information forensics and security, 2015, 11(1): 35-45.
[23] Kim I, Susilo W, Baek J, et al. Harnessing policy authenticity for hidden ciphertext policy attribute-based encryption[J]. IEEE Transactions on Dependable and Secure Computing, 2020, 19(3): 1856-1870.
[24] Wang S, Zhou J, Liu J K, et al. An efficient file hierarchy attribute-based encryption scheme in cloud computing[J]. IEEE Transactions on Information Forensics and Security, 2016, 11(6): 1265-1277.
[25] Wan Z, Deng R H. HASBE: A hierarchical attribute-based solution for flexible and scalable access control in cloud computing[J]. IEEE transactions on information forensics and security, 2011, 7(2): 743-754.
[26] Sahai A, Seyalioglu H, Waters B. Dynamic credentials and ciphertext delegation for attribute-based encryption[C]//Advances in Cryptology–CRYPTO 2012: 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings. Springer Berlin Heidelberg, 2012: 199-217.
[27] Barcelo M, Correa A, Llorca J, et al. IoT-cloud service optimization in next generation smart environments[J]. IEEE Journal on Selected Areas in Communications, 2016, 34(12): 4077-4090.
[28] Baker T, Asim M, Tawfik H, et al. An energy-aware service composition algorithm for multiple cloud-based IoT applications[J]. Journal of Network and Computer Applications, 2017, 89: 96-108.
[29] Xu X, Liu Z, Wang Z, et al. S-ABC: A paradigm of service domain-oriented artificial bee colony algorithms for service selection and composition[J]. Future Generation Computer Systems, 2017, 68: 304-319.
[30] Chatterjee S, Misra S, Khan S U. Optimal data center scheduling for quality of service management in sensor-cloud[J]. IEEE Transactions on Cloud Computing, 2015, 7(1): 89-101.
[31] Boneh D, Boyen X, Goh E J. Hierarchical identity based encryption with constant size ciphertext[C]//Annual international conference on the theory and applications of cryptographic techniques. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005: 440-456.
[32] Granlund T. The GNU multiple precision arithmetic library[J]. http://gmplib. org/, 2010.
[33] Lynn B. The pairing-based cryptography library[J]. Internet: crypto. stanford.edu/pbc/[Mar.27,2013], 2006.

Luqi Huang received the M.S. degree from the Xi'an University of Science and Technology, China. She is currently working toward the Ph.D. degree in the University of Wollongong, Australia. Her current research interests include public key cryptography and information security.

Fuchun Guo received his Ph.D. degree from University of Wollongong, Australia, in 2013. He is currently an Associate Professsor at the School of Computing and Information Technology, University of Wollongong. He is ARC DECRA Fellow (2017-2019) and ARC Future Fellow (2023-2026). His primary research interest is the public-key cryptography; in particular, protocols, encryption and signature schemes, and security proof.

Willy Susilo received his Ph.D. degree in Computer Science from University of Wollongong, Australia. He is a Distinguished Professor and the Head of School of Computing and Information Technology and the director of Institute of Cybersecurity and Cryptology (iC2) at the University of Wollongong. Recently, he was awarded an Australian Laureate Fellowship, which is the most prestigious award in Australia, due to his contribution in cloud computing security. He was previously awarded a prestigious ARC Future Fellow by the Australian Research Council (ARC) and the Researcher of the Year award in 2016 by the University of Wollongong. He is a Fellow of IEEE, Australian Computer Society (ACS), IET and AAAI. His main research interests include cybersecurity, cryptography and information security. His work has been cited more than 25,000 times in Google Scholar. He is the Editor-in-Chief of the Elsevier Computer Standards and Interfaces and the MDPI Information journal. He has served as a program committee member in dozens of international conferences. He is currently serving as an Associate Editors in several international journals, including IEEE Transactions in Dependable and Secure Computing. Previously, he has served in many top tier journals, such as IEEE Transactions in Informaton Forensics and Security. He has published more than 500 research papers in the area of cybersecurity and cryptology.

Li Wang is currently working toward the PhD degree with the School of Computer Science and Technology, Anhui University, Hefei, China. Her research focuses on the security of vehicular ad hoc network.