Public key encryption with equality test from generic assumptions in the random oracle model

Lee, Hyung Tae; Ling, San; Seo, Jae Hong; Wang, Huaxiong

2019

Lee, H. T., Ling, S., Seo, J. H., & Wang, H. (2019). Public key encryption with equality test from generic assumptions in the random oracle model. Information Sciences, 500, 15-33. doi:10.1016/j.ins.2019.05.026

https://hdl.handle.net/10356/142922

https://doi.org/10.1016/j.ins.2019.05.026

Public Key Encryption with Equality Test from Generic Assumptions in the Random Oracle Model

Hyung Tae Lee[a], San Ling[b], Jae Hong Seo[c,*], Huaxiong Wang[b]

[a] *Division of Computer Science and Engineering, College of Engineering, Chonbuk National University, Jeonju, Republic of Korea*

[b] *Division of Mathematical Sciences, School of Physical & Mathematical Sciences, Nanyang Technological University, Singapore, Singapore*

[c] *Department of Mathematics & Research Institute for Natural Sciences, Hanyang University, Seoul, Republic of Korea*

Abstract

Public key encryption with equality test (PKEET) is a variant of classical public key encryption (PKE) with the special functionality of an equality test, and can be used in many applications such as in keyword search on encrypted data and for efficient management by partitioning encrypted data in the cloud. Since the original proposal of Yang et al. (CT-RSA, 2010), several subsequent proposals to improve the efficiency or functionality of PKEET have been reported.

We present a PKEET construction from generic assumptions in the random oracle model. In particular, whereas previous results require number-theoretic assumptions or strictly stronger generic assumptions such as the existence of secure hierarchical identity-based encryption, our proposal requires only the existence of cryptographic hash functions and secure PKE schemes satisfying a special property, called *randomness extractability*. Informally, randomness extractability means that one can recover the randomness used in a ciphertext when given a secret key corresponding to a public key for the ciphertext. We investigate the fact that PKE schemes satisfying this property can be designed by the Fujisaki-Okamoto (FO) transformation, which is the widely utilized method to obtain secure PKE schemes from basic cryptographic primitives in the random oracle model. As a result, in combination with the FO transformation, we obtain a PKEET construction in the random oracle model if there exist a one-way PKE scheme, a one-time secure symmetric key encryption scheme, collision-resistant and one-way hash functions, and a pseudorandom function. In this sense, we remark that our PKEET construction is derived from fundamental generic assumptions only.

Keywords: Public key encryption, equality test, random oracle model

1. Introduction

Public key encryption (PKE) is one of the most important primitives in modern cryptography. Classical PKE schemes generally guarantee at least the semantic security of encrypted messages, but the range of their usage is relatively restricted to basic applications, such as 'secure communication systems. Lately, many applications have been demanding additional PKE functionality, even though they allow partial leakage of information of encrypted data. Several kinds of PKE schemes have been proposed in response to these demands. Notably, schemes capable of supporting additional functionality by considering a trade-off between security and functionality in the sense that a user who has a sub-key can perform some pre-determined operations over encrypted data while obtaining some information of encrypted data, have been suggested. In general, such a variant of PKE schemes is known as functional encryption [7].

*Corresponding author

Email addresses: hyungtaelee@chonbuk.ac.kr (Hyung Tae Lee), lingsan@ntu.edu.sg (San Ling), jaehongseo@mju.ac.kr (Jae Hong Seo), hxwang@ntu.edu.sg (Huaxiong Wang)

Recently, many researchers have been trying to develop theories and applications of functional encryption. There are two notable research directions for functional encryption. One of which is to design functional encryption for arbitrary functions [17, 18]. The current state of the art result of this research direction, however, seems impractical or not yet provably secure [10, 11]. The other direction, which we follow, is to design practical functional encryption for specific functionality and there are notable results in this research direction, which includes from classical identity-based encryption (IBE) [5], public key searchable encryption [1, 4] and recently proposed inner production encryption [2, 3].

In this paper, we deal with a subclass of general functional encryption schemes that enables equality tests on ciphertexts to be performed, even though they may be encrypted by different public keys. We call such the subclass *public key encryption with equality test* (PKEET). Its features resemble those of another class of functional encryption, so-called PKE with keyword search (PKEKS) [4]. PKEKS allows equality tests to be performed between a message in a ciphertext and a keyword embedded in a sub-key (it has different names in the literature, e.g., token and trapdoor). Basically, both primitives can be applied to manage encrypted databases efficiently, but the most remarkable difference between them is that PKEET supports equality tests on ciphertexts under *different* public keys as well as under the same public key, whereas PKEKS supports tests on ciphertexts under the same public key only.

Before introducing our results, let us clarify a PKEET system model that we consider in this paper. Among several existing variations of PKEET systems, our work follows Tang's all-or-nothing PKEET system model [33]: It consists of three entities, a sender(s), a receiver(s), and a tester(s). When a sender wants to pass his data to a receiver, he encrypts his data using a receiver's public key and sends a generated ciphertext to the receiver. The receiver may decrypt the given ciphertext using his secret key and/or store it at the server. Once a need arises, the receiver issues a sub-key for equality tests on all his ciphertexts to a tester. Since then, the tester who can access the server can perform equality tests on ciphertexts under public keys of receivers who passed the sub-keys to the tester.

A PKEET system under Tang's all-or-nothing model has various application scenarios in practice. As an example, let us consider spam filtering in an (encrypted) email service. Suppose that all emails of each user are stored as encrypted for supporting privacy and keywords in each email are stored as encrypted by appending to the email for providing keyword search over encrypted emails on this system efficiently as well. On the other hand, to maintain the security of the system, the system needs to monitor stored emails and thus needs to check keywords appended to each email. For this purpose, if PKEET is exploited for encrypting keywords, the system can request a sub-key for equality tests to each user and then he/she issues and passes it to the server. Since then, the server can generate a ciphertext of a keyword that it wants to monitor by the server itself and perform equality tests. Besides the above scenario, PKEET can be also applied to an internet-based personal health record system [31], secure outsourced database management [34], and so on.

Due to its wide applicability in practice, there have been proposed various PKEET constructions based on number-theoretic or generic assumptions. Yang et al. [34] first proposed the basic concept of PKEET and its instantiation. In their PKEET system, anyone can perform equality tests on ciphertexts publicly. Soon after, Tang proposed variants of Yang et al.'s PKEET scheme that designate persons who can perform equality tests by issuing them with a sub-key [31, 32, 33]. Among them, we follow Tang's all-or-nothing PKEET system model [33], described in the above. This was followed by subsequent proposals [19, 20, 21, 26, 27] for improving efficiency or providing additional functionality, such as supporting different levels of authority for equality tests. However, all the aforementioned works are based on the hardness of the computational Diffie-Hellman (CDH) problem or its variants in the random oracle model.

Motivation and Contribution. Very recently, the first generic PKEET construction [22] was proposed under Tang's all-or-nothing PKEET system. More precisely, it relies on not any number-theoretic assumptions, but standard primitives such as hierarchical identity-based encryption (HIBE) and digital signatures. Furthermore, the PKEET construction in [22] is the first construction without relying on the random oracles, that is, it is secure in the standard model.

Nevertheless, there still remains a challenge in this promising primitive. From a theoretical standpoint, it is quite natural and important to determine fundamental assumptions for designing each specific cryptographic construction. Contrary to all the prior constructions, the generic PKEET construction in [22] re-

quires a *stronger* primitive, HIBE. Indeed, there is a black-box separation result between PKE and IBE (and thus that between PKE and HIBE) [6]. Furthermore, PKEET schemes that employ neither (H)IBE nor pairing/lattice structures, on which the most practical (H)IBE instantiations are built, are known to exist. Therefore, (H)IBE does not seem essential for PKEET. From a practical standpoint, the realization of HIBE is usually inefficient in comparison with that of PKE. There are two notable approaches for realizing HIBE, pairing-based and lattice-based. However, both approaches require more expensive operations in comparison with group-based PKE realizations. In this sense, we need to address the following natural question for PKEET functionality.

Can we build a generic construction for PKEET from fundamental primitives such as secure PKE schemes and one-way functions, even in the random oracle model?

This paper provides a positive answer for the above question by establishing the first generic method for PKEET from basic primitives in the random oracle model. More precisely, our construction is obtained from a PKE scheme satisfying the indistinguishability against adaptive chosen ciphertext attacks (IND-CCA2) with a mild property, called *randomness extractability*, and cryptographic hash functions modeled as random oracles. Informally, randomness extractability means that it is possible to recover the randomness used in a ciphertext when given a secret key corresponding to a public key for the ciphertext. We remark that PKE schemes satisfying this property can be easily designed by applying the Fujisaki-Okamoto (FO) transformation [16], which is the well-known technique to obtain IND-CCA2 secure PKE schemes from standard cryptographic primitives in the random oracle model. (See Section 5.1 for the details.)

We note that when proving the security of a cryptographic scheme in the random oracle model, a cryptographic hash function exploited in the scheme is replaced by a random oracle that is a function which returns a truly random element from the range of that hash function. In reality, random oracles do not exist and it was reported that some of cryptographic constructions which were proven secure in the random oracle model are vulnerable in practice [9, 13, 23]. However, since almost all cryptographic constructions designed in the random oracle model are more efficient than those in the standard model, many researchers and practitioners steadily pay attention to cryptographic constructions in the random oracle model. By the similar reason, we also expect that our solution presented in this paper provides more efficient specific PKEET schemes than instantiations of the Lee et al.'s generic construction in the standard model [22].

Now, let us elaborate our construction a bit. Similarly to existing (semi-)generic PKEET constructions [21, 22], our encryption algorithm begins by generating two ciphertexts of message M and its hash value $\mathcal{H}_1(M)$, respectively,

$$C_1 = \mathsf{PKE.Enc}(pk_1, M; \sigma_1) \text{ and } C_2 = \mathsf{PKE.Enc}(pk_2, \mathcal{H}_1(M); \sigma_2),$$

where $\mathsf{PKE.Enc}$ is an encryption algorithm of PKE scheme PKE that satisfies randomness extractability, pk_1, pk_2 are public keys of PKE, σ_1, σ_2 are randomness, and $\mathcal{H}_1$ is a cryptographic hash function. Here, C_1 is utilized to recover a message M using a secret key sk_1 corresponded to pk_1 and C_2 is utilized to perform equality tests. If a tester has a secret key sk_2, which is corresponded to pk_2, she can obtain $\mathcal{H}_1(M)$ by decrypting C_2 with sk_2 and perform equality tests by comparing $\mathcal{H}_1(M)$ values.

However, the above provisional encryption algorithm is not IND-CCA2 secure by the following attack: When a ciphertext $CT = (C_1, C_2)$ of message M is given, the adversary can generate another ciphertext $CT' = (C_1', C_2)$ or $CT'' = (C_1, C_2')$ by executing $C_1' = \mathsf{PKE.Enc}(pk_1, M'; \sigma_1')$ or $C_2' = \mathsf{PKE.Enc}(pk_2, \mathcal{H}_1(M'); \sigma_2')$ for some message M' and randomness σ_1', σ_2'. Thereafter, the adversary can query on CT' or CT'' to the decryption oracle and it may return M if $M = M'$ or $\perp$ if $M \neq M'$. Thus, the adversary can distinguish whether a target ciphertext contains which message between two candidates.

To prevent adaptive chosen ciphertext attacks, we give a link between C_1 and C_2 by adding a hash value of C_1, C_2, σ_1, and σ_2,

$$C_3 = \mathcal{H}_2(C_1, C_2, \sigma_1, \sigma_2),$$

to the ciphertext, where $\mathcal{H}_2$ is a cryptographic hash function. Informally, to obtain a valid ciphertext from the target ciphertext, the adversary should know at least one of σ_1 and σ_2 to generate C_3 part correctly. However, if the exploited encryption scheme PKE is IND-CCA2 secure, the adversary cannot obtain σ_1 or σ_2 from C_1 and C_2. Furthermore, C_3 looks random in the random oracle model and thus the adversary cannot extract any information about σ_1 and σ_2 from C_3 as well. On the other hand, the decryption algorithm can recover σ_1 and σ_2 from C_1 and C_2 by using secret keys corresponded to pk_1 and pk_2 since PKE has randomness extractability. Therefore, the decryption algorithm can check the validity of C_3 by computing $\mathcal{H}_2(C_1, C_2, \sigma_1, \sigma_2)$. We remark that the previous work given by Lee et al. [22] cannot prevent this type of attacks without heavy assumptions in the standard model. In fact, they resolved this issue by employing HIBE and signature schemes.

We formally show that our construction achieves one-wayness under adaptive chosen ciphertext attacks (OW-CCA2) against Type-I adversaries who have a trapdoor for the target ciphertext and is IND-CCA2 secure against Type-II adversaries who do not have a trapdoor for the target ciphertext. Those are proven under assuming that the employed PKE scheme is IND-CCA2 secure and randomness extractable, and the exploited hash functions are modeled as random oracles.

As mentioned earlier, we again emphasize that randomness extractability is not an onerous property since most IND-CCA2 secure PKE schemes already satisfy it. To support this argument, we show that IND-CCA2 secure PKE schemes obtained by using the FO transformation in the random model, have the randomness extractability. (Refer to Section 5.1 for the details.) As a result, in combination of our proposed construction with the FO transformation, we obtain a PKEET construction if there exist a one-way PKE scheme, a one-time secure symmetric key encryption (SKE) scheme, collision-resistant and one-way hash functions, and a pseudorandom function in the random oracle model. Therefore, we reach the conclusion that our PKEET construction is derived from fundamental generic assumptions only.

Finally, we provide a comparison of our generic construction and main PKEET schemes under Tang's all-or-nothing PKEET system model. Our comparison shows that an instantiation of our generic construction is much more efficient than that of the generic construction in the standard model [22]. Furthermore, it also shows that our instantiation has comparable performance to Tang's specific PKEET scheme [33] in terms of computational costs and parameter sizes. Refer to Section 5.2 for the detailed comparison.

1.1. Related Work

In this subsection, we introduce previous results related to our work.

PKEET under Specific Setting. Since Yang et al. [34] proposed a concept and an instantiation of PKEET that allows anyone to perform equality tests on ciphertexts, there have been proposed various PKEET schemes to improve efficiency and/or to support advanced functionality. Tang [33] proposed the all-or-nothing PKEET scheme that allows only a tester authorized by two users to perform equality tests on all ciphertexts of those users. Later, Ma et al. [27] designed a PKEET scheme that hires a delegated party, who is the only entity that can perform equality tests by interacting with the server that stores the ciphertexts. Huang et al. [19] proposed the scheme that a user separately issues sub-keys on all of his/her ciphertexts or only a specified ciphertext, but it has a security flaw and was fixed by Lee et al. [20]. Subsequently, Ma et al. [26] presented the scheme that simultaneously supports four types of flexible authorization. Later, Lin et al. [24] proposed an improved construction under Ma et al.'s model [26] without the use of bilinear maps. Very recently, Qu et al. [29] introduced the concept of certificateless PKEET and presented its instantiation. All results listed above are secure in the random oracle model and there exists the only PKEET scheme under the specific setting in the standard model, presented by Zhang et al. [35].

PKEET under Generic Setting. To the best of our knowledge, there have been proposed two (semi-) generic constructions [21, 25] for PKEET in the random oracle model. The main difference between them is techniques exploited to give a link between two ciphertexts of a message and its hash value to prevent adaptive chosen ciphertext attacks. To this end, Lee et al.'s construction [21] appends an instance of the computational Diffie-Hellman (CDH) problem to a message and its hash value. As a result, their construction additionally requires the CDH assumption and so the authors of [21] argued that their construction is semi-generic.

Subsequently and independently with our result, Lin et al. [25] removed the need of the CDH assumption by replacing CDH instances with points on a randomly selected polynomial. (Refer to Section 5.2 for the details.)

In the standard model, on the other hand, there is the only result [22] that provides a generic construction for PKEET obtained by exploiting HIBE and signature schemes.

Public Key Encryption with Keyword Search. Public key encryption with keyword search (PKEKS), which was firstly presented in [4], is a cryptographic primitive that allows equality tests between a message in a ciphertext and a keyword embedded in a sub-key. It is very similar to PKEET in the way that both support equality tests. But, the main difference between PKEKS and PKEET is that the latter supports equality tests on ciphertexts under different public keys as well as under the same public key, whereas the former supports tests on ciphertexts under the same public key only.

In PKEKS, Abdalla et al. [1] first presented a generic construction, which transforms anonymous IBE into a secure PKEKS. Following their work, there were presented some studies on generic transformations from IBE to PKEKS under various settings, e.g., under the existence of designated tester [30] and under the secure-channel free setting [15]. We omit detailed comparison of them and generic constructions for PKEET because they support different functionalities ultimately.

1.2. Organization of the Paper

In the next section, we provide preliminaries including syntax and security models for PKEET. Section 3 presents our proposal for generic PKEET constructions from PKE with randomness extractability and cryptographic hash functions, and Section 4 gives its security proofs. In Section 5, we show that randomness extractability is easily achievable from the FO transformation and also provide comparisons of ours with existing PKEET constructions under the same system model as ours. Finally, we give concluding remarks in Section 6.

2. Preliminaries

In this section, we provide basic concepts of PKE and PKEET, and present properties of cryptographic hash functions, which will be utilized in our construction. We begin by introducing notations that will be used throughout the paper.

Notation. $\mathsf{Alg} \to a$ denotes that a is an outcome of Alg if Alg is an algorithm. On the contrary, $\mathsf{Alg} \not\to a$ denotes that Alg does not output a. When A is a set and a is an element in A, $a \xleftarrow{\$} A$ denotes that a is uniformly and independently sampled from A.

We say that a function $\nu : \mathbb{N} \to \mathbb{R}$ is negligible if for all positive polynomials $p(\cdot)$ and sufficiently large λ, $\nu(\lambda) \leq \frac{1}{p(\lambda)}$.

2.1. Public Key Encryption and IND-CCA2 Security

We present the definition of PKE and its security notion that will be utilized in this paper.

Definition 1 (Public Key Encryption)**.** *A public key encryption scheme* PKE *consists of the following three polynomial-time algorithms* ($\mathsf{PKE.KeyGen}, \mathsf{PKE.Enc}, \mathsf{PKE.Dec}$)*.*

- $\mathsf{PKE.KeyGen}(\lambda)$*: It takes a security parameter* λ *as an input and returns a public key* pk *and a secret key* sk*. It is assumed that* pk *includes the information of the plaintext space* $\mathcal{P}$ *and the randomness space* $\mathcal{R}$ *of the scheme.*
- $\mathsf{PKE.Enc}(pk, M; \sigma)$*: Given the public key* pk*, a message* $M \in \mathcal{P}$ *and a randomness* $\sigma \in \mathcal{R}$*, it returns a ciphertext* C*. For simplicity, we sometimes omit the randomness* σ *used for encryption.*
- $\mathsf{PKE.Dec}(sk, C)$*: Given the secret key* sk *and a ciphertext* C*, it returns a plaintext* M'*.*

A PKE scheme PKE is *correct* if for any $M \in \mathcal{P}$ and security parameter λ, it holds that

$$\Pr[\mathsf{PKE.Dec}(sk, \mathsf{PKE.Enc}(pk, M)) \to M] = 1$$

where $\mathsf{PKE.KeyGen}(\lambda) \to (pk, sk)$.

The following definition provides a formal security notion of PKE, which considers the indistinguishability of plaintexts in ciphertexts against adaptive chosen ciphertext attacks.

Definition 2 (IND-CCA2 Security for PKE)**.** *A PKE scheme* $\mathsf{PKE} = (\mathsf{PKE.KeyGen}, \mathsf{PKE.Enc}, \mathsf{PKE.Dec})$ *is IND-CCA2 secure if for any probabilistic polynomial-time (PPT) adversary* $\mathcal{A}$*, the advantage of* $\mathcal{A}$ *in the following game with the challenger* $\mathcal{C}$ *is negligible in the security parameter* λ*:*

- **Setup:** $\mathcal{C}$ *runs* $\mathsf{PKE.KeyGen}(\lambda) \to (pk, sk)$ *and passes the public key* pk *to* $\mathcal{A}$*.*
- **Phase 1:** $\mathcal{A}$ *may request queries to a decryption oracle that takes a ciphertext* C *and returns the output of* $\mathsf{PKE.Dec}(sk, C)$*, polynomially many times adaptively.*
- **Challenge:** $\mathcal{A}$ *selects two messages* M_0, M_1 *of the same length and forwards them to* $\mathcal{C}$*. Then,* $\mathcal{C}$ *chooses a random bit* $b \in \{0, 1\}$*, runs* $\mathsf{PKE.Enc}(pk, M_b) \to C_b^*$*, and passes* C_b^* *to* $\mathcal{A}$*.*
- **Phase 2:** *For* $\mathcal{A}$*'s queries,* $\mathcal{C}$ *responds as in* **Phase 1***. The constraint for* $\mathcal{A}$*'s queries that* C_b^* *cannot be queried.*
- **Guess:** $\mathcal{A}$ *outputs* $b' \in \{0, 1\}$*.*

We say that $\mathcal{A}$ *wins in the above game if* $b = b'$ *and the advantage of* $\mathcal{A}$ *is defined to*

$$\mathbf{Adv}^{\text{IND-CCA2}}_{\mathcal{A},\mathsf{PKE}}(\lambda) := \left| \Pr[b = b'] - \frac{1}{2} \right|.$$

2.2. Definitions of Public Key Encryption with Equality Test

In this subsection, we introduce the system model for PKEET that we will consider in this paper and look into basic definitions for our PKEET. We note that our system model and security model for PKEET in this paper follow Tang's all-or-nothing PKEET models [33].

System Model for Our PKEET. A PKEET system that we consider throughout this paper consists of users (e.g., senders and receivers) and testers. In the system, a sender encrypts a data using a receiver's public key and passes the receiver a ciphertext. The receiver may decrypt his ciphertexts using his secret key and store ciphertexts at the server. Once a need arises, the receiver issues a trapdoor for equality tests to a tester who can access to the server that stores encrypted data of the receiver. Since then, the tester can perform equality tests on ciphertexts under the public key of the receiver who delegated the test authority for all of his ciphertexts to the tester.

Definitions of PKEET. Since a PKEET system is a multi-user setting, we begin by introducing some notations for convenience. We assume that each user is assigned an index i for $1 \leq i \leq N$ where N is the number of users in the system. U_i denotes the user whose index is i. We use a subscripted index to denote keys, ciphertexts and trapdoors for each user, e.g., pk_i and CT_i are a public key and a ciphertext of user U_i, respectively.

Definition 3 (Public Key Encryption with Equality Test)**.** *A public key encryption with equality test (PKEET) consists of the following six polynomial-time algorithms* $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Td}, \mathsf{Test})$*.*

- $\mathsf{Setup}(\lambda)$*: It takes a security parameter* λ *as an input and returns a system parameter* params *that includes the information of the plaintext space* $\mathcal{P}$*. We assume that all other algorithms take* params *as an input implicitly, though it is not stated.*

- $\mathsf{KeyGen}(\mathsf{params})$*: Given the system parameter* params*, it returns a pair of user's public and secret keys* $(\mathsf{pk}, \mathsf{sk})$*.*
- $\mathsf{Enc}(\mathsf{pk}, M)$*: Given the public key* pk *and a message* $M \in \mathcal{P}$*, it returns a ciphertext* CT*.*
- $\mathsf{Dec}(\mathsf{sk}, CT)$*: Given the secret key* sk *and a ciphertext* CT*, it returns a message* M' *or* $\perp$*.*
- $\mathsf{Td}(\mathsf{sk}_i)$*: It takes a secret key* sk_i *of user* U_i *and returns a trapdoor* td_i *for user* U_i*.*
- $\mathsf{Test}(\mathsf{td}_i, CT_i, \mathsf{td}_j, CT_j)$*: It takes two ciphertexts* CT_i*,* CT_j *and two trapdoors* td_i*,* td_j *as inputs, and returns* 1 *or* 0 *where* 1 *indicates that* CT_i *and* CT_j *contain the same message and* 0 *indicates that they contain different messages.*

Correctness of PKEET. A PKEET scheme is *correct* if the following three conditions hold: For any $1 \leq i, j \leq N$ and any $M_i, M_j \in \mathcal{P}$,

1. $\Pr[\mathsf{Dec}(\mathsf{sk}_i, CT_i) \rightarrow M_i] = 1$;
2. $\Pr[\mathsf{Test}(\mathsf{td}_i, CT_i, \mathsf{td}_j, CT_j) \rightarrow 1] = 1$ if $\mathsf{Dec}(\mathsf{sk}_i, CT_i) = \mathsf{Dec}(\mathsf{sk}_j, CT_j) \neq \perp$;
3. $\Pr[\mathsf{Test}(\mathsf{td}_i, CT_i, \mathsf{td}_j, CT_j) \rightarrow 1]$ is negligible in the security parameter λ if $\mathsf{Dec}(\mathsf{sk}_i, CT_i) \neq \mathsf{Dec}(\mathsf{sk}_j, CT_j)$

where $\mathsf{Setup}(\lambda) \rightarrow \mathsf{params}$, $\mathsf{KeyGen}(\mathsf{params}) \rightarrow (\mathsf{pk}_i, \mathsf{sk}_i)$, $\mathsf{KeyGen}(\mathsf{params}) \rightarrow (\mathsf{pk}_j, \mathsf{sk}_j)$, $\mathsf{Enc}(\mathsf{pk}_i, M_i) \rightarrow CT_i$, $\mathsf{Enc}(\mathsf{pk}_j, M_j) \rightarrow CT_j$, $\mathsf{Td}(sk_i) \rightarrow \mathsf{td}_i$ and $\mathsf{Td}(sk_j) \rightarrow \mathsf{td}_j$.

We remark that the first condition is related to the correctness of Enc and Dec algorithms since a PKEET scheme is a kind of PKE schemes. On the other hand, the second and third conditions are related to the functionality of Td and Test algorithms.

Security Model for PKEET. Since an adversary may have a trapdoor for equality tests in some scenarios, we consider the following two types of adversaries according to whether the adversary has a trapdoor for the target user or not.

- Type-I adversary: We assume that this type of adversaries has the trapdoor for all of target user's ciphertexts. This adversary can perform an equality test with the challenge ciphertext and so we cannot expect to achieve the indistinguishability-based security notion against him. Instead, we consider the one-wayness security notion, which is probably the best achievable security, and we regard that the goal of this adversary is to reveal the message contained in the challenge ciphertext.
- Type-II adversary: We assume that this type of adversaries does not have the trapdoor for all of target user's ciphertexts. This adversary is almost the same as that of traditional PKE schemes and we can expect to achieve the indistinguishability-based security notion. Hence, we regard that the goal of this adversary is to distinguish whether the challenge ciphertext contains which message between two candidates.

We first consider the formal security definition for PKEET constructions against Type-I adversaries. By considering that a Type-I adversary can have the trapdoor for all of target user's ciphertext, a Type-I adversary allows to execute the followings at any time and in any order:

- He/She can request decryption oracle queries on all ciphertexts, except for the challenge ciphertext.
- He/She can request secret key generation queries on all users, except for the target user.
- He/She can request trapdoor generation queries on all users. (Again, remark that he/she can have the trapdoor for all of target user's ciphertexts.)

Below is the formal security definition for PKEET schemes against Type-II adversaries.

Definition 4 (OW-CCA2 against Type-I Adversaries)**.** *A PKEET scheme is OW-CCA2 secure against Type-I adversaries if for any PPT adversary $\mathcal{A}$, the success probability of $\mathcal{A}$ in the following game with the challenger $\mathcal{C}$ is negligible in the security parameter λ: Let U_t be the target user in the game.*

1. **Setup:** *$\mathcal{C}$ takes a security parameter λ as an input, runs $\mathsf{Setup}(\lambda) \to \mathsf{params}$ and passes the system parameter params to $\mathcal{A}$. Then, $\mathcal{C}$ runs $\mathsf{KeyGen}(\mathsf{params}) \to (\mathsf{pk}_i, \mathsf{sk}_i)$ for $1 \leq i \leq N$ and sends all pk_i's to $\mathcal{A}$.*
2. **Phase 1:** *$\mathcal{A}$ may request queries to the following oracles polynomially many times adaptively and in any order. The constraint is that an index t cannot be queried to the key extraction oracle $\mathcal{O}^{\mathsf{sk}}$.*
   - *$\mathcal{O}^{\mathsf{sk}}$: an oracle that on input an index i, returns the U_i's secret key sk_i.*
   - *$\mathcal{O}^{\mathsf{Dec}}$: an oracle that on input an index i and a ciphertext CT_i, returns the outcome of $\mathsf{Dec}(\mathsf{sk}_i, CT_i)$ using the U_i's secret key sk_i.*
   - *$\mathcal{O}^{\mathsf{Td}}$: an oracle that on input an index i, returns td_i by performing $\mathsf{Td}(\mathsf{sk}_i) \to \mathsf{td}_i$ with the U_i's secret key sk_i.*
3. **Challenge:** *$\mathcal{C}$ selects a random message M from the plaintext space $\mathcal{P}$, runs $\mathsf{Enc}(\mathsf{pk}_t, M) \to CT_t^*$, and sends CT_t^* to $\mathcal{A}$.*
4. **Phase 2:** *For $\mathcal{A}$'s queries, $\mathcal{C}$ responds as in* **Phase 1**. *The constraints for $\mathcal{A}$'s queries are that*
   (a) the index t cannot be queried to the key extraction oracle $\mathcal{O}^{\mathsf{sk}}$;
   (b) the pair of the index t and the ciphertext CT_t^* cannot be queried to the decryption oracle $\mathcal{O}^{\mathsf{Dec}}$.
5. **Guess:** *$\mathcal{A}$ outputs M'.*

We say that $\mathcal{A}$ wins in the above game if $M = M'$ and the success probability of $\mathcal{A}$ is defined to

$$\mathbf{Adv}^{\text{OW-CCA2}}_{\mathcal{A},\text{PKEET}}(\lambda) := \Pr[M = M'].$$

We give a remark about the constraint on the plaintext space for the security of PKEET schemes. Because a Type-I adversary has a trapdoor for the target user, he can recover the plaintext contained in the challenge ciphertext by performing equality tests between the challenge ciphertext and ciphertexts for all plaintexts generated by himself, if the size of the plaintext space is polynomial in the security parameter or the min-entropy of the plaintext distribution is lower. Therefore, we assume that the size of the plaintext space is exponential in the security parameter and the min-entropy of the plaintext distribution is sufficiently larger than the security parameter.

Next, we consider the formal security definition for PKEET schemes against Type-II adversaries. Since we assume that a Type-II adversary cannot have the trapdoor for all of target user's ciphertexts, his/her ability is almost the same as that of an adversary in the IND-CCA2 security game for traditional PKE schemes, except that some additional information of non-target users are given because a PKEET scheme is formulated in a multi-user setting. Concretely, a Type-II adversary allows to execute the followings at any time and in any order:

- He/She can request decryption oracle queries on all ciphertexts, except for the challenge ciphertext.
- He/She can request secret key generation queries on all users, except for the target user.
- He/She can request trapdoor generation queries on all users, except for the target user. (This is a main difference between the abilities of Type-I and Type-II adversaries.)

The formal security definition for PKEET schemes against Type-II adversaries is given below.

Definition 5 (IND-CCA2 against Type-II Adversaries)**.** *A PKEET scheme is IND-CCA2 secure against Type-II adversaries if for any PPT adversary $\mathcal{A}$, the advantage of $\mathcal{A}$ in the following game with the challenger $\mathcal{C}$ is negligible in the security parameter λ: Let U_t be the target user in the game.*

1. **Setup:** *This step is exactly the same as that of the OW-CCA2 security game in Definition 4.*
2. **Phase 1:** *This step is almost the same as that of the OW-CCA2 security game in Definition 4, except that the constraint is that an index t cannot be queried to the trapdoor extraction oracle $\mathcal{O}^{\mathsf{Td}}$ as well as the key extraction oracle $\mathcal{O}^{\mathsf{sk}}$.*
3. **Challenge:** *$\mathcal{A}$ selects two message $M_0, M_1 \in \mathcal{M}$ of the same length and sends them to $\mathcal{C}$. Then, $\mathcal{C}$ selects a random bit $b \in \{0,1\}$, runs $\mathsf{Enc}(\mathsf{pk}_t, M_b) \to CT^*_{t,b}$, and passes $CT^*_{t,b}$ to $\mathcal{A}$.*
4. **Phase 2:** *For $\mathcal{A}$'s queries, $\mathcal{C}$ responds as in* **Phase 1**. *The constraints for $\mathcal{A}$'s queries are that*
   (a) *the index t cannot be queried to the key extraction oracle $\mathcal{O}^{\mathsf{sk}}$ and the trapdoor extraction oracle $\mathcal{O}^{\mathsf{Td}}$;*
   (b) *the pair of the index t and the ciphertext $CT^*_{t,b}$ cannot be queried to the decryption oracle $\mathcal{O}^{\mathsf{Dec}}$.*
5. **Guess:** *$\mathcal{A}$ outputs b'.*

We say that $\mathcal{A}$ wins in the above game if $b = b'$ and the advantage of $\mathcal{A}$ is defined to

$$\mathbf{Adv}^{\text{IND-CCA2}}_{\mathcal{A},\text{PKEET}}(\lambda) := \left| \Pr[b = b'] - \frac{1}{2} \right|.$$

2.3. Cryptographic Hash Functions and Random Oracle Models

Now, we introduce some properties of cryptographic hash functions, which will be utilized in this paper, and random oracle models.

Properties of Hash Functions. We look at the requirements for cryptographic hash functions that will be exploited in our PKEET construction. First, we provide the definition of a one-way function.

Definition 6 (One-Way Function)**.** *We say that a function $\mathcal{H} : \{0,1\}^* \to Y$ is one-way if the following conditions hold:*

- *There exists a polynomial-time algorithm to compute $\mathcal{H}$.*
- *For any PPT adversary $\mathcal{A}$, the success probability of $\mathcal{A}$ in the following game with the challenger $\mathcal{C}$ is negligible in the security parameter λ:*
  1. *For a given security parameter λ, $\mathcal{C}$ selects x from $\{0,1\}^\lambda$, computes $y = f(x)$, and sends y to $\mathcal{A}$.*
  2. *$\mathcal{A}$ outputs x'.*

  The success probability of $\mathcal{A}$ in the above game is defined to $\Pr[y = f(x')]$.

Next, we define a hash function and provide the definition of a collision-resistant hash function below.

Definition 7 (Hash Function)**.** *A hash function is a pair of PPT algorithms $(\mathsf{Gen}, \mathcal{H})$ where Gen is an algorithm which takes a security parameter λ as an input and returns a key s, and there exists a polynomial $\ell(\cdot)$ such that $\mathcal{H}$ takes a key s and a string $x \in \{0,1\}^*$ as inputs and returns a string $\mathcal{H}^s(x) := \mathcal{H}(s,x) \in \{0,1\}^{\ell(\lambda)}$.*

Definition 8 (Collision-Resistant Hash Function)**.** *We say that a hash function $(\mathsf{Gen}, \mathcal{H})$ is collision-resistant if for any PPT adversary $\mathcal{A}$, the success probability of $\mathcal{A}$ in the following game with the challenger $\mathcal{C}$ is negligible in the security parameter λ:*

1. *$\mathcal{C}$ runs $\mathsf{Gen}(\lambda) \to s$ and passes s to $\mathcal{A}$.*
2. *$\mathcal{A}$ returns x and x'.*

The success probability of $\mathcal{A}$ in the above game is defined to $\Pr[\mathcal{H}(s,x) = \mathcal{H}(s,x')]$.

Throughout the paper, we simply use $\mathcal{H}$ to represent a hash function $(\mathsf{Gen}, \mathcal{H})$ unless confusion arises.

Random Oracle Models. We will prove the security of our proposed construction in the random oracle model. For the security analysis in the random oracle model, a cryptographic hash function employed in a scheme is replaced by a random oracle that is a function which returns a truly random element from the range of that hash function. Thus, the challenger should additionally provide oracles for exploited hash functions to the adversary at Phases 1 and 2 for the security games of PKEET schemes, described in Definitions 4 and 5. See Section 4 for details.

3. Our Generic Construction

In this section, we present our generic construction for PKEET that uses a PKE scheme with special property, called *randomness extractability*. Informally, PKE with randomness extractability allows one who has the secret key to extract the randomness used in any ciphertexts encrypted by the corresponding public key. The formal definition is given below. We note that randomness extractability is satisfied in well-known strategies for obtaining IND-CCA2 secure PKE schemes; e.g., the FO transformation [16]. In this sense, it is not an onerous requirement. We will discuss an example of IND-CCA2 secure PKE schemes with randomness extractability in Section 5.1.

Definition 9 (Randomness Extractability)**.** *Given a PKE scheme* $\mathsf{PKE} = (\mathsf{PKE.KeyGen}, \mathsf{PKE.Enc}, \mathsf{PKE.Dec})$*, we say that* PKE *is randomness extractable if there exists a PPT algorithm* Ext *that takes a secret key* sk *and a ciphertext* C *as inputs and returns an element in* $\mathcal{R}$ *with the following property, where* $\mathcal{R}$ *is the domain for randomness used in the* $\mathsf{PKE.Enc}$ *algorithm: For any security parameter* λ*, any message* $M \in \mathcal{P}$ *and any randomness* $\sigma \xleftarrow{\$} \mathcal{R}$*,*

$$\Pr[\mathsf{Ext}(sk, C) \to \sigma | \mathsf{PKE.Enc}(pk, M; \sigma) \to C] = 1$$

where $\mathsf{PKE.KeyGen}(\lambda) \to (pk, sk)$*.*

Hereafter, we assume that all PKE schemes we consider have randomness extractability.

Design Rationale of Our PKEET Scheme. Similarly to the previous semi-generic or generic constructions for PKEET [21, 22], we design an encryption algorithm of our PKEET scheme so that it first generates two ciphertexts of a message M and its hash value $\mathcal{H}_1(M)$, respectively, i.e.,

$$\mathsf{PKE.Enc}(pk_1, M; \sigma_1) \to C_1 \text{ and } \mathsf{PKE.Enc}(pk_2, \mathcal{H}_1(M); \sigma_2) \to C_2$$

by exploiting an underlying IND-CCA2 secure PKE scheme $\mathsf{PKE} = (\mathsf{PKE.KeyGen}, \mathsf{PKE.Enc}, \mathsf{PKE.Dec})$ where $\mathsf{PKE.KeyGen}(\lambda) \to (pk_1, sk_1)$, $\mathsf{PKE.KeyGen}(\lambda) \to (pk_2, sk_2)$ and $\mathcal{H}_1$ is a cryptographic hash function. While the first component C_1 of a ciphertext is utilized to recover an original message M by a receiver, the second component C_2 of a ciphertext is utilized to perform equality tests by issuing the secret key sk_2 to a tester as a trapdoor and then letting her compare hash values after decrypting the second components of ciphertexts.

Since we assume that the exploited PKE scheme is IND-CCA2 secure, each ciphertext may not leak any partial information of its message to the adversary. However, when a ciphertext $CT = (C_1, C_2)$ is given, the adversary can generate another ciphertext $CT' = (C_1', C_2)$ or $CT'' = (C_1, C_2')$ by running $\mathsf{PKE.Enc}(pk_1, M'; \sigma_1') \to C_1'$ or $\mathsf{PKE.Enc}(pk_2, \mathcal{H}_1(M'); \sigma_2') \to C_2'$ for some message $M' \in \mathcal{P}$ and randomness $\sigma_1', \sigma_2' \in \mathcal{R}$. Thereafter, the adversary can query on CT' or CT'' to the decryption oracle and it may return M if $M = M'$ or $\perp$ if $M \neq M'$. Therefore, we cannot expect the IND-CCA2 security of the above provisional construction.

To prevent the above attack, we modify an encryption algorithm so that it additionally returns a hash value of two ciphertexts C_1, C_2 and randomness σ_1, σ_2 used in ciphertexts, i.e., $C_3 = \mathcal{H}_2(C_1, C_2, \sigma_1, \sigma_2)$ for a cryptographic hash function $\mathcal{H}_2$. Since the adversary does not know σ_1 and σ_2 used to generate C_1 and C_2, respectively, he cannot compute an appropriate value C_3 for another ciphertext which involves C_1 or C_2. On

Table 1: Symbols used in Our PKEET Scheme

Symbol	Description
PKE	a utilized public key encryption scheme
$\mathsf{PKE.KeyGen}$	a key generation algorithm of PKE
$\mathsf{PKE.Enc}$	an encryption algorithm of PKE
$\mathsf{PKE.Dec}$	a decryption algorithm of PKE
$\mathcal{P}$	a plaintext space of PKE
$\mathcal{R}$	a domain of randomess for $\mathsf{PKE.Enc}$
Ext	an extraction algorithm for PKE
λ	a security parameter
params	a system parameter
$\mathcal{H}_1, \mathcal{H}_2$	hash functions
$\mathsf{pk}_i = (pk_{i,1}, pk_{i,2})$	the i-th user's public key consisting of two public keys $pk_{i,1}$ and $pk_{i,2}$ for PKE
$\mathsf{sk}_i = (sk_{i,1}, sk_{i,2})$	the i-th user's secret key consisting of two secret keys $sk_{i,1}$ and $sk_{i,2}$ for PKE
td_i	the trapdoor for the i-th user
$CT_i = (C_{i,1}, C_{i,2}, C_{i,3})$	the i-th user's ciphertext consisting of three components $C_{i,1}, C_{i,2}$ and $C_{i,3}$

the other hand, a decryption algorithm can extract the randomness because it is assumed that the exploited PKE scheme is randomness extractable. Hence, it can check the validity of the value C_3 in any ciphertexts.

Description of Our PKEET Scheme. We provide a description of our PKEET construction below.

- $\mathsf{Setup}(\lambda)$: It takes a security parameter λ as an input. Then, it generates

  1. a PKE scheme $\mathsf{PKE} = (\mathsf{PKE.KeyGen}, \mathsf{PKE.Enc}, \mathsf{PKE.Dec})$, where its plaintext space and domain of randomness for $\mathsf{PKE.Enc}$ are $\mathcal{P}$ and $\mathcal{R}$, respectively, and
  2. hash functions $\mathcal{H}_1 : \mathcal{P} \to \mathcal{P}$ and $\mathcal{H}_2 : \{0,1\}^* \to \{0,1\}^\ell$ for a parameter $\ell = \ell(\lambda)$.

  It outputs a system parameter $\mathsf{params} = (\lambda, \mathsf{PKE}, \mathcal{H}_1, \mathcal{H}_2)$.

- $\mathsf{KeyGen}(\mathsf{params})$: On input the system parameter params, it runs $\mathsf{PKE.KeyGen}(\lambda) \to (pk_1, sk_1)$ and $\mathsf{PKE.KeyGen}(\lambda) \to (pk_2, sk_2)$, and outputs a public key $\mathsf{pk} = (pk_1, pk_2)$ and a secret key $\mathsf{sk} = (sk_1, sk_2)$.

- $\mathsf{Enc}(\mathsf{pk}, M)$: It takes the public key pk and a message $M \in \mathcal{P}$ as inputs and parses pk as (pk_1, pk_2). Then, it performs as follows:

  1. Select randomness $\sigma_1, \sigma_2 \xleftarrow{\$} \mathcal{R}$.
  2. Run $\mathsf{PKE.Enc}(pk_1, M; \sigma_1) \to C_1$ and $\mathsf{PKE.Enc}(pk_2, \mathcal{H}_1(M); \sigma_2) \to C_2$.
  3. Compute $C_3 = \mathcal{H}_2(C_1, C_2, \sigma_1, \sigma_2)$.
  4. Ouptut $CT = (C_1, C_2, C_3)$.

- $\mathsf{Dec}(\mathsf{sk}, CT)$: On input the secret key sk and a ciphertext CT, parse sk and CT as (sk_1, sk_2) and (C_1, C_2, C_3), respectively. Then, it performs as follows:

  1. Extract messages and randomness from C_1 and C_2 by using sk_1 and sk_2; that is, run $\mathsf{PKE.Dec}(sk_1, C_1) \to M'$, $\mathsf{Ext}(sk_1, C_1) \to \sigma_1'$, $\mathsf{PKE.Dec}(sk_2, C_2) \to h'$ and $\mathsf{Ext}(sk_2, C_2) \to \sigma_2'$.
  2. Check whether $h' = \mathcal{H}_1(M')$ and $C_3 = \mathcal{H}_2(C_1, C_2, \sigma_1', \sigma_2')$.
  3. If both hold, output M'. Otherwise, output $\perp$.

- $\mathsf{Td}(\mathsf{sk}_i)$: On input a user U_i's secret key sk_i, parse it as $(sk_{i,1}, sk_{i,2})$ and output $\mathsf{td}_i = sk_{i,2}$.
- $\mathsf{Test}(\mathsf{td}_i, CT_i, \mathsf{td}_j, CT_j)$: It takes trapdoors td_i, td_j and ciphertexts CT_i, CT_j for users U_i and U_j, respectively, as inputs. For $k = i, j$,
  1. parse CT_k as $(C_{k,1}, C_{k,2}, C_{k,3})$ and
  2. run $\mathsf{PKE.Dec}(\mathsf{td}_k, C_{k,2}) \to h'_k$.

  If $h'_i = h'_j$, then output 1. Otherwise, output 0.

Correctness. The following theorem demonstrates the correctness of our proposed construction.

Theorem 1. *Our PKEET scheme is correct if the exploited PKE scheme* PKE *is correct and randomness extractable, and the utilized hash function $\mathcal{H}_1$ is collision-resistant.*

Proof. The correctness of the decryption algorithm Dec directly comes from the correctness of the decryption algorithm PKE.Dec and the extraction algorithm Ext of the PKE scheme PKE. Further, we easily obtain the correctness of our trapdoor and test algorithms from the correctness of the PKE.Dec algorithm and the collision-resistant property of the utilized hash function $\mathcal{H}_1$.

More precisely, let $CT_k = (C_{k,1}, C_{k,2}, C_{k,3})$ be a valid ciphertext of a message $M_k \in \mathcal{P}$ for any $1 \leq k \leq N$, i.e.,

$$\begin{aligned}&\mathsf{PKE.Enc}(pk_{k,1}, M_k; \sigma_{k,1}) \to C_{k,1},\\ &\mathsf{PKE.Enc}(pk_{k,2}, \mathcal{H}_1(M_k); \sigma_{k,2}) \to C_{k,2}, \text{ and}\\ &C_{k,3} = \mathcal{H}_2(C_{k,1}, C_{k,2}, \sigma_{k,1}, \sigma_{k,2}),\end{aligned}$$

where $\mathsf{Setup}(\lambda) \to \mathsf{params} = (\lambda, \mathsf{PKE}, \mathcal{H}_1, \mathcal{H}_2)$ and $\mathsf{KeyGen}(\lambda) \to (\mathsf{pk}_k, \mathsf{sk}_k)$ with $\mathsf{pk}_k = (pk_{k,1}, pk_{k,2})$, $\mathsf{sk}_k = (sk_{k,1}, sk_{k,2})$. Then, we can easily check that the following conditions hold:

1. Since PKE is correct and randomness extractable, $\mathsf{PKE.Dec}(sk_{k,1}, C_{k,1}) \to M'_k = M_k$, $\mathsf{PKE.Dec}(sk_{k,2}, C_{k,2}) \to h'_k = \mathcal{H}_1(M_k)$, $\mathsf{Ext}(sk_{k,1}, C_{k,1}) \to \sigma'_{k,1} = \sigma_{k,1}$, and $\mathsf{Ext}(sk_{k,2}, C_{k,2}) \to \sigma'_{k,2} = \sigma_{k,2}$. Hence, both $h'_k = \mathcal{H}_1(M'_k)$ and $C_{k,3} = \mathcal{H}_2(C_{k,1}, C_{k,2}, \sigma'_{k,1}, \sigma'_{k,2})$ hold and the Dec algorithm on inputs sk_k and CT_k always outputs M_k.
2. Since $\mathsf{Td}(\mathsf{sk}_k) \to \mathsf{td}_k = sk_{k,2}$, $\mathsf{PKE.Dec}(\mathsf{td}_k, C_{k,2}) \to h'_k = \mathcal{H}_1(M_k)$. Hence, for any $1 \leq i, j \leq N$, $h'_i = \mathcal{H}_1(M_i) = \mathcal{H}_1(M_j) = h'_j$ always holds if $M_i = M_j$. Therefore, the Test algorithm outputs 1 with probability 1 for this case.
3. On the other hand, in the above, if $M_i \neq M_j$, then $h'_i = \mathcal{H}_1(M_i) = \mathcal{H}_1(M_j) = h'_j$ holds with negligible probability since $\mathcal{H}_1$ is collision-resistant. Hence, the Test algorithm outputs 1 with negligible probability for this case.

From the above, our PKEET construction is correct. □

4. Security Analysis

In this section, we provide two theorems proving the OW-CCA2 security against Type-I adversaries and the IND-CCA2 security against Type-II adversaries, respectively.

4.1. IND-CCA2 Security against Type-II Adversaries

We first consider the IND-CCA2 security of our PKEET construction against Type-II adversaries.

Theorem 2 (IND-CCA2)**.** *If* PKE *is IND-CCA2 secure and randomness extractable, then the proposed PKEET scheme in Section 3 exploiting* PKE *is IND-CCA2 secure against Type-II adversaries in the random oracle model.*

Proof. We prove this theorem by using the standard hybrid arguments. First, we define a series of games between the adversary $\mathcal{A}$ and the challenger $\mathcal{C}$ as follows. Let N be the number of users in the PKEET system and denote the challenge ciphertext of the target user U_t by $CT_t^* = (C_{t,1}^*, C_{t,2}^*, C_{t,3}^*)$.

- $\mathbf{Game}_0$ is the original game described in Definition 5.
- $\mathbf{Game}_1$ is basically equivalent to $\mathbf{Game}_0$, except the following case: If $\mathcal{A}$ requests decryption queries on ciphertexts of form $CT_t' = (C_{t,1}^*, C_{t,2}', C_{t,3}')$, then $\mathcal{C}$ returns $\perp$.
- $\mathbf{Game}_2$ is almost the same as $\mathbf{Game}_1$, except the following case: If $\mathcal{A}$ requests decryption queries on ciphertexts of form $CT_t' = (C_{t,1}', C_{t,2}^*, C_{t,3}')$, then $\mathcal{C}$ returns $\perp$.
- In $\mathbf{Game}_3$, $\mathcal{C}$ behaves almost the same as that in $\mathbf{Game}_2$, except for generating the challenge ciphertext. $\mathcal{C}$ first tosses unbiased coins α and b. If $\alpha = 0$, then $\mathcal{C}$ computes

  $$\mathsf{PKE.Enc}(pk_{t,1}, M_b; \sigma_{t,1}^*) \to C_{t,1}^* \text{ and } \mathsf{PKE.Enc}(pk_{t,2}, \mathcal{H}_1(M_{1-b}); \sigma_{t,2}^*) \to C_{t,2}^*.$$

  Otherwise (i.e., $\alpha = 1$), it computes

  $$\mathsf{PKE.Enc}(pk_{t,1}, M_{1-b}; \sigma_{t,1}^*) \to C_{t,1}^* \text{ and } \mathsf{PKE.Enc}(pk_{t,2}, \mathcal{H}_1(M_b); \sigma_{t,2}^*) \to C_{t,2}^*.$$

  For both cases, $\mathcal{H}_2(C_{t,1}^*, C_{t,2}^*, \sigma_{t,1}^*, \sigma_{t,2}^*) = C_{t,3}^*$.

Lemma 1. *If* PKE *is IND-CCA2 secure and randomness extractable, and* $\mathcal{H}_2$ *is modeled as a random oracle, then any PPT adversary* $\mathcal{A}$ *has a negligible difference between its advantages in* $\mathbf{Game}_0$ *and* $\mathbf{Game}_1$*.*

Proof of Lemma 1. Suppose that there exists a PPT adversary $\mathcal{A}$ who has a non-negligible difference between its advantages in $\mathbf{Game}_0$ and $\mathbf{Game}_1$. We construct a simulator $\mathcal{S}$ that breaks the IND-CCA2 security of PKE using $\mathcal{A}$ as a subroutine.

We first describe $\mathcal{S}$'s behaviours. It starts with sending the system parameter $(\lambda, \mathsf{PKE}, \mathcal{H}_1)$ to $\mathcal{A}$. Once receiving a public key pk_1 from the IND-CCA2 game challenger $\mathcal{C}_{\mathsf{PKE}}$ for PKE, $\mathcal{S}$ runs $\mathsf{PKE.KeyGen}(\lambda) \to (pk_{i,1}, sk_{i,1})$ for $1 \le i \ne t \le N$, $\mathsf{PKE.KeyGen}(\lambda) \to (pk_{i,2}, sk_{i,2})$ for $1 \le i \le N$ by itself and sets $pk_{t,1} = pk_1$. $\mathcal{S}$ passes all $\mathsf{pk}_i = (pk_{i,1}, pk_{i,2})$'s to $\mathcal{A}$. In the challenge phase, once $\mathcal{S}$ receives two messages M_0, M_1 from $\mathcal{A}$, it transfers them to $\mathcal{C}_{\mathsf{PKE}}$. $\mathcal{C}_{\mathsf{PKE}}$ may response with $C_{t,1,b}^*$ obtained by running $\mathsf{PKE}(pk_1, M_b; \sigma_{t,1}^*) \to C_{t,1,b}^*$ for a randomly chosen bit b by $\mathcal{C}_{\mathsf{PKE}}$. Then, $\mathcal{S}$ tosses a random coin β and runs $\mathsf{PKE.Enc}(pk_{t,2}, \mathcal{H}_1(M_\beta); \sigma_{t,2}^*) \to C_{t,2,\beta}^*$. It chooses a random value R from $\{0,1\}^\ell$ and passes $(C_{t,1}^*, C_{t,2}^*, C_{t,3}^*) = (C_{t,1,b}^*, C_{t,2,\beta}^*, R)$ to $\mathcal{A}$ as a challenge ciphertext.

Next, we explain how $\mathcal{S}$ handles $\mathcal{A}$'s queries.

- $\mathcal{O}^{\mathsf{sk}}$ queries: On input $i \ne t$, it returns sk_i.
- $\mathcal{O}^{\mathsf{Td}}$ queries: On input $i \ne t$, it returns $\mathsf{td}_i = sk_{i,2}$.
- $\mathcal{O}^{\mathcal{H}_2}$ queries: It generates a table $\mathsf{H2Table}$, which is initiated as an empty set. The table has five columns for input and output of $\mathcal{H}_2$ queries. On input a pair $(C_1, C_2, \sigma_1, \sigma_2)$, it first checks whether there exists the same input stored at the table $\mathsf{H2Table}$. If exists, it basically returns the stored output value in $\mathsf{H2Table}$. Specially, if $(C_1, C_2) = (C_{t,1}^*, C_{t,2}^*)$, then it additionally checks whether $(\sigma_1, \sigma_2) = (\sigma_{t,1}^*, \sigma_{t,2}^*)$ where $\sigma_{t,i}^*$ is the randomness used in $C_{t,i}^*$ for $i = 1, 2$. This can be done as follows. As for the σ_2, $\mathcal{S}$ can extract the randomness of C_2 by using $sk_{t,2}$. On the other hand, as for the σ_1, it checks whether C_1 is

equal to one of $\mathsf{PKE.Enc}(pk_{t,1}, M_0; \sigma_1)$ and $\mathsf{PKE.Enc}(pk_{t,1}, M_1; \sigma_1)$. If the equality $(\sigma_1, \sigma_2) = (\sigma^*_{t,1}, \sigma^*_{t,2})$ holds, $\mathcal{S}$ returns R.[1] For all the other cases, $\mathcal{S}$ chooses a random value $\mathsf{h2value}$ from $\{0,1\}^\ell$, returns it to $\mathcal{A}$, and stores it with the query $(C_1, C_2, \sigma_1, \sigma_2)$ into a new row of $\mathsf{H2Table}$.

- $\mathcal{O}^{\mathsf{Dec}}$ queries: $\mathcal{S}$'s responses vary with the form of inputs $CT_i = (C_{i,1}, C_{i,2}, C_{i,3})$. On input a pair of an index i and a ciphertext CT_i, $\mathcal{S}$ performs as follows.
  - If $i \neq t$, it runs $\mathsf{Dec}(\mathsf{sk}_i, CT_i) \to M'$ using the secret key sk_i and returns M'.
  - If $i = t$ and $C_{i,1} \neq C^*_{t,1}$, $\mathcal{S}$ decrypts $C_{i,1}$ by requesting a decryption query on $C_{i,1}$ to $\mathcal{C}_{\mathsf{PKE}}$ and runs $\mathsf{PKE.Dec}(sk_{t,2}, C_{i,2})$. If one of decryption results of $C_{i,1}$ and $C_{i,2}$ is $\perp$, $\mathcal{S}$ returns $\perp$. Otherwise, let M' be the decryption result of $C_{i,1}$ received from $\mathcal{C}_{\mathsf{PKE}}$ and h' be the decryption result of $C_{i,2}$. If $\mathcal{H}_1(M') \neq h'$, then $\mathcal{S}$ returns $\perp$. Otherwise, $\mathcal{S}$ searches the table $\mathsf{H2Table}$ to find all rows of the form $(C_{i,1}, C_{i,2}, \sigma_1, \sigma'_2, \mathsf{h2value})$ for some σ_1 and $\mathsf{h2value}$ where $\mathsf{Ext}(sk_{i,2}, C_{i,2}) \to \sigma'_2$. If there exist such the rows in $\mathsf{H2Table}$, $\mathcal{S}$ checks whether $C_{i,1}$ is equal to one of $\mathsf{PKE.Enc}(pk_{t,1}, M'; \sigma_1)$'s for all candidates of σ_1. If the equality holds, $\mathcal{S}$ returns M' to $\mathcal{A}$. Otherwise, it returns $\perp$.
  - If $i = t$ and $C_{i,1} = C^*_{t,1}$, $\mathcal{S}$ runs $\mathsf{PKE.Dec}(sk_{t,2}, C_{i,2})$ using the secret key $sk_{t,2}$. If the outcome is $\perp$, $\mathcal{S}$ also returns $\perp$. Otherwise, let h' be the decryption result of $C_{i,2}$ and σ'_2 be the outcome of the Ext algorithm of PKE. Then, $\mathcal{S}$ searches the table $\mathsf{H2Table}$ to find all rows of the form $(C_{i,1}, C_{i,2}, \sigma_1, \sigma'_2, \mathsf{h2value})$ for some σ_1 and $\mathsf{h2value}$. If there exist such the rows in $\mathsf{H2Table}$, $\mathcal{S}$ checks whether $C_{i,1}$ is equal to one of $\mathsf{PKE.Enc}(pk_{t,1}, M_0; \sigma_1)$ and $\mathsf{PKE.Enc}(pk_{t,1}, M_1; \sigma_1)$ for all candidates of σ_1. If there exists a case that the equality holds, then $\mathcal{S}$ stops all interactions with $\mathcal{A}$ and sends $\mathcal{C}_{\mathsf{PKE}}$ the corresponding bit b' satisfying $C_{i,1} = \mathsf{PKE.Enc}(pk_{t,1}, M_{b'}; \sigma_1)$. Otherwise, $\mathcal{S}$ returns $\perp$ to $\mathcal{A}$.

Finally, at the end of interaction, $\mathcal{S}$ returns a random bit b' to $\mathcal{C}_{\mathsf{PKE}}$, in particular, regardless of $\mathcal{A}$'s output.

Now, we analyze the advantage of $\mathcal{S}$. We first note that the challenger differs in two games $\mathbf{Game}_0$ and $\mathbf{Game}_1$ only when $\mathcal{A}$ asks a decryption query on a ciphertext of form $CT'_t = (C^*_{t,1}, C'_{t,2}, C'_{t,3})$ such that $CT'_t \neq CT^*_t$, where it is valid in the sense that its decryption result is not $\perp$. Therefore, by the hypothesis, we know that $\mathcal{A}$ issues such decryption queries with non-negligible probability since $\mathcal{A}$'s advantages in two games are exactly the same *without* such queries. Thus, we may assume that $\Pr[\mathbf{E} \text{ in } \mathbf{Game}_0] = \varepsilon$ is non-negligible where $\mathbf{E}$ denotes the event that $\mathcal{A}$ issues a decryption query on a valid ciphertext of form $CT'_t = (C^*_{t,1}, C'_{t,2}, C'_{t,3})$ such that $CT'_t \neq CT^*_t$.

To argue the advantage of $\mathcal{S}$ in the IND-CCA2 game for PKE, we first consider some important cases; (1) In the challenge phase, $\mathcal{S}$ tosses a random coin β, which is completely hidden from the adversarial viewpoint. If $\beta = b$ (this occurs with probability $1/2$ regardless of all the other previous events), then $\mathcal{S}$ correctly simulates the challenge ciphertext. Furthermore, if $\mathcal{S}$ does not stop during the simulation (in particular, in the process of answering decryption queries), the simulated transcripts are identical to those in $\mathbf{Game}_0$. (2) If $\beta \neq b$, then we cannot expect how $\mathcal{A}$ behaves. However, we can see that (3) if $\mathbf{E}$ occurs (regardless of whether $\beta = b$ or not), then the advantage of $\mathcal{S}$ is $1/2$ in the IND-CCA2 game since $\mathcal{S}$ can extract the message from $C^*_{t,1}$. (4) Furthermore, if $\mathbf{E}$ does not occur (again, regardless of whether $\beta = b$ or not), $\mathcal{S}$'s advantage should be 0 since in this case $\mathcal{S}$ does not stop during the simulation and outputs a random bit regardless of $\mathcal{A}$'s output at the end of the simulation.

From the above considerations, we have the following probability:

$$\begin{aligned}
\mathbf{Adv}^{\text{IND-CCA2}}_{\mathcal{S},\mathsf{PKE}}(\lambda) &= \left| \Pr[b = b'] - \frac{1}{2} \right| \\
&= \left| \Pr[\mathbf{E}] \cdot \Pr[b = b' | \mathbf{E}] + \Pr[\neg\mathbf{E}] \cdot \Pr[b = b' | \neg\mathbf{E}] - \frac{1}{2} \right|
\end{aligned}$$

[1] In fact, in this case, the simulator can directly break the IND-CCA2 security of PKE. However, for simplicity of the proof, we ignore this strategy in our proof.

$$
\begin{aligned}
&\overset{(3),(4)}{=} && \Big|\Pr[\mathbf{E}]\cdot 1+\Pr[\neg\mathbf{E}]\cdot\frac{1}{2}-\frac{1}{2}\Big| \\
&= && \frac{1}{2}\Pr[\mathbf{E}] \\
&= && \frac{1}{2}\cdot\big(\frac{1}{2}\Pr[\mathbf{E}|b=\beta]+\frac{1}{2}\cdot\Pr[\mathbf{E}|b\neq\beta]\big) \\
&\overset{(1)}{=} && \frac{1}{2}\big(\frac{1}{2}\cdot\varepsilon+\frac{1}{2}\Pr[\mathbf{E}|b\neq\beta]\big) \\
&\geq && \frac{\varepsilon}{4} \quad = \quad \text{non-negligible},
\end{aligned}
$$

where if the simulation is correct, $\mathcal{A}$ makes the event $\mathbf{E}$ with non-negligible probability ε in $\mathbf{Game}_0$. This contradicts with the IND-CCA2 security of PKE and thus the proof of Lemma 1 is completed. □

Lemma 2. *If* PKE *is IND-CCA2 secure and randomness extractable, and* $\mathcal{H}_2$ *is modeled as a random oracle, then any PPT adversary* $\mathcal{A}$ *has a negligible difference between its advantages in* $\mathbf{Game}_1$ *and* $\mathbf{Game}_2$*.*

Proof of Lemma 2. The proof of this lemma is almost the same as that of Lemma 1, except that the simulator $\mathcal{S}$ uses the second component of ciphertexts, not the first component. Let $\mathcal{A}$ be a PPT adversary who has a non-negligible difference between its advantages in $\mathbf{Game}_1$ and $\mathbf{Game}_2$. Then, we construct a simulator $\mathcal{S}$ that breaks the IND-CCA2 security of PKE using $\mathcal{A}$.

$\mathcal{S}$ begins by sending the system parameter $(\lambda, \mathsf{PKE}, \mathcal{H}_1)$ to $\mathcal{A}$. Once receiving a target public key pk_2 from the IND-CCA2 game challenger $\mathcal{C}_{\mathsf{PKE}}$ for PKE, $\mathcal{S}$ executes $\mathsf{PKE.KeyGen}(\lambda) \to (pk_{i,1}, sk_{i,1})$ for $1 \leq i \leq N$, $\mathsf{PKE.KeyGen}(\lambda) \to (pk_{i,2}, sk_{i,2})$ for $1 \leq i \neq t \leq N$ and sets $pk_{t,2} = pk_2$. $\mathcal{S}$ forwards all $\mathsf{pk}_i = (pk_{i,1}, pk_{i,2})$'s to $\mathcal{A}$. In the challenge phase, once $\mathcal{A}$ submits two messages M_0, M_1 to $\mathcal{S}$, $\mathcal{S}$ computes $\mathcal{H}_1(M_0), \mathcal{H}_1(M_1)$ and forwards them to $\mathcal{C}_{\mathsf{PKE}}$. $\mathcal{C}_{\mathsf{PKE}}$ may response with $C^*_{t,2,b}$ obtained by executing $\mathsf{PKE}(pk_2, \mathcal{H}_1(M_b); \sigma^*_{t,2}) \to C^*_{t,2,b}$ for a randomly chosen bit b by $\mathcal{C}_{\mathsf{PKE}}$. Then, $\mathcal{S}$ tosses a random coin $\beta \in \{0,1\}$, runs $\mathsf{PKE.Enc}(pk_{t,1}, M_\beta; \sigma^*_{t,1}) \to C^*_{t,1,\beta}$, selects a random value R from $\{0,1\}^\ell$, and passes $(C^*_{t,1}, C^*_{t,2}, C^*_{t,3}) = (C^*_{t,1,\beta}, C^*_{t,2,b}, R)$ to $\mathcal{A}$ as a challenge ciphertext.

$\mathcal{S}$'s responses to $\mathcal{A}$'s queries are also very similar to those in the proof of Lemma 1.

- $\mathcal{O}^{\mathsf{sk}}$ and $\mathcal{O}^{\mathsf{Td}}$ queries: These are exactly the same as those in the proof of Lemma 1.
- $\mathcal{O}^{\mathcal{H}_2}$ queries: These are almost the same as those in the proof of Lemma 1, except that when $(C_1, C_2) = (C^*_{t,1}, C^*_{t,2})$ for an input pair $(C_1, C_2, \sigma_1, \sigma_2)$, $\mathcal{S}$ extracts the randomness of C_1 by using $sk_{t,1}$, but checks whether C_2 is equal to one of $\mathsf{PKE.Enc}(pk_{t,2}, \mathcal{H}_1(M_0); \sigma_2)$ and $\mathsf{PKE.Enc}(pk_{t,2}, \mathcal{H}_1(M_1); \sigma_2)$, because $\mathcal{S}$ has the secret key $sk_{t,1}$ and does not have $sk_{t,2}$, differently from the proof of Lemma 1.
- $\mathcal{O}^{\mathsf{Dec}}$ queries: We note that $\mathcal{S}$ returns $\perp$ if $\mathcal{A}$ requests decryption queries on ciphertexts of form $CT'_t = (C^*_{t,1}, C'_{t,2}, C'_{t,3})$ by the definition of $\mathbf{Game}_1$. As in the proof of Lemma 1, $\mathcal{S}$'s responses vary with the form of inputs $CT_i = (C_{i,1}, C_{i,2}, C_{i,3})$ and they are almost the same as those in the proof of Lemma 1, except that $\mathcal{S}$ has the secret key $sk_{t,1}$, not $sk_{t,2}$. Concretely, $\mathcal{S}$ performs as follows.
  - If $i \neq t$, it runs $\mathsf{Dec}(\mathsf{sk}_i, CT_i) \to M'$ using the secret key sk_i and returns M'.
  - If $i = t$ and $C_{i,2} \neq C^*_{t,2}$, $\mathcal{S}$ decrypts $C_{i,2}$ by requesting a decryption query on $C_{i,2}$ to $\mathcal{C}_{\mathsf{PKE}}$ and runs $\mathsf{PKE.Dec}(sk_{t,1}, C_{i,1})$. If one of decryption results of $C_{i,1}$ and $C_{i,2}$ is $\perp$, $\mathcal{S}$ returns $\perp$. Otherwise, let M' be the decryption result of $C_{i,1}$ and h' be the decryption result of $C_{i,2}$ received from $\mathcal{C}_{\mathsf{PKE}}$. If $\mathcal{H}_1(M') \neq h'$, then $\mathcal{S}$ returns $\perp$. Otherwise, $\mathcal{S}$ searches the table $\mathsf{H2Table}$ to find all rows of the form $(C_{i,1}, C_{i,2}, \sigma'_1, \sigma_2, \mathsf{h2value})$ for some σ_2 and $\mathsf{h2value}$ where $\mathsf{Ext}(sk_{i,1}, C_{i,1}) \to \sigma'_1$. If there exist such the rows in $\mathsf{H2Table}$, $\mathcal{S}$ checks whether $C_{i,2}$ is equal to one of $\mathsf{PKE.Enc}(pk_{t,2}, h'; \sigma_2)$'s for all candidates of σ_2. If the equality holds, $\mathcal{S}$ returns M' to $\mathcal{A}$. Otherwise, it returns $\perp$.
  - If $i = t$ and $C_{i,2} = C^*_{t,2}$, $\mathcal{S}$ runs $\mathsf{PKE.Dec}(sk_{t,1}, C_{i,1})$ using the secret key $sk_{t,1}$. If the outcome is $\perp$, $\mathcal{S}$ also returns $\perp$. Otherwise, let M' be the decryption result of $C_{i,1}$ and σ'_1 be the outcome of the Ext algorithm of PKE. Then, $\mathcal{S}$ searches the table $\mathsf{H2Table}$ to find all rows of the form

$(C_{i,1}, C_{i,2}, \sigma'_1, \sigma_2, \mathsf{h2value})$ for some σ_2 and $\mathsf{h2value}$. If there exist such the rows in $\mathsf{H2Table}$, $\mathcal{S}$ checks whether $C_{i,2}$ is equal to one of $\mathsf{PKE.Enc}(pk_{t,2}, \mathcal{H}_1(M_0); \sigma_2)$ and $\mathsf{PKE.Enc}(pk_{t,2}, \mathcal{H}_1(M_1); \sigma_2)$ for all candidates of σ_2. If there exists a case that the equality holds, then $\mathcal{S}$ stops all interactions with $\mathcal{A}$ and sends $\mathcal{C}_{\mathsf{PKE}}$ the corresponding bit b' satisfying $C_{i,2} = \mathsf{PKE.Enc}(pk_{t,2}, \mathcal{H}_1(M_{b'}); \sigma_2)$. Otherwise, $\mathcal{S}$ returns $\perp$ to $\mathcal{A}$.

Finally, at the end of interaction, $\mathcal{S}$ returns a random bit b' to $\mathcal{C}_{\mathsf{PKE}}$, in particular, regardless of $\mathcal{A}$'s output.

Now, we analyze the advantage of $\mathcal{S}$. Denote by $\mathbf{E}$ the event that $\mathcal{A}$ issues a decryption query on a valid ciphertext of form $CT'_t = (C'_{t,1}, C^*_{t,2}, C'_{t,3})$ such that $CT'_t \neq CT^*_t$. Then, we may assume that $\Pr[\mathbf{E} \text{ in } \mathbf{Game}_1]$ is non-negligible and we obtain that $\mathbf{Adv}^{\text{IND-CCA2}}_{\mathcal{S},\mathsf{PKE}}(\lambda)$ is non-negligible from the same argument as in the proof of Lemma 1, except that $C^*_{t,1}$, $\mathbf{Game}_0$, and $\mathbf{Game}_1$ are replaced by $C^*_{t,2}$, $\mathbf{Game}_1$, and $\mathbf{Game}_2$, respectively. This contradicts with the IND-CCA2 security of PKE. Thus, $\mathcal{A}$ should have a negligible difference between its advantages in $\mathbf{Game}_1$ and $\mathbf{Game}_2$ for any PPT adversary $\mathcal{A}$ and the proof of Lemma 2 is completed. □

Lemma 3. *If* PKE *is IND-CCA2 secure and randomness extractable, and* $\mathcal{H}_2$ *is modeled as a random oracle, then any PPT adversary* $\mathcal{A}$ *has a negligible difference between its advantages in* $\mathbf{Game}_2$ *and* $\mathbf{Game}_3$.

Proof of Lemma 3. Suppose that there exists a PPT adversary $\mathcal{A}$ differing in two games $\mathbf{Game}_2$ and $\mathbf{Game}_3$ with non-negligible probability. More precisely, we assume that the difference between two advantages

$$\left| \Pr[\mathcal{A} \to b \text{ in } \mathbf{Game}_2] - \frac{1}{2} \right| = \varepsilon_2 \text{ and } \left| \Pr[\mathcal{A} \to b \text{ in } \mathbf{Game}_3] - \frac{1}{2} \right| = \varepsilon_3$$

is non-negligible. We construct a simulator $\mathcal{S}$ that breaks the IND-CCA2 security of PKE using $\mathcal{A}$. We first describe $\mathcal{S}$'s behaviors and then analyze the advantage of $\mathcal{S}$ in the IND-CCA2 security game for PKE. Denote by $\mathcal{C}_{\mathsf{PKE}}$ the challenger of the IND-CCA2 security game for PKE.

At the beginning of the simulation, $\mathcal{S}$ passes the system parameter $(\lambda, \mathsf{PKE}, \mathcal{H}_1)$ to $\mathcal{A}$ and tosses a random coin $\alpha \in \{0,1\}$. Next, it receives a public key of the PKE scheme from $\mathcal{C}_{\mathsf{PKE}}$. According to the value α, $\mathcal{S}$ behaves differently. If $\alpha = 0$, then $\mathcal{S}$ sets the received public key to $pk_{t,1}$. $\mathcal{S}$ then performs $\mathsf{PKE.KeyGen}(\lambda) \to (pk_{i,1}, sk_{i,1})$ for $1 \leq i \neq t \leq N$ and $\mathsf{PKE.KeyGen}(\lambda) \to (pk_{i,2}, sk_{i,2})$ for $1 \leq i \leq N$, and passes all $\mathsf{pk}_i = (pk_{i,1}, pk_{i,2})$'s to $\mathcal{A}$. In the challenge phase, once $\mathcal{A}$ sends two messages M_0 and M_1, $\mathcal{S}$ transfers them to $\mathcal{C}_{\mathsf{PKE}}$ and receives a ciphertext $C^*_{t,1,b}$, which is a ciphertext of message M_b using the public key $pk_{t,1}$ for a random bit b chosen by $\mathcal{C}_{\mathsf{PKE}}$. $\mathcal{S}$ chooses a random bit β and a random value R from $\{0,1\}^\ell$, and runs $\mathsf{PKE.Enc}(pk_{t,2}, \mathcal{H}_1(M_\beta); \sigma^*_{t,2}) \to C^*_{t,2,\beta}$. It returns $CT^*_t = (C^*_{t,1}, C^*_{t,2}, C^*_{t,3}) = (C^*_{t,1,b}, C^*_{t,2,\beta}, R)$ to $\mathcal{A}$ as the challenge ciphertext. We should describe how to respond to $\mathcal{A}$'s queries. As for decryption queries, if an input contains the first or second component of the challenge ciphertext (i.e., $C^*_{t,1}$ or $C^*_{t,2}$), $\mathcal{S}$ returns $\perp$. Otherwise, $\mathcal{S}$ can correctly answer by using the decryption oracle offered by $\mathcal{C}_{\mathsf{PKE}}$ and running $\mathsf{PKE.Dec}$ with the secret key $sk_{t,2}$. As for all other queries, $\mathcal{S}$ behaves exactly the same as in the proof of Lemma 1.

On the other hand, if $\alpha = 1$, then $\mathcal{S}$ sets the received public key to $pk_{t,2}$, performs $\mathsf{PKE.KeyGen}(\lambda) \to (pk_{i,1}, sk_{i,1})$ for $1 \leq i \leq N$ and $\mathsf{PKE.KeyGen}(\lambda) \to (pk_{i,2}, sk_{i,2})$ for $1 \leq i \neq t \leq N$, and passes all $\mathsf{pk}_i = (pk_{i,1}, pk_{i,2})$'s to $\mathcal{A}$. In the challenge phase, once $\mathcal{S}$ receives two messages M_0 and M_1 from $\mathcal{A}$, it delivers $\mathcal{H}_1(M_0)$ and $\mathcal{H}_1(M_1)$ to $\mathcal{C}_{\mathsf{PKE}}$ and then receives a ciphertext $C^*_{t,2,b}$, which is a ciphertext of message $\mathcal{H}_1(M_b)$ using the public key $pk_{t,2}$ for a random bit b chosen by $\mathcal{C}_{\mathsf{PKE}}$. $\mathcal{S}$ chooses a random bit β and a random value R from $\{0,1\}^\ell$, runs $\mathsf{PKE.Enc}(pk_{t,1}, M_\beta; \sigma^*_{t,1}) \to C^*_{t,1,\beta}$, and sends $CT^*_t = (C^*_{t,1}, C^*_{t,2}, C^*_{t,3}) = (C^*_{t,1,\beta}, C^*_{t,2,b}, R)$ to $\mathcal{A}$ as the challenge ciphertext. We note that $\mathcal{S}$ can correctly answer all $\mathcal{A}$'s queries, similarly to the case that $\alpha = 0$.

Finally, for both cases ($\alpha = 0$ and $\alpha = 1$), once $\mathcal{A}$ outputs b' at the end of the interaction, $\mathcal{S}$ transfers it to $\mathcal{C}_{\mathsf{PKE}}$ as its answer.

Next, we analyze the advantage of the above simulator $\mathcal{S}$ in the IND-CCA2 security game for PKE. $\mathcal{S}$'s selection of β is completely hidden from the viewpoint of $\mathcal{A}$. If β is equal to b, then all $\mathcal{S}$'s transcripts are identical to those of the challenger in $\mathbf{Game}_2$, in particular, regardless of α. On the other hand, if $\beta = 1 - b$, then one can easily check that $\mathcal{S}$ actually simulates the challenger in $\mathbf{Game}_3$. Even though $\mathcal{S}$ chooses α at

the very early stage, it does not matter since α effects on the location of b between $C^*_{t,1}$ and $C^*_{t,2}$ only and the simulation is identical to the real game in the viewpoint of $\mathcal{A}$. From the above facts, we compute the advantage of $\mathcal{S}$ as follows:

$$\begin{aligned}
\mathbf{Adv}^{\text{IND-CCA2}}_{\mathcal{S},\mathsf{PKE}}(\lambda) &= \Big|\Pr[b'=b]-\frac{1}{2}\Big| \\
&= \Big|\frac{1}{2}\big(\Pr[b'=b|b=\beta]+\Pr[b'=b|b\neq\beta]\big)-\frac{1}{2}\Big| \\
&= \frac{1}{2}\Big|\Pr[\mathcal{A}\to b \text{ in } \mathbf{Game}_2]+\Pr[\mathcal{A}\to b \text{ in } \mathbf{Game}_3]-1\Big| \\
&= \frac{1}{2}\Big|\big(\Pr[\mathcal{A}\to b \text{ in } \mathbf{Game}_2]-\frac{1}{2}\big)+\big(\Pr[\mathcal{A}\to b \text{ in } \mathbf{Game}_3]-\frac{1}{2}\big)\Big| \\
&\geq \frac{1}{2}\Big|\Big(\Big|\Pr[\mathcal{A}\to b \text{ in } \mathbf{Game}_2]-\frac{1}{2}\Big|-\Big|\Pr[\mathcal{A}\to b \text{ in } \mathbf{Game}_3]-\frac{1}{2}\Big|\Big)\Big| \\
&= \frac{1}{2}\Big|\varepsilon_2-\varepsilon_3\Big|.
\end{aligned}$$

Since we assume that $|\varepsilon_2-\varepsilon_3|$ is non-negligible at the beginning of the proof, the advantage of $\mathcal{S}$ is also non-negligible. This contradicts with the IND-CCA2 security of PKE and thus the proof of Lemma 3 is completed. □

Lemma 4. *For any PPT adversary $\mathcal{A}$, its advantage in* $\mathbf{Game}_3$ *is* 0.

Proof of Lemma 4. In the challenge phase of $\mathbf{Game}_3$, the challenger behaves differently according to α. If $\alpha=0$, the challenger uses M_b and $\mathcal{H}_1(M_{1-b})$ in generation of $C^*_{t,1}$ and $C^*_{t,2}$, respectively. On the other hand, if $\alpha=1$, the challenger uses M_{1-b} and $\mathcal{H}_1(M_b)$, instead of M_b and $\mathcal{H}_1(M_{1-b})$. We argue that $\mathcal{A}$ cannot find the value b correctly even when $\mathcal{A}$ can decrypt ciphertexts, because α is completely hidden from the viewpoint of $\mathcal{A}$ and thus $\mathcal{A}$ cannot distinguish which part of the ciphertext contains b. Therefore, the advantage of $\mathcal{A}$ in $\mathbf{Game}_3$ is 0. □

Putting all the results in the series of the above lemmas together, we conclude that any PPT adversary cannot have a non-negligible advantage in $\mathbf{Game}_0$ if PKE is IND-CCA2 secure and randomness extractable, and $\mathcal{H}_2$ is modeled as a random oracle. □

4.2. OW-CCA2 Security against Type-I Adversaries

Now, we analyze the OW-CCA2 security of our PKEET construction against Type-I adversaries.

Theorem 3 (OW-CCA2)**.** *If PKE is IND-CCA2 secure and randomness extractable, and $\mathcal{H}_1$ is one-way, then the proposed PKEET scheme in Section 3 exploiting PKE and $\mathcal{H}_1$ is OW-CCA2 secure against Type-I adversaries in the random oracle model.*

Proof. As in the proof of Theorem 2, we will use the standard hybrid arguments. Let N be the number of users in the system and t be the index of the target user. Denote the challenge ciphertext of the target user U_t by $CT^*_t=(C^*_{t,1},C^*_{t,2},C^*_{t,3})$. We begin by defining the following two games between the adversary $\mathcal{A}$ and the challenger $\mathcal{C}$.

- $\mathbf{Game}_0$ is the original OW-CCA2 security game described in Definition 4.
- $\mathbf{Game}_1$ is almost the same as $\mathbf{Game}_0$, except that if $\mathcal{A}$ queries the decryption oracle on a ciphertext of form $CT'_t=(C^*_{t,1},C'_{t,2},C'_{t,3})$, then $\mathcal{C}$ returns $\perp$.

Now, we analyze success probabilities of adversaries in the above two games. For $i=0,1$, denote the success probability of $\mathcal{A}$ in $\mathbf{Game}_i$ by ε_i, i.e., $\varepsilon_i=\Pr[\mathcal{A}$ wins in $\mathbf{Game}_i]$.

Lemma 5. *If* PKE *is IND-CCA2 secure and randomness extractable, and* $\mathcal{H}_1$ *is one-way, then any PPT adversary* $\mathcal{A}$ *has a negligible difference between success probabilities in* $\mathbf{Game}_0$ *and* $\mathbf{Game}_1$ *under assuming that* $\mathcal{H}_1$ *and* $\mathcal{H}_2$ *are modeled as random oracles.*

Proof of Lemma 5. The proof of this lemma is almost the same as that of Lemma 1. We describe main differences between the behaviours of the simulators $\mathcal{S}$ for the proofs of this lemma and Lemma 1.

- While $\mathcal{A}$ in the proof of Lemma 1 aims to break the IND-CCA2 security of our PKEET, $\mathcal{A}$ in the proof of this lemma aims to break the OW-CCA2 security. Thus, in the challenge phase, we modify so that $\mathcal{S}$ selects two messages M_0 and M_1 by itself.
- For $\mathcal{A}$'s queries, $\mathcal{S}$ responds as in the proof of Lemma 1. Since $\mathcal{S}$ interacts with $\mathcal{A}$ in the OW-CCA2 security game for our PKEET construction, it additionally provides the following queries to $\mathcal{A}$.
  - For $\mathcal{O}^{\mathsf{Td}}$ queries, t can be asked. That is, $\mathcal{A}$ can request a trapdoor for the target user U_t to $\mathcal{S}$ and $\mathcal{S}$ can respond correctly since it generated the secret key $sk_{t,2}$ by itself.
  - $\mathcal{O}^{\mathcal{H}_1}$ queries: It generates a table $\mathsf{H1Table}$, which is initiated as an empty set. The table has two columns for input and output of $\mathcal{H}_1$ queries. On input M, it first checks whether there exists the same input stored at the table $\mathsf{H1Table}$. If exists, it returns the stored output value in $\mathsf{H1Table}$. Otherwise, it selects a random value $\mathsf{h1value}$ from the plaintext space $\mathcal{P}$, returns it to $\mathcal{A}$, and stores it with the query M into a new row of $\mathsf{H1Table}$.

All the rest of $\mathcal{S}$'s behaviours and its analysis are exactly the same as those in the proof of Lemma 1. □

Next, the following lemma shows that the success probability of any PPT adversary in $\mathbf{Game}_1$ is negligible.

Lemma 6. *If* PKE *is IND-CCA2 secure and randomness extractable, and* $\mathcal{H}_1$ *is one-way, then any PPT adversary* $\mathcal{A}$ *has a negligible success probability in* $\mathbf{Game}_1$ *under assuming that* $\mathcal{H}_1$ *and* $\mathcal{H}_2$ *are modeled as random oracles.*

Proof of Lemma 6. We construct a simulator $\mathcal{S}$ that breaks the IND-CCA2 security of PKE using $\mathcal{A}$ in $\mathbf{Game}_1$ as follows. Denote by $\mathcal{C}_{\mathsf{PKE}}$ the challenger of the IND-CCA2 security game for PKE, who interacts with $\mathcal{S}$.

At the beginning of the game, $\mathcal{S}$ passes the system parameter $\mathsf{params} = (\lambda, \mathsf{PKE})$ to $\mathcal{A}$. Once receiving the target public key of PKE from $\mathcal{C}_{\mathsf{PKE}}$, $\mathcal{S}$ sets it to $pk_{t,1}$, runs $\mathsf{PKE.KeyGen}(\lambda) \to (pk_{i,1}, sk_{i,1})$ for $1 \leq i \neq t \leq N$ and $\mathsf{PKE.KeyGen}(\lambda) \to (pk_{i,2}, sk_{i,2})$ for $1 \leq i \leq N$. Then, it sends all $\mathsf{pk}_i = (pk_{i,1}, pk_{i,2})$'s to $\mathcal{A}$. In the challenge phase, $\mathcal{S}$ selects two messages M_0 and M_1 of the same length, forwards them to $\mathcal{C}_{\mathsf{PKE}}$, and then receives $C^*_{t,1,b}$, which is a ciphertext of message M_b using $pk_{t,1}$ for a random bit b chosen by $\mathcal{C}_{\mathsf{PKE}}$. $\mathcal{S}$ selects a random bit β from $\{0,1\}$ and a random value R from $\{0,1\}^\ell$, and runs $\mathsf{PKE.Enc}(pk_{t,2}, \mathcal{H}_1(M_\beta); \sigma^*_{t,2}) \to C^*_{t,2,\beta}$. It passes $CT^*_t = (C^*_{t,1}, C^*_{t,2}, C^*_{t,3}) = (C^*_{t,1,b}, C^*_{t,2,\beta}, R)$ to $\mathcal{A}$ as the challenge ciphertext. We note that $\mathcal{S}$ responds to all $\mathcal{A}$'s queries as in the proofs of Lemma 1 and Lemma 5. Once $\mathcal{A}$ returns M' as an answer, if $M' = M_\beta$, then $\mathcal{S}$ sets $b' = \beta$. Otherwise, $\mathcal{S}$ selects a random bit b'. Finally, $\mathcal{S}$ sends b' to $\mathcal{C}_{\mathsf{PKE}}$ as its answer.

In the above simulation, if $b = \beta$, then all the simulated transcripts are identical to the real transcripts. But, if $b \neq \beta$, CT^*_t is anomalous and so we cannot expect $\mathcal{A}$'s behaviors. However, we can obtain some probability regarding $\mathcal{A}$'s output by considering another simulation that breaks one-wayness of the hash function $\mathcal{H}_1$. More precisely, by generalizing the case when $b \neq \beta$, at the challenge phase, suppose that $\mathcal{S}$ selects two different messages M'_0 and M'_1, and generates

$$\begin{aligned} C^*_{t,1} &= \mathsf{PKE.Enc}(pk_{t,1}, M'_0; \sigma^*_{t,1}), \\ C^*_{t,2} &= \mathsf{PKE.Enc}(pk_{t,2}, \mathcal{H}_1(M'_1); \sigma^*_{t,2}), \\ C^*_{t,3} &= \mathcal{H}_2(C^*_{t,1}, C^*_{t,2}, \sigma^*_{t,1}, \sigma^*_{t,2}) \end{aligned}$$

for some randomness $\sigma^*_{t,1}, \sigma^*_{t,2}$ and then passes it to $\mathcal{A}$ as the challenge ciphertext. Here, if $\mathcal{A}$ outputs M'_1 correctly, then we can use $\mathcal{A}$ to break one-wayness of $\mathcal{H}_1$ by encrypting the target value, instead of $\mathcal{H}_1(M'_1)$, when generating the challenge ciphertext. Hence, from this simulation, we obtain the relation

$$\Pr[\mathcal{A} \to M'_1 | M'_0 \neq M'_1] < \varepsilon_{\mathcal{H}_1} \tag{1}$$

where we assume that there is no PPT adversary who breaks the one-wayness of $\mathcal{H}_1$ with at least $\varepsilon_{\mathcal{H}_1}$ success probability.

Now, we attempt to compute the advantage of $\mathcal{S}$.

$$\begin{aligned}
\varepsilon_{\mathsf{PKE}} &:= \mathbf{Adv}^{\text{IND-CCA2}}_{\mathcal{S},\mathsf{PKE}}(\lambda) \\
&= \Big| \Pr[b = b'] - \frac{1}{2} \Big| \\
&= \Big| \Big(\Pr[b = b' | b = \beta] \Pr[b = \beta] + \Pr[b = b' | b \neq \beta] \Pr[b \neq \beta] \Big) - \frac{1}{2} \Big| \\
&= \Big| \frac{1}{2} \Big(\Pr[b = b' | b = \beta] + \Pr[b = b' | b \neq \beta] \Big) - \frac{1}{2} \Big| \\
&= \Big| \frac{1}{2} \Big(\Pr[(\mathcal{A} \to M_b) \vee ((\mathcal{A} \not\to M_b) \wedge (b' = \beta)) | b = \beta] + \Pr[b = b' | b \neq \beta] \Big) - \frac{1}{2} \Big| \\
&= \Big| \frac{1}{2} \Big(\Pr[\mathcal{A} \to M_b | b = \beta] + \Pr[(\mathcal{A} \not\to M_b) \wedge (b' = \beta) | b = \beta] + \Pr[b = b' | b \neq \beta] \Big) - \frac{1}{2} \Big| \\
&= \frac{1}{2} \Big| \Big(\Pr[\mathcal{A} \text{ wins in } \mathbf{Game}_1] + \frac{1}{2} \Big(1 - \Pr[\mathcal{A} \text{ wins in } \mathbf{Game}_1] \Big) + \Pr[b = b' | b \neq \beta] \Big) - 1 \Big| \\
&= \frac{1}{2} \Big| \frac{1}{2} \varepsilon_1 - \frac{1}{2} + \Pr[b = b' | b \neq \beta] \Big| \\
&= \frac{1}{2} \Big| \frac{1}{2} \varepsilon_1 - \frac{1}{2} + \Pr[(\mathcal{A} \not\to M_\beta) \wedge (b = \beta') | b \neq \beta] \Big| \\
&= \frac{1}{2} \Big| \frac{1}{2} \varepsilon_1 - \frac{1}{2} + \frac{1}{2} \Big(1 - \Pr[\mathcal{A} \to M_\beta | b \neq \beta] \Big) \Big| \\
&= \frac{1}{4} | \varepsilon_1 - \Pr[\mathcal{A} \to M_\beta | b \neq \beta] |.
\end{aligned} \tag{2}$$

Let $\Gamma = \Pr[\mathcal{A} \to M_\beta | b \neq \beta]$. Then, from the relations (1) and (2), we have

$$\frac{1}{4}(\varepsilon_1 - \varepsilon_{\mathcal{H}_1}) \leq \frac{1}{4}(\varepsilon_1 - \Gamma) \leq \varepsilon_{\mathsf{PKE}} \leq \frac{1}{4}(\varepsilon_1 + \Gamma) \leq \frac{1}{4}(\varepsilon_1 + \varepsilon_{\mathcal{H}_1}).$$

Hence, we obtain the relation

$$4\varepsilon_{\mathsf{PKE}} - \varepsilon_{\mathcal{H}_1} \leq \varepsilon_1 \leq 4\varepsilon_{\mathsf{PKE}} + \varepsilon_{\mathcal{H}_1}$$

and ε_1 is negligible since we assume that PKE is IND-CCA2 secure and $\mathcal{H}_1$ is one-way. Therefore, the proof of Lemma 6 is completed. □

Putting the results in Lemmas 5 and 6 together, we conclude that any PPT adversary cannot have a non-negligible success probability in $\mathbf{Game}_0$ if PKE is IND-CCA2 secure and randomness extractable, and $\mathcal{H}_1$ is one-way in the random oracle model.

□

5. Discussion

In this section, we show that the outcome of the FO transformation [16], which is broadly utilized to obtain IND-CCA2 secure PKE schemes in the random oracle model, has randomness extractability. We also provide a comparison of our generic construction with existing PKEET schemes under Tang's all-or-nothing PKEET model [33].

5.1. Fujisaki-Okamoto Trasformation implies Randomness Extractability

We recall the FO transformation [16], which is a generic transformation for IND-CCA2 secure PKE. Let $\mathsf{PKE} = (\mathsf{PKE.KeyGen}, \mathsf{PKE.Enc}, \mathsf{PKE.Dec})$ and $\mathsf{SKE} = (\mathsf{SKE.Enc}, \mathsf{SKE.Dec})$ be a PKE scheme and a symmetric key encryption scheme (SKE), respectively. Let $H : \{0,1\}^* \to \mathcal{R}$ be a hash function and G be a pseudorandom function, where $\mathcal{R}$ is the domain of the randomness utilized in the $\mathsf{PKE.Enc}$ algorithm. The FO transformation FO consists of the following three PPT algorithms:

- $\mathsf{FO.KeyGen}(\lambda)$: It takes a security parameter λ as an input, performs $\mathsf{PKE.KeyGen}(\lambda) \to (pk, sk)$, and returns (pk, sk).
- $\mathsf{FO.Enc}(pk, M; \sigma)$: It takes the public key pk, a message M and a value σ as inputs. Then, it runs $\mathsf{SKE.Enc}(G(\sigma), M) \to c_1$ and $\mathsf{PKE.Enc}(pk, \sigma; H(\sigma, c_1)) \to c_2$, and outputs $C = (c_1, c_2)$. Note that the $\mathsf{SKE.Enc}(sk', m)$ algorithm takes a secret key sk' of the SKE scheme and a message m as inputs and returns a ciphertext.
- $\mathsf{FO.Dec}(sk, C)$: It takes a secret key sk and a ciphertext C as inputs. Then, it parses C as (c_1, c_2) and runs $\mathsf{PKE.Dec}(sk, c_2) \to \sigma'$. If the outcome of $\mathsf{PKE.Enc}(pk, \sigma'; H(\sigma', c_1))$ is equal to c_2, then run $\mathsf{SKE.Dec}(G(\sigma'), c_1) \to M'$ and output M'. Otherwise, output $\perp$. Note that $\mathsf{SKE.Dec}(sk', c)$ takes a secret key sk' and a ciphertext c as inputs and returns a message.

Now, we argue that the outcome of the FO transformation has randomness extractability. To this end, it is sufficient to show that one who has the secret key sk can obtain the randomness σ used in the ciphertext $C = (c_1, c_2)$. If the exploited PKE scheme PKE is correct, the $\mathsf{PKE.Dec}(sk, c_2)$ algorithm returns σ and thus we confirm that it has randomness extractability. In other words, the randomness extract algorithm Ext for the above scheme takes the secret key sk and the ciphertext $C = (c_1, c_2)$ as inputs and returns σ, which is the output of $\mathsf{PKE.Dec}(sk, c_2)$.

To obtain an IND-CCA2 secure PKE scheme using the FO transformation in [16], the underlying primitives must satisfy the following conditions:

- The exploited PKE scheme PKE is *one-way* in the sense that for any PPT adversary $\mathcal{A}$, the advantage of $\mathcal{A}$ defined as

  $$\Pr[\mathcal{A}(pk, c) = \mathsf{PKE.Dec}(sk, c)]$$

  is negligible in the security parameter λ where $\mathsf{PKE.KeyGen}(\lambda) \to (pk, sk)$, $\mathsf{PKE.Enc}(pk, M) \to c$ for a random message M chosen by the challenger from the plaintext space $\mathcal{P}$.
- The exploited SKE scheme SKE is *one-time secure* in the sense that for any PPT adversary $\mathcal{A}$, the advantage of $\mathcal{A}$ defined as

  $$\left| \Pr[\mathcal{A}(c) \to b] - \frac{1}{2} \right|$$

  is neglgibile in the security parameter where $\mathsf{SKE.Enc}(sk', M_b) \to c$ for a random secret key sk' and a random bit b from the secret key space and $\{0,1\}$, respectively, chosen by the challenger, and two messages M_0, M_1 chosen by $\mathcal{A}$.

Refer to [16] for the details. By combining the above result with our proposed generic construction, we summarize that one can obtain a PKEET construction in the random oracle model if there exist a one-way PKE scheme, a one-time secure SKE scheme, collision-resistant and one-way hash functions, and a pseudorandom function.

5.2. Comparison with Previous Results of (Semi-)Generic PKEET Schemes

To the best of our knowledge, there exist only three (semi-)generic constructions [21, 22, 25] for PKEET, except our proposed scheme. All constructions follow Tang's all-or-nothing PKEET model [33]. First, we

Table 2: Feature Comparison of Ours with Previous (Semi-)Generic Constructions

	[21]	[22]	[25]	This Paper
Type	Semi-Generic	Generic	Generic	Generic
Security Model	RO	Standard	RO	RO
Underlying Primitives (Assumption)	– PKE (IND-CCA2) – Hash function (One-way) – Number-theoretic assumption (CDH)	– HIBE (IND-ID-CCA2) – Hash function (One-way) – Signature (One-time SUF)	– PKE (IND-CCA2) – Hash function (One-way) – Lagrange interpolation	– PKE with RE (IND-CCA2) – Hash function (One-way)

Legends: RO: random oracle model, PKE: public key encryption, HIBE: hierarchical identity-based encryption, RE: randomness extractability, IND-CCA2: the indistinguishability against adaptive chosen ciphertext attacks, IND-ID-CCA2: the indistinguishability against adaptive identity and chosen ciphertext attacks, CDH: computational Diffie-Hellman assumption, SUF: strong unforgeability

provide a feature comparison of the proposed construction with the existing (semi-)generic constructions for PKEET. In Table 2, the second, third, fourth and last columns present the features of the recently proposed semi-generic construction in the random oracle model [21], the recently proposed generic construction in the standard model [22], the recently and independently presented generic construction in the random oracle model [25] and the current construction in this paper, respectively. While the construction in [22] is the first and only secure generic PKEET in the standard model, it exploits more advanced primitives such as HIBE and strongly unforgeable (SUF) signatures. Informally, strongly unforgeability of a signature means that it is hard to forge a signature even though a PPT adversary can access to signing oracles adaptively and polynomially many times. (Refer to [8, 22] for the formal definition of SUF signature.) On the other hand, all other schemes in [21], [25] and this paper are secure in the random oracle model, but exploit more fundamental primitives such as (IND-CCA2 secure) PKE schemes and cryptographic hash functions. The main differences among the schemes in the random oracle model are techniques used for preventing CCA2 attacks: While the scheme in [21] attaches instances of the computational Diffie-Hellman (CDH) problem to a message and its hash value and the scheme in [25] replaces instances of the CDH problem by points on a randomly selected polynomial, our construction exploits the randomness extractability, which is naturally achieved by applying the FO transformation. (See the latter part of this subsection for the details.) We remark that the authors of [21] argued that their scheme is *semi-generic* in the sense that along with generic assumptions it additionally requires the CDH assumption, which is one of specific number-theoretic assumptions widely utilized in many cryptographic constructions.

Next, we give an efficiency comaprison of specific PKEET schemes. For fair comparison, we restrict our attention to existing schemes in the random oracle model under Tang's all-or-nothing model [33]. In Table 3, the third, fourth, fifth, and last columns present efficiency of Tang's PKEET scheme [33], an instantiation of the semi-generic construction in the random oracle model [21], an instantiation of the generic construction in the random oracle model [25], and ours, respectively. To obtain specific PKEET schemes, we exploited the FO transformation [16] by employing hashed ElGamal encryption [14] as a PKE scheme and AES encryption [12] as a SKE scheme. We set both the ciphertext size of AES encryption and the output size of exploited hash functions to 2λ for the security parameter λ. We also set message spaces for PKEET schemes listed in Table 3 to the set $\{0,1\}^{2\lambda}$, because the size of message space should be exponential in the security parameter λ and the min-entropy of message distribution should be as high as λ in PKEET schemes to prevent brute-force attacks.

For better understanding, let us look at differences among instantiations obtained from (semi-)generic constructions. Before introducing them, we first recall a hashed ElGamal encryption scheme below, which

Table 3: Efficiency comparison of specific schemes in the random oracle model under Tang's all-or-nothing PKEET model

		[33]	[21]	[25]	This Paper
Comp of	Enc	5Exp	6Exp+$(\frac{\lvert\mathbb{G}\rvert}{\lambda}+2)$SE	4Exp + 6SE	4Exp+2SE
	Dec	2Exp	3Exp+$(\frac{\lvert\mathbb{G}\rvert}{\lambda}+2)$SE	2Exp + 6SE	2Exp+2SE
	Test	4Exp	2Exp+$(\frac{\lvert\mathbb{G}\rvert}{\lambda}+2)$SE	2Exp + 6SE	2Exp+2SE
Size of	PK	$2\lvert\mathbb{G}\rvert$	$3\lvert\mathbb{G}\rvert$	$2\lvert\mathbb{G}\rvert$	$2\lvert\mathbb{G}\rvert$
	CT	$4\lvert\mathbb{G}\rvert + \lvert\mathbb{Z}_p\rvert + 2\lambda$	$4\lvert\mathbb{G}\rvert + 10\lambda$	$2\lvert\mathbb{G}\rvert + 18\lambda$	$2\lvert\mathbb{G}\rvert + 10\lambda$
	TD	$\lvert\mathbb{Z}_p\rvert$	$\lvert\mathbb{Z}_p\rvert$	$\lvert\mathbb{Z}_p\rvert$	$\lvert\mathbb{Z}_p\rvert$

Legends: Comp: computational cost, Enc: encryption algorithm, Dec: decryption algorithm, Test: test algorithm, PK: public key, CT: ciphertext, TD: trapdoor, Exp: cost for an exponentiation, BP: cost for a bilinear map evaluation, SE: cost for an AES encryption/decryption when a message is 2λ-bit, $|\mathbb{G}|$: the bit-length required to represent elements in an underlying group $\mathbb{G}$, $|\mathbb{Z}_p|$: the bit-length to represent elements in $\mathbb{Z}_p$ where p is an order of $\mathbb{G}$, λ: security parameter, RO: random oracle

will be employed to generate instantiations from (semi-)generic constructions.

- $\mathsf{HEIG.Gen}(\lambda)$: Given a security parameter λ,
  1. Generate a multiplicative group $\mathbb{G}$ of prime order $p = p(\lambda)$.
  2. Select a random generator g of $\mathbb{G}$.
  3. Select a random element x from $\mathbb{Z}_p^*$ and compute $y = g^x$.
  4. Generate a cryptographic hash function $\mathcal{H}' : \{0,1\}^* \to \{0,1\}^{2\lambda}$.
  5. Output a public key $pk = y$ and a secret key x.

  Note that algorithms below take parameters $(p, g, \mathcal{H}')$ as inputs implicitly, though it is not stated.
- $\mathsf{HEIG.Enc}(pk, M; r)$: Given a public key $pk = y$, a message $M \in \{0,1\}^{2\lambda}$ and a randomness r,
  1. Compute $c_1 = g^r$ and $c_2 = M \oplus \mathcal{H}'(y^r)$.
  2. Output $C = (c_1, c_2)$.
- $\mathsf{HEIG.Dec}(sk, C)$: Given a secret key $sk = x$ and a ciphertext $C = (c_1, c_2)$, compute $M' = c_2 \oplus \mathcal{H}'(c_1^x)$ and output M'.

From the above, we can easily check that encryption and decryption algorithms for hashed ElGamal encryption take two exponentiations and one exponentiation, respectively. A ciphertext consists of one group element in $\mathbb{G}$ and a string which has the same size as an output of hash function. Thus, its ciphertext size is $|\mathbb{G}| + 2\lambda$ where $|\mathbb{G}|$ is the bit-length required for representing an element in $\mathbb{G}$.

Now, we are ready to look into differences among instantiations obtained from generic constructions. Let $\mathbb{G}$ be a multiplicative group of order p and g be a generator of $\mathbb{G}$. $\mathcal{H}_1 : \{0,1\}^{2\lambda} \to \{0,1\}^{2\lambda}$, $\mathcal{H}_2 : \{0,1\}^* \to \{0,1\}^{2\lambda}$, and $\mathcal{H}_3 : \{0,1\}^* \to \mathbb{Z}_p^*$ are cryptographic hash functions. G is a pseudo-random function. We note that the public parameter params includes $(\mathbb{G}, g, p, \mathcal{H}_1, \mathcal{H}_2, \mathcal{H}_3, G)$. $\mathsf{AES.Enc}(sk, M)$ (resp., $\mathsf{AES.Dec}(sk, C)$) denotes the AES encryption (resp., decryption) algorithm that takes a secret key sk and a message M (resp., a ciphertext C) as inputs and returns an AES ciphertext (resp., a message). When we exploit hashed ElGamal encryption and AES encryption for our generic construction, our encryption and decryption algorithms are defined as follows. Let $\mathsf{pk} = (y_1, y_2)$ be a public key of our instantiation where $\mathsf{sk} = (x_1, x_2)$ is a corresponding secret key such that $y_1 = g^{x_1}$ and $y_2 = g^{x_2}$.

- $\mathsf{Ours.Enc}(\mathsf{pk}, M; \sigma)$: Given a public key $\mathsf{pk} = (y_1, y_2)$, a message $M \in \{0,1\}^{2\lambda}$ and a randomness $\sigma = (\sigma_1, \sigma_2)$,
  1. Run $\mathsf{AES.Enc}(G(\sigma_1), M) \to c_1$ and $\mathsf{AES.Enc}(G(\sigma_2), \mathcal{H}_1(M)) \to c_4$
  2. Compute $r_1 = \mathcal{H}_3(\sigma_1, c_1)$ and $r_2 = \mathcal{H}_3(\sigma_2, c_4)$.
  3. Run $\mathsf{HElG.Enc}(y_1, \sigma_1; r_1) \to (c_2, c_3)$ and $\mathsf{HElG.Enc}(y_2, \sigma_2; r_2) \to (c_5, c_6)$.
  4. Set $C_1 = (c_1, c_2, c_3)$ and $C_2 = (c_4, c_5, c_6)$.
  5. Compute $C_3 = \mathcal{H}_2(C_1, C_2, \sigma_1, \sigma_2)$.
  6. Output $CT = (C_1, C_2, C_3)$.
- $\mathsf{Ours.Dec}(\mathsf{sk}, CT)$: Given a secret key $\mathsf{sk} = (x_1, x_2)$ and a ciphertext $CT = (C_1, C_2, C_3)$,
  1. Parse $C_1 = (c_1, c_2, c_3)$ and $C_2 = (c_4, c_5, c_6)$.
  2. Run $\mathsf{HElG.Dec}(x_1, (c_2, c_3)) \to \sigma_1'$ and $\mathsf{HElG.Dec}(x_2, (c_5, c_6)) \to \sigma_2'$.
  3. Run $\mathsf{AES.Dec}(G(\sigma_1'), c_1) \to m'$ and $\mathsf{AES.Enc}(G(\sigma_2'), c_4) \to h'$.
  4. Return 1 if both $h' = \mathcal{H}_1(m')$ and $C_3 = \mathcal{H}_2(C_1, C_2, \sigma_1', \sigma_2')$ hold.

Our encryption algorithm takes 2 AES encryption for Step 1 and 2 hashed ElGamal encryption for Step 3.[2] Thus, it requires 2 AES encryption and 4 exponentiations for encryption in total. Our decryption algorithm takes 2 hashed ElGamal decryption for Step 2 and 2 AES decryption for Step 3. Thus, it requires 2 AES encryption and 2 exponentiation for decryption in total. We note that a trapdoor for equality test is x_2 and the test algorithm performs equality test by decrypting C_2 using x_2. Thus, the test algorithm takes 2 exponentiations and 2 AES decryptions to decrypt C_2's in two ciphertexts. Finally, a ciphertext consists of 2 AES ciphertexts, 2 hashed ElGamal ciphertexts and one hashed value. Thus, a ciphertext size is $2|\mathbb{G}| + 10\lambda$.

On the other hand, in a ciphertext of an encryption algorithm $\mathsf{SG.Enc}(\mathsf{pk}, M; \sigma)$ for an instantiation of the semi-generic construction under the same setting, c_1, c_4, and C_3 are replaced by

$$\begin{aligned} c_1 &= \mathsf{AES.Enc}(G(\sigma_1), M \| g^{\overline{r}}), \\ c_4 &= \mathsf{AES.Enc}(G(\sigma_2), \mathcal{H}_1(M) \| g^{\overline{r}}), \text{ and} \\ C_3 &= \mathcal{H}_2(C_1, C_2, \overline{y}^{\overline{r}}), \end{aligned}$$

respectively, where $\overline{x}$, which is randomly chosen from $\mathbb{Z}_p^*$, and $\overline{y} = g^{\overline{x}}$ are included in the secret key and the public key, respectively, and $\overline{r}$ is a randomly chosen element from $\mathbb{Z}_p^*$. Thus, their encryption algorithm takes 2 additional exponentiations to compute $g^{\overline{r}}$ and $\overline{y}^{\overline{r}}$. In their decryption algorithm, it obtains $g^{\overline{r}}$ by decrypting c_1 and c_4 and computes $\overline{y}^{\overline{r}}$ to check the validity of C_3 component. Hence, the decryption algorithm additionally requires 1 exponentiation to compute $\overline{y}^{\overline{r}} = (g^{\overline{r}})^{\overline{x}}$. Furthermore, while the message size of each AES encryption in our instantiation is 2λ, that in the instantiation of the semi-generic construction is $|\mathbb{G}|+2\lambda$. So, if we assume that AES algorithm encrypts a message of 2λ-bit at once, an instantiation for the semi-generic algorithm requires $\dfrac{|\mathbb{G}|}{\lambda}$ AES encryption/decryption additionally for encryption and decryption each. The ciphertext size also occurs an increase of $2|\mathbb{G}|$. Therefore, the instantiation of our generic construction is more efficient and simpler than the instantiation of the semi-generic construction in [21].

In case of a ciphertext for an instantiation of the generic construction in [25], c_1, c_4 and C_3 are replaced by

$$\begin{aligned} c_1 &= \mathsf{AES.Enc}(G(\sigma_1), M \| a_1 \| b_1), \\ c_4 &= \mathsf{AES.Enc}(G(\sigma_2), \mathcal{H}_1(M) \| a_2 \| b_2), \text{ and} \\ C_3 &= \mathcal{H}_2(C_1, C_2, f(0)), \end{aligned}$$

[2]Generally, an exponentiation is much more expensive than a hash computation and thus we ignore the cost for hash computations in our analysis.

respectively, where $f(X)$ is a randomly selected polynomial of degree 2 in $\mathbb{Z}_q^*[X]$ for sufficiently large prime q and (a_1, b_1), (a_2, b_2) are points on $Y = f(X)$ randomly chosen in the encryption algorithm. The above algorithm has almost the same performance as ours. However, each message size of AES encryption is 3 times larger than ours when we set q to a 2λ-bit prime. So, if we assume that AES algorithm encrypts a message of 2λ-bit at once, it causes 4 additional AES encryption and decryption for their encryption and decryption algorithms, respectively. In addition, the ciphertext size occurs an increase of 8λ. (That is, each c_1, c_4 in their scheme is 4λ-bit longer than ours.) Therefore, our instantiation slightly outperforms that of the recent generic construction in [25].

Finally, Table 3 also tells us that the instantiation of our generic construction has comparable efficiency to Tang's PKEET scheme [33]: By considering that AES encryption and decryption are much cheaper than an exponentiation at the same security level, our encryption and test algorithms are more efficient than those of Tang's scheme. The ciphertext size of ours is also shorter than that of Tang's scheme since $|\mathbb{G}|$ is larger than 10λ when $\mathbb{G}$ is a multiplicative subgroup of $\mathbb{Z}_q^*$ for some prime q. For example, $|\mathbb{G}|$ is 3,072 and the order of $\mathbb{G}$ $(= p)$ is 256-bit for $\lambda = 128$ when we use a multiplicative subgroup of $\mathbb{Z}_q^*$ [28]. (One may consider to use elliptic curve settings to reduce the size of $|\mathbb{G}|$, but it causes expensive computational costs for exponentiations.)

6. Conclusion

In this paper, we presented a PKEET construction from only generic assumptions in the random oracle model. Our proposal exploits cryptographic hash functions and an IND-CCA2 secure PKE scheme with randomness extractability. We showed that PKE schemes obtained by using the FO transformation, which is widely utilized to obtain IND-CCA2 secure PKE schemes in the random oracle model, satisfy this property. As a result, we obtain a PKEET construction in the random oracle model if we have a one-way PKE scheme, a one-time secure SKE scheme, collision-resistant and one-way hash functions, and a pseudorandom function. Finally, we provided comparisons of ours and other generic/specific PKEET schemes under the same setting with ours.

From the viewpoint of theoretical aspects, our proposal requires weaker generic assumptions in the sense that it requires very fundamental primitives only. However, this does not guarantee that the efficiency of the concrete instantiation obtained from our construction is optimal. Therefore, for practical use, it would be interesting to improve the efficiency of the currently existing PKEET constructions.

Acknowledgements

The authors would like to thank the anonymous reviewers for their helpful comments. Hyung Tae Lee was supported by research funds for newly appointed professors of Chonbuk National University in 2017 and Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science and ICT (No. NRF-2018R1C1B6008476). San Ling and Huaxiong Wang were supported by Research Grant TL-9014101684-01 and the Singapore Ministry of Education under Research Grants MOE2013-T2-1-041 and MOE2016-T2-2-014 (S). Huaxiong Wang was also supported by the National Research Foundation, Prime Ministers Office, Singapore under its Strategic Capability Research Centres Funding Initiative. Jae Hong Seo was supported by Institute for Information & communications Technology Promotion (IITP) grant funded by the Korea government (MSIT) (No. 2016-6-00600, A Study on Functional Encryption: Construction, Security Analysis, and Implementation).

References

[1] M. Abdalla, M. Bellare, D. Catalano, E. Kiltz, T. Kohno, T. Lange, J. Malone-Lee, G. Neven, P. Paillier, and H. Shi. Searchable encryption revisited: Consistency properties, relation to anonymous ibe, and extensions. *J. Cryptology*, 21(3):350–391, 2008.

[2] M. Abdalla, F. Bourse, A. D. Caro, and D. Pointcheval. Simple functional encryption schemes for inner products. In J. Katz, editor, *Public-Key Cryptography - PKC 2015*, volume 9020 of *LNCS*, pages 733–751. Springer, 2015.

[3] S. Agrawal, B. Libert, and D. Stehlé. Fully secure functional encryption for inner products, from standard assumptions. In M. Robshaw and J. Katz, editors, *Advances in Cryptology - CRYPTO 2016 Part III*, volume 9816 of *LNCS*, pages 333–362. Springer, 2016.

[4] D. Boneh, G. D. Crescenzo, R. Ostrovsky, and G. Persiano. Public key encryption with keyword search. In C. Cachin and J. Camenisch, editors, *Advances in Cryptology - EUROCRYPT 2004*, volume 3027 of *LNCS*, pages 506–522. Springer, 2004.

[5] D. Boneh and M. K. Franklin. Identity-based encryption from the weil pairing. In J. Kilian, editor, *Advances in Cryptology - CRYPTO 2001*, volume 2139 of *LNCS*, pages 213–229. Springer, 2001.

[6] D. Boneh, P. A. Papakonstantinou, C. Rackoff, Y. Vahlis, and B. Waters. On the impossibility of basing identity based encryption on trapdoor permutations. In *Foundations of Computer Science (FOCS) 2008*, pages 283–292. IEEE Computer Society, 2008.

[7] D. Boneh, A. Sahai, and B. Waters. Functional encryption: Definitions and challenges. In Y. Ishai, editor, *Theory of Cryptography (TCC) 2011*, volume 6597 of *LNCS*, pages 253–273. Springer, 2011.

[8] D. Boneh, E. Shen, and B. Waters. Strongly unforgeable signatures based on computational Diffie-Hellman. In M. Yung, Y. Dodis, A. Kiayias, and T. Malkin, editors, *Public Key Cryptography (PKC) 2006*, volume 3958 of *LNCS*, pages 229–240. Springer, 2006.

[9] R. Canetti, O. Goldreich, and S. Halevi. The random oracle methodology, revisited. *J. ACM*, 51(4):557–594, 2004.

[10] J. H. Cheon, P. Fouque, C. Lee, B. Minaud, and H. Ryu. Cryptanalysis of the new CLT multilinear map over the integers. In M. Fischlin and J. Coron, editors, *Advances in Cryptology - EUROCRYPT 2016 Part I*, volume 9665 of *LNCS*, pages 509–536. Springer, 2016.

[11] J. H. Cheon, J. Jeong, and C. Lee. An algorithm for ntru problems and cryptanalysis of the ggh multilinear map without a low level encoding of zero. In *LMS Journal of Computation and Mathematics*, volume 19, pages 255–266, 2016.

[12] J. Daemen and V. Rijmen. Rijndael for AES. In *AES Candidate Conference*, pages 343–348, 2000.

[13] Y. Dodis, R. Oliveira, and K. Pietrzak. On the generic insecurity of the full domain hash. In V. Shoup, editor, *Advances in Cryptology - CRYPTO 2005*, volume 3621 of *LNCS*, pages 449–466. Springer, 2005.

[14] T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Trans. Information Theory*, 31(4):469–472, 1985.

[15] K. Emura, A. Miyaji, M. S. Rahman, and K. Omote. Generic constructions of secure-channel free searchable encryption with adaptive security. *Security and Communication Networks*, 8(8):1547–1560, 2015.

[16] E. Fujisaki and T. Okamoto. Secure integration of asymmetric and symmetric encryption schemes. *J. Cryptology*, 26(1):80–101, 2013.

[17] S. Garg, C. Gentry, S. Halevi, M. Raykova, A. Sahai, and B. Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. *SIAM J. Comput.*, 45(3):882–929, 2016.

[18] S. Garg, C. Gentry, S. Halevi, and M. Zhandry. Functional encryption without obfuscation. In E. Kushilevitz and T. Malkin, editors, *Theory of Cryptography (TCC) 2016-A, Part II*, volume 9563 of *LNCS*, pages 480–511. Springer, 2016.

[19] K. Huang, R. Tso, Y. Chen, S. M. M. Rahman, A. Almogren, and A. Alamri. PKE-AET: Public key encryption with authorized equality test. *Comput. J.*, 58(10):2686–2697, 2015.

[20] H. T. Lee, S. Ling, J. H. Seo, and H. Wang. CCA2 attack and modification of Huang et al.'s public key encryption with authorized equality test. *Comput. J.*, 59(11):1689–1694, 2016.

[21] H. T. Lee, S. Ling, J. H. Seo, and H. Wang. Semi-generic construction of public key encryption and identity-based encryption with equality test. *Information Sciences*, 373:419–440, 2016.

[22] H. T. Lee, S. Ling, J. H. Seo, H. Wang, and T.-Y. Youn. Public key encryption with equality test in the standard model. Cryptology ePrint Archive, Report 2016/1182, 2016. Available at `http://eprint.iacr.org/2016/1182`.

[23] G. Leurent and P. Q. Nguyen. How risky is the random-oracle model? In S. Halevi, editor, *Advances in Cryptology - CRYPTO 2009*, volume 5677 of *LNCS*, pages 445–464. Springer, 2009.

[24] X.-J. Lin, H. Qu, and X. Zhang. Public key encryption supporting equality test and flexible authorization without bilinear pairings. *IACR Cryptology ePrint Archive*, 2016:277, 2016.

[25] X. J. Lin, L. Sun, and H. Qu. Generic construction of public key encryption, identity-based encryption and signcryption with equality test. *Inf. Sci.*, 453:111–126, 2018.

[26] S. Ma, Q. Huang, M. Zhang, and B. Yang. Efficient public key encryption with equality test supporting flexible authorization. *IEEE Trans. Information Forensics and Security*, 10(3):458–470, 2015.

[27] S. Ma, M. Zhang, Q. Huang, and B. Yang. Public key encryption with delegated equality test in a multi-user setting. *Comput. J.*, 58(4):986–1002, 2015.

[28] NIST. Recommendation for key management. Special Publication 800-57 Part 1 Rev. 4, January 2016.

[29] H. Qu, Z. Yan, X. J. Lin, Q. Zhang, and L. Sun. Certificateless public key encryption with equality test. *Inf. Sci.*, 462:76–92, 2018.

[30] H. S. Rhee, J. H. Park, and D. H. Lee. Generic construction of designated tester public-key encryption with keyword search. *Inf. Sci.*, 205:93–109, 2012.

[31] Q. Tang. Towards public key encryption scheme supporting equality test with fine-grained authorization. In U. Parampalli and P. Hawkes, editors, *ACISP 2011*, volume 6812 of *LNCS*, pages 389–406. Springer, 2011.

[32] Q. Tang. Public key encryption schemes supporting equality test with authorisation of different granularity. *IJACT*, 2(4):304–321, 2012.

[33] Q. Tang. Public key encryption supporting plaintext equality test and user-specified authorization. *Security and Communication Networks*, 5(12):1351–1362, 2012.

[34] G. Yang, C. H. Tan, Q. Huang, and D. S. Wong. Probabilistic public key encryption with equality test. In J. Pieprzyk, editor, *Topics in Cryptology - CT-RSA 2010*, volume 5985 of *LNCS*, pages 119–131. Springer, 2010.

[35] K. Zhang, J. Chen, H. T. Lee, H. Qian, and H. Wang. Efficient public key encryption with equality test in the standard model. *Theor. Comput. Sci.*, 755:65–80, 2019.