Research Article

Efficient KDM-CCA Secure Public-Key Encryption via Auxiliary-Input Authenticated Encryption

Shuai Han,1,2 Shengli Liu,1,2,3 and Lin Lyu1,2

1Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai 200240, China
2State Key Laboratory of Cryptology, P.O. Box 5159, Beijing 100878, China
3Westone Cryptologic Research Center, Beijing 100070, China

Correspondence should be addressed to Shengli Liu; sliu@sjtu.edu.cn

Received 1 April 2017; Revised 13 June 2017; Accepted 6 July 2017; Published 11 December 2017

1. Introduction

For public-key encryption (PKE) schemes, Chosen-Ciphertext Attack (CCA) security is the de facto security notion. In the CCA security model, the adversary sees the public key and gets challenge ciphertexts, which are encryptions of messages of its choices. It is also allowed to make decryption queries and obtain the decrypted messages for ciphertexts (but not the challenge ciphertexts) of its choices. CCA security considers whether the challenge ciphertexts can protect the security of messages. Observe that the adversary does not know the secret keys; thus it is not able to submit messages that are closely related to the secret keys. Thus, there is a corner that is not covered by CCA security, that is, the security of messages which are closely dependent on the secret keys. It was Goldwasser and Micali [1] who first pointed out this problem. In 2002, the security of such key-dependent messages (KDM) was formalized by Black et al. [2]. Up to now, KDM-security has found many applications, such as anonymous credential systems [3] and hard disk encryption [4].

KDM-$\mathcal{F}$-security means KDM-security for a set $\mathcal{F}$ of functions. Loosely speaking, in the $n$-KDM-$\mathcal{F}$-security model, the adversary obtains public keys $(pk_1, \ldots, pk_n)$ of $n$ users and has access to an encryption oracle. Each time, the adversary submits a function $f$ in the function set $\mathcal{F}$, the encryption oracle will encrypt $f(sk_1, \ldots, sk_n)$ or a dummy message (say 0) and output the challenge ciphertext to the adversary. The $n$-KDM-$\mathcal{F}$-CCA security stipulates that the adversary cannot distinguish the two cases, and the $n$-KDM-$\mathcal{F}$-CCA security demands the indistinguishability of the two cases even if the adversary is also allowed to make decryption queries. KDM-CCA is obviously stronger than KDM-CPA security notion. Moreover, the KDM-security is stronger when the function set $\mathcal{F}$ is larger.
In 2008, Boneh et al. (BHBO) [4] proposed the first KDM($\mathcal{F}_{\text{aff}}$)-CPA secure PKE construction for the affine function set $\mathcal{F}_{\text{aff}}$, from the Decisional Diffie-Hellman (DDH) assumption. Soon after, the BHBO scheme was generalized by Brakerski and Goldwasser [5], who presented KDM($\mathcal{F}_{\text{aff}}$)-CPA secure PKE constructions under the Quadratic Residuosity (QR) assumption or the Decisional Composite Residuosity (DCR) assumption. However, these schemes suffer from incompact ciphertext, which contains $O(\lambda)$ group elements ($\lambda$ denotes the security parameter throughout the paper).

Applebaum et al. [6] proved that a variant of the Regev scheme [7] is KDM($\mathcal{F}_{\text{aff}}$)-CPA secure and enjoys compact ciphertexts, that is, encompassing only $O(1)$ group elements.

Brakerski et al. [8] provided a KDM($\mathcal{F}_{\text{poly}}$)-CPA secure PKE scheme for the polynomial function set $\mathcal{F}_{\text{poly}}$, which contains all polynomials whose degrees are at most $d$. The drawback of the scheme is incompact ciphertext, which contains $O(d^{d+1})$ group elements.

Barak et al. [9] presented a KDM-CPA secure PKE for the set of Boolean circuits whose sizes are a priori bounded, which is a very large function set. Nevertheless, their scheme is neither practical nor flexible.

In 2011, Malkin et al. [10] proposed the first efficient KDM($\mathcal{F}_{\text{aff}}$)-CPA secure PKE. The ciphertext of their PKE construction is almost compact and consists of only $O(d)$ group elements.

The Groth-Sahai proofs [13] are the only practical NIZK. To construct efficient KDM($\mathcal{F}_{\text{aff}}$)-CPA secure PKE, schemes which either preserve compactness or achieve KDM-CCA security are required. However, the only known efficient KDM($\mathcal{F}_{\text{aff}}$)-CPA secure PKE [10] is incompatible with the Groth-Sahai NIZK proofs [13]; thus the CCS approach must adopt a general inefficient NIZK.

Our Contribution. In this work, we focus on the design of efficient PKE schemes possessing KDM($\mathcal{F}_{\text{aff}}$)-CPA security and KDM($\mathcal{F}_{\text{poly}}$)-CPA security, respectively.

(i) We develop a new primitive named “Auxiliary-Input Authenticated Encryption” (AIAE). We introduce new related-key attack (RKA) security notions for it, called IND-$\mathcal{F}^1$-RKA and weak-INT-$\mathcal{F}^1$-RKA.

(a) We show a general paradigm for constructing such an AIAE from a one-time secure AE and a tag-based hash proof system (HPS) that is universal, extracting, and key-homomorphic.

(b) We present an instantiation of tag-based HPS under the DDH assumption. Following our paradigm, we immediately obtain a DDH-based AIAE for the set of restricted affine functions.

(ii) Using AIAE as an essential building block, we design the first PKE scheme enjoying KDM($\mathcal{F}_{\text{aff}}$)-CPA security and compactness of ciphertexts simultaneously. Specifically, the ciphertext of our scheme contains only $O(1)$ group elements.

(iii) Furthermore, we design the first PKE scheme enjoying KDM($\mathcal{F}_{\text{poly}}$)-CPA security and almost compactness of ciphertexts simultaneously. More precisely, the number of group elements contained in a ciphertext is independent of the security parameter $\lambda$.

In Table 1, we list the existing PKE schemes which either achieve KDM-CCA security or are KDM-secure for the set $\mathcal{F}_{\text{poly}}$ of polynomial functions.
Table 1: Comparison among PKE schemes achieving either KDM-CCA security or security against the set \( \mathcal{F}_d \) of polynomial functions.

Here, we denote by \( \lambda \) the security parameter and by \( \mathcal{F}_{\text{circ}}, \mathcal{F}_{\text{aff}}, \) and \( \mathcal{F}_d \) the set of selection functions, the set of affine functions, and the set of polynomial functions of bounded degree \( d \), respectively. “CCA” indicates that the scheme is KDM-CCA secure. By the symbol “\( ? \)”, we mean that the security proof is not rigorous. \( G, Z_{N^2}, Z_{N^3}, Z_{N^4}, \) and \( Z_{\pi} \) are the underlying groups, where \( s \geq 1 \).

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Set</th>
<th>CCA?</th>
<th>Free of pairing?</th>
<th>The size of ciphertext</th>
<th>Assumption</th>
</tr>
</thead>
<tbody>
<tr>
<td>BHHO08 [4] + CCS09 [11]</td>
<td>( \mathcal{F}_{\text{aff}} )</td>
<td>( \sqrt{\ldots} )</td>
<td>( \sqrt{\ldots} )</td>
<td>( (6\lambda + 13)</td>
<td>G</td>
</tr>
<tr>
<td>BKG11 [8]</td>
<td>( \mathcal{F}_d )</td>
<td>( \sqrt{\ldots} )</td>
<td>( \sqrt{\ldots} )</td>
<td>( (\lambda^{d+1})</td>
<td>G</td>
</tr>
<tr>
<td>MTY11 [10]</td>
<td>( \mathcal{F}_d )</td>
<td>( \sqrt{\ldots} )</td>
<td>( \sqrt{\ldots} )</td>
<td>( (d+2)</td>
<td>Z_{N^2}</td>
</tr>
<tr>
<td>Hof13 [15]</td>
<td>( \mathcal{F}_{\text{circ}} )</td>
<td>( \sqrt{\ldots} )</td>
<td>( \sqrt{\ldots} )</td>
<td>( 6</td>
<td>Z_{N^2}</td>
</tr>
<tr>
<td>LLIH16 [16]</td>
<td>( \mathcal{F}_{\text{aff}} )</td>
<td>?</td>
<td>( \sqrt{\ldots} )</td>
<td>( 3</td>
<td>Z_{N^2}</td>
</tr>
<tr>
<td>Our scheme in Section 4</td>
<td>( \mathcal{F}_{\text{aff}} )</td>
<td>( \sqrt{\ldots} )</td>
<td>( \sqrt{\ldots} )</td>
<td>( 9</td>
<td>Z_{N^2}</td>
</tr>
<tr>
<td>Our scheme in Section 5</td>
<td>( \mathcal{F}_d )</td>
<td>( \sqrt{\ldots} )</td>
<td>( \sqrt{\ldots} )</td>
<td>( 9</td>
<td>Z_{N^2}</td>
</tr>
</tbody>
</table>

Figure 1: Our approach of PKE construction.

Overview of Our Construction. In the construction of our KDM-CCA secure PKE schemes, we adopt a key encapsulation mechanism (KEM) + data encapsulation mechanism (DEM) approach [18] and employ three building blocks: KEM, \( \mathcal{E} \), and AIAE, as shown in Figure 1.

(i) KEM and \( \mathcal{E} \) share the same pair of public and secret keys.

(ii) A key \( k \) is encapsulated by KEM.Encrypt, and an encapsulation \( kem.c \) is generated by KEM.Encrypt along the way.

(iii) The message \( m \) is encrypted by \( \mathcal{E} \).Encrypt, and the resulting \( \mathcal{E} \)-ciphertext is \( \mathcal{E} \cdot c \).

(iv) The key \( k \) generated by KEM is used by AIAE.Encrypt to encrypt \( \mathcal{E} \cdot c \) with auxiliary input \( ai = kem.c \), and the resulting AIAE-ciphertext is \( aiae.c \).

(v) The ciphertext of our PKE scheme is \( (kem.c, aiae.c) \).

Following this approach, we design KDM[\( \mathcal{F}_{\text{aff}} \)]-CCA and KDM[\( \mathcal{F}_d \)]-CCA secure PKE schemes, respectively, by constructing specific building blocks.

Differences to Conference Version. This paper constitutes an extended full version of [17]. The new results in this paper are as follows.

(i) In contrast to presenting a concrete construction of AIAE in the conference paper, we give a general paradigm for constructing AIAE from a one-time secure authenticated encryption (AE) and a tag-based hash proof system (HPS) in this paper.

(a) In Section 3.2, we show that the resulting AIAE is IND-RKA secure and weak-INT-RKA secure, as long as the underlying tag-based HPS is universal, extracting, and key-homomorphic.

(b) In Section 3.3, we give an instantiation of tag-based HPS based on the DDH assumption. Following our paradigm, we obtain a DDH-based AIAE scheme in Section 3.4.

We view the specific AIAE proposed in the conference paper as an instantiation of the general paradigm presented in this paper.

(ii) In this paper, we provide the full proofs of the theorems regarding the KDM[\( \mathcal{F}_{\text{aff}} \)]-CCA security and KDM[\( \mathcal{F}_d \)]-CCA security of our PKEs. Compared with the conference paper, we add the proofs of Lemmas 16, 18, 25, 26, and 29, and the proof of indistinguishability between Hybrids 2 and 3 in Section 5.3.

2. Preliminaries

Throughout this paper, denote by \( \lambda \in \mathbb{N} \) the security parameter. \( y \leftarrow \mathcal{Y} \) means choosing an element \( y \) from set \( \mathcal{Y} \) uniformly. \( y \leftarrow A \left( x; r \right) \) means executing algorithm \( A \) with
input $x$ and randomness $r$ and assigning output to $y$. We sometimes abbreviate this to $y \leftarrow_f x$. “PPT” is short for probabilistic polynomial-time. For integers $n < m$, we denote $[n] = \{1, 2, \ldots, n\}$ and $[n, m] = \{n, n + 1, \ldots, m\}$. For a security notion $\mathcal{Y}$ and a primitive $XX$, the advantage of a PPT adversary $A$ is typically denoted by $\text{Adv}_{X,Y,XX,A}(\lambda)$ and we denote $\text{Adv}_{X,Y}(\lambda) = \max_{\text{PPT},A} \text{Adv}_{X,Y,XX,A}(\lambda)$. Let $\text{negl}(\cdot)$ denote an unspecified negligible function.

**Games.** We will use games in our security definitions and proofs. Typically, a game $G$ begins with an **initialize** procedure and ends with a **finalize** procedure. In the game, there might be other procedures $\text{PROC}_1, \ldots, \text{PROC}_n$ which perform as oracles. All procedures are presented with pseudocode, all sets are initialized as empty sets, and all variables are initialized as empty strings. In the execution of a game $G$ with an adversary $A$, firstly $A$ calls initialize and obtains its output; then $A$ makes arbitrary oracle queries to $\text{PROC}_i$ according to their specifications and obtains their outputs; finally $A$ calls finalize. In the end of the execution, if finalize outputs $b$, then we write this as $G^A \rightarrow b$. The statement $a \xrightarrow{G} b$ means that, in game $G$, $a$ is computed as $b$ or $a$ equals $b$.

**2.1. Public-Key Encryption.** There are four PPT algorithms $\text{PKE} = (\text{ParGen}, \text{KeyGen}, \text{Encrypt}, \text{Decrypt})$ in a public-key encryption (PKE) scheme:

(i) $\text{ParGen}(1^\lambda)$ outputs a public parameter $\text{pars}$. We assume that $\text{pars}$ implicitly defines a secret key space $\mathcal{S}$ and a message space $\mathcal{M}$.

(ii) $\text{KeyGen}(\text{pars})$ takes $\text{pars}$ as input and outputs a public key $\text{pk}$ and a secret key $\text{sk}$.

(iii) $\text{Encrypt}(\text{pk}, m)$ takes $\text{pk}$ and a message $m \in \mathcal{M}$ as input and outputs a ciphertext $\text{ciphertext}$.

(iv) $\text{Decrypt}(\text{sk}, \text{ciphertext})$ takes $\text{sk}$ and a ciphertext $\text{ciphertext}$ as input and outputs either a message $m$ or a symbol $\bot$ indicating the failure of the decryption.

We require PKE to have perfect correctness; that is, for all possible $\text{pars} \leftarrow \text{ParGen}(1^\lambda)$ and all $m \in \mathcal{M}$, we have

$$\Pr[\text{(pk, sk)} \leftarrow \text{ParGen}(1^\lambda) : \text{Decrypt}(\text{sk}, \text{Encrypt} (\text{pk}, m)) = m] = 1. \quad (1)$$

**Definition 1** ($\text{KDM}[\mathcal{F}]-\text{CCA}$ security). Let $n \in \mathbb{N}$ and let $\mathcal{F}$ denote a set of functions from $(\mathcal{S})^n$ to $\mathcal{M}$. A scheme $\text{PKE}$ is $n$-$\text{KDM}[\mathcal{F}]-\text{CCA}$ secure, if for any PPT adversary $A$, we have $\text{Adv}_{\text{KDM-CCA}}^\lambda = |\Pr[\text{PKE},\mathcal{F},\mathcal{M}] = 1| - 1/2 \leq \text{negl}(\lambda)$, where $n$-$\text{KDM}[\mathcal{F}]-\text{CCA}$ is the security game shown in Figure 2.

**2.2. Authenticated Encryption.** There are three PPT algorithms $\text{AE} = (\text{AE.ParGen}, \text{AE.Encrypt}, \text{AE.Decrypt})$ in an authenticated encryption (AE) scheme:

(i) $\text{AE.ParGen}(1^\lambda)$ generates a system parameter $\text{pars}_{\text{AE}}$. We require $\text{pars}_{\text{AE}}$ to be an implicit input to other algorithms and assume that $\text{pars}_{\text{AE}}$ implicitly defines a key space $\mathcal{K}_{\text{AE}}$ and a message space $\mathcal{M}$.

(ii) $\text{AE.Encrypt}(k, m)$ takes a key $k \in \mathcal{K}_{\text{AE}}$ and a message $m \in \mathcal{M}$ as input and outputs a ciphertext $\text{ae}$.

(iii) $\text{AE.Decrypt}(k, \text{ae})$ takes a key $k \in \mathcal{K}_{\text{AE}}$ and a ciphertext $\text{ae}$ as input and outputs a message $m \in \mathcal{M}$ or a symbol $\bot$.

We require $\text{AE}$ to have perfect correctness; that is, for all possible $\text{pars}_{\text{AE}} \leftarrow \text{AE.ParGen}(1^\lambda)$, all keys $k \in \mathcal{K}_{\text{AE}}$, and all $m \in \mathcal{M}$,

$$\Pr[\text{AE.Decrypt}(k, \text{AE.Encrypt}(k, m)) = m] = 1. \quad (2)$$

**Definition 2** (one-time security). A scheme $\text{AE}$ is one-time secure ($\text{OT}$-secure), that is, IND-OT and INT-OT secure, if for any PPT adversary $A$, both $\text{Adv}_{\text{OT-secure}}^\lambda = |\Pr[\text{IND-OT-secure}] = 1| - 1/2 \leq \text{negl}(\lambda)$ and $\text{Adv}_{\text{INT-OT-secure}}^\lambda = |\Pr[\text{IND-OT-secure}] = 1| \leq \text{negl}(\lambda)$, where IND-OT and INT-OT are the security games presented in Figure 3.

**2.3. Key Encapsulation Mechanism.** There are three PPT algorithms $\text{KEM} = (\text{KEM.KeyGen}, \text{KEM.Encrypt}, \text{KEM.Decrypt})$ in a key encapsulation mechanism (KEM):

(i) $\text{KEM.KeyGen}(1^\lambda)$ generates a public key $\text{pk}$ and a secret key $\text{sk}$.

(ii) $\text{KEM.Encrypt}(\text{pk})$ takes $\text{pk}$ as input and outputs a key $k$ together with a ciphertext $\text{ke}$.

(iii) $\text{KEM.Decrypt}(\text{sk}, \text{ke})$ takes $\text{sk}$ and a ciphertext $\text{ke}$ as input and outputs either a key $k$ or a symbol $\bot$. 

![Figure 2: $n$-KDM$[\mathcal{F}]$-CCA security game.](image-url)
2.4. Tag-Based Hash Proof System: Universal

Comprised of three PPT algorithms: THPS (tag-based hash proof system). A tag-based hash property is slightly different. THPS requires (6) to hold for all \( \mathcal{H} \mathcal{P} \mathcal{S} \) lies in the definition of the universal property [19]. The key difference between tag-based HPS and extended HPS lies in the definition of the universal property [20], but the universal_2 property is slightly different.

**Definition 3** (tag-based hash proof system). A tag-based hash proof system \( \text{THPS} = (\text{THPS.Setup}, \text{THPS.Pub}, \text{THPS.Priv}) \) is comprised of three PPT algorithms:

(i) \( \text{THPS.Setup}(1^k) \) outputs a parameterized instance \( \text{pars}_{\text{THPS}} \), which implicitly defines \( (\mathcal{H}, \mathcal{C}, \mathcal{V}, \mathcal{I}, \mathcal{K}, \mathcal{P}, \Lambda_{(\cdot)}, \mu) \), where \( \mathcal{X} \subseteq \mathcal{C}, \Lambda_{(\cdot)} : \mathcal{C} \times \mathcal{I} \rightarrow \mathcal{X} \) is a set of hash functions indexed by \( h_k \in \mathcal{H} \), and \( \mu : \mathcal{H} \mathcal{A} \rightarrow \mathcal{P} \mathcal{A} \) is a function. We assume that \( \mu \) is efficiently computable, and there are PPT algorithms sampling \( h_k \sim \mathcal{H} \mathcal{A} \) uniformly, sampling \( C \sim \mathcal{C} \) uniformly, sampling \( C \sim \mathcal{V} \) uniformly with a witness \( w \), and checking membership in \( \mathcal{S} \).

(ii) \( \text{THPS.Pub}(pk, C, w, t) \) takes a projection key \( pk = \mu(h_k) \in \mathcal{P} \mathcal{A} \), an element \( C \in \mathcal{V} \) with a witness \( w \), and a tag \( t \in \mathcal{I} \) as input and outputs a hash value \( K = \Lambda_{\text{hk}}(C, t) \in \mathcal{H} \).

(iii) \( \text{THPS.Priv}(hk, C, t) \) takes a hashing key \( h_k \in \mathcal{H} \mathcal{A} \), an element \( C \in \mathcal{C} \), and a tag \( t \in \mathcal{I} \) as input and outputs a hash value \( K = \Lambda_{\text{hk}}(C, t) \in \mathcal{H} \) without knowing a witness.

We require KEM to have perfect correctness; that is, for all possible \( (pk, sk) \leftarrow \mathcal{K} \mathcal{E} \mathcal{M} \mathcal{G} \mathcal{E} (1^k) \), we have

\[
\Pr[(k, \text{km} \mathcal{C}) \leftarrow \mathcal{K} \mathcal{E} \mathcal{M}.\text{Encrypt}(pk) : \text{KEM.Decrypt}(sk, km \mathcal{C}) = k] = 1.
\]

2.4. Tag-Based Hash Proof System: Universal_2, Extracting, and Key-Homomorphism. Tag-based hash proof system (HPS) was first defined in [19]. The definition is similar to extended HPS [20], but the universal_2 property is slightly different.

**Definition 4** (SMP). The Subset Membership Problem (SMP) related to THPS is hard, if for any PPT adversary \( \mathcal{A} \), one has

\[
\text{Adv}_{\text{THPS}, \mathcal{A}}^{\text{SMP}, \lambda}(\lambda) = \left| \Pr[\mathcal{A}(\text{pars}_{\text{THPS}}, C) = 1] - \Pr[\mathcal{A}(\text{pars}_{\text{THPS}}, C') = 1] \right| \leq \text{negl}(\lambda),
\]

where \( \text{pars}_{\text{THPS}} \leftarrow \mathcal{S} \mathcal{T} \mathcal{H} \mathcal{P} \mathcal{S} . \text{Setu}p(1^k) \), all \( h_k \in \mathcal{H} \mathcal{A} \) and \( pk = \mu(h_k) \in \mathcal{P} \mathcal{A} \), all \( C \in \mathcal{V} \) with all witnesses \( w \) and all \( t \in \mathcal{I} \), it holds that

\[
\text{THPS.Pub}(pk, C, w, t) = \Lambda_{\text{hk}}(C, t)
\]

\[
= \text{THPS.Priv}(hk, C, t).
\]

Tag-based HPS is associated with a subset membership problem. Informally speaking, it asks to distinguish the uniform distribution over \( \mathcal{V} \) from the uniform distribution over \( \mathcal{C} \setminus \mathcal{V} \).

We require THPS to be projective, that is, for all \( \text{pars}_{\text{THPS}} \leftarrow \mathcal{S} \mathcal{T} \mathcal{H} \mathcal{P} \mathcal{S} . \text{Setu}p(1^k) \), all \( h_k \in \mathcal{H} \mathcal{A} \) and \( pk = \mu(h_k) \in \mathcal{P} \mathcal{A} \), all \( C \in \mathcal{V} \) with all witnesses \( w \) and all \( t \in \mathcal{I} \), it holds that

\[
\text{THPS.Pub}(pk, C, w, t) = \Lambda_{\text{hk}}(C, t)
\]

\[
= \text{THPS.Priv}(hk, C, t).
\]

The key difference between tag-based HPS and extended HPS lies in the definition of the universal_2 property [19]. Extended HPS requires (6) to hold only for \( t \neq t' \). Hence,
any (universal) extended HPS is also a (universal) tag-based HPS, but not vice versa. Tag-based HPS is essentially a weaker variant of extended HPS and admits more efficient constructions.

Dodis et al. [21] defined an extracting property for extended HPS, which requires the hash value $\Lambda_{hk}(C, t)$ to be uniformly distributed over $\mathcal{H}$ for any $C \in \mathcal{C}$ and $t \in \mathcal{T}$, as long as $hk$ is randomly chosen from $\mathcal{H}$ and $t \in \mathcal{T}$. Besides, Xagawa [22] considered a key-homomorphic property for extended HPS, which stipulates that $\Lambda_{hk\Lambda}(C, t) = \Lambda_{hk}(C, t) \cdot \Lambda_{\Lambda}(C, t)$ holds for any $hk, \Lambda \in \mathcal{H}, C \in \mathcal{C}$, and $t \in \mathcal{T}$. Here we adapt these notions to tag-based HPS.

Definition 6 (extracting). THPS is called extracting, if for all $\text{pars}_{\text{THPS}} \leftarrow \$ THPS.Setups$(\lambda)$, all $C \in \mathcal{C}$, all $t \in \mathcal{T}$, and all $K \in \mathcal{K}$, it holds that

$$\Pr \left[ \Lambda_{hk}(C, t) = K \right] = \frac{1}{|\mathcal{H}|},$$

where $hk \leftarrow \$ \mathcal{H}$.

Definition 7 (key-homomorphism). THPS is called keyhomomorphic, if for all $\text{pars}_{\text{THPS}} \leftarrow \$ THPS.Setups$(\lambda)$, which defines $(\mathcal{H}, \mathcal{C}, \mathcal{T}, \mathcal{K}, \mathcal{R}, \Lambda(\cdot), \mu)$, one has the following:

(i) Both $(\mathcal{H}, \mathcal{C}, +)$ and $(\mathcal{H}, \cdot)$ are groups.

(ii) For all $C \in \mathcal{C}$ and all $t \in \mathcal{T}$, the mapping $\Lambda(\cdot): \mathcal{H} \rightarrow \mathcal{H}$ is a group homomorphism. That is, for all $hk, \Lambda, C \in \mathcal{H}$ and $a \in \mathbb{Z}$, it holds that $\Lambda_{a}h_{k} = (\Lambda_{a}h)_{k}$. We define the following:

(i) $\mathcal{Q}(\mathcal{R}) = \{a \mod N \mid a \in \mathbb{Z}_{N}\}$.

Then $\mathcal{Q}(\mathcal{R})$ is a cyclic group of order $N$. For $s \in \mathbb{N}$ and $T = 1 + N$, we define

(i) $\mathcal{Q}(\mathcal{R})s = \{a \mod N^s \mid a \in \mathbb{Z}_{N^s}\}$,

(ii) $\mathcal{S}(\mathcal{C}) = \{a \mod N^s \mid a \in \mathbb{Z}_{N^s}\}$,

(iii) $\mathcal{R}(\mathcal{U}) = \{r \mod N^s \mid r \in [N^s - 1]\}$.

Then $\mathcal{S}(\mathcal{C})$ is a cyclic group of order $\phi(N)/4$, and $\mathcal{Q}(\mathcal{R}) = \mathcal{S}(\mathcal{C}) \otimes \mathcal{R}(\mathcal{U})$, where $\otimes$ represents the external direct product.

Dangdang and Jurik [23] showed that the discrete logarithm $d\log_T(u) \in [N^{-1}]$ of an element $u \in \mathcal{R}(\mathcal{U})$, can be efficiently computed from $u$ and $N$. Observe that $\mathbb{Z}_N = \mathbb{Z}_2 \otimes \mathbb{Z}_2 \otimes \mathcal{S}(\mathcal{C}) \otimes \mathcal{R}(\mathcal{U})$; thus for any $v = \nu(\mathbb{Z}_2) \cdot \nu(\mathbb{Z}_2) \cdot \nu(\mathcal{S}(\mathcal{C})) \cdot T^\nu \in \mathbb{Z}_N$, we have $\nu(\phi(N)) = T^\nu \nu(\phi(N)) \in \mathcal{R}(\mathcal{U})$, and

$$d \log_T(\phi(N)) \mod N^s + 1 = x.$$  

Definition 8 (DCR assumption). The Decisional Composite Residuosity (DCR) assumption holds for $\text{GenN}$ and $\mathcal{Q}(\mathcal{R})$, if for any PPT $\mathcal{A}$, it holds that

$$\text{Adv}_{\text{GenN},\phi}(\lambda) = \frac{\left| \Pr \left[ \mathcal{A}(N, u) = 1 \right] - \Pr \left[ \mathcal{A}(N, v) = 1 \right] \right|}{\text{negl}(\lambda)},$$

where $(p, q, N, \overline{N}) \leftarrow \$ \text{GenN}(\lambda)$, $u \leftarrow \$ \mathcal{Q}(\mathcal{R})$, and $v \leftarrow \$ \mathcal{S}(\mathcal{C})$.

The Interactive Vector (IV$_d$) assumption is implied by the DCR assumption, as shown in [5]. Here we recall the IV$_d$ assumption according to [16].

Definition 9 (IV$_d$ assumption). The IV$_d$ assumption holds for $\text{GenN}$ and $\mathcal{Q}(\mathcal{R})$, if for any PPT $\mathcal{A}$, it holds that

$$\text{Adv}_{\text{GenN},d}(\lambda) = \frac{\left| \Pr \left[ \mathcal{A}(\lambda) = 1 \right] - \Pr \left[ \mathcal{A}(\lambda) = 1 \right] \right|}{\text{negl}(\lambda)},$$

where $(p, q, N, \overline{N}) \leftarrow \$ \text{GenN}(\lambda)$, $g_1, \ldots, g_d \leftarrow \$ \mathcal{S}(\mathcal{C})$, $b \leftarrow \$ \{0, 1\}$, and $\mathcal{A}$ is allowed to query the oracle $\text{chal}_{\text{IV}_d}^b(\cdot)$ adaptively. Each time, $\mathcal{A}$ can submit $(\delta_1, \ldots, \delta_d)$ to the oracle, and $\text{chal}_{\text{IV}_d}^{b}(\delta_1, \ldots, \delta_d)$ selects $r \leftarrow \$ \{0, 1\}$ randomly: if $b = 0$, the oracle outputs $(g_1^{\delta_1}, \ldots, g_d^{\delta_d})$ to $\mathcal{A}$; otherwise it outputs $(g_1^{r\delta_1}, \ldots, g_d^{r\delta_d})$ to $\mathcal{A}$, where $T = 1 + N$.

Definition 10 (DDH assumption). The DDH assumption holds for $\text{GenN}$ and $\mathcal{Q}(\mathcal{R})$, if for any PPT $\mathcal{A}$, it holds that

$$\text{Adv}_{\text{GenN},d}(\lambda) = \frac{\left| \Pr \left[ \mathcal{A}(\lambda) = 1 \right] - \Pr \left[ \mathcal{A}(\lambda) = 1 \right] \right|}{\text{negl}(\lambda)},$$

where $(p, q, N, \overline{N}) \leftarrow \$ \text{GenN}(\lambda)$, $g_1, g_2 \leftarrow \$ \mathcal{S}(\mathcal{C})$, $x, y \leftarrow \$ \mathbb{Z}_N \setminus \{0\}$.

Definition 11 (DL assumption). The Discrete Logarithm (DL) assumption holds for $\text{GenN}$ and $\mathcal{S}(\mathcal{C})$, if for any PPT $\mathcal{A}$, it holds that

$$\text{Adv}_{\text{GenN},d}(\lambda) = \frac{\left| \Pr \left[ \mathcal{A}(\lambda) = 1 \right] - \Pr \left[ \mathcal{A}(\lambda) = 1 \right] \right|}{\text{negl}(\lambda)},$$

where $(p, q, N, \overline{N}) \leftarrow \$ \text{GenN}(\lambda)$, $g \leftarrow \$ \mathcal{S}(\mathcal{C})$, $x \leftarrow \$ \mathbb{Z}_N \setminus \{0\}$.
2.6. Collision-Resistant Hashing

Definition 12 (collision-resistant hashing). Let $\mathcal{H} = \{H : \mathcal{X} \rightarrow \mathcal{Y}\}$ be a set of hash functions. $\mathcal{H}$ is said to be collision-resistant, if for any PPT $\mathcal{A}$, one has

$$\text{Adv}_{\mathcal{H},\mathcal{A}}^\text{cr} (\lambda) = \Pr \left[ H \leftarrow \mathcal{H}, (x, x') \leftarrow \mathcal{A} (H) : x \neq x' \land H(x) = H(x') \right] \leq \text{negl}(\lambda). \quad (13)$$

3. Auxiliary-Input Authenticated Encryption

Our PKE constructions in Sections 4 and 5 will resort to a new primitive AIAE. To serve the KDM-CCA security of our PKE construction in Figure 1, our AIAE should satisfy the following properties.

(i) AIAE must take an auxiliary input $a_i$ in both the encryption and decryption algorithms.

(ii) AIAE must have IND-\text{F-RKA} and weak-\text{INT-\text{F-RKA}} security. Compared to the \text{INT-\text{F-RKA}} security proposed in [16], the weak-\text{INT-\text{F-RKA}} security imposes a special rule to determine whether the adversary’s forgery is successful or not.

In the following, we present the syntax of AIAE and define its IND-\text{F-RKA} Security and Weak-\text{INT-\text{F-RKA}} Security. We also show a general paradigm of AIAE from tag-based HPS and give an instantiation of AIAE under the DDH assumption.

3.1. Auxiliary-Input Authenticated Encryption

Definition 13 (AIAE). There are three PPT algorithms $\text{AIAE} = (\text{AIAE.ParGen}, \text{AIAE.Encrypt}, \text{AIAE.Decrypt})$ in an AIAE scheme:

(i) The parameter generation algorithm $\text{AIAE.ParGen}(1^\lambda)$ generates a system parameter $\text{pars}_{\text{AIAE}}$. We require $\text{pars}_{\text{AIAE}}$ to be an implicit input to other algorithms and assume that $\text{pars}_{\text{AIAE}}$ implicitly defines a key space $\mathcal{K}_{\text{AIAE}}$, a message space $\mathcal{M}$, and an auxiliary-input space $\mathcal{F}$.\!

(ii) The encryption algorithm $\text{AIAE.Encrypt}(k,m,a_i)$ takes a key $k \in \mathcal{K}_{\text{AIAE}}$, a message $m \in \mathcal{M}$, and an auxiliary input $a_i \in \mathcal{F}$ as input and outputs a ciphertext $\text{aiae}_c$.

(iii) The decryption algorithm $\text{AIAE.Decrypt}(k,\text{aiae}_c,a_i)$ takes a key $k \in \mathcal{K}_{\text{AIAE}}$, a ciphertext $\text{aiae}_c$, and an auxiliary input $a_i \in \mathcal{F}$ as input and outputs a message $m \in \mathcal{M}$ or a symbol $\bot$.

We require AIAE to have perfect correctness; that is, for all possible $\text{pars}_{\text{AIAE}} \leftarrow \text{AIAE.ParGen}(1^\lambda)$, all keys $k \in \mathcal{K}_{\text{AIAE}}$, all messages $m \in \mathcal{M}$, and all auxiliary-inputs $a_i \in \mathcal{F},$

$$\Pr \left[ \text{AIAE.Decrypt} (k, \text{AIAE.Encrypt} (k, m, a_i), a_i) = m \right] = 1. \quad (14)$$

In fact, AIAE is a generalization of traditional AE, and traditional AE can be viewed as AIAE with $\mathcal{F} = \emptyset$.

Definition 14 (RKA security). Denote by $\mathcal{F}$ a set of functions from $\mathcal{K}_{\text{AIAE}}$ to $\mathcal{K}_{\text{AIAE}}$. A scheme AIAE is IND-\text{F-RKA} secure and weak-\text{INT-\text{F-RKA}} secure, if for any PPT $\mathcal{A},$

$$\text{Adv}^\text{ind-rka}_{\text{AIAE},\mathcal{A}} (\lambda) = \Pr \left[ \text{IND-\text{F-RKA}}^\mathcal{F} \Rightarrow 1 \right] - \frac{1}{2} \leq \text{negl}(\lambda),$$

$$\text{Adv}^\text{weak-int-rka}_{\text{AIAE},\mathcal{A}} (\lambda) = \Pr \left[ \text{weak-\text{INT-\text{F-RKA}}}^\mathcal{F} \Rightarrow 1 \right] \leq \text{negl}(\lambda),$$

where IND-\text{F-RKA} and weak-\text{INT-\text{F-RKA}} are the security games presented in Figure 4.

3.2. Generic Construction of AIAE from Tag-Based HPS and OT-Secure AE. Our construction of AIAE needs the following ingredients.

(i) A tag-based hash proof system THPS = (THPS.Setup, THPS.Pub, THPS.Priv), where the hash value space is $\mathcal{H}$, the tag space is $\mathcal{T}$, and the hashing key space is $\mathcal{K}$.\!

(ii) A (traditional) authenticated encryption scheme $\text{AE} = (\text{AE.ParGen}, \text{AE.Encrypt}, \text{AE.Decrypt})$, where the message space is $\mathcal{M}$ and the key space is $\mathcal{K}$.\!

(iii) A set of hash functions $\mathcal{H} = \{H : \{0, 1\}^* \rightarrow \mathcal{T}\}$.\!

We present our AIAE construction $\text{AIAE} = (\text{AIAE.ParGen}, \text{AIAE.Encrypt}, \text{AIAE.Decrypt})$ in Figure 5, whose key space is $\mathcal{K}_{\text{AIAE}} = \mathcal{K}$, message space is $\mathcal{M}$, and auxiliary-input space is $\mathcal{F} = \{0, 1\}^*$.\!

By the perfect correctness of AE, it is routine to check that AIAE has perfect correctness.

Theorem 15. If (i) THPS is universal, extracting, key-homomorphic and has a hard subset membership problem, (ii) AE is one-time secure, and (iii) $\mathcal{H}$ is collision-resistant, then the scheme AIAE in Figure 5 is IND-\text{F-raff}-RKA secure and weak-\text{INT-\text{F-raff}}-RKA secure. Here $\mathcal{F}_{\text{raff}} = \{f_{(a,b)} : h \in \mathcal{H} \mapsto a \cdot h + b \in \mathcal{H} \mid a \in \mathbb{Z}^*_\mathcal{H}, b \in \mathcal{H}\}$ is the set of restricted affine functions.

Proof of Theorem 15 (IND-\text{F-raff}-RKA Security). Denote by $\mathcal{A}$ a PPT adversary who is against the IND-\text{F-raff}-RKA security and queries \text{ENCRYPT} oracle for at most $Q_\mathcal{E}$ times. We show the IND-\text{F-raff}-RKA security through a series of games. For an event $E$, we denote by $\text{Pr}_E[H], \text{Pr}_E[E]$, and $\text{Pr}_E[E]$ the probability of E occurring in games $G_j, G_j',$ and $G''$, respectively.

Game $G_j$. It is the original IND-\text{F-raff}-RKA game. Denote the event $\beta' = \beta$ by Succ. According to the definition, $\text{Adv}^\text{ind-rka}_{\text{AIAE},\mathcal{A}} (\lambda) = |\text{Pr}_E[\text{Succ}] - 1/2|$.
As for the $\ell$th ($\ell \in [Q_1]$) ENCRYPT query ($m_{\ell,0}, m_{\ell,1}, a_{\ell,0}, f_{\ell}$), where $f_{\ell} = \langle a_{\ell}, b_{\ell} \rangle \in \mathcal{X}_{\text{ralf}}$, the challenger prepares the challenge ciphertext as follows:

(i) pick $C_{\ell} \leftarrow \mathcal{X}$ together with witness $w_{\ell}$,

(ii) compute $t_{\ell} = H(C_{\ell}, a_{\ell}) \in \mathcal{X}$,

(iii) compute $\kappa_{\ell} = \Lambda_{a_{\ell} \cdot \chi + b_{\ell}}(C_{\ell}, t_{\ell}) \in \mathcal{X}$,

(iv) invoke $\chi_{\ell} \leftarrow \text{AE.Encrypt}(\kappa_{\ell}, m_{\ell,0})$,

and it outputs the challenge ciphertext $\langle C_{\ell}, \chi_{\ell} \rangle$ to $\mathcal{F}$.

**Game $G_{1,j}$** $j \in [Q_1] + 1$. It is identical to $G_1$ except that, for the first $j-1$ times of ENCRYPT queries, that is, $\ell \in [j-1]$, the challenger chooses $\kappa_{\ell} \leftarrow \mathcal{X}$ randomly for the AE scheme.

Clearly, $G_{1,j}$ is identical to $G_1$; thus $\text{Pr}_{1}[\text{Succ}] = \text{Pr}_{1,j}[\text{Succ}]$.

**Game $G'_{1,j}$** $j \in [Q_1]$. It is identical to $G_{1,j}$ except that, for the $j$th ENCRYPT query, the challenger samples $C_j \leftarrow \mathcal{E} \setminus \mathcal{Y}$ uniformly.

The difference between $G_{1,j}$ and $G'_{1,j}$ lies in the distribution of $C_j$. In game $G_{1,j}$, $C_j$ is uniformly chosen from $\mathcal{Y}$; in game $G'_{1,j}$, $C_j$ is uniformly chosen from $\mathcal{E} \setminus \mathcal{Y}$. Any difference between $G_{1,j}$ and $G'_{1,j}$ results in a PPT adversary solving the subset membership problem related to THPS; thus we have that $|\text{Pr}_{1,j}[\text{Succ}] - \text{Pr}_{1,j'}[\text{Succ}]| \leq \text{Adv}_{\text{THPS}}(\lambda)$.

**Game $G''_{1,j}$** $j \in [Q_1]$. It is identical to $G'_{1,j}$ except that, for the $j$th ENCRYPT query, the challenger chooses $\kappa_j \leftarrow \mathcal{X}$ randomly.

**Lemma 16.** For all $j \in [Q_1]$, $\text{Pr}_{1,j}[\text{Succ}] = \text{Pr}_{1,j'}[\text{Succ}]$.

**Proof.** For game $G'_{1,j}$ and game $G''_{1,j}$, the difference between them lies in the computation of $\kappa_j$ in the $j$th ENCRYPT query. In $G'_{1,j}$, $\kappa_j$ is properly computed, while in $G''_{1,j}$, it is chosen from $\mathcal{X}$ uniformly.

We analyze the information about the key $hk$ that is used in game $G''_{1,j}$.

(i) For the $\ell$th ($\ell \in [j-1]$) query, ENCRYPT does not use $hk$ at all since $\kappa_{\ell}$ is randomly chosen from $\mathcal{X}$.
(ii) For the $\ell$th ($\ell \in [j + 1, Q_\ell]$) query, ENCRYPT can use $pk = \mu(hk)$ to compute $\kappa_\ell$:

$$
\kappa_\ell = \Lambda_{a_\ell \cdot bk + b_\ell} (C_\ell, t_\ell): \quad C_\ell \leftarrow_s \mathcal{V} \quad \text{with witness } w_\ell \\
= (\Lambda_{hk} (C_\ell, t_\ell))^{a_\ell} \cdot \Lambda_{b_\ell} (C_\ell, t_\ell):
$$

via key-homomorphism (16)

$$
= (\text{THPS Pub} (pk, C_\ell, w_\ell, t_\ell))^{a_\ell} \cdot \Lambda_{b_\ell} (C_\ell, t_\ell):
$$

via projective property.

(iii) For the $j$th query, ENCRYPT uses $\Lambda_{hk}(C_j, t_j)$ to compute $\kappa_j$:

$$
\kappa_j = \Lambda_{a_j \cdot bk + b_j} (C_j, t_j):
$$

$$
C_j \leftarrow_s \mathcal{G} \setminus \mathcal{V}
$$

via key-homomorphism.

where $C_j \in \mathcal{G} \setminus \mathcal{V}$, by the universal property of THPS, $\Lambda_{hk}(C_j, t_j)$ is uniformly distributed over $\mathcal{H}$ conditioned on $pk = \mu(hk)$. Then $a_j \in \mathbb{Z}_r^*$, $\kappa_j = (\Lambda_{hk}(C_j, t_j))^{a_j} \cdot \Lambda_{b_j} (C_j, t_j)$ is also randomly distributed over $\mathcal{H}$. Consequently, $G'_{j,\ell}$ is essentially the same as $G''_{j,\ell}$ and $\text{Pr}_{1,j,\ell}[\text{Succ}] = \text{Pr}_{1,j,\ell}[\text{Succ}]$. \qed

Now, we show that game $G'_{j,\ell}$ is computationally indistinguishable from game $G_{j+1,\ell}$, $j \in [Q_\ell]$. Note that the divergence between $G''_{j,\ell}$ and $G_{j+1,\ell}$ lies in the distribution of $C_j$ in the $j$th ENCRYPT query. In game $G'_{j,\ell}$, $C_j$ is uniformly chosen from $\mathcal{G} \setminus \mathcal{V}$; in game $G_{j+1,\ell}$, $C_j$ is uniformly chosen from $\mathcal{V}$. Any difference between these two games results in a PPT adversary solving the subset membership problem related to THPS; thus we have that $|\text{Pr}_{1,j,\ell}[\text{Succ}] - \text{Pr}_{1,j,\ell}[\text{Succ}]| \leq \text{Adv}_{\text{impTHPS}}^{\text{AE.Encrypt}}(\lambda)$.

**Game $G_2$.** It is identical to $G_0$, except that when answering ENCRYPT queries, the challenger invokes $\chi_\ell \leftarrow_s \mathcal{AE.Encrypt}(\kappa_\ell, 0^{m_{\text{enc}}})$.

In game $G_{1,Q+1}$, the challenger computes $\chi_\ell \leftarrow_s \mathcal{AE.Encrypt}(\kappa_\ell, m_\ell |_b)$; in game $G_2$, the challenger computes $\chi_\ell \leftarrow_s \mathcal{AE.Encrypt}(\kappa_\ell, 0^{m_{\text{enc}}})$. Since each $\kappa_\ell$ is chosen from $\mathcal{H}$ uniformly at random, $\ell \in [Q_\ell]$, by a standard hybrid argument, any difference between $G_{1,Q+1}$ and $G_2$ results in a PPT adversary against the IND-OT security of $\mathcal{AE}$, so that $|\text{Pr}_{1,Q+1}[\text{Succ}] - \text{Pr}_2[\text{Succ}]| \leq \text{Adv}^{\text{AE.Decrypt}}_{\text{AE}}(\lambda)$.

Finally, in game $G_3$, since the challenge ciphertexts are encryptions of $0^{m_{\text{enc}}}$, hence $\beta$ is perfectly hidden to $\mathcal{A}$. So $\text{Pr}_2[\text{Succ}] = 1/2$.

Summing up, we proved the IND-\(\mathcal{F}_{\text{ralf}}\)-RKA security.

This completes the proof of Theorem 15 (IND-\(\mathcal{F}_{\text{ralf}}\)-RKA security).

**Proof of Theorem 15 (Weak-INT-\(\mathcal{F}_{\text{ralf}}\)-RKA Security).** Denote by $\mathcal{A}$ a PPT adversary who is against the weak-INT-\(\mathcal{F}_{\text{ralf}}\)-RKA security and queries ENCRYPT oracle for at most $Q_\ell$ times. Similarly, the proof goes through a series of games, which are defined analogously, just like those games of the previous proof.

**Game $G_0$.** It is the original weak-INT-\(\mathcal{F}_{\text{ralf}}\)-RKA game.

As for the $\ell$th ($\ell \in [Q_\ell]$) ENCRYPT query $(m_\ell, a_\ell f_\ell, f_\ell)$, the challenger computes the challenge ciphertext $(C_\ell, \chi_\ell)$ in similar steps as the previous proof and outputs $(C_\ell, \chi_\ell)$ to $\mathcal{A}$. Moreover, the challenger will put $(a_\ell f_\ell, f_\ell, \chi_\ell)$ to a set $\mathcal{G}_{\mathcal{A},F,\mathcal{F}_{\text{ralf}}}$, put $(a_\ell f_\ell, f_\ell)$ to a set $\mathcal{G}_{\mathcal{A},F,\mathcal{F}_{\text{ralf}}}$, and put $(C_\ell, a_\ell f_\ell, t_\ell)$ to a set $\mathcal{G}_{\mathcal{A},F,\mathcal{F}_{\text{ralf}}}$. In the end, the adversary outputs a forgery $(a^*, f^*, (C^*, \chi^*))$, where $f^* = (a^*, b^*)$, and the challenger invokes the FINALIZE procedure as follows:

(i) If $(a^*, f^*, (C^*, \chi^*)) \in \mathcal{G}_{\mathcal{A},F,\mathcal{F}_{\text{ralf}}}$, output 0.

(ii) If $\exists(a_\ell f_\ell, f_\ell) \in \mathcal{G}_{\mathcal{A},F,\mathcal{F}_{\text{ralf}}}$ such that $a_\ell f_\ell = a^*$ but $f_\ell \neq f^*$, output 0.

(iii) If $C^* \notin \mathcal{G}$, output 0.

(iv) Compute $t^* = \text{H}(C^*, a^*) \in \mathcal{F}$ and $\lambda^* = \Lambda_{a^* \cdot bk + b_\ell}(C^*, t^*) \in \mathcal{H}$. Output $(\text{AE.Decrypt}(\kappa^*, \chi^*) \neq \bot)$.

Denote the event that FINALIZE outputs 1 by Forge. According to the definition, $\text{Adv}^{\text{weak-INT-\(\mathcal{F}_{\text{ralf}}\)-RKA}}_{\text{AE.Encrypt}}(\lambda) = \text{Pr}_0[\text{Forge}]$.

**Game $G_1$.** It is identical to $G_0$, except that the following rule is added to the procedure FINALIZE by the challenger:

(i) If $\exists(C_\ell, a_\ell f_\ell, t_\ell) \in \mathcal{G}_{\mathcal{A},F,\mathcal{F}_{\text{ralf}}}$ such that $t_\ell = t^*$ but $(C_\ell, a_\ell f_\ell) \neq (C^*, a^*)$, output 0.

Since $t_\ell = \text{H}(C_\ell, a_\ell f_\ell)$ and $t^* = \text{H}(C^*, a^*)$, any difference between $G_0$ and $G_1$ implies a hash collision of $\text{H}$. So $|\text{Pr}_0[\text{Forge}] - \text{Pr}_1[\text{Forge}]| \leq \text{Adv}^{\text{AE.Decrypt}}_{\text{AE}}(\lambda)$.

**Game $G_{j,\ell}$, $j \in [Q_\ell + 1]$.** It is identical to $G_1$, except that, for the first $j-1$ times of ENCRYPT queries, that is, $\ell \in [j-1]$, the challenger chooses $\kappa_\ell \leftarrow_s \mathcal{H}$ uniformly for the $\mathcal{AE}$ scheme.

Clearly $G_{1,1}$ is identical to $G_1$; thus $\text{Pr}_1[\text{Forge}] = \text{Pr}_{1,1}[\text{Forge}]$.

**Game $G'_{1,j,\ell}$, $j \in [Q_\ell]$.** It is identical to $G_{1,j}$, except that, for the $j$th ENCRYPT query, the challenger samples $C_j \leftarrow_s \mathcal{G} \setminus \mathcal{V}$ uniformly.

The difference between $G_{1,j}$ and $G'_{1,j}$ lies in the distribution of $C_j$. In game $G_{1,j}$, $C_j$ is uniformly chosen from $\mathcal{V}$; in game $G'_{1,j}$, $C_j$ is uniformly chosen from $\mathcal{G} \setminus \mathcal{S}$. Any difference between these two games results in a PPT adversary solving the subset membership problem related to THPS. We emphasize that the PPT adversary (simulator) is able to check the occurrence of Forge in an efficient way, because the key $hk$ can be chosen by the simulator itself. Consequently, the difference between $G_{1,j}$ and $G'_{1,j}$ can be reduced to the subset membership problem smoothly.
Lemma 17. For all $j \in [Q_\ell]$, $|Pr_{1,j}[\text{Forge}] - Pr_{1,j}[\text{Forge}]| \leq Adv^{\text{imp}}_{\text{THPS}}(\lambda)$.

Proof. To bound the difference between $G_{1,j}$ and $G'_{1,j}$, we build an efficient adversary $B$ solving the subset membership problem. Given $(\text{pars}_{\text{THPS}}, C)$, where $\text{pars}_{\text{THPS}} : \rightarrow$ THPS.Setup(1^\lambda), $B$ aims to distinguish $C \rightarrow \mathcal{Y}$ from $C \rightarrow \mathcal{Y} \setminus \mathcal{Y}'$.

$B$ simulates $G_{1,j}$ or $G'_{1,j}$ for $A$. Firstly, $B$ invokes pars_{\text{AE}} : \rightarrow \text{AE.ParGen}(1^\lambda)$, picks $H \rightarrow \mathcal{Y}$ randomly, and sends $\text{pars}_{\text{AE}} = (\text{pars}_{\text{THPS}}, \text{pars}_{\text{AE}}, H)$ to $A$. Next, $B$ chooses $h_k \rightarrow \mathcal{H}$.

As for the $t$th encrypt query $(m_0, a_i, f_t)$, where $f_t = (a_i, b_i) \in \mathcal{F}_{\text{ruff}}$, $B$ prepares the challenge ciphertext $(C_t, \chi_t)$ in the following way:

(i) If $\ell \in \{j-1\}$, $B$ computes $(C_{\ell}, \chi_{\ell})$ just like that in both $G_{1,j}$ and $G'_{1,j}$. That is, $B$ chooses $C_{\ell} \rightarrow \mathcal{Y}'$ with witness $u_{\ell}$, chooses $\kappa_{\ell} \rightarrow \mathcal{H}$ randomly, and invokes $\chi_{\ell} \rightarrow \mathcal{Y}$ AE.Encrypt($\kappa_{\ell}, m_{\ell}$).

(ii) If $\ell \in \{j+1\}$, $B$ computes $(C_{\ell}, \chi_{\ell})$ just like that in both $G_{1,j}$ and $G'_{1,j}$. That is, $B$ chooses $C_{\ell} \rightarrow \mathcal{Y}'$ with witness $u_{\ell}$, computes $t_{\ell} = H(C_{\ell}, a_{\ell})$ and $\kappa_{\ell} = \Lambda_{a_{\ell}, h_k+b}C_{\ell}, t_{\ell})$, and invokes $\chi_{\ell} \rightarrow \mathcal{Y}$ AE.Encrypt($\kappa_{\ell}, m_{\ell}$).

(iii) If $\ell = j$, $B$ embeds its own challenge $C \rightarrow \mathcal{Y}$ and simulates $G'_{1,j}$ in the case of $C \rightarrow \mathcal{Y}'$ and simulates $G_{1,j}$ in the case of $C \rightarrow \mathcal{Y} \setminus \mathcal{Y}'$.

Finally, $A$ sends a forgery $(a_i, b_i, (C', \chi')) \rightarrow B$, with $f^* = (a_i, b_i) \in \mathcal{F}_{\text{ruff}}$. Then $B$ decides whether $\text{FINALIZE}$ outputs 1 or not with the help of $hk$.

(i) If $\exists (a_i, b_i, (C', \chi')) \in \mathcal{G}_{\ell, k', \mathcal{Y}}$, $B$ outputs 0 (to its own challenger).

(ii) If $\ell \notin \mathcal{G}_{\ell, k', \mathcal{Y}}$ such that $a_{\ell} = a_i$ but $f_{\ell} \neq f^*$, $B$ outputs 0.

(iii) $B$ computes $t^* = H(C', a_i) \in \mathcal{Y}$.

(iv) $B$ computes $t^* = H(C', a_i) \in \mathcal{Y}$.

(v) $B$ computes $t^* = H(C', a_i) \in \mathcal{Y}$.

(vi) $B$ computes $t^* = H(C', a_i) \in \mathcal{Y}$.

With the help of $hk$, $B$ is able to perfectly simulate $\text{FINALIZE}$, just like that in both $G_{1,j}$ and $G'_{1,j}$. Moreover, $B$ outputs 1 to its own challenger if and only if the event $\text{Forge}$ occurs.

As a result, we have that $|Pr_{1,j}[\text{Forge}] - Pr_{1,j}[\text{Forge}]| \leq Adv^{\text{imp}}_{\text{THPS}}(\lambda)$.

Lemma 18. For all $j \in [Q_\ell]$, $Pr_{1,j}[\text{Forge}] \leq Pr_{1,j}[\text{Forge}] + Adv^{\text{int-or}}_{\text{AE}}(\lambda)$.

Proof. For game $G'_{1,j}$ and game $G''_{1,j}$, the difference between them lies in the computation of $\kappa_j$ in the $j$th encrypt query. In $G'_{1,j}, \kappa_j$ is properly computed; in $G''_{1,j}, \kappa_j$ is chosen from $\mathcal{H}$ uniformly.

We consider the information about the key $hk$ that is used in $G'_{1,j}$.

(i) For the $\ell$th $(\ell \in [j - 1])$ query, $\text{ENCRYPT}$ does not use $hk$ at all since $\kappa_j$ is randomly chosen from $\mathcal{H}$.

(ii) For the $\ell$th $(\ell \in [j + 1, Q_\ell])$ query, similar to the proof of Lemma 16, $\text{ENCRYPT}$ can use $pk = \mu(sk)$ to compute $\kappa_j$.

(iii) For the $j$th query, similar to the proof of Lemma 16, $\text{ENCRYPT}$ uses $\Lambda_{a}C_{j}, t_{j}$ to compute $\kappa_j$:

\[
\kappa_j = \Lambda_{a, h_k+b}C_{j}, t_{j} : C_{j} \rightarrow \mathcal{Y} \setminus \mathcal{Y}'
\]

\[
= (\Lambda_{h_k}C_{j}, t_{j})^{a_i} \cdot \Lambda_{b}C_{j}, t_{j}.
\]

via key-homomorphism.

(iv) The $\text{FINALIZE}$ procedure, which defines the event $\text{ Forge}$, uses $\Lambda_{h_k}C_{j}, t_{j}$ to compute $\kappa^*$:

\[
\kappa^* = \Lambda_{a^*, h_k+b^*}C_{*, t^*} = (\Lambda_{h_k}C_{j}, t_{j})^{a^*_i} \cdot \Lambda_{b^*}C_{j}, t_{j}.
\]

via key-homomorphism.

We divide the event $\text{ Forge}$ into the following two subevents:

(i) Subevent: $\text{ Forge} \wedge t_j \neq t^*$. Let us first consider the event $t_j \neq t^*$. We show that

\[
Pr_{1,j}[t_j \neq t^*] = Pr_{1,j}[t_j \neq t^*].
\]

By the fact that $C_{j} \in \mathcal{Y} \setminus \mathcal{Y}'$ and by the universal property of $\text{THPS}$, $\Lambda_{h_k}C_{j}, t_{j}$ is uniformly distributed over $\mathcal{H}$ conditioned on $pk = \mu(hk)$. Then as long as $a_{j} \in \mathbb{Z}^*_n$, $\kappa_j = (\Lambda_{h_k}C_{j}, t_{j})^{a_{j}} \cdot \Lambda_{b}C_{j}, t_{j}$ is also randomly distributed over $\mathcal{H}$. Hence, $G'_{1,j}$ is the same as $G''_{1,j}$ before $A$ queries $\text{FINALIZE}$, and consequently, $t_j \neq t^*$ occurs with the same probability in $G'_{1,j}$ and $G''_{1,j}$.

Next, we consider the event $\text{ Forge}$ conditioned on $t_j \neq t^*$.

We show that

\[
Pr_{1,j}'[t_j \neq t^*] = Pr_{1,j}'[t_j \neq t^*].
\]
Since $t_j \neq t^*$ and $C_j \in \mathcal{G} \setminus \mathcal{V}$, by the universality property of THPS, $\Lambda_{hk}(C_j, t_j)$ is uniformly distributed over $\mathcal{R}$ conditioned on pk = $\mu(hk)$ and $\Lambda_{hk}(C^*, t^*)$. With a similar argument, $\kappa_j$ is also randomly distributed over $\mathcal{R}$. Hence, $G_{1, j}$ is the same as $G''_{1, j}$ when $t_j \neq t^*$, and consequently, the probability that $\text{Forge}$ occurs in $G_{1, j}$ and $G''_{1, j}$ conditioned on $t_j \neq t^*$ is the same.

In conclusion, we have that

$$\Pr_{1,j'}[\text{Forge} \wedge t_j \neq t^*] \leq \Pr_{1,j''}[\text{Forge} \wedge t_j \neq t^*] \tag{22}$$

(ii) Subevent: $\text{Forge} \wedge t_j = t^*$. By the new rule added in game $G_1$, $\text{Forge}$ and $t_j = t^*$ will imply $(C_j, a_i) = (C^*, a_i^*)$. In addition, $\text{Forge}$ and $a_i = a_i^*$ will imply that $f_j = f^*$, due to the special rule in the weak-INT-$\mathcal{F}$-RKA game (see Figure 4). Then it is straightforward to check that $\Lambda_{hk}(C_j, t_j) = \Lambda_{hk}(C^*, t^*)$ and

$$\kappa_j = (\Lambda_{hk}(C_j, t_j))^y \cdot \Lambda_{b_i}(C_j, t_j) = (\Lambda_{hk}(C^*, t^*))^y \cdot \Lambda_{b_i}(C^*, t^*) = \kappa^*.$$  

Since $C_j \in \mathcal{G} \setminus \mathcal{V}$, by the universality property of THPS, $\Lambda_{hk}(C_j, t_j) = \Lambda_{hk}(C^*, t^*)$ is uniformly distributed over $\mathcal{R}$ conditioned on pk = $\mu(hk)$. Then as long as $a_i$ (which equals $a^*$) is $Z^*_{|\mathcal{X}|}$, $\kappa_j$ (which equals $\kappa^*$) is also randomly distributed over $\mathcal{R}$. Also in this subevent, $(a_i^*, f^*, C^*) = (a_i, f_j, C_j)$ implies $\chi^* = \chi_j$; thus the probability of $A.E.\text{Decrypt}(\kappa^*, \chi^*) \neq \perp$ is bounded by $\operatorname{Adv}_{\text{AE}}^{\text{INT-OT}}(\lambda)$. So we have the following claim. We present the full description of the reduction in Appendix A.

**Claim 19.** One has $\Pr_{1,j'}[\text{Forge} \wedge t_j = t^*] \leq \operatorname{Adv}_{\text{AE}}^{\text{INT-OT}}(\lambda)$.

Combining the above two subevents together, Lemma 18 follows. □

Now, we show that game $G''_{1, j}$ is computationally indistinguishable from game $G_{1, j+1}$, $j \in [Q_c]$. Note that the divergence between $G''_{1, j}$ and $G_{1, j+1}$ lies in the distribution of $C_j$ in the $j$th $\text{ENCRYPT}$ query. In game $G''_{1, j}$, $C_j$ is uniformly chosen from $\mathcal{G} \setminus \mathcal{V}$; in game $G_{1, j+1}$, $C_j$ is uniformly chosen from $\mathcal{V}$. Similar to Lemma 17, any difference between these two games results in a PPT adversary solving the subset membership problem related to THPS; thus we have that $|\Pr_{1,j''}[\text{Forge}] - \Pr_{1,j''}[\text{Forge}]| \leq \operatorname{Adv}_{\text{THPS}}^{\text{RMP}}(\lambda)$.

Finally, in game $G_{1, j+1}$, note that the challenger does not use $hk$ to compute $\kappa_j$ at all; thus $hk$ is uniformly random to $\mathcal{A}$. Consequently, in the $\text{FINALIZE}$ procedure, we have

$$\kappa^* = (\Lambda_{hk}(C^*, t^*))^y \cdot \Lambda_{b_i}(C^*, t^*).$$  

By the extracting property of THPS, $\Lambda_{hk}(C^*, t^*)$ is uniformly random over $\mathcal{R}$. Therefore, as long as $a^* \in Z^*_{|\mathcal{X}|}$, $\kappa^*$ is uniformly random over $\mathcal{R}$ as well. Hence, the probability of $\text{AE.\text{Decrypt}}(\kappa^*, \chi^*) \neq \perp$ is bounded by $\operatorname{Adv}_{\text{AE}}^{\text{INT-OT}}(\lambda)$, and we have $\Pr_{1,\mathcal{Q}_2}[\text{Forge}] \leq \operatorname{Adv}_{\text{AE}}^{\text{INT-OT}}(\lambda)$.

In all, we proved the weak-INT-$\mathcal{F}$-RKA security. This completes the proof of Theorem 15 (weak-INT-$\mathcal{F}$-RKA security).

**Remark 20.** We emphasize that the special rule in the weak-INT-$\mathcal{F}$-RKA game (cf. Figure 4) plays an essential role in proving Lemma 18. Below is the reason.

Without this special rule, the adversary is allowed to submit $f^* = (\langle a^*, b^* \rangle)$ which is different from $f_j = (a_j, b_j)$, even if $a_i^* = a_i$ holds. In this case, we cannot expect to employ the INT-OT security of the underlying AE scheme to show that the second subevent ($\text{Forge} \wedge t_j = t^*$) occurs with only a negligible probability. To demonstrate the problem clearly, suppose that the adversary $\mathcal{A}$ submits $f_j = (a_j, b_j)$ in the $j$th $\text{ENCRYPT}$ query and submits $f^* = (\langle a^*, b^* \rangle) = (a_j, b_j + \Delta)$ in the $\text{FINALIZE}$ procedure, where $\Delta$ is a constant. Then we have

$$\kappa^* = (\Lambda_{hk}(C_j, t_j))^{a_j} \cdot \Lambda_{b_j}(C_j, t_j) \cdot \Lambda(\chi_j) = (\Lambda_{hk}(C_j, t_j))^{a_j} \cdot \Lambda_{b_j}(C_j, t_j) \cdot \Lambda(\chi_j) \tag{25}$$

where the second equality follows from the key-homomorphism of THPS. Thus, $\kappa^*$ and $\kappa_j$ are closely related but may not be equal; in particular, the quotient $\kappa^*/\kappa_j = (\Lambda(\chi_j))$ is a constant.

Consequently, it is hard for us to show that the subevent $\text{Forge} \wedge t_j = t^*$ occurs with a negligible probability. The reason is as follows. To show that it is infeasible for any PPT adversary $\mathcal{A}$, who obtains $\chi_j \leftarrow s \text{AE.\text{Encrypt}}(k_j, m_j)$ in the $j$th $\text{ENCRYPT}$ query, to generate an AE-ciphertext $\chi^*$ satisfying $\text{AE.\text{Decrypt}}(\kappa^*, \chi^*) = (\Lambda(\chi_j), \Lambda(\chi^*)) \neq \perp$, it seems that INT-RKA security of $\mathcal{AE}$ is required to some extent. We definitely cannot require INT-RKA security for the underlying AE scheme, since we are constructing (weak) INT-RKA secure (AE) AE scheme $\text{AIAE}$. As a result, it is hard to prove Lemma 18 without our special rule in the weak-INT-$\mathcal{F}$-RKA game.

### 3.3 Tag-Based HPS from the DDH Assumption

Qin et al. [19] gave a construction of tag-based HPS from the d-LIN assumption. Here we construct a key-homomorphic THPS$_{\text{DDH}}$ under the DDH assumption in Figure 6. With a routine check, the projective property of THPS$_{\text{DDH}}$ follows.

**Theorem 21.** THPS$_{\text{DDH}}$ in Figure 6 is universal, extracting, and key-homomorphic. Moreover, the subset membership problem related to THPS$_{\text{DDH}}$ is hard under the DDH assumption for $\text{GenN}$ and $\text{QR}$.

**Proof of Theorem 21.**

Universal$_2$. Suppose that $C = (g_1^{u_1}, g_2^{u_2}) \in \mathcal{G}$, $C' = (g_1^{u'_1}, g_2^{u'_2}) \in \mathcal{G} \setminus \mathcal{V}$, and $t, t' \in \mathcal{T}$ with $t \neq t'$. For $hk = (k_1, k_2, k_3,$
$k_4 \leftarrow \{Z_N\}^4$, we analyze the distribution of $\Lambda_{hk}(C', t')$ conditioned on $pk = \mu(hk)$ and $\Lambda_{hk}(C, t)$.

Denote $d = d log_2 g_2 \in \mathbb{Z}_N$. Firstly $pk = \mu(hk) = (g_1^{k_1}, g_2^{k_2}, g_3^{k_3})$, which may leak the values of $k_1 + d k_2$ and $k_2 + d k_4$.

Next

$$\Lambda_{hk}(C, t) = \left(g_1^{u_1} \cdot g_2^{u_2} \cdot g_3^{u_3}\right)$$

which may further leak the value of $X$.

Similarly,

$$\Lambda_{hk}(C', t') = \left(g_1^{u_1} \cdot g_2^{u_2} \cdot g_3^{u_3}\right)$$

By the fact that $C' = (g_1^{u_1}, g_2^{u_2}) \not\in \mathcal{Y}$, we have $u_1 \neq u_2$. Then as long as $t \neq t'$, $Y$ is independent of $k_1 + d k_2, k_3 + d k_4$, and $X$, and consequently, $Y$ is uniformly distributed over $\mathbb{Z}_N$.

Therefore, conditioned on $pk = \mu(hk)$ and $\Lambda_{hk}(C, t)$, $\Lambda_{hk}(C', t') = (g_1^{u_1} \cdot g_2^{u_2} \cdot g_3^{u_3})$ is randomly distributed over $\mathcal{X} = \mathbb{Q}_N$.

Extracting. Suppose that $C = (g_1^{u_1}, g_2^{u_2}) \in \mathcal{Y}$ and $t \in \mathcal{T}$. For $hk = (k_1, k_2, k_3, k_4) \leftarrow \{Z_N\}^4$, we analyze the distribution of $\Lambda_{hk}(C, t)$.

By (26), $\Lambda_{hk}(C, t) = g_1^{X}$ with $X = (w_1 k_1 + w_2 d k_2) + t \cdot (w_1 k_3 + w_2 d k_4)$, since $C = (g_1^{u_1}, g_2^{u_2}) \in \mathcal{Y}$, we have $(w_1, w_2) \neq (0, 0)$. Then when $(k_2, k_3, k_4)$ is randomly chosen from $(Z_N)^4$, $X$ is uniformly distributed over $Z_N$. Consequently, $\Lambda_{hk}(C, t)$ is randomly distributed over $\mathcal{X} = \mathbb{Q}_N$.

Key-Homomorphism. For all $hk = (k_1, k_2, k_3, k_4) \in (Z_N)^4$, all $a \in Z$, all $b = (b_1, b_2, b_3, b_4) \in (Z_N)^4$, all $C = (c_1, c_2) \in \mathcal{Y}$, and $t \in \mathcal{T}$, we have $a \cdot h_k + b = a \cdot (k_1, k_2, k_3, k_4) + (b_1, b_2, b_3, b_4) = (a k_1 + b_1, a k_2 + b_2, a k_3 + b_3, a k_4 + b_4)$. Then it follows that

$$\Lambda_{a \cdot h_k + b}(C, t) = \left(g_1^{(a k_1 + b_1)} \cdot g_2^{(a k_2 + b_2)} \cdot g_3^{(a k_3 + b_3)} \cdot g_4^{(a k_4 + b_4)}\right)$$

Subset Membership Problem. The subset membership problem related to THPS$_{DDH}$ requires that (pars$_{THPS}$ = $(N, p, q, \overline{N}, g_1, g_2, C) = (g_1^{u_1}, g_2^{u_2})$) is computationally indistinguishable from (pars$_{THPS}$ = $(N, p, q, \overline{N}, g_1, g_2, C) = (g_1^{u_1}, g_2^{u_2})$), where $C \leftarrow \mathcal{Y}$ and $C' \leftarrow \mathcal{Y} \setminus \mathcal{Y}$. It trivially holds under the DDH assumption for $\mathbb{G}$ and $\mathbb{Q}_N$.

3.4. Instantiation: AIAE$_{DDH}$ From DDH-Based THPS$_{DDH}$ and OT-Secure AE. When plugging the THPS$_{DDH}$ (cf. Figure 6) into the paradigm in Figure 5, we immediately obtain an AIAE scheme AIAE$_{DDH}$ under the DDH assumption, as shown in Figure 7. The key space is $\mathcal{K}_{AIAE} = (Z_N)^4$.

By combining Theorem 15 with Theorem 21, we have the following corollary regarding the RKA security of AIAE$_{DDH}$.

Corollary 22. If (i) the DDH assumption holds for $\mathbb{G}$ and $\mathbb{Q}_N$, (ii) AE is one-time secure, and (iii) $\mathcal{K}$ is collision-resistant, then the scheme AIAE$_{DDH}$ in Figure 7 is IND-\text{-RKA}$\text{-}$RKA and weak-INT-\text{-RKA}$\text{-}$RKA secure. Here $\mathcal{F}_{\text{riff}} = \{f_{(a, b)} : (k_1, k_2, k_3, k_4) \in (Z_N)^4 \mapsto (a k_1 + b_1, a k_2 + b_2, a k_3 + b_3, a k_4 + b_4) \in (Z_N)^4 \}$. It trivially holds under the DDH assumption, as shown in Figure 7.

Remark 23. Our AIAE$_{DDH}$ enjoys the following property: $\kappa = c_1^{k_1 + k_4} \cdot c_2^{k_2 + k_3}$ will be randomly distributed over $QR_{\mathbb{G}}$, as long as any element $k_j$ in $k = (k_1, k_2, k_3, k_4)$ is uniformly chosen. As a result, the one-time security of AE will guarantee that AIAE Decrypt $(k, a i e, c, a i)$ holds for any $(a i e, c, a i)$ weak-int-RKA$(\lambda)$. This

Figure 6: Construction of THPS$_{DDH}$. 
fact will be used in the security proof of the PKE schemes presented in Sections 4 and 5.

4. PKE with \( n \)-KDM[\( F_{\text{aff}} \)]-CCA Security

Denote by \( \text{AIAE}_{\text{DDH}} = (\text{AIAE},\text{ParGen},\text{AIAE.Encrypt}, \text{AIAE.Decrypt}) \) the DDH-based AIAE scheme in Figure 7, where the key space is \((\mathbb{Z}_N)^4\). We need two other building blocks, following the approach in Figure 1.

- **KEM**: to be compatible with this \( \text{AIAE}_{\text{DDH}} \), we have to design a \( \text{KEM} \) encapsulating a key tuple \((k_1, k_2, k_3, k_4)\) in \((\mathbb{Z}_N)^4\).

- \( \mathcal{B} \): to support the set \( F_{\text{aff}} \) of affine functions, we have to construct a special public-key encryption \( \mathcal{B} \), so that after a computationally indistinguishable change, \( \mathcal{B}.\text{Encrypt} \) can serve as an entropy filter for the affine function set.

The proposed PKE scheme \( \text{PKE} = (\text{ParGen},\text{KeyGen},\text{Encrypt},\text{Decrypt}) \) is presented in Figure 8, in which the shadowed parts highlight algorithms of KEM and \( \mathcal{B} \).

The correctness of PKE is guaranteed by the correctness of \( \text{AIAE}_{\text{DDH}}, \mathcal{B} \), and KEM.

**Theorem 24.** If (i) the DCR assumption holds for \( \text{GenN} \) and \( QR_{\mathbb{N}} \), (ii) \( \text{AIAE}_{\text{DDH}} \) is \( \text{IND}-F_{\text{aff}}-\text{RKA} \) and weak-\( \text{INT}-F_{\text{aff}}-\text{RKA} \)-secure, and (iii) the DL assumption holds for \( \text{GenN} \) and \( \text{SCRN}_{\mathbb{N}} \), then the proposed scheme PKE in Figure 8 is \( n \)-KDM[\( F_{\text{aff}} \)]-CCA secure.

**Proof of Theorem 24.** Denote by \( \mathcal{A} \) a PPT adversary who is against the \( n \)-KDM[\( F_{\text{aff}} \)]-CCA security, querying \( \text{Encrypt} \) oracle for at most \( Q_e \) times and \( \text{Decrypt} \) oracle for at most \( Q_d \) times. The theorem is proved through a series of games. A rough description of differences between adjacent games is summarized in Table 2.

In the proof, \( G_1-G_3 \) deals with the \( n \)-user case; \( G_3-G_4 \) is used to eliminate the utilization of the (mod \( N \)) part of \((x_j, y_j)\) in the \( \text{Encrypt} \) oracle; the aim of \( G_5-G_6 \) is to use \((x_j, y_j) \mod N \) to hide a base key \( k^* = (k_1^*, \ldots, k_4^*) \) of \( \text{AIAE}_{\text{DDH}} \) in the \( \text{Encrypt} \) oracle; \( G_7-G_8 \) is used to eliminate the utilization of \((x_j, y_j) \mod N \) in the \( \text{Decrypt} \) oracle; in \( G_9-G_{10} \), the \( \text{IND}-F_{\text{aff}}-\text{RKA} \) security of \( \text{AIAE}_{\text{DDH}} \) leads to the \( n \)-KDM[\( F_{\text{aff}} \)]-CCA security, because \( k^* = (k_1^*, \ldots, k_4^*) \) now is concealed by \((x_j, y_j) \mod N \). perfectly.

**Game \( G_3 \).** It is the \( n \)-KDM[\( F_{\text{aff}} \)]-CCA game. Denote the event \( \beta = \beta \) by Succ. According to the definition, \( \text{Adv}^{\text{kdmc}_{\text{aff}}}_{\text{PKE,aff}}(\lambda) = |\text{Pr}_{\mathcal{P}_{\text{E,aff}}}[\text{Succ}] - 1/2| \).

For the \( i \)th user, \( i \in [n] \), let \( pk_i = (h_{i,1}, \ldots, h_{i,4}) \) and \( sk_i = (x_{i,1}, y_{i,1}, \ldots, x_{i,4}, y_{i,4}) \) denote the corresponding public key and secret key, respectively.

**Game \( G_1 \).** It is identical to \( G_0 \), except the way of answering the \( \text{Decrypt} \) query \((\alpha_i, a_{i,a,e}, c_i, \ell) \) (for any \( \ell \in [Q_e] \)), where \( a_{i,a,e}, c_i \) is the challenge ciphertext of the \( \ell \)th \( \text{Encrypt} \) oracle query \((f_{\ell}, i_{\ell}) \).

- **Case 1** \((\alpha_i, a_{i,a,e}, c_i, \ell) = (\alpha_{i,a,e}, c_i, \ell) \). \( \text{Decrypt} \) will output \( \bot \) in \( G_0 \) since \((\alpha_{i,a,e}, c_i, \ell) \in \mathcal{G}_{F_{\text{aff}}} \) is prohibited by \( \text{Decrypt} \).

- **Case 2** \((\alpha_i, a_{i,a,e}, c_i) = (\alpha_{i,a,e}, c_i) \) but \( i \neq i_{\ell} \). We show that, in \( G_0 \), \( \text{Decrypt} \) will output \( \bot \), due to \( e_{i,e_1} u_{\ell,1} u_{\ell,2} \notin \mathbb{R} \cup \mathbb{N}_e \), with overwhelming probability. Recall that \( u_{\ell,1} = g_1^{e_{\ell,1}}, u_{\ell,2} = g_2^{e_{\ell,2}}, e_{\ell,1} = h_{i_{\ell,1}}^{k_{i_{\ell,1}}} \).

\[
\begin{align*}
e_{i,e_1} u_{\ell,1} u_{\ell,2} &= h_{i_{\ell,1}}^{k_{i_{\ell,1}}} \cdot (g_1^{e_{\ell,1}})^{x_{i,1}} \cdot (g_2^{e_{\ell,2}})^{y_{i,1}} \quad (29) \\
&= (h_{i_{\ell,1}} h_{i_{\ell,1}})^{x_{i,1}} T^{k_{i_{\ell,1}}} \mod N^2,
\end{align*}
\]

where \( h_{i_{\ell,1}} \) and \( h_{i,1} \) are parts of public keys of \( i_{\ell} \)th user and \( i \)th user, respectively, and are uniformly random over \( \mathcal{S}_{\mathbb{C}} \).
So $h_i, h_{i,1}^{-1} \neq 1$; hence $e_{r_2}u_{r_2}^i u_{r_2}^{i,3} \not\in \mathbb{R} \cup N$, except with negligible probability $2^{-Ω(\lambda)}$.

Thus $G_0$ and $G_1$ are the same except with probability at most $Q_d \cdot 2^{-Ω(\lambda)}$ according to the union bound, and $\Pr[0|\text{Succ}] - \Pr[1|\text{Succ}] \leq Q_d \cdot 2^{-Ω(\lambda)}$.

**Game $G_2$.** It is identical to $G_1$, except the way the challenger samples the secret keys $s_i = (x_{i,1}, y_{i,1}, \ldots, x_{i,4}, y_{i,4})$, $i \in [n]$. In game $G_2$, the challenger first chooses $(x_1, y_1, \ldots, x_4, y_4)$ and $(\bar{x}_i, \bar{y}_i, \ldots, \bar{x}_i, \bar{y}_i)$ randomly from $\lfloor N^2/4 \rfloor$; next it computes $(x_{i,1}, y_{i,1}, \ldots, x_{i,4}, y_{i,4}) = (x_1, y_1, \ldots, x_4, y_4) + (\bar{x}_i, \bar{y}_i, \ldots, \bar{x}_i, \bar{y}_i)$ mod $\lfloor N^2/4 \rfloor$ for $i \in [n]$. Obviously, the secret keys $s_i$ are uniformly distributed. Hence $G_2$ is identical to $G_1$, and $\Pr[0|\text{Succ}] = \Pr[1|\text{Succ}]$.

**Game $G_3$.** It is identical to $G_1$, except the way the challenger responds to the $\ell$th ($\ell \in [Q_d]$) ENCRYPT query $(f, i)$. In game $G_3$, instead of using the public key $pk_{\ell} = (h_{\ell,1}, \ldots, h_{\ell,4})$, the challenger uses the secret key $sk_{\ell} = (y_{\ell,1}, y_{\ell,1}, \ldots, y_{\ell,4}, y_{\ell,4})$ to prepare $(\epsilon_{\ell,1}, \ldots, \epsilon_{\ell,4})$ and $\tilde{e}_\ell$ in the following way:

\[
\begin{align*}
\epsilon_{\ell,1}, \ldots, \epsilon_{\ell,4} & \leftarrow (u_{\ell,1}^{-x_{\ell,1}} u_{\ell,2}, \ldots, u_{\ell,4}^{-x_{\ell,4}} u_{\ell,5})^{y_{\ell,4}} T_{k_{\ell,4}} & (30) \\
\tilde{e}_\ell & \leftarrow (h_{\ell,1}^{-x_{\ell,1}} u_{\ell,2}^{-y_{\ell,2}} u_{\ell,3}^{y_{\ell,3}}, \ldots, u_{\ell,4}^{-x_{\ell,4}} u_{\ell,5}^{y_{\ell,5}}) T_{k_{\ell,4}}^m & (31)
\end{align*}
\]

Note that for $j \in [4]$,

\[
\begin{align*}
\epsilon_{\ell,j} & \leftarrow (g_j^{-x_{\ell,j}} y_{\ell,j+1})^{y_{\ell,j}} T_{k_{\ell,j}} & (i) \\
G_3 & \leftarrow (u_{\ell,j}^{-x_{\ell,j}} u_{\ell,j+1}^{y_{\ell,j}}) T_{k_{\ell,j}} & (ii)
\end{align*}
\]
Table 2: Brief description of the security proof of Theorem 24.

<table>
<thead>
<tr>
<th>Changes between adjacent games</th>
<th>Assumptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>G₀ The original n-KDM-CCA security game.</td>
<td>—</td>
</tr>
<tr>
<td>G₁ decrypt: reject if ((a_i, a_i a_i e_i)) for some (\ell \in {Q_i}).</td>
<td>(G_0 \approx G_1)</td>
</tr>
<tr>
<td>G₂ initialize: sample secret keys with ((x_i, y_i, \ldots, x_i, y_i) := (x_i, y_i, \ldots, x_i, y_i) + (x_i, y_i, \ldots, x_i, y_i)).</td>
<td>(G_1 = G_2)</td>
</tr>
<tr>
<td>G₃ encrypt((f_{\ell, i})): use the secret keys to run KEM Encrypt and (&amp;) Encrypt.</td>
<td>(G_2 \approx G_3)</td>
</tr>
<tr>
<td>G₄ encrypt((f_{\ell, i})) when encrypt oracle encrypts affine function of secret keys, (\varepsilon e_i) is computed with ((\tilde{u}<em>{\ell, i})</em>{i \in [n]} := (g_{1}^{t_{\ell, i}} s_{\ell, i}^{t_{\ell, i}} \ldots g_{5}^{t_{\ell, i}} s_{\ell, i}^{t_{\ell, i}})) instead of ((g_{1}^{t_{\ell, i}} \ldots g_{5}^{t_{\ell, i}})). encrypt does not use ((x_{p, y})<em>{p, y}^{t</em>{p, y}}) mod (N) any more if ((\delta_{p})_{p \in [n]}) is carefully chosen.</td>
<td>(G_3 \approx G_4) by IV₅</td>
</tr>
<tr>
<td>G₅ encrypt((f_{\ell, i})): KEM ct (= ai) of KEM Encrypt is computed with ((u_{\ell, i})<em>{i \in [n]} := (g</em>{1}^{t_{\ell, i}} s_{\ell, i}^{t_{\ell, i}} \ldots g_{5}^{t_{\ell, i}} s_{\ell, i}^{t_{\ell, i}})) instead of ((g_{1}^{t_{\ell, i}} \ldots g_{5}^{t_{\ell, i}})). Now (\text{KEM Encrypt}) encapsulates four keys ((k_{\ell, j} - r_{\ell} \cdot (\alpha_{i} x_{i} + \alpha_{j} y_{i}))<em>{j = 1}^{i} \text{mod} N) but ((k</em>{\ell, j})_{j = 1}^{i} \text{is the key used in AIAE Encrypt.})</td>
<td>(G_4 \approx G_5) by IV₅</td>
</tr>
<tr>
<td>G₆ encrypt((f_{\ell, i})): sample (k_{\ell, j} = r_{\ell} k_{\ell, j}^{<em>} + s_{\ell, j}) for (j \in [4]). Now (\text{KEM Encrypt}) encapsulates four keys ((r_{\ell}(k_{\ell}^{</em>} - \alpha_{j} x_{i} - \alpha_{j} y_{i}) - r_{\ell}(\alpha_{j} y_{i} + \alpha_{j} x_{i}) + s_{\ell, j}^{t_{\ell, j}})<em>{j = 1}^{i} \text{mod} N) but ((r</em>{\ell} k_{\ell}^{*} + s_{\ell, j})_{j = 1}^{i} \text{is the key used in AIAE Encrypt.})</td>
<td>(G_5 \approx G_6)</td>
</tr>
<tr>
<td>G₇ decrypt: use (\phi(N)) and secret keys to answer decryption queries.</td>
<td>(G_6 \approx G_7)</td>
</tr>
<tr>
<td>G₈ decrypt: add an additional rejection rule. Reject if (\text{Bad'} := ((u_{\ell, j} \notin \text{SCR}<em>{\ell, j})) or (\text{Bad} := (u</em>{\ell, j} \notin \text{SCR}<em>{\ell, j}) \land (\tilde{u}</em>{\ell, j} \notin \text{SCR}<em>{\ell, j})) happens. (\text{Bad'}) and (\text{Bad}) can be detected by using (\phi(N)). Now only the ((\text{mod} \phi(N)/4)) part of secret keys and (\phi(N)) are used in decrypt. The randomness of ((\alpha</em>{j} x_{i} + \alpha_{j} y_{i})<em>{j = 1}^{i} \text{mod} N) perfectly hides ((k</em>{0}, \ldots, k_{i})) in encrypt, thus ((k_{0}, \ldots, k_{i})) is uniform. (\text{Bad'}) may lead to a fresh successful forgery for AIAE_DHO.</td>
<td>(G_7 = G_8) if neither (\text{Bad'}) nor (\text{Bad}) happens. (\text{Pr}[\text{Bad'}] = \text{negl}) due to weak INT-(\Sigma_\text{aff})-RKA security of AIAE_DHO.</td>
</tr>
<tr>
<td>G₉ initialize: sample an independent random tuple ((k_{1}, \ldots, k_{i})). encrypt((f_{\ell, i})): use ((r_{\ell} k_{\ell}^{*} + s_{\ell, j})_{j = 1}^{i} \text{in AIAE Encrypt.})</td>
<td>(G_8 \approx G_9) by IND-(\Sigma_\text{aff})-RKA security of AIAE_DHO. (\text{Pr}[\text{Bad}] = \text{negl})</td>
</tr>
<tr>
<td>G₁₀ encrypt: encrypt zeros instead of the affine function of secret keys. Bad happens with negligible probability, since (i \neq g_{1}^{t_{\ell, i}} \text{mod} N) in decrypt. (\text{Adversary of wins with probability 1/2.})</td>
<td>(G_9 \approx G_{10}) by IND-(\Sigma_\text{aff})-RKA security of AIAE_DHO. (\text{Pr}[\text{Bad}] = \text{negl})</td>
</tr>
</tbody>
</table>

Thus, \(G_3\) is the same as \(G_2\), and \(\text{Pr}[\text{Succ}] = \text{Pr}[\text{Succ}]\).

Game \(G_4\). It is identical to \(G_2\), except the way the challenger responds to the \(\ell\)th \((\ell \in [Q_3])\) Encrypt query \((f_{\ell, i}, \ell)\). In game \(G_3\), in the case of \(\beta = 1\), \((\tilde{u}_{\ell, 1}, \ldots, \tilde{u}_{\ell, 8})\) and \(\tilde{e}_{\ell}\) are computed without the use of \((x_{1}, y_{1}, \ldots, x_{4}, y_{4})\) mod \(N\):

\[
\tilde{e}_{\ell} = \tilde{h}_{\ell, 1}^{t_{\ell, 1}} \cdots \tilde{h}_{\ell, 4}^{t_{\ell, 4}} \text{mod} N^{4}.
\]

\[
\tilde{e}_{\ell} = \left(\tilde{g}_{1}^{t_{\ell, 1}} \cdots \tilde{g}_{5}^{t_{\ell, 5}}\right) \text{mod} N^{5}.
\]

(ii)

\[
\tilde{e}_{\ell} = \left(\tilde{g}_{1}^{t_{\ell, 1}} \cdots \tilde{g}_{5}^{t_{\ell, 5}}\right) \text{mod} N^{4}.
\]

Thus, \(G_3\) is the same as \(G_2\), and \(\text{Pr}[\text{Succ}] = \text{Pr}[\text{Succ}]\).

Note that

\[
\tilde{e}_{\ell} = \left(\tilde{h}_{\ell, 1}^{t_{\ell, 1}} \cdots \tilde{h}_{\ell, 8}^{t_{\ell, 8}}\right) \text{mod} N^{4}.
\]
where the third equality follows from \( m_1 = \sum_{i=1}^{n} (a_i x_{i1} + b_i x_{i2} + ... + a_{i4} x_{i4} + b_{i4} y_{i4}) + c \).

We analyze the difference between \( G_3 \) and \( G_4 \) via the following lemma.

**Lemma 25.** One has \( |Pr_s[\text{Succ}] - Pr_s[\text{Succ}]| \leq Adv_{\text{Gen}}(\lambda) \).

**Proof.** According to the last line of (35), the way that \( \tilde{e}_i \) is computed from \((u_{e_1}, \ldots, u_{e_8})\) is the same in \( G_3 \) and \( G_4 \). Therefore the only divergence between \( G_3 \) and \( G_4 \) lies in \((\tilde{u}_{e_1}, \ldots, \tilde{u}_{e_8})\).

We show that any difference between \( G_3 \) and \( G_4 \) results in a PPT adversary \( B_1 \) solving the IV\_S problem. \( B_1 \) is provided with \((N, g_1, \ldots, g_5)\) and has access to its \( \text{chal}_b \) oracle. \( B_1 \) simulates game \( G_3 \) or game \( G_4 \) for \( \mathcal{A} \). Firstly, \( B_1 \) prepares pars and generates \((pk, sk)\), \( i \in [n] \), as in \( G_3 \) and \( G_4 \). As for the \( \ell \)th \((\ell \in [Q_2])\) encrypt query \((f_{e_i}, t)\) from \( \mathcal{A} \), where \( f_{e_i} = (a_{i1}, a_{i2}, a_{i3}, a_{i4}, b_{i1} e_{i1} e_{i2} e_{i3} e_{i4}) \) \( \in \mathcal{F}_{\text{aff}} \), \( B_1 \) proceeds as follows: it queries its own \( \text{chal}_b \) oracle with \((\sum_{i=1}^{n} a_{i1} x_{i1} + \sum_{i=1}^{n} a_{i2} x_{i2} + \sum_{i=1}^{n} a_{i3} x_{i3} + \sum_{i=1}^{n} a_{i4} x_{i4} + \sum_{i=1}^{n} b_{i1} y_{i1} + \sum_{i=1}^{n} b_{i2} y_{i2} + \sum_{i=1}^{n} b_{i3} y_{i3} + \sum_{i=1}^{n} b_{i4} y_{i4})\), where the symbol \( \ast \) denotes dummy messages. Then \( B_1 \) obtains its challenges \((\tilde{u}_{e_1}, \tilde{u}_{e_2}, \tilde{u}_{e_3}, \tilde{u}_{e_4}, \tilde{u}_{e_5}, \tilde{u}_{e_6}, \tilde{u}_{e_7}, \tilde{u}_{e_8})\) \( (\ast, \ast, \ast, \ast, \ast, \ast, \ast, \ast) \) \( \neq \) the \( \ast \) and \( \ast \) terms. According to the definition of \( \text{chal}_b \) oracle, \((\tilde{u}_{e_1}, \ldots, \tilde{u}_{e_8})\) is one of the following:

**Case 1** \((b = 0)\), \( (g_{e_1}^{f_{e_1}}, g_{e_2}^{f_{e_2}}, g_{e_3}^{f_{e_3}}, g_{e_4}^{f_{e_4}}, g_{e_5}^{f_{e_5}}, g_{e_6}^{f_{e_6}}) \),

**Case 2** \((b = 1)\), \( (g_{e_1}^{f_{e_1}}, g_{e_2}^{f_{e_2}}, g_{e_3}^{f_{e_3}}, g_{e_4}^{f_{e_4}}, g_{e_5}^{f_{e_5}}, g_{e_6}^{f_{e_6}}, g_{e_7}^{f_{e_7}}, g_{e_8}^{f_{e_8}}) \).

Next \( B_1 \) uses the obtained \((\tilde{u}_{e_1}, \ldots, \tilde{u}_{e_8})\) and the secret keys to compute \( \tilde{e}_i \) via (35) for \( \mathcal{A} \). In the meantime, \( B_1 \) can also simulate decrypt for \( \mathcal{A} \) since it knows the secret keys. Finally, \( B_1 \) outputs 1 if the event Succ occurs.

In Case 1, \( B_1 \) simulates game \( G_3 \) perfectly for \( \mathcal{A} \); in Case 2, \( B_1 \) simulates game \( G_4 \) perfectly for \( \mathcal{A} \). Any difference between \( Pr_s[\text{Succ}] \) and \( Pr_r[\text{Succ}] \) results in \( B_1 \)'s advantage over the IV\_S problem. Thus Lemma 25 follows.

**Game \( G_5 \).** It is identical to \( G_3 \), except for the following differences. In the initialize procedure of game \( G_3 \), the challenger picks \( r^* \leftarrow \{\{N/4\}\} \) and \( a_i, \ldots, a_N \leftarrow \mathbb{Z}_N \) randomly. As for the \( \ell \)th \((\ell \in [Q_2])\) encrypt query \((f_{e_i}, t)\), the challenger computes \((u_{e_1}, \ldots, u_{e_5})\) as follows:

(i) \((u_{e_1}, \ldots, u_{e_5}) = ((g_1^r T^{a_1})^{r_1}, \ldots, (g_5^r T^{a_5})^{r_5}) \mod N^2\).

The only difference between \( G_4 \) and \( G_5 \) is the distribution of \((u_{e_1}, \ldots, u_{e_5})\). In game \( G_4 \), \((u_{e_1}, \ldots, u_{e_5}) = ((g_1^r T^{a_1})^{r_1}, \ldots, (g_5^r T^{a_5})^{r_5}) \mod N^2\), while in game \( G_5 \), \((u_{e_1}, \ldots, u_{e_5}) = ((g_1^r T^{a_1})^{r_1}, \ldots, (g_5^r T^{a_5})^{r_5}) \mod N^2\). Just like Lemma 25, any difference between \( G_4 \) and \( G_5 \) results in a PPT adversary solving IV\_S problem by invoking \( \mathcal{A} \). Therefore, \( |Pr_s[\text{Succ}] - Pr_r[\text{Succ}]| \leq Adv_{\text{Gen}}(\lambda) \).

**Game \( G_6 \).** It is identical to \( G_5 \), except for the following differences. In the initialize procedure of game \( G_6 \), the challenger picks \( r^* \leftarrow \{\{N/4\}\} \) and \( a_i, \ldots, a_N \leftarrow \mathbb{Z}_N \) randomly. As for the \( \ell \)th \((\ell \in [Q_2])\) encrypt query \((f_{e_i}, t)\), the challenger computes \((u_{e_1}, \ldots, u_{e_5})\) as follows:

(i) \((u_{e_1}, \ldots, u_{e_5}) = ((g_1^r T^{a_1})^{r_1}, \ldots, (g_5^r T^{a_5})^{r_5}) \mod N^2\).
\[
\left(y'_1, \ldots, y'_4\right) = \left(\frac{d \log_T \left(e_1^{\phi(N)}\right)}{\phi(N)}, \ldots, \frac{d \log_T \left(e_4^{\phi(N)}\right)}{\phi(N)}\right)
\]

\mod N,

\(k = (k_1, \ldots, k_4) = (\alpha'_1 x_{i,1} + \alpha'_2 y_{i,1} + \gamma', \ldots, \alpha'_4 x_{i,4} + \alpha'_5 y_{i,4} + \gamma')\)

\mod N,

\[
\begin{align*}
\tilde{v}_c &= (\bar{u}_1, \ldots, \bar{u}_k, \bar{t}) / \perp \leftarrow \text{AIAE-Decrypt}(k, aiae_c, ai), \quad (39) \\
\tilde{\alpha}_1, \ldots, \tilde{\alpha}_9 &= \left(\frac{d \log_T \left(e_1^{\phi(N)}\right)}{\phi(N)}, \ldots, \frac{d \log_T \left(e_8^{\phi(N)}\right)}{\phi(N)}\right) \mod N'^{-1}, \\
\tilde{y} &= \frac{d \log_T \left(e^{\phi(N)}\right)}{\phi(N)} \mod N'^{-1}, \\
m &= \tilde{\alpha}_1 x_{i,1} + \tilde{\alpha}_2 y_{i,1} + \tilde{\alpha}_3 x_{i,2} + \tilde{\alpha}_4 y_{i,2} + \tilde{\alpha}_5 x_{i,3} + \tilde{\alpha}_6 y_{i,3} \\
&\quad + \tilde{\alpha}_7 x_{i,4} + \tilde{\alpha}_8 y_{i,4} + \tilde{\alpha}_9 \mod N'^{-1}.
\end{align*}
\]

According to (8), for \(j \in [4]\), we have that

\[
k_j \overset{G_7}{=} d \log_T \left(e_j \frac{u_{x_{i,j}}^{x_{i,j}} u_{y_{i,j}}^{y_{i,j}}}{\phi(N)}\right) \mod N
\]

\[
= \frac{d \log_T \left(u_j^{\phi(N) x_{i,j}}\right)}{\phi(N)} + \frac{d \log_T \left(u_{j+1}^{\phi(N) y_{i,j}}\right)}{\phi(N)}
\]

\[
+ \frac{d \log_T \left(e_j^{\phi(N)}\right)}{\phi(N)} + \frac{d \log_T \left(u_j^{\phi(N) x_{i,j}}\right)}{\phi(N)} \cdot x_{i,j} + \frac{d \log_T \left(u_{j+1}^{\phi(N) y_{i,j}}\right)}{\phi(N)} \cdot y_{i,j}
\]

\[
= \frac{G_j}{\phi(N)} \cdot \frac{u_{x_{i,j}}^{x_{i,j}} u_{y_{i,j}}^{y_{i,j}}}{\phi(N)} \mod N
\]

\[
m \overset{G_8}{=} d \log_T \left(\frac{\bar{u}_1^{\phi(N)}}{\phi(N)} \cdot x_{i,1} + \ldots + \frac{\bar{u}_6^{\phi(N)}}{\phi(N)} \cdot y_{i,4} + \frac{d \log_T \left(e^{\phi(N)}\right)}{\phi(N)} \cdot \frac{\bar{u}_7^{\phi(N)}}{\phi(N)} \cdot y_{i,4} \right) \mod N'^{-1}
\]

\[
g \overset{G_9}{=} d \log_T \left(\frac{\bar{u}_1^{\phi(N)}}{\phi(N)} \cdot x_{i,1} + \ldots + \frac{\bar{u}_6^{\phi(N)}}{\phi(N)} \cdot y_{i,4} + \frac{d \log_T \left(e^{\phi(N)}\right)}{\phi(N)} \cdot \frac{\bar{u}_7^{\phi(N)}}{\phi(N)} \cdot y_{i,4} \right) \mod N'^{-1}
\]

Hence \(G_7\) is essentially the same as \(G_6\), and \(Pr_6[\text{Succ}] = Pr_7[\text{Succ}]\).

**Game G_8.** It is identical to \(G_7\), except the way of answering the \textsc{decrypt} oracle queries (\((ai, ai.e, c), i \in [n]\)). More precisely, a rejection rule is added in \(\text{DECRYPT}\):

1. If \(\alpha'_1 \neq 0 \vee \cdots \vee \alpha'_5 \neq 0 \vee \tilde{\alpha}_1 \neq 0 \vee \cdots \vee \tilde{\alpha}_9 \neq 0\), output \(\perp\).

Denote by \(\text{Bad}\) the event that \(\mathcal{A}\) ever queries the \textsc{decrypt} oracle with \((\langle ai, ai.e, c\rangle, i \in [n]), satisfying

\[
e_1 u_{x_{i,1}}^{x_{i,1}} u_{y_{i,1}}^{y_{i,1}} \ldots e_4 u_{x_{i,4}}^{x_{i,4}} u_{y_{i,4}}^{y_{i,4}} \in \mathbb{R}_N^4 \]

\begin{align*}
&\land \text{AIAE-Decrypt}(k, aiae_c, ai) \neq \perp \\
&\land \bar{e}_1 u_{x_{i,1}}^{x_{i,1}} u_{x_{i,2}}^{x_{i,2}} u_{x_{i,3}}^{x_{i,3}} u_{x_{i,4}}^{x_{i,4}} u_{y_{i,1}}^{y_{i,1}} u_{y_{i,2}}^{y_{i,2}} u_{y_{i,3}}^{y_{i,3}} u_{y_{i,4}}^{y_{i,4}} \in \mathbb{R}_N^4, \\
&\land t = g_1^m \mod N \\
&\land \left(\alpha'_1 \neq 0 \vee \cdots \vee \alpha'_5 \neq 0 \vee \tilde{\alpha}_1 \neq 0 \vee \cdots \vee \tilde{\alpha}_9 \neq 0\right). \quad (44)
\end{align*}

Obviously, \(G_8\) is identical to \(G_7\) unless \(\text{Bad}\) occurs. Thus, \([Pr_7[\text{Succ}] - Pr_8[\text{Succ}]| \leq Pr_8[\text{Bad}]\).

To show the computational indistinguishability of \(G_7\) and \(G_8\), we must prove that \(Pr_8[\text{Bad}]\) is negligible. To this end, \(\text{Bad}\) is divided into two subevents:

1. **\(\text{Bad}'\):** \(\mathcal{A}\) ever queries the \textsc{decrypt} oracle with \((\langle ai, ai.e, c\rangle, i \in [n]), satisfying

\[
e_1 u_{x_{i,1}}^{x_{i,1}} u_{y_{i,1}}^{y_{i,1}} \ldots e_4 u_{x_{i,4}}^{x_{i,4}} u_{y_{i,4}}^{y_{i,4}} \in \mathbb{R}_N^4 \]

\begin{align*}
&\land \text{AIAE-Decrypt}(k, aiae_c, ai) \neq \perp \\
&\land \bar{e}_1 u_{x_{i,1}}^{x_{i,1}} u_{x_{i,2}}^{x_{i,2}} u_{x_{i,3}}^{x_{i,3}} u_{x_{i,4}}^{x_{i,4}} u_{y_{i,1}}^{y_{i,1}} u_{y_{i,2}}^{y_{i,2}} u_{y_{i,3}}^{y_{i,3}} u_{y_{i,4}}^{y_{i,4}} \in \mathbb{R}_N^4, \\
&\land t = g_1^m \mod N \\
&\land \left(\alpha'_1 = \cdots = \alpha'_5 = 0 \land \tilde{\alpha}_1 = \cdots = \tilde{\alpha}_9 = 0\right). \quad (45)
\end{align*}

2. **\(\text{Bad}''\):** \(\mathcal{A}\) ever queries the \textsc{decrypt} oracle with \((\langle ai, ai.e, c\rangle, i \in [n]), satisfying

\[
e_1 u_{x_{i,1}}^{x_{i,1}} u_{y_{i,1}}^{y_{i,1}} \ldots e_4 u_{x_{i,4}}^{x_{i,4}} u_{y_{i,4}}^{y_{i,4}} \in \mathbb{R}_N^4 \]

\begin{align*}
&\land \text{AIAE-Decrypt}(k, aiae_c, ai) \neq \perp \\
&\land \bar{e}_1 u_{x_{i,1}}^{x_{i,1}} u_{x_{i,2}}^{x_{i,2}} u_{x_{i,3}}^{x_{i,3}} u_{x_{i,4}}^{x_{i,4}} u_{y_{i,1}}^{y_{i,1}} u_{y_{i,2}}^{y_{i,2}} u_{y_{i,3}}^{y_{i,3}} u_{y_{i,4}}^{y_{i,4}} \in \mathbb{R}_N^4, \\
&\land t = g_1^m \mod N \\
&\land \left(\alpha'_1 = \cdots = \alpha'_5 = 0 \land \tilde{\alpha}_1 = \cdots = \tilde{\alpha}_9 = 0\right). \quad (46)
\end{align*}

Obviously, \(Pr_8[\text{Bad}] \leq Pr_8[\text{Bad}'] + Pr_8[\text{Bad}]\). We will defer the analysis of \(Pr_8[\text{Bad}'] \) to subsequent games. Through the following lemma, we provide the analysis of \(Pr_8[\text{Bad}]\).

**Lemma 26.** One has \(Pr_8[\text{Bad}'] \leq 2Q_4 \cdot Ad_{\text{AIAE-Dec}}^{\text{weak-int-rkMr}}(\lambda)\).
Proof. In Decrypt of game $G_8$, the challenger will reply \( \perp \) to $\mathcal{A}$ unless $\alpha_1 = \cdots = \alpha_5 = 0$ and $\xi_8 = 0$. Consequently, the $(\text{mod } \phi(N)/4)$ part of $s_k$, that is, $(x_{i,1}, y_{i,1}, \ldots, x_{i,4}, y_{i,4}) \mod \phi(N)/4$, $i \in [n]$, and the value of $\phi(N)$, is enough for answering Decrypt queries. In particular, the values of $(x_1, y_1, \ldots, x_4, y_4) \mod N$ are not necessary in Decrypt.

$\mathcal{B}'$ is further divided into the following two subevents:

(i) $\mathcal{B}'\cdot 1$: $\mathcal{A}$ ever queries the Decrypt oracle with \((\langle a_i, \text{iaae.c}, \rangle, i \in [n]), \text{satisfying}
\begin{align}
\text{Conditions (42), (43)} \land (\alpha'_1 \neq 0 \lor \cdots \lor \alpha'_5 \neq 0) \\
\land (\exists j \in [4], \alpha'_j \neq \alpha_{j+1} \mod N).
\end{align}
(47)

(ii) $\mathcal{B}'\cdot 2$: $\mathcal{A}$ ever queries the Decrypt oracle with \((\langle a_i, \text{iaae.c}, \rangle, i \in [n]), \text{satisfying}
\begin{align}
\text{Conditions (42), (43)} \land (\alpha'_1 \neq 0 \lor \cdots \lor \alpha'_5 \neq 0) \\
\land (\alpha'_1 = \cdots = \alpha'_5 \mod N).
\end{align}
(48)

Recall that $(\alpha_1, \ldots, \alpha_5)$ are chosen in initialization.

We will consider the two subevents in game $G_8$ separately via the following two claims.

\textbf{Claim 27.} One has $Pr_{G_8}[\mathcal{B}'\cdot 1] \leq Q_d \cdot Adv_{\text{IAE}_{\text{DDH}}}^{\text{weak-int-rka}}(\lambda)$.

\textbf{Proof.} In game $G_8$, the values of $(x_1, y_1, \ldots, x_4, y_4) \mod N$ are not needed in Decrypt, and the computation of $t_k = g_q^{y_k} \mod N$ in Encrypt only makes use of $(x_1, y_1, \ldots, x_4, y_4) \mod \phi(N)/4$. Thus the only information about $(x_1, y_1, \ldots, x_4, y_4) \mod N$ leaked to $\mathcal{A}$ is through the computation of $(e_{t_{1,j}}, \ldots, e_{t_{4,j}})$ in Encrypt, which may leak the values of $(\alpha_1 x_1 + \alpha_2 y_1), (\alpha_2 x_2 + \alpha_3 y_2), (\alpha_3 x_3 + \alpha_4 y_3), (\alpha_4 x_4 + \alpha_5 y_4) \mod N$; for $j \in [4],
\begin{align}
e_{t_{1,j}} &= g_{e_{t_{1,j}}}^{x_{i,1}} \mod N^2 \\
r_e (k_{i} - \alpha_1 x_1 - \alpha_2 y_1)^{r_{e,t_{1,j}}^{i}} &= h_{e_{t_{1,j}}}^{r_{e,t_{1,j}}^{i}} T^{\xi_{t_{1,j}}} \mod N^2.
\end{align}
(49)

If $\mathcal{B}'\cdot 1$ occurs, for concreteness, say that $\alpha'_1/\alpha_1 \neq \alpha'_2/\alpha_2 \mod N$, then
\begin{align}
k_1 &= \alpha'_1 x_{i,1} + \alpha'_2 y_{i,1} + y'_i \\
&= \alpha'_1 x_{i,1} + \alpha'_2 y_{i,1} + \alpha'_3 x_{i,2} + \alpha'_4 y_{i,2} + y'_i \mod N,
\end{align}
(50)

where $k_1$ is independent of $(\alpha_1 x_1 + \alpha_2 y_1) \mod N$, thus uniformly distributed over $\mathbb{Z}_N$ from $\mathcal{A}'$'s view. By Remark 23, for $k = (k_1, k_2, k_3, k_4)$ where $k_1 \leftarrow s \mathbb{Z}_N$, the probability of $\text{AIAE}.\text{Decrypt}(k, \text{iaae.c}, a_i) \neq \perp$ is upper bounded by $Adv_{\text{IAE}_{\text{DDH}}}^{\text{weak-int-rka}}(\lambda)$.

Then $Pr_{G_8}[\mathcal{B}'\cdot 1] \leq Q_d \cdot Adv_{\text{IAE}_{\text{DDH}}}^{\text{weak-int-rka}}(\lambda)$.

\textbf{Claim 28.} One has $Pr_{G_8}[\mathcal{B}'\cdot 2] \leq Q_d \cdot Adv_{\text{IAE}_{\text{DDH}}}^{\text{weak-int-rka}}(\lambda)$.

\textbf{Proof.} Similar to the discussion in the proof for the previous claim, in game $G_8$, the only information about $(x_1, y_1, \ldots, x_4, y_4) \mod N$ and $k' = (k'_1, k'_2, k'_3, k'_4)$ involved is through Encrypt, which uses the value of $k_1 = (k_1 - \alpha_1 x_1 - \alpha_2 y_1), k_2 = (k'_2 - \alpha_3 x_3 - \alpha_4 y_3), k_3 = (k'_3 - \alpha_5 x_4 - \alpha_6 y_4), \hat{k}_4 = (k'_4 - \alpha_1 x_4 - \alpha_6 y_4) \mod N$ via computing $(e_{t_{1,j}}, \ldots, e_{t_{4,j}})$ (see (49)) and also uses $k_j = r_{e,t_{j}}^{i} (k'_1, k'_2, k'_3, k'_4)$ as the encryption key of $\text{IAE}.\text{Encrypt}$.

Note that because of the randomness of $(x_1, y_1, \ldots, x_4, y_4) \mod N$, $(\hat{k}_1, \hat{k}_2, \hat{k}_3, \hat{k}_4)$ are uniformly distributed and independent of $(k'_1, k'_2, k'_3, k'_4)$. Therefore it is possible to construct an algorithm to simulate Decrypt and Encrypt of game $G_8$ without $k'' = (k'_1, k'_2, k'_3, k'_4)$ and $(x_1, y_1, \ldots, x_4, y_4) \mod N$. The algorithm can also simulate $\text{IAE}.\text{Encrypt}$ as long as it has access to a weak-INT-$\mathcal{F}_{\text{raff}}$-RKA encryption oracle of the $\text{IAE}_{\text{DDH}}$ scheme.

More precisely, we construct a PPT adversary $\mathcal{B}_2(\text{par}_{\text{iaae}}, \text{which has access to}\text{ENC}_{\text{IAE}}$ oracle, against the weak-INT-$\mathcal{F}_{\text{raff}}$-RKA security of the $\text{IAE}_{\text{DDH}}$ scheme, where $\text{par}_{\text{iaae}} = (N, p, q, \ldots)$. $\mathcal{B}_2$ does not choose $k'' = (k'_1, k'_2, k'_3, k'_4)$ in initialize any more, and it implicitly sets $k''$ to be the encryption key used by its weak-INT-$\mathcal{F}_{\text{raff}}$-RKA challenger. $\mathcal{B}_2$ does not choose $(x_1, y_1, \ldots, x_4, y_4) \mod N$ either, and instead, it chooses $\hat{k} = (\hat{k}_1, \hat{k}_2, \hat{k}_3, \hat{k}_4)$ uniformly from $(\mathbb{Z}_N)^4$. $\mathcal{B}_2$ picks $(x_1, y_1, \ldots, x_4, y_4) \mod \phi(N)/4$ and $(\bar{x}_1, \bar{y}_1, \ldots, \bar{x}_4, \bar{y}_4) \in \{0,1\}^4, i \in [n]$, randomly. To simulate Encrypt, $\mathcal{B}_2$ can use $(\bar{x}_1, \bar{y}_1, \ldots, \bar{x}_4, \bar{y}_4)_{j=1}$ to compute $(e_{t_{j}}, y_{i,j})$ via (49) and use $(\bar{x}_1, \bar{y}_1, \ldots, \bar{x}_4, \bar{y}_4)_{j=1}$, $i \in [n]$, to compute $e_{t_{j}}$. Note that $\mathcal{B}_2$ is able to compute $t_k = g_q^{y_k} \mod N$, even if $\beta = 1$, because it knows the $(\text{mod } \phi(N)/4)$ part of $s_k$, that is, $(x_1, y_1, \ldots, x_4, y_4) \mod N$, and hence, it follows that $\mathcal{B}_2$ answers decryption queries with the $(\text{mod } \phi(N)/4)$ part of all the secret keys and $\phi(N) = (p - 1)(q - 1)$, just like $G_0$.

Suppose that $\mathcal{A}$ ever queries the Decrypt oracle with \((\langle a_i, \text{iaae.c}, \rangle, i \in [n]), \text{satisfying}
\begin{align}
k_j &= \alpha'_x x_{i,j} + \alpha'_y y_{i,j} + y'_i \\
&= r \cdot (\alpha'_x x_{i,j} + \alpha'_y y_{i,j} + y'_i) \mod N
\end{align}
(50)
In $G_8$, the only place that involves the value of $(x_1,y_1,\ldots,x_4,y_4)$ mod $N$ is in the computation of $(e_{\ell,1},\ldots,e_{\ell,4})$ in the Encrypt oracle. Specifically, for $j \in [4]$,\[ e_{\ell,j} = h_{\ell,j}^r g_{\ell,j}^s u_{\ell,j}^a \mod N^2 \]
\[ = h_{\ell,j}^{r_r} r_{\ell,j}^{r_r} (a_{\ell,j} - a_{j+1,j}) y_j \mod N^2 \]
\[ \mod N^2. \]

Note that the computation of $t_\ell = g_{\ell}^{m_{g_\ell}} \mod N$ in the Encrypt oracle only involves $(x_1,y_1,\ldots,x_4,y_4)$ mod $\phi(N)/4$. Moreover, observe that neither $k^* = (k_1^*,k_2^*,k_3^*,k_4^*)$ nor $(x_1,y_1,\ldots,x_4,y_4) \mod N$ is used in Decrypt. Hence, $k^*$ is perfectly hidden by $(x_1,y_1,\ldots,x_4,y_4) \mod N$.

Therefore, the challenger could always employ another $k^* = (k_1^*,k_2^*,k_3^*,k_4^*)$ in the computation of $\kappa_\ell$ for the AIAE$_{D\text{DDH}}$ encryption in the Encrypt oracle as in $G_8$.

Then game $G_4$ and game $G_8$ are essentially the same from $s$’s view, so $Pr_{s}[\text{Succ}] = Pr_{s}[\text{Succ}]$ and $Pr_{\tilde{s}}[\text{Bad}] = Pr_{s}[\text{Bad}]$.

Game $G_{10}$. It is identical to $G_9$, except the way the challenger answers the $\ell$th ($\ell \in [Q_1]$) Encrypt oracle query $(f_{\ell},t_{\ell})$. More precisely, in game $G_{10}$, the challenger computes $\text{aiae}_c$ in the following way:

(i) $\text{aiae}_c \leftarrow s$ AIAE.Encrypt($\kappa_\ell, \phi^{\ell,s}, a_{\ell}$).

Observe that, in $G_3$ and $G_{10}$, $k^*$ is employed only in the AIAE$_{D\text{DDH}}$ encryption, where it uses $\kappa_\ell = r_{\ell} \cdot \kappa^* + s_{\ell}$ as the encryption key with $s_{\ell} = (s_{\ell,1},\ldots,s_{\ell,4})$. Any difference between $G_3$ and $G_{10}$ results in a PPT adversary against the IND-$\text{FF}^{\text{raff-RKA}}$ security of the AIAE$_{D\text{DDH}}$ scheme. Therefore, $|Pr_{s}[\text{Succ}] - Pr_{10}[\text{Succ}]| \leq \text{Adv}^{\text{ind-RKA}}_{\text{AIAE}_{D\text{DDH}}} (\lambda)$ and $|Pr_{s}[\text{Bad}] - Pr_{10}[\text{Bad}]| \leq \text{Adv}^{\text{ind-RKA}}_{\text{AIAE}_{D\text{DDH}}} (\lambda)$.

Finally in $G_{10}$, the challenger always computes the AIAE$_{D\text{DDH}}$ encryption of $0^\lambda$ in the Encrypt oracle, so $\beta$ is perfectly hidden from $s$’s view. Thus, $Pr_{10}[\text{Succ}] = 1/2$.

To complete the proof of Theorem 24, we only need to prove the following lemma.

Lemma 29. One has $Pr_{10}[\text{Bad}] \leq (Q_d + 1) \cdot 2^{\Omega(\lambda)} + \text{Adv}_{\text{Gen}}^{\text{IND-RKA}} (\lambda)$.

Proof. In $G_{10}$, neither Decrypt nor Encrypt uses the values of $(x_1,y_1,\ldots,x_4,y_4) \mod \phi(N)/4$. The only information leaked about them lies in the public keys $pk_i$, $i \in [n]$, which reveal the values of $(w_1x_1 + w_2y_1, w_2x_2 + w_3y_2, w_3x_3 + w_4y_3, w_4x_4 + w_5y_4) \mod \phi(N)/4$, where we denote $w_i = \log g_j \mod \phi(N)/4$ for some base $g \in \text{SCR}_N$, $j \in [5]$. 

Bad is further divided into the following disjoint two subevents:

(i) **Bad-1**: \(\mathcal{A}\) ever queries the decrypt oracle with \((\langle a_i, a_{i.e.}\rangle, i \in [n])\), satisfying

\[
\begin{align*}
\mathcal{A}_i & \neq \mathcal{A}_j, \\
\mathcal{A}_i & \neq \mathcal{A}_k, \\
\forall i & \neq j & & \text{if } i \neq j, \quad (\mathcal{A}_i, a_{i.e.}\rangle) & \neq (\mathcal{A}_j, a_{j.e.}\rangle), \\
\forall i & \neq j & & \text{if } i \neq j, \quad (\mathcal{A}_i, a_{i.e.}\rangle) & \neq (\mathcal{A}_j, a_{j.e.}\rangle). \\
\end{align*}
\]

Conditions (42), (43) \(\wedge \left(\alpha'_i = \cdots = \alpha'_j = 0\right) \wedge (\alpha_i)

\[
\begin{align*}
\# \neq 0 & \wedge \left(\sum_{i=1}^{n} \frac{(\alpha_i)}{w_i} \neq \sum_{i=1}^{n} \frac{(\alpha_j)}{w_j} \neq \sum_{i=1}^{n} \frac{(\alpha_k)}{w_k} \right) \\
\# & = \sum_{i=1}^{n} w_i. 
\end{align*}
\]

(ii) **Bad-2**: \(\mathcal{A}\) ever queries the decrypt oracle with \((\langle a_i, a_{i.e.}\rangle, i \in [n])\), satisfying

Conditions (42), (43) \(\wedge \left(\alpha'_i = \cdots = \alpha'_j = 0\right) \wedge (\alpha_i)

\[
\begin{align*}
\# & = \sum_{i=1}^{n} w_i, \\
\# & = \sum_{i=1}^{n} \frac{(\alpha_i)}{w_i} \neq \sum_{i=1}^{n} \frac{(\alpha_j)}{w_j} \neq \sum_{i=1}^{n} \frac{(\alpha_k)}{w_k} \right) \\
\# & = \sum_{i=1}^{n} w_i. 
\end{align*}
\]

In all, we proved the \(n\)-KDM\(\mathcal{F}_{\text{aff}}\)-CCA security. This completes the proof of Theorem 24.

### 5. PKE with \(n\)-KDM\(\mathcal{F}_{\text{poly}}\)-CCA Security

#### 5.1. The Basic Idea

We extend the construction of \(n\)-KDM\(\mathcal{F}_{\text{aff}}\)-CCA secure PKE to that of \(n\)-KDM\(\mathcal{F}_{\text{poly}}\)-CCA secure PKE. We allow adversaries to submit polynomial function in \(\mathcal{F}_{\text{poly}}\), in the form of modular arithmetic circuit (MAC) [10], which is a polynomial-sized circuit computing \(f \in \mathcal{F}_{\text{poly}}\). We stress that there is no a priori bound on the size of modular arithmetic circuits. The only requirement is that the degree \(d\) of the polynomials is a priori bounded. We still follow the approach in Figure 1 in our PKE construction. Indeed, we use the same AIAE\(\mathcal{H}_{\text{DHE}}\) and KEM as those in the previous \(n\)-KDM\(\mathcal{F}_{\text{aff}}\)-CCA secure PKE in Figure 8. We only need to construct a new \(\mathcal{S}\) to serve as an entropy filter for the polynomial function set. Moreover, the new \(\mathcal{S}\) should employ the same pair of public and secret keys with KEM. That is, we have \(sk_{i} = (x_{i,1}, y_{i,1}, \ldots, x_{i,d}, y_{i,d})\) and \(pk_{i} = (h_{i,1}, \ldots, h_{i,d})\) with \(h_{i,1} = g_{1}^{x_{i,1}y_{i,1}}, \ldots, h_{i,d} = g_{d}^{x_{i,d}y_{i,d}} \mod N^{d}\), for \(i \in [n]\).

#### 5.2. Reducing Polynomials of \(8n\) Variables to Polynomials of 8 Variables

**How to Reduce \(8n\)-Variable Polynomial \(f_{c}\).** In the \(n\)-KDM\(\mathcal{F}_{\text{poly}}\)-CCA security game, the adversary is allowed to query the encrypt oracle with \((f_{c}, c) \in [n])\) for \(c \in [Q]\). Note that the function \(f_{c}\) is a polynomial in the \(n\) secret keys \((x_{i,j}, y_{i,j})_{i \in [n], j \in [d]}\); thus \(f_{c}\) has \(8n\) variables and is of degree at most \(d\). The bad news is that \(f_{c}\) contains as many as \(\Theta(d^{8n})\) monomial functions. Note that this number can be exponentially large.

The good news is that we found an efficient way to greatly reduce the number of monomials from \(\Theta(d^{8n})\) to \(\Theta(d^{8})\). In particular, the polynomial \(f_{c}(x_{i,j}, y_{i,j})_{i \in [n], j \in [d]}\) can always be changed to a polynomial \(f_{c}'(x_{i,j}, y_{i,j})_{i \in [n], j \in [d]}\) of 8 variables,
consisting of at most \((\frac{8d}{3})\) monomial functions. Now this number is polynomial in \(\lambda\).

The efficient method for reducing the \(8n\)-variable polynomial \(f_\ell\) as follows. In the initialize procedure, \(sk_i\) could be computed as \(x_{ij} = x_i + \overline{x}_{ij}\) and \(y_{ij} = y_j + \overline{y}_{ij} \mod N^2/4\) for \(i \in [n]\) and \(j \in [4]\). By using \((\overline{x}_{ij}, \overline{y}_{ij})e[n],j\in[4]\) \((x_{ij}, y_{ij})e[n],j\in[4]\) could be represented as shifts of \((x_{ij}, y_{ij})e[n],j\in[4]\) that is,

\[
x_{ij} = x_{ij} + \overline{x}_{ij} - \overline{x}_{ij},
\]
\[
y_{ij} = y_{ij} + \overline{y}_{ij} - \overline{y}_{ij}. \tag{57}
\]

Consequently, \(f_\ell\) in \(8n\) variables \((x_{ij}, y_{ij})e[n],j\in[4]\) can be reduced to \(f'_{\ell}\) in \(8\) variables \((x_{ij}, y_{ij})e[4]\); that is,

\[
f_\ell \left((x_{ij}, y_{ij})e[n],j\in[4]\right) = f'_{\ell} \left((x_{ij}, y_{ij})e[4]\right) = \sum_{0 \leq c_1, \ldots, c_n \leq d} a_{(c_1, \ldots, c_n)}
\]
\[
\cdot x_{ij}^c,1 \cdot x_{ij}^c,2 \cdot x_{ij}^c,3 \cdot x_{ij}^c,4.
\]

The degree of the resulting polynomial \(f'_{\ell}\) is still upper bounded by \(d\). Moreover, the coefficients \(a_{(c_1, \ldots, c_n)}\) of \(f'_{\ell}\) are completely determined by the shifts of \((x_{ij}, y_{ij})e[n],j\in[4]\).

5.3. How to Design \(\mathcal{E}\): A Warmup. To illustrate the ideas behind our construction, we take a simple case as consideration: construct \(\mathcal{E}\) for a concrete type of monomial function; that is,

\[
f'_{\ell} \left((x_{ij}, y_{ij})e[4]\right) = a \cdot x_{ij,1}y_{ij,1}x_{ij,2}y_{ij,2}x_{ij,3}y_{ij,3}x_{ij,4}y_{ij,4}. \tag{60}
\]

Algorithms \(\mathcal{E}\). Encrypt and \(\mathcal{E}\). Decrypt are shown in Figure 9.

Security Proof. Now we sketch the proof of KDM-CCA security for this concrete type of monomial functions, that is, \(a \cdot x_{ij,1}y_{ij,1}x_{ij,2}y_{ij,2}x_{ij,3}y_{ij,3}x_{ij,4}y_{ij,4}\). The proof is similar to that for Theorem 24 (cf. Table 2). The only difference lies in games \(G_1-G_4\), which are related to the building block \(\mathcal{E}\). Next, we will replace \(G_1-G_4\) with the following hybrids (i.e., Hybrid 1–Hybrid 3), as shown in Figure 10. Concretely, the \(\mathcal{E}\). Encrypt part of \(\mathcal{E}\) is changed in a computationally indistinguishable way, so that it can serve as an entropy filter for this concrete monomial function, reserving the entropy of \((x_1, y_1, \ldots, x_t, y_t) \mod N\).

Suppose that the adversary submits \((f_\ell, i_\ell) \in [n]\) to the encrypt oracle. Our purpose is to eliminate the use of \((x_1, y_1, \ldots, x_t, y_t) \mod N\) in the computation of \(\mathcal{E}\). Encrypt \((pk_{i_\ell}, f_\ell((x_{ij}, y_{ij})e[n],j\in[4]))\), so the entropy of \((x_{ij}, y_{ij})e[4]\) \(\mod N\) is reserved.

Hybrid 0. In the initialize procedure, the secret keys are computed as \(x_{ij} = x_i + \overline{x}_{ij}\) and \(y_{ij} = y_j + \overline{y}_{ij} \mod N^2/4\) for \(i \in [n], j \in [4]\). This hybrid is identical to \(G_2\) in the proof of Theorem 24.

Hybrid 1. Using \((\overline{x}_{ij}, \overline{y}_{ij})e[n],j\in[4]\), reduce \((f_\ell, i_\ell) \in [n]\) to \((f'_{\ell}, i_\ell) \in [n]\), and calculate the coefficient \(a\) of \(f'_{\ell}\), such that

\[
f'_{\ell} \left((x_{ij}, y_{ij})e[4]\right) = a \cdot x_{ij,1}y_{ij,1}x_{ij,2}y_{ij,2}x_{ij,3}y_{ij,3}x_{ij,4}y_{ij,4}. \tag{61}
\]

Hybrid 2. Implement \(\mathcal{E}\). Encrypt using \(sk_{i_\ell} = (x_{ij}, y_{ij})e[4]\). This hybrid corresponds to \(G_3\) in the proof of Theorem 24.

(i) Invoke \(\mathcal{E}\). Encrypt to set up table.

(ii) Invoke \(\mathcal{E}\). Decrypt to compute \(\overline{v}_0, \ldots, \overline{v}_8\) from table.

(iii) Employ \(\overline{v}_8\) rather than \(\overline{v}_9\) in the computation of \(\overline{v}\), that is, \(\overline{v} = \overline{v}_8 : T'_{\overline{v}}((x_{ij}, y_{ij})e[4]) \mod N\^4\), and compute \(t = g_1^{(x_{ij,1}y_{ij,1})e[4]} \mod N\).

Clearly, \(\overline{v}_0, \ldots, \overline{v}_8\) computed via \(\mathcal{E}\). Decrypt are the same as \(\overline{v}_0, \ldots, \overline{v}_8\) computed via \(\mathcal{E}\). Encrypt. Therefore, this is just a conceptual change.

Hybrid 3. This hybrid corresponds to \(G_4\) in the proof of Theorem 24.

(i) Table is computed similarly as that in \(\mathcal{E}\). Encrypt, except for a small difference. More precisely, in table,
**Figure 9:** \( \mathcal{E} \) designed for a concrete type of monomial functions \( a \cdot x_{i_1}y_{i_2}x_{i_3}y_{i_4} \).  

For \( t \in \{0, 1, \ldots, 8\} \), 

\[
\tilde{r}_{i_1, i_2, i_3, i_4} \leftarrow \left[ \begin{array}{c} \frac{N}{4} \end{array} \right].
\]

\( (\tilde{u}_{i_1, i_2, i_3, i_4}) := \left( \tilde{r}_{i_1, i_2, i_3, i_4}, \tilde{r}_{i_1, i_2, i_3, i_4}, \tilde{r}_{i_1, i_2, i_3, i_4}, \tilde{r}_{i_1, i_2, i_3, i_4} \right) \mod N^t. \)

\( \tilde{e}_l := \tilde{V}_l \cdot T^{m} \mod N^t. \)

\( t := g_1^m \mod N \in \mathbb{Z}_N. \)

Output \( \tilde{e} := (table, \tilde{e}_l). \)

**Figure 10:** Security proof of \( \mathcal{E}.\text{Encrypt} \) as an entropy filter for concrete monomials \( a \cdot x_{i_1}y_{i_2}x_{i_3}y_{i_4} \).  

For \( t \in \{0, 1, \ldots, 8\} \), 

\[
\tilde{r}_{i_1, i_2, i_3, i_4} \leftarrow \left[ \begin{array}{c} \frac{N}{4} \end{array} \right].
\]

\( (\tilde{u}_{i_1, i_2, i_3, i_4}) := \left( \tilde{r}_{i_1, i_2, i_3, i_4}, \tilde{r}_{i_1, i_2, i_3, i_4}, \tilde{r}_{i_1, i_2, i_3, i_4}, \tilde{r}_{i_1, i_2, i_3, i_4} \right) \mod N^t. \)

\( \tilde{v}_l := \tilde{V}_l \cdot T^{m} \mod N^t. \)

\( t := g_1^m \mod N \in \mathbb{Z}_N. \)

Output \( \tilde{e} := (table, \tilde{e}_l). \)
the entry located in row 1 and column 1 is now computed as \( \overline{u}_{1,1} = (\overline{u}_{1,1}, T^d) \cdot \overline{v}_0 \) rather than \( u_{1,1} = \overline{u}_{1,1} \cdot \overline{v}_0 \). By the IV's assumption, this difference is computationally undetectable (see Appendix B for a formal analysis).

(ii) Invoke \( \mathcal{E}.\text{Decrypt} \) to compute \( \overline{v}_0, \ldots, \overline{v}_8 \) from table.

(iii) Compute \( \overline{e} = \overline{v}_8, T^d ((x_{i,4}, y_{i,4})) \mod N^4, \) and \( t := g_1^d ((x_{i,4}, y_{i,4})) \mod N \).

Through a routine calculation, we have \( \overline{v}_0 = \overline{v}_0, \overline{v}_1 = \overline{v}_1, T^{(x_{i,4})} \cdot \overline{v}_2 = \overline{v}_2, T^{(x_{i,4})} \cdot \overline{v}_1 = \overline{v}_1, \overline{v}_3 = \overline{v}_3, T^{(x_{i,4})} \cdot \overline{v}_4 = \overline{v}_4, T^{(x_{i,4})} \cdot \overline{v}_5 = \overline{v}_5, T^{(x_{i,4})} \cdot \overline{v}_6 = \overline{v}_6, T^{(x_{i,4})} \cdot \overline{v}_7 = \overline{v}_7, \) and \( \overline{v}_8 = \overline{v}_8, T^d ((x_{i,4}, y_{i,4})) \mod N \).

Consequently, Hybrid 3 can be implemented in an equivalent way.

**Hybrid 3 (Equivalent Form).** (i) Table is computed similarly as that in \( \mathcal{E}.\text{Encrypt} \), except for a small difference. More precisely, the entry located in row 1 and column 1 in table is now computed at \( u_{1,1} = (\overline{u}_{1,1}, T^d) \cdot \overline{v}_0 \) rather than \( u_{1,1} = \overline{u}_{1,1} \cdot \overline{v}_0 \).

(ii) Compute \( \overline{e} = \overline{v}_8, \mod N^4, \) and \( t := g_1^d ((x_{i,4}, y_{i,4})) \mod N \).

Now \((x_1, y_1, \ldots, x_4, y_4) \mod N\) is not used in \( \mathcal{E}.\text{Encrypt} \) anymore.

After these computationally indistinguishable changes, the \( \mathcal{E}.\text{Encrypt} \) part of the encrypt oracle reserves the entropy of \((x_1, y_1, \ldots, x_4, y_4) \mod N\).

Similarly, we can change the decrypt oracle in a computationally indistinguishable way, so that \((x_1, y_1, \ldots, x_4, y_4) \mod N\) is not involved at all. More precisely, \( \text{Decrypt} \) uses only the \((\text{mod} \phi(N)/4)\) part of secret key and \( \phi(N) \). This change corresponds to \( \mathcal{G}_1 \cdot \mathcal{G}_2 \) in the proof of Theorem 24. Loosely speaking, \( \phi(N) \) is used to ensure that all entries in table are elements in \( \mathbb{Z}_N \). If this is not the case, \( \text{Decrypt} \) rejects immediately. Consequently, the decrypt oracle leaks nothing about \((x_1, y_1, \ldots, x_4, y_4) \mod N\). We can also show the computational indistinguishability of this change, through a similar analysis as that of \( \text{Pr}[\text{Bad}] \) in the proof of Theorem 24.

### 5.4. The General \( \mathcal{E} \) Designed for \( \mathbb{F}_d \)

In Section 5.3, we presented the construction of \( \mathcal{E} \) for a concrete type of monomial functions. Generally, a polynomial function \( f' \) of degree \( d \) might contain as many as \((\binom{n+d}{d}) = \Theta(d^n)\) monomials. In order to construct a general \( \mathcal{E} \) for the set \( \mathbb{F}_d \) of polynomial functions, we must handle all types of monomial functions. To this end, we generate a table for each type of nonconstant monomial and associate it with a \( \overline{v} \), which is named as a title. Algorithms \( \mathcal{E}.\text{Encrypt} \) and \( \mathcal{E}.\text{Decrypt} \) are shown in Figure 11.

Neglecting the coefficients of monomials, there are \((\binom{n+d}{d}) - 1\) types of nonconstant monomial functions whose degrees are at most \( d \). For each nonconstant monomial type \( x_{i,1} x_{i,2} y_{i,1} y_{i,2} x_{i,3} y_{i,3} x_{i,4} y_{i,4} \), we can associate it with a degree tuple \( c = (c_1, \ldots, c_8) \). Let \( \delta \) denote the set of all such degree tuples, that is, \( \delta = \{c = (c_1, \ldots, c_8) \mid 1 \leq c_1 + \cdots + c_8 \leq d\} \).

For each degree tuple \( c = (c_1, \ldots, c_8) \in \delta \), which corresponds to the monomial \( x_{i,1} x_{i,2} y_{i,1} y_{i,2} x_{i,3} y_{i,3} x_{i,4} y_{i,4} \), we generate table\(^{(c)} \) and \( \overline{v}^{(c)} \) by invoking the algorithm TableGen shown in Figure 11. Finally in \( \overline{e}, T^d \) is hidden by the product of all the titles.

Meanwhile, with the help of the secret key \( sk = (x_1, y_1, \ldots, x_4, y_4) \), we can recover \( \overline{v}^{(c)} = \overline{v}^{(c)} \) from table\(^{(c)} \) by invoking the algorithm CalculateV in Figure 11. Thus, the \( \overline{v}^{(c)} \) can always be extracted from (table\(^{(c)} \))ee,δ one by one, and finally \( m \) is recovered.

**Security Proof.** We sketch the proof of KDM[\( \mathbb{F}_d \)]-CCA security for the set of polynomial functions. The proof is also similar to that for Theorem 24 (cf. Table 2). The only difference lies in games \( \mathcal{G}_1 \sim \mathcal{G}_2 \). Next, we will replace \( \mathcal{G}_1 \sim \mathcal{G}_2 \) with the following hybrids (Hybrid 1–Hybrid 3). Specifically, the \( \mathcal{E}.\text{Encrypt} \) part of encrypt is changed in a computationally indistinguishable way, so that it can serve as an entropy filter for polynomial functions of degree at most \( d \), preserving the entropy of \((x_1, y_1, \ldots, x_4, y_4) \mod N\).

Suppose that the adversary submits \((f_{i,k} \in [n]) \) to the encrypt oracle. Our purpose is to eliminate the use of \((x_j, y_j)^i \mod N \) in the computation of \( \mathcal{E}.\text{Encrypt}(pk, f_i((x_j, y_j))_{i \in [n], j \in [d]}) \), so the entropy of \((x_j, y_j)^i \mod N \) is reserved.

**Hybrid 0.** In the initialize procedure, the secret keys are computed as \((x_j, y_j)^i \mod N \) for \((f_{i,k} \in [n]) \) and \((f_{i,k} \in [n]) \) for \((f_{i,k} \in [n]) \). Then compute the coefficients \( a_{(c_1, \ldots, c_8)} \) of \( f' \), as discussed in Section 5.2. Then

\[
\frac{f'_i \left( (x_{i,j}, y_{i,j}) \right)_{j \in [d]}}{a_{(c_1, \ldots, c_8)}} = \frac{x_{i,1} x_{i,2} y_{i,1} y_{i,2} x_{i,3} y_{i,3} x_{i,4} y_{i,4} + \delta}{c}
\]

where \( \delta \) is the constant term \( a_{(0, \ldots, 0)} \) of \( f' \).

**Hybrid 2.** Implement \( \mathcal{E}.\text{Encrypt} \) using \( sk_c = (x_{i,j}, y_{i,j}) \neq [4] \). This hybrid corresponds to \( \mathcal{G}_2 \) in the proof of Theorem 24.

(i) For each \( c = (c_1, \ldots, c_8) \in \delta \)

1. invoke (table\(^{(c)} \), \( \overline{v}^{(c)} \)) \rightarrow 5 TableGen(pk, c),
2. invoke \( \overline{v}^{(c)} \leftarrow \text{CalculateV}(sk_c, \text{table}^{(c)}), c) \).

(ii) Employ \( \overline{v}^{(c)} \neq (\overline{v}^{(c)}) e,\delta \) in the computation of \( \overline{e} \), that is, \( \overline{e} = \sum_{c \in \delta} \overline{v}^{(c)} \cdot T^d ((x_{i,4}, y_{i,4})) \mod N' \), and compute \( t := g_1^d ((x_{i,4}, y_{i,4})) \mod N \).
\[ \mathcal{E} \leftarrow \mathcal{E}.\text{Encrypt}(pk, m) \]:

For each \( c = (c_1, \ldots, c_8) \in \mathcal{G} \) :

\[ (\text{table}^{(c)}, \tilde{\psi}^{(c)}) \leftarrow \text{TableGen}(pk, c) \].

\[ \tilde{\psi} := \prod_{c \in \mathcal{G}} \tilde{\psi}^{(c)} \cdot T^m \mod N^4. \]

\[ t := \tilde{g}^m \mod N \in \mathbb{Z}_N. \]

Output \( \mathcal{E} := ((\text{table}^{(c)}), \tilde{\psi}, t) \).

\[ m/l \leftarrow \mathcal{E}.\text{Decrypt}(sk, \mathcal{E}) \]:

Parse \( \mathcal{E} := ((\text{table}^{(c)}), \tilde{\psi}, t) \).

For each \( c = (c_1, \ldots, c_8) \in \mathcal{G} \) :

\[ \psi^{(c)} \leftarrow \text{CalculateV}(sk, \text{table}^{(c)}, c) \].

If \( \tilde{\psi} \cdot (\prod_{c \in \mathcal{G}} \psi^{(c)})^{-1} \in RU_N \),

\[ m := d \log_T (\tilde{\psi} \cdot (\prod_{c \in \mathcal{G}} \psi^{(c)})^{-1}) \mod N^4. \]

If \( t = \tilde{g}^m \mod N \), Output \( m \).

Otherwise, Output \( l \).

\( \text{TableGen}(pk = (h_1, h_2, h_3, h_4), c = (c_1, \ldots, c_8)) \):

For each \( l \in \{0, 1, \ldots, \sum_{j=1}^8 c_j\} \)

\[ \tilde{u}_{l,1}, \tilde{u}_{l,2}, \tilde{u}_{l,3}, \tilde{u}_{l,4} \leftarrow \left[ \begin{array}{c} N \\ 4 \end{array} \right]. \]

\[ (\tilde{u}_{l,1}, \ldots, \tilde{u}_{l,8}) := (\tilde{r}_{l,1}, \tilde{r}_{l,2}, \tilde{r}_{l,3}, \tilde{r}_{l,4}, \tilde{r}_{l,5}, \tilde{r}_{l,6}, \tilde{r}_{l,7}) \mod N^4. \]

\[ \psi_{c_1} := h_1^{r_{c_1}1} h_2^{r_{c_1}2} h_4^{r_{c_1}4} \mod N^4. \]

Output \( (\text{table}^{(c)}, \tilde{\psi}^{(c)} := \psi_{c_1 \ldots c_8}) \).

\( \text{CalculateV}(sk = (x_1, y_1, x_3, y_4), \text{table}^{(c)}, c = (c_1, \ldots, c_8)) \):

Parse \( \text{table}^{(c)} = \left[ \begin{array}{cccc} u_{1,1} & u_{2,2} & \cdots & u_{8,8} \end{array} \right]_{l \in \{0, 1, \ldots, \sum_{j=1}^8 c_j\}} \).

\[ \psi_0 := u_{0,0}^{-y_0} u_{0,1}^{-y_1} u_{0,2}^{-y_2} u_{0,3}^{-y_3} u_{0,4}^{-y_4} u_{0,5}^{-y_5} u_{0,6}^{-y_6} u_{0,7}^{-y_7} u_{0,8}^{-y_8}. \]

For each \( l \in \{1, \ldots, c_1\} \)

\[ \psi_l := (u_{l,1}^{-y_1} u_{l,2}^{-y_2} u_{l,3}^{-y_3} u_{l,4}^{-y_4} u_{l,5}^{-y_5} u_{l,6}^{-y_6} u_{l,7}^{-y_7} u_{l,8}^{-y_8}). \]

For each \( l \in \{c_1 + 1, \ldots, c_1 + c_2\} \)

\[ \psi_l := (u_{l,1}^{-y_1} u_{l,2}^{-y_2} u_{l,3}^{-y_3} u_{l,4}^{-y_4} u_{l,5}^{-y_5} u_{l,6}^{-y_6} u_{l,7}^{-y_7} u_{l,8}^{-y_8}). \]

For each \( l \in \{\sum_{j=1}^8 c_j + 1, \ldots, \sum_{j=1}^8 c_j + c_3\} \)

\[ \psi_l := (u_{l,1}^{-y_1} u_{l,2}^{-y_2} u_{l,3}^{-y_3} u_{l,4}^{-y_4} u_{l,5}^{-y_5} u_{l,6}^{-y_6} u_{l,7}^{-y_7} (u_{l,8} / \tilde{r}_{l-1})^{-y_8}). \]

Output \( \psi^{(c)} := \psi_{c_1 \ldots c_8} \).

Figure II: (a) \( \mathcal{E}.\text{Encrypt} \) (left) and \( \mathcal{E}.\text{Decrypt} \) (right) of \( \mathcal{E} \) designed for \( \mathcal{F}_d \); (b) \( \text{TableGen} \), which generates \( \text{table}^{(c)} \) together with a title \( \tilde{\psi}^{(c)} \); (c) \( \text{CalculateV} \), which calculates a title \( \psi^{(c)} \) from \( \text{table}^{(c)} \) using secret key.
Clearly, for each \( c = (c_1, \ldots, c_8) \in \mathcal{S} \), \( \vec{v}^{(c)} \) computed via CalculateV is the same as \( \vec{v}^{(c)} \) computed via TableGen. Therefore, this change is just conceptual.

**Hybrid 3.** This hybrid corresponds to \( G_4 \) in the proof of Theorem 24.

(i) For each \( c = (c_1, \ldots, c_8) \in \mathcal{S} \),

1. table\(^{(c)}\) is computed by \((\text{table}^{(c)}, \vec{v}^{(c)}) \leftarrow \_\mathcal{S} \) TableGen\((pk_x, c)\), except for a small difference; more precisely, in table\(^{(c)}\), the entry located in row \( 1 \) and column \( j = \min\{i | 1 \leq i \leq 8, c_i \neq 0\} \) is now computed as \( \vec{u}_{i,j} = (\vec{u}_{i,j}T^{a_{i,j}}) \cdot \vec{v}_0 \) rather than \( \vec{u}_{i,j} = \vec{u}_{i,j} \cdot \vec{v}_0 \); by the IV’s assumption, this difference is computationally undetectable.

2. extract \( \vec{v}^{(c)} \) from the (modified) table\(^{(c)}\) by invoking \( \vec{v}^{(c)} \leftarrow \text{CalculateV}(sk_x, \text{table}^{(c)}, c) \).

(ii) Compute \( \vec{c} = \prod_{c \in \mathcal{S}} \vec{v}^{(c)} \cdot T_{f_1^I(x_{\ell,1}y_{\ell,1}), (c_1, \ldots, c_8)} \mod N \), and \( t \equiv g_1^{f_1^I(x_{\ell,1}y_{\ell,1}), (c_1, \ldots, c_8)} \mod N \).

Through a routine calculation, for each \( c = (c_1, \ldots, c_8) \in \mathcal{S} \), we have

\[
\vec{v}^{(c)} = \vec{v}^{(c)} \cdot T^{a_{\ell,1}a_{\ell,2} \ldots a_{\ell,8}}_{y_{\ell,1}y_{\ell,2} \ldots y_{\ell,8}} \mod N.
\]  

Hence,

\[
\vec{c} = \prod_{c \in \mathcal{S}} \vec{v}^{(c)} \cdot T_{f_1^I(x_{\ell,1}y_{\ell,1}), (c_1, \ldots, c_8)} \mod N
\]

\[
\equiv \prod_{c \in \mathcal{S}} \vec{v}^{(c)} \cdot T_{f_1^I(x_{\ell,1}y_{\ell,1}), (c_1, \ldots, c_8)} \mod N.
\]

Consequently, Hybrid 3 can be implemented in an equivalent way.

**Hybrid 3 (Equivalent Form).** (i) For each \( c = (c_1, \ldots, c_8) \in \mathcal{S} \),

1. table\(^{(c)}\) is computed by \((\text{table}^{(c)}, \vec{v}^{(c)}) \rightarrow \_\mathcal{S} \) TableGen\((pk_x, c)\), except for a small difference. More precisely, in table\(^{(c)}\), the entry located in row \( 1 \) and column \( j = \min\{i | 1 \leq i \leq 8, c_i \neq 0\} \) is now computed as \( \vec{u}_{i,j} = (\vec{u}_{i,j}T^{a_{i,j}}) \cdot \vec{v}_0 \) rather than \( \vec{u}_{i,j} = \vec{u}_{i,j} \cdot \vec{v}_0 \).

2. Compute \( \vec{c} = \prod_{c \in \mathcal{S}} \vec{v}^{(c)} \cdot T^{a_{\ell,1}a_{\ell,2} \ldots a_{\ell,8}}_{y_{\ell,1}y_{\ell,2} \ldots y_{\ell,8}} \mod N \), and \( t \equiv g_1^{f_1^I(x_{\ell,1}y_{\ell,1}), (c_1, \ldots, c_8)} \mod N \).

Now \((x_1, y_1, \ldots, x_4, y_4) \mod N\) is not used in \( \mathcal{A}.\)Encrypt any more.

After these computationally indistinguishable changes, the \( \mathcal{A}.\)Encrypt part of the encrypt oracle reserves the entropy of \((x_1, y_1, \ldots, x_4, y_4) \mod N\).

With a similar argument as that in Section 5.3, we can change the decrypt oracle in a computationally indistinguishable way, so that \((x_j, y_j) \mod N\) is not employed at all.

**Appendix**

**A. Proof of Claim 19**

We build a PPT adversary \( \mathcal{B} \) against the INT-OT security of \( \mathcal{A}.E. \) Suppose that the INT-OT challenger picks a key \( \vec{r} \rightarrow \_\mathcal{G} \) randomly. \( \mathcal{B} \) is given pars\(_{\mathcal{A}.E.} \) and has access to the oracle encrypt\(_{\mathcal{A}.E.}(\cdot) \) for one time.

Firstly, \( \mathcal{B} \) prepares pars\(_{\mathcal{A}.E.} \) in the same way as in \( G_{1,1} \).

That is, invoke \( \text{pars}_{\mathcal{THPS}} \rightarrow \_\mathcal{S} \text{THPS.Setup}(1^T) \), pick \( H \rightarrow \_\mathcal{G} \) randomly, and set \( \text{pars}_{\mathcal{A}.E.} = (\text{pars}_{\mathcal{THPS}}, \text{pars}_{\mathcal{A}.E.}, H) \). \( \mathcal{B} \) sends \( \text{pars}_{\mathcal{A}.E.} \) to \( \mathcal{A}. \). Besides, \( \mathcal{B} \) chooses \( h \leftarrow \_\mathcal{G} \).

As for the \( t \)th \((t \in \{Q_1\})\) encrypt query \((m_t, \text{ai}_t, f_t)\), where \( f_t = (a_t, b_t) \in \_\mathcal{G} \), \( \mathcal{B} \) prepares the challenge ciphertext \((C_t, \chi_t)\) in the following way.

(i) If \( t \in \{j-1\}, \mathcal{B} \) computes \((C_t, \chi_t)\) just like that in \( G_{1,1} \).

That is, \( \mathcal{B} \) picks \( C_t \rightarrow \_\mathcal{G} \) \( \mathcal{V} \) with witness \( w_t \), chooses \( k_t \leftarrow \_\mathcal{G} \), and invokes \( \chi_t \rightarrow \_\mathcal{S} \mathcal{A}.E. \mathcal{E}.\mathcal{N}.r(t) \), and has access to the \( \mathcal{G}_{1,1}^{\mathcal{A}.E.}(\cdot) \) oracle with \( m_t \) and gets the challenge \( \chi_t \).

According to the encrypt\(_{\mathcal{A}.E.}(\cdot) \) oracle, we have \( \chi_t \rightarrow \_\mathcal{S} \mathcal{A}.E. \mathcal{E}.\mathcal{N}.r(t) \). As discussed in the proof of Lemma 18, \( k_t \) is uniformly random in \( \mathcal{G}_{1,1} \). Therefore, the simulation of \( \mathcal{B} \) is the same as that in \( G_{1,1} \).

\( \mathcal{B} \) outputs the challenge ciphertext \((C_t, \chi_t)\) to \( \mathcal{A}. \). Moreover, \( \mathcal{B} \) puts \((\text{ai}_t, f_t, (C_t, \chi_t)) \rightarrow \_\mathcal{S} \mathcal{G}_{\mathcal{A}.E.}, \text{ai}_t, f_t, (C_t, \chi_t) \rightarrow \_\mathcal{S} \mathcal{G}_{\mathcal{A}.E}, \text{and} \ (C_t, \text{ai}_t, f_t) \rightarrow \_\mathcal{S} \mathcal{G}_{\mathcal{A}.E.} \).

Finally, \( \mathcal{B} \) sends a forgery \((\text{ai}^*, f^*, (C^*, \chi^*))\) to \( \mathcal{A}. \), with \( f^* = (a^*, b^*) \in \_\mathcal{G} \). \( \mathcal{B} \) prepares its own forgery with respect to the \( \mathcal{A}.E. \) scheme as follows.

(i) If \((\text{ai}^*, f^*, (C^*, \chi^*)) \in \_\mathcal{S} \mathcal{G}_{\mathcal{A}.E.} \), \( \mathcal{B} \) aborts the game.

(ii) If \((\exists (\text{ai}^*, f^* \in \_\mathcal{G} \mathcal{A}.E.)) \) such that \( \text{ai}^* = \text{ai}^* \) but \( f^* \neq f^* \), \( \mathcal{B} \) aborts the game.

(iii) If \( C^* \neq C^* \), \( \mathcal{B} \) aborts the game.

(iv) \( \mathcal{B} \) computes \( t^* = H(C^*, \text{ai}^*) \in \_\mathcal{G} \).

(v) If \((\exists (C_t, \text{ai}_t, t_t) \in \_\mathcal{S} \mathcal{G}_{\mathcal{A}.E.}) \) such that \( t_t = t^* \) but \((C_t, \text{ai}_t) \neq (C^*, \text{ai}^*), \mathcal{B} \) aborts the game.

(vi) If \( t^* \neq t_t \), \( \mathcal{B} \) aborts the game. If \( t^* = t_t \), \( \mathcal{B} \) outputs \( \chi^* \) to its INT-OT challenger.
We analyze $\mathcal{B}$’s success probability. As discussed in the proof of Lemma 18, the subevent $\text{Forfe} \wedge t_j = t^*$ will imply that $(a^t, \gamma^t, C^t) = (a_j, f_j, C_j)$, $\gamma^t \neq \gamma_j$, $\kappa^t = \kappa_j$, and $\text{AE-Decrypt}(\kappa^t, \gamma^t) \neq 1$. Since $\mathcal{B}$ implicitly sets $k_j = \overline{k}$ as the key used by its challenger, then $\gamma^t \neq \gamma_j$, $\kappa^t = \kappa_j$, and $\text{AE-Decrypt}(\kappa^t, \gamma^t) \neq 1$ implies that $\gamma^t \neq \gamma_j$ and $\text{AE-Decrypt}(\kappa, \gamma^t) \neq 1$; that is, the $\gamma^t$ output by $\mathcal{B}$ is a fresh forgery.

In summary, $\mathcal{B}$ perfectly simulates $G^l_i$ for $\mathcal{A}$ and outputs a fresh forgery as long as the subevent $\text{Forfe} \wedge t_j = t^*$ occurs. Thus, we have that $\Pr[\text{Forfe} \wedge t_j = t^*] \leq \text{Adv}^\text{int-ot}(\lambda)$. This completes the proof of Claim 19.

B. Proof of Indistinguishability between Hybrids 2 and 3 in Section 5.3

To show the indistinguishability between Hybrids 2 and 3, we build a PPT adversary $\mathcal{B}^{\text{chal}_{IV}}(N, g_1, \ldots, g_5)$ to solve the IV$_5$ problem. Firstly, $\mathcal{B}$ generates secret and public keys in $\text{initialize}$ as Hybrid 0 does. When $\mathcal{A}$ submits an encryption query $(f_i, t_i \in [n])$, $\mathcal{B}$ reduces $(f_i, t_i \in [n])$ to $(f_i, t_i \in [n])$ as Hybrid 1 does and obtains the coefficient $a$. Then $\mathcal{B}$ simulates $\mathcal{A}$. Encrypt as follows.

(i) For the 0th row of table, $\mathcal{B}$ computes $(\overline{u}_{0,1}, \ldots, \overline{u}_{0,8})$ and $\overline{v}_0$ as in Hybrids 2 and 3.

(ii) For the 1st row, $\mathcal{B}$ queries its own $\text{chal}_{IV}$ oracle with $(a, 0, *, *, *)$ and obtains its challenge $(\overline{u}_{1,1}, \overline{u}_{1,2}, *, *, *)$; that is,

\[
\text{Case (b = 0): } (\overline{u}_{1,1}', \overline{u}_{1,2}') = (g \overline{f}_1 \gamma_1, g_2 \overline{f}_1) = (\overline{u}_{1,1}, \overline{u}_{1,2}) \\
\text{Case (b = 1): } (\overline{u}_{1,1}', \overline{u}_{1,2}') = (g \overline{f}_1, g_2) = (\overline{u}_{1,1}, \overline{u}_{1,2})
\]

$\mathcal{B}$ sets $\overline{u}_{1,1} = \overline{u}_{1,1} \cdot \overline{v}_0$, which is $\overline{u}_{1,1} \cdot \overline{v}_0$ if $b = 0$ and $\overline{u}_{1,1} = \overline{u}_{1,1} \cdot \overline{v}_0$ if $b = 1$. Then $\mathcal{B}$ generates the remaining elements $(\overline{u}_{1,3}, \ldots, \overline{u}_{1,8})$ in the 1st row of table using its public keys and sets the 1st row of table to be

\[
\overline{u}_{1,1} = \overline{u}_{1,1} \cdot \overline{v}_0 \frac{\overline{u}_{1,2}'}{\overline{u}_{1,2}} \frac{\overline{u}_{1,3}}{\overline{u}_{1,8}}
\]

Thus $\overline{u}_{2,1} = \overline{u}_{2,1} \cdot \overline{v}_1$ in both cases. Then $\mathcal{B}$ generates the remaining elements $(\overline{u}_{2,3}, \ldots, \overline{u}_{2,8})$ in the 2nd row of table using its public keys and sets the 2nd row of table to be

\[
\overline{u}_{2,1} = \overline{u}_{2,1} = \overline{u}_{2,1} \cdot \overline{v}_1 \frac{\overline{u}_{2,3}'}{\overline{u}_{2,3}} \frac{\overline{u}_{2,4}}{\overline{u}_{2,8}}
\]

$\mathcal{B}$ also computes $\overline{v}_2$ from $(\overline{u}_{1,1}', \overline{u}_{1,2}', \overline{u}_{1,3}, \ldots, \overline{u}_{1,8})$ via

\[
\overline{v}_2 = \overline{u}_{1,1} \overline{u}_{1,2} \overline{u}_{1,3} \cdots \overline{u}_{1,8}
\]

which equals

\[
\text{Case (b = 0): } \overline{v}_2 = \overline{v}_1 \\
\text{Case (b = 1): } \overline{v}_2 = \overline{v}_1 T^{-a x_k y_k}
\]

(iv) For the 3rd row, $\mathcal{B}$ queries its own $\text{chal}_{IV}$ oracle with $(*, a \cdot x_i y_i, 0, 0, *, *)$ and obtains its challenge $(*, \overline{u}_{3,1}', \overline{u}_{3,2}, *, *, *)$; that is,

\[
\text{Case (b = 0): } (\overline{u}_{3,1,3}', \overline{u}_{3,4}') = (g_2 \overline{f}_1 \gamma_1, g_2) = (\overline{u}_{3,1}, \overline{u}_{3,4}) \\
\text{Case (b = 1): } (\overline{u}_{3,1}', \overline{u}_{3,4}') = (g_2, 3, T^{x_k y_k} y_k, g_3) = (\overline{u}_{3,1}, T^{x_k y_k} y_k, \overline{u}_{3,4})
\]

$\mathcal{B}$ sets $\overline{u}_{3,3} = \overline{u}_{3,3} \cdot \overline{v}_2$; similarly, it is easy to check that $\overline{u}_{3,3} = \overline{u}_{3,3} \cdot \overline{v}_2$ in both cases. Then $\mathcal{B}$ generates the remaining elements in the 3rd row of table using its public keys and sets the 3rd row of table to be

\[
\overline{u}_{3,1} \frac{\overline{u}_{3,2}}{\overline{u}_{3,3}} = \overline{u}_{3,3} \frac{\overline{v}_2}{\overline{v}_1} \frac{\overline{v}_3}{\overline{v}_4} \frac{\overline{v}_5}{\overline{v}_6} \frac{\overline{v}_7}{\overline{v}_8}
\]

$\mathcal{B}$ also computes $\overline{v}_3$ from $(\overline{u}_{3,1}, \overline{u}_{3,2}, \overline{u}_{3,3}, \overline{u}_{3,4}, \overline{u}_{3,5}, \ldots, \overline{u}_{3,8})$ via $\overline{v}_3 = \overline{u}_{3,1} - \overline{u}_{3,2} - \overline{u}_{3,3} - \overline{u}_{3,4} - \overline{u}_{3,5} - \overline{u}_{3,6} - \overline{u}_{3,7}$, which equals

\[
\text{Case (b = 0): } \overline{v}_3 = \overline{v}_3 \\
\text{Case (b = 1): } \overline{v}_3 = \overline{v}_3 T^{-a x_k y_k}
\]

(v) For the 4–8th rows, $\mathcal{B}$ computes table similarly as above.

(vi) Finally, $\mathcal{B}$ computes $\overline{v}_9, \ldots, \overline{v}_8$ from table, just as in Hybrids 2 and 3 (also as the original $\mathcal{B}$-Decrypt algorithm), and computes $\overline{c} = \overline{v}_9 \cdot T^{f(t'_i x_k y_k _k)} \mod N^2$, $t = g_1(t_i x_k y_k _k) \mod N$ using the secret keys.

If $b = 1$, $\mathcal{B}$ perfectly simulates Hybrid 2. If $b = 0$, $\mathcal{B}$ perfectly simulates Hybrid 3. Any difference between Hybrids 2 and 3 results in $\mathcal{B}$’s advantage over the IV$_5$ problem.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This work was supported by the National Natural Science Foundation of China Grant nos. 61672346 and 61373153.
References


