Abstract—The emergence of 5G networks will allow Cloud Computing providers to offer more convenient services. However, security and privacy issues of cloud services in 5G networks represent huge challenges. Recently, to improve security and privacy, a novel primitive was proposed by Ma et al. in TIFS 2015, called Public Key Encryption with Equality Test supporting Flexible Authorization (PKEET-FA). However, the PKEET scheme lacks verification for equality test results to check whether the cloud performed honestly. In this research, we expand the study of PKEET-FA and propose a verifiable PKEET scheme, called V-PKEET, which, to the best of our knowledge, is the first work that achieves verification in PKEET. Moreover, V-PKEET has been designed for three types of authorization to dynamically protect the privacy of data owners. Therefore, it further strengthens security and privacy in 5G networks.

Index Terms—Public key encryption, equality test, privacy-preservation, verification, Cloud Computing, 5G networks.

I. INTRODUCTION

EMPOWERED by new Radio Access Technologies (RATs), 5G wireless access solutions are designed to support a variety of applications. This will enable a significant growth in mobile use of cloud-based services, and an upwards trend in outsourcing of privacy sensitive data to cloud servers. 5G networks exacerbate the privacy problems intrinsic to Cloud Computing. For example, cloud storage can reduce the client storage burden but, in order to prevent the cloud from modifying the stored data, its integrity should always be verifiable [1] [2]. For the purpose of security and privacy-preservation, this data also needs to be encrypted before it is stored in cloud servers. For this reason, a large number of researchers have lately concentrated in supporting operations on encrypted data, such as, searchable encryption [3] [4] [5] and outsourced computing [6] [7]. In this paper, we focus on public key encryption with equality test (PKEET), which was firstly proposed by Yang et al. in CT-RSA 2010 [8]. PKEET is used to test whether two messages encrypted by using (possibly) two different public keys are identical. In such application of PKEET scheme as outsourced database, once receiving the trapdoors from users, the database manager can then partition the database in accordance with the encrypted messages without the need to interact with data users. PKEET has also various applications in Cloud Computing. For example, the property of equality test can be used to search ciphertexts and classify encrypted messages.

The paper uses as example an email system able to classify all emails by encrypted keywords generated by an email security monitor. Therefore, the monitor is in charge of producing encrypted sensitive keywords, and the cloud email server uses PKEET to search if such sensitive keywords are involved in the mails or not, without knowledge of the emails content. Suppose that Alice encrypts emails using her public key \( pk_A \), encryption will conceal content, and prevent some crucial information from being searched. To increase search precision, the encrypted emails are attached to a small number of keywords encrypted in PKEET system, and a single ciphertext expressed as \( C = \text{Enc}(M, pk_A)[|\text{PKEET}(W_1, pk_A)||PKEET(W_2, pk_A)||...||PKEET(W_n, pk_A)] \). Finally, Alice transfers the ciphertext \( C \) to the email cloud server for further classification. The key point of the PKEET scheme is that the cloud email server can check if the encrypted keywords are attached to an email including some given keywords generated by the email security monitor.

However, Yang et al. [8] showed that anyone has the ability to test if two ciphertexts are encrypted under the same plaintext without any authorization. In other words, the attacker can choose an user as target and generate a ciphertext using its public key, however, due to the function of equality test so that the attacker can guess the plaintext under the ciphertext. Therefore, it is essential to implement an authorization strategy to protect data owners privacy. Only after the users send delegations to the cloud, can the cloud provider perform the equality test. Furthermore, different applications demand different delegation strategies to dynamically protect the privacy of data owners. Recently, Ma et al. [9] designed a PKEET-FA scheme without verification for equality test results. However, in practice, the cloud may be malicious and, therefore, may compute and deliver to users a wrong result. Furthermore, the PKEET-FA scheme supports four scenarios S1-S4 with different delegation granularity levels. There are, however, two other scenarios S5-S6 with different levels of delegation which are not supported. One is receiver-specific level authorization(S5), i.e. all ciphertexts of Alice can only make a comparison with all ciphertexts of a specific receiver (e.g. Bob), but could not make a comparison with any ciphertext of anyone other than Bob. The other one is receiver-specific ciphertext-to-receiver level authorization(S6), i.e a specific ciphertext of Alice could only make a comparison with all ciphertexts of a specific receiver such as Bob, but could not make a comparison with any ciphertext of any receiver other than Bob.

S5. Assume that the receiver (Alice) and the email security monitor work in the same company. By obtaining the delegations from Alice and the email security monitor, respectively,
the cloud email server has the ability to test whether the emails encrypted by Alices public key include the keywords encrypted under the email security monitors public key. The key is that Alice intends to affirm that the equality test is surely performed on her ciphertexts and the specified monitors ciphertexts (see Fig.1(a)). In other words, Alice and a specific entity produce a trapdoor each to perform equality test in regard to all their ciphertexts.

S6. Assume that the email security monitor belongs to a public institution, different than the company where Alice works, and the institution works in a pay-per-keyword manner. The cloud email server only has the ability to test whether the specific email encrypted by Alices public key is contained in keywords encrypted under the specific monitors public key (see Fig.1(b)). In other words, Alice produces a trapdoor to perform the equality test on her specific ciphertext while the monitor produces a trapdoor to perform the equality test on all of its ciphertexts.

A. Our Contribution

The proposed scheme supports three types of authorization (See Fig.2) simultaneously, which are described in detail in the following. Type-1 authorization is the same as that in [9], however, in our construction, this type is more efficient. Type-2 authorization requires no proxies other than that in FG-PKEET [10] and FG-PKEET+ [11]. Type-3 authorization is novel and has not been implemented in earlier works.

1) Type-1
   Receiver level authorization: All ciphertexts of Alice (receiver) can make a comparison with all ciphertexts of any other receiver. (This type represents a coarse-grained delegation.)

2) Type-2
   Receiver-specific level authorization: All ciphertexts of Alice can only make a comparison with all ciphertexts of a specific receiver (e.g., Bob). However, it could not make a comparison with ciphertext of anyone other than Bob. (This type represents a fine-grained delegation)

3) Type-3
   Receiver-specific ciphertext-to-receiver level authorization or receiver-specific receiver-to-ciphertext level authorization: A specific ciphertext of Alice could only make a comparison with all ciphertexts of a specific receiver, such as Bob. However, it could not make a comparison with any ciphertext of any receiver other than Bob (or vice versa).

Furthermore, we propose the concept of enhanced PKEET, called verifiable PKEET (V-PKEET). It supports verification of results from an untrusted cloud server, in other words, we consider the cloud is malicious, i.e., that could compute a wrong result to users. In our system, the user checks whether the cloud server has devotedly performed the authorized equality test, which, to the best of our knowledge, has not been researched previously. Finally, the proposed scheme completes a satisfying security.

B. Related Works

1) Public Key Encryption with Keyword Search (PEKS): The primitive of public key encryption with keyword search was originally introduced by Boneh et al. [12]. In this research, a sender encrypts plaintexts using the receivers public key enabling the receiver to search over the ciphertexts in the asymmetric encryption system. The receiver generates a tag to check if a keyword embedded in the token is equal to the message underlying the ciphertext. Bellare et al. [13] showed an efficient searchable scheme based on a deterministic encryption algorithm to effectively complete retrieval of encrypted data. Boneh and Waters [14] presented a new searchable public key encryption (PKE) scheme which provided the functions of comparison search, ordinary subset search and connectives search. Later Curtmola et al. [15] proposed a secure searchable scheme, called dPEKS, which enhanced the security model of PEKS by taking advantage of a servers public key in the encryption process. Before long, Yau et al. [16] used the technique of proxy re-encryption to achieve bidirectional Re-dPEKS and Guo et al. [17] enhanced the scheme to be more efficient. These PKE schemes could produce a token associated with the keyword, and one would be capable of classifying the ciphertexts in accordance with the plaintexts, taking advantage of the token. Nevertheless, this approach has a weakness: a same message encrypted using different public keys cannot be divided into one category. To address this problem, we concentrate on studying PKEET.

2) Public Key Encryption with Equality Test (PKEET): The concept of public key encryption with equality test (PKEET) was firstly introduced by Yang et al. [8]. The study offers a new function to test whether two ciphertexts that could be generated under different public keys are the same plaintext. However, the scheme has no delegation mechanism for data owners to assign someone to test. To realize authorization on PKEET, Tang [10] proposed a novel scheme which provided fine-grained authorization strategy (FG-PKEET) for users. However, in this scheme, the two users need to interact to produce a random value. Besides, Tang [18] came up with a new primitive, called AoN-PKEET, to complete a coarse-grained delegation. Furthermore, Tang [11] applied FG-PKEET to a two-proxy system in which two agent servers needed to work together to achieve an equality test (FG-PKEET+). However, the proposal requires a protocol to interact between two agent servers, and this is too costly. In this paper, we improve the FG-PKEET+ and present the Type-2 authorization, where a proxy server is not required to participate such that it can reduce the communication cost. Huang et al. [19] introduced a new proposal, called PKE-AET, which can achieve the equality test between authorized ciphertexts and users through warrants. Lee et al. [20] identified a serious flaw and modified the PKE-AET so as to achieve the IND-CCA2 secure. Recently, Ma et al. [9] has proposed a scheme that supports various levels of authorizations simultaneously, called PKEET-FA. All the above-mentioned approaches lack verification for the result of equality test. This means that receivers cannot confirm whether the cloud server has faithfully performed the protocol. However, if a malicious cloud server deceives the user for
the purpose of self-interest, the equality test is meaningless. To innovate it, it is imperative to verify the return result of equality test from the malicious cloud.

3) **Verifiable Computation**: Verifiable computation plays a particularly important role in outsourced computing, especially in a malicious cloud environment. Generally speaking, verifiable computation protocols [21] [22] [23] [24] enable a client with weak computation ability to outsource the processed data to the cloud. The cloud then returns corresponding results, as well as proofs that the data has been processed correctly. Gennaro et al. proposed the concept of non-interactive verifiable computation in CRYPTO 2010 [21]. In this case, the client can verify the outsourced result through a proof. Shortly afterwards, Benabbas et al. [22] conceived the first practical verifiable computation scheme for large datasets. In [23], Fiore and Gennaro made use of a pseudo-random function with closed-form efficiency and designed two novel constructions for publicly verification: one was evaluation of large polynomials and another was matrix multiplication. Parno et al. [24] combined the concepts of verifiable calculation and attributed-based encryption to complete public delegation and public verification. Recently, Zheng et al. [25] presented a verifiable authorized set intersection scheme over ciphertexts. They expanded the single-accumulator scheme [26] [27] to test whether members belong to the set. In our paper, we consider the malicious cloud may compute the inaccurate result to
users and use variant of the technique in [25] to verify its authenticity.

C. Organization
In the next part of the paper, we introduce the preliminaries. Then we give the system model and introduce its formal definition and threat models in Section 3. Furthermore, a construction of V-PKEET is presented in Section 4, and security analyses are showed in the next part. Section 6 presents performance analyses. Finally, we conclude this paper in Section 7.

II. PRELIMINARIES

A. Bilinear Map
Let $G$, $G_1$ and $G_T$ be three cyclic groups of prime order $p$. We denote that $g$ and $g_1$ are generators of $G$ and $G_1$, respectively. A bilinear map $e : G \times G_1 \rightarrow G_T$ owns the following properties:
1) Bilinear: For any $S \in G$ and $T \in G_1$ and $u, v \in \mathbb{Z}_p$, $e(S^u, T^v) = e(S, T)^{uv}$.
2) Non-degenerate: $e(S, T) \neq 1$.
3) Computable: There is an efficient algorithm to compute $e(S, T)$ for any $S \in G$ and $T \in G_1$.

B. Computational Diffie-Hellman (CDH) Assumption

Given $(g, g^a, g^b)$, where $g$ is the generator of $G$ with prime order $p$ and $a, b \in \mathbb{Z}_p$. It is intractable to calculate $g^{ab}$.

C. Decisional Diffie-Hellman (DDH) Assumption

Given $(g^a, g^b, g^c)$, where $g$ is the generator of $G$ with prime order $p$ and $a, b, c \in \mathbb{Z}_p$. It is intractable to distinguish $(g^a, g^b, g^c)$ from $(g^a, g^b, g^d)$.

III. DEFINITION OF V-PKEET

A. System Model
In our system (see Fig.3), there are several entities involved: trusted authority (TA), data owners described as Alice and Bob, cloud server (CS).
1) Trusted Authority (TA). The TA is in charge of generating system public parameters that will be used by the CS and data owners, and of generating public/private key pairs for data owners.
2) Data Owners. Alice and Bob use their respective public key to encrypt their private data, called $M_A$ and $M_B$, then outsource the resulting ciphertexts denoted by $C_A$ and $C_B$ to the CS. At any point in time when the data owner intends to authorize the equality test, he (or she) will generate a trapdoor to delegate the CS to perform the test.
3) Cloud Server (CS). The CS stores the resulting ciphertexts, and performs the equality test algorithm but without decrypting ciphertexts. After the cloud has performed the equality test, it returns a proof to Alice (or Bob) which is used to verify the validity of the result.

B. Definition of V-PKEET

Definition 1. A V-PKEET cryptosystem has the following algorithms:
1) $\text{Setup}(\lambda)$: It inputs a security parameter $\lambda$, and generates a public parameter $pp$ as output.
2) $\text{KeyGen}(pp)$: It inputs the public parameter $pp$, and generates a public/secret key pair($pk_i$, $sk_i$) for receiver $U_i$ as output.
3) $\text{Enc}(M_i, pk_i)$: It inputs a message $M_i$ and the public key $pk_i$ of receiver $U_i$, and generates a ciphertext $C_i$ as output.
4) $\text{Dec}(C_i, sk_i)$: It inputs a ciphertext $C_i$ and the secret key $sk_i$ of receiver $U_i$, and returns a plaintext $M_i$ as output.

In order to complete Type-$\gamma$($\gamma = 1, 2, 3$) delegation for $U_i$ and $U_j$, we give the definitions of $\text{Auth}_i(\gamma = 1, 2, 3)$ algorithm to compute trapdoors for $U_i$’s $U_j$’s ciphertext (or ciphertexts), and $\text{Test}_i(\gamma = 1, 2, 3)$ algorithm to test if the two ciphertexts are encrypted under the identical plaintext.

Type-1 Authorization:
Type-1-1 $\text{Auth}_1(sk_i)$: This algorithm inputs the secret key $sk_i$, and calculates a trapdoor $td_{1,i}$ for $U_i$ as output.
Type-1-2 $\text{Auth}_2(M_i, pk_i)$: This algorithm uses $U_i$’s message $M_i$ and $U_j$’s public key as inputs and produces some auxiliary information $Di_i$ and $au_i$ as output, then gives $au_i$ to the cloud.
Type-1-3 $\text{Auth}_3(C_i, td_{1,i}, au_i)$: This algorithm inputs $U_i$’s ciphertext $C_i$, the trapdoor $td_{1,i}$, the auxiliary information $au_i$, $U_j$’s ciphertext $C_j$, the trapdoor $td_{1,j}$ and the auxiliary information $au_j$, and outputs 1 if $C_i$ and $C_j$ are encrypted under the same plaintext and 0 otherwise.

After that the cloud calculates a proof for $U_i$ (or $U_j$) which can verify if the cloud has devotedly executed the protocol.

Type-2 Authorization:
Type-2-1 \textit{Aux}(sk_i, pk_j): This algorithm inputs $U_i$'s secret key $sk_i$ and $U_i$'s public key $pk_j$, and calculates a trapdoor $td_{(2,i,j)}$ for $(U_i, U_j)$ as output. 

Type-2-2 \textit{Auxiliary}_{2}(M_i, pk_j): This algorithm uses $U_i$'s message $M_i$ and $U_i$'s public key $pk_j$ as inputs, and produces some auxiliary information $ Dig_i$ and $au_i$ as output, then gives $au_i$ to cloud. 

Type-2-3 Test$_2(C_i, td_{(2,i,j)}, au_i, C_j, td_{(2,j,i)}, au_j)$: It inputs $U_i$'s ciphertext $C_i$, the trapdoor $td_{(2,i,j)}$, the auxiliary information $ au_i$, $U_i$'s ciphertext $C_j$, the trapdoor $td_{(2,j,i)}$, and the auxiliary information $ au_j$, and outputs 1 if $C_i$ and $C_j$ are encrypted under the same plaintext and 0 otherwise. After that the cloud calculates a proof for $U_i$(or $U_j$) and returns it to $U_i$(or $U_j$) which can verify if the cloud has devotedly executed the protocol. 

Type-3 Authorization: 

Type-3-1(1) \textit{Aux}(3),(sk_i, C_i, pk_j): This algorithm inputs $U_i$'s secret key $sk_i$, its ciphertext $C_i$ and $U_i$'s public key $pk_j$, and computes a trapdoor $td_{(3,i,j)}$ for $(U_i, C_i, U_j)$ as output. 

Type-3-1(2) \textit{Aux}(3),(sk_j, pk_i): This algorithm inputs $U_j$'s secret key $sk_j$, $U_j$'s public key $pk_i$, and calculates a trapdoor $td_{(3,j,i)}$ for $(U_j, U_i)$ as output. 

Type-3-2(1) \textit{Auxiliary}(3),(M_i, pk_j): This algorithm inputs $U_i$'s message $M_i$ and $U_i$'s public key $pk_j$, and produces some auxiliary information $ Dig_i$ and $au_i$ as output, and gives $au_i$ to cloud. 

Type-3-2(2) \textit{Auxiliary}(3),(M_j, pk_i): This algorithm inputs $U_j$'s message $M_j$ and $U_j$'s public key $pk_i$, and produces some auxiliary information $ Dig_j$ and $au_j$ as output, and gives $au_j$ to cloud. 

Type-3-3 Test$_3(C_i, td_{(3,i,j)}, au_i, C_j, td_{(3,j,i)}, au_j)$: It uses $U_i$'s ciphertext $C_i$, the trapdoor $td_{(3,i,j)}$, the auxiliary information $ au_i$, $U_j$'s ciphertext $C_j$ the trapdoor $td_{(3,j,i)}$ and the auxiliary information $ au_j$ as inputs, and outputs 1 if $C_i$ and $C_j$ are encrypted under the same plaintext or 0 otherwise. At the same time, the cloud returns a proof to $U_i$(or $U_j$) which can verify if the cloud has devotedly executed the protocol. 

5) Prove: The cloud server generates a proof which allows verification whether it has faithfully performed the protocol. If $C_i$ and $C_j$ are encrypted under the same plaintext, the cloud server executes Prove1, or else the cloud server executes Prove2. In Prove1/Prove2, $H(T_i)$ is generated in Test algorithm as $U_i$'s ciphertext conversion section. 

(1) Prove1($H(T_i), H(T_j)$): This algorithm inputs $H(T_i), H(T_j)$, and outputs a proof to $U_i$(or $U_j$). 

(2) Prove2($H(T_i), H(T_j)$): This algorithm inputs $H(T_i)$ and $H(T_j)$, and outputs a proof sends to $U_i$(or $U_j$). 

6) Verify$(sk_i, Dig_i, rslt_i, proof_i)$: This algorithm inputs $U_i$'s secret key $sk_i$, digest $Dig_i$, result $rslt_i$, and the proof $proof_i$, and outputs 1 if the cloud has faithfully executed the protocol and 0 otherwise. 

C. Security Models 

In this paper, we consider two types of adversaries for the security model of the V-PKEET scheme: 

1) Type-I adversary: For $Type-\gamma(\gamma = 1, 2, 3)$ authorization, the adversary owns $Type-\gamma$ trapdoor and is looking forward to disclose the message under the challenge ciphertext. 

2) Type-II adversary: For $Type-\gamma(\gamma = 1, 2, 3)$ authorization, the adversary does not obtain $Type-\gamma$ trapdoor and is looking forward to tell apart that $C_i^*$ is the encrypted value of which plaintext.

In the following, we give the definition of OW-CCA security for $Type-\gamma(\gamma = 1, 2, 3)$ authorization against Type-I adversary in V-PKEET. 

Game 1: Let $A_1$ is a Type-I adversary and the underlying receivers with the subscript $k(1 \leq k \leq n)$. The game between the adversary $A_1$ and the challenger $C$ is described as below: 

1) Setup: $C$ first executes the Setup algorithm and gives $A_1$ the outputting public parameter $pp$. Then $C$ executes $KeyGen$ to produce $n$ pairs of public/secret keys $(pk_i, sk_i)(1 \leq i \leq n)$ and gives all $pk_i$ to $A_1$.

2) Phase1: the adversary $A_1$ is permitted to ask all sorts of queries for a polynomial number of times, as follows. The restriction is that $t < t$ does not appear in the key retrieve queries.

- Key retrieve queries $< i >$: $C$ returns $sk_i$ to the adversary $A_1$. 
- Decryption queries $< i, C_i >$: $C$ executes $Dec$ algorithm on input $(C_i, sk_i)$ to decrypt $C_i$ using $sk_i$ and returns the output to the adversary $A_1$. 
- Authorization queries: For $Type-\gamma(\gamma = 1, 2, 3)$ authorization, $-when inputs < i >, C$ returns $td_{(1,i)}$; $-when inputs < i, j >, C$ returns $td_{(2,i,j)}$; $-when inputs < i, C_i, j >, C$ returns $td_{(3,i,j)}$. 

3) Challenge: $C$ selects a random message $M_i \in M$ , executes $C_i^* = Enc(M_i, pk_i)$ and sends $C_i^*$ to the adversary $A_1$ as the challenge ciphertext. 

4) Phase 2: $A_1$ requests the same as the queries in Phase 1. The restrictions are that $< t >$ is not used as an input of the key retrieve queries; $< t, C_i^* >$ is not used as an input of the decryption queries.

5) Guess: $A_1$ returns $M_i' \in M$ . In the game, when $M_i' = M_i, A_1$ is the winner. The probability that $A_1$ is the winner is denoted by

$$Adv_{\gamma-\text{PKEET}, A_1}^{OW-CCA, Type-\gamma}(\lambda) = Pr[M_i = M_i'](\gamma = 1, 2, 3)$$

(1)

Definition 2: We say that the V-PKEET is OW-CCA secure for $Type-\gamma(\gamma = 1, 2, 3)$ authorization if for any PPT adversaries $A_1$, its advantage $Adv_{\gamma-\text{PKEET}, A_1}^{OW-CCA, Type-\gamma}(\lambda)$ is negligible in the security parameter $\lambda$.

Now we give the definition of the IND-CCA security for the $Type-\gamma(\gamma = 1, 2, 3)$ authorization against Type-II adversary in V-PKEET below.

Game 2: Let $A_2$ is a Type-II adversary and the underlying receiver with the subscript $k(1 \leq k \leq n)$. The game between the adversary $A_2$ and the challenger $C$ is described below.

1) Setup: $C$ first executes the Setup algorithm and gives $A_2$ the outputting public parameter $pp$. Then $C$ exe-
cutes KeyGen to produce \(n\) pairs of public/secret keys \((pk_i, sk_i)\) \((1 \leq i \leq n)\) and gives all \(pk_i\) to \(A_2\).

2) **Phase 1**: \(A_2\) requests the same as the queries in **Game 1**.

3) **Challenge**: \(A_2\) submits two plaintexts \(M_0, M_1 \in M\) of same length. \(C\) randomly selects a bit \(b \in \{0, 1\}\), executes and returns \(C^*_b = Enc(M_b, pk_t)\) to the \(A_2\) as the challenge ciphertext and returns \(C^*_t\) to the \(A_2\) as the challenge ciphertext.

4) **Phase 2**: \(A_2\) requests the same as the queries in **Phase 1**. The restrictions are that
- \(< t >\) is not used as an input in the key retrieve queries.
- \(< t, C^*_t >\) is not used as an input in the decryption queries.

   For **Type - \(\gamma\)(\(\gamma = 1, 2, 3\))** authorization queries,
   \(<\gamma = 1 >\) is not used as an input in the authorization queries;
   \(<\gamma = 2 >\) is not used as an input in the authorization queries;
   \(<\gamma = 3 >\) is not used as an input in the authorization queries.

5) **Guess**: \(A_2\) returns \(b' \in \{0, 1\}\). In this game, when \(b = b', A_2\) is the winner. The probability that \(A_2\) is the winner is denoted by

   \[
   Adv_{\text{V-PKEET}, A_2}^{\text{IND-CCA}, \text{Type-}\gamma} (\lambda) = \Pr[b = b'] - \frac{1}{2} \quad (\gamma = 1, 2, 3) \tag{2}
   \]

**Definition 3**: We say that the **V-PKEET** is IND-CCA secure for Type - \(\gamma\)(\(\gamma = 1, 2, 3\)) authorization if for any **PPT** adversaries \(A_2\), its advantage \(Adv_{\text{V-PKEET}, A_2}^{\text{IND-CCA}, \text{Type-}\gamma}(\lambda)\) is negligible in the security parameter \(\lambda\).

### IV. OUR V-PKEET SCHEME

Now we present an effective V-PKEET scheme.

1) **Setup(\(\lambda\))**: It inputs a security parameter \(\lambda\) and returns the public parameters \(pp\). Firstly, the algorithm constructs a bilinear map, where the \(G, G_1\), and \(G_T\) are three groups with prime order \(p\) and \(g, g_1\) are the generators of \(G, G_1\) respectively and randomly chooses \(a \in \mathbb{Z}_p^*\) and calculates \(g^a, g_1^a\). Then it selects four cryptographic hash functions: \(H : G_T \rightarrow \mathbb{Z}_p, H_1 : \{0, 1\}^* \rightarrow \{0, 1\}^l, H_2 : G_1 \rightarrow \{0, 1\}^{l_1 + l_2}, H_3 : G \rightarrow G\), where \(l_1\) and \(l_2\) represent lengths of element of \(G\) and \(\mathbb{Z}_p\), respectively. \(pp = \{p, e, G, G_1, G_T, g, g_1, g^a, g_1^a, H_1, H_2, H_3\}\).

2) **KeyGen(pp)**: It inputs \(pp\) and randomly chooses \(x_i, y_i \in \mathbb{Z}_p^*\), then outputs the key pair for receiver \(U_i\):

   \((pk_i, sk_i) = ((X_i = g^{x_i}, Y_i = g^{y_i}), (x_i, y_i))\)

3) **Enc(M_i, pk_i)**: It inputs \(U_i\)'s plaintext \(M_i\) and public key \(pk_i\) and randomly selects \(r_{i,1}, r_{i,2} \in \mathbb{Z}_p^*\), then generates a ciphertext \(C_i = (C_{i,1}, C_{i,2}, C_{i,3}, C_{i,4}, C_{i,5})\) as following:

   \[
   \begin{align*}
   C_{i,1} &= g_1^{r_{i,1}}, C_{i,2} = g^{r_{i,2}}. \\
   C_{i,3} &= H_2(X_i^{r_{i,1}}) \oplus (M_i || r_{i,1}) \\
   C_{i,4} &= Y_1^{r_{i,2}} \cdot H_3(M_i) \\
   C_{i,5} &= H_1(C_{i,1} || C_{i,2} || C_{i,3} || C_{i,4} || M_i || r_{i,1})
   \end{align*}
   \]

4) **Dec(C_i, sk_i)**: It inputs a ciphertext \(C_i = (C_{i,1}, C_{i,2}, C_{i,3}, C_{i,4}, C_{i,5})\) and private key \(sk_i\), then computes \(M_i || r_{i,1}\). If \(C_{i,1} = g_1^{r_{i,1}}\) and \(C_{i,5} = H_1(C_{i,1} || C_{i,2} || C_{i,3} || C_{i,4} || M_i || r_{i,1})\) both hold, it returns \(M_i\); otherwise, it returns an error indicator \(\perp\).

**Type 1 Authorization**:

- **Type 1-1 Auth1(sk_i)**: The algorithm returns a trapdoor \(td_{(1,i)} = y_i\).

- **Type 1-2 Auxiliary1(M_i, pk_i)**: \(U_i\) generates auxiliary information \(au_i = (cph_i, \sigma_i)\) for each message of \(U_i\). Firstly, \(U_i\) computes \(H(e(H_3(M_i), g_1))\) for each message of \(U_i\) and \(Dg_{g_i} = g^h(e(H_3(M_i), g_1))\) for verification. Then \(U_i\) transforms \(Dg_{g_i}\) to \(cph_i\) using \(U_i\)'s public key \(pk_{cph}\), \(cph_i = (g^{w_{i,1}}, g^{w_{i,2}}, Dg_{g_i} Y_1^{(w_{i,1} + w_{i,2})})\), where \(w_{i,1}, w_{i,2} \leftrightarrow Z_p\). Finally, \(U_i\) signs \(cph_i\) and gets \(\sigma_i \leftarrow \text{sigSign}(\text{sigSk}, cph_i)\) to prevent the cloud from tampering \(cph_i\), which could bring a wrong ciphertext. Similarly, \(U_j\) can run the same algorithm to generate a trapdoor and auxiliary information \(au_j = (cph_j, \sigma_j)\).

**Type 1-3 Test1(C_i, td_{(1,i)}, au_i, C_j, td_{(1,j)}, au_j)**: Firstly, it calculates

   \[
   C_{i,4}/C_{i,2}^{w_{i,1}} = H_3(M_i) \\
   C_{j,4}/C_{j,2}^{w_{j,1}} = H_3(M_j)
   \]

If \(H_3(M_i) = H_3(M_j)\) holds set \(T_i = e(H_3(M_i), g_1)\), \(T_j = e(H_3(M_j), g_1)\) return 1 and then run \(Prove1(H(T_i), H(T_j)) \rightarrow proof_{ij}\) for \(U_i\), or return 0 and run \(Prove2(H(T_i), H(T_j)) \rightarrow proof_{ij}\) for \(U_i\) otherwise. (Similarly, the cloud can generate \(proof_{ij}\) for \(U_j\)).

**Type 2 Authorization**:

- **Type 2-1 Auth2(sk_i, pk_i)**: The algorithm returns a trapdoor \(td_{(2,i)} = (X_i^{w_{i,1}}, X_i^{w_{i,2}})\).

- **Type 2-2 Auxiliary2(M_i, pk_i)**: \(U_i\) computes \(H(e(H_3(M_i), X_i^{w_{i,1}}))\) for each message of \(U_i\) and lets \(Dg_{g_i} = g^h(e(H_3(M_i), X_i^{w_{i,1}}))\) for verification. Then \(U_i\) transforms \(Dg_{g_i}\) to \(cph_i\) using \(U_i\)'s public key \(pk_{cph}\), \(cph_i = (g^{w_{i,1}}, g^{w_{i,2}}, Dg_{g_i} Y_1^{(w_{i,1} + w_{i,2})})\), where \(w_{i,1}, w_{i,2} \leftrightarrow Z_p\). Finally, \(U_i\) runs \(\sigma_i \leftarrow \text{sigSign}(\text{sigSk}, cph_i)\) to obtain a signature \(\sigma_i\) on ciphertext \(cph_i\). \(U_i\) generates auxiliary information \(au_i = (cph_i, \sigma_i)\) for each message of \(U_i\). Similarly, \(U_j\) can run the same algorithm to generate a trapdoor \(td_{(2,j)} = (X_j^{w_{j,1}}, X_j^{w_{j,2}})\) and auxiliary information \(au_j = (cph_j, \sigma_j)\).
Type-2-3 Test3(Ct, td(2, i, j), au, Ct, td(2, i, j), auj): The algorithm calculates

\[ T_i = \frac{e(C_{1,4}, X^x_j)}{e(C_{1,2}, X^x_{j(yi)})} = e(H_3(M_i), g_1)^x_{i,j} \]

\[ T_j = \frac{e(C_{1,4}, X^x_{ji})}{e(C_{1,2}, X^x_{ji})} = e(H_3(M_j), g_1)^x_{i,j} \]

If \( T_i = T_j \) holds, return 1 and run \( \text{Prove}(H(T_i), H(T_j)) \to \text{proof}_i \) for \( U_i \), or return 0 and run \( \text{Prove}(2)(H(T_i), H(T_j)) \to \text{proof}_i \) for \( U_i \) otherwise. (Similarly, the cloud can generate \( \text{proof}_j \) for \( U_j \))

Type-3 Authorization:

Type-3-1(1) \( \text{Auxiliary}(3, i)(M_i, pk_i) \): The algorithm calculates a trapdoor \( td(3, i, C_i) = (C_{1,2}^{m_i}, Y^{x_i}_{j}, X^x_{ji}, X^x_{j(yi)}) \).

Type-3-1(2) \( \text{Auxiliary}(3, j)(M_j, pk_j) \): The algorithm calculates a trapdoor \( td(3, j, C_j) = (X^x_{ji}, X^x_{j(yi)}) \).

Type-3-2(1) Auxiliary(3, i)(M_i, pk_i): \( U_i \) computes \( \tau = e(H_3(M_i), X^x_{ji}) \) for message of specific \( C_i \) and let \( \text{Sign}_{g_i} = g^{h(sei+\alpha)} \) for verification. Then \( U_i \) transforms \( \text{Sign}_{g_i} \) to \( \text{cph}_{i} \) using \( U_i \)'s public key \( pk_i \), \( \text{cph}_{i} = (g^{w_{i,1}}, g^{w_{i,2}}, D_i Y_{1}^{w_{i,1}+w_{i,2}}) \), where \( w_{i,1}, w_{i,2} \in \mathbb{Z}_p \). Finally, \( U_i \) runs \( \sigma_i \leftarrow \text{Sign}(\text{sig}(\text{sk}_i, \text{cph}_{i})) \). To obtain a signature \( \sigma_i \) on ciphertext \( \text{cph}_{i} \), \( U_i \) generates auxiliary information \( \text{aux} = (\text{cph}_{i}, \sigma_i) \) for the message of specific \( C_i \).

Type-3-2(2) Auxiliary(3, j)(M_j, pk_j): \( U_j \) computes \( \tau = e(H_3(M_j), X^x_{ji}) \) for each message of \( U_j \) and let \( \text{Sign}_{g_j} = g^{h(sej+\alpha)} \) for verification. Then \( U_j \) transforms \( \text{Sign}_{g_j} \) to \( \text{cph}_{j} \) using \( U_j \)'s public key \( pk_j \), \( \text{cph}_{j} = (g^{w_{j,1}}, g^{w_{j,2}}, D_i Y_{1}^{w_{j,1}+w_{j,2}}) \), where \( w_{j,1}, w_{j,2} \in \mathbb{Z}_p \). Finally, \( U_j \) runs \( \sigma_j \leftarrow \text{Sign}(\text{sig}(\text{sk}_j, \text{cph}_{j})) \) to obtain a signature \( \sigma_j \) on ciphertext \( \text{cph}_{j} \). \( U_j \) generates auxiliary information \( \text{aux} = (\text{cph}_{j}, \sigma_j) \) for each message of \( U_j \).

Type-3-3 Test3(Ct, td(3, i, j), au, Ct, td(3, j, i), auj): The algorithm calculates

\[ T_i = \frac{e(C_{1,4}, X^x_j)}{e(C_{1,2}, Y^{x_{ji}}_{j(yi)})} = e(H_3(M_i), g_1)^x_{i,j} \]

\[ T_j = \frac{e(C_{1,4}, X^x_{ji})}{e(C_{1,2}, X^x_{ji})} = e(H_3(M_j), g_1)^x_{i,j} \]

If \( T_i = T_j \) holds, return 1 and run \( \text{Prove}(H(T_i), H(T_j)) \to \text{proof}_i \) for \( U_i \), or return 0 and run \( \text{Prove}(2)(H(T_i), H(T_j)) \to \text{proof}_i \) for \( U_i \) otherwise. (Similarly, the cloud can generate \( \text{proof}_j \) for \( U_j \))

5) Prove:

(1) \( \text{Prove}(H(T_i), H(T_j)) \to \text{proof}_i \):
In this case, \( H(T_i) \neq H(T_j) \), \( \text{rslt}_i = \phi \).

- Let \( P(x) = x \) and compute \( g^{P(x)} \) by replacing \( x \) with \( \alpha \).
- Let \( M(x) = x \) and \( N(x) = H(M(x)) + x \), and find two polynomials \( M'(x), N'(x) \) so that \( M(x)N(x) + N(x)N'(x) = 1 \) mod \( p \) by satisfying \( \text{gcd}(M(x), N(x)) = 1 \). Then computing \( (g^{M'(\alpha)})(g^{M'(\alpha)})(g^{N'(\alpha)})(g^{P(\alpha)}) \) by replacing \( x \) with \( \alpha \). \( \text{Witness} = (g^{M'(\alpha)}, g^{M'(\alpha)}, g^{N'(\alpha)}, g^{P(\alpha)}) \), set \( \text{proof}_i = \text{Witness}, \text{cph}, \sigma_j, \text{rslt}_j \).

(2) \( \text{Prove}(2)(H(T_i), H(T_j)) \to \text{proof}_i \):

In this case, \( H(T_i) \neq H(T_j) \), \( \text{rslt}_i = \phi \).

- Let \( P(x) = x + H(T_j) \) and calculate \( g^{P(x)} \) by replacing \( x \) with \( \alpha \).
- Let \( M(x) = x + H(T_i) \) and \( N(x) = x + H(T_j) \), and find two polynomials \( M'(x), N'(x) \) so as to \( M(x)N(x) + N(x)N'(x) = 1 \) mod \( p \) by utilizing \( \text{gcd}(M(x), N(x)) = 1 \). Then computing \( (g^{M'(\alpha)})(g^{M'(\alpha)})(g^{N'(\alpha)})(g^{P(\alpha)}) \) by replacing \( x \) with \( \alpha \). \( \text{Witness} = (g^{M'(\alpha)}, g^{M'(\alpha)}, g^{N'(\alpha)}, g^{P(\alpha)}) \), \( \text{proof}_i = \text{Witness}, \text{cph}, \sigma_j, \text{rslt}_j \).

6) \( \text{Verify}(sk_i, \text{Dig}_i, \text{rslt}_i, \text{proof}_i) \):

On inputs \( \text{rslt}_i \) and \( \text{proof}_i \), \( U_i \) validates whether the cloud server has trustedly performed the V-PKET scheme as follows:

- Validate the completeness of \( cph_j \) by executing \( \text{sig}\text{Verify}(\text{sig}(\text{pk}_j, \text{cph}_j)) \). If the verification is not passed, then aborts; otherwise continue to execute the following.

- Decrypt \( cph_j \) using private key \( sk_i \) according to

\[ \text{Dig}_j = \text{Dig}_j g^{y_{(t,j+1),j}} / (g^{t_j}, g^{t_j}) \]

- If \( \text{rslt}_i \) is empty, let \( Y_i = \phi \). Otherwise decrypt \( \text{rslt}_i \) to obtain the message \( M_i \), and compute \( Y_i \) according to each type of authorization as follows:

- For Type-1 Authorization: compute \( Y_i = H(e(H_3(M_i), g_1)) \);
- For Type-2 Authorization: compute \( Y_i = H(e(H_3(M_i), g_1)^x_{i,j}) \);
- For Type-3 Authorization: compute \( Y_i = H(e(H_3(M_i), g_1)^{x_{ji}}) \).

- Continue to verify the result. If the verification is not passed, it returns 0; otherwise returns 1.

- If \( Y_i = \phi \), compute \( g^{P(\alpha)} \) according to \( P(x) = x + Y_i \). Otherwise, set \( P(x) = 1 \) and \( g^{P(\alpha)} = g_1 \).

- If \( g^{P(\alpha)} \neq \text{Dig}_i g_1 \), return 0. Otherwise keep on running the following.

- If \( g^{P(\alpha)} \neq \text{Dig}_i g_1 \), return 0. Otherwise keep on running the following.

- If \( g^{P(\alpha)} \neq \text{Dig}_i g_1 \), return 0. Otherwise return 1.

If it outputs 1, it means that the two users' ciphertexts are the same plaintext and the cloud faithfully executes the protocol. Similarly, \( U_j \) can perform the same algorithm to check if the cloud has devotedly executed the equality test.

V. Security Analysis

Theorem 1: Our V-PKET has OW-CCA security for Type – γ(γ = 1, 2, 3) authorization against Type – I adversary under the random oracle model.

Proof: Assume an adversary with advantage \( \varepsilon \) in the game given in Game 1.

Let \( A_1 \) is a probabilistic polynomial time \( T \) adversary breaking the OW-CCA security. We assume that \( A_1 \) requests at most \( q_{H_1} \) hash queries, \( q_{H_2} \) hash queries,
This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2017.2716971, IEEE Access

The OW-CCA security is proven by a string of games as follows. Let $H$, $H_1$, $H_2$, $H_3$ hash queries, $G$, $G_1$, $G_2$ decryptions and $q_{Aut}$ authorization queries for $Type = \gamma = 1, 2, 3$ authorization. Note that if the same request is made many times in an oracle query, the same reply will be obtained. The OW-CCA security is proven by a string of games as follows.

**Game 1.0:** We give the game defined in Definition 3 as follows.

1. **pp** $\leftarrow \{p, e, G, G_1, G_T, g, g_1, g_1^\alpha, g_1^\beta, H, H_1, H_2, H_3\},$
   \[\forall 1 \leq i \leq n, x_i, y_i \leftarrow Z_p^*, X_i = g_1^{x_i}, Y_i = g_1^{y_i}.\]
2. **state** $\leftarrow A_{\gamma}^{H_1, H_2, H_3, \gamma, D, Aut}(\{X_i, Y_i\})_{i=1}^n$ these oracles are defined as below.

   - $o_{H_1}$ query $< v_1 >$: It is identical with that in Game 1.0.
   - $o_{H_2}$ query $< v_2 >$: It is identical with that in Game 1.0.
   - $o_{H_3}$ query $< v_3 >$: It is identical with that in Game 1.0.

3. $M_t \leftarrow G^*, r_{t_1}, r_{t_2} \leftarrow Z_p^*, W^* \leftarrow \{0, 1\}^{1+t_2}, C_t^* = (C_t^*{\gamma}, C_t^*{1_2}, C_t^*{1_3}, C_t^*{4}, C_t^*{4_5})$ is defined as follows:
   \[C_t^*{\gamma} = g_1^{r_1_{\gamma}}, C_t^*{1_2} = g_1^{r_{1_2}}, C_t^*{1_3} = h_2(X_t^{1_3}) \oplus (M_t \oplus r_{t_1}),\]
   \[C_t^*{4} = Y_t^{1_4} \cdot H_3(M_t)\]
   \[C_t^*{4_5} = H_1(C_t^*{1_2}, C_t^*{1_3}, C_t^*{4}, M_t \oplus r_{t_1}).\]

4. $M'_t \leftarrow A_{\gamma}^{H_1, H_2, H_3, \gamma, D, Aut}(state, C^*_t).$ Denoted by $S_{1,0}$ the event that $M_t = M'_t$ in Game 1.0. We have the following.

   \[\varepsilon_0 = \operatorname{Adv}_{\gamma}^{OW-CCA_Type - \gamma}(q_{H_1}, q_{H_2}, q_{H_3}, q_G, q_{Aut}) = Pr[S_{1,0}](\gamma = 1, 2, 3).\]

**Game 1.1:**

1. **pp** $\leftarrow \{p, e, G, G_1, G_T, g, g_1, g_1^\alpha, g_1^\beta, H, H_1, H_2, H_3\},$
   \[\forall 1 \leq i \leq n, x_i, y_i \leftarrow Z_p^*, X_i = g_1^{x_i}, Y_i = g_1^{y_i}.\]
2. **state** $\leftarrow A_{\gamma}^{H_1, H_2, H_3, \gamma, D, Aut}(\{X_i, Y_i\})_{i=1}^n$ these oracles are defined as below.

   - $o_{H_1}$ query $< v_1 >$: It is identical with that in Game 1.1.
   - $o_{H_2}$ query $< v_2 >$: It is identical with that in Game 1.1.
   - $o_{H_3}$ query $< v_3 >$: It is identical with that in Game 1.1.

3. $M_t \leftarrow G^*, r_{t_1}, r_{t_2} \leftarrow Z_p^*, W^* \leftarrow \{0, 1\}^{1+t_2}, C_t^* = (C_t^*{\gamma}, C_t^*{1_2}, C_t^*{1_3}, C_t^*{4}, C_t^*{4_5})$ is defined as follows:
   \[C_t^*{\gamma} = g_1^{r_1_{\gamma}}, C_t^*{1_2} = g_1^{r_{1_2}}, C_t^*{1_3} = W^* \oplus (M_t \oplus r_{t_1}),\]
   \[C_t^*{4} = Y_t^{1_4} \cdot H_3(M_t)\]
   \[C_t^*{4_5} = H_1(C_t^*{1_2}, C_t^*{1_3}, C_t^*{4}, M_t \oplus r_{t_1}).\]

4. $M'_t \leftarrow A_{\gamma}^{H_1, H_2, H_3, \gamma, D, Aut}(state, C^*_t).$ Denoted by $S_{1,0}$ the event that $M_t = M'_t$ in Game 1.1. Let

\[\varepsilon_1 = Pr[S_{1,1}]\]

Because of the idealness of the random oracle, $Pr[E_1]$ is negligible. Refer to the Difference Lemma in [28], we have

\[|\varepsilon_1 - \varepsilon_0| = Pr[E_1].\]
Because $H_1$ and $H_3$ are viewed as random oracles, the $\varepsilon_2$ is negligible. According to the aforementioned analysis, combining Eqs.(3)-(8), we have

$$\varepsilon \leq Pr[E_1] + Pr[E_2] + \varepsilon_2$$

(9)

The equation is negligibility under the CDH problem in $G_1$. The Theorem 1 has been proved completely.

Theorem 2: Our V-PKETT has IND-CCA security for $Type = \gamma (\gamma = 1, 2, 3)$ authorization against $Type = II$ adversary under CDH assumption and DDH assumption with the random oracle model.

Proof: Assume an adversary has the advantage $\varepsilon$ in the game given in Game 1. Let $A_2$ be a probabilistic polynomial time $Type = II$ adversary breaching the IND-CCA security of our scheme. $A_2$ requests at most $q_{H_1}$ $H_1$ hash queries, $q_{H_2}$ $H_2$ hash queries, $q_{H_3}$ $H_3$ hash queries, $q_K$ key retrieve queries, $q_d$ decryption queries and $q_{Aut}$ authorization queries for $Type = \gamma (\gamma = 1, 2, 3)$ authorization. Note that if the same request is made many times in an oracle query, the same reply will be obtained.

The security is proven by a battery of games as follows.

Game 2.0: We give the game that defined in Definition 3 as follows.

1) $pp \leftarrow \{ p, e, G, G_1, G_T, g, g_1, g_2, y_1, y_2, H, H_1, H_2, H_3 \}$,

\[ \forall 1 \leq i \leq n, x_i, y_i \leftarrow Z_p^*, X_i = g_{x_i}^y, Y_i = g_{y_i}^y. \]

2) $(M_0, M_1) \leftarrow A^{o_{Aut}}_{2_{H_1}, o_{H_2}, o_{H_3}, o_K, o_{D, D}, o_{Aut}}((X_1, Y_1))^n$.

These oracles are issued the same as Game 1.1.

3) $b \leftarrow \{ 0, 1 \}, r_{t_1}, r_{t_2} \leftarrow Z_p, C^* = (C_{t_1,1}, C_{t_2,2}, C_{t_3,3}, C_{t_4,4}, C_{t_5,5})$ is computed as follows:

$$C^*_{t_1,1} = g_1^{r_{t_1}^{-1}}, C^*_{t_2,2} = g^{r_{t_2}^{-1}}$$

$$C^*_{t_3,3} = H_2(X^{r_{t_1}}_t), (M_0)_{r_{t_1}}$$

$$C^*_{t_4,4} = Y^{r_{t_2}}_t, H_3(M_0)$$

$$C^*_{t_5,5} = H_1(C^*_{t_1,1}, ||C^*_{t_2,2}, ||C^*_{t_3,3}, ||C^*_{t_4,4}, ||M_0)_{r_{t_1}}$$

(13)

Add the tuple $((C^*_{t_1,1}), Z_2, (M_0))$ to table $T_2$ for $H_2$.

4) $b' \leftarrow A^{o_{Aut}}_{2_{H_1}, o_{H_2}, o_{H_3}, o_K, o_{D, D}, o_{Aut}}((C^*)^*)$ We called the event $b = b'$ $S_{2,0}$ in this game. So we have

$$\varepsilon_0 = Pr[b \equiv b'_E]\leftarrow \gamma (q_{H_1}, q_{H_2}, q_{H_3}, q_K, q_{D, D}, q_{Aut})$$

$$= Pr[S_{2,0}](\gamma = 1, 2, 3)$$

(10)

Clearly,

$$|\varepsilon_0 - 1/2| = \varepsilon$$

(11)

holds.

Game 2.1:

1) $pp \leftarrow \{ p, e, G, G_1, G_T, g, g_1, g_2, y_1, H, H_1, H_2, H_3 \}$,

\[ \forall 1 \leq i \leq n, x_i, y_i \leftarrow Z_p^*, X_i = g_{x_i}^y, Y_i = g_{y_i}^y. \]

2) $(M_0, M_1) \leftarrow A^{o_{Aut}}_{2_{H_1}, o_{H_2}, o_{H_3}, o_K, o_{D, D}, o_{Aut}}((X_1, Y_1))^n$.

These oracles are issued the same as Game 1.1.

3) $b \leftarrow \{ 0, 1 \}, r_{t_1}, r_{t_2} \leftarrow Z_p, W^*_2 \leftarrow \{ 0, 1 \}^{l_{t_1}+l_2}, C^* = (C_{t_1,1}, C_{t_2,2}, C_{t_3,3}, C_{t_4,4}, C_{t_5,5})$ is computed as follows:

$$C^*_{t_1,1} = g_1^{r_{t_1}^{-1}}, C^*_{t_2,2} = g^{r_{t_2}^{-1}}$$

$$C^*_{t_3,3} = W^*_2 \oplus (M_0)_{r_{t_1}}$$

$$C^*_{t_4,4} = Y^{r_{t_2}^{-1}}, H_3(M_0)$$

$$C^*_{t_5,5} = H_1(C^*_{t_1,1}, ||C^*_{t_2,2}, ||C^*_{t_3,3}, ||C^*_{t_4,4}, ||M_0)_{r_{t_1}}$$

(14)

Add the tuple $((C^*_{t_1,1}), W^*_2, (M_0))$ to table $T_2$ for $H_2$.

4) $b' \leftarrow A^{o_{Aut}}_{2_{H_1}, o_{H_2}, o_{H_3}, o_K, o_{D, D}, o_{Aut}}((C^*)^*)$ We called the event $b = b'$ $S_{2,1}$ in this game. Let

$$\varepsilon_1 = Pr[S_{2,1}]$$

(12)

Because of the idealness of the random oracle, $Pr[E_2]$ is negligible. Refer to the Difference Lemma in [28], the following inequality holds

$$|\varepsilon_1 - \varepsilon_0| = Pr[E_1]$$

(13)

Game 2.2:

1) $pp \leftarrow \{ p, e, G, G_1, G_T, g, g_1, g_2, y_1, H, H_1, H_2, H_3 \}$,

\[ \forall 1 \leq i \leq n, x_i, y_i \leftarrow Z_p^*, X_i = g_{x_i}^y, Y_i = g_{y_i}^y. \]

2) $(M_0, M_1) \leftarrow A^{o_{Aut}}_{2_{H_1}, o_{H_2}, o_{H_3}, o_K, o_{D, D}, o_{Aut}}((X_1, Y_1))^n$.

These oracles are issued the same as Game 1.1.

3) $b \leftarrow \{ 0, 1 \}, r_{t_1}, r_{t_2} \leftarrow Z_p, W^*_2 \leftarrow \{ 0, 1 \}^{l_{t_1}+l_2}, C^* = (C_{t_1,1}, C_{t_2,2}, C_{t_3,3}, C_{t_4,4}, C_{t_5,5})$ is computed as follows:

$$C^*_{t_1,1} = g_1^{r_{t_1}^{-1}}, C^*_{t_2,2} = g^{r_{t_2}^{-1}}$$

$$C^*_{t_3,3} = W^*_2 \oplus (M_0)_{r_{t_1}}$$

$$C^*_{t_4,4} = Y^{r_{t_2}^{-1}}, H_3(M_0)$$

$$C^*_{t_5,5} = H_1(C^*_{t_1,1}, ||C^*_{t_2,2}, ||C^*_{t_3,3}, ||C^*_{t_4,4}, ||M_0)_{r_{t_1}}$$

(15)

Add the tuple $((C^*_{t_1,1}), W^*_2, (M_0))$ to table $T_2$ for $H_2$.

4) $b' \leftarrow A^{o_{Aut}}_{2_{H_1}, o_{H_2}, o_{H_3}, o_K, o_{D, D}, o_{Aut}}((C^*)^*)$ We called the event $b = b'$ $S_{2,2}$ in this game. Let

$$\varepsilon_2 = Pr[S_{2,2}]$$

The game is the same as the distribution in Game 2.1 except that the event $E_2$ occurs, in other words $g_1^{r_{t,1}^*}$ is requested to the oracle $H_2$. It is worth noting that the secret key $x_1$ is never acted as the response that the attacker requests. Consequently, $Pr[E_2]$ is negligible under the CDH problem in $G_1$. Refer to the Difference Lemma in [28], the following inequality holds

$$|\varepsilon_2 - \varepsilon_1| = Pr[E_2]$$

(16)
Game 2.3:
1) \( pp \leftarrow \{p, e, G, G_1, G_T, g, g_1, g_2, g_3, Y, H, H_1, H_2, H_3\}, \forall 1 \leq i \leq n, x_i, y_i \leftarrow Z_p^2, X_i = g_1^{x_i}, Y_i = g_2^{y_i}. \)
2) \((M_0, M_1) \leftarrow A_2^{\{H_1, H_2, H_3, \sigma_H, \sigma_D, \sigma_A\}}(\{X_i, Y_i\})^n, \)

These oracles are defined as below.

\( o_{H_1}, query < v_1 >: \) Same as that in Game 2.2 except that if \( v_1 = (C^*_{t1})_{\{C^*_{t2} | C^*_{t3} | C^*_{t4} | M_b | r_{t1}, t\}}, \)

The event is denoted by \( E_3. \)

\( o_{H_3}, query < v_2 >, o_{K}, query < v_3 >, o_{K}, query < i >, o_{PKEET}, query < i >, \)

These queries are issued identical to Game 2.2.

3) \( b \leftarrow \{0, 1\}, r_{t1}, r_{t2} \leftarrow Z_p, W_{r_{t2}} \leftarrow \{0, 1\}^{l_1}, l_2, \)

\( W_{r_{t2}} \leftarrow \{0, 1\}^{l_1}, C^*_{t1} = (C^*_{t1, 1}, C^*_{t1, 2}, C^*_{t1, 3}, C^*_{t1, 4}), \)

\( C^*_{t1} \) is computed as follows:

\[ C^*_{t1, 1} = g_1^{t_1}, C^*_{t1, 2} = g_1^{r_{t1}}, C^*_{t1, 3} = H_1^l, \quad \]

\[ C^*_{t1, 4} = Y^{r_{t1}, H_2(M_b)}, C^*_{t1, 5} = W_3^{l_2}. \]

Add the tuple \( (\{C^*_{t1, 1}, W_3^{l_2} \oplus M_b(r_{t1})\}) \) to the table \( T_2 \)

for \( H_2 \) and \( (C^*_{t1, 1}, C^*_{t1, 2}, C^*_{t1, 3}, C^*_{t1, 4}, M_b) \) to the table \( T_1 \)

for \( H_1. \)

4) \( b' \leftarrow A_2^{\{H_1, H_2, \sigma_H, \sigma_D, \sigma_A\}}(C^*_{t1}), \)

We called the event \( b = b' \) in this game. Let \( \varepsilon_3 = Pr[S_{2,3}]. \)

The game is the same as the distribution in Game 2.2 only if the event \( E_3 \) occurs, in other words \( (C^*_{t1, 1} | C^*_{t1, 2} | C^*_{t1, 3} | C^*_{t1, 4} | M_b | r_{t1}, t) \) is requested to the random oracle \( H_1 \). We have

\[ |\varepsilon_3 - \varepsilon_2| \leq Pr[E_3]. \]

is negligible under the CDH problem in \( G_1. \)

On referring to the proof of semantic security of ElGamal scheme [28], we note that it is obvious to validate that \( |\varepsilon_3 - 1/2| \) is negligible under the DDH problem in \( G. \)

According to the aforementioned analysis, combining Eqs.(10)-(17), we have

\[ |\varepsilon_0 - \varepsilon_3| \leq Pr[E_1 + Pr[E_2] + Pr[E_3]. \]

The equation is negligible in the oracle under the CDH problem in \( G_1 \) and DDH problem in \( G. \)

It is worth noting that \( \varepsilon = |\varepsilon_0 - 1/2| \) and \( |\varepsilon_3 - 1/2| \) is negligible, so \( \varepsilon \) is negligible. The Theorem 2 has been proved completely.

VI. COMPARISON

Table I compares the computational complexity between previous and our PKEET schemes in terms of Type-1 authorization (coarse-grained authorization). We observe that our design has mildly smaller space in terms of the public key, the secret key and the ciphertexts than PKEET-FA (Type-1) [9] and costs a bit less than PKEET-FA (Type-1) in terms of encryption, decryption and test, noting that we only consider the equality test rather than the cost of generating the proof for verification. Moreover, in our authorization, we add the function of verification for the result from the cloud server.

Table II compares the computational complexity between previous and our PKEET schemes in the other two authorizations (fine-grained authorizations). Compared with [11], our Type-2 authorization requires no interactive protocol to produce the trapdoor \( \tau d \) and completes an additional function of verification. Type-3 authorization is firstly presented in our scheme, which could protect the users’ privacy more finely-grained. Our Type-2 and Type-3 authorizations have very small computational overheads in the light of Aut algorithm. Furthermore, the Type-2 and Type-3 authorizations have considerably larger computational overheads over equality tests. Because they need several bilinear pairing operations and a bilinear pairing costs about four times longer than an exponentiation in a conventional desktop computer environment according to the experimental results in [29] [30]. With respect to the security, same as [9] [11] [18], V-PKEET accomplishes OW-CCA authorized security and IND-CCA unauthorized security. Our scheme achieves verification, while the existing schemes do not take this into account.

VII. CONCLUSION

In this research, we expand the study of PKEET-FA, and design a verifiable PKEET scheme, called V-PKEET, which, to the best of our knowledge, is the first PKEET scheme.
providing a verification mechanism to check whether the cloud server has devotedly executed the equality test. Furthermore, V-PKEET achieves three types of authorization to protect the dynamic privacy of data owners. Finally, we prove that our scheme achieves OW-CCA authorized security and IND-CCA unauthorized security under the CDH assumption and DDH assumption in a random oracle model. This proposed scheme can make the cloud computing in 5G networks more secure.

ACKNOWLEDGMENT

The work was supported by the National Natural Science Foundation of China (No. 61572001, No. 61502008), The Natural Science Foundation of Anhui Province (No. 1508085QF132, No. 1708085QF136), Doctoral Research Start-up Foundation of Anhui University. The authors are very grateful to the anonymous referees for their detailed comments and suggestions regarding this paper.

REFERENCES

Ming Wang received the B.S. degree in computer science from Anhui University, Hefei, China, in 2014. She is currently pursuing the M.S. degree in the school of computer science and technology, Anhui University. Her research interests include data security and privacy protection technology.

Hong Zhong is a Professor (from 2009) and Executive Dean of the School of Computer Science and Technology, Anhui University, China. She received PhD degree in University of Science and Technology of China in 2005. Her research interests cover network and information security.

Jie Cui is now an Associate Professor in the School of Computer Science and Technology, Anhui University. He received PhD degree in University of Science and Technology of China in 2012. He has published over 20 papers. His research interests include network and information security.

Lu Liu is the Professor of Distributed Computing in the University of Derby, United Kingdom. Prof Liu received his PhD degree from University of Surrey, UK (funded by DIF DTC) and MSc in Data Communication Systems from Brunel University, UK. Prof Liu's research interests are in areas of cloud computing, service computing, computer networks and peer-to-peer networking. He is a Fellow of British Computer Society (BCS).

Virginia N. L. Franqueira is currently a senior lecturer at the University of Derby, UK. Prior to that, she held a lecturer position at the University of Central Lancashire (UK), a postdoc research position at the University of Twente (NL), and worked as an information security consultant (UK). She received her Ph.D. in Computer Science from the University of Twente (NL) in 2009, and her M.Sc. from the Federal University of Espirito Santo (BR). She is a member of the IEEE Computer Society and the British Computer Society. Her topics of research interest include security engineering, risk management and estimation, attack modelling and external insider threat. For more information about her see http://www.derby.ac.uk/staff/virginia-franqueira/.