Traceable Scheme of Public Key Encryption with Equality Test

Huijun Zhu 1,2,3,*, Qingji Xue 1,3, Tianfeng Li 1,3 and Dong Xie 4

1 School of Digital Media and Art Design, Nanyang Institute of Technology, Nanyang 473004, China; xue_qj@sina.com (Q.X.); 3071066@nyist.edu.cn (T.L.)
2 State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing 100876, China
3 Graphic Image and Intelligent Processing in Henan Province, International Joint Laboratory, Nanyang Institute of Technology, Nanyang 473000, China
4 School of Computer and Information, Anhui Normal University, Wuhu 241002, China; xiedong@ahnu.edu.cn

* Correspondence: zhuhj121@nyist.edu.cn

Abstract: Public key encryption supporting equality test (PKEwET) schemes, because of their special function, have good applications in many fields, such as in cloud computing services, blockchain, and the Internet of Things. The original PKEwET has no authorization function. Subsequently, many PKEwET schemes have been proposed with the ability to perform authorization against various application scenarios. However, these schemes are incapable of traceability to the ciphertexts. In this paper, the ability of tracing to the ciphertexts is introduced into a PKEwET scheme. For the ciphertexts, the presented scheme supports not only the equality test, but also has the function of traceability. Meanwhile, the security of the proposed scheme is revealed by a game between an adversary and a simulator, and it achieves a desirable level of security. Depending on the attacker’s privileges, it can resist OW-CCA security against an adversary with a trapdoor, and can resist IND-CCA security against an adversary without a trapdoor. Finally, the performance of the presented scheme is discussed.

Keywords: public key encryption; equality test; blockchain; cloud server

1. Introduction

With the continuous development of the Internet of Things (IoT), the security of data has gotten more attention. In order to ensure the security of data, data are stored on a server by encryption. However, it is inconvenient for effective application when the data are encrypted, making it impossible to search within encrypted data. Therefore, searchable encryption (SE) is presented [1]. The aim of SE is to produce a tag related to ciphertext, and to classify the ciphertexts. Since this primitive approach was proposed, many cryptographers have studied it extensively and deeply [2–6]. However, the same ciphertext cannot be classified and stored by SE schemes. A new cryptographic primitive approach emerged as the times required, namely the public key encryption supporting equality test (PKEwET) [7]. In this paper, traceability is introduced into the PKEwET scheme.

1.1. Related Work

The PKEwET scheme resolves the problem of data matching in many application environments, such as in cloud computing, health service systems, and IoT. It can compare the consistency of the ciphertexts without the secret key. Recently, the research scope of PKEwET has focused on the three aspects of authorization, security scheme, and efficiency of the PKEwET scheme. Some progress in PKEwET is reviewed as follows:

For the authorization, Tang et al. and Huang et al. proposed PKEwET schemes supporting authorization from the user and ciphertext, respectively [8–13]. Then, Ma et al. extended the authorization mechanism to multi-user environments [14]. For more convenient application, Ma et al. proposed four types of authorization policies, namely...
user level, ciphertext level, user-specific ciphertext level, and ciphertext-to-user level authorization [15]. To simplify the maintenance of public key certificates, Ma et al. introduced the equality test algorithm into an identity-based encryption scheme [16]. For more convenient application to smart cities, Yang et al. proposed a filtered equality test scheme [17]. Later, Wang et al. combined signcryption and an equality test [18]. Recently, Duong et al. presented new lattice-based PKEwET schemes [19].

For the security, in 2016, Lee et al. improved the scheme of Ma, and proposed a new scheme that achieved IND-CCA security [20], and presented an equality test scheme based on the standard model for the first time [21]. In 2017, Wang et al. and Huang et al. proposed a PKEwET scheme from the ciphertext level, and presented the proof of security under the standard model [22,23]. Subsequently, some other PKEwET schemes based on the standard model have been proposed [24,25].

For the efficiency of the PKEwET schemes, Lin et al. and Zhu et al. proposed pairing-free equality test schemes [26,27]. The scheme of Tang was improved upon by Wu et al. [28,29], where the efficiency of computing increased by 36.7% in encryption and by 39.24% in the test algorithm. In 2018, Qu et al. introduced a certificate-less PKEwET scheme [30]. This scheme was improved upon by Elhabob et al. [31,32]. In 2019, Wu et al. combined Zhu et al.’s and Ma et al.’s schemes, and proposed the pairing-free scheme based identity [33]. In the same year, Lee et al. proposed a new PKEwET scheme, from generic assumptions in the random oracle model [34]. To reduce the cost of computing and communication, Ling et al. introduced the group mechanism into a PKEwET algorithm [35].

For convenience in cloud computing of the PKEwET scheme, key-policy attribute-based encryption was introduced by Zhu et al. [36]. In 2018, ciphertext-policy attribute-based encryption was introduced into a PKEwET scheme by Wang et al. [37]. Subsequently, some improvement schemes were put forward [38–41].

Driven by interests, some users may disclose their own secret keys to non-group users intentionally or unintentionally. However, it is difficult for the malicious user to be tracked down by the system. The problem of key abuse brings great security risks to PKEwET systems. To solve this problem, we introduce a tracking function into a PKEwET system.

1.2. Contributions

In this paper, traceability is introduced into a group ID-based encryption (GIBE) scheme. The motivation is to make a GIBE supporting traceability and an equality test function to the ciphertexts. The key contributions can be listed as follows:

- We show that the GIBE algorithm is unable to compare ciphertexts, and has no equality test function without the secret key $sk$. To overcome these limitations, we combine the GIBE and PKEwET algorithms. Additionally, all of PKEwET algorithms are untraceable to the encrypted ciphertexts, the idea of traceability is introduced into the PKEwET algorithm, and we propose the traceable GIBE with an equality test scheme (T-GIBEwET).
- Two types of adversaries are described, and the security of the proposed scheme is proved in details from two types of adversaries. The presented scheme achieves a desirable security. With a trapdoor, the T-GIBEwET scheme can resist OW-CCA security. Without a trapdoor, the T-GIBEwET scheme can resist IND-CCA security.
- The performance of the T-GIBEwET scheme is discussed. Compared to existing equality test schemes, it is more efficient and more practical in many scenarios.

1.3. Outline of This Paper

The rest of the proposal is organized as follows: some preliminaries, some basic definitions, assumptions and the security model are presented in Section 2. The details of the T-GIBEwET scheme are presented in Section 3. The security of the T-GIBEwET scheme is discussed in Section 4. In Section 5, the performance analysis of the T-GIBEwET scheme is represented. Finally, the concluding remarks of this paper are summarized in Section 6.
2. Preliminaries

In this section, we present the safety objectives, cryptographic assumptions and security models used in this paper.

2.1. Decisional Bilinear Diffie–Hellman Assumption

The proposed scheme is secure under the decisional bilinear Diffie–Hellman assumption. In this algorithm, the challenger $S$ picks $a, b, c, z \in Z^*_p$ and flips coin $\text{coin} \in \{0, 1\}$ randomly.

- If $\text{coin} = 0$, $S$ outputs $(g, g^a, g^b, g^c, e(g, g)^z)$.
- Otherwise, $S$ outputs $(g, g^a, g^b, g^c, e(g, g)^{abc})$.

Then, the adversary $A$ gives a guess of $\text{coin}$.

2.2. Definition of PKEwET

The PKEwET scheme contains four algorithms [7]:

1. **KeyGen** ($1^l$): This procedure randomly selects $x \in Z^*_q$, and outputs the public/secret key pair $(pk = g^x, sk = x)$, where $g$ is a generator of $G$.

2. **Encrypt** $(M, pk)$: This procedure selects the numbers $r \in Z^*_q$ randomly. Then, it outputs the ciphertext $CT$ as follows:

   Use $r$ to compute:
   
   $C_1 = g^r$
   
   $C_2 = M^r$

   $C_3 = \text{Hash}(C_1, C_2, g^{xt}) \oplus (M \mid r)$

   Output the ciphertext $CT = (C_1, C_2, C_3)$.

3. **Decrypt** $(CT, sk)$: Given $sk$ and a ciphertext $CT$, the procedure runs as follows:

   $M \mid r = C_3 \oplus \text{Hash}(C_1, C_2, C_1^r)$

   If $C_1 = g^r$ and $C_2 = M^r$, output $M$; otherwise, return $\bot$.

4. **Test** $(CT_i, CT_j)$: Given $CT_i = (C_{i,1}, C_{i,2}, C_{i,3}), CT_j = (C_{j,1}, C_{j,2}, C_{j,3}$ the procedure runs as follows:

   $T_1 = e(C_{i,1}, C_{j,2})$

   $T_2 = e(C_{i,1}, C_{i,2})$

   Then, check whether $T_1 = T_2$ holds. If yes, it means that $M_i = M_j$ and output 1. Otherwise, it means that $M_i \neq M_j$ and output 0.

2.3. Group ID-Based Encryption

A group ID-based encryption scheme consists of the following six algorithms [42]:

1. **Setup** ($l$): With the security parameter $l$, this procedure exports system public parameters $sp$ and $msk$.

2. **KeyGengroup** ($sp$): With system public parameters $sp$, this procedure exports the public key and secret key $gsk$ of group users.

3. **Extract** ($msk, sp, ID$): With a user’s identity $ID \in \{0, 1\}^*$, this procedure outputs the public key and secret key $dk$ of users.

4. **Join** ($gsk, hID$): This algorithm is an interactive protocol between the group manager and the prospective user; it takes the group user’s ID as inputs, and outputs the group public key $gpk$.

5. **Encrypt** $(M, sp, gpk_i, dk_{ID_i}, ID_i)$: This algorithm takes the public keys $sp, gpk_i$ of the group manager, $dk_{ID_i}$ of the user $i$, and the receiver’s public key $ID_i$ and the message $M$ as inputs, and outputs a ciphertext $CT$. 

(6) **Decrypt** \((CT, gpk, dk_{ID_j})\): This algorithm is run by the receiver; it takes the group public key \(gpk\), the receiver’s secret key \(dk_{ID_j}\), and the ciphertext \(CT\) as inputs, and outputs the message \(M\) or an error symbol \(\perp\).

### 2.4. System Models

Figure 1 illustrates the system model of T-GIBEwET. The system has four roles: the group manager, the users, the tester, and a trusted third party. The trusted third party generates the private key \(dk\) for users. The group manager generates the group public key and group secret key for the group users. The group users encrypt and send the private data to the tester. The tester is authorized and gains a trapdoor \(gtd\).

![Figure 1. System Model.](image-url)

An integrated T-GIBEwET scheme consists of nine algorithms: **Setup**, **KeyGengroup**, **Extract**, **Join**, **Encrypt**, **Decrypt**, **Trace**, **Auth**, and **Test**.

1. **Setup** \((l)\): With the security parameter \(l\), this procedure exports the system public parameters \(sp\) and \(msk\).
2. **KeyGengroup** \((sp)\): With system public parameters \(sp\), this procedure exports the public key and secret key \(gsk\) of group users.
3. **Extract** \((msk, sp, ID)\): With a user’s identity \(ID \in \{0, 1\}^*\), this procedure outputs the public key and secret key \(dk\) of users.
4. **Join** \((gsk, h_{ID})\): This algorithm is an interactive protocol between the group manager and the prospective user; it takes the group user’s \(ID\) as inputs, and outputs the group public key \(gpk\).
5. **Encrypt** \((M, sp, gpk_i, dk_{ID_i}, ID)\): This algorithm takes the public keys \(sp\) and \(gpk_i\) of the group manager, \(dk_{ID_i}\) of the user \(i\), the receiver’s public key \(ID_j\), and the message \(M\) as inputs, and outputs a ciphertext \(CT\).
6. **Decrypt** \((CT, gpk, dk_{ID_j})\): This algorithm is run by the receiver, it takes the group public key \(gpk\), the receiver’s secret key \(dk_{ID_j}\), and the ciphertext \(CT\) as inputs, and outputs the message \(M\) or an error symbol \(\perp\).
7. **Trace** \((CT, gsk, h_{1ID_j}, gpk)\): This algorithm is run by the group manger; it takes group secret key \(gsk\), \(h_{1ID_j}\), \(gpk\), and a ciphertext \(CT\) as inputs, and outputs the user’s \(ID\).
8. **Auth** \((gsk)\): This algorithm is run by the group manger, and outputs the group trapdoor \(gtd\).
9. **Test** \((CT_i, CT_j, gtd)\): This algorithm is run by the tester; it takes the two ciphertexts \(CT_i, CT_j\) and \(gtd\) as inputs, and outputs 1 or 0.
2.5. Security Models

According to different permissions, we show two kinds of adversaries in our proposal.

- **Type $-\alpha_1$ adversary**: With a trapdoor, the adversary cannot recover the plaintext after receiving the challenge ciphertext.
- **Type $-\alpha_2$ adversary**: Without a trapdoor, the adversary cannot tell by which message is CT* encrypted.

**OW-CCA security in T-GIBEwET.**

Type $-\alpha_1$ adversary $A_1$ and simulator $S$’s game is played as in Figure 2.

![Figure 2. OW-CCA security model.](image)

In Figure 2, $O_1$ represents the $H_1, H_2, H_3, H_4$, and $H_5$ queries. $O_2(ID) \triangleq \text{Extract}(\text{msk}, ID)$, $O_3(M, ID, gpk, sp, dk_{ID}) \triangleq \text{Encrypt}(M, ID, gpk, sp, dk_{ID})$, $O_4(ID, CT) \triangleq \text{Decrypt}(dk_{ID}, CT)$, $O_5(gtd, \cdot) \triangleq \text{Auth}(gtd, \cdot)$, $O_6 = O_1$, $O_8(M, ID, gpk, sp, dk_{ID}) = O_3(M, ID, gpk, sp, dk_{ID}) \triangleq \text{Encrypt}(M, ID, gpk, sp, dk_{ID})$, $O_{10}(gtd, \cdot) = O_5(gtd, \cdot) \triangleq \text{Auth}(gtd, \cdot)$, but

$$O_7(i) = \begin{cases} O_2(i) & i \neq t \\ \bot & \text{otherwise} \end{cases}$$

and

$$O_9(i, CT_i) = \begin{cases} O_4(i, CT_i) & CT_i \neq CT^* \\ \bot & \text{otherwise} \end{cases}$$

The advantage of $A_1$ in the aforementioned game is defined as follows:

$$\text{Adv}_{\text{PKEwET-FA-A1}}^{\text{OW-CCA}}(k) = \Pr[M_i = M_i^*]$$

As described in Figure 2, $A_1$ enjoys $O_1$, $O_2$, $O_3$, $O_4$, and $O_5$ queries in Phase 1, and $S$ answers all queries truthfully. When $A_1$ decides to discontinue queries, $S$ selects a
challenge message $M$ and generates the challenge ciphertext $CT^*$. Then, $A_1$ enjoys $O_6$, $O_7$, $O_8$, $O_9$, and $O_{10}$ queries as Phase 1, but the condition is that $CT^*$ does not appear in $O_9$. When $A_1$ decides to discontinue queries, $A_1$ guesses $M'$ to $S$.

**Definition 1.** The T-GIBEwET scheme is OW-CCA security, if all polynomial time and the advantage of $A_1$ ($\text{Adv}_{T\text{-GIBEwET}_A}^{\text{OW-CCA}}(1) = \text{Pr}[M = M']$) is negligible in the above game.

**IND-CCA security in T-GIBEwET.**

Type $\alpha_2$ adversary $A_2$ and simulator $S$’s game is played as in Figure 3.

**Figure 3.** IND-CCA Security Model.

In Figure 3, $O_1$ represents $H_1, H_2, H_3, H_4,$ and $H_5$ queries. $O_2(ID) \triangleq \text{Extract}(msk, ID)$, $O_3(M, ID, gpk, sp, dk_{ID}) \triangleq \text{Encrypt}(M, ID, gpk, sp, dk_{ID})$, $O_4(ID, CT) \triangleq \text{Decrypt}(dk_{ID}, CT)$, $O_5 = O_1$, $O_7(M, ID, gpk, sp, dk_{ID}) = O_3(M, ID, gpk, sp, dk_{ID})$, and $O_8(M, ID, gpk, sp, dk_{ID}) \triangleq \text{Encrypt}(M, ID, gpk, sp, dk_{ID})$.

Let

$$O_6(i) = \begin{cases} O_2(i) & i \neq t \\ \bot & \text{otherwise} \end{cases}$$

and

$$O_8(i, CT_i) = \begin{cases} O_4(i, CT_i) & CT_i \neq CT^* \\ \bot & \text{otherwise} \end{cases}$$

The advantage of $A_2$ in the aforementioned game is defined as follows:

$$\text{Adv}_{\text{IND-CCA}}^{\text{PKEwET-FA}}(k) = |\text{Pr}[b = b^*] - 1/2|)$$

As described in Figure 3, $A_2$ enjoys $O_1$, $O_2$, $O_3$, and $O_4$ queries in Phase 1, and $S$ answers all queries truthfully. When $A_2$ decides to discontinue queries, $A_2$ selects the
two challenge messages $M_0, M_1$. Given $M_0$ and $M_1$, $S$ outputs $CT^*$ based on a random selection of $M_0$ and $M_1$. Then, $A_2$ enjoys $O_5, O_6, O_7$, and $O_8$ queries as Phase 1, but the condition is that $CT^*$ does not appear in $O_6$. When $A_2$ decides to discontinue queries, $A_2$ guesses $b'$ to $S$.

**Definition 2.** The T-GIBEwET scheme is IND-CCA security, if all polynomial time and the advantage of $A_2 (Adv_{T-GIBEwET}^{\text{IND-CCA}}(l)) = |Pr[b = b'] − 1/2|)$ is negligible in the above game.

**Definition 3 (Correctness).** If a $T \rightarrow$ GIBEwET scheme is correct, for any $sp \leftarrow \text{Setup}(l)$, $gsk \leftarrow \text{KeyGengroup}(sp), dk \leftarrow \text{Extract}(msk, sp, ID), gpk \leftarrow \text{Join}(gsk, h_1ID), CT_i \leftarrow \text{Encrypt}(M, sp, gpk_i, dk_{1ID}, ID_j)$, $CT_i \leftarrow \text{Encrypt}(M, sp, gpk_j, dk_{1ID}, ID_j)$ and $gtd \leftarrow \text{Auth}(gsk)$, the following conditions must be satisfied:

1. For any $M \in M$, $\text{Decrypt}(\text{Encrypt}(M, sp, gpk_j, dk_{1ID}, ID_j), dk_{1ID}) = M$ always holds.
2. For any ciphertexts $CT_i$ and $CT_j$, if $\text{Decrypt}(CT_i, dk_{1ID}) = \text{Decrypt}(CT_j, dk_{1ID}) \neq ⊥$, it holds that $\text{Test}(CT_i, CT_j, gtd) = 1$.
3. For any ciphertexts $CT_i$ and $CT_j$, if $\text{Decrypt}(CT_i, dk_{1ID}) \neq \text{Decrypt}(CT_j, dk_{1ID}) \neq ⊥$, it holds that $\text{Test}(CT_i, CT_j, gtd) = 0$.

### 2.6. Symbols

In this paragraph, we summarize some symbols used in the proposed scheme. These symbols will assist readers to read and understand the following sections. These symbols are listed in Table 1.

**Table 1.** Symbols used in the proposed scheme.

| Symbol | Description |
|--------|-------------|
| $l$    | A security parameter |
| $G$    | A cyclic group |
| $g$    | The generator of G |
| $M$    | The plaintext |
| $CT$   | The ciphertext |
| $CT^*$ | The challenge ciphertext |
| $M$    | The message space |
| $Z$    | Set of integers |
| $H$    | A hash function |
| $s$    | The master key (keep it as a secret) |
| $ID$   | A user’s identity |
| $gsk$  | The group secret key (kept as a secret by group manager) |
| $gpk$  | The group public key (share to all users in the group) |
| $dk_{1ID}$ | A user’s secret key (keep it as a secret) |
| $A$    | The adversary |
| $S$    | The simulator |

### 3. Our Constructions

This section provides the proposed T-GIBEwET scheme as follows.

1. **Setup ($l$):** With the security parameter $l$, this procedure exports the system public parameters $sp = (g, G, G_T, e, g^s, H_1, H_2, H_3, H_4, H_5)$. Choose hash functions: $H_i: \{0, 1\}^* \rightarrow G^*$, $H_2: G \rightarrow G$, $H_3: G_T \rightarrow \{0, 1\}^{l+1}$, $H_4, H_5: \{0, 1\}^* \rightarrow \{0, 1\}^l$. Here $l_1$ means the length of elements in $Z_q$. The master key $msk$ is $s$.

2. **KeyGengroup ($sp$):** This procedure randomly selects $s_1, s_2 \in Z_q^*$, and outputs the group secret key $gsk = (s_1, s_2)$.

3. **Extract ($msk, sp$):** With a string $ID \in \{0, 1\}^*$, this procedure outputs the public key and secret key as follows:
The first condition is easy to verify.

(4) **Join** \((gsk, h_{1D})\): This procedure outputs the group public key \(gpk = (h_{1D}^{r_1}, g^{s_1}, g^{s_2})\) for user \(ID\).

(5) **Encrypt** \((M, sp, gpk_i, dk_{1D_j}, ID_j)\): This procedure selects numbers \(r_1, r_2 \in Z_q^\ast\) randomly. Then, it outputs the ciphertext \(CT\) as follows:

- Outputs a secret key \(dk\)
- Outputs a public key \(gsk\)
- Use \(r\) to compute:

\[
\begin{align*}
C_1 &= h_{1D_i}^{r_1} \\
C_2 &= M^2 H_2(U_1^1) \\
C_3 &= g^{s_1} \\
C_4 &= g^{s_2} \\
C_5 &= h_{1D_i}^{r_1} \\
C_6 &= H_3(U_2^1) \oplus (M \parallel r_1) \\
C_7 &= H_5(C_1 \parallel C_2 \parallel C_3 \parallel C_4 \parallel C_5 \parallel C_6 \parallel C_7 \parallel M \parallel r_1) \\
C_8 &= H_4(C_1 \parallel C_2 \parallel C_3 \parallel C_4 \parallel C_5 \parallel C_6 \parallel C_7 \parallel M \parallel r_1).
\end{align*}
\]

Output the ciphertext \(CT = (C_1, C_2, C_3, C_4, C_5, C_6, C_7, C_8)\), where:

\[
\begin{align*}
U_1 &= e(h_{1D_i}^{r_1}, g^{s_1}) \\
U_2 &= e(h_{1D_i}^{r_1}, g^{s_2}).
\end{align*}
\]

(6) **Decrypt** \((CT, dk_{1D_j})\): Given \(dk_{1D_j}\) and a ciphertext \(CT\), the procedure runs as follows:

\[
M \parallel r_1 = C_6 \oplus H_3(e(C_5, h_{1D_i}^{r_1})).
\]

If \(C_1 = h_{1D_i}^{r_1}\) and \(C_8 = H_4(C_1 \parallel C_2 \parallel C_3 \parallel C_4 \parallel C_5 \parallel C_6 \parallel C_7 \parallel M \parallel r_1)\), output \(M\); otherwise, return \(\perp\).

(7) **Trace** \((CT, dk_{1D_j}, sp)\): Given \(dk_{1D_j}, sp\) and a ciphertext \(CT\), the procedure runs as follows:

\[
\begin{align*}
D_1 &= e(g, C_5) \\
D_2 &= e(h_{1D_i}^{r_1}, C_3)
\end{align*}
\]

Then, check whether \(C_7 = H_5(C_1 \parallel C_2 \parallel C_3 \parallel C_4 \parallel C_5 \parallel C_6 \parallel h_{1D_i}^{r_1})\) and \(D_1 = D_2\) holds. If yes, it means that \(CT\) is encrypted by \(ID_j\).

(8) **The algorithm from the authorization function and test function**: Suppose \(CT_i\) (resp. \(CT_j\)) is a ciphertext of \(ID_i\) (resp. \(ID_j\)).

- **Auth**\((gsk)\): Outputs the group trapdoor \(gtd = s_2\).
- **Test**\((CT_i, CT_j, gtd)\):
  This procedure takes the inputs \(CT_i, CT_j\) and \(gtd\) and exports as follows:

\[
\begin{align*}
M_{i}^{r_2} &= C_{i,2} / e(C_{i,1}, g^{s_2})^{s_2} \\
M_{j}^{r_2} &= C_{j,2} / e(C_{j,1}, g^{s_2})^{s_2}
\end{align*}
\]

Use \(M_{i}^{r_2}\) and \(M_{j}^{r_2}\) to decide whether \(e(M_{i}^{r_2}, C_{j,4}) = e(M_{j}^{r_2}, C_{i,4})\). If yes, output 1, which means \(M_i = M_j\). Otherwise, export 0, which means \(M_i \neq M_j\).

**Theorem 1.** According to Definition 3, the above T-GIBEwET scheme is correct.

**Proof.** We show in turn that the three conditions of Definition 3 are all satisfied.

(1) The first condition is easy to verify.
(2) Considering the second condition, for any \( sp \leftarrow \text{Setup}(1^t), gsk \leftarrow \text{KeyGengroup}(sp), dk \leftarrow \text{Extract}(msk, sp, ID) \), \( gpk \leftarrow \text{Join}(gsk, h_{ID}) \), \( CT_i \leftarrow \text{Encrypt}(M, sp, gpk, dk_{ID_1}, ID_1) \), \( CT_j \leftarrow \text{Encrypt}(M, sp, gpk, dk_{ID_1}, ID_1) \), the following equalities hold.

Given a group trapdoor \( gtd = s_2 \) and two ciphertexts \( CT_i = \text{Encrypt} \left( M_i, sp, gpk, dk_{ID_1}, ID_1 \right) \) and \( CT_j = \text{Encrypt} \left( M_j, sp, gpk, dk_{ID_1}, ID_1 \right) \), we can compute as follows:

\[
C_{l2}/e(C_{l1}, g^s) = M_i^{r_1} H_2(e(h_{ID_1}^{s+2}, g^{s+1})/e(h_{ID_1}^{s+2}, g^s))
\]

\[
= M_i^{r_1} H_2(e(h_{ID_1}^{s+2}, g^{s+1})/e(h_{ID_1}^{s+2}, g^s)) = M_i^{r_1}
\]

\[
C_{l2}/e(C_{l1}, g^s) = M_j^{r_2} H_2(e(h_{ID_1}^{s+2}, g^{s+1})/e(h_{ID_1}^{s+2}, g^s))
\]

\[
= M_j^{r_2} H_2(e(h_{ID_1}^{s+2}, g^{s+1})/e(h_{ID_1}^{s+2}, g^s)) = M_j^{r_2}
\]

Use \( M_i^{r_1} \) to compute \( e(M_i^{r_1}, C_{lA}) = e(M_i^{r_1}, g^{s_i}) = e(M_i, g)^{s_i r_1} \).

Use \( M_j^{r_2} \) to compute \( e(M_j^{r_2}, C_{lA}) = e(M_j^{r_2}, g^{s_j}) = e(M_j, g)^{s_j r_2} \).

If \( M_i = M_j \), then \( e(M_i^{r_1}, C_{lA}) = e(M_j^{r_2}, C_{lA}) \), which means \( \text{Test}(CT_i, CT_j, gtd) = 1 \).

(3) As for the third condition, we have the following facts:

As in the above calculation, for any message \( M_i, \text{resp.} M_j \), if \( M_i \neq M_j \), which means \( e(M_i, g)^{s_i r_1} \neq e(M_j, g)^{s_j r_2} \). Then, \( \text{Test}(CT_i, CT_j, gtd) = 0 \).

\[\square\]

4. Security Analysis

This section analyzes the security of the scheme and authorization.

**Theorem 2.** For a type-1 adversary, under the random oracle model, the presented T-GIBEwET scheme is OW-CCA secure.

**Proof.** Let \( A_1 \) be a Type-1 adversary breaking the T-GIBEwET scheme in polynomial time. \( A_1 \) makes at most \( q_{H_1} > 0 \) \( H_1 \) queries, \( q_{H_2} > 0 \) \( H_2 \) queries, \( q_{H_3} > 0 \) \( H_3 \) queries, \( q_{H_4} > 0 \) \( H_4 \) queries, \( q_{key} > 0 \) key retrieval queries, \( q_{enc} > 0 \) encryption queries, and \( q_{dec} > 0 \) decryption queries. We give \( CT^* \) to the simulator \( S \). The aim of \( S \) is to recover the plaintext of \( CT^* \) with a non-negligible advantage.

The game between \( A_1 \) and \( S \) is described as follows:

**Game G1.0**

| **Setup:** | \( S \) runs the algorithm \( \text{Setup}(1^t) \) to create the system parameters \( sp = (g, G, G_T, g^s, e, H_1, H_2, H_3, H_4) \), runs the algorithm \( \text{KeyGengroup}(sp) \) to create a group private key \( gsk = (s_1, s_2) \), runs the algorithm \( \text{Join}(gsk, h_{ID}) \) to create a group public key \( gpk = (h_{ID}^{s_1}, g^{s_1}, s_2) \) for user \( ID \), and runs \( \text{Auth}(gsk) \) to create a group trapdoor \( gtd = s_2 \). Then, \( S \) randomly selects \( ID_1, ID_2 \) as a challenger sender and a challenger receiver, respectively. Then, \( S \) gives the public key and \( ID_1, ID_2 \) to \( A_1 \).

Moreover, the challenger \( S \) prepares the five hash lists \( H_1, H_2, H_3, H_4, H_5 \) to record all hash queries and answer the random oracle queries, where all hash lists are empty at the beginning. If the same input is asked multiple times, the same answer will be returned.

| **Phase 1:** | \( S \) responds to the queries made by \( A_1 \) in the following ways:

| • | \( H_1 \)-query: \( S \) maintains a list of 3-tuples \( (ID_i, a_i, x_i, coin_i) \) in \( H_1 \). When \( A_1 \), ask for \( ID_i \) queries, and \( S \) runs as follows:

| | If the query \( ID_i \) already in the \( H_1 \) list in the form of \( (ID_i, a_i, x_i, coin_i) \), \( S \) outputs \( H_1(ID_i) = a_i \) to \( A_1 \).

| | Otherwise, \( S \) generates \( coin_i \in \{0, 1\} \) randomly. Then, it outputs as follows:

| | | • If \( coin_i = 0 \), \( S \) chooses a random number \( x_i \in Z_q^* \) and computes \( a_i = g^{s_i} \) to \( A_1 \).

| | | • Otherwise, \( S \) computes \( a_i = h_{ID_2}^{r_2} \) to \( A_1 \).
- $S$ adds the tuple $\{ID_i, r_i, \theta_i, x_i, coin_i\}$ into the $H_1$ list.

- $H_2$-query: $S$ maintains a list of 2-tuples $(\theta_i, \phi_i)$ in $H_2$. $S$ chooses $\theta_i \in G$ randomly, returns $\theta_i$ to $A_1$, and adds the tuple $(\theta_i, \phi_i)$ to the $H_2$ list.

- $H_3$-query: $S$ maintains a list of 2-tuples $(\mu_i, \nu_i)$ in $H_3$. $S$ chooses $\nu_i \in \{0,1\}^{l+1}$ randomly, returns $\mu_i$ to $A_1$, and adds the tuple $(\mu_i, \nu_i)$ to the $H_3$ list.

- $H_4$-query: $S$ maintains a list of 2-tuples $(\rho_i, \xi_i)$ in $H_4$. $S$ chooses $\xi_i \in \{0,1\}^l$ randomly, returns $\rho_i$ to $A_1$, and adds the tuple $(\rho_i, \xi_i)$ to the $H_4$ list.

- $H_5$-query: $S$ maintains a list of 2-tuples $(\phi_i, \varphi_i)$ in $H_5$. $S$ chooses $\varphi_i \in \{0,1\}^l$ randomly, returns $\phi_i$ to $A_1$, and adds the tuple $(\phi_i, \varphi_i)$ to the $H_5$ list.

- Extract Query $(ID)$: When inputting $ID_i$, $S$ sends $dk_{ID_i} = a_i$ to $A_1$. If $coin_i = 1$, it means that $ID \neq I D_2$. Then, $S$ sends $\perp$ to $A_1$.

- Encryption Query: $S$ runs an encryption algorithm and outputs $CT = \text{Encrypt}(M, ID, gpk, sp, dk)$.  

- Decryption queries: With the CT to the decryption query, $S$ returns $M = \text{Decrypt}(CT, dk_i)$ to $A_1$ as follows:

  - If $coin_i = 0$, $S$ uses the private key and outputs the decryption query to $A_1$.
  - Otherwise, $S$ outputs $\perp$ to $A_1$.

- Authorization Query: $S$ outputs the group trapdoor $s_2$ to $A_1$.

**Challenge:** $S$ chooses $M^* \in M$ and $r_1, r_2 \in \{0,1\}^l$. It then outputs $CT^*$ as follows:

\[
\begin{align*}
C_1^* &= h_{ID_1}^{s_1 r_1} \\
C_2^* &= M'^* H_2(U_1^1) \\
C_3^* &= g^{r_1} \\
C_4^* &= g^{s_2 r_2} \\
C_5^* &= h_{ID_1}^{s_2 r_2} \\
C_6^* &= H_3(U_2^2) \oplus (M^* \parallel r_1) \\
C_7^* &= H_4(C_1^* \parallel C_2^* \parallel C_3^* \parallel C_4^* \parallel C_5^* \parallel C_6^* \parallel h_{ID_1}^{s_2 r_2}) \\
C_8^* &= H_4(C_1^* \parallel C_2^* \parallel C_3^* \parallel C_4^* \parallel C_5^* \parallel C_6^* \parallel C_7^* \parallel M^* \parallel r_1).
\end{align*}
\]

The ciphertext $CT^*$ is $(C_1^*, C_2^*, C_3^*, C_4^*, C_5^*, C_6^*, C_7^*, C_8^*)$ is output, where:

\[
\begin{align*}
U_1 &= e(h_{ID_1}^{s_1 r_1}, g^{s_2 r_2}) \\
U_2 &= e(h_{ID_2}^{s_2 r_2}, g^{s_2 r_2})
\end{align*}
\]

Finally, it sends $CT^*$ to $A_1$ as the challenge ciphertext.

**Phase 2:** $A_1$ performs the same queries as in **Phase 1**; the constraint is that $CT^*$ does not appear in the decryption queries.

**Guess:** $A_1$ outputs $M^* \in M$ in Game $G_{1,0}$. Then, the advantage is:

\[
\text{Adv}^{OW-CCA}_{T \leftarrow \text{GPKE}_{ET}A_1}(q_{H_1}, q_{H_2}, q_{H_3}, q_{H_4}, q_{H_5}, q_{\text{Extr}}, q_{\text{Enc}}, q_{\text{Dec}}) = Pr[E_{1,0}]
\]

**Game $G_{1,1}$**

**Setup:** $S$ runs the algorithm $\text{Setup}(1^l)$ to create the system parameters $sp = (g, G, G_T, g^s, e, H_1, H_2, H_3, H_4, H_5)$, runs the algorithm $\text{KeyGengroup}(sp)$ to create a group private key $gsk = (s_1, s_2)$, runs the algorithm $\text{Join}(gsk, h_{ID})$ to create group public key $gpk = (h_{ID}^{s_1}, g^{s_1}, g^{s_2})$ for user $ID$, and runs $\text{Auth}(gsk)$ to create the group trapdoor $gtd = s_2$. Then, $S$ randomly selects $ID_1, ID_2$ as a challenger sender and a challenger receiver, respectively. Then, $S$ gives the public key and $ID_1, ID_2$ to $A_1$.

Moreover, the challenger $S$ prepares the five hash lists $H_1, H_2, H_3, H_4, H_5$ to record all hash queries and answer the random oracle queries, where all hash lists are empty at the beginning. If the same input is asked multiple times, the same answer will be returned.
**Phase 1:** $S$ responds to the queries made by $A_1$ in the following ways:

- $H_1$-query ($ID$), $H_2$-query ($\theta_i$), $H_3$-query ($\alpha_i$), $H_4$-query ($\rho_i$), and $H_5$-query ($\phi_i$) are the same in Game $G_{1.0}$.
- Extract Query ($ID$): Same as in Game $G_{1.0}$.
- Encryption Query: $S$ outputs $CT$ to $A_1$ as follows: $S$ chooses $r_1, r_2 \in \{0,1\}^h$ randomly, and performs the $H_1$-query($ID_i$), $H_2$-query($ID_i$) to obtain $\alpha_i$, $\alpha_j$, the $H_2$-query($e(C_1, g_i^\alpha, g_j^\alpha)$) to obtain $\theta_i$, the $H_3$-query($e(\alpha_j, g_i^\alpha)$) to obtain $\nu_i$, the $H_5$-query($C_1 \parallel C_2 \parallel C_3 \parallel C_4 \parallel C_5 \parallel C_6 \parallel C_7 \parallel M \parallel r_1$) to obtain $\xi_i$.

\[
\begin{align*}
C_1 &= \alpha_i^{r_1} \\
C_2 &= M^2 \theta_i \\
C_3 &= g^{r_2} \\
C_4 &= g^{r_2} \\
C_5 &= \alpha_i^{r_1} \\
C_6 &= \nu_i \oplus (M \parallel r_1) \\
C_7 &= \phi_i \\
C_8 &= \xi_i
\end{align*}
\]

$S$ adds $(e(C_1, g_i^\alpha, \theta_i))$ to the $H_2$ list, adds $(e(\alpha_j, g_i^\alpha)^{r_1}, \nu_i)$ to the $H_3$ list, adds $(C_1 \parallel C_2 \parallel C_3 \parallel C_4 \parallel C_5 \parallel C_6 \parallel C_7 \parallel M \parallel r_1, \xi_i)$ to the $H_4$ list, and adds $(C_1 \parallel C_2 \parallel C_3 \parallel C_4 \parallel C_5 \parallel C_6 \parallel h_{ID_i}^{r_1}, \phi_i)$ to the $H_5$ list.

- Decryption queries: With the $CT$ to the decryption query, $S$ returns $M = \text{Decrypt}(CT, dk_i)$ to $A_1$ as follows: $S$ performs the $H_3$($e(\alpha_j, g_i^\alpha)^{r_1}$) to obtain answer $\nu_i$, and performs the $H_4$-query($C_1 \parallel C_2 \parallel C_3 \parallel C_4 \parallel C_5 \parallel C_6 \parallel C_7 \parallel M \parallel r_1$) to obtain answer $\xi_i$. Then, $S$ performs

\[
M \parallel r_1 = C_6 \oplus \nu_i.
\]

Then, it verifies $C_1 = \alpha_i^{r_1}$ and $C_8 = \xi_i$. If the verification fails, it returns $\bot$. Otherwise, $S$ outputs $M$ to $A_1$.

- Authorization Query: Same as in Game $G_{1.0}$.

**Challenge:** $S$ chooses $M^* \subseteq M$, $W \in \{0,1\}^{1+h}$ and $r_1, r_2 \in \{0,1\}^h$. Then, it outputs $CT^*$ as follows:

\[
\begin{align*}
C_1^* &= h_{ID_i}^{r_1} \\
C_2^* &= M^* h_2(U_1^{1}) \\
C_3^* &= g^{r_1} \\
C_4^* &= g^{r_2} \\
C_5^* &= h_{ID_1}^{r_1} \\
C_6^* &= W \oplus (M^* \parallel r_1) \\
C_7^* &= H_5(C_1^* \parallel C_2^* \parallel C_3^* \parallel C_4^* \parallel C_5^* \parallel C_6^* \parallel h_{ID_i}^{r_1}) \\
C_8^* &= H_4(C_1^* \parallel C_2^* \parallel C_3^* \parallel C_4^* \parallel C_5^* \parallel C_6^* \parallel C_7^* \parallel M^* \parallel r_1)
\end{align*}
\]

where $U_1 = e(h_{ID_1}^{r_1}, g^{r_2})$. It outputs the ciphertext $CT^* = (C_1^*, C_2^*, C_3^*, C_4^*, C_5^*, C_6^*, C_7^*, C_8^*)$, and adds $(e(h_{ID_2}, g^\alpha)^{r_1}), W$ into $H_5$.

Finally, it sends $CT^*$ to $A_1$ as the challenge ciphertext.

**Phase 2:** $A_1$ performs the same queries as in **Phase 1**, where the constraint is that $CT^*$ does not appear in the decryption queries.
**Guess:** $A_1$ outputs $M' \subset M$. Let $E_{1.1}$ be the event that $M' = M^*$ in **Game** $G_{1.1}$. Then, the advantage is:

$$Pr[E_{1.1}] = Pr[E_{1.0}].$$

**Game** $G_{1.2}$

**Setup:** $S$ runs the algorithm **Setup**($1^l$) to create the system parameters $sp = (g, G, G_T, g^e, e, H_1, H_2, H_3, H_4)$, runs the algorithm **KeyGengroup**($sp$) to create a group private key $gsk = (s_1, s_2)$, runs the algorithm **Join**($gsk, h_{ID}$) to create the group public key $gpk = (h_{ID}^3, g^{s_1}, g^{s_2})$ for user $ID$, and runs **Auth**($gsk$) to create the group trapdoor $gtd = s_2$. Then, $S$ randomly select $ID_1, ID_2$ as a challenger sender and a challenger receiver, respectively. Then, $S$ gives the public key and $ID_1, ID_2$ to $A_1$.

Moreover, the challenger $S$ prepares the five hash lists $H_1, H_2, H_3, H_4, H_5$ to record all hash queries and answer the random oracle queries, where all hash lists are empty at the beginning. If the same input is asked multiple times, the same answer will be returned.

**Phase 1:** $S$ responds to the queries made by $A_1$ in the following ways:

- The $H_1$-query($ID$), $H_2$-query($\theta$), $H_3$-query($\phi$), and $H_4$-query($\rho$) are the same as in **Game** $G_{1.1}$.
- The $H_5$-query($\mu$) is the same as in **Game** $G_{1.1}$, except that $A_1$ asks $e(C_3, h_{ID_2}^{s_1})$.
- Extract Query($ID$): Same as in **Game** $G_{1.1}$.
- Encryption Query: Same as in **Game** $G_{1.1}$.
- Decryption Queries: Same as in **Game** $G_{1.1}$.
- Authorization Query: Same as in **Game** $G_{1.1}$.

**Challenge:** $S$ chooses $M' \subset M, W^* \in \{0, 1\}^{l+1}$ and $r_1, r_2 \in \{0, 1\}^l$. Then, it outputs $CT^*$ as follows:

$$
C_1^* = h_{ID_1}^{s_{1r_1}} \\
C_2^* = M'^* H_2(U_1^1) \\
C_3^* = g^{r_1} \\
C_4^* = g^{s_{2r_2}} \\
C_5^* = h_{ID_1}^{s_{1r_2}} \\
C_6^* = W^* \\
C_7^* = H_4(C_1^* \parallel C_2^* \parallel C_3^* \parallel C_4^* \parallel C_5^* \parallel C_6^* \parallel h_{ID_1}^{s_{1r_2}}).
$$

where $U_1 = e(h_{ID_2}^{r_1}, g^{s_{2r_2}})$. It outputs the ciphertext $CT^* = (C_1^*, C_2^*, C_3^*, C_4^*, C_5^*, C_6^*, C_7^*)$, and adds $(e(h_{ID_2}^{r_1}, g^{s_{2r_2}}), W^* \oplus (M'^* \parallel r_1))$ into $H_3$.

Finally, it sends $CT^*$ to $A_1$ as the challenge ciphertext.

**Phase 2:** $A_1$ performs the same queries as in **Phase 1**, where the constraint is that $CT^*$ does not appear in the decryption Queries, and if $A_1$ asks for the decryption of $CT^* = (C_1^*, C_2^*, C_3^*, C_4^*, C_5^*, C_6^*, C_7^*)$, where $C_6^* \neq C_0^*$, $S$ outputs $\perp$.

**Guess:** $A_1$ outputs $M' \subset M$.

Let $E_{1.2}$ be the event that $M' = M^*$ in **Game** $G_{1.2}$. Because $C_6^*$ is a random value in **Game** $G_{1.1}$ and **Game** $G_{1.2}$, the challenge ciphertexts generated in **Game** $G_{1.1}$ and **Game** $G_{1.2}$ follow the same distribution. Therefore, if the event $E_1$ does not occur, **Game** $G_{1.2}$ is identical to **Game** $G_{1.1}$, and we can figure out

$$|Pr[E_{1.2}] - Pr[E_{1.1}]| \leq Pr[E_1].$$

Next, we show that the probability of event $E_1$ occurring in **Game** $G_{1.2}$ is negligible.
Lemma 1. When the C-BDH problem is intractable, there is a negligible probability that the event $E_1$ happens in Game $G_{1.2}$.

Proof. Suppose that $Pr[E_1]$ is non-negligible; we can construct a simulator $S$ to break the C-BDH assumption by using $A_1$’s attacks. With the tuple $(c, G, G_T, g^e, g^f, g^d)$, the aim is to obtain $e(g, g)^{aad}$.

Setup: $S$ randomly selects $ID_1, ID_2$ as a challenger sender and a challenger receiver, respectively. Then, $S$ gives the public key and $ID_1, ID_2$ to $A_1$. $S$ runs the algorithm $Setup(1^l)$ to create the system parameters $sp = (g, G, G_T, g^e, c, H_1, H_2, H_3, H_4, H_5)$, runs the algorithm $KeyGenGroup(sp)$ to create a group private key $gsk = (s_1, s_2)$, runs the algorithm $Join(gsk, h_{1D})$ to create the group public key $gpk = (h_{1D}^s, g^{s_1}, g^{s_2})$ for user $ID$, and runs $Auth(gsk)$ to create the group trapdoor $g^{lid} = s_2$.

Phase 1: $S$ responds to the queries made by $A_1$ in the following ways:

- $H_1$-query($ID$), $H_2$-query($\theta_i$), $H_3$-query($\phi_i$), and $H_4$-query($\rho_i$) are same as in Game $G_{1.1}$.
- $H_5$-query($\mu_i$) is same as in Game $G_{1.1}$, except that $A_1$ asks $e(C_3, h_{1D_1}^s)$.
- Extract Query($ID$): Same as in Game $G_{1.1}$.
- Encryption Query: Same as in Game $G_{1.1}$, except that for the query ($ID_2, *, *), S$ selects $r_1, r_2 \in \{0, 1\}^l$ randomly and outputs a ciphertext $CT = (C_1, C_2, C_3, C_4, C_5, C_6, C_7, C_8)$ as follows:
  
  $S$ performs the $H_1$-query($ID_i$) and $H_1$-query($ID_i$) to obtain $\alpha_i$ and $\alpha_j$, respectively, the $H_2$-query($\alpha_i, g^{s_2}$) to obtain $\theta_i$, the $H_3$-query($\alpha_j, g^{s_2}$) to obtain $\phi_i$, and the $H_4$-query($\alpha_i, C_3 \parallel C_4 \parallel C_5 \parallel C_6 \parallel h_{1D_1}^s$) to obtain $\rho_i$, and the $H_4$-query($\alpha_i, C_3 \parallel C_4 \parallel C_5 \parallel C_6 \parallel C_7 \parallel M \parallel r_1$) to obtain $\xi_i$.
  
  $C_1 = \alpha_i^{s_1^{r_1}}$
  $C_2 = M^{r_2}\theta_i$
  $C_3 = g^{s_2}$
  $C_4 = g^{s_2}$
  $C_5 = \alpha_j^{s_2}$
  $C_6 = v_i \oplus (M \parallel r_1)$
  $C_7 = \varphi_i$
  $C_8 = \xi_i$

  $S$ adds $(e(C_1, g^{s_2}), \theta_i)$ to the $H_2$ list, adds $(e(\alpha_j, g^{s_2}), v_i)$ to the $H_3$ list, and adds $(C_1 \parallel C_2 \parallel C_3 \parallel C_4 \parallel C_5 \parallel C_6 \parallel C_7 \parallel M \parallel r_1, \xi_i)$ to the $H_4$ list.

- Decryption queries: Same as in Game $G_{1.1}$.
- Authorization Query: Same as in Game $G_{1.1}$.

Challenge: $S$ chooses $M^* \in M, W^* \in \{0, 1\}^{l+l_1}$ and $r_1, r_2 \in \{0, 1\}^{l_1}$. Then, it outputs $CT^*$ as follows:

  $C_1^* = h_{1D_1}^{s_1^{r_1}}$
  $C_2^* = M^{r_2}H_2(U_1^{r_2})$
  $C_3^* = g^{r_1}$
  $C_4^* = g^{s_2}$
  $C_5^* = h_{1D_1}^{s_2}$
  $C_6^* = W^*$
  $C_7^* = H_5(C_1^* \parallel C_2^* \parallel C_3^* \parallel C_4^* \parallel C_5^* \parallel C_6^* \parallel h_{1D_1}^{r_1})$. 

Under the random oracle model, the proposed T-GIBEwET scheme is IND-CCA secure (Theorem 3).

Proof. Let $A_2$ be a type-2 adversary breaking the T-GIBEwET scheme in polynomial time. $A_2$ makes at most $q_{ID_1} > 0$ $H_1$-queries, $q_{H_2} > 0$ $H_2$-queries, $q_{H_3} > 0$ $H_3$-queries, $q_{H_4} > 0$ $H_4$-queries, $q_{H_5} > 0$ $H_5$-queries, $q_{Key} > 0$ key retrieve queries, $q_{Enc} > 0$ encryption queries, and $q_{Dec} > 0$ decryption queries. We give $C^*$ to the simulator $S$. The aim of $S$ is to recover the plaintext of $C^*$ with a non-negligible advantage.

The game between $A_2$ and $S$ is described as follows:

**Game $G_{2,0}$**

**Setup:** $S$ runs the algorithm $Setup(1^l)$ to create the system parameters $sp = (g, G, C_T, g^s, c, H_1, H_2, H_3, H_4, H_5)$, runs the algorithm $KeyGengroup(sp)$ to create a group private key $gsk = (s_1, s_2)$, runs the algorithm $Join(gsk, h_{ID})$ to create the group public key $gpk = (h_{ID}^1, g^a, s^{612})$ for user $ID$, and runs $Auth(gsk)$ to create the group trap-door $gtd = s_2$. Then, $S$ randomly selects $ID_1, ID_2$ as a challenger sender and a challenger receiver, respectively. Then, $S$ gives the public key and $ID_1, ID_2$ to $A_2$.

Moreover, the challenger $S$ prepares the five hash lists $H_1, H_2, H_3, H_4, H_5$ to record all hash queries and answer the random oracle queries, where all hash lists are empty at the beginning. If the same input is asked multiple times, the same answer will be returned.

**Phase 1:** $S$ responds to the queries made by $A_2$ in the following ways:

- $H_1$-query: $S$ maintains a list of 3-tuples $(ID_i, a_i, x_i, coin_i)$ in $H_1$. When $A_2$ asks for $ID_i$ queries, $S$ runs as follows:
  - If the query $ID_i$ is already in the $H_1$ list in the form of $(ID_i, a_i, x_i, coin_i)$, $S$ outputs $H_1(ID_i) = a_i \in G^*$ to $A_2$.
  - Otherwise, $S$ generates $coin_i \in \{0, 1\}$ randomly. Then, it outputs as follows:
    - If $coin_i = 0$, $S$ chooses a random number $x_i \in Z_q^*$ and computes $a_i = g^{x_i}$ to $A_2$.
    - Otherwise, $S$ computes $a_i = h_{ID}^1$ to $A_2$.

- $H_2$-query: $S$ maintains a list of 2-tuples $(\theta_i, \theta_i)$ in $H_2$. $S$ chooses $\theta_i \in G$ randomly, puts out $\theta_i$ to $A_2$ and adds the tuple $(\theta_i, \theta_i)$ to the $H_2$ list.

- $H_3$-query: $S$ maintains a list of 2-tuples $(\mu_i, \nu_i)$ in $H_3$. $S$ chooses $v_i \{0, 1\}^{l+1}$ randomly, puts out $v_i$ to $A_2$ and adds the tuple $(\mu_i, \nu_i)$ to the $H_3$ list.

- $H_4$-query: $S$ maintains a list of 2-tuples $(\rho_i, \xi_i)$ in $H_4$. $S$ chooses $\xi_i \{0, 1\}^l$ randomly, puts out $\rho_i$ to $A_2$ and adds the tuple $(\rho_i, \xi_i)$ to the $H_4$ list.

- $H_5$-query: $S$ maintains a list of 2-tuples $(\phi_i, \psi_i)$ in $H_5$. $S$ chooses $\psi_i \{0, 1\}^l$ randomly, returns $\phi_i$ to $A_1$ and adds the tuple $(\phi_i, \psi_i)$ to the $H_5$ list.

- Extract Query($ID$): On input of the $ID_i$, $S$ sends $dk_{ID_i} = a_i$ to $A_2$. If $coin_i = 1$, which means $ID \neq ID_2$, then $S$ sends $\perp$ to $A_2$.

- Encryption Query: $S$ runs the encryption algorithm and outputs $CT = Encrypt(M, gpk, dk, sp)$.

- Decryption queries: With the $CT$ in the decryption query, $S$ returns $M = Decrypt(CT, dk_i)$ to $A_2$ as follows:
  - If $coin_i = 0$, $S$ uses the private key and outputs the decryption query to $A_2$. 

$$C^*_{s} = H_4(C^*_1 \parallel C^*_2 \parallel C^*_3 \parallel C^*_4 \parallel C^*_5 \parallel C^*_6 \parallel C^*_7 \parallel M^* \parallel r_1),$$

where $U_1 = e(h_{ID}^1, g^{s^{12}})$. It outputs the ciphertext $CT^* = (C^*_1, C^*_2, C^*_3, C^*_4, C^*_5, C^*_6, C^*_7, C^*_8)$, and adds $(e(h_{ID}^2, g^{s^{12}}), W^* \oplus (M^* \parallel r_1))$ into $H_3$.

Finally, it sends $CT^*$ to $A_1$ as the challenge ciphertext.

**Phase 2:** $A_1$ performs the same queries as in **Phase 1**: the constraint is that $CT^*$ does not appear in the decryption queries, and if $A_1$ asks for the decryption of $CT^* = (C^*_1, C^*_2, C^*_3, C^*_4, C^*_5, C^*_6, C^*_7, C^*_8)$, where $C^*_6 \neq C^*_6$, $S$ outputs $\perp$. 

**Guess:** $A_1$ outputs $M^* \in M$. □
- Otherwise, S outputs $\perp$ to $A_2$.

- **Authorization Query**: It is not allowed.

  **Challenge**: $A_2$ chooses $M_0, M_1 \subset \mathcal{M}$ randomly and sends them to $S$. Then, $S$ takes $b \in \{0, 1\}$ and $r^*_1, r^*_2 \in \{0, 1\}^l$. It then outputs $CT^*$ as follows:

  $$C_1^* = h^{s_1}_{ID_1}$$
  $$C_2^* = M_b H_2(U^*_1)$$
  $$C_3^* = g^{r_1}$$
  $$C_4^* = g^{s_1^2}$$
  $$C_5^* = h_g^{r_1}$$
  $$C_6^* = H_3(U^*_1) \oplus (M_b \parallel r_1)$$
  $$C_7^* = H_5(C_1^* \parallel C_2^* \parallel C_3^* \parallel C_4^* \parallel C_5^* \parallel h_{ID_1}^{s_1^2})$$
  $$C_8^* = H_4(C_1^* \parallel C_2^* \parallel C_3^* \parallel C_4^* \parallel C_5^* \parallel C_6^* \parallel C_7^* \parallel M_b \parallel r_1).$$

  Output the ciphertext $CT^* = (C_1^*, C_2^*, C_3^*, C_4^*, C_5^*, C_6^*, C_7^*, C_8^*)$, where:

  $U_1 = e(h_{ID_1}^* g^{s_1})$
  $U_2 = e(h_{ID_1}^* g^*)$

  Finally, it sends $CT^*$ to $A_2$ as the challenge ciphertext.

  **Phase 2**: $A_2$ performs the same queries as in **Phase 1**, where the constraint are as follows:

  - $CT^*$ does not appear in the decryption queries.
  - In the authorization query, all of the group users cannot be authorized.

  **Guess**: $A_2$ outputs $b^* \in \{0, 1\}$. Let $E_{2, 0}$ be the event that $b = b^*$ in Game $G_{2, 0}$. Then, the advantage is:

  $$Adv^{OW-CCA}_{G_{2, 0}}(\mathcal{T} - GPKE_{-ET}, A_2) = Pr[E_{2, 0}]$$

  **Game $G_{2, 1}$**

  **Setup**: $S$ runs the algorithm $\text{Setup}(1^l)$ to create the system parameters $sp = (g, G, g_T, g^2, e, H_1, H_2, H_3, H_4)$, runs the algorithm $\text{KeyGengroup}(sp)$ to create a group private key $gsk = (s_1, s_2)$, runs the algorithm $\text{Join}(gsk, h_{ID})$ to create the group public key $gpk = (h_{ID}^* g^{s_1}, g^{s_1^2})$ for user $ID$, and runs $\text{Auth}(gsk)$ to create the group trapdoor $gtd = s_2$. Then, $S$ randomly selects $ID_1, ID_2$ as a challenger sender and a challenger receiver, respectively. Then, $S$ gives the public key and $ID_1, ID_2$ to $A_2$.

  Moreover, the challenger $S$ prepares the four hash lists $H_1, H_2, H_3, H_4$ to record all hash queries and answer the random oracle queries, where all hash list are empty at the beginning. If the same input is asked multiple times, the same answer will be returned.

  **Phase 1**: $S$ responds to the queries made by $A_2$ in the following ways:

  - $H_1$-query($ID$), $H_2$-query($\theta_i$), $H_3$-query($\mu_i$), $H_5$-query($\phi_i$), and $H_4$-query($\rho_i$) are the same as in Game $G_{2, 0}$.
  - Extract Query($ID$): Same as in Game $G_{2, 0}$.
  - Encryption Query: $S$ outputs $CT$ to $A_2$ as follows:
    $S$ chooses $r_1, r_2 \in \{0, 1\}^l$ randomly, and performs the $H_1$-query($ID$) and $H_1$-query($ID$) to obtain $\alpha_i$ and $\alpha_i$, respectively, the $H_2$-query($e(C_1, g^s)^{\nu_i}$) to obtain $\theta_i$, the $H_3$-query($e(\alpha_i, g^s)^{\nu_i}$) to obtain $\nu_i$, the $H_5$-query($C_1 \parallel C_2 \parallel C_3 \parallel C_4 \parallel C_5 \parallel C_6 \parallel h_{ID_1}^{s_1^2}$) to obtain $\phi_i$, and the $H_4$-query($C_1 \parallel C_2 \parallel C_3 \parallel C_4 \parallel C_5 \parallel C_6 \parallel C_7 \parallel M \parallel r_1$) to obtain $\xi_i$.

  $$C_1 = \alpha_i^{s_1^2}$$
\[ C_2 = M^{\theta_1} \]
\[ C_3 = g^{\theta_2} \]
\[ C_4 = g^{s_{r_1}} \]
\[ C_5 = h_{r_1} \]
\[ C_6 = v_i \oplus (M \parallel r_1) \]
\[ C_7 = \psi_i \]
\[ C_8 = \xi_i. \]

\( S \) adds \((e(C_1, g^\theta), \theta_1)\) to the \( H_2 \) list, adds \((e(\alpha_i, g^\theta)^{s_{r_1}}, v_i)\) to the \( H_3 \) list, adds \((C_1 || C_2 || C_3 || C_4 || C_5 || C_6 || M \parallel \xi_i)\) to the \( H_4 \) list, and adds \((C_1 || C_2 || C_3 || C_4 || C_5 || C_6 || M \parallel r_1)\) to the \( H_5 \) list.

- **Decryption queries:** With the CT to the decryption query, \( S \) returns \( M = \text{Decrypt}(CT, sk_i) \) to \( A_2 \) as follows: \( S \) performs the \( H_5(e(\alpha_i, g^\theta)^{s_{r_1}}) \) to obtain answer \( v_i \), and performs the \( H_4\)-query\((C_1 || C_2 || C_3 || C_4 || C_5 || C_6 || M \parallel M \parallel r_1)\) to obtain answer \( \xi_i \). Then, \( S \) performs

\[ M \parallel r_1 = C_6 \oplus v_i. \]

Then, \( C_1 = \alpha_i^{s_{r_1}} \) and \( C_8 = \xi_i \) are verified. If the verification fails, it returns \( \bot \). Otherwise, \( S \) outputs \( M \) to \( A_2 \).

- **Authorization Query:** It is not allowed.

**Challenge:** \( A_2 \) chooses \( M_0, M_1 \subset M \) randomly and sends them to \( S \). Then, \( S \) takes \( b \in \{0, 1\} \) and \( W \in \{0, 1\}^{l+1} \) and \( r_1^*, r_2^* \in \{0, 1\}^h \). It then outputs \( CT^* \) as follows:

\[ C_1^* = h_{ID_1}^{s_{r_1}} \]
\[ C_2^* = M_0^2 H_2(U_1) \]
\[ C_3^* = g^{s_{r_1}} \]
\[ C_4^* = g^{s_{r_2}} \]
\[ C_5^* = h_{ID_1}^{s_{r_1}} \]
\[ C_6^* = W \oplus (M_0 \parallel r_1) \]
\[ C_7^* = H_4(C_1^* || C_2^* || C_3^* || C_4^* || C_5^* || C_6^* || h_{ID_1}^{s_{r_1}}) \]
\[ C_8^* = H_3(C_1^* || C_2^* || C_3^* || C_4^* || C_5^* || C_6^* || C_7^* || M_0 \parallel r_1) \]

where \( U_1 = e(h_{ID_1}^{s_{r_1}}, g^{s_{r_2}}) \). It outputs the ciphertext \( CT^* = (C_1^*, C_2^*, C_3^*, C_4^*, C_5^*, C_6^*, C_7^*, C_8^*) \), and adds \((e(h_{ID_2}^{s_{r_1}}, g^{s_{r_2}}), W)\) into \( H_2 \).

Finally, it sends \( CT^* \) to \( A_2 \) as the challenge ciphertext.

**Phase 2:** \( A_2 \) performs the same queries as in **Phase 1**; the constraint are as follows:

- **CT** does not appear in the decryption queries.
- In the authorization query, all of the group users cannot be authorized.

**Guess:** \( A_2 \) outputs \( b^* \in \{0, 1\} \).

Let \( E_{2,1} \) be the event that \( b = b^* \) in **Game G_{2,1}**. Then, the advantage is

\[ Pr[E_{2,1}] = Pr[E_{2,0}]. \]

**Game G_{2,2}**

**Setup:** \( S \) runs the algorithm **Setup**\((1^l)\) to create the system parameters \( sp = (g, G, g_T, g^\theta, e, H_1, H_2, H_3, H_4, H_5) \), runs the algorithm **KeyGengroup**(\( sp \)) to create a group private key \( sk = (s_1, s_2) \), runs the algorithm **Join**(\( sk, h_{ID} \)) to create the group public key \( gpk = (h_{ID}^{s_1}, g^{s_2}) \) for user \( ID \), and runs **Auth**(\( sk \)) to create the group trap-
door \( gtd = s_2 \). Then, \( S \) randomly selects \( ID_1, ID_2 \) as a challenger sender and a challenger receiver, respectively. Then, \( S \) gives the public key and \( ID_1, ID_2 \) to \( A_2 \).

Moreover, the challenger \( S \) prepares the five hash lists \( H_1, H_2, H_3, H_4, H_5 \) to record all hash queries and answers the random oracle queries, where all hash list are empty at the beginning. If the same input is asked multiple times, the same answer will be returned.

**Phase 1:** \( S \) responds to the queries by \( A_2 \) in the following ways:

- \( H_1 \)-query(ID), \( H_2 \)-query(\( \theta_i \)), \( H_3 \)-query(\( \phi_i \)), and \( H_4 \)-query(\( \rho_i \)) are the same as in Game \( G_{2,1} \).
- \( H_5 \)-query(\( \mu_i \)) is the same as in Game \( G_{2,1} \), except that \( A_2 \) asks \( e(h_1^{ID_2}) \).
- Extract Query(ID): Same as in Game \( G_{2,1} \).
- Encryption Query: Same as in Game \( G_{2,1} \).
- Decryption Queries: Same as in Game \( G_{2,1} \).
- Authorization Query: Same as in Game \( G_{2,1} \).

**Challenge:** \( A_2 \) chooses \( M_0, M_1 \subset M \) randomly and sends them to \( S \). Then, \( S \) takes \( b \in \{0,1\} \), \( W^* \in \{0,1\}^{l+1} \) and \( r_1, r_2 \in \{0,1\}^l \). It then outputs \( CT^* \) as follows:

\[
\begin{align*}
C_1^* &= h_{ID_1}^{r_1} \\
C_2^* &= M_b^2 H_2(U_1^{r_1}) \\
C_3^* &= g^{r_1} \\
C_4^* &= g^{s r_2} \\
C_5^* &= h_{ID_1}^{r_2} \\
C_6^* &= W^* \\
C_7^* &= H_3(C_1^* \parallel C_2^* \parallel C_3^* \parallel C_4^* \parallel C_5^* \parallel C_6^* \parallel l_{ID_1}^*) \\
C_8^* &= H_4(C_1^* \parallel C_2^* \parallel C_3^* \parallel C_4^* \parallel C_5^* \parallel C_6^* \parallel C_7^* \parallel M_b \parallel r_1)
\end{align*}
\]

where \( U_1 = e(h_{ID_1}^{r_1}, g^{s r_2}) \). It outputs the ciphertext \( CT^* = (C_1^*, C_2^*, C_3^*, C_4^*, C_5^*, C_6^*, C_7^*, C_8^*) \), and adds \( e(h_{ID_2}, g^{s r_1})W^* \parallel (M^* \parallel r_1) \) into \( H_3 \).

Finally, it sends \( CT^* \) to \( A_2 \) as the challenge ciphertext.

**Phase 2:** \( A_2 \) performs the same queries as in **Phase 1**, where the constraint is that \( CT^* \) does not appear in the decryption queries, and if \( A_2 \) asks for the decryption of \( CT^* = (C_1^*, C_2^*, C_3^*, C_4^*, C_5^*, C_6^*, C_7^*, C_8^*) \), where \( C_6^* \neq C_6^, S \) outputs \( \bot \).

**Guess:** \( A_2 \) outputs \( b^* \in \{0,1\} \). \( \Box \)

Let \( E_{2,2} \) be the event that \( b = b^* \) in Game \( G_{2,2} \).

Because \( C_6^* \) is a random value in Game \( G_{2,1} \) and Game \( G_{2,2} \), the challenge ciphertexts generated in Game \( G_{2,1} \) and Game \( G_{2,2} \) follow the same distribution. Therefore, if the event \( E_2 \) does not occur, Game \( G_{2,2} \) is identical to Game \( G_{2,1} \). And we can figure out that

\[
|Pr[E_{2,2}] - Pr[E_{2,1}]| \leq Pr[E_2].
\]

Next, we show that the probability of event \( E_2 \) occurring in Game \( G_{2,2} \) is negligible.

**Lemma 2.** When the C-BDH problem is intractable, there is negligible probability that the event \( E_2 \) will happen in Game \( G_{2,2} \).

**Proof.** Suppose that \( Pr[E_2] \) is non-negligible; we can construct a simulator \( S \) to break the C-BDH assumption by using the \( A_2 \)'s attacks. With the tuple \( (e, G, G_T, g^a, g^b, g^c, g^d) \), the aim is to obtain \( e(g, g)^{ac} \).

**Setup:** \( S \) randomly select \( ID_1, ID_2 \) as a challenger sender and a challenger receiver, respectively. Then, \( S \) gives the public key and \( ID_1, ID_2 \) to \( A_2 \). \( S \) runs the algorithm \( \text{Setup}'(1') \) to create the system parameters \( sp = (g, G, G_T, g^a, e, H_1, H_2, H_3, H_4, H_5) \), runs the algorithm \( \text{KeyGengroup}(sp) \) to create a group private key \( gsk = (s_1, s_2) \), runs the
algorithm $\text{Join}(gsk, h_{ID})$ to create the group public key $gpk = (h_{ID}^{s_{1}}, g^{s_{2}})$ for user $ID$, and runs $\text{Auth}(gsk)$ to create the group trapdoor $gId = s_{2}$.

**Phase 1:** $S$ responds to the queries made by $A_{2}$ in the following ways:

- $H_{1}$-query($ID$), $H_{2}$-query($\theta_{i}$), $H_{5}$-query($\phi_{i}$), and $H_{4}$-query($\rho_{i}$) are the same as in $\text{Game } G_{2.1}$.
- $H_{3}$-query($\mu_{i}$) is the same as in $\text{Game } G_{2.1}$, except that $A_{2}$ asks for $e(C_{3}, h_{ID}^{s_{1}})$.
- Extract Query ($ID$): Same as in $\text{Game } G_{2.1}$.
- Encryption Query: Same as in $\text{Game } G_{2.1}$, except that for the query $(ID_{2}, *, *)$, $S$ selects $r_{1}, r_{2} \in \{0,1\}^{l}$ randomly and outputs a ciphertext $CT = (C_{1}, C_{2}, C_{3}, C_{4}, C_{5}, C_{6}, C_{7}, C_{8})$ as follows:
  
  $S$ performs the $H_{1}$-query($ID$) and $H_{2}$-query($ID$) to obtain $\alpha_{i}$ and $\alpha_{j}$, respectively, the $H_{2}$-query($e(C_{1}, g^{s_{2}})$) to obtain $\theta_{i}$, the $H_{3}$-query($e(a_{j}, g^{s_{1}})$) to obtain $\nu_{i}$, the $H_{5}$-query($C_{1} || C_{2} || C_{3} || C_{4} || C_{5} || C_{6} || h_{ID}^{s_{1}}$) to obtain $\varphi_{i}$, and the $H_{4}$-query($C_{1} || C_{2} || C_{3} || C_{4} || C_{5} || C_{6} || C_{7} || M || r_{1}$) to obtain $\xi_{j}$.

  \[
  \begin{align*}
  C_{1} &= \alpha_{i}^{s_{1}r_{1}} \\
  C_{2} &= M_{1}^{\gamma} \theta_{i} \\
  C_{3} &= g^{s_{2}} \\
  C_{4} &= g^{s_{1}} \\
  C_{5} &= \alpha_{j}^{s_{1}r_{1}} \\
  C_{6} &= \nu_{i} \oplus (M \parallel r_{1}) \\
  C_{7} &= \varphi_{i} \\
  C_{8} &= \xi_{i}.
  \end{align*}
  \]

  $S$ adds $(e(C_{1}, g^{s_{2}}), \theta_{i})$ to the $H_{2}$ list, adds $(e(a_{j}, g^{s_{1}}), \nu_{i})$ to the $H_{3}$ list, adds $(C_{1} || C_{2} || C_{3} || C_{4} || C_{5} || C_{6} || C_{7} || M || r_{1}, \xi_{j})$ to the $H_{4}$ list, and adds $(C_{1} || C_{2} || C_{3} || C_{4} || C_{5} || C_{6} || h_{ID}^{s_{1}})$ to the $H_{5}$ list.

- Decryption Queries: Same as in $\text{Game } G_{2.1}$.

**Challenge:** $A_{2}$ chooses $M_{0}, M_{1} \subset M$ randomly and sends them to $S$. Then, $S$ takes $b \in \{0,1\}$, $W^{*} \in \{0,1\}^{l+l_{1}}$, and $r_{1}, r_{2} \in \{0,1\}^{l}$. Then, it outputs $CT^{*}$ as follows:

  \[
  \begin{align*}
  C_{1}^{*} &= h_{ID}^{s_{1}r_{1}} \\
  C_{2}^{*} &= M_{b}^{s_{1}H_{2}(U_{1}^{r_{1}})} \\
  C_{3}^{*} &= g^{s_{2}} \\
  C_{4}^{*} &= g^{s_{1}} \\
  C_{5}^{*} &= h_{ID}^{s_{1}} \\
  C_{6}^{*} &= W^{*} \\
  C_{7}^{*} &= H_{3}(C_{1}^{*} || C_{2}^{*} || C_{3}^{*} || C_{4}^{*} || C_{5}^{*} || C_{6}^{*} || h_{ID}^{s_{1}}). \\
  C_{8}^{*} &= H_{4}(C_{1}^{*} || C_{2}^{*} || C_{3}^{*} || C_{4}^{*} || C_{5}^{*} || C_{6}^{*} || C_{7}^{*} || M_{b} || r_{1})
  \end{align*}
  \]

where $U_{1} = e(h_{ID}^{s_{1}}, g^{s_{2}})$. It outputs the ciphertext $CT^{*} = (C_{1}^{*}, C_{2}^{*}, C_{3}^{*}, C_{4}^{*}, C_{5}^{*}, C_{6}^{*}, C_{7}^{*}, C_{8}^{*})$, and adds $(e(h_{ID}^{s_{1}}, g^{s_{2}}), W^{*} \oplus (M_{b} \parallel r_{1}))$ into $H_{3}$.

Finally, it sends $CT^{*}$ to $A_{2}$ as the challenge ciphertext.

**Phase 2:** $A_{2}$ performs the same queries as in **Phase 1**, where the constraint is that $CT^{*}$ does not appear in the decryption queries, and if $A_{2}$ asks for the decryption of $CT^{*} = (C_{1}^{*}, C_{2}^{*}, C_{3}^{*}, C_{4}^{*}, C_{5}^{*}, C_{6}^{*}, C_{7}^{*}, C_{8}^{*})$, where $C_{6}^{*} \neq C_{6}^{*}$, $S$ outputs ⊥.

**Guess:** $A_{2}$ outputs $b^{*} \in \{0,1\}$. □
5. Performance Comparison

In this section, a performance comparison between the presented T-GIBEwET scheme and other related schemes is discussed. As illustrated in Table 2, our proposal supports the traceability function and others do not. In Table 3, the comparison of efficiency with PKEwET variants is shown. The second to sixth columns reveal the computational efficiency for the algorithms of encryption, decryption, authorization, testing, and tracing. Compared to [7,16,17,35], the proposed T-GIBEwET scheme is more efficient than [7,16,17] in the decryption algorithm and more efficient than [17] in the authorization algorithm. Both authorization and tracking are supported in this paper.

Table 2. Comparison with other schemes.

| Scheme | Authorized | Ciphertext Test | Traceable |
|--------|------------|-----------------|-----------|
| [7]    | -          | √               | -         |
| [16]   | √          | √               | -         |
| [17]   | √          | √               | -         |
| [35]   | √          | √               | -         |
| T-GIBEwET | √      | √               | √         |

Table 3. Comparison of efficiency with other schemes.

| Scheme   | $C_{Enc}$ | $C_{Dec}$ | $C_{Auth}$ | $C_{Test}$ | $C_{Trac}$ |
|----------|-----------|-----------|------------|------------|------------|
| [7]      | 3E        | 3E        | -          | 2P         | -          |
| [16]     | 5E+2P     | 2E+2P     | 0          | 4P         | -          |
| [17]     | (n+2)E+2P | (n+1)E+P  | nE         | nP         | -          |
| [35]     | 5E        | 2E        | 0          | 2E+2P     | -          |
| [34]     | 4E        | 2E        | 0          | 2E         | -          |
| T-GIBEwET | 7E+2P     | E+P       | 0          | 2E+4P     | 2P         |

$E$ and $P$ are the exponentiation operation and the the pairing operation, respectively, in group $G$.

6. Conclusions

In this paper we analyzed the PKEwET scheme, pointed out that the PKEwET algorithm is unable to keep track of ciphertexts in the cloud sever, and proposed the a traceable group ID-based encryption with an equality test scheme (T-GIBEwET). The T-GIBEwET algorithm is endowed with a special function: the users who are authorized by a trapdoor can test the ciphertexts in the cloud sever. Moreover, the proposed scheme supports the traceability function.

To simplify the public key management mechanism, the proposed scheme was designed with ID-based encryption. According to the competence of different users, the proposal can resist OW-CCA and IND-CCA security. Additionally, the T-GIBEwET scheme can resist a plaintext space attack.

Compared with other existing works, our proposal is more practical for use in cloud computing services.

Author Contributions: H.Z. provided the method. Q.X. verified the correctness of the method. H.Z. and D.X. wrote the first draft of the manuscript. H.Z. and D.X. provided the funding acquisition and T.L. provided the experiments. All authors contributed equally to this work and approved the submission. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the National Natural Science Foundation of China (no. 61801004), the National Natural Science Foundation of China (NSFC) (no. 61972050), the Projects of Henan Provincial Department of Science and Technology (no.212102310297), the Shandong Provincial Key Research and Development Program of China (2018CXGC0701), the Open Foundation of State Key Laboratory of Networking and Switching Technology (Beijing University of Posts and Telecommunications) (SKLNST-2019-2-17).

Institutional Review Board Statement: Not applicable.
Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations
The following abbreviations are used in this manuscript:

IoT Internet of Things
SE Searchable Encryption
IBEwET ID-Based Encryption with Equality Test
GIBE Group ID-Based Encryption
T-GIBEwET Traceable GIBE with Equality Test Scheme

References
1. Boneh, D.; Crescenzo, G.D.; Ostrovsky, R.; Persiano, G. Public key encryption with keyword search. In Proceedings of the International Conference on the Theory and Applications of Cryptographic Techniques, Interlaken, Switzerland, 13–17 April 2004; pp. 506–522.
2. Curtmola, R.; Garay, J.A.; Kamara, S.; Ostrovsky, R. Searchable symmetric encryption: Improved definitions and efficient constructions. J. Comput. Secur. 2011, 19, 895–934. [CrossRef]
3. Wang, C.; Cao, N.; Li, J.; Ren, K.; Lou, W. Secure ranked keyword search over encrypted cloud data. In Proceedings of the 2010 IEEE 30th International Conference on Distributed Computing Systems, Genova, Italy, 21–25 June 2010.
4. Benaloh, J.; Chase, M.; Horvitz, E. Patient controlled encryption: Ensuring privacy of electronic medical records. In Proceedings of the 2009 ACM Workshop on Cloud Computing Security, Chicago, CA, USA, 9–13 November 2009; pp. 103–114.
5. Ma, M.; He, D.; Kumar, N. Certificateless Searchable Public Key Encryption Scheme for Industrial Internet of Things. IEEE Trans. Ind. Inform. 2018, 14, 759–767. [CrossRef]
6. Wang, Y.; Sun, S.F.; Wang, J. Achieving Searchable Encryption Scheme with Search Pattern Hidden. IEEE Trans. Serv. Comput. 2020. [CrossRef]
7. Yang, G.; Tan, C.H.; Huang, Q. Probabilistic public key encryption with equality test. In Proceedings of the Cryptographers Track at the RSA Conference, San Francisco, CA, USA, 1–5 March 2010; Springer: Berlin, Germany, 2010; pp. 119–131.
8. Tang, Q. Towards public key encryption scheme supporting equality test with fine-grained authorization. In Proceedings of the Australasian Conference on Information Security and Privacy, Melbourne, VIC, Australia, 11–13 July 2011; Springer: Berlin/Heidelberg, Germany, 2011; pp. 389–406.
9. Tang, Q. Public key encryption schemes supporting equality test with authorisation of different granularity. Int. J. Appl. Cryptogr. 2012, 2, 304–321. [CrossRef]
10. Tang, Q. Public key encryption supporting plaintext equality test and user-specified authorization. Secur. Commun. Netw. 2012, 5, 1351–1362. [CrossRef]
11. Huang, K.; Tso, R.; Chen, Y. A New Public Key Encryption with Equality Test. In Proceedings of the International Conference on Network and System Security, New York, NY, USA, 3–5 November 2015; pp. 550–557.
12. Huang, K.; Tso, R.; Chen, Y. PKE-AET: Public Key Encryption with Authorized Equality Test. Br. Comput. Soc. 2015, 2686–2697. [CrossRef]
13. Huang, K.; Yu-Chi, C. Semantic Secure Public Key Encryption with Filtered Equality Test. In Proceedings of the 2015 12th International Joint Conference on e-Business and Telecommunications (ICETE), Alsace, France, 20–22 July 2015; pp. 327–334.
14. Ma, S.; Zhang, M.; Huang, Q.; Yang, B. Public key encryption with delegated equality test in a multi-user setting. Comput. J. 2015, 58, 986–1002. [CrossRef]
15. Huang, S.M.Q.; Zhang, M.; Yang, B. Efficient Public Key Encryption With Equality Test Supporting Flexible Authorization. IEEE Trans. Inf. Forensics Secur. 2015, 10, 458–470.
16. Ma, S. Identity-based encryption with outsourced equality test in cloud computing. Inform. Sci. 2016, 328, 389–402. [CrossRef]
17. Yang, M.; Wang, E. Identity-Based Encryption with Filtered Equality Test for Smart City Applications. Sensors 2019, 19, 3046.
18. Wang, Y.; Pang, H.; Deng, R. Securing messaging services through efficient signcryption with designated equality test. Inf. Sci. 2019, 490, 146–165. [CrossRef]
19. Duong, D.H.; Fukushima, K.; Kiyomoto, S.; Roy, P.S.; Susilo, W. Lattice-based public key encryption with equality test in standard model, revisited. arXiv 2020, arXiv:2005.03178.
20. Lee, T.; San, L.; Seo, J.H.; Huaxiong, W. Semi-generic construction of public key encryption and identity-based encryption with equality test. Inf. Sci. 2016, 373, 419–440. [CrossRef]
21. Lee, H.T.; Ling, S.; Seo, J.H.; Wang, H.; Youn, T.Y. Public Key Encryption with Equality Test in the Standard Model. Inf. Sci. 2020, 516, 89–108. [CrossRef]
22. Huang, K.; Tso, R.; Chen, Y.C. Somewhat semantic secure public key encryption with filtered-equality-test in the standard model and its extension to searchable encryption. J. Comput. Syst. Sci. 2017, 89, 400–409. [CrossRef]
23. Wang, Y.; Pang, H.; Tran, N.H. CCA Secure encryption supporting authorized equality test on ciphertexts in standard model and its applications. *Inf. Sci.* 2017, 414, 289–305. [CrossRef]
24. Zhang, K.; Chen, J.; Lee, H. Efficient Public Key Encryption with Equality Test in The Standard Model. *Theor. Comput. Sci.* 2019, 755, 65–80. [CrossRef]
25. Elhabob, R.; Zhao, Y.; Sella, I.; Xiong, H. Public Key Encryption with Equality Test for Heterogeneous Systems in Cloud Computing. *KSII Trans. Internet Inf. Syst.* 2019, 13, 4742–4770.
26. Lin, X.J.; Qu, H.; Zhang, X. Public Key Encryption Supporting Equality Test and Flexible Authorization without Bilinear Pairings. *Comput. Commun.* 2021, 170, 190–199. [CrossRef]
27. Zhu, H.; Wang, L.; Ahmad, H.; Niu, X. Pairing-free equality test over short ciphertexts. *Int. J. Distrib. Sens. Netw.* 2017, 13, 1550147717715605. [CrossRef]
28. Wu, L.; Zhang, Y.; Choo, K. Efficient and secure identity-based encryption scheme with equality test in cloud computing. *Future Gener. Comput. Syst.* 2017, 73, 22–31. [CrossRef]
29. Wu, L.; Zhang, Y.; Choo, K. Efficient Identity-Based Encryption Scheme with Equality Test in Smart City. *IEEE Trans. Sustain. Comput.* 2018, 3, 44–55. [CrossRef]
30. Qu, H.; Zhen, Y.; Lin, X. Certificateless Public Key Encryption with Equality Test. *Inf. Sci.* 2018, 462, 76–92. [CrossRef]
31. Elhabob, R.; Zhao, Y.; Hassan, A.; Xiong, H. PKE-ET-HS: Public Key Encryption with Equality Test for Heterogeneous Systems in IoT. *Wirel. Pers. Commun.* 2020, 113, 313–335. [CrossRef]
32. Elhabob, R.; Zhao, Y.; Sella, I.; Xiong, H. An efficient certificateless public key cryptography with authorized equality test in IoT. *J. Ambient. Intell. Humaniz. Comput.* 2020, 11, 1065–1083. [CrossRef]
33. Wu, L.; Zhang, Y.; Choo, K.R.; He, D. Pairing-Free Identity-Based Encryption with Authorized Equality Test in Online Social Networks. *Int. J. Found. Comput. Sci.* 2019, 30, 647–664. [CrossRef]
34. Lee, H.T.; Ling, S.; Seo, J.H.; Wang, H. Public Key Encryption with Equality Test from Generic Assumptions in the Random Oracle Model. *Inf. Sci.* 2019, 500, 15–33. [CrossRef]
35. Ling, Y.; Ma, S.; Huang, Q. Group Public Key Encryption with Equality Test Against Off-line Message Recovery Attack. *Inf. Sci.* 2020, 510, 16–32. [CrossRef]
36. Zhu, H.; Wang, L.; Ahmad, H. Key-policy attribute-based encryption with equality test in cloud computing. *IEEE Access* 2017, 5, 20428–20439. [CrossRef]
37. Wang, Q.; Peng, L.; Hu, X. Ciphertext-Policy Attribute-Based Encryption With Delegated Equality Test in Cloud Computing. *IEEE Access* 2018, 6, 760–771. [CrossRef]
38. Eltayieb, N.; Elhabob, R.; Hassan, A. Fine-grained attribute-based encryption scheme supporting equality test. In Proceedings of the International Conference on Algorithms and Architectures for Parallel Processing, Guangzhou, China, 15–17 November 2018; Springer: Berlin, Germany, 2018; pp. 220–233.
39. Sun, J.; Bao, Y.; Nie, X. Attribute-hiding predicate encryption with equality test in cloud computing. *IEEE Access* 2018, 6, 31621–31629. [CrossRef]
40. Cui, Y.; Huang, Q.H.Q.J. Ciphertext-policy attribute-based encrypted data equality test and classification. *Comput. J.* 2019, 62, 1166–1177. [CrossRef]
41. Lin, X.J.; Wang, Q.; Sun, L. Identity-based encryption with equality test and datestamp-based authorization mechanism. *Theor. Comput. Sci.* 2021, 117–132. [CrossRef]
42. Luo, X.; Ren, Y.; Liu, J. Identity-based group encryption. In Proceedings of the Australasian Conference on Information Security and Privacy, Melbourne, Australia, 4–6 July 2016; Springer: Berlin, Germany, 2016; pp. 87–102.