Hindawi
Security and Communication Networks
Volume 2022, Article ID 4605685, 14 pages
https://doi.org/10.1155/2022/4605685

Research Article

Group Public Key Encryption with Equality Test under Standard Model

Xiangtian Deng and Haifeng Qian

Software Engineering Institute, East China Normal University, Shanghai 200062, China

Correspondence should be addressed to Haifeng Qian; hfqian@cs.ecnu.edu.cn

Received 25 December 2021; Revised 2 June 2022; Accepted 10 June 2022; Published 7 July 2022

Academic Editor: Jie Cui

Group public key encryption with equality test (G-PKEET) scheme supports group granularity authorization on the equality test. An authorized proxy is able to check whether two ciphertexts belonging to the same group are encrypted from identical plaintext without decrypting them. However, in indistinguishability-based security notion, current existing PKEET and G-PKEET schemes do not allow adversary to invoke equality test as a service. In contrast, under practical circumstance, an adversary is probably able to exploit the equality test service offered by proxy to decipher a ciphertext, leading to unexpected and unwanted privacy leakage. In this paper, we propose a security definition that includes the abovementioned adversary ability. Through extending the functionality of current G-PKEET scheme, we design a concrete scheme that satisfies our new security definition. Furthermore, our G-PKEET scheme is the first G-PKEET scheme whose security properties can be proved under the standard model.

1. Introduction

Public key encryption with equality test (PKEET) is originally proposed by Yang et al. [1]. In PKEET scheme, users are able to check the equality of a pair of ciphertext without decrypting them, i.e., whether plaintexts decrypted from the ciphertext pair are equal. The tested ciphertext pairs are unnecessary to be encrypted by same public key. However, in PKEET scheme proposed by Yang et al., any user is allowed to perform equality test on arbitrary ciphertexts. In order for private key holders to manage access to equality test on corresponding ciphertexts, Tang proposes PKEET scheme in [2, 3], which introduces the concepts of "proxy" and "token." Tokens are generated by private key holders, and proxy can correctly perform equality test on ciphertext pairs only after obtaining corresponding tokens. This concept has been applied in a series of subsequent related works [4–7].

In common PKEET scheme, proxy has to request authorization from a user for executing equality test on ciphertexts encrypted by the user's public key. This authorization mode is referred to in [8] as "user granularity authorization."

To cope with different application scenarios, the concept "group granularity authorization" has been put forward [8, 9]. Under this authorization mode, a normal user can generate group ciphertexts with the permission of group administrator. Meanwhile, token issued by group administrator can be and only be used to conduct equality test on all group ciphertexts within the same group, thus helping to reduce the computational, storage, and communication overheads caused by token issuance. However, current PKEET definitions are not suitable for group granularity authorization in terms of efficiency [8]. Group public key encryption with equality test (G-PKEET) scheme is firstly proposed by Ling et al. [7], which is the first PKEET scheme supporting group granularity authorization. Besides the concepts of "proxy" and "token," a trustable "group administrator" role is additionally introduced in G-PKEET, in charge of issuing tokens. Each normal user, holding his own key pair, may apply to a "group administrator" for joining his group. With the permission of administrator, user is able to generate group ciphertexts. Similar to the definition of PKE, a group ciphertext can only be successfully decrypted by corresponding private key holder. In order to conduct equality test on group ciphertexts

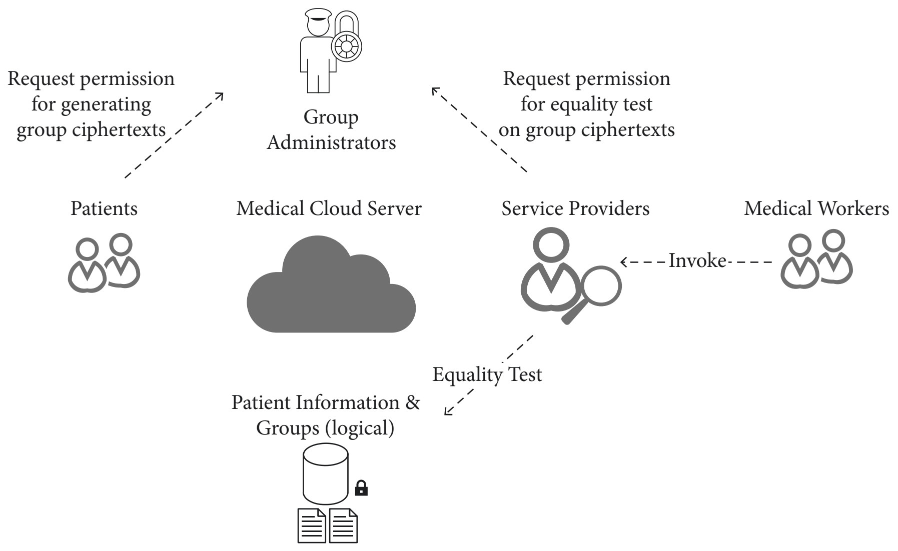
Request permission for generating group ciphertexts
Group Administrators
Request permission for equality test on group ciphertexts
Patients
Medical Cloud Server
Service Providers
Medical Workers
Invoke
Equality Test
Patient Information & Groups (logical)

Figure 1: G-MHSN scenario of G-PKEET.

within same group, a proxy only needs to submit an authorization request to the group administrator.

We remind readers that the primary feature of another existing definition Group Encryption (GE) [10] is allowing encryptor to conceal a recipient (decryptor) within a group of legitimate receivers. In G-PKEET, the entity of a group is a ciphertext. Detailedly speaking, users generate key pairs on their own and are able to join multiple group with same key pair. The group public key used when generating ciphertext decides which group the ciphertext belongs to, while proxy can only perform equality test on ciphertexts within the same group. In short, GE achieves anonymization of decryptor identity, while G-PKEET aims to save the overhead of communication and storage in aspect of token issuance. The two definitions are similar in naming but far various from each other in functionality.

In terms of security, several recently proposed PKEET schemes [4–6] have proved their security properties under standard model. Their security properties are more robust than those relying on random oracle model [11]. Scheme under random oracle model does not always guarantee its security when random oracle is instantiated with real hash function [12]. No G-PKEET scheme has proved to be secure under standard model up till now.

Existing works generally classify potential adversaries into two major categories relying on whether adversary is authorized to perform equality test on target user's ciphertext, i.e., obtaining corresponding token.

However, with respect to definition on adversary abilities, almost none of the existing schemes consider the situation where unauthorized adversary can restrictively invoke a proxy able to perform equality test on target user's ciphertext; i.e., he cannot directly perform equality test on target ciphertext. This gives rise to potential risk that unauthorized adversary exploits this service to arouse unwanted information leakage.

On the other hand, PKEET and G-PKEET schemes proposed in related works commonly embed plaintext and corresponding hash values into different components of ciphertext, aiming to make the ciphertext structurally capable of equality test. Thus we propose the definition of "improper ciphertext," standing for ciphertexts whose embedded plaintext, hash values, and other components are noncorrespondent. These ciphertexts may be submitted by malicious encryptor in order to interfere the correctness of equality test. Once such interference attack is launched in existing G-PKEET scheme, authorized proxy and private key holder have to interact to judge the properness of ciphertext, leading to communication overhead.

We use the following example to demonstrate our motivation more concretely: Consider a group based medic social network scenario (G-MHSN) in Figure 1.

The medical cloud server (MS) is responsible for storing encrypted patient information. A medical worker uses patient's public key to generate ciphertexts of patient information. A Service Provider (SP) owning computation power provides patient information equality test service for medical workers.

"Groups" represent different medical databases or different partitions of single medical database, depending on demanding of medical workers. The "group" mechanism allows SPs to request authorization only once to check the equality of ciphertexts belonging to the same group, regardless of whose public key these ciphertexts are encrypted by.

The group administrator (GA) is responsible for managing the group, including granting users to generate group ciphertexts and granting SPs to perform equality test on any pair of ciphertexts within the group.

Assume patient information of patient A and patient B needs to be tested in form of ciphertext. The workflow of this scenario is as follows: Firstly, the patient information of A and B is encrypted using their respective public keys and group public keys. Both ciphertexts need to belong to the same group. Secondly, the service provider SP requests permission from the group administrator GA to enable itself to perform equality test within this group. Finally, medical worker can invoke SP to perform equality test on patient information of user A and user B.

Furthermore, we use the above scenario to describe the security concerns that existing G-PKEET solution can not address.

(i) Service provider (SP): the SP should ensure that patient information equality test service it provides to medic workers does not reveal information other than equality test result.

(ii) Patients: Patients should have the ability to check properness of patient information ciphertext to prevent equality test service from misjudging their ciphertexts.

In coping with the security concerns mentioned above, we make extensions on existing G-PKEET definition. The contribution of our proposed scheme is listed as follows:

(1) Our G-PKEET scheme is the first G-PKEET scheme proved to be secure under standard model. Compared with existing G-PKEET scheme, our scheme is no longer dependent on random oracle to prove its security property. Simultaneously, our scheme is acceptable in aspect of communication and storage overhead.

(2) In response to the status that current G-PKEET scheme is incapable of permitting private key holder to verify whether embedded message value and hash value are corresponded, we modify the definition of decryption algorithm. The new definition allows private key holder to check whether components are corresponded and output $\perp$ in case of incorrespondence. This modification allows private key holder to detect and remove improper ciphertext more efficiently, ensuring correctness of equality test service. Implicitly, we demonstrate that the modification itself would not compromise security properties of our scheme; i.e., adversary cannot exploit this function to arouse information leakage.

(3) Our scheme strengthens the capability of unauthorized adversary, allowing adversaries of this type to invoke equality test oracle. Correspondingly, we prove our scheme to be secure under newly defined security property. This modification would guarantee that the unwanted leakage of plaintext information will not occur even when equality test service would be provided publicly.

2. Related Works

2.1. G-PKEET. The G-PKEET scheme is first proposed by Ling et al. [7], which introduces a trustable "group administrator" role in charge of a group secret key. Each user, holding their own public/private key pair, can apply to a "group administrator" for joining his group, i.e., obtaining a group public key. With the use of group public key, corresponding user private key, and the other user's public key, a group ciphertext is generated, able to be decrypted by private key of the other user. In order to test the equality of two group ciphertexts belonging to the same group, a proxy only needs to apply to "group administrator" for a group trapdoor.

Furthermore, this scheme is also claimed to be resistant to OMRA attack (offline message recovery attack) proposed by Tang in [13]. In OMRA attack, an attacker authorized by token distributor, who knows the distribution of message space in advance, can recover plaintext from target ciphertext through repeatedly performing equality test between target ciphertext and ciphertext encrypted from guessing message.

Ling et al. show that their scheme is resistant to OMRA attack through avoiding group ciphertexts from being generated publicly. This technique is similar to which used by Wu et al. in [14]. Detailedly speaking, only users who are granted with group public keys by the group administrator can generate a group ciphertext. Thus an OMRA attacker can not organize an attack unless he colludes with the group administrator or any user in the group.

2.2. PKEET under the Standard Model. Zhang et al. put forward a PKEET scheme in standard model [4], which is based on a specific IND-CCA2 public key encryption scheme [15]. Lee et al. also realize a PKEET scheme in standard model [6]. Their solution is generic, using HIBE and one-time signature as building blocks. Compared with Zhang's scheme, Lee's scheme gets rid of relying on specific number theoretic assumptions, at the cost of computational efficiency and dependency on strong cryptography primitive. Recently, Zeng et al. propose a generic PKEET scheme [5] in standard model based on hash proof system [16].

2.3. Other Related Works. In terms of refining the security performance of PKEET, Lin et al. put forward PKEET supporting flexible designated authorization (PKEET-FDA) [17]; PKEET-FDA requires proxy to use its secret key together with token to perform equality test, thus preventing attacker from testing ciphertexts with stolen token. Flexibility is embodied in allowing user to authorize multiple testers by once instead of generating specific token for each tester. Zhao and Zeng include access to equality test service within the scope of adversary capabilities and propose a PKEET scheme with corresponding security definition [18]. In aspect of authorization granularity, Lin et al. present PKEET supporting partial authentication (PKEET-PA) [19], where user can adaptively authorize the test right of any number of ciphertexts to a tester by providing constant-size token to tester. With regard to improvement on computation efficiency, several works have explored designing PKEET scheme without relying on bilinear pairing [20, 21].

3. Preliminaries

Notation 1. For a finite set S, we denote by $s \leftarrow_r S$ the process of uniformly sampling a random element from S. We say that a function f is negligible if $f(\lambda) \le 1/p(\lambda)$ for all polynomials $p(\cdot)$ and sufficiently large λ s.

3.1. Cryptography Primitives

3.1.1. Bilinear Map. Let $G_1 = \langle g_1 \rangle, G_2 = \langle g_2 \rangle$ and G_T be cyclic groups of prime order p. A bilinear map $e\colon G_1 \times G_2 \longrightarrow G_T$ satisfies the following properties:

(i) Bilinear: for any $g_1 \in G_1$, $g_2 \in G_2$ and $a, b \in \mathbb{Z}_p^*$, $e(g_1^a, g_2^b) = e(g_1, g_2)^{ab}$

(ii) Nondegenerate: for any $g_1 \in G_1$, $g_2 \in G_2$, $e(g_1, g_2) \neq 1_{G_T}$, where 1_{G_T} is the generator of G_T

(iii) Computable: e is efficiently computable

Additionally, in type-1 bilinear pairing $G_1 = G_2$, in type-3 bilinear pairing $G_1 \neq G_2$, and there are no efficiently computable homomorphisms between G_1, G_2 [22].

3.2. Cryptography Assumptions

3.2.1. Decisional Bilinear Diffie–Hellman (DBDH) Assumption. A game corresponding to DBDH assumption is constructed as follows, the roles included in which are challenger C and adversary A. We define Gen as a function which takes a security parameter λ as input and outputs a tuple (p, G, G_T, e, g) where G, G_T are multiplicative cyclic groups of order p, e is a bilinear map from $G \times G$ to G_T, and g is a generator of G.

Firstly C obtains a tuple (p, G, G_T, e, g) by executing Gen(λ). C then randomly selects elements a, b, c from $\mathbb{Z}_p$ and β from $\{0, 1\}$. If $\beta = 1$, C randomly selects an element in G_T as T; otherwise it sets $T = e(g, g)^{abc}$.

C passes the challenge tuple (g, g^a, g^b, g^c, T) to A. A then outputs a guess β'. We define an advantage of A in the above game as $|\Pr[\beta = \beta'] - 1/2|$.

The DBDH assumption in G, G_T claims that, for any PPT adversary, its advantage in the above game is negligible in the security parameter λ.

3.2.2. External Decisional Diffie–Hellman (XDH) Assumption. This assumption is firstly defined in [23]. Consider the following game between challenger C and the adversary A: Let Gen be an algorithm that takes a security parameter λ as an input and outputs a tuple $(p, G_1, G_2, G_T, e, g_1)$ where G_1, G_2, G_T are multiplicative cyclic groups of order p, e is a type-3 asymmetric bilinear map from $G_1 \times G_2$ to G_T, and g_1 is a generator of G_1.

Firstly C obtains a tuple $(p, G_1, G_2, G_T, e, g_1)$ by executing Gen(λ). C then chooses random elements a,b from $\mathbb{Z}_p$ and β from $\{0, 1\}$. If $\beta = 1$, C chooses a random element in G_1 to be T; otherwise $T = g_1^{ab}$.

C passes the challenge tuple (g_1, g_1^a, g_1^b, T) to A. A then outputs a guess β'. We define an advantage of A in the above game as $|\Pr[\beta = \beta'] - 1/2|$.

The XDH assumption in G_1 claims that, for any PPT adversary, its advantage in the above game is negligible in the security parameter λ.

4. Definition

4.1. Definition of G-PKEET. To help understand the relationship between algorithms and entities of G-PKEET, we illustrate execution entities of various algorithms and the interaction specification by Figure 2.

We remind readers that all algorithms of our G-PKEET scheme do not require multiround interaction. Thus we divide involved parties simply into executor and invoker. Invoker provides part of parameters needed for executing function to executor. Then executor, holding remaining secret parameters, executes the function and returns the result to invoker.

We describe the definition of G-PKEET functions as follows:

(i) Setup (λ): this nondeterministic function takes a security parameter λ as input and returns public system parameter PP.

(ii) KeyGen $_{\text{user}}$ (PP):this nondeterministic function produces a public/private key pair (pk, sk) for a normal user.

(iii) KeyGen $_{\text{group}}$ (PP):this nondeterministic function produces a group secret key gsk for a group administrator.

(iv) Join (gsk, pk_i): this nondeterministic function permits a user holding user key pk_i to join a specific group holding group secret key gsk. If executed properly, this function will generate and output a group public key gpk_i for user U_i.

(v) Enc $(gpk_i, sk_i, gpk_j, pk_j, m)$: this nondeterministic function takes in a group public key, its corresponding private key, another group public key, its corresponding public key, and a plaintext m to generate a ciphertext, where indices i, j, respectively, refer to encryptor and decryptor index.

(vi) Dec $(gpk_i, gpk_j, sk_j, C_{i,j})$: this deterministic function takes in a ciphertext $C_{i,j}$, group public key gpk_i of encryptor U_i and group public key gpk_j, and private key sk_j of decryptor U_j to recover plaintext from ciphertext. The function may output $\perp$ representing rejection under certain circumstances.

(vii) Verify $(gpk_i, gpk_j, sk_j, C_{i,j})$: this deterministic function takes in the same parameters as function Dec. Verify outputs $\perp$ if and only if Dec taking in the same parameters outputs $\perp$; otherwise Verify outputs 1 standing for the passing of verification.

One key purpose of Verify algorithm is allowing proxy to confirm to private key holder whether ciphertext is proper. Thus it needs to differ from decryption algorithm for not outputting plaintext as a result. On the other hand, Verify helps private key holder to detect improper ciphertext on his own and takes further reaction, such as removing them from the cloud server:

(i) Aut (gsk): this function generates a group trapdoor tuple gtd used for equality test.

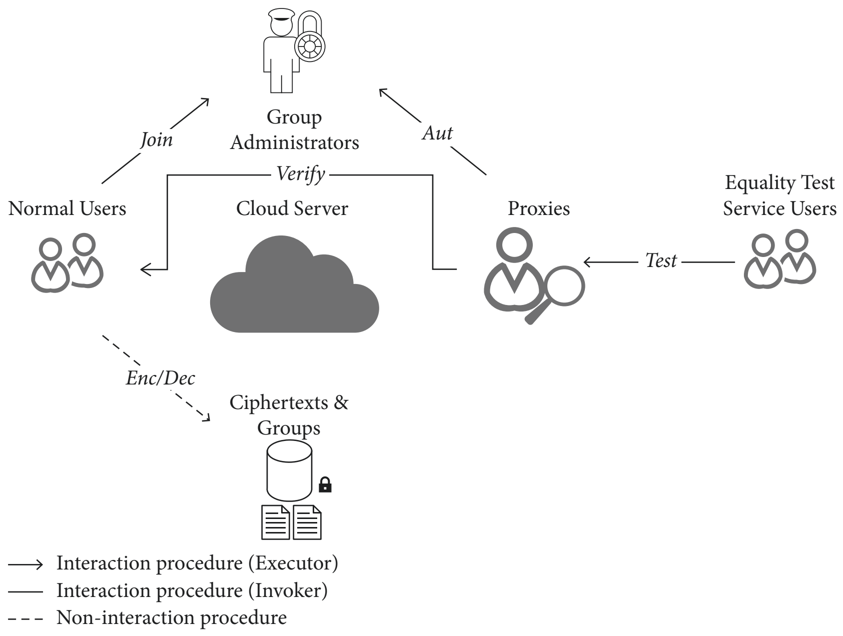
Group Administrators
Join
Aut
Verify
Normal Users
Cloud Server
Proxies
Equality Test Service Users
Test
Enc/Dec
Ciphertexts & Groups
⟶ Interaction procedure (Executor)
— Interaction procedure (Invoker)
- - - Non-interaction procedure

FIGURE 2: Interaction specification and function executor entity of G-PKEET.

(ii) Test $(C_{i,j}, C_{i',j'}, pk_j, pk_{j'}, gpk_j, gpk_{j'}, gtd)$: this deterministic function takes in two ciphertexts $C_{i,j}, C_{i',j'}$ belonging to the same group, corresponding group trapdoor gtd, group public key, and public key of two decryptors as input. It returns either $0, 1$ representing result of equality test or $\perp$ representing rejection under certain circumstances.

Optionally, proxy in advance queries private key holders with Verify$(gpk_i, gpk_j, sk_j, C_{i,\ \ j})$ and Verify$(gpk_{i'}, gpk_{j'}, sk_{j'}, C_{i',j'})$. In such cases, proxy may output $\perp$, indicating that the equality test is ceased due to failure on passing ciphertext verification.

4.1.1. Scheme Correctness. Let i, j, k, k' be distinct user indexes; gsk be group secret key generated by KeyGen$_{\text{group}}$; (pk_i, sk_i), (pk_j, sk_j), (pk_k, sk_k), $(pk_{k'}, sk_{k'})$ be corresponding user key pairs; $gpk_i, gpk_j, gpk_k, gpk_{k'}$ be group public keys generated, respectively, by Join(gsk, pk_i), Join(gsk, pk_j), Join(gsk, pk_k), Join$(gsk, pk_{k'})$; gtd be group trapdoor generated by Aut(gsk).

Given the aforementioned symbols, G-PKEET scheme achieves correctness if all following statements hold:

(1) $\text{Dec}(gpk_i, gpk_j, sk_j, \text{Enc}(gpk_i, sk_i, gpk_j, pk_j, m)) = m$.

(2) In the following procedure, $Test$ outputs 1 when $m = m'$; otherwise, it outputs 1 with negligible probability:

$$\begin{aligned} C_{i,k} &= \text{Enc}(gpk_i, sk_i, gpk_k, pk_k, m), \\ C_{j,k'} &= \text{Enc}(gpk_j, sk_j, gpk_{k'}, pk_{k'}, m'), \\ &\text{Test}(C_{i,k}, C_{j,k'}, pk_k, pk_{k'}, gpk_k, gpk_{k'}, gtd). \end{aligned} \quad (1)$$

4.2. Security Model. First, we make some foundational assumptions related to adversary's behavior pattern for simplicity similarly as in [3]:

(1) All users honestly generate their public/private keys

(2) There is no overlap between the user set and the proxy set

Due to the functionality of G-PKEET, an adversary who obtains a group token can trivially break the indistinguishability-based security notion of corresponding ciphertexts. Thus we introduce definition on two types of adversary, in the view of an arbitrary user U_t and a group administrator GA:

(1) Type-1 adversary: given a target ciphertext, the purpose of type-1 adversary is to guess the plaintext decrypted from it. The adversary has no access to corresponding private key, but he is able to invoke function Dec with this private key as one parameter (in a black-box fashion). Restrictedly, adversary is not allowed to invoke this function to decrypt the target ciphertext. In addition, type-1 adversary obtains the token required to check equality of the target ciphertext. Dec t.

In practical scenario shown by Figure 2, type-1 adversary can be regarded as an adversary stronger than authorized proxy due to its ability to invoke the Dec function black box mentioned above.

(2) Type-2 adversary: given a target ciphertext, the purpose of type-2 adversary is to distinguish which plaintext a target ciphertext is decrypted from. The adversary is assured that target ciphertext is encrypted from either of two plaintexts designated by him. In contrast to type-1 adversary, the adversary

1. The adversary claims a specific index t, representing the target user he would attack. Challenger runs $KeyGen_{group}$ to generate group public/secret key pair gpk, gsk.Challenger runs $KeyGen_{user}$, to generate key pairs(pk_i, sk_i) for all users index among $1 <= i <= N$, and runs $Join(sk_i, gsk)$ for each user to generate public user-group key gpk_i.
2. Query Phase 1: The adversary is allowed to issue following oracle queries:
   (a) $Oracle_{Dec}(C, i, j)$: Adversary submits ciphertext C, encryptor index and decryptor index i, j. Challenger returns the result of $Dec(gpk_i, gpk_j, sk_j, C_{i,j})$.
   (b) $Oracle_{Aut}$: Challenger runs algorithm $Aut(gsk)$ and returns the result to adversary.
   (c) $Oracle_{Enc}(m, i, j)$: Adversary submits plaintext message m, encryptor index and decryptor index i, j. Challenger returns the result of $Enc(gpk_i, sk_i, gpk_j, pk_j, m)$.
   (d) $Oracle_{Verify}(C, i, j)$: Challenger is queried with data C, encryptor index and decryptor index i, j. Challenger returns the result of $Verify(gpk_i, gpk_j, sk_j, C)$.
3. Challenge Phase: Challenger chooses a message $m_t \leftarrow_r M$ and sends $Enc(gpk_i, sk_i, gpk_t, pk_t, m_t)$ to the adversary, where $i \leftarrow_r [1, N]$ and $i \neq t$.
4. Query Phase 2: The adversary is allowed to launch same type of oracle queries as in Query Phase 1 with additional restriction: The challenge ciphertext C_t^* can not be submitted to Dec oracle together with decryptor index t.
5. The adversary terminates by outputting his guess m_t'

FIGURE 3: OW-CCA2 security game.

has no access to the corresponding token required for equality test on target ciphertext. However, he is able to invoke a specific Test function with this token as a parameter (in a black-box fashion). Similar to the restriction imposed on type-1 adversary, type-2 adversary is not allowed to invoke Test function to check equality of the target ciphertext.

In practical scenario shown by Figure 2, a type-2 adversary can be regarded as adversary stronger than service user due to its ability to invoke the Dec function black box mentioned above and designate target ciphertext. In existing G-PKEET scheme, type-2 adversary can however be only regarded as an adversary stronger than normal user.

As for adversary of different types, corresponding security property is designed as follows (in the view of U_t):

(i) Type-1 adversary: OW-CCA. A type-1 adversary can not recover the plaintext from a ciphertext, even if it is allowed to query the decryption oracle, verification oracle with specific restrictions. This is the best achievable security guarantee considering the equality test functionality.

(ii) Type-2 adversary: IND-CCA. A type-2 adversary is not allowed to obtain equality test token from GA. But he is allowed to access the decryption oracle, verification oracle, and test oracle with specific restrictions and enables choosing the challenge plaintext pair.

We remind readers that, for type-2 adversaries, IND-CCA2 security to be proved in our paper covers OW-CCA2 security. Specifically, the game definition of OW-CCA2 differs from that of IND-CCA2 only in how the challenge ciphertext is generated. Moreover, the proof of OW-CCA2 security can be reduced to IND-CCA2 security.

4.3. Security Properties

Definition 1. A G-PKEET cryptosystem achieves OW-CCA2 security against a type-1 adversary, if, for $1 \le t \le N$, any PPT adversary has only a negligible advantage in the attack game shown in Figure 3, where the advantage is defined to be $\Pr[m_t' = m_t]$.

We notify our readers that the definition of our scheme is weakened compared to that in the original G-PKEET definition of [7] in aspect of restriction imposed on decryption oracle in Query phase 2. Detailedly, in Ling et al.'s scheme, decryption oracle refuses to execute Dec algorithm only when challenge ciphertext and corresponding encryptor index and corresponding decryptor index are requested together. In our scheme, oracle refuses to execute Dec algorithm when challenge ciphertext and corresponding decryptor index are requested together, no matter what encryptor index is.

Definition 2. A G-PKEET cryptosystem achieves IND-CCA2 security against a type-2 adversary, if, for $1 \le t \le N$, any polynomial-time adversary has only a negligible advantage in the attack game shown in Figure 4, where the advantage is defined to be $|\Pr[b' = b] - 1/2|$

We notify our readers that the definition of our scheme is weakened compared to that in the original G-PKEET definition of [7] in aspect of restriction imposed on decryption oracle in Query phase 2 for the reason mentioned above. Corresponding to Oracle$_{\text{test}}$ which has not been included in related works, a new precondition is added: Target user and proxy should interact honestly in Verify procedure when proxy intends to conduct equality test on target user's ciphertext.

5. Construction

We remind our readers that building a generic G-PKEET scheme through combining an IND-CCA2 secure public key

1. The adversary claims a specific index t, representing the target user he would attack. Challenger runs $KeyGen_{group}$ to generate group public/secret key pair gpk, gsk. Challenger runs $KeyGen_{user}$ to generate key pairs(pk_i, sk_i) for all users index among $1 <= i <= N$, and runs $Join(sk_i, gsk)$ for each user to generate public user-group key gpk_i.
2. Query Phase 1: The adversary is allowed to issue following oracle queries:
   (a) $Oracle_{Dec}(C, i, j)$: Challenger is queried with data C, encryptor index and decryptor index i, j. Challenger returns the result of $Dec(gpk_i, gpk_j, sk_j, C)$.
   (b) $Oracle_{Enc}(M, i, j)$: Adversary submits plaintext message M, encryptor index and decryptor index i, j. Challenger returns the result of $Enc(gpk_i, sk_i, gpk_j, pk_j, m)$.
   (c) $Oracle_{Verify}(C, i, j)$: Challenger is queried with data C, encryptor index and decryptor index i, j. Challenger returns the result of $Verify(gpk_i, gpk_j, sk_j, C)$.
   (d) $Oracle_{Test}(C_{i,j}, i, j, C_{i',j'}, i', j')$: Challenger is queried with ciphertext $C_{i,j}, C_{i',j'}$, corresponding encryptor index i, i' and decryptor index j, j'. At least one of the decryptor index should be t. Challenger returns the result of $Test(C_{i,j}, C_{i',j'}, pk_j, pk_{j'}, gpk_j, gpk_{j'}, gtd)$

   At some point, the adversary sends two messages m_0, m_1 to challenger, and turns into next phase.
3. Challenge Phase: Challenger randomly chooses a random bit b. If $b = 0$, send $Enc(gpk_i, sk_i, gpk_t, pk_t, m_b)$ to the adversary, where $i \leftarrow_r [1, N]$ and $i \neq t$.
4. Query Phase 2: The adversary is allowed to launch same type of oracle queries as in Query Phase 1 with additional restrictions: The challenge ciphertext C_t^* can not be submitted to Dec oracle or $Test$ oracle together with decryptor index t;
5. The adversary terminates by outputting his guess b'.

Figure 4: IND-CCA2 security game.

encryption with a OW-CCA2 secure equality test scheme may be able to satisfy G-PKEET definition in aspect of functionality. However, it is difficult to prove its security properties, because trivially combining two schemes together will allow adversary to break security game through decryption oracle.

To be detailed, adversary modifies the ciphertext component of target ciphertext which supports equality test (specifically, the component embedded with plaintext hash value) and then submits new ciphertext to decryption oracle. According to game definition, the oracle would work normally by returning decrypted plaintext, leading to a successful attack. Our method adopts specific construction strategy, allowing components respectively embedded with plaintext and plaintext hash value to be bonded together. Consequently, in our scheme the ciphertext modified as mentioned above would be rejected by decryption oracle. It is worth noting that current general PKEET scheme (not G-PKEET) is built upon cryptographic primitives with specific properties, such as IBE and hash proof system. None of these related works adopts abstract PKEET definition to build a general PKEET scheme with special properties.

We present our construction for G-PKEET scheme as follows:

(i) Setup (λ): this algorithm outputs public parameter $PP = (p, q, G, G_T, \widetilde{G}, \widetilde{G}_T, g, \widetilde{g}_1, \widetilde{g}_2, e, \widetilde{e}, H_1, H_2)$ as follows.

(1) G, G_T are groups of prime order p, satisfying bilinear map $e\colon G \times G \longrightarrow G_T$. g is a random generator of G.

(2) $\widetilde{G}_1, \widetilde{G}_2, \widetilde{G}_T$ are groups of prime order q corresponding to size of message space M, satisfying asymmetric bilinear map $\widetilde{e}\colon \widetilde{G}_1 \times \widetilde{G}_2 \longrightarrow \widetilde{G}_T$. $\widetilde{g}_1, \widetilde{g}_2$ are random generators of $\widetilde{G}_1, \widetilde{G}_2$.

(3) Hash functions H_1, H_2, respectively, map $M \longrightarrow \widetilde{G}_2$; $(G_T, \widetilde{G}_1, \widetilde{G}_1, \widetilde{G}_2, G, \widetilde{G}_1, \widetilde{G}_2, \widetilde{G}_2) \longrightarrow \mathbb{Z}_p$.

(ii) KeyGen $_{\text{user}}(PP)$: this algorithm randomly selects $x, y, z, \alpha \leftarrow_r \mathbb{Z}_p$; $g_1 = g^{\alpha}$; $g_2 \leftarrow_r G$; $\chi, \phi \leftarrow_r \mathbb{Z}_q$. The structure of generated key pair is as follows:

$sk = (\chi, \phi, g_2^{\alpha}, x, y, z)$, $pk = (Z = e(g_1, g_2), u, v, w, \widetilde{g}_1^{\chi}, \widetilde{g}_2^{\chi}, \widetilde{g}_1^{\phi}, \widetilde{g}_2^{\phi})$, where $u = g^x, v = g^y, w = g^z$.

(iii) KeyGen $_{\text{group}}(PP)$: this algorithm outputs $gsk = (\theta_1, \theta_2, \theta_3)$ where $\theta_1, \theta_2, \theta_3 \leftarrow_r \mathbb{Z}_q$.

(iv) Join (gsk, pk_i): this algorithm outputs group public key for U_i in the following structure: $gpk_i = (\widetilde{g}_1^{\chi_i\theta_1}, \widetilde{g}_2^{\chi_i\theta_1}, \widetilde{g}_1^{\theta_2}, \widetilde{g}_1^{\theta_3\phi_i}, \widetilde{g}_1^{\theta_3})$.

(v) Encrypt $(gpk_i, sk_i, gpk_j, pk_j, m)$: the ciphertext is generated with structure as follows:

$$\begin{gathered} C_1 = \left(m \| \gamma_1 \| \gamma_2\right)_{\text{encode}} Z_j^s, C_2 = H_1(m)^{\chi_i\gamma_1} \widetilde{g}_1^{\theta_2\gamma_2} \widetilde{g}_1^{\theta_3\phi_j\gamma_1}, \\ C_3 = \widetilde{g}_1^{\gamma_2}, C_4 = \widetilde{g}_2^{\gamma_1\theta_1\chi_i}, C_5 = g^s, C_6 = r, \\ C_7 = \left(u_j^t v_j^r w_j\right)^s, C_8 = \widetilde{g}_1^{\gamma_1\theta_1\chi_i}, C_9 = \widetilde{g}_2^{\gamma_1\phi_j}, C_{10} = \widetilde{g}_2^{\gamma_1}, \end{gathered} \tag{2}$$

where $\gamma_1, \gamma_2 \leftarrow_r \mathbb{Z}_q$; $s \leftarrow_r \mathbb{Z}_p$, $t = H_2(C_1, C_2, C_3, C_4, C_5, C_8, C_9, C_{10})$. The encoding function and decoding function (to be mentioned later) map between 3-element tuple m, γ_1, γ_2 and G_T. Assume that the message space M is $\mathbb{Z}_q$. The definitions of *encode* and *decode* are as follows:

(1) encode(m, γ_1, γ_2): this function takes in plaintext element m and 2 group elements $\gamma_1, \gamma_2 \in \mathbb{Z}_q$. It outputs a group element $(m \| \gamma_1 \| \gamma_2)_{\text{encode}} \in G_T$.

(2) decode$((m \| \gamma_1 \| \gamma_2)_{\text{encode}})$: this function takes in a group element that belongs to G_T. It outputs 3 group elements $m, \gamma_1, \gamma_2 \in \mathbb{Z}_q$ or $\perp$ to indicate decoding failure.

Both of these functions are public, satisfying equation decode$(\text{encode}(m, \gamma_1, \gamma_2)) = (m, \gamma_1, \gamma_2)$ for any $m, \gamma_1, \gamma_2 \in \mathbb{Z}_q$. Note that these two functions impose an implicit restriction on $|\mathbb{Z}_p|, |\mathbb{Z}_q|$: the parameter should satisfy the inequality $|\mathbb{Z}_q|^3 \leq |\mathbb{Z}_p|$, corresponding to the size of $M \times \mathbb{Z}_q \times \mathbb{Z}_q$ and G_T. We will omit subscripts "encode" and "decode" in subsequent formulas for conciseness:

(i) Decrypt $(gpk_i, gpk_j, sk_j, C_{i,j})$: firstly, decryptor checks whether the following equation stands:

$$e\left(C_5, u_j^t v_j^{C_6} w_j\right) = e\left(C_7, g\right), \tag{3}$$

where $t = H_2(C_1, C_2, C_3, C_4, C_5, C_8, C_9, C_{10})$.

Decryptor outputs $\perp$ and aborts decryption procedure if aforementioned equation does not stand; otherwise he continues to execute the following steps:

$$m^* \| \gamma_1^* \| \gamma_2^* \leftarrow \frac{C_1}{e\left(C_5, g_{2,j}^{\alpha}\right)}; , \tag{4}$$

where component $g_{2,j}^{\alpha}$ is taken from sk_j.

Additionally, decryptor should judge whether the following equations stand:

$$\tilde{g}_1^{\gamma_2^*} = C_3; \frac{\tilde{e}\left(C_2\left(\tilde{g}_1^{\theta_3}\right)^{-\phi_j \gamma_1^*}, \tilde{g}_2\right)}{\tilde{e}\left(H_1(m^*)^{\gamma_1^*}, \tilde{g}_2^{\chi_i}\right)} = \tilde{e}\left(\tilde{g}_1^{\theta_2}, \tilde{g}_2^{\gamma_2^*}\right), \tag{5}$$

where $\tilde{g}_1^{\theta_2}$ is taken from group public key gpk_i.

$$\left(\tilde{g}_2^{\theta_1 \chi_i}\right)^{\gamma_1^*} = C_4; \left(\tilde{g}_1^{\theta_1 \chi_i}\right)^{\gamma_1^*} = C_8; \tilde{g}_2^{\gamma_1^* \phi_j} = C_9; \tilde{g}_2^{\gamma_1^*} = C_{10}, \tag{6}$$

where $\tilde{g}_2^{\theta_1 \chi_i}, \tilde{g}_1^{\theta_1 \chi_i}$ are taken from gpk_i.

Decryptor returns m^* if the above equations stand; otherwise it outputs $\perp$. We define ciphertexts which do not satisfy the equations above as "improper ciphertext."

(ii) Verify $(gpk_i, gpk_j, sk_j, C_{i,j})$: this algorithm works the same as Dec except for returning 1 instead of m^*.

In terms of concrete construction, private key holder has to check the correspondence of embedded plaintext and hash value by extracting plaintext value from ciphertext $\tilde{e}(\tilde{g}_1^{\theta_3 \phi_i}, C_{10}) = \tilde{e}(\tilde{g}_1^{\theta_3}, C_9)$ in advance. Thus the Verify algorithm unavoidably has similar procedure to Dec algorithm. How to allow Verify algorithm to check the properness of ciphertext without needing of extracting plaintext value is an issue that requires further discussion.

(iii) Aut (gsk): this algorithm returns tuple $gtd = \{\theta_2, \theta_3\}$.

(iv) Test $(C_{i,j}, C_{i',j'}, gtd, pk_i, pk_{i'})$: firstly, tester needs to verify the correspondence of C_9 and C_{10} through judging whether the following equation stands, where the component of gpk_i is used:$\tilde{e}(\tilde{g}_1^{\theta_3 \phi_i}, C_{10}) = \tilde{e}(\tilde{g}_1^{\theta_3}, C_9)$

Similar process will be performed on the other ciphertext $C_{i',j'}$. Tester proceeds only when both equations stand; otherwise, it aborts by outputting $\perp$.

The algorithm outputs 1 if

$$\frac{\tilde{e}\left(C_2/C_3^{\theta_2}, C_4'\right)}{\tilde{e}\left(C_8', C_9^{\theta_3}\right)} = \frac{\tilde{e}\left(C_2'/(C_3')^{\theta_2}, C_4\right)}{\tilde{e}\left(C_8, (C_9')^{\theta_3}\right)}. \tag{7}$$

Otherwise, it outputs 0.

Optionally, proxy in advance queries private key holders with Verify$(gpk_i, sk_j, C_{i,j})$, Verify$(gpk_i', sk_{j'}, C_{i',j'})$ in order for detecting whether ciphertexts to be tested are proper. In such cases, proxy outputs $\perp$ when Verify procedure outputs $\perp$, indicating that the equality test is ceased due to improper ciphertext.

Theorem 1. *The proposed G-PKEET cryptosystem achieves correctness mentioned on Section 4.4.1.*

Proof. As for the two stated statements, we correspondingly have the following:

(i) The first step of decryption judges whether

$$e\left(C_5, u_j^t v_j^{C_6} w_j\right) = e\left(C_7, g\right), \tag{8}$$

corresponds to $e(g^{s_j}, u_j^t v_j^r w_j) = e((u_j^t v_j^r w_j)^{s_j}, g)$, where $t = H_2(C_1, C_2, C_3, C_4, C_5, C_8, C_9, C_{10})$.

The second step of decryption, which calculates

$$m^* \| \gamma_1^* \| \gamma_2^* \leftarrow \frac{C_1}{e\left(C_5, g_{2,j}^{\alpha}\right)}; \tag{9}$$

corresponds to $(m \| \gamma_1 \| \gamma_2) Z_j^s / e(g^s, g_2^{\alpha})$.

In the final step, we have

$$\frac{\tilde{e}\left(C_2(\tilde{g}_1)^{-\theta_3 \phi_j \gamma_1^*}, \tilde{g}_2\right)}{\tilde{e}\left(H_1(m^*)^{\gamma_1^*}, \tilde{g}_2^{\chi_i}\right)} = \frac{\tilde{e}\left(H_1(m^*)^{\gamma_1^* \chi_i}, \tilde{g}_2\right) \tilde{e}\left(\tilde{g}_1^{\gamma_2^* \theta_2}, \tilde{g}_2\right)}{\tilde{e}\left(H_1(m^*)^{\gamma_1^* \chi_i}, \tilde{g}_2\right)}. \tag{10}$$

It is obvious that all aforementioned steps stand if ciphertext $c = \text{Enc}(gpk_i, sk_i, gpk_j, pk_j, m)$ is correctly encrypted.

(ii) Set C, C', respectively, as $\text{Enc}(gpk_i, sk_i, gpk_k, pk_k, m)$ and $\text{Enc}(gpk_j, sk_j, gpk_{k'}, pk_{k'}, m')$. $\tilde{e}(C_2/C_3^{\theta_2}, C_4')/\tilde{e}(C_8', C_9^{\theta_3})$ can be reduced to

$$\widetilde{e}\left(H_1(m), \widetilde{g}_2\right)^{\chi_i \chi_j \gamma_1 \gamma_1\prime \theta_1}. \tag{11}$$

Obviously, $\widetilde{e}(C_2'/(C_3')^{\theta_2}, C_4)/\widetilde{e}(C_8, (C_9')^{\theta_3})$ equals the above formula if $m = m'$; otherwise the equation stands only when a hash collision happens, the rate of which is negligible. □

6. Security Analysis

6.1. OW-CCA2 Security

Theorem 2. *The proposed G-PKEET cryptosystem achieves OW-CCA2 property against a type-1 PPT adversary in the standard model based on the DBDH assumption on G.*

Proof. The main idea of proof is to construct a series of games which are reduced from the original OW-CCA2 game. The adversary unconditionally gains negligible advantage in the last game and is unable to distinguish each pair of games adjacent in reduction.

To complete the proof formally, we need to construct a series of games [24]:

Game 0. Challenger behaves the same as OW-CCA2 game in Figure 3.

Game 1. In Game 1, challenger modifies component C_1 challenge ciphertext C_t^* consequently altered ciphertext is given as the following structure using corresponding components of pk_t, gpk_i, sk_i of

$$\begin{aligned} &\text{Invalid}_{m\gamma_1\gamma_2} \leftarrow_r G_T; C_1^* = \left(\text{Invalid}_{m\gamma_1\gamma_2}\right) Z_t^s, C_2 = H_1(m)^{\chi_i\gamma_1} \widetilde{g}_1^{\theta_2\gamma_2} \widetilde{g}_1^{\theta_3\phi_t\gamma_1}, C_3 = \widetilde{g}_1^{\gamma_2}, \\ &C_4 = \widetilde{g}_2^{\gamma_1\theta_1\chi_i}, C_5 = g^s, C_6 = r, C_7 = \left(u_t^t v_t^r w_t\right)^s, C_8 = \widetilde{g}_1^{\gamma_1\theta_1\chi_i}, C_9 = \widetilde{g}_2^{\gamma_1\phi_t}, C_{10} = \widetilde{g}_2^{\gamma_1}. \end{aligned} \tag{12}$$

Superscript $t = H_2(C_1, C_2, C_3, C_4, C_5, C_8, C_9, C_{10})$; $s, r \leftarrow_r \mathbb{Z}_p$; $\gamma_1, \gamma_2 \leftarrow_r \mathbb{Z}_q$.

We will prove that there exists no PPT adversary able to distinguish Game 0 from Game 1 with nonnegligible advantage as long as DBDH assumption holds on G. □

6.1.1. Indistinguishability between Game 0 and Game 1.

Briefly speaking, we construct an attacker B against DBDH assumption. He simultaneously invokes A as subprocedure, who attempts to distinguish between Game 0 and Game 1; i.e., B will play the role as challenger against A.

Setup. After receiving a DBDH tuple (g, g^a, g^b, g^c, T), B replaces corresponding elements of public key pk_t with DBDH challenge tuple elements as follows:

$$\begin{aligned} &g_1 = g^a, g_2 = g^b, u = g^b g^{y_u}, v = \left(g^b\right)^{(x_v)} g^{(y_v)}, \\ &w = \left(g^b\right)^{(x_w)} g^{(y_w)}, \end{aligned} \tag{13}$$

where $y_u, x_v, y_v, x_w, y_w \leftarrow_r \mathbb{Z}_p$. Other unmentioned elements contained in pk_t are generated according to our scheme.

Correspondingly, the private key tuple sk_t can be represented as $sk_t = \{\phi_t, \chi_t, g^{ab}, x = b + y_u, y = bx_v + y_v, w = bx_w + y_w\}$, where x, y, z, g^{ab} is not known to B.

Query Phase 1. (1) Decryption Oracle: when B is queried with user index t, firstly he checks whether the following statement stands:

$$e\left(C_5, u^t v^{C_6} w\right) = e(C_7, g), \tag{14}$$

where superscript $t = H_1(C_1, C_2, C_3, C_4, C_5, C_8, C_9, C_{10})$.

If $t + rx_v + x_d = 0$, B aborts the whole procedure and outputs a random bit to DBDH challenger. Since adversary has no idea of x_v and x_d, the probability for this type of event is query$/p$, where query is the times that adversary visits decryption oracle.

Otherwise, B begins to extract plaintext from non-challenge ciphertext, by firstly generating $\delta \leftarrow_r \mathbb{Z}_p$:

$$\begin{aligned} &d_{C,1} = g_1^{-ty_u + ry_v + y_w/(t + rx_v + x_w)} \left(u^t v^r w\right)^\delta, \\ &d_{C,2} = g_1^{\frac{-1}{(t + rx_v + x_w)}} g^\delta. \end{aligned} \tag{15}$$

Let $\widetilde{\delta} = \delta - a/(t + rx_v + x_w)$. Then, we have

$$d_{C,1} = g_2^a \left(u^t v^r d\right)^{\widetilde{\delta}}, d_{C,2} = g^{\widetilde{\delta}}, m \| \gamma_1 \| \gamma_2 = C_1 \frac{e\left(d_{C,1}, C_5\right)}{e\left(C_7, d_{C,2}\right)}. \tag{16}$$

Finally, it checks the consistency of plain text, hash message, and other components embedded in the ciphertext and outputs result of decryption:

$$\text{Decryption}_{\text{Oracle}}(ct, sk^*, pk) = \begin{cases} \perp, & \text{if } (1) \\ m, & \text{otherwise} \end{cases} \tag{17}$$

where (1) refers to the following statement:

$$\begin{aligned} &\left(\widetilde{g}_1^{\phi_t\theta_3}\right)^{\gamma_1} C_3^{\theta_2} H_1(m)^{\chi_i\gamma_1} \neq C_2 \vee \widetilde{g}_2^{\chi_i\theta_1\gamma_1} \neq C_4 \vee \widetilde{g}_1^{\gamma_2} \neq C_3 \\ &\vee \widetilde{g}_1^{\chi_i\theta_1\gamma_1} \neq C_8 \vee \widetilde{g}_2^{\gamma_1\phi_t} \neq C_9 \vee \widetilde{g}_2^{\gamma_1} \neq C_{10}. \end{aligned} \tag{18}$$

(2) Encryption Oracle: challenger B works the same as mentioned in original G-PKEET scheme, since he knows the value of χ_t when being required to generate a ciphertext with encryptor index t.

(3) Verification Oracle: challenger B works in the same way as mentioned in Decryption Oracle except for following the description of Verify on output.

Challenge. Challenger B selects $m \leftarrow_r M$ and generates challenge ciphertext tuple in the following structure:

$$C_1^* = (m\|\gamma_1\|\gamma_2)T, C_2^* = H_1(m)^{\chi_i\gamma_1}\tilde{g}_1^{\theta_2\gamma_2}\tilde{g}_1^{\theta_3\phi_t\gamma_1}, C_3^* = \tilde{g}_1^{\gamma_2}, C_4^* = \tilde{g}_2^{\gamma_1\theta_1\chi_i}, C_5^* = g^c, C_6^* = \frac{t^* + x_w}{x_v}, C_7^* = (g^c)^{t^* y_u + (C_6^*) y_v + y_w}, C_8^* = \tilde{g}_1^{\gamma_1\theta_1\chi_i}, C_9 = \tilde{g}_2^{\gamma_1\phi_t}, C_{10} = \tilde{g}_2^{\gamma_1}, \quad (19)$$

where superscript $t = H_1(C_1, C_2, C_3, C_4, C_5, C_8, C_9, C_{10})$; $\gamma_1, \gamma_2 \leftarrow_r \mathbb{Z}_q$.

Query Phase 2. In this phase, decryption oracle is added with new restrictions: (1) challenge ciphertext cannot be requested together with decryptor index "t"; (2) when $C \neq C^*$ and hash value $H_1(C_1, C_2, C_3, C_4, C_5, C_8, C_9, C_{10}) = t^*$, challenger B aborts the whole game and outputs a random bit to DBDH challenger.

Verification oracle works the same as mentioned in phase 1 except when being queried with challenge ciphertext and t as decryptor index. More detailedly, output is defined as follows:

$$\text{Oracle}_{\text{Verification}}(ct, sk^*, pk) = \begin{cases} 1, & \text{if } i = i^* \\ \perp, & \text{otherwise} \end{cases} \quad (20)$$

Under actual circumstances, Verify might output 1 when $i \neq i^*$, consequently breaking the indistinguishability between Game 2 and actual circumstance. However, this situation happens when and only when a hash collision happens, the rate of which is negligible.

Guess Phase. The adversary A outputs 0 or 1 representing the game he recognizes. B outputs the guess $T = e(g,g)^{(abc)}$ when A outputs 0, otherwise 1.

Similar to [4], the probability that B aborts during the simulation is at most $ADV^{CR} + \text{query}/p$, corresponding to the illustration in Query phases 1,2, where ADV^{CR} refers to the probability of a hash collision.

When B does not abort then A's view is identical to its view in Game 0. Otherwise, his view is identical to that of Game 1. Uniform randomly generated $T = e(g,g)^{abc}$ in Game 1 is statistically indistinguishable to $\text{Invalid}_{m\gamma_1\gamma_2}$ when $\text{Invalid}^*_{m\gamma_1\gamma_2}$ when $T = e(g,g)^Z$. The reason is as follows:

$$\begin{aligned} T &= e(g,g)^z z \leftarrow_r \mathbb{Z}_p \ s.t.\ T = e(g^a, g^b)^z, \\ (m^*\|\gamma_1^*\|\gamma_2^*)T &= (m^*\|\gamma_1^*\|\gamma_2^*)e(g^a,g^b)^c e(g^a,g^b)^{z-c}, \\ &= \left((m^*\|\gamma_1^*\|\gamma_2^*)\cdot e(g^a,g^b)^{z-c}\right)e(g^a,g^b)^c, \\ \text{Invalid}^*_{m\gamma_1\gamma_2} &\Leftrightarrow (m^*\|\gamma_1^*\|\gamma_2^*)\cdot e(g^a,g^b)^{z-c}. \end{aligned} \quad (21)$$

Since adversary A can be invoked to attack DBDH problem, his advantage can be expressed as

$$\text{distinguish}_{0,1} \leq \epsilon_{dbdh} + Pr_{\text{Abort}} \leq \epsilon_{dbdh} + ADV^{CR} + \frac{\text{query}}{p}. \quad (22)$$

In Game 1, C_1^* in challenge ciphertext is irrelevant to plaintext message; therefore adversary 's advantage can be reduced to the success rate of breaking hash function's preimage properties. Eventually, the advantage of adversary in OW-CCA2 game can be reduced as follows:

$$\epsilon OW - CCA2 = \text{distinguish}_{0,1} + \epsilon\text{Game2} \leq \epsilon dbdh + ADV^{CR} + \frac{\text{query}}{p} + ADV^{PI}, \quad (23)$$

where ADV^{PI} refers to advantage of adversary in preimage attack on hash function H_2. Thus Theorem 2 is proved.

6.2. IND-CCA2 Security

Theorem 3. *The proposed G-PKEET cryptosystem achieves IND-CCA2 property against a type-2 PPT adversary in the standard model, as long as DBDH assumption on G and XDH assumption on $\tilde{G}_1$ hold.*

Proof. Game 0. Challenger behaves the same as in IND-CCA2 game of Figure 4. Game 1. In Game 1, challenger modifies component C_1 of challenge ciphertext C_t^*; consequently altered ciphertext is given as the following structure using corresponding components of gpk_t, pk_t, gpk_i, sk_i:

$$\text{Invalid}_{m\gamma_1\gamma_2} \leftarrow_r G_T, C_1^* = (\text{Invalid}_{m\gamma_1\gamma_2})Z_t^s, C_2 = H_1(m)^{\chi_i\gamma_1}\tilde{g}_1^{\theta_2\gamma_2}\tilde{g}_1^{\theta_3\phi_t\gamma_1}, C_3 = \tilde{g}_1^{\gamma_2}, C_4 = \tilde{g}_2^{\gamma_1\theta_1\chi_i}, C_5 = g^s, C_6 = r, C_7 = (u_t^t v_t^r w_t)^s, C_8 = \tilde{g}_1^{\gamma_1\theta_1\chi_i}, C_9 = \tilde{g}_2^{\gamma_1\phi_t}, C_{10} = \tilde{g}_2^{\gamma_1}. \quad (24)$$

We will prove that there exists no PPT adversary able to distinguish between Game 0 and Game 1 as long as DBDH assumption holds on G. □

6.2.1. Indistinguishability between Game 0 and Game 1. Using similar technique referred to in Section 6.1.1,, we construct an attacker B against DBDH assumption. He simultaneously invokes A as subprocedure, who attempts to distinguish between Game 0 and Game 1; i.e., B will play the role as challenger against A.

Setup. After receiving a DBDH tuple (g, g^a, g^b, g^c, T), B replaces corresponding elements of public key pk_t and private key sk_t in the same way referred to in Section 6.1.1.

Query phase 1.

(1) Decryption Oracle: when Decryption oracle is queried with parameter decryptor index t, B deals with the ciphertext using the same way mentioned in Section 6.1.1.

(2) Encryption Oracle: challenger B works the same as mentioned in original G-PKEET scheme.

(3) Verification Oracle: challenger B works the same as mentioned in Section 6.1.1.

(4) Test Oracle: before performing the computation in procedure $Test$ and outputting result, the test oracle takes corresponding measures according to decryptor index:

When decryptor index equals t, the test oracle invokes verification oracle to process queried ciphertext together with decryptor index t. The test oracle refuses to continue the test by outputting $\perp$ if verification oracle returns $\perp$. This step corresponds to the precondition that user t and the proxy will interact with each other honestly.

Otherwise, decryptor will assume submitted ciphertext is proper, corresponding to precondition that other users may not honestly interact with the proxy, i.e., maliciously claiming that an improper ciphertext is proper.

Challenge. After receiving plaintext m_0, m_1 from adversary, challenger B selects $b \leftarrow_r 0, 1$ and generates challenge ciphertext tuple in the same way as mentioned in Section 6.1.1.

Query Phase 2. Decryption oracle and verification oracle are applied with the same modification referred to in Section 6.1.1.

Guess Phase. The adversary A outputs 0 or 1 representing the game he recognizes. B outputs the guess $T = e(g, g)^{(abc)}$ when A outputs 0, otherwise 1.

Similar to that referred to in Section 6.1.1, the probability that B aborts during the simulation is at most $ADV^{CR} + \text{query}/p$.

When $T = e(g, g)^{abc}$ and B does not abort then A's view is identical to its view in Game 0. Otherwise, his view is identical to that of Game 1.

Since adversary A can be invoked to attack DBDH problem, his advantage is expressed as

$$\text{distinguish}_{0,1} \le \epsilon_{dbdh} + \Pr_{\text{Abort}} \le \epsilon_{dbdh} + ADV^{CR} + \frac{\text{query}}{p}. \tag{25}$$

Game 2. In Game 2, challenger modifies components C_2^*, C_3^* of challenge ciphertext C_t^* on the basis of Game 1. Consequently altered ciphertext is generated using corresponding components of gpk_t, pk_t, gpk_i, sk_i as follows:

$$\begin{gathered}\text{Invalid}_{m\gamma_1\gamma_2} \leftarrow_r G_T; C_1^* = \left(Invalid_{m\gamma_1\gamma_2}\right) Z_t^s, C_2^* = \widetilde{g}_1^{\gamma_3}, C_3^* = \widetilde{g}_1^{\gamma_2}, C_4^* = \widetilde{g}_2^{\gamma_1 \theta_1 \chi_i}, \\ C_5^* = g^s, C_6^* = r, C_7^* = \left(u_t^t v_t^r w_t\right)^s, C_8 = \widetilde{g}_1^{\gamma_1 \theta_1 \chi_i}, C_9 = \widetilde{g}_2^{\gamma_1 \phi_t}, C_{10} = \widetilde{g}_2^{\gamma_1}.\end{gathered} \tag{26}$$

where $\gamma_3, \gamma_2 \leftarrow_r \mathbb{Z}_q$, superscript $t = H_1(C_1^*, C_2^*, C_3^*, C_4^*, C_5^*, C_8^*, C_9^*, C_{10}^*)$.

We will prove that there exists no PPT adversary able to distinguish between Game 1 and Game 2 as long as XDH assumption holds on $\widetilde{G}_1$.

6.2.2. Indistinguishability between Game 1 and Game 2. We construct an attacker B against XDH assumption. He simultaneously invokes A as subprocedure, who attempts to distinguish between Game 1 and Game 2; i.e., B will play the role as challenger against A.

Setup. Challenger B receives a XDH tuple $(\widetilde{g}_1^{\alpha}, \widetilde{g}_1^{\beta}, \widetilde{g}_1^{\zeta})$ and sets $\widetilde{g}_1^{\theta_2} = \widetilde{g}_1^{\beta}$; thus he does not know concrete value of θ_2 included in gsk. Other public parameters and key pairs are generated in the same way according to our scheme definition.

Query phase 1.

(1) Decryption oracle: When being queried with decryptor index t, since challenger B does not know the value of θ_2, he has to check THE consistency of C_1 and C_2 by judging whether

$$C_3 = \widetilde{g}_1^{\gamma_2^*}; \left(\widetilde{g}_2^{\theta_1 \chi_i}\right)^{\gamma_1^*} = C_4. \tag{27}$$

$$\widetilde{e}\left(\widetilde{g}_1^{\theta_2}, \widetilde{g}_2^{\gamma_2^*}\right) = \widetilde{e}\left(\frac{C_2 \widetilde{g}_1^{-\theta_3 \gamma_1^* \phi_t}}{H_1(m^*)^{\chi_i \gamma_1}}, \widetilde{g}_2\right), \tag{28}$$

where m^* and γ_1^*, γ_2^* are extracted from C_1, which is available to B since he has access to corresponding components of sk_t.

(2) Encryption Oracle: challenger B works the same as mentioned in original G-PKEET scheme.

(3) Verification Oracle: challenger B works similarly as mentioned in decryption oracle except for following the definition of Verify during output.

(4) Test Oracle: when decryptor index equals t, the test oracle processes submitted ciphertext using corresponding method mentioned in Section 6.2.1.

Otherwise, test oracle will handle query by situation. Let ciphertexts C, C' be queried ciphertexts, one of whose decryptor index is t and the other is not.

Situation 1. The adversary stores γ_2', corresponding to component $C_3' = \tilde{g}_1^{\gamma_2}$.

In this situation, the test oracle judges whether the following equation stands:

$$\frac{\tilde{e}\left(C_2/\tilde{g}_1^{\theta_2\gamma_2}, C_4'\right)}{\tilde{e}\left(C_8', C_9^{\theta_3}\right)} = \frac{\tilde{e}\left(C_2'/\tilde{g}_1^{\theta_2\gamma_2'}, C_4\right)}{\tilde{e}\left(C_8, (C_9')^{\theta_3}\right)}. \tag{29}$$

If the equation stands, test oracle outputs 1; otherwise, it outputs 0.

Situation 2. The adversary takes $\tilde{g}_1^{\alpha}$ as C_3'.

In this situation, the test oracle judges whether the following equation stands:

$$\frac{\tilde{e}\left(C_2/\tilde{g}_1^{\theta_2\gamma_2}, C_4'\right)}{\tilde{e}\left(C_8', C_9^{\theta_3}\right)} = \frac{\tilde{e}\left(C_2'/\tilde{g}_1^{\chi}, C_4\right)}{\tilde{e}\left(C_8, (C_9')^{\theta_3}\right)}. \tag{30}$$

Table 1: Scheme properties.

Scheme	Security model	Functionality
Yang et al. [1]	RO	PKEET
Tang [2]	RO	PKEET
Zhang et al. [4]	SM	PKEET
Zeng et al. [5]	SM	PKEET
Lee et al. [6]	SM	PKEET
Ling et al. [7]	RO	G-PKEET
Ours	SM	G-PKEET

Note. RO: random oracle model. SM: standard model.

If the equation stands, test oracle outputs 1; otherwise, it outputs 0.

Situation 3. The adversary himself has no idea of value γ_2' and directly sets component $C_3' = \tilde{g}_1^{\gamma_2}$.

In this situation, the test oracle will directly output 0 as a result if the equality test procedure is not aborted by outputting $\perp$ in other steps.

Since adversary does not know γ_2', θ_2, according to the definition of CDH assumption, he is also incapable of knowing $\tilde{g}_1^{\gamma_2'\theta_2}$. Thus probability for adversary to find out exact $\tilde{g}_1^{\gamma_2}$ that satisfies the following equation is negligible.

$$\frac{\tilde{e}\left(C_2/\tilde{g}_1^{\theta_2\gamma_2}, C_4'\right)}{\tilde{e}\left(C_8', C_9^{\theta_3}\right)} = \frac{\tilde{e}\left(C_2'/\tilde{g}_1^{\theta_2\gamma_2'}, C_4\right)}{\tilde{e}\left(C_8, (C_9')^{\theta_3}\right)}. \tag{31}$$

From the above, we believe that PPT adversary cannot distinguish Game 1 from Game 2 through exploiting inconsistency output which may happen in Situation 3.

Challenge. B makes additional modifications to challenge ciphertext of Game 1, embedding the third element of XDH challenge tuple $\tilde{g}_1^{\zeta}$ into it:

$$\begin{aligned}&\text{Invalid}_{m\gamma_1\gamma_2} \leftarrow_r G_T; C_1^* = \left(\text{Invalid}_{m\gamma_1\gamma_2}\right) Z_t^s, C_2^* = H_1(m)^{\chi_i\gamma_1}\tilde{g}_1^{\zeta}\tilde{g}_1^{\theta_3\phi_t\gamma_1}, C_3^* = \tilde{g}_1^{\alpha},\\ &C_4^* = \tilde{g}_2^{\gamma_1\theta_1\chi_i}, C_5^* = g^s, C_6^* = r, C_7^* = \left(u_t^t v_t^r w_t\right)^s.\end{aligned} \tag{32}$$

Superscript $t = H_1(C_1^*, C_2^*, C_3^*, C_4^*, C_5^*, C_8^*, C_9^*, C_{10}^*); \gamma_1 \leftarrow_r \mathbb{Z}_q; r, s \leftarrow_r \mathbb{Z}_p$.

Query Phase 2. Verification oracle is applied with the same change as it is on page 16. The test oracle is supplemented with following rule:

Challenger B aborts and outputs random bit when a ciphertext whose decryptor index is not t (we refer to it by j) satisfies the following formula:

$$\tilde{e}\left(\tilde{g}_1^{\gamma_1'}, \tilde{g}_2^{\phi_j\theta_3}\right) = \tilde{e}\left(\tilde{g}_1^{\theta_3}, \tilde{g}_2^{\gamma_1^*\phi_t}\right), \tag{33}$$

where component $\tilde{g}_2^{\phi_j\theta_3}$ is taken from gpk_j, γ_1' is extracted from component C_1 of queried ciphertext, and γ_1' corresponds to challenge ciphertext component C_2^*.

Since adversary has no idea of value $\theta_3, \gamma_1^*, \phi_t$, the probability of abortion owing to the reason above is negligible.

Guess Phase. If A recognizes the current game as Game 1, B outputs guess $\tilde{g}_1^{\zeta} = \tilde{g}_1^{\alpha\beta}$; otherwise it outputs the opposite guess.

Apparently, when $\tilde{g}_1^{\zeta} = \tilde{g}_1^{\alpha\beta}$, A's view is identical to his view in Game 1. Otherwise, his view is identical to that of Game 2. Since adversary A can be invoked to attack XDH problem, his advantage is expressed as

$$\text{distinguish}_{1,2} \le \int_{xdh}. \tag{34}$$

This is negligible when corresponding assumption stands.

Table 2: Computational efficiency.

Scheme	Enc	Dec	Test	Communication round(s) of test
Yang et al. [1]	3Exp	3Exp	$2P$	1
Tang [2]	5Exp	2Exp	4Exp	1
Zhang et al. [4]	6Exp	2BP + 1Exp	$2P$	1
Zeng et al. [5]	4Exp	4Exp	4Exp	1
Lee et al. [6]	$1P$ + 14Exp	$9P$ + 11Exp	$6P$ + 10Exp	1
Ling et al. [7]	5Exp	2Exp	$2P$ + 2Exp	1
Ours	9Exp	9Exp + $3P + 3P_3$	6Exp + $6P_3$	2

Note. Exp: the exponent computation. P: the type-1 bilinear pairing computation. P_3: the type-3 bilinear pairing computation.

In Game 2, the adversary 's advantage is negligible since the challenge tuple has no relation with m_b. Eventually, we can reduce the adversary's advantage in attacking IND-CCA2 Game as

$$\begin{aligned}\int_{IND-CCA2} &= \text{distinguish}_{0,1} + \text{distinguish}_{1,2} + \int_{\text{Game2}} \\ &\leq \int_{dbdh} + ADV^{CR} \\ &\quad + \frac{\text{query}}{p} + \int_{xdh} + 0.\end{aligned} \tag{35}$$

Thus, Theorem 3 is proved.

7. Comparison

7.1. Theoretical Analysis. In Table 1, we list out the security model and functionality of related PKEET schemes and G-PKEET schemes, showing that our scheme is the first G-PKEET scheme under standard model. All listed schemes obtain IND-CCA2 security against unauthorized users and OW-CCA2 security against authorized users except [1], since it does not introduce token mechanism and is only able to obtain OW-CCA2 security.

In Table 2, we compare the computational efficiency of our schemes with related PKEET schemes and G-PKEET schemes. The second to fourth column show the computational cost of Enc algorithm, Dec algorithm, and Test algorithm. In the aspect of computational efficiency, our scheme is acceptable comparing with other schemes listed.

The reason why computational cost of our scheme is higher than existing G-PKEET scheme [7] is that we add more ciphertext components to ensure our scheme satisfies new security and functional definition. And correspondingly, we have to supply decrypt, test algorithm with corresponding verification steps on these components. These verification steps guarantee scheme security at cost of increasing computational complexity.

Through the fifth column of the table, we remind readers that security properties of our scheme rely on a prerequisite that proxy should interact with user on whether the ciphertext to be tested is proper. This prerequisite implicitly requires a two-round communication for equality test service, i.e., the algorithm Test; otherwise the security properties cannot be completely guaranteed.

Table 3: Time consumption (ms).

Scheme	Enc	Dec	Test
Ling et al. [7]	82	37	59
Ours	1724	953	895

As for two generic PKEET schemes, we employed [25, 26] to instantiate Lee et al.'s scheme and use DDH assumption to construct hash proof system, on which Zeng et al.'s scheme is based.

7.2. Experimental Evaluation. We implement experimental performance analysis by using Java Pairing Based Cryptography (jPBC 2.0.0) as underlying cryptographic library [27]. The JDK version is Oracle jdk1.8.0_121. More detailedly, we, respectively, choose type-A curve and type-F curve provided by jPBC to perform symmetric bilinear pairing and asymmetric bilinear pairing. The field size bit length of all curves is at least 256. Our machine is equipped with Intel Core i7-10510U CPU 2.30 GHz processor and 16 GB RAM, running Ubuntu 16.04.

We compare time consumption between our scheme and Ling et al. [7] in aspect of three algorithms: Enc algorithm, Dec algorithm, and Test algorithm. For each algorithm, we run it for ten times and calculate its average time cost. The result is shown as in Table 3.

The gap between two schemes on computation time consumption reflects, on one hand, the gap in theory time complexity. On the other hand, type-A curve pairing operation provided in jPBC takes 10% of the time required by type-F curve pairing operation. Furthermore, the implicit restriction on public parameters in our scheme makes the field size bit length of G larger than $256 * 3$, three times as that of $\widetilde{G}$, and the computation time is equally increased. We conclude that constructing a more efficient G-PKEET scheme with equivalent security property is an issue that requires further discussion.

8. Conclusion

In this paper, we firstly broaden the current existing security notion of G-PKEET by granting specific adversaries restricted access to the equality test service. The new security notion ensures that unexpected privacy leakage can be avoided even when proxy may provide equality service to malicious users. To construct a solution that matches our definition, we secondly extend the functionality of G-PKEET scheme by allowing private key holder to detect whether a ciphertext is proper while proxy can optionally request private key holder to verify a ciphertext, before he conducts equality test on it. Eventually, our new G-PKEET scheme is acceptable in aspect of computational efficiency and proved to be OW-CCA2 secure against adversary authorized by group administrator and IND-CCA2 secure against unauthorized adversary. Furthermore, our scheme is the first G-PKEET proved to be secure under standard model.

Data Availability

The data used to support the findings of this study are included within the article.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

Acknowledgments

This research was supported by the NSFC-ISF Joint Scientific Research Program (61961146004) and Innovation Program of Shanghai Municipal Education Commission (2021-01-07-00-08-E00101).

References

[1] G. Yang, C. Tan, Q. Huang, and D. Wong, "Probabilistic public key encryption with equality test," in *Topics in Cryptology - CT-RSA 2010 CT-RSA 2010*, pp. 119–131, Springer, San Francisco, CA, USA, March 2010.

[2] Q. Tang, "Public key encryption schemes supporting equality test with authorisation of different granularity," *International Journal of Applied Cryptography*, vol. 2, no. 4, pp. 304–321, 2012.

[3] Q. Tang, "Public key encryption supporting plaintext equality test and user-specified authorization," *Security and Communication Networks*, vol. 5, no. 12, pp. 1351–1362, 2012.

[4] K. Zhang, J. Chen, H. Lee, H. Qian, and H. Wang, "Efficient public key encryption with equality test in the standard model," *Theoretical Computer Science*, vol. 755, pp. 65–80, 2019.

[5] M. Zeng, J. Chen, K. Zhang, and H. Qian, "Public key encryption with equality test via hash proof system," *Theoretical Computer Science*, vol. 795, pp. 20–35, 2019.

[6] H. Lee, S. Ling, J. Seo, H. Wang, and T. Youn, "Public key encryption with equality test in the standard model," *Information Scientist*, vol. 516, no. 89–108, 2020.

[7] Y. Ling, S. Ma, Q. Huang, X. Li, and Y. Ling, "Group public key encryption with equality test against offline message recovery attack," *Information Scientist*, vol. 510, no. 16–32, 2020.

[8] Y. Ling, S. Ma, Q. Huang, X. Li, Y. Zhong, and Y. Ling, "Efficient group id-based encryption with equality test against insider attack," *The Computer Journal*, vol. 64, no. 4, pp. 661–674, 2021.

[9] Y. Ling, S. Ma, Q. Huang, R. Xiang, and X. Li, "Group id-based encryption with equality test," in *Information Security and Privacy ACISP 2019*, pp. 39–57, Springer, Christchurch, New Zealand, July 2019.

[10] A. Kiayias, Y. Tsiounis, and M. Yung, "Group encryption," in *ASIACRYPT 2007*, pp. 181–199, Springer, Kuching, Malaysia, 2007.

[11] M. Bellare and P. Rogaway, "Random oracles are practical: a paradigm for designing efficient protocols," in *CCS '93*, pp. 62–73, Fairfax, Virginia, USA, 1993.

[12] R. Canetti, O. Goldreich, and S. Halevi, "The random oracle methodology, revisited," *Journal of the ACM*, vol. 51, no. 4, pp. 557–594, 2004.

[13] Q. Tang, "Towards public key encryption scheme supporting equality test with fine-grained authorization," in *Information Security and Privacy ACISP 2011*, pp. 389–406, Springer, Melbourne, Australia, July 2011.

[14] T. Wu, S. Ma, Y. Mu, and S. Zeng, "Id-based encryption with equality test against insider attack," in *Information Security and Privacy ACISP 2017, Auckland, New Zealand*, pp. 168–183, Springer, July 2017.

[15] J. Lai, R. Deng, S. Liu, and W. Kou, "Efficient cca-secure PKE from identity-based techniques," in *Proceedings of the Interlaken, Topics in Cryptology - CT-RSA 2010*, pp. 132–147, Springer, Switzerland, June2010.

[16] R. Cramer and V. Shoup, "Universal hash proofs and a paradigm for adaptive chosen ciphertext secure public-key encryption," in *Advances in Cryptology - EUROCRYPT 2002 EUROCRYPT 2002*, pp. 45–64, Springer, Amsterdam, The Netherlands, 2002.

[17] H. Lin, F. Gao, H. Zhang, Z. Jin, W. Li, and Q. Wen, "Public key encryption with equality test supporting flexible designated authorization in cloud storage," *IEEE Systems Journal*, vol. 16, no. 1, pp. 1460–1470, 2022.

[18] Z. Zhao and P. Zeng, "Efficient all-or-nothing public key encryption with authenticated equality test," *IEEE Access*, vol. 9, pp. 94099–94108, 2021.

[19] H. Lin, Z. Zhao, F. Gao et al., "Lightweight public key encryption with equality test supporting partial authorization in cloud storage," *The Computer Journal*, vol. 64, no. 8, pp. 1226–1238, 2021.

[20] H. Zhu, L. Wang, H. Ahmad, and D. Xie, "Pairing-free for public key encryption with equality test scheme," *IEEE Access*, vol. 9, pp. 77239–77249, 2021.

[21] X. J. Lin, L. Sun, H. Qu, and X. Zhang, "Public key encryption supporting equality test and flexible authorization without bilinear pairings," *Computer Communications*, vol. 170, pp. 190–199, 2021.

[22] S. Galbraith, K. Paterson, and N. Smart, "Pairings for cryptographers," *Discrete Applied Mathematics*, vol. 156, no. 16, pp. 3113–3121, 2008.

[23] L. Ballard, M. Green, B. Medeiros, and F. Monrose, "Correlation-resistant storage via keyword-searchable encryption," *IACR Cryptol. ePrint Arch.*vol. 417, p. 2005, 2005.

[24] V. Shoup, "Sequences of games: a tool for taming complexity in security proofs," *IACR Cryptol. ePrint Arch.*vol. 332, p. 2004, 2004.

[25] D. Boneh, E. Shen, and B. Waters, "Strongly unforgeable signatures based on computational diffie-hellman," in *Public Key Cryptography - PKC 2006 PKC 2006*, pp. 229–240, Springer, New York, NY, USA, April 2006.

[26] D. Boneh and X. Boyen, "Efficient selective-id secure identity-based encryption without random oracles," in *Advances in Cryptology - EUROCRYPT 2004 EUROCRYPT 2004*, pp. 223–238, Springer, Interlaken, Switzerland, May 2004.

[27] A. D. Caro and V. Iovino, "jpbc: Java pairing based cryptography," in *Proceedings of the 16th IEEE Symposium on Computers and Communications, ISCC 2011*, pp. 850–855, Kerkyra, Corfu, Greece, July 2011.