Received August 4, 2017, accepted September 15, 2017, date of publication September 26, 2017,
date of current version October 25, 2017.

Digital Object Identifier 10.1109/ACCESS.2017.2756070

Key-Policy Attribute-Based Encryption With Equality Test in Cloud Computing

HUIJUN ZHU[1], LICHENG WANG[1], HASEEB AHMAD[2], AND XINXIN NIU[1]
[1]State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing 100876, China
[2]Department of Computer Science, National Textile University, Faisalabad 37610, Pakistan

Corresponding author: Licheng Wang (wanglc2012@126.com)

This work was supported in part by the National Key Research and Development Program under Grant 2016YFB0800602 and in part by the National Natural Science Foundation of China under Grant 61370194 and Grant 61502048.

ABSTRACT The privacy of users must be considered as the utmost priority in distributed networks. To protect the identities of users, attribute-based encryption (ABE) was presented by Sahai *et al.* ABE has been widely used in many scenarios, particularly in cloud computing. In this paper, public key encryption with equality test is concatenated with key-policy ABE (KP-ABE) to present KP-ABE with equality test (KP-ABEwET). The proposed scheme not only offers fine-grained authorization of ciphertexts but also protects the identities of users. In contrast to ABE with keyword search, KP-ABEwET can test whether the ciphertexts encrypted by different public keys contain the same information. Moreover, the authorization process of the presented scheme is more flexible than that of Ma *et al.*'s scheme. Furthermore, the proposed scheme achieves one-way against chosen-ciphertext attack based on the bilinear Diffie–Hellman (BDH) assumption. In addition, a new computational problem called the twin-decision BDH problem (tDBDH) is proposed in this paper. tDBDH is proved to be as hard as the decisional BDH problem. Finally, for the first time, the security model of authorization is provided, and the security of authorization based on the tDBDH assumption is proven in the random oracle model.

INDEX TERMS Cloud service, attribute-based encryption, public key encryption, equality test, keyword search.

I. INTRODUCTION

In the current network era, cloud service providers offer infinite storage space and computing power for users to manage their data. To enjoy these services, individuals and organizations store their private data on cloud servers. However, in the case of security breaches, users' private data stored in the cloud are no longer safe. When users outsource their data to cloud servers, they expect complete privacy of their data stored in the cloud. Protecting the privacy and data of users has remained a very crucial problem for cloud servers. To avoid any inconvenience, users store their private data in encrypted form.

For fine-grained sharing of encrypted data, Sahai and Waters presented attribute-based encryption (ABE) [2]. ABE is a public key cryptosystem variant that allows users to access secret data based on their attributes. This cryptosystem enriches the flexibility of the encryption policy and the description of users' rights, and it changes from a one-one to one-many scenario during the encryption and decryption phases. Moreover, it hides the identities of the users in appropriate terms. In a subsequent work, Goyal et al. proposed key-policy attribute-based encryption (KP-ABE) in 2006 [18]. The underlying cryptosystem combines the secret key and the access structure. Bethencourt et al. proposed ciphertext-policy attribute-based encryption (CP-ABE) in 2007 [19], which combines the ciphertext and the access structure. Thereafter, numerous cryptographers presented many research works based on ABE [20]–[24]. Soon after its conceptualization, ABE reached prime importance in our daily life (for example, in television payment systems, personal health record systems and so on). Moreover, ABE is also being widely incorporated in cloud computing. However, if one wants to compare plaintexts corresponding to two ciphertexts, the secret key must be used to decrypt the two ciphertexts.

To overcome this problem, Yang *et al.* [25] presented a new cryptosystem called public key encryption with equality test (PKEwET) in 2010. His proposed system can test whether two ciphertexts contain the same plaintexts without decryption. However, this scheme allows anyone to perform such a test. To overcome this defect, Tang [26] made some improvements to the scheme (e.g., PKEET with fine-grained authorization (FGwPKEET), all-or-nothing PKEET (AoNwPKEET) [28] and an extension of FG-PKEwET [27]). In 2015, Ma *et al.* [29] proposed a new primitive called PKEwET supporting flexible authorization (PKEwET-FA). There are 4 types of flexible authorizations in their scheme. To simplify the certificate management of PKEwET, Ma [30] combined the concepts of PKEwET and identity-based encryption to present identity-based encryption with equality test (IBEET). Recently, in 2017, Wu *et al.* [31] improved Ma et al.'s scheme by reducing the computational time cost.

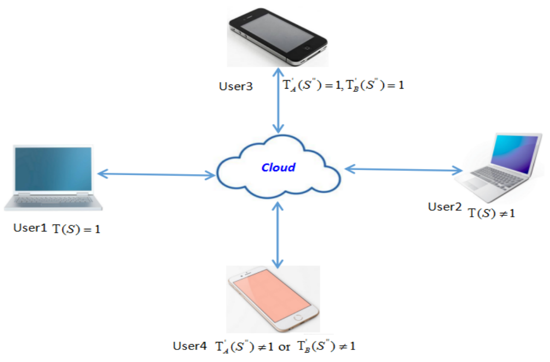
User3
$T'_A(S'')=1, T'_B(S'')=1$
Cloud
User1 $T(S)=1$
User2 $T(S)\neq 1$
User4 $T'_A(S'')\neq 1$ or $T'_B(S'')\neq 1$

FIGURE 1. **Example for KP-ABEwET.**

To offer more fine-grained authorization, we propose a new primitive called key-policy attribute-based encryption with equality test (KP-ABEwET). We combine the concepts of PKEwET and KP-ABE. As presented in Fig. 1, suppose that there are four users. S and S' are the sets of attributes for encryption, and $\mathbb{T}$ and $\mathbb{T}'$ refer to the access structures used by the decryption secret key. S'' denotes the set of attributes of the tester, and $\mathbb{T}'_A$ is the access structure used for the authorization of the attribute set of S'_A. $\mathbb{T}'_B$ is the access structure used for the authorization of the attribute set of S'_B. We describe the underlying scenario as follows: User 1 can store his private data in the cloud and can decrypt the ciphertexts that are encrypted by a set of attributes S with $\mathbb{T}(S) = 1$. User 2 can store his private data in the cloud, but he cannot decrypt the ciphertexts that are encrypted by a set of attributes S with $\mathbb{T}(S) \neq 1$. User 3 has the attribute S'', where $\mathbb{T}'_A(S'') = 1$ and $\mathbb{T}'_B(S'') = 1$, and he can perform the test over two different ciphertexts encrypted by attribute S'_A and attribute S'_B. User 4 does not have the attribute S'' satisfying $\mathbb{T}'_A(S'') = 1$ and $\mathbb{T}'_B(S'') = 1$, and he cannot perform the test over two different ciphertexts encrypted by attribute S'_A and attribute S_B'.

A. CONTRIBUTION

This paper presents a new primitive called key-policy attribute-based encryption with equality test (KP-ABEwET). Our objective is to achieve a fine-grained authorization of ciphertexts. The main technologies in our scheme include key-policy attribute-based encryption (KP-ABE) [18] and public key encryption with equality test (PKEwET) [25]. The main contributions can be summarized as follows:

(1) First, we design a new scheme by combining KP-ABE with PKEwET. Compared with the existing PKEwET schemes, our proposed scheme supports performing the fine-grained test of ciphertexts and changes from one-one to one-many for users in the testing algorithm.

(2) Our scheme can be viewed as an extension of attribute-based encryption with keyword search (ABEwKS). Along with other aspects, the proposed scheme allows testing whether the ciphertexts contain the same information that are encrypted by different public keys.

(3) The proposed scheme achieves one-way against chosen-ciphertext attack (OW-CCA) based on the bilinear Diffie-Hellman (BDH) assumption in the random oracle model.

(4) A new computational problem called the twin-decision bilinear Diffie-Hellman problem (tDBDH) is also presented and is proven to be as hard as the DBDH problem.

(5) We provide the security model of authorization and prove the security of authorization based on the tDBDH assumption in the random oracle model. To the best of our knowledge, this work is the first to prove the security of authorization in such a manner.

B. RELATED WORK

Deterministic encryption, proposed by Bellare *et al.* [8], is another primitive that supports the equality test on ciphertexts. This primitive was thoroughly studied in many subsequent works [1], [7], [32], but all of them are deterministic algorithms. Conversely, PKEwET is a probabilistic algorithm that supports the equality test on ciphertexts.

PKEwET can be viewed as an extension of public key encryption with keyword search (PEKS). The concept of PEKS was proposed by Boneh *et al.* [4]. It can perform keyword searches over ciphertexts without decrypting them. Later, several modified schemes of PEKS were proposed [6], [9], [11], [12]. To solve the problem of access control in a multi-user environment, PEKS was combined with ABE for achieving the applied perspective in cloud computing. In [5], [10], [13], [15], [17], and [33], the authors combined PKES with KP-ABE. In some other works, including [3], [14], [16], the authors combined PKES with CP-ABE while incorporating the access structure with the ciphertext of the keyword search. Although the results were slightly different, none of the works presented a mechanism to check whether two different ciphertexts encrypted by different public keys contain the same information. To overcome this limitation, we present an effective KP-ABEwET mechanism.

C. ORGANIZATION

The remainder of this paper is organized as follows. In Section 2, we introduce related preliminaries. Section 3 describes the system and the security model. Our scheme is presented in Section 4. Section 5 provides the security proof of our scheme and of authorization. In Section 6, the performance evaluations are briefly discussed. Finally, Section 7 presents the concluding remarks.

II. PRELIMINARIES

In this part, we introduce some basic knowledge, including cryptographic assumptions, Shamir's secret sharing scheme and access tree, that is employed in this paper.

A. CRYPTOGRAPHIC ASSUMPTIONS

The following subsection presents the definitions of bilinear maps and the problem formulation.

Definition 1: Bilinear Maps: Let $\mathbb{G}_1$ and $\mathbb{G}_2$ be multiplicative groups of prime order q, $e : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2$ be a bilinear map, and g be a generator of $\mathbb{G}_1$. Bilinear maps fulfill the following conditions:

(1) Bilinearity: $\forall g_1, g_2 \in \mathbb{G}_1$ and $\forall a, b \in Z_q$, we have $e(g_1^a, g_2^b) = e(g_1, g_2)^{ab}$.

(2) Non-degeneracy: $e(g, g) \neq 1$.

(3) Computability: $\forall g_1, g_2 \in \mathbb{G}_1$, we can compute $e(g_1, g_2)$.

Definition 2: Bilinear Diffie-Hellman (BDH) problem: Let $\mathbb{G}_1$ and $\mathbb{G}_2$ be multiplicative groups of prime order q, $e : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2$ be a bilinear map, and g be a generator of $\mathbb{G}_1$. The BDH problem is that given a 4-tuple (g, g^a, g^b, g^c), the aim is to compute $e(g, g)^{abc}$, where $a, b, c \in Z_q$.

Definition 3: Decisional Bilinear Diffie-Hellman (DBDH) problem: Let $\mathbb{G}_1$ and $\mathbb{G}_2$ be multiplicative groups of prime order q, $e : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2$ be a bilinear map, and g be a generator of $\mathbb{G}_1$. The DBDH problem is to distinguish between the distributions of 5-tuples $(g, g^a, g^b, g^c, e(g, g)^{abc})$ and $(g, g^a, g^b, g^c, e(g, g)^d)$, where $a, b, c, d \in Z_q$.

Definition 4: Twin-Decision Bilinear Diffie-Hellman (tDBDH) problem: Let $\mathbb{G}_1$ and $\mathbb{G}_2$ be multiplicative groups of prime order q, $e : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2$ be a bilinear map, and g be a generator of $\mathbb{G}_1$. The tDBDH problem is to distinguish between the following two distributions: $\mathcal{D}_0 = \{(g, g^a, g^b, g^c, g^u, g^v, e(g, g)^{abc}, e(g, g)^{auv}) : a, b, c, u, v \overset{\$}{\leftarrow} Z_q\}$ and $\mathcal{D}_1 = \{(g, g^a, g^b, g^c, g^u, g^v, e(g, g)^d, e(g, g)^w) : a, b, c, d, u, v, w \overset{\$}{\leftarrow} Z_q\}$.

In general, the tDBDH problem appears to be weaker than the DBDH problem. However, this problem is in fact as hard as the DBDH problem. (The tDBDH problem is different from the twin bilinear Diffie-Hellman inversion problem that proposed by Chen et al.)

Theorem 1: The tDBDH problem is as hard as the DBDH problem.

Proof: It is quite clear that tDBDH $\preceq$ DBDH. Next, we present the proof of DBDH $\preceq$ tDBDH.

To prove DBDH $\preceq$ tDBDH, we suppose that there is an algorithm $\mathfrak{A}$ that can solve the tDBDH problem in polynomial time. We construct an algorithm $\mathfrak{B}$ as follows. $\mathfrak{B}$ takes a 4-tuple $(g^a, g^b, g^c, e(g, g)^d)$ as input, and its objective is to determine whether $e(g, g)^d = e(g, g)^{abc}$ holds.

$\mathfrak{B}$ chooses a random number x and constructs a 7-tuple $(g^a, g^b, g^c, e(g, g)^d, g^{bx}, g^{cx}, e(g, g)^{dx^2})$. Then, it calls the algorithm $\mathfrak{A}$. The algorithm $\mathfrak{A}$ checks whether $e(g, g)^d = e(g, g)^{abc}$ and $e(g, g)^{dx^2} = e(g, g)^{abcx^2}$ hold.

If $\mathfrak{A}$ outputs yes, then it implies that $e(g, g)^d = e(g, g)^{abc}$ and $e(g, g)^{dx^2} = e(g, g)^{abcx^2}$. Apparently, it is doubly confirming that the input is a yes DBDH instance. Thus, $\mathfrak{B}$ replies "yes".

If $\mathfrak{A}$ outputs no, then it implies that either $e(g, g)^d \neq e(g, g)^{abc}$ or $e(g, g)^{dx^2} \neq e(g, g)^{abcx^2}$. Regardless of which is true, $\mathfrak{B}$ can quickly deduce that the input is a no DBDH instance. Thus, $\mathfrak{B}$ replies "no". ■

B. SHAMIR'S SECRET SHARING SCHEME

Shamir's (t, n)-threshold secret sharing scheme is based on the Lagrange interpolation polynomial. A detailed introduction is described as follows:

Given t distinct points $(x_i, f(x_i))$, where $f(x)$ is a polynomial of degree less than t, $f(x)$ is determined as follows:

$$f(x) = \sum_{i=1}^{t} \prod_{j=1, j\neq i}^{t} (x - x_j)/(x_i - x_j)$$

Shamir's scheme is defined for a secret $s \in Z_p$ by setting $a_0 = s$ and choosing $a_1, a_2, \cdots, a_{t-1} \in Z_q$. For all $1 \leq x_i \leq q$, $1 \leq i \leq n$, the trusted party computes $f(x_i)$, where $f(x) = \sum_{k=0}^{t-1} a_k x^k$. The shares $(x_i, f(x_i))$ are distributed to n distinct parties. Since the secret is a constant term $s = a_0 = f(0)$, the secret can be recovered from any t shares $(x_i, f(x_i))$ as follows:

$$s = f(0) = \sum_{i=1}^{t} f(x_i) \prod_{j=1, j\neq i}^{t} x_i/(x_j - x_i)$$

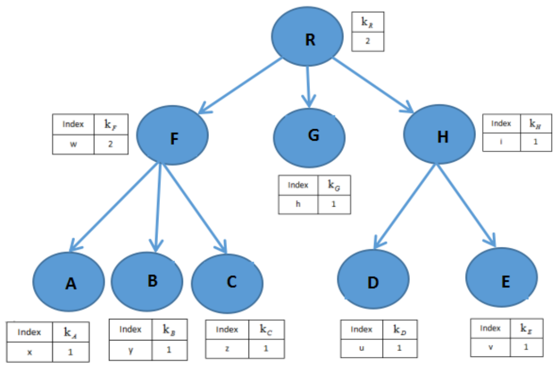

FIGURE 2. Example of access tree $\mathbb{T}$.

C. ACCESS TREE

We suppose that $\mathbb{T}$ is an access tree composed of leaf nodes and non-leaf nodes (e.g., Fig. 2). Each leaf node represents an attribute, and each non-leaf node represents a threshold gate. Each threshold gate is represented by its children and the threshold value. Let num_x be the number of children of a node x and k_x be the threshold value of the node x; we have $0 \leq k_x \leq num_x$. Then, each leaf node has a threshold value $k_x = 1$.

We suppose that the children of every node do have orders from 1 to num. Next, we define some new functions. The function parent(x) represents the parent of node x. The function att(x) is defined as an attribute associated with the leaf node. The function index(x) returns the number associated with node x.

Let r be the root of an access tree $\mathbb{T}$, expressed as $\mathbb{T}_r$. $\mathbb{T}_x$ refers to the subtree of $\mathbb{T}$ rooted at node x. $\mathbb{T}_x(S) = 1$ means that the set of attributes S satisfies the tree $\mathbb{T}_x$. Here, we use a recursive algorithm to compute $\mathbb{T}_x(S)$.

- If x is a non-leaf node, we compute $\mathbb{T}_{x'}(S)$ for all children x' of x. If at least k_x children return 1, then $\mathbb{T}_x(S)$ returns 1.
- If x is a leaf node, then $\mathbb{T}_x(S)$ returns 1 if att(x) $\in S$.

III. PROBLEM FORMULATION

In this section, we present the system and the security model.

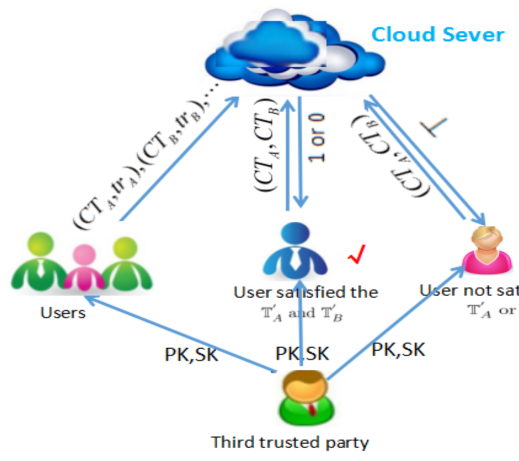
Cloud Sever
$(CT_A,tr_A),(CT_B,tr_B),\ldots$
(CT_A,CT_B)
1 or 0
(CT_A,CT_B)
$\perp$
Users
User satisfied the $\mathbb{T}'_A$ and $\mathbb{T}'_B$
User not satisfied the $\mathbb{T}'_A$ or $\mathbb{T}'_B$
PK,SK
PK,SK
PK,SK
Third trusted party

FIGURE 3. System model for KP-ABEwET.

A. SYSTEM MODEL

Fig. 3 illustrates the system model of KP-ABEwET. The system has three participating entities: the cloud server, the users and a trusted third party. The trusted third party generates public key pk and private key sk for users. The users encrypt and send their private data to the cloud server. If a user wants the cloud server to test the ciphertext, then the cloud server is authorized and gains a trapdoor tr. However, the cloud server can only test whether the two ciphertexts contain the same information and cannot decrypt them using the trapdoor. The legitimate users access data according to their attributes and can decrypt their ciphertexts or test the ciphertexts. If the legitimate users satisfy the access structure for the test, they can get the test results of the ciphertexts from the cloud server. If the legitimate users satisfy the access structure for the decryption, they can decrypt the ciphertexts.

An integrated KP-ABEwET scheme consists of six algorithms: **Setup**, **Encrypt**, **KeyGen**, **Trapdoor**, **Decrypt** and **Test**. Here, we let $\mathbb{M}$ be plaintext space and $\mathbb{C}$ be ciphertext space.

(1) **Setup**(k): It takes a security parameter k as input, and then it outputs the public parameters pp and pk and the master key mk.

(2) **Encrypt**(M, pk, S, S'): It takes a message $M \in \mathbb{M}$, public key pk and two sets of attributes S, S' as inputs, and then it outputs the ciphertext $CT \in \mathbb{C}$.

(3) **KeyGen**($\mathbb{T}, \mathbb{T}', S, S', pp, mk$): This algorithm takes as inputs the master key mk, two access trees $\mathbb{T}, \mathbb{T}'$, and two sets of attributes S, S' that satisfy $\mathbb{T}(S) = 1$ and $\mathbb{T}'(S') = 1$, and it subsequently outputs the private key sk.

(4) **Trapdoor**($S', \mathbb{T}', mk$): It takes mk, $\mathbb{T}'$ and S' as inputs, and it outputs the trapdoor td.

(5) **Decrypt**(CT, sk, S, S'): It takes as inputs a ciphertext $CT \in \mathbb{C}, S, S'$ and the private key sk, and it outputs the message M if $\mathbb{T}(S) = 1$ and $\mathbb{T}'(S') = 1$. Here, CT is encrypted using the sets S and S'.

(6) **Test**($CT_A, CT_B, td_A, td_B, S'$): Suppose that CT_A is a ciphertext of the sets of attributes S_A and S'_A and that CT_B is a ciphertext of the sets of attributes S_B and S'_B. This algorithm takes as inputs two ciphertexts CT_A, CT_B, the trapdoors td_A, td_B and the set S' of attributes that satisfy $\mathbb{T}'_A(S') = 1$ and $\mathbb{T}'_B(S') = 1$, and then it outputs 1 if CT_A and CT_B contain the same message; otherwise, it returns 0.

B. SECURITY MODEL

Here, the security model of the proposed scheme and the security model of authorization are presented.

First, we define one-way against chosen-ciphertext attack (OW-CCA) for KP-ABEwET under a chosen set of attributes, as follows.

Game 1: Suppose that $\mathcal{A}$ is the adversary. $\mathcal{A}$ announces a set of attributes that he wishes to be challenged, shown as $\mathcal{S}$.

(1) *Setup.* The challenger $\mathcal{C}$ takes a security parameter k as input and outputs public parameters pp to $\mathcal{A}$ with the **Setup** algorithm of KP-ABEwET.

(2) *Phase 1.* $\mathcal{A}$ performs the following types of queries polynomially many times.

- Key retrieve queries: $\mathcal{A}$ performs any queries for private keys for many access structures $\mathbb{T}_i$, where $\mathcal{S} \notin \mathbb{T}_i$ for all i. $\mathcal{C}$ sends sk to $\mathcal{A}$.
- Decryption queries: $\mathcal{A}$ performs many queries for ciphertexts. $\mathcal{C}$ runs the **Decrypt** algorithm and outputs the plaintext corresponding to the ciphertext or $\perp$ to $\mathcal{A}$.
- Trapdoor queries: $\mathcal{C}$ runs the **Trapdoor** algorithm and outputs td to $\mathcal{A}$.

(3) *Challenge:* $\mathcal{C}$ randomly chooses a message $M \in \mathbb{M}$, sets $CT^* = Encrypt(pk, M)$ and sends CT^* to $\mathcal{A}$ as his challenge ciphertext.

(4) *Phase 2:* Phase 1 is repeated. The constraints are that CT^* does not appear in the decryption queries.

(5) *Guess:* $\mathcal{A}$ outputs a guess $M^* \in \mathbb{M}$ and wins the game if $M^* = M$.

The advantage of $\mathcal{A}$ is defined as $\Pr[M^* = M]$.

Definition 5: The KP-ABEwET scheme is OW-CCA secure if the advantage of all polynomial time adversaries is negligible in the above game.

Finally, we define a testable against chosen-ciphertext attack (T-CCA) of authorization for KP-ABEwET under the chosen sets of attributes, as follows:

Game 2: Suppose that $\mathcal{A}_2$ is an adversary. $\mathcal{A}_2$ announces two sets of attributes $\mathcal{S}$ and $\mathcal{S}'$ that he wishes to be challenged. Here, $(\mathcal{S} \cap \mathcal{S}') = \varnothing$, $\mathcal{S}$ is used for decryption, and $\mathcal{S}'$ is used for the trapdoor.

(1) *Setup.* The challenger, $\mathcal{C}$, takes a security parameter k as input and outputs public parameters pp to $\mathcal{A}_2$ by using the **Setup** algorithm of KP-ABEwET.

(2) *Phase 1.* $\mathcal{A}_2$ performs the following types of queries polynomially many times.

- Key retrieve queries: $\mathcal{A}_2$ performs many queries for private keys for any access structures $\mathbb{T}_i$ and $\mathbb{T}'_j$, where $\mathcal{S} \notin \mathbb{T}_i$ for all i and $\mathcal{S}' \notin \mathbb{T}'_j$ for all j. $\mathcal{C}$ sends sk to $\mathcal{A}_2$.
- Decryption queries: $\mathcal{A}_2$ performs many queries for ciphertexts. $\mathcal{C}$ runs the **Decrypt** algorithm and outputs the plaintext corresponding to the ciphertext or $\perp$ to $\mathcal{A}_2$.
- Trapdoor queries: $\mathcal{C}$ runs the **Trapdoor** algorithm and outputs td to $\mathcal{A}_2$.
- Test queries: $\mathcal{C}$ runs the **Test** algorithm and outputs 1 for equality ciphertexts and 0 for unequal ciphertexts or $\perp$.

(3) *Challenge:* $\mathcal{C}$ chooses a random number $\vartheta \in \{0, 1\}$. If $\vartheta = 1$, then $\mathcal{C}$ chooses one message M, sets

$$CT_1^* = Encrypt(pk, M), CT_2^* = Encrypt(pk, M)$$

and sends CT_1^*, CT_2^* to $\mathcal{A}_2$ as his challenge ciphertexts. If $\vartheta = 0$, $\mathcal{C}$ chooses two unequal messages, M_1 and M_2; sets

$$CT_1^* = Encrypt(pk, M_1), CT_2^* = Encrypt(pk, M_2)$$

; and sends CT_1^*, CT_2^* to $\mathcal{A}_2$ as his challenge ciphertexts.

(4) *Phase 2:* Phase 1 is repeated with the conditions that CT_1^* and CT_2^* do not appear in decryption queries and CT_1^* and CT_2^* do not appear in test queries.

(5) *Guess:* $\mathcal{A}_2$ outputs a guess ϑ^* and wins the game if $\vartheta = \vartheta^*$, meaning 1 for $M_1 = M_2$ or 0 for $M_1 \neq M_2$.

The advantage of $\mathcal{A}_2$ is defined as $|\Pr[\vartheta^* = \vartheta] - 1/2|$.

Definition 6: The KP-ABEwET scheme is T-CCA secure in terms of authorization if the advantage of all polynomial time adversaries is negligible in the aforementioned game.

IV. OUR CONSTRUCTIONS

The following section presents the proposed KP-ABEwET scheme.

Setup (k): It takes a security parameter k as input and outputs public parameters pp as follows:

(1) Generate bilinear groups, $\mathbb{G}_1, \mathbb{G}_2$ and $|\mathbb{G}_1| = q$, $|\mathbb{G}_2| = q$, and choose a random generator $g \in \mathbb{G}_1$. Then, let $e : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$ be a bilinear map.

(2) Let A be a universe of properties of attributes. For simplicity, we take the first A elements of Z_q^* as the universe, formally as $1, 2, \cdots, |A| (\text{mod } q)$.

(3) Let $H_1 : \{0, 1\}^{|A|} \times \mathbb{G}_2 \rightarrow \{0, 1\}^{k+l}$, $H_2 : \{0, 1\}^{|A|} \times \mathbb{G}_2 \rightarrow \mathbb{G}_1$, and $H_3 : 5\mathbb{G}_1 \times \{0, 1\}^{k+l} \rightarrow \{0, 1\}^k$ be hash functions, where l is the length of the elements of Z_q.

(4) Choose $x_1, x_2, \cdots, x_{|A|}, y_1, y_2 \in Z_q^*$ randomly, and then output public keys pk,

$$X_1 = g^{x_1}, \cdots, X_{|A|} = g^{x_{|A|}}, Y_1 = e(g, g)^{y_1}, Y_2 = e(g, g)^{y_2}$$

, and the master key mk, $(x_1, x_2, \cdots, x_{|A|}, y_1, y_2)$.

Encrypt (M, pk, S, S')*:* It takes a message M, public key pk and two sets of attributes S, S' as inputs, where $(S \cap S') = \emptyset$, S is used for decryption, and S' is used for testing. Then, it outputs the ciphertext as follows:

Choose $r_1, r_2, r_3 \in Z_q$ randomly, and then formulate the following:

$$\begin{aligned} CT &= (S, S', C_1 = g^{r_1}, C_2 = M \parallel r_1 \oplus H_1(S, Y_1^{r_2}), \\ C_3 &= M^{r_1} \cdot H_2(S', Y_2^{r_3}), C_4 = \{E_i = X_i^{r_2}\}_{i \in S}, \\ C_5 &= \{E_j = X_j^{r_3}\}_{j \in S'}, C_6 = H_3(M^{r_1}, C_1, C_2, C_3, C_4, C_5)) \end{aligned}$$

KeyGen $(\mathbb{T}, \mathbb{T}', S, S', pp, mk)$*:* This algorithm takes the master key mk, two sets of attributes S, S' satisfying $\mathbb{T}(S) = 1$ and $\mathbb{T}'(S') = 1$ and $(S' \bigcap S) = \varnothing$ as inputs, and it outputs the private key as follows:

(1) The algorithm chooses a polynomial q_x for each node x in the tree $\mathbb{T}$. The polynomials are chosen from top to bottom, starting from the root node r. The details are presented as follows:

- For each node x in $\mathbb{T}$, it sets the degree d_x of the polynomial q_x to be one less than the threshold value k_x of that node, which means that $d_x = k_x - 1$.
- Then, for the root node r, it sets $q_r(0) = y_1$ and randomly chooses d_r other points of the polynomial q_r to define it completely.
- For any other node x, it sets $q_x(0) = q_{parent(x)}(index(x))$ and randomly chooses d_x other points to completely define q_x.
- For each leaf node x, it outputs $D_x = g^{q_x(0)/x_i}$, where $i = att(x)$.

(2) The algorithm chooses a polynomial q_t for each node t in $\mathbb{T}'$. The polynomials are chosen from top to bottom, starting from the root node r'. The details are described as follows:

- For each node t in $\mathbb{T}'$, it sets the degree d_t of the polynomial q_t to be one less than the threshold value k_t of that node, which means that $d_t = k_t - 1$.
- Then, for the root node r', it sets $q'_r(0) = y_2$ and randomly chooses $d_{r'}$ other points of the polynomial $q_{r'}$ to define it completely.
- For any other node t, the algorithm sets $q_t(0) = q_{parent(t)}(index(t))$ and randomly chooses d_t other points to completely define q_t.
- For each leaf node t, it outputs $T_t = g^{q_t(0)/x_j}$, where $j = att(t)$.

(3) The algorithm outputs secret key $sk = (D_x = g^{q_x(0)/x_i}, T_t = g^{q_t(0)/x_j})$, where $i = att(x), j = att(t)$.

Trapdoor $(\mathbb{T}', S', mk)$*:* It takes $\mathbb{T}'$, S' and mk as inputs; subsequently, it outputs a trapdoor td to test the ciphertexts as follows. Here, $\mathbb{T}'(S') = 1$.

The algorithm chooses a polynomial q_t for each node t in $\mathbb{T}'$. The polynomials are chosen from top to bottom, starting from the root node r'. The details are as follows:

- For each node t in $\mathbb{T}'$, it sets the degree d_t of the polynomial q_t to be one less than the threshold value k_t of that node, which means that $d_t = k_t - 1$.
- Then, for the root node r', it sets $q'_r(0) = y_2$ and randomly chooses $d_{r'}$ other points of the polynomial $q_{r'}$ to define it completely.
- For any other node t, the algorithm sets $q_t(0) = q_{parent(t)}(index(t))$ and randomly chooses d_t other points to completely define q_t.
- For each leaf node t, it outputs $T_t = g^{q_t(0)/x_j}$, where $j = att(t)$.

The algorithm outputs trapdoor $td = g^{q_t(0)/x_j}$, where $j = att(t)$.

Decrypt (CT, sk, S, S')*:* For decryption, we define a recursive algorithm, which is described as follows:

It takes the ciphertext CT, the private key sk and a node x in the tree $\mathbb{T}$ as inputs and outputs a group element of $\mathbb{G}_2$ or $\perp$.

If x is a leaf node and $i \in S$, where $i = att(x)$, then the algorithm computes as follows:

$DecryptNode(CT, sk, x) = e(D_x, E_i) = e(g^{q_x(0)/x_i}, g^{r_2 x_i}) = e(g, g)^{r_2 q_x(0)}$.

Otherwise, we define $DecryptNode(CT, sk, x) = \perp$.

Now, we consider that x is a non-leaf node. $DecryptNode$ (CT, sk, x) executes as follows:

Suppose that z is the child of x. Then, it executes algorithm $DecryptNode(CT, sk, z)$ and stores the output as O_z. Let F_x be an arbitrary k_x-sized set of child nodes z such that $O_z \neq \perp$. If no such set exists, then the node is not satisfied and the function returns $\perp$.

Otherwise, we compute the following:

$$\begin{aligned} O_x &= \prod_{z \in F_x} O_z^{\Delta_{i,F'_x}(0)} \\ &= \prod_{i \in F_x} (e(g,g)^{r_2 q_z(0)})^{\Delta_{i,F'_x}(0)} \\ &= \prod_{i \in F_x} (e(g,g)^{r_2 q_{parent(z)}(index(z))})^{\Delta_{i,F'_x}(0)} \\ &= \prod_{i \in F_x} e(g,g)^{r_2 q_x(i)\, \Delta_{i,F'_x}(0)} \\ &= e(g,g)^{r_2 q_x(0)}, \end{aligned}$$

where $i = index(z)$ and $F'_x = \{index(z) : z \in F_x\}$.

Then, $DecryptNode(CT, sk, r) = e(g, g)^{y_1 r_2} = Y_1^{r_2}$. Finally, it recovers $M || r_1 = C_2 \oplus H_1(S, Y_1^{r_2})$.

If $C_1 = g^{r_1}$ and $C_6 = H_3(M^{r_1}, C_1, C_2, C_3, C_4, C_5)$, it outputs M; otherwise, it outputs $\perp$.

Test $(CT_A, CT_B, td_A, td_B, S'')$*:* Suppose that CT_A, CT_B are two ciphertexts encrypted by $(S_A, S_{A'})$ and $(S_B, S_{B'})$ independently, $\mathbb{T}'_A(S'') = 1$, and $\mathbb{T}'_B(S'') = 1$, where $CT_A = (S_A, S'_A, C_{A,1}, C_{A,2}, C_{A,3}, C_{A,4}, C_{A,5}, C_{A,6})$ and $CT_B = (S_B, S'_B, C_{B,1}, C_{B,2}, C_{B,3}, C_{B,4}, C_{B,5}, C_{B,6})$. Subsequently, it uses td_A for the computation as follows:

If node t_A is the leaf node, then we let $j = att(t_A)$. If $j \in S'_A$, then $DecryptNode(CT_A, T'_{t_A}, t_A) = e(T'_{t_A}, C_{A,5}) = e(T'_{t_A}, E_{A,j}) = e(g^{q_{t_A}(0)/t_{A,j}}, g^{r_{A,3} t_{A,j}}) = e(g, g)^{r_{A,3} q_{t_A}(0)}$. Otherwise, we define $DecryptNode(CT_A, T'_{t_A}, t_A) = \perp$.

Next, we consider t_A to be a non-leaf node. $DecryptNode$ (CT_A, T'_{t_A}, t_A) executes as follows:

Suppose that z'_A is the child of t_A. Then, it runs algorithm $DecryptNode(CT_A, T'_{t_A}, t_A)$ and stores the output as O_{A,z'_A}. Let F_{t_A} be an arbitrary $k_{A,t}$-sized set of child nodes z'_A such that $O_{A,z'_A} \neq \perp$. If no such set exists, then the node is not satisfied and the function returns $\perp$.

Otherwise, it computes the following:

$$\begin{aligned} O_{t_A} &= \prod_{z'_A \in F_{t_A}} O_{z'_A}^{\Delta_{i,F'_{t_A}}(0)} \\ &= \prod_{i \in F_{t_A}} (e(g,g)^{r_{A,3} q_{t_A}(0)})^{\Delta_{i,F'_{t_A}}(0)} \\ &= \prod_{i \in F_{t_A}} (e(g,g)^{r_{A,3} q_{parent(z'_A)}(index(z'_A))})^{\Delta_{i,F'_{t_A}}(0)} \\ &= \prod_{i \in F_{t_A}} e(g,g)^{r_{A,3} q_{t_A}(i)\, \Delta_{i,F'_{t_A}}(0)} \\ &= e(g,g)^{r_{A,3} q_{t_A}(0)}, \end{aligned}$$

where $i = index(z'_A)$, $F'_{t_A} = index(z'_A) : z'_A \in F_{t_A}$.

Then, $DecryptNode(CT_A, T'_{t_A}, r'_A) = e(g, g)^{y_{A,2} r_{A,3}} = Y_{A,2}^{r_{A,3}}$. Finally, it recovers

$$M_A^{r_{A,1}} = C_{A,3}/H_2(S'_A, Y_{A,2}^{r_{A,3}}).$$

Then, it uses td_B to recover $M_B^{r_{B,1}} = C_{B,3}/H_2(S'_B, Y_{B,2}^{r_{B,3}})$ using the same method as above.

- If $C_{A,6} = H_3(M_A^{r_{A,1}}, C_{A,1}, C_{A,2}, C_{A,3}, C_{A,4}, C_{A,5})$ (resp. $C_{B,6} = H_3(M_B^{r_{B,1}}, C_{B,1}, C_{B,2}, C_{B,3}, C_{B,4}, C_{B,5})$), it computes the following: $e(M_A^{r_{A,1}}, C_{B,1})$ and $e(M_B^{r_{B,1}}, C_{A,1})$. Then, it outputs 1 for $e(M_A^{r_{A,1}}, C_{B,1}) = e(M_B^{r_{B,1}}, C_{A,1})$; otherwise, it outputs 0. Here, $r_{A,1}, r_{A,3}$ (resp. $r_{B,1}, r_{B,3}$) are the randomness used in the generation of CT_A (resp. CT_B).
- Otherwise, it outputs $\perp$.

V. SECURITY ANALYSIS

A. SECURITY OF SCHEME

The following subsection provides the security proof of the presented KP-ABEwET scheme.

Theorem 2: Our proposed scheme is OW-CCA secure against the adversary who is authorized with a trapdoor based on the BDH assumption in the random oracle model.

Proof: Suppose that $\mathcal{A}$ is the adversary that can break the presented KP-ABEwET scheme. Then, there is an algorithm $\mathcal{C}$ to solve the BDH problem with a non-negligible advantage. Given a 4-tuple $(g, A, B, C) = (g, g^a, g^b, g^c)$, the objective of algorithm $\mathcal{C}$ is to compute $e(g, g)^{abc}$.

Init Suppose that there is a universe $\mathcal{U}$. $\mathcal{A}$ chooses a set of attributes $\mathcal{S}$ as his target.

Setup Let $Y_1 = e(A, B) = e(g, g)^{ab}, Y_2 = e(g, g)^y$ $(y \in Z_p)$. For $i \in \mathcal{U}$, $\mathcal{C}$ sets X_i as follows: if $i \in \mathcal{S}$, it chooses a random $\alpha_i \in Z_p$ and sets $X_i = g^{\alpha_i}(x_i = \alpha_i)$; otherwise, it chooses a random $\tau_i \in Z_p$ and sets $X_i = g^{b\tau_i} = B^{\tau_i}$. Then, $\mathcal{C}$ gives the public parameters $pp = (X_1, X_2, \cdots, X_{|\mathcal{U}|}, Y_1, Y_2, H_1, H_2, H_3)$ to $\mathcal{A}$. Here, H_1 is a random oracle controlled by $\mathcal{C}$, as described below.

Phase 1 $\mathcal{A}$ performs the following types of queries polynomially times.

- H_1-query: $\mathcal{A}$ may issue queries to the random oracle H_1. To respond to these queries, $\mathcal{C}$ maintains a list of tuples H_1. Each element in the list is a tuple of the form $(S_\lambda, \delta_\lambda, \eta_\lambda)$. The list is initially empty. Responding to query $(S_\lambda, \delta_\lambda)$, $\mathcal{C}$ runs as follows:
  - If the query $(S_\lambda, \delta_\lambda)$ already appears in the H_1 list in the form $(S_\lambda, \delta_\lambda, \eta_\lambda)$, then $\mathcal{C}$ responds to $\mathcal{A}$ with $H_1(S_\lambda, \delta_\lambda) = \eta_\lambda$.
  - Otherwise, $\mathcal{C}$ just takes $\eta_\lambda \in G_2$, and then it responds to $\mathcal{A}$ with $H_1(S_\lambda, \delta_\lambda) = \eta_\lambda$. $\mathcal{C}$ adds the tuple $(S_\lambda, \delta_\lambda, \eta_\lambda)$ to the H_1 list.
- Key retrieve queries: $\mathcal{A}$ performs many queries for private keys for many access structures $\mathbb{T}$, where $\mathcal{S}$ does not satisfy $\mathbb{T}$. $\mathcal{C}$ sends sk to $\mathcal{A}$ as follows:
  (1) $\mathcal{C}$ builds two algorithms: Sat$\mathbb{T}$ and DNSat$\mathbb{T}$, as follows:
  Sat$\mathbb{T}(\mathbb{T}_x, \mathcal{S}, v_x)$: This algorithm constructs the polynomials for the nodes of an access sub-tree with a satisfied root node when $\mathbb{T}_x(\mathcal{S}) = 1$. It takes as inputs a set of attributes $\mathcal{S}$, an access tree $\mathbb{T}_x$ and a random number $v_x \in Z_p$, and it outputs a polynomial q_x of degree d_x for the root node x as follows:
  Let $q_x(0) = v_x$ and randomly choose d_x other points of the polynomial q_x to construct q_x. The algorithm constructs polynomials for each child node x' of x by executing the algorithm Sat$\mathbb{T}(\mathbb{T}_{x'}, \mathcal{S}, q_x(index(x')))$.
  DNSat$\mathbb{T}(\mathbb{T}_x, \mathcal{S}, g^{v_x})$: This algorithm constructs the polynomials for the nodes when $\mathbb{T}_x(\mathcal{S}) = 0$. It takes a set of attributes $\mathcal{S}$, an access tree $\mathbb{T}_x$ and a random element $g^{v_x} \in G_1$, where $v_x \in Z_p$, and it outputs a polynomial q_x of degree d_x for the root node x as follows:
  Because $\mathbb{T}_x(\mathcal{S}) = 0$, the root node has less than d_x satisfied children. Suppose that s_x is the number of satisfied children of x, which implies that $s_x < d_x$. The algorithm chooses a random number $v_{x'} \in Z_p$ for each satisfied child x' of x. Let $q_x(index(x')) = v_{x'}$ and randomly choose other $d_x - s_x$ points of the polynomial q_x to construct q_x.
  We obtain $q_x(\cdot)$ for each node in $\mathbb{T}$ as follows.
  If the node x' is a satisfied node, then it executes the algorithm Sat$\mathbb{T}(\mathbb{T}_{x'}, \mathcal{S}, q_x(index(x')))$. Here, we know $q_x(index(x'))$.
  Otherwise, it runs the algorithm DNSat$\mathbb{T}(\mathbb{T}_{x'}, \mathcal{S}, g^{q_x(index(x'))})$. Here, we know $g^{q_x(index(x'))}$.
  In the above algorithms, we know q_x for each leaf node x clearly satisfying $\mathbb{T}_x$; otherwise, we know $g^{q_x(0)}$. Furthermore, $q_r(0) = a$.
  (2) $\mathcal{C}$ constructs a polynomial $Q_x(\cdot) = bq_x(\cdot)$ and sets $y_1 = Q_r(0) = bq_r(0) = ab$.
  (3) Let $i = att(x)$ and $j = att(t)$. If $i \in \mathcal{S}$, then $D_x = g^{Q_x(0)/x_i} = g^{bq_x(0)/\alpha_i} = B^{q_x(0)/\alpha_i}$; otherwise, $D_x = g^{Q_x(0)/x_i} = g^{bq_x(0)/b\tau_i} = g^{q_x(0)/\tau_i}$. Then, $T_t = g^{Q_t(0)/x_j} = g^{bq_t(0)/b\tau_j} = g^{q_t(0)}/\tau_j$.
- Decryption queries: Suppose that the ciphertext $CT_\lambda = (S_\lambda, S'_\lambda, C_{\lambda,1}, C_{\lambda,2}, C_{\lambda,3}, C_{\lambda,4}, C_{\lambda,5}, C_{\lambda,6})$, i=att(x).
  - If $i \notin \mathcal{S}$, then $\mathcal{C}$ generates a private key as above and calls the Decrypt algorithm with the valid sk and provides the output to $\mathcal{A}$.
  - Otherwise, $\mathcal{C}$ proceeds as follows:
    * If S_λ is already included in the H_1 list in the form of $(S_\lambda, \delta_\lambda, \eta_\lambda)$, then $\mathcal{C}$ executes as follows:
      a. $M||r_1 = C_{\lambda,2} \oplus H_1(S_\lambda, \delta_\lambda)$
      b. Determine whether $C_{\lambda,1} = g^{r_1}$ and $C_{\lambda,6} = H_3(M^{r_1}, C_{\lambda,1}, C_{\lambda,2}, C_{\lambda,3}, C_{\lambda,4}, C_{\lambda,5})$ are established. If yes, $\mathcal{C}$ outputs M to $\mathcal{A}$. Otherwise, it outputs $\perp$ to $\mathcal{A}$.
    * Otherwise, it outputs $\perp$ to $\mathcal{A}$.
- Trapdoor queries: $\mathcal{A}$ performs many queries for trapdoor for many access structures $\mathbb{T}'$. $\mathcal{C}$ sends tr to $\mathcal{A}$ as follows: $\mathcal{C}$ constructs a polynomial $Q_t(\cdot)$ as in key retrieve queries. Let $j = att(t)$; then, output $td = T_t = g^{Q_t(0)/x_j} = g^{bq_t(0)/b\tau_j} = g^{q_t(0)/\tau_j}$ to $\mathcal{A}$.

Challenge: $\mathcal{C}$ randomly chooses a message M^* as a challenge message and outputs the ciphertext as follows:

$\mathcal{C}$ chooses random numbers $r_1, r_2 \in Z_q$, S' and $W \in \{0, 1\}^{k+l}$. Then, it outputs

$CT^* = (\mathcal{S}, S', C_1 = g^{r_1}, C_2 = M^*||r_1 \oplus H_1(\mathcal{S}, W), C_3 = M^{*r_1}H_2(S', Y_2^{r_3}), C_4 = \{E_i = X_i^c\}_{i\in\mathcal{S}} = \{E_i = C^{\alpha_i}\}_{i\in\mathcal{S}}, C_5 = \{E_j = X_j^{r_2}\}_{j\in S'}, C_6 = H_3(M^{*r_1}, C_1, C_2, C_3, C_4, C_5))$.

$\mathcal{C}$ responds to $\mathcal{A}$ with the challenge ciphertext CT^*.

Phase 2: $\mathcal{A}$ performs more queries as in phase 1. However, the restriction is that CT^* does not appear in the decryption queries.

Guess: $\mathcal{A}$ submits a guess M'. If $M' = M^*$, then it means that $\mathcal{A}$ can recover $e(g, g)^{abc}$ from (g^a, g^b, g^c). ■

B. SECURITY OF AUTHORIZATION

Finally, we provide the security proof of authorization.

Theorem 3: Our proposed scheme is T-CCA secure in terms of authorization against the adversary $\mathcal{A}_2$ based on the tDBDH assumption in the random oracle model.

Proof: Suppose that $\mathcal{A}_2$ is the adversary that can break our cryptosystem. Then, there is an algorithm $\mathcal{C}$ to solve the tDBDH problem as follows. The objective of algorithm $\mathcal{C}$ is to distinguish between the 7-tuples $(A, B, C, E, F, D, G) = (g^a, g^b, g^c, g^u, g^v, e(g, g)^{abc}, e(g, g)^{ubv})$ and $(A', B', C', E', F', D', G') = (g^a, g^b, g^c, g^u, g^v, e(g, g)^d, e(g, g)^w)$, where $a, b, c, d, u, v, w \in Z_p$.

Init Suppose that there is a universe $\mathcal{U}$. $\mathcal{A}_2$ chooses two sets of attributes $\mathcal{S}$ and $\mathcal{S}'$ as his target, where $(\mathcal{S}' \cap \mathcal{S}) = \emptyset$. Here, $\mathcal{S}$ is used for decryption, and $\mathcal{S}'$ is used for the trapdoor.

Setup Let $Y_1 = e(A, B) = e(g, g)^{ab}$, $Y_2 = e(E, B) = e(g, g)^{ub}$. For $i, j \in \mathcal{U}$, $\mathcal{C}$ sets X_i as follows:

- If $i \in \mathcal{S}$, it chooses a random $\alpha_i \in Z_p$ and sets $X_i = g^{\alpha_i}(x_i = \alpha_i)$.
- Otherwise, it sets as follows:
  - If $j \in \mathcal{S}'$, it chooses a random $\beta_j \in Z_p$ and sets $X_j = g^{\beta_j}(x_j = \beta_j)$;
  - Otherwise, it chooses a random $\tau_i \in Z_p$ and sets $X_i = g^{b\tau_i} = B^{\tau_i}(x_i = b\tau_i)$.

Subsequently, $\mathcal{C}$ provides the public parameters $pp = (X_1, X_2, \cdots, X_{|\mathcal{U}|}, Y_1, Y_2, H_1, H_2, H_3)$ to $\mathcal{A}_2$. Here, H_1 and H_2 are random oracles controlled by $\mathcal{C}$, as described below.

Phase 1 $\mathcal{A}_2$ performs the following types of queries polynomially times.

- H_1-query: $\mathcal{A}$ may issue queries to the random oracle H_1. To respond to these queries, $\mathcal{C}$ maintains a list of tuples H_1. Each element in the list is a tuple of the form $(S_\lambda, \delta_\lambda, \eta_\lambda)$. The list is initially empty. Responding to query $(S_\lambda, \delta_\lambda)$, $\mathcal{C}$ runs as follows:
  - If the query $(S_\lambda, \delta_\lambda)$ already appears in the H_1 list in the form $(S_\lambda, \delta_\lambda, \eta_\lambda)$, then $\mathcal{C}$ responds to $\mathcal{A}$ with $H_1(S_\lambda, \delta_\lambda) = \eta_\lambda$.
  - Otherwise, $\mathcal{C}$ just takes $\eta_\lambda \in G_2$, and $\mathcal{C}$ responds to $\mathcal{A}$ with $H_1(S_\lambda, \delta_\lambda) = \eta_\lambda$ and adds the tuple $(S_\lambda, \delta_\lambda, \eta_\lambda)$ to the H_1 list.
- H_2-query: $\mathcal{A}$ may issue queries to the random oracle H_2. To respond to these queries, $\mathcal{C}$ maintains a list of tuples H_2. Each element in the list is a tuple of the form $(S'_\lambda, \theta_\lambda, \mu_\lambda)$. The list is initially empty. Responding to query $(S'_\lambda, \theta_\lambda)$, $\mathcal{C}$ runs as follows:
  - If the query $(S'_\lambda, \theta_\lambda)$ already appears in the H_2 list in the form $(S'_\lambda, \theta_\lambda, \mu_\lambda)$, $\mathcal{C}$ responds to $\mathcal{A}_2$ with $H_2(S'_\lambda, \theta_\lambda) = \mu_\lambda$.
  - Otherwise, $\mathcal{C}$ just takes $\mu_\lambda \in \mathbb{G}_1$, and $\mathcal{C}$ responds to $\mathcal{A}_2$ with $H_2(S'_\lambda, \theta_\lambda) = \mu_\lambda$ and adds the tuple $(S'_\lambda, \theta_\lambda, \mu_\lambda)$ to the H_2 list.
- Key retrieve queries: $\mathcal{A}_2$ performs many queries for private keys for many access structures $\mathbb{T}$ and $\mathbb{T}'$, where $\mathcal{S}$ and $\mathcal{S}'$ do not satisfy $\mathbb{T}$ and $\mathbb{T}'$, respectively. $\mathcal{C}$ sends sk to $\mathcal{A}_2$ as follows:
  (1) To generate secret key (D_x, T_t), $\mathcal{C}$ builds two algorithms: Sat$\mathbb{T}$ and DNSat$\mathbb{T}$.
  Sat$\mathbb{T}(\mathbb{T}_x, \mathcal{S}, v_x)$: This algorithm constructs the polynomials for the nodes when $\mathbb{T}_x(\mathcal{S}) = 1$. It takes a set of attributes $\mathcal{S}$, an access tree $\mathbb{T}_x$ and a random number $v_x \in Z_p$, and it outputs a polynomial q_x of degree d_x for the root node x as follows:
  Let $q_x(0) = v_x$, and randomly choose d_x other points of the polynomial q_x to construct q_x. It constructs polynomials for each child node x' of x by running the algorithm Sat$\mathbb{T}(\mathbb{T}_{x'}, \mathcal{S}, q_x(index(x')))$.
  DNSat$\mathbb{T}(\mathbb{T}_x, \mathcal{S}, g^{v_x})$: This algorithm constructs the polynomials for the nodes when $\mathbb{T}_x(\mathcal{S}) = 0$. It takes a set of attributes $\mathcal{W}$, an access tree $\mathbb{T}_x$ and a random element $g^{v_x} \in G_1$, where $v_x \in Z_p$, and it outputs a polynomial q_x of degree d_x for the root node x as follows:
  Because $\mathbb{T}_x(\mathcal{S}) = 0$, the root node has less than d_x satisfied children. Suppose that s_x is the number of satisfied children of x, which implies that $s_x < d_x$. The algorithm chooses a random number $v_{x'} \in Z_p$ for each satisfied child x' of x. Let $q_x(index(x')) = v_{x'}$ and randomly choose other $d_x - s_x$ points of the polynomial q_x to construct q_x. If the node x' is a satisfied node, then it runs the algorithm Sat$\mathbb{T}(\mathbb{T}_{x'}, \mathcal{S}, q_x(index(x')))$. If the node x' is not a satisfied node, then it executes the algorithm DNSat$\mathbb{T}(\mathbb{T}_{x'}, \mathcal{S}, g^{q_x(index(x'))})$.
  In the above algorithms, we know q_x for each leaf node x clearly satisfying $\mathbb{T}_x$; otherwise, we know $g^{q_x(0)}$. Furthermore, $q_r(0) = a$.
  $\mathcal{C}$ constructs a polynomial $Q_x(\cdot) = bq_x(\cdot)$ and sets $y_1 = Q_r(0) = bq_r(0) = ab$.
  For $\mathbb{T}'$, it obtains $q_t(\cdot)$ for each node in $\mathbb{T}'$ as follows.
  If the node t' is a satisfied node, then it runs the algorithm Sat$\mathbb{T}'(\mathbb{T}'_{t'}, \mathcal{S}', q_t(index(t')))$. Here, we know $q_t(index(t'))$.
  Otherwise, it executes the algorithm DNSat$\mathbb{T}'(\mathbb{T}'_{x'}, \mathcal{S}', g^{q_t(index(t'))})$. Here, we know $g^{q_t(index(t'))}$.
  In the above algorithms, we know q_t for each leaf node t clearly satisfying $\mathbb{T}'_t$; otherwise, we know $g^{q_t(0)}$. Furthermore, $q_{r'}(0) = u$.
  $\mathcal{C}$ constructs a polynomial $Q_t(\cdot) = bq_t(\cdot)$ and sets $y_2 = Q_{r'}(0) = uq_{r'}(0) = ub$.
  Let $i = \text{att}(x)$ and $j = \text{att}(t)$.
  - If $i = j$, then it outputs $\perp$.
  - Otherwise,
    * If $i \in \mathcal{S}$, then $D_x = g^{Q_x(0)/x_i} = g^{bq_x(0)/\alpha_i} = B^{q_x(0)/\alpha_i}$;
    * Otherwise, $D_x = g^{Q_x(0)/x_i} = g^{bq_x(0)/b\tau_i} = g^{q_x(0)/\tau_i}$.
  - If $j = i$, then it outputs $\perp$.
  - Otherwise,
    * If $j \in \mathcal{S}'$, then $T_t = g^{Q_t(0)/t_j} = g^{bq_t(0)/\beta_j} = B^{q_t(0)/\beta_j}$;
    * Otherwise, $T_t = g^{Q_t(0)/t_j} = g^{bq_t(0)/b\tau_j} = g^{q_t(0)/\tau_j}$.
- Decryption queries: Suppose that the ciphertext $CT_\lambda = (S_\lambda, S'_\lambda, C_{\lambda,1}, C_{\lambda,2}, C_{\lambda,3}, C_{\lambda,4}, C_{\lambda,5}, C_{\lambda,6})$, $i = \text{att}(x)$.
  - If $i \notin \mathcal{S}$, $\mathcal{C}$ generates a private key of D_x as above and calls the Decrypt algorithm with D_x and saves M and r_1, then it continues as follows:

  - ∗ If $C_{\lambda,1} = g^{r_1}$ and $C_{\lambda,6} = H_3(M^{r_1}, C_{\lambda,1}, C_{\lambda,2}, C_{\lambda,3}, C_{\lambda,4}, C_{\lambda,5})$ are established, then $\mathcal{C}$ outputs M to $\mathcal{A}_2$.
  - ∗ Otherwise, $\mathcal{C}$ outputs $\perp$ to $\mathcal{A}_2$.
- – If $i \in \mathcal{S}$, $\mathcal{C}$ proceeds as follows:
  - ∗ If S_λ belongs to the H_1 list in the form of $(S_\lambda, \delta_\lambda, \eta_\lambda)$, then $\mathcal{C}$ executes as follows:
    a. $M||r_1 = C_{\lambda,2} \oplus H_1(S_\lambda, \delta_\lambda)$
    b. Checks whether $C_{\lambda,1} = g^{r_1}$ and $C_{\lambda,6} = H_3(M^{r_1}, C_{\lambda,1}, C_{\lambda,2}, C_{\lambda,3}, C_{\lambda,4}, C_{\lambda,5})$ are established. If yes, $\mathcal{C}$ outputs M to $\mathcal{A}_2$. Otherwise, $\mathcal{C}$ outputs $\perp$ to $\mathcal{A}_2$.
  - ∗ Otherwise, it outputs $\perp$.

- Trapdoor queries: $\mathcal{A}_2$ performs many queries for the trapdoor for many access structures $\mathbb{T}'$, where $\mathcal{S}'$ does not satisfy $\mathbb{T}'$. $\mathcal{C}$ sends td to $\mathcal{A}_2$ as follows:
  $\mathcal{C}$ uses the above algorithms: Sat$\mathbb{T}$ and DNSat$\mathbb{T}$.
  Then, it obtains $q_t(\cdot)$ for each node in the $\mathbb{T}'$ as follows. If the node t' is a satisfied node, then it runs the algorithm Sat$\mathbb{T}'(\mathbb{T}'_{t'}, \mathcal{S}', q_t(index(t')))$. Here, we already know $q_t(index(t'))$.
  Otherwise, it runs the algorithm DNSat$\mathbb{T}'(\mathbb{T}'_{x'}, \mathcal{S}', g^{q_t(index(t'))})$. Here, we already know $g^{q_t(index(t'))}$.
  In the above algorithms, we know q_t for each leaf node t clearly satisfying $\mathbb{T}'_t$; otherwise, we know $g^{q_t(0)}$. Furthermore, $q_{r'}(0) = u$.
  $\mathcal{C}$ constructs a polynomial $Q_t(\cdot) = bq_t(\cdot)$ and sets $y_2 = Q_{r'}(0) = uq_{r'}(0) = ub$. Let $j = att(t)$.
  If $j \in \mathcal{S}$, then it outputs $\perp$.
  Otherwise,
  - – If $j \in \mathcal{S}'$, then $T_t = g^{Q_t(0)/t_j} = g^{bq_t(0)/\beta_j} = B^{q_t(0)/\beta_j}$.
  - – Otherwise, $T_t = g^{Q_t(0)/t_j} = g^{bq_t(0)/b\tau_j} = g^{q_t(0)/\tau_j}$.
- Test queries: Suppose that the ciphertext $CT_\lambda = (S_\lambda, S'_\lambda, C_{\lambda,1}, C_{\lambda,2}, C_{\lambda,3}, C_{\lambda,4}, C_{\lambda,5}, C_{\lambda,6})$, $CT_{\lambda'} = (S_{\lambda'}, S'_{\lambda'}, C_{\lambda',1}, C_{\lambda',2}, C_{\lambda',3}, C_{\lambda',4}, C_{\lambda',5}, C_{\lambda',6})$, j_λ =att(t_λ), $j_{\lambda'}$ =att$(t_{\lambda'})$.
  - – If $j_\lambda \notin \mathcal{S}'$ and $j_{\lambda'} \notin \mathcal{S}'$, then $\mathcal{C}$ generates td_λ and $td_{\lambda'}$ as above and calls the Test algorithm with the valid td_λ and $td_{\lambda'}$, and it provides the output to $\mathcal{A}_2$.
  - – If $j_\lambda \notin \mathcal{S}'$ and $j_{\lambda'} \in \mathcal{S}'$, then $\mathcal{C}$ runs as follows:
    - ∗ For ciphertext CT_λ, $\mathcal{C}$ generates a td_λ as above, runs the Test algorithm and can obtain $M_\lambda^{r_{\lambda,1}}$ with the valid td_λ. Then, it checks whether $C_{\lambda,6} = H_3(M^{r_{\lambda,1}}, C_{\lambda,1}, C_{\lambda,2}, C_{\lambda,3}, C_{\lambda,4}, C_{\lambda,5})$. If yes, it stores $M_\lambda^{r_{\lambda,1}}$. Otherwise, it outputs $\perp$.
    - ∗ For ciphertext $CT_{\lambda'}$,
      a. If $S'_{\lambda'}$ belongs to the H_2 list in the form of $(S'_{\lambda'}, \theta_{\lambda'}, \mu_{\lambda'})$, then $\mathcal{C}$ computes $M_{\lambda'}^{r_{\lambda',1}} = C_{\lambda',3}/H_2(S'_{\lambda'}, \theta_{\lambda'})$. Then, it checks whether $C_{\lambda',6} = H_3(M^{r_{\lambda',1}}, C_{\lambda',1}, C_{\lambda',2}, C_{\lambda',3}, C_{\lambda',4}, C_{\lambda',5})$ holds. If yes, it stores $M_{\lambda'}^{r_{\lambda',1}}$. Otherwise, it outputs $\perp$.
      b. Otherwise, it outputs $\perp$
    - ∗ Then, $\mathcal{C}$ uses the storage elements of $M_{\lambda'}^{r_{\lambda',1}}$ and $M_\lambda^{r_{\lambda,1}}$ to compute $e(M_\lambda^{r_{\lambda,1}}, C_{\lambda',1})$ and $e(M_{\lambda'}^{r_{\lambda',1}}, C_{\lambda,1})$, respectively, and outputs 1 for $e(M_\lambda^{r_{\lambda,1}}, C_{\lambda',1}) = e(M_{\lambda'}^{r_{\lambda',1}}, C_{\lambda,1})$ and outputs 0 otherwise. Here, $r_{\lambda,1}$ (resp. $r_{\lambda',1}$) is the randomness used in the generation of CT_λ (resp. $CT_{\lambda'}$).
  - – If $j_\lambda \in \mathcal{S}'$ and $j_{\lambda'} \notin \mathcal{S}'$, $\mathcal{C}$ proceeds as follows:
    - ∗ For ciphertext CT_λ,
      a. If S'_λ belongs to the H_2 list in the form of $(S'_\lambda, \theta_\lambda, \mu_\lambda)$, then $\mathcal{C}$ computes $M_\lambda^{r_{\lambda,1}} = C_{\lambda,3}/H_2(S'_\lambda, \theta_\lambda)$. Then, it checks whether $C_{\lambda,6} = H_3(M^{r_{\lambda,1}}, C_{\lambda,1}, C_{\lambda,2}, C_{\lambda,3}, C_{\lambda,4}, C_{\lambda,5})$ holds. If yes, it stores $M_\lambda^{r_{\lambda,1}}$. Otherwise, it outputs $\perp$.
      b. Otherwise, it outputs $\perp$.
    - ∗ For ciphertext $CT_{\lambda'}$, $\mathcal{C}$ generates a $td_{\lambda'}$ as above, and then it runs the Test algorithm and can obtain $M_{\lambda'}^{r_{\lambda',1}}$ with the valid $td_{\lambda'}$. Then, it checks whether $C_{\lambda',6} = H_3(M^{r_{\lambda',1}}, C_{\lambda',1}, C_{\lambda',2}, C_{\lambda',3}, C_{\lambda',4}, C_{\lambda',5})$ holds. If yes, it stores $M_{\lambda'}^{r_{\lambda',1}}$. Otherwise, it outputs $\perp$.
    - ∗ Then, $\mathcal{C}$ uses the storage elements of $M_{\lambda'}^{r_{\lambda',1}}$ and $M_\lambda^{r_{\lambda,1}}$ to compute $e(M_\lambda^{r_{\lambda,1}}, C_{\lambda',1})$ and $e(M_{\lambda'}^{r_{\lambda',1}}, C_{\lambda,1})$, respectively, and outputs 1 for $e(M_\lambda^{r_{\lambda,1}}, C_{\lambda',1}) = e(M_{\lambda'}^{r_{\lambda',1}}, C_{\lambda,1})$ and outputs 0 otherwise. Here, $r_{\lambda,1}$ (resp. $r_{\lambda',1}$) is the randomness used in the generation of CT_λ (resp. $CT_{\lambda'}$).
  - – If $j_\lambda \in \mathcal{S}'$ and $j_{\lambda'} \in \mathcal{S}'$, then $\mathcal{C}$ proceeds as follows:
    - ∗ For ciphertext CT_λ,
      a. If S'_λ belongs to the H_2 list in the form of $(S'_\lambda, \theta_\lambda, \mu_\lambda)$, then $\mathcal{C}$ computes $M_\lambda^{r_{\lambda,1}} = C_{\lambda,3}/H_2(S'_\lambda, \theta_\lambda)$. Then, it checks whether $C_{\lambda,6} = H_3(M^{r_{\lambda,1}}, C_{\lambda,1}, C_{\lambda,2}, C_{\lambda,3}, C_{\lambda,4}, C_{\lambda,5})$ holds. If yes, it stores $M_\lambda^{r_{\lambda,1}}$. Otherwise, it outputs $\perp$.
      b. Otherwise, it outputs $\perp$.
    - ∗ For ciphertext $CT_{\lambda'}$,
      a. If $S'_{\lambda'}$ belongs to the H_2 list in the form of $(S'_{\lambda'}, \theta_{\lambda'}, \mu_{\lambda'})$, then $\mathcal{C}$ computes $M_{\lambda'}^{r_{\lambda',1}} = C_{\lambda',3}/H_2(S'_{\lambda'}, \theta_{\lambda'})$. Then, it checks whether $C_{\lambda',6} = H_3(M^{r_{\lambda',1}}, C_{\lambda',1}, C_{\lambda',2}, C_{\lambda',3}, C_{\lambda',4}, C_{\lambda',5})$ holds. If yes, it stores $M_{\lambda'}^{r_{\lambda',1}}$. Otherwise, it outputs $\perp$.
      b. Otherwise, it outputs $\perp$
    - ∗ Then, $\mathcal{C}$ uses the storage elements of $M_{\lambda'}^{r_{\lambda',1}}$ and $M_\lambda^{r_{\lambda,1}}$ to compute $e(M_\lambda^{r_{\lambda,1}}, C_{\lambda',1})$ and $e(M_{\lambda'}^{r_{\lambda',1}}, C_{\lambda,1})$, respectively, and outputs 1 for $e(M_\lambda^{r_{\lambda,1}}, C_{\lambda',1}) = e(M_{\lambda'}^{r_{\lambda',1}}, C_{\lambda,1})$ and outputs 0 otherwise. Here, $r_{\lambda,1}$ (resp. $r_{\lambda',1}$) is the randomness used in the generation of CT_λ (resp. $CT_{\lambda'}$).

Challenge: $\mathcal{C}$ chooses a random number $\vartheta \in \{0, 1\}$ and runs as follows:

If $\vartheta = 1$, then $\mathcal{C}$ chooses one message M, $D = e(g, g)^{abc}$, $G = e(g, g)^{ubv}$, and $r_1, r_2 \in Z_q$. Let $CT_1^* = (\mathcal{S}, \mathcal{S}', C_1 = g^{r_1}, C_2 = M||r_1 \oplus H_1(\mathcal{S}, D), C_3 = M^{r_1}H_2(\mathcal{S}', G), C_4 = \{E_i = X_i^c\}_{i\in\mathcal{S}} = \{E_i = C^{\alpha_i}\}_{i\in\mathcal{S}}, C_5 = \{E_j = X_j^z\}_{j\in\mathcal{S}'} = \{E_j = F^{\beta_j}\}_{j\in\mathcal{S}'}, C_6 = H_3(M^{r_1}, C_1, C_2, C_3, C_4, C_5))$.

TABLE 1. The comparison of computational complexity.

	C_{Enc}	C_{Dec}	C_{Test}	Attribute-based	POA	Security	Assumption
[25]	3E	3E	2P	No	No	OW-CCA	CDH
[26], [27]	4E	2E	4P	No	No	OW-CCA+IND-CCA	CDH+DDH
[28]	5E	2E	4E	No	No	OW-CCA+IND-CCA	CDH
[29]	6E	5E	2E+2P	No	No	OW-CCA+IND-CCA	CDH
[30]	6E+2P	2E+2P	4P	No	No	OW-ID-CCA	CDH
[31]	2E	2P	2P+2E	No	No	OW-ID-CCA	CDH
[18]	$(\|S_C\|+1)$E	$(\|S_C\|+1)$E $+(\|S_C\|)$P	No	Yes	No	CPA	DBDH
Our scheme	$(\|S_C\|+\|S_C^{'}\|+3)$E	$(\|S_C\|+\|S_C^{'}\|+2)$E $+(\|S_C\|+\|S_C^{'}\|)$P	$2(\|S_C^{'}\|)$E $+2(\|S_C^{'}\|)$P	Yes	Yes	OW-CCA+T-CCA	BDH+tDBDH

C_{Enc}, C_{Dec} and C_{Test}: the computational complexities of algorithms for encryption, decryption and test; POA: proof of authorization; E and P: the exponentiation operation and the pairing operation in the group $\mathbb{G}$; $|S_C|$ and $|S_C^{'}|$: the number of attributes required by the ciphertext; CPA: chosen plaintext attack; OW-CCA: one-way against chosen-ciphertext attack; IND-CCA: indistinguishable against chosen-ciphertext attack; T-CCA: tested against chosen-ciphertext attack; CDH: computational Diffie-Hellman assumption; DDH: decisional Diffie-Hellman assumption; BDH: bilinear Diffie-Hellman assumption; DBDH: decision bilinear Diffie-Hellman assumption; tDBDH: twin-decision bilinear Diffie-Hellman assumption;

$CT_2^* = (\mathcal{S}, \mathcal{S}', C_1 = g^{r_2}, C_2 = M||r_2 \oplus H_1(\mathcal{S}, D), C_3 = M^{r_2}H_2(\mathcal{S}', G), C_4 = \{E_i = X_i^c\}_{i\in\mathcal{S}} = \{E_i = C^{\alpha_i}\}_{i\in\mathcal{S}}, C_5 = \{E_j = X_j^z\}_{j\in\mathcal{S}'} = \{E_j = F^{\beta_j}\}_{j\in\mathcal{S}'}, C_6 = H_3(M^{r_1}, C_1, C_2, C_3, C_4, C_5))$.

Then, it sends the challenge ciphertexts CT_1^* and CT_2^* to the adversary.

If $\vartheta = 0$, then $\mathcal{C}$ chooses two unequal messages, M_1 and M_2, $D' = e(g, g)^d$, $G' = e(g, g)^w$ and $r_1, r_2 \in Z_q$.

Let $CT_1^* = (\mathcal{S}, \mathcal{S}', C_1 = g^{r_1}, C_2 = M_1||r_1 \oplus H_1(\mathcal{S}, D'), C_3 = M_1^{r_1}H_2(\mathcal{S}', G'), C_4 = \{E_i = X_i^c\}_{i\in\mathcal{S}} = \{E_i = C^{\alpha_i}\}_{i\in\mathcal{S}}, C_5 = \{E_j = X_j^v\}_{j\in\mathcal{S}'} = \{E_j = F^{\beta_j}\}_{j\in\mathcal{S}'}, C_6 = H_3(M^{r_1}, C_1, C_2, C_3, C_4, C_5))$.

$CT_2^* = (\mathcal{S}, \mathcal{S}', C_1 = g^{r_2}, C_2 = M_2||r_2 \oplus H_1(\mathcal{S}, D'), C_3 = M_2^{r_2}H_2(\mathcal{S}', G'), C_4 = \{E_i = X_i^c\}_{i\in\mathcal{S}} = \{E_i = C^{\alpha_i}\}_{i\in\mathcal{S}}, C_5 = \{E_j = X_j^v\}_{j\in\mathcal{S}'} = \{E_j = F^{\beta_j}\}_{j\in\mathcal{S}'}, C_6 = H_3(M^{r_1}, C_1, C_2, C_3, C_4, C_5))$.

Then, it sends the challenge ciphertexts CT_1^* and CT_2^* to the adversary.

Phase 2: $\mathcal{A}_2$ performs more queries as in phase 1. However, the restriction is that CT_1^* and CT_2^* do not appear in the decryption and the test queries.

Guess: $\mathcal{A}_2$ provides a guess ϑ^*. If $\vartheta^* = \vartheta$, then $\mathcal{C}$ outputs $\vartheta = 1$, which means that CT_1^* and CT_2^* contain the same message, and declares that it was given a valid 7-tuple: $(g^a, g^b, g^c, g^u, g^v, e(g, g)^{abc}, e(g, g)^{ubv})$; $\mathcal{C}$ outputs $\vartheta = 0$, which means that CT_1^* and CT_2^* contain different messages, and declares that it was given a random 7-tuple: $(g^a, g^b, g^c, g^u, g^v, e(g, g)^d, e(g, g)^w)$. ■

VI. PERFORMANCE EVALUATION

We theoretically analyze the asymptotic complexity of the proposed scheme and other PKEwET schemes in Table 1. We describe the computational complexity in terms of the exponentiation operation E and the pairing operation P. We denote the number of attributes required in the ciphertext by $|S_C|$ and $|S_C'|$. In Table 1, C_{Enc}, C_{Dec} and C_{Test} represent the encryption algorithms, decryption algorithms and test algorithms, respectively. POA represents the proof of authorization. From the second to the fourth columns, we present the computational complexities of C_{Enc}, C_{Dec} and C_{Test}. The fifth column indicates whether the underlying schemes are attribute based. The sixth column shows whether the schemes have the proof of authorization. The seventh column highlights the security levels of the schemes. The last column presents the underlying assumptions for guaranteeing the security.

From Table 1, we observe that the computational complexity of our scheme depends on the number of attributes required by the ciphertext. Because our scheme incorporates the ABE scenario, it may not be as efficient as the prevalent works. The trade off is adjusted while offering the protection of user identities. Furthermore, in contrast to previous works, our scheme also allows the users to obtain fine-grained authorization of ciphertexts. To the best of our knowledge, Ma et al. first presented four types of authorizations in [29]. We find that our proposed scheme can perform the authorization and test in a more flexible manner because in our scheme, we can perform the authorization using the attributes of users. Furthermore, for the first time, the proof of authorization is proven based on the tDBDH assumption.

In general, our scheme is more practical in the cloud computing era. Users can store their ciphertexts on the could server by using KP-ABEwET. The cloud server can be authorized to perform some functionalities by transforming some trapdoors secretly. Therefore, the could server can perform the equality test independently. When users want to test the ciphertexts, they submit their attributes and the tested ciphertexts to the cloud sever. Accordingly, the cloud server responds with a reasonable answer or ⊥. Meanwhile, it also protects the identities of users.

VII. CONCLUSION

In this paper, a new cryptosystem called key-policy attribute-based encryption with equality test (KP-ABEwET) is presented. To the best of our knowledge, KP-ABEwET is the first attempt to combine the public key encryption supporting equality test with key-policy attribute-based encryption. The proposed scheme can be viewed as an extension of attribute-based encryption with keyword search (ABEwKS) with the difference that it can test whether the ciphertexts contain the same information that were encrypted by different public keys. In contrast to previous schemes with equality test, the new scheme supports testing the ciphertexts with fine-grained authorization and also hides the identity of the user. Moreover, the proposed scheme is one-way secure against chosen-ciphertext attack (OW-CCA) based on the bilinear Diffie-Hellman (BDH) problem. Furthermore, a new computational problem called twin-decision bilinear Diffie-Hellman problem (tDBDH) is proposed and is proven to be as hard as the DBDH problem. Finally, the security model of authorization is presented, and the security of authorization based on the tDBDH assumption is proven in the random oracle model. To the best of our knowledge, this work is the first to prove the security of authorization in such a scenario.

REFERENCES

[1] A. Boldyreva, S. Fehr, and A. O'Neill, "On notions of security for deterministic encryption, and efficient constructions without random oracles," in *Proc. Annu. Int. Cryptol. Conf.*, 2008, pp. 335–359.

[2] A. Sahai and B. Waters, "Fuzzy identity-based encryption," in *Proc. Annu. Int. Conf. Theory Appl. Cryptograph. Techn.*, 2005, pp. 457–473.

[3] C. Wang, W. Li, Y. Li, and X. L. Xu, "A ciphertext-policy attribute-based encryption scheme supporting keyword search function," in *Proc. CSS*, 2013, pp. 377–386.

[4] D. Boneh, G. Di Crescenzo, R. Ostrovsky, and G. Persiano, "Public key encryption with keyword search," in *Proc. Int. Conf. Theory Appl. Cryptograph. Techn.*, 2004, pp. 506–522.

[5] F. Han *et al.*, "A general transformation from KP-ABE to searchable encryption," *Future Generat. Comput. Syst.*, vol. 30, pp. 107–115, Jan. 2014.

[6] M. Abdalla *et al.*, "Searchable encryption revisited: Consistency properties, relation to anonymous IBE, and extensions," *J. Cryptol.*, vol. 21, no. 3, pp. 350–391, Jul. 2008.

[7] M. Bellare, M. Fischlin, A. O'Neill, and T. Ristenpart, "Deterministic encryption: Definitional equivalences and constructions without random oracles," in *Advances in Cryptology—CRYPTO* (Lecture Notes Comput. Science), vol. 5157. Berlin, Germany: Springer-Verlag, Aug. 2008, pp. 360–378.

[8] M. Bellare, A. Boldyreva, and A. O'Neill, "Deterministic and efficiently searchable encryption," in *Proc. Annu. Int. Cryptol. Conf.*, 2007, pp. 535–552.

[9] M. Nishioka, "Perfect keyword privacy in PEKS systems," in *Provable Security*. Berlin, Germany: Springer, 2012, pp. 175–192.

[10] M. Chase and S. S. Chow, "Improving privacy and security in multi-authority attribute-based encryption," in *Proc. 16th ACM Conf. Comput. Commun. Secur.*, 2009, pp. 121–130.

[11] L. Ibraimi, S. Nikova, P. Hartel, and W. Jonker, "Public-key encryption with delegated search," in *Proc. Int. Conf. Appl. Cryptogr. Netw. Secur.*, 2011, pp. 532–549.

[12] H. S. Rhee, W. Susilo, and H.-J. Kim, "Secure searchable public key encryption scheme against keyword guessing attacks," *IEICE Electron. Exp.*, vol. 6, no. 5, pp. 237–243, 2009.

[13] J. Lai, X. Zhou, R. H. Deng, Y. Li, and K. Chen, "Expressive search on encrypted data," in *Proc. 8th ACM SIGSAC Symp. Inf.*, 2013, pp. 243–252.

[14] J. Li and L. Zhang, "Attribute-based keyword search and data access control in cloud," in *Proc. 10th Int. Conf. Comput. Intell. Secur. (CIS)*, Nov. 2014, pp. 382–386.

[15] J. Han, W. Susilo, Y. Mu, and J. Yan, "Privacy-preserving decentralized key-policy attribute-based encryption," *IEEE Trans. Parallel Distrib. Syst.*, vol. 23, no. 11, pp. 2150–2162, Nov. 2012.

[16] S. Li and M. Z. Xu, "Attribute-based public encryption with keyword search," *Chin. J. Comput.*, vol. 37, no. 5, pp. 1017–1024, 2014.

[17] P. Liu, J. Wang, H. Ma, and H. Nie, "Efficient verifiable public key encryption with keyword search based on KP-ABE," in *Proc. 9th Int. Conf. Broadband Wireless Comput., Commun. Appl. (BWCCA)*, Nov. 2014, pp. 584–589.

[18] V. Goyal, O. Pandey, A. Sahai, and B. Waters, "Attribute-based encryption for fine-grained access control of encrypted data," in *Proc. 13th ACM Conf. Comput. Commun. Secur.*, 2006, pp. 89–98.

[19] J. Bethencourt, A. Sahai, and B. Waters, "Ciphertext-policy attribute-based encryption," in *Proc. IEEE Symp. Secur. Privacy*, May 2007, pp. 321–334.

[20] Y. Ji *et al.*, "A privacy protection method based on CP-ABE and KP-ABE for cloud computing," *J. Softw.*, vol. 9, no. 6, pp. 1367–1375, 2014.

[21] C. Wang and J. Luo, "An efficient key-policy attribute-based encryption scheme with constant ciphertext length," *Math. Problems Eng.*, vol. 2013, Apr. 2013, Art. no. 810969.

[22] A. Lewko and B. Waters, "Decentralizing attribute-based encryption," in *Proc. Annu. Int. Conf. Theory Appl. Cryptogr. Techn.*, 2011, pp. 568–588.

[23] S. Hohenberger and B. Waters, "Online/offline attribute-based encryption," in *Proc. Int. Workshop Public Key Cryptogr.*, 2014, pp. 293–310.

[24] P. Datta, R. Dutta, and S. Mukhopadhyay, "Fully secure online/offline predicate and attribute-based encryption," in *Proc. ISPEC*, 2015, pp. 331–345.

[25] G. Yang, C. H. Tan, Q. Huang, and D. S. Wong , "Probabilistic public key encryption with equality test," in *Proc. Cryptogr.-Track RSA Conf.*, 2010, pp. 119–131.

[26] Q. Tang, "Towards public key encryption scheme supporting equality test with fine-grained authorization," in *Proc. Australasian Conf. Inf. Secur. Privacy*, 2011, pp. 389–406.

[27] Q. Tang, "Public key encryption schemes supporting equality test with authorisation of different granularity," *Int. J. Appl. Cryptogr.*, vol. 2, no. 4, pp. 304–321, 2012.

[28] Q. Tang, "Public key encryption supporting plaintext equality test and user-specified authorization," *Secur. Commun. Netw.*, vol. 5, no. 12, pp. 1351–1362, 2012.

[29] S. Ma, Q. Huang, M. Zhang, and B. Yang, "Efficient public key encryption with equality test supporting flexible authorization," *IEEE Trans. Inf. Forensics Security*, vol. 10, no. 3, pp. 458–470, Mar. 2015.

[30] S. Ma, "Identity-based encryption with outsourced equality test in cloud computing," *Inf. Sci.*, vol. 328, pp. 389–402, Jan. 2016.

[31] L. Wu, Y. Zhang, K.-K. R. Choo, and D. He, "Efficient and secure identity-based encryption scheme with equality test in cloud computing," *Future Generat. Comput. Syst.*, vol. 73, pp. 22–31, Aug. 2017.

[32] Z. Brakerski and G. Segev, "Better security for deterministic publickey encryption: The auxiliary-input setting," in *Proc. Annu. Cryptol. Conf.*, 2011, pp. 543–560.

[33] Z. Lv, C. Hong, M. Zhang, and D. Feng, "Expressive and secure searchable encryption in the public key setting," in *Proc. Int. Conf. Inf. Secur.*, 2014, pp. 364–376.

HUIJUN ZHU received the B.S. degree from Luoyang Normal University in 2007, and the M.S. degree from Henan Polytechnic University in 2010. She is currently pursuing the Ph.D. degree with the Beijing University of Posts and Telecommunications. Her research interests include modern cryptography, network security, and cloud computing.

LICHENG WANG received the B.S. degree from Northwest Normal University in 1995, the M.S. degree from Nanjing University in 2001, and the Ph.D. degree from Shanghai Jiao Tong University in 2007. He is currently an Associate Professor with the Beijing University of Posts and Telecommunications. His current research interests include modern cryptography, network security, and trust management.

HASEEB AHMAD received the B.S. degree from G.C. University, Faisalabad, Pakistan, in 2010, the master's degree from the Virtual University of Pakistan in 2012, and the Ph.D. degree from the Beijing University of Posts and Telecommunications, Beijing, China in 2017. He is currently an Assistant Professor with the Department of Computer Science, National Textile University, Faisalabad. His current research interest includes data mining, information retrieval, and information security.

XINXIN NIU received the M.S. degree from the Beijing University of Posts and Telecommunications in 1988, and the Ph.D. degree from The Chinese University of Hong Kong in 1997. She is currently a Professor of computer science and technology with the Beijing University of Posts and Telecommunications. Her current research interests include network security, digital watermarking, and digital rights management.