Novel Inner Product Encryption Resistant to Partial Collusion Attacks

Yuqiao Deng[a], Ge Song[b]

[a]*School of Mathematics And Statistics, Guangdong University of Finance and Economics, Guangzhou, China*
[b]*College of Mathematics And Informatics, South China Agricultural University, Guangzhou, China*

Abstract

Inner product encryption (IPE) is a new cryptographic primitive initially proposed by Abdalla et al. in 2015. IPE can be classified into public-key IPE and secret-key IPE. The currently proposed PK-IPE schemes can-not resist the following collusion attack: an invalid user that holds no private key can "buy" a combined key generated from multiple collusion adversaries, and the user can use this private key to decrypt a ciphertext. Furthermore, the user should not let the collusion adversaries know the ciphertext, and the collusion adversaries should not let the user learn their private keys. We present a new PK-IPE to resist such collusion attack. Our PK-IPE is proven secure under the selective-vector chosen-plaintext attack model. We formalize a selective vector collusion attack model to prove that our scheme is secure under this model.

Keywords:
Inner Product Encryption, Selective Security, Bilinear Paring, TCA, PCA

1. Introduction

Functional encryption (FE) is a versatile cryptographic primitive first formalized by Boneh et al [1]. Since the emergence of FE, an increasing number of studies have considered the construction of generic FE that implements universal circuits [2] [3][4][5].

However, these works have introduced the need for heavy-duty tools, such as indistinguishable obfuscation (IO) and multilinear maps; therefore, the practicality of these works remains questionable. In PKC 2015, Abdalla et al. [6] proposed a new primitive: inner product encryption (IPE). IPE is a special case of FE that executes the computation for the inner product of vectors; it is remarkably useful in applications, such as privacy-preserving statistical analysis and conjunctive normal form / disjunctive normal form formulas.

IPE can be classified into two categories, namely, public key IPE (PK-IPE) [6][7] and secret key IPE (SK-IPE) [8] [9] [10]. In PK-IPE, one vector (i.e., $\vec{x}$) is encrypted by the public key PK, whereas the other vector (i.e., $\vec{y}$) is encoded by the private key SK. A user holds the private key SK as well as the ciphertext of vector $\vec{x}$, the inner product of $\langle \vec{x}, \vec{y} \rangle$ can be derived without knowing any information of vector $\vec{x}$. In SK-IPE, the situation is similar, however, a master secret key (MSK) (i.e., a secret key maintained by the authority) should be used within the generation of the ciphertext and the private key. Thus, *ciphertext cannot be independently formed by an encryptor without an interaction with authority in SK-IPE.* Therefore, SK-IPE is impractical for applications where many users need to generate ciphertext at the same time, i.e., SK-IPE can inevitably influence the performance of the authority.

We believe that only two works are presently focused on PK-IPE [6][7]. Nevertheless, the security built by these two works is insufficient to prevent collusion attacks from malicious users. To demonstrate the collusion attacks that may be realized by adversaries, we provide the following examples.

1.1. Motivation

We briefly review the *KeyDer* algorithm of the simple PK-IPE scheme proposed by Abdalla et al. [6]. (A detailed description of the entire scheme is available in Section 3 of [6]). The algorithm uses a MSK, i.e., a vector $\vec{s} = (s_1, \cdots, s_n)$ to encode the vector $\vec{y} = (y_1, \cdots, y_n)$ into the private key by forming the private key SK as follows: $SK_{\vec{y}} = \langle \vec{y}, \vec{s} \rangle$.

We argue that the aforementioned scheme is vulnerable to trivial collude attacks (TCAs). If ℓ malicious users exist and each of them holds a private key $SK_{\vec{y_i}} = \langle \vec{y_i}, \vec{s} \rangle$, then *they can collude to generate a new private key* with respect to arbitrary vector $\vec{y'}$ as long as $\vec{y'}$ is the linear combination of $\vec{y_1}, \cdots, \vec{y_\ell}$. The attack can be described as follows. Given a private key set

$$\{SK_{\vec{y_i}} = \langle \vec{y_i}, \vec{s} \rangle\}_{i=1,\cdots,\ell}$$

and ℓ integers $k_1, \cdots, k_\ell$, will form a new private key associated with a new vector

$$\vec{y'} = k_1\vec{y_1} + \cdots + k_\ell\vec{y_\ell},$$

the malicious users can simply compute and output the private key

$$SK_{\vec{y'}} = \sum_{i=1}^{\ell} k_i SK_{\vec{y_i}} = \sum_{i=1}^{\ell} k_i\langle\vec{y_i}, \vec{s}\rangle = \langle\sum_{i=1}^{\ell} k_i\vec{y_i}, \vec{s}\rangle = \langle\vec{y'}, \vec{s}\rangle,$$

as desired.

We emphasize that a similar attack can be applied to the two schemes proposed in Section 4 of [7]. Abdalla et al. proposed two IPEs with the same technique for generating private keys to strengthen security. Their *KeyDer* algorithms can be simplified as follows.

Given a vector $\vec{y} = (y_1, \cdots, y_n)$ to generate the private key, the algorithm adopts *two* MSKs, i.e., $\vec{s} = (s_1, \cdots, s_n)$ and $\vec{t} = (t_1, \cdots, t_n)$ to form the private key SK. The algorithm computes and generates

$$SK_{\vec{y}} = (SK_{\vec{y},1}, SK_{\vec{y},2}) = (\langle\vec{y}, \vec{s}\rangle, \langle\vec{y}, \vec{t}\rangle).$$

However, nearly the same collusion attack can be realized to obtain a new private key. For instance, given a private key set

$$SK_{\vec{y_i}} = (SK_{\vec{y_i},1}, SK_{\vec{y_i},2}) = (\langle\vec{y_i}, \vec{s}\rangle, \langle\vec{y_i}, \vec{t}\rangle)$$

for $i = 1, \cdots, l$, and ℓ integers $k_1, \cdots, k_\ell$ as previously described, malicious users form a new private key associated with a new vector

$$\vec{y'} = k_1\vec{y_1} + \cdots + k_\ell\vec{y_\ell}$$

and simply compute and output the private key

$$\begin{aligned} SK_{\vec{y'}} &= (\sum_{i=1}^{\ell} k_i SK_{\vec{y_i},1}, \sum_{i=1}^{\ell} k_i SK_{\vec{y_i},2}) \\ &= \sum_{i=1}^{\ell}(k_i\langle\vec{y_i}, \vec{s}\rangle, k_i\langle\vec{y_i}, \vec{t}\rangle) \\ &= (\langle\vec{y'}, \vec{s}\rangle, \langle\vec{y'}, \vec{t}\rangle). \end{aligned}$$

as desired.

1.2. TCA versus Partial Collusion Attack (PCA)

One may argue that, PK-IPE inherently can-not resist multiple adversaries to collude the decryption of a given ciphertext using a new key that is a linear combination of the keys they hold. The reason may be that supposing collusion users $A_1, \cdots, A_\ell$ hold ℓ private keys, i.e., $\vec{y_1}, \cdots, \vec{y_\ell}$, and they need to decrypt a ciphertext $\vec{x}$ using a new private key $\vec{y'} = k_1\vec{y_1} + \cdots + k_\ell\vec{y_\ell}$, they do not actually need to generate the private key of the vector $\vec{y'}$ because they can run the decrypt algorithm for ℓ times and achieves ℓ results $z_1, \cdots, z_\ell$, i.e.,

$$z_1 = \langle \vec{y_1}, \vec{x} \rangle, \cdots, z_\ell = \langle \vec{y_\ell}, \vec{x} \rangle.$$

They then compute

$$\begin{aligned} z &= k_1 z_1 + \cdots + k_\ell z_\ell \\ &= k_1\langle \vec{y_1}, \vec{x} \rangle + \cdots + k_\ell \langle \vec{y_\ell}, \vec{x} \rangle \\ &= \langle k_1\vec{y_1}, \vec{x} \rangle + \cdots + \langle k_\ell\vec{y_\ell}, \vec{x} \rangle \\ &= \langle k_1\vec{y_1} + \cdots + k_\ell\vec{y_\ell}, \vec{x} \rangle. \\ &= \langle \vec{y'}, \vec{x} \rangle. \end{aligned}$$

According to the preceding analysis, it seems that PK-IPE is vulnerable to the collusion attack. We call the above collusion attack as " *TCA*" in this study. Although TCA is difficult to prevent, it may not be a "satisfactory collusion attack" for "hybrid adversaries" in some scenario. The following scene is provided for example.

We assume that two types of adversaries exist (we call them "hybrid adversaries" in this study); the first type is collusion adversaries $A_1, \cdots, A_\ell$ who hold multiple private keys, and the second type is a solo invalid adversary A' who obtains a ciphertext but holds no private key. A' needs to decrypt the ciphertext, and the ciphertext is so confidential that A' does not like to make it accessible to anyone else. If $A_1, \cdots, A_\ell$ can generate a private key using its own private keys and "sell" this private key to A', then invalid user A' recovers the message, and collusion users $A_1, \cdots, A_\ell$ take the profit. We call this collusion attack PCA.

If A' and $A_1, \cdots, A_\ell$ utilize TCA to collude the decryption of ciphertext, then it is not satisfactory for both of them. The reason is that either A' needs to send the ciphertext to $A_1, \cdots, A_\ell$, and then the message is opened to $A_1, \cdots, A_\ell$; or $A_1, \cdots, A_\ell$ sends all their private keys to A', and then A' may learn the private keys of $A_1, \cdots, A_\ell$. However, if $A_1, \cdots, A_\ell$ can generate a new key, then A' can decrypt the ciphertext without leaking their privacy to the other party. Thus, PCA may be a serious security issue in this scenario .

Therefore, it is still necessary to keep a generic PK-IPE away from PCA although TCA is inevitable. In this study, we establish a security model to capture the behavior of adversary and to construct a new PCA-resistant PK-IPE.

1.3. Our Technique and Contribution

We initially analyze our technique in this study. Intuitively, PCA originates from the homomorphism property of an underlying private key structure. Therefore, how to "break" this homomorphism is a question of both foundational and technical interests. To address this problem, we introduce a new technique for encoding private key vectors.

In the preparation phase, we omit some details and roughly describe the core technique used in our KeyGen algorithm. Given a n-dimensional vector such as $\vec{y} = (y_1, \cdots, y_n)$, and public key components, $(h_1, \cdots, h_n, h, \mathcal{H})$ from group G, the algorithm samples an integer t and then generates private keys as follows: $K_t = g^t, K_h = (\prod_{j=1}^{n} h_j^{y_j} h)^t \mathcal{H}^b$, where the integer b is an MSK of authority. This private key structure is *no longer* homomorphic, because of the fresh random element t, the public key components $h, \mathcal{H}$ and the MSK b.

1.4. Related Work

In PKC 2015, Abdalla et al. [6] proposed a new cryptographic primitive, i.e., IPE. IPE was defined to as an encryption that reveals nothing else except $\langle x, y \rangle$ when decrypting an encrypted vector x with a private key in respond to a vector y. Further more, Abdalla et al. constructed a new PK-IPE and proved its indistinguishability-based security. However, the PK-IPE scheme proposed by [6] was only proven to be selectively secure, and therefore, can-not resist adaptive adversaries. Abdalla et al. further proposed a new methodology to generate PK-IPE against adaptive adversaries in [7]. Moreover, they showed three instantiations generated by this methodology. Although the PK-IPE described in [7] can be proved adaptively secure, it is still vulnerable as the PCA mentioned above.

Another research interest on IPE is the SK-IPE trend. Bishop et al. proposed an IPE providing function-hiding; unlike PK-IPE, their scheme needs to introduce a secret vector during the encrypt procedure [8]. The scheme was proven fully secure in the use of the hybrid argument technique.

Nonetheless, the proof in [8] was insufficient because it introduced a special restriction for the attacker (we refer readers to Section 2.1 in [8] for detailed description). Datta et al. proposed a new SK-IPE to improve the security and eliminate the aforementioned restriction. They employed the technique from [11] and renewed the restriction for attackers (see Formula (2) in [9]). Compared with the restriction by Bishop et al., the restriction by Datta et al. was more practical.

2. Preliminaries

2.1. Bilinear Maps

Assuming $\mathbb{G}, \mathbb{G}_T$ are multiplication cyclic groups with prime order p and e is a map having the following property

1. Bilinearity: for all $u, v \in \mathbb{G}$ and $a, b \in \mathbb{Z}_p$, the following equation holds $e(u^a, v^b) = e(u, v)^{ab}$.
2. Non-degeneracy: $e(g, g) \neq 1$.

2.2. Assumptions

Definition 1(Computation Diffie-Hellman Assumption)[12]Let G be a group of prime order p. Let $x, y \in Z_p$ be selected randomly and g be a generator of G. The adversary is given as g^x, g^y and should compute g^{xy}.

Definition 2(Decisional Diffie-Hellman Assumption (DDH))[12]Let G be a group of prime order p. Let $x, y \in Z_p$ be selected randomly and g be a generator of G. The adversary is given as $\{g, g^x, g^y\}$ and needs to determine if $T = g^{xy}$ or $T = R$, where R denotes a random element in G.

A slight variant of DDH (sv-DDH) is defined and described as follows

Definition 3 (sv-DDH Assumption) Let G be a group of prime order p. Let $x, y, z \in Z_p$ be chosen randomly and g be a generator of G. The adversary is given g, g^x, g^y, g^z and should determine $T = g^{xy}$ or $T = R$, where R denotes a random element in G.

sv-DDH adds a new known term, g^z, to the DDH assumption. However, this new term cannot help the challenger achieve any useful information with respect to T, because z is not related to x and y (or the inherent relations among x, y and z are not revealed). Thus, the term g^z cannot provide any advantage for the challenger to address the assumption. The formal proof for the equivalence between DDH and sv-DDH is straightforward, and it is omitted in this paper.

3. PK-IPE Resistant to PCA

We define a security model to capture the behavior of the partial collusion attackers. We prove that no polynomial adversary can selectively collude to generate even *one* new private key in this model. We validate the security of the entire scheme against the selective-vector chosen-plaintext attack (sv-CPA). We provide the formal definition of IPE and our concrete construction.

3.1. New PK-IPE And Security Model

Definition 4 [PK-IPE [6]]A PK-IPE scheme, denoted as *IPE=(Setup, Encrypt, KeyGen, Decrypt)*, is described as follows.

Setup$(1^\lambda, n)$. The Setup algorithm takes the security parameter λ and the desired length n for the vector as inputs. The output of this algorithm are the public key PK and master secret key MSK.

Encrypt(PK,$\vec{\mathbf{x}}$). The Encrypt algorithm takes the public key PK, a vector $\vec{x}$ as inputs. This algorithm outputs a ciphertext CT.

KeyGen(MSK, $\vec{\mathbf{y}}$). The KeyGen algorithm takes MSK and a vector $\vec{y}$ as inputs. This algorithm outputs the private key SK.

Decrypt(CT,SK). The Decrypt algorithm takes ciphertexts CT that corresponds to vector $\vec{x}$ and private key SK associated with $\vec{y}$ as inputs. This algorithm outputs the inner product $\langle \vec{x}, \vec{y} \rangle$.

Our IPE scheme is secure against the sv-CPA. Unlike the CPA security, our security model requires the adversary to issue its challenge vectors *before* the public key is set by the challenger. The formal definition of security against sv-CPA is presented as follows.

Definition 5 [Security Against sv-CPA [6]]A PK-IPE scheme is secure against sv-CPA if for any polynomial adversary $\mathcal{A}$, the advantage ϵ defined as follows is negligible.

Init. $\mathcal{A}$ issues two challenge vectors $\overrightarrow{x_0}, \overrightarrow{x_1}$ that it attempts to attack.

Setup. The challenger generates and issues public parameters and then sends these parameters to $\mathcal{A}$.

Query phase. $\mathcal{A}$ can query the arbitrary private key associated with vector $\vec{y}$, but with the limitation that $\langle \overrightarrow{x_0}, \vec{y} \rangle = \langle \overrightarrow{x_1}, \vec{y} \rangle$ must hold. *Without this limitation, i.e., existence of some $\vec{y}$ whose private key is queried by the adversary and satisfies $\langle \overrightarrow{x_0}, \vec{y} \rangle \neq \langle \overrightarrow{x_1}, \vec{y} \rangle$, the adversary can simply distinguish one encrypted vector from the other by decrypting the challenge ciphertext with a private key with respect to $\vec{y}$. Then, this adversary can easily confirm which vector is encrypted.*

Guess. The challenger flips a coin and encrypts the challenge vector $\overrightarrow{x_\beta}$ where $\beta = (0, 1)$. This challenger sends this ciphertext to $\mathcal{A}$. $\mathcal{A}$ outputs a guess β' of β.

The advantage of $\mathcal{A}$ in winning the preceding game is defined as $Pr[\beta' = \beta] - 1/2$.

We *initially* formalize the security against a *Selective Vector Collusion Attack (sv-CA)* as follows to guarantee IPE security during attacks from a PCA. Recall that, PCA is such an attack that the collusion adversaries can generate a new private key according to the keys they hold. Hence, the security against sv-CA guarantees that multiple collusion adversaries cannot derive new private key from the keys they hold.

Definition 6 [**sv-CA**] A PK-IPE scheme is secure against sv-CA, if for any polynomial adversary $\mathcal{A}$, the probability that it successfully generates a challenge private key by interacting with the challenger with the following phases is negligible.

Init. $\mathcal{A}$ issues a challenge vector $\vec{y^*}$ that it attempts to attack (that is, the task of $\mathcal{A}$ is to forge the private key as that for vector $\vec{y^*}$).

Setup. The challenger generates and issues public parameters and then sends them to $\mathcal{A}$.

Query phase. $\mathcal{A}$ can query an arbitrary private key that is associated with vector $\vec{y}$, with the only limitation that $\vec{y} \neq \vec{y^*}$.

Forge. After querying for as many private keys as it needs, $\mathcal{A}$ outputs the resulting challenge private key with respect to challenge vector $\vec{y^*}$.

3.2. Our PK-IPE construction

We now present our PK-IPE construction.

Setup$(1^\lambda, n)$. The Setup algorithm takes the security parameter λ, and the desired length n for the vector as inputs. It chooses asymmetric bilinear groups G, G_T, both with prime order p. It also samples random elements $\alpha, a, b \in Z_p$, group elements $h, \mathcal{H} \in G$ and $h_i \in G$ for $i = \{1, \cdots, n\}$. The algorithm outputs the public key as

$$PK = (G, G_T, g, p, e(g,g)^\alpha, g^a, g^b, h, \mathcal{H}, \{h_i\}_{i=1,\cdots,n}).$$

Integers α, a, b are set to be MSK.

Encrypt(PK,$\vec{\mathbf{x}}$). The encrypt algorithm takes the public key PK, and a vector $\vec{x}$ whose length is n (i.e. $\vec{x} = \{x_1, x_2, \cdots, x_n\}$) as inputs. The algorithm then samples two random elements, namely, $r, s \in Z_p$, and generates the following ciphertexts

$$\begin{aligned} C_r &= g^r, \quad C_h = h^r, \quad C_{\mathcal{H}} = \mathcal{H}^r, \\ C_{x,i} &= g^{ax_i} h_i^r : (i = 1, \cdots, n). \end{aligned}$$

The algorithm outputs ciphertext CT as $C = \{(C_{x,1}, \cdots, C_{x,n}), C_r, C_h, C_{\mathcal{H}}\}$.

KeyGen(MSK, $\vec{\mathbf{y}}$). KeyGen algorithm takes MSK α, a, b and a vector $\vec{y}$ as inputs. It selects an element $t \in Z_p$ and for $j = 1, \cdots, n$, and it generates private key SK associated with vector $\vec{y}$ as follows

$$\begin{aligned} K_\alpha &= g^\alpha g^{at}, \quad K_t = g^t, \\ K_h &= (\prod_{j=1}^{n} h_j^{y_j} h)^t \mathcal{H}^b. \end{aligned}$$

This algorithm outputs private key SK=$\{K_\alpha, \vec{\mathbf{y}}, K_h, K_t\}$.

Decrypt(CT,SK). The Decrypt algorithm takes public key PK, ciphertext CT and private key SK as inputs. We note that SK includes the vector $\vec{y} = (y_1, \cdots, y_n)$. Then, the algorithm computes the following

$$\prod_{i=1}^{n} e(C_{x,i}, K_t)^{y_i} = e(g,g)^{at\langle x \cdot y \rangle} \prod_{i=1}^{n} e(g, h_i)^{rty_i}.$$

The algorithm computes the following to cancel the term $\prod_{i=1}^{n} e(g, h_i)^{rty_i}$

$$\begin{aligned} e(C_r, K_h) &= \prod_{i=1}^{n} e(g, h_i)^{rty_i} e(g,h)^{rt} e(g, \mathcal{H})^{rb}, \\ e(C_h, K_t) &= e(g,h)^{rt}, \\ e(g^b, C_{\mathcal{H}}) &= e(g, \mathcal{H})^{rb}. \end{aligned}$$

The algorithm finally derives the term $e(g,g)^{at\langle x \cdot y \rangle}$ through the following computation

$$e(g,g)^{at\langle x \cdot y \rangle} = \prod_{i=1}^{n} e(C_{x,i}, K_t)^{y_i} e(C_r, K_h)^{-1} e(g^b, C_{\mathcal{H}}) e(C_{\mathcal{H}}, K_t).$$

This algorithm also evaluates

$$e(g, K_\alpha)(e(g,g)^\alpha)^{-1} = e(g,g)^{at},$$

Then, it computes $\mathcal{M}$ to satisfy $e(g,g)^{at\mathcal{M}} = e(g,g)^{at\langle x \cdot y \rangle}$. It outputs $\mathcal{M}$ as the plaintext. We can ensure that Decrypt algorithm runs in polynomial time when the algorithm is restricted to check a fixed, polynomial size range of possible values for $\mathcal{M}$ [8], and that it outputs $\perp$ when none of $\mathcal{M}$ satisfies the equation $e(g,g)^{at\mathcal{M}} = e(g,g)^{at\langle x \cdot y \rangle}$.

4. Security Analysis

4.1. Proof for sv-CPA security

In this section, we prove that our scheme is secure against sv-CPA attacks.

Theorem 1 *If the sv-DDH assumption holds true, then our scheme is secure against sv-CPA attacks.*

The challenger $\mathcal{C}$ takes a given tuple $\{g, g^x, g^y, g^z\}$ and a challenge parameter T as inputs. The task of the challenger is to distinguish two terms: $T = g^{xy}$ or $T = R$, where R is a random element in group G.

Init. The adversary $\mathcal{A}$ chooses two challenge vectors, i.e.,

$$\overrightarrow{x_0} = (x_{0,1}, x_{0,2}, \cdots, x_{0,n}), \overrightarrow{x_1} = (x_{1,1}, x_{1,2}, \cdots, x_{1,n}),$$

and these challenger are sent to the challenger $\mathcal{C}$.

Setup. The challenger $\mathcal{C}$ is provided with the security parameter λ and the desired length n for the vector, and it chooses asymmetric bilinear groups G, G_T, both with prime order p. It sets PK as follows: it chooses a generator g of group G, and it *implicitly* sets $\alpha = xz, a = x, b = -z$. Then, it forms $e(g,g)^\alpha = e(g^x, g^z), g^a = g^x, g^b = g^{-z}$. The challenger then chooses two random elements, namely, $h', H' \leftarrow Z_p$ and generates $h = g^{h'}, \mathcal{H} = g^{H'}$.

For every h_i where $i = 1, \cdots, n$, the challenger samples a random element h'_i and sets

$$h_i = g^{x(x_{0,i} - x_{1,i}) + h'_i} = g^{h'_i}(g^x)^{(x_{0,i} - x_{1,i})}$$

The challenger publishes the public parameter:

$$PK = (G, G_T, g, p, e(g,g)^\alpha, g^a, g^b, h, \mathcal{H}, \{h_i\}_{i=1,\cdots,n}).$$

Query Phase 1,2. The challenger is required to generate private keys corresponding to the arbitrary vector, namely, $\vec{\mathbf{y}} = (y_1, y_2, \cdots, y_n)$, with the restriction that $\langle \overrightarrow{x_0}, \vec{\mathbf{y}} \rangle = \langle \overrightarrow{x_1}, \vec{\mathbf{y}} \rangle$. The preceding equation is equivalent to $\langle (\overrightarrow{x_0} - \overrightarrow{x_1}), \vec{\mathbf{y}} \rangle = 0$.

The challenger chooses $t' \in Z_p$ and implicitly sets $t = -z + t'$. Then, it generates $K_t = g^{-z} g^{t'}, K_\alpha = g^\alpha g^{at} = g^{xz} g^{x(-z+t')} = (g^x)^{t'}$.

Generating K_h is complicated because the parameter h_i includes a term g^x and $t = -z + t'$. If we "place" these two components together, then the term g^{xz} which the challenger does not know appears. However, the term g^{xz} is canceled out as follows through the setting of the simulation (recall that we have $\langle (\overrightarrow{x_0} - \overrightarrow{x_1}), \vec{\mathbf{y}} \rangle = 0$).

$$\begin{aligned}
K_h &= (\prod_{j=1}^{n} h_j^{y_j} h)^t \mathcal{H}^b \\
&= (g^{\sum\limits_{j=1}^{n}((x_{0,j}-x_{1,j})x+h'_j)y_j} g^{h'})^{(-z+t')} g^{-zH'} \\
&= g^{\sum\limits_{j=1}^{n}(x_{0,j}-x_{1,j})y_j x(-z+t')} g^{\sum\limits_{j=1}^{n} h'_j y_j(-z+t')} g^{-h'z+h't'} g^{-H'z} \\
&= g^{\langle(\overrightarrow{x_0}-\overrightarrow{x_1}),\vec{\mathbf{y}}\rangle(-z+t')x} g^{\sum\limits_{j=1}^{n} -h'_j y_j z + h'_j y_j t'} g^{-h'z+h't'-H'z} \\
&= (g^z)^{-\sum\limits_{j=1}^{n} h'_j y_j} (g^z)^{-h'-H'} g^{\sum\limits_{j=1}^{n} h'_j y_j t'} g^{h't'}.
\end{aligned}$$

The challenger outputs private key SK=$\{K_t, K_\alpha, \vec{\mathbf{y}}, K_h\}$.

Challenge Ciphertext. The challenger flips a coin $\beta \leftarrow \{0,1\}$ and encrypts the challenge vector $\overrightarrow{\mathbf{x}_\beta} = (x_{\beta,1}, x_{\beta,2}, \cdots, x_{\beta,n})$, as described in the following.

The challenger *implicitly* sets the parameter $r = y$, then it generates $C_r = g^r = g^y$. It also forms $C_{\mathcal{H}} = (\mathcal{H})^r = (g^y)^{H'}$ and $C_h = h^r = (g^y)^{h'}$.

Finally, for $i = 1, \cdots, n$, let $\triangle x_i = x_{0,i} - x_{1,i}$; then, the challenger computes

$$C_{x,i} = (g^x)^{x_{\beta,i}} T^{\triangle x_i} (g^y)^{h'_i}.$$

We assume $T = g^{(xy+\widetilde{r})}$ to explain why the preceding computation is valid. We have

$$\begin{aligned}
C_{x,i} &= (g^x)^{x_{\beta,i}} T^{\triangle x_i} (g^y)^{h'_i} \\
&= (g^x)^{x_{\beta,i}} (g^{xy+\widetilde{r}})^{\triangle x_i} (g^y)^{h'_i} \\
&= (g^x)^{x_{\beta,i}} (g^{x\triangle x_i + h'_i})^y g^{\widetilde{r}\triangle x_i} \\
&= (g^x)^{(x_{\beta,i}+\widetilde{r}\triangle x_i/x)} (g^{x\triangle x_i + h'_i})^y \\
&= g^{a(x_{\beta,i}+\widetilde{r}\triangle x_i/a)} (h_i)^r.
\end{aligned}$$

1) If $\widetilde{r} = 0$, then $T = g^{xy}$, and we have $C_{x,i} = g^{ax_{\beta,i}} {h_i}^r$. This finding indicates that the encrypted vector is $\overrightarrow{\mathbf{x}_\beta}$. Therefore, the challenger successfully simulates the security game with the adversary $\mathcal{A}$.

2) If $\widetilde{r} \neq 0$, then the encrypted vector is unrelated to the challenge vectors $\overrightarrow{\mathbf{x_0}}$ and $\overrightarrow{\mathbf{x_1}}$: the encrypted vector is actually

$$\overrightarrow{\mathbf{x}'} = (x_{\beta,1} + \frac{\widetilde{r}\triangle x_1}{a}, x_{\beta,2} + \frac{\widetilde{r}\triangle x_2}{a}, \cdots, x_{\beta,n} + \frac{\widetilde{r}\triangle x_n}{a}).$$

The adversary cannot obtain any useful information from this challenge ciphertext. Thus, in this situation, the adversary can just ”guess” which vector is encrypted.

Finally, the adversary $\mathcal{A}$ outputs a guess β' of β. If $\beta' = \beta$, then the challenger outputs a bit 1 to indicate that $T = g^{xy}$; otherwise, the challenger outputs a bit 0 to claim that $T = R$.

Apparently, if the adversary has a non-negligible advantage to win the preceding game, then the challenger is also has a non-negligible advantage to solve the *sv*-DDH assumption. Theorem 1 holds.

4.2. Resist PCA

We prove the following theorem to claim that our scheme can resist PCA.

Theorem 2 *If the CDH assumption holds true, then our scheme is secure against sv-CA.*

Proof The challenger $\mathcal{C}$ takes a tuple $\{g, g^x, g^y\}$ as input. Its task is to evaluate g^{xy} by interacting with the adversary.

Init. The adversary $\mathcal{A}$ chooses a challenge vector, i.e., $\overrightarrow{y^*} = (y_1^*, y_2^*, \cdots, y_n^*)$, for which it proves to the challenger that it can generate keys for this vector. Then $\mathcal{A}$ sends the vector to $\mathcal{C}$. If the adversary colludes with multiple users and can achieve as many private key as it needs, then it is subjected to the only limitation that it cannot *directly* obtain the private keys for the challenge vector.

Setup. The challenger $\mathcal{C}$ is given the security parameter λ and the desired length n for the vector. It chooses asymmetric bilinear map G, G_T, both with prime order p. It sets the PK as follows. It chooses a generator g of group G, elements $\alpha', a' \in Z_p$ and sets $\alpha = \alpha'$. Then, it forms $e(g,g)^\alpha = e(g,g)^{\alpha'}, g^a = g^{a'}, g^b = g^y$.

For every h_i, where $i = 1, \cdots, n$, the challenger samples two random elements, namely, k_i, h_i' and sets:

$$h_i = (g^x)^{k_i} g^{h_i'}.$$

Furthermore, the challenger sets $\mathcal{H} = g^x$ and

$$h = (g^x)^{-k_1 y_1^*} (g^x)^{-k_2 y_2^*} \cdots (g^x)^{-k_n y_n^*}.$$

Notice that, h is encoded with the challenge vector $\overrightarrow{y^*}$. The challenger publishes the public parameter as:

$$PK = (G, G_T, g, p, e(g,g)^\alpha, g^a, g^b, h, \mathcal{H}, \{h_i\}_{i=1,\cdots,n}).$$

Querying Phase. The challenger is required to answer the private key queries corresponding to the arbitrary vector, namely $\vec{\mathbf{y}} = (y_1, y_2, \cdots, y_n)$ from the adversary, with the only restriction that $\vec{\mathbf{y}} \neq \vec{\mathbf{y}^*}$.

Let $F(\vec{\mathbf{y}}, \vec{\mathbf{y}^*}) = \sum_{i=1}^{n} k_i(y_i - y_i^*) \mod q$. If $F(\vec{\mathbf{y}}, \vec{\mathbf{y}^*}) = 0$, then challenger outputs $\perp$ and aborts; otherwise, the challenger sets $K_t = g^t = (g^y)^{-\frac{1}{F(\vec{\mathbf{y}}, \vec{\mathbf{y}^*})}}$ and $K_h = \prod_{i=1}^{n} (g^y)^{-\frac{h'_i y_i}{F(\vec{\mathbf{y}}, \vec{\mathbf{y}^*})}}$.

Notice that $k_1, \cdots, k_n$ are chosen randomly and independently from $\vec{\mathbf{y}}$ and $\vec{\mathbf{y}^*}$. Thus, $F(\vec{\mathbf{y}}, \vec{\mathbf{y}^*})$ is distributed randomly as for inputs $\vec{\mathbf{y}}$ and $\vec{\mathbf{y}^*}$. Therefore, $Pr[F(\vec{\mathbf{y}}, \vec{\mathbf{y}^*}) = 0] = 1/q$, which is negligible, i.e., the challenger aborts the game with a negligible probability.

Then, we show that the challenger successfully generates the keys of K_t, K_h. If $F(\vec{\mathbf{y}}, \vec{\mathbf{y}^*}) \neq 0$, then the challenger indeed *implicitly* sets $t = -\frac{1}{F(\vec{\mathbf{y}}, \vec{\mathbf{y}^*})} y$. Thus, we have:

$$\begin{aligned} K_h &= (\prod_{i=1}^{n} h_i^{y_i} h)^t \mathcal{H}^b \\ &= (\prod_{i=1}^{n} (g^{k_i y_i x} g^{h'_i y_i})(g^x)^{\sum_{j=1}^{n} -k_j y_j^*})^{-\frac{1}{F(\vec{\mathbf{y}}, \vec{\mathbf{y}^*})} y} \cdot (g^x)^y \\ &= ((g^x)^{F(\vec{\mathbf{y}}, \vec{\mathbf{y}^*})} \prod_{i=1}^{n} g^{h'_i y_i})^{-\frac{1}{F(\vec{\mathbf{y}}, \vec{\mathbf{y}^*})} y} \cdot (g^x)^y \\ &= \prod_{i=1}^{n} (g^y)^{-\frac{h'_i y_i}{F(\vec{\mathbf{y}}, \vec{\mathbf{y}^*})}}, \end{aligned}$$

as desired.

The challenger outputs private key SK=$\{K_t, K_\alpha, \vec{\mathbf{y}}, K_h\}$.

Challenge Key.

At some point, the adversary outputs the key $\{K_\alpha^*, K_h^*, K_t^*\}$ with respect to the challenge vector $\vec{\mathbf{y}^*}$ with non-negligible probability. The challenger then outputs

$$g^{xy} = K_h^* (K_t^*)^{-h'_i y_i}.$$

This observation holds true as shown follows (assume $K_t^* = g^{t^*}$ for some

$t^* \in Z_p$ that the challenger will not know):

$$\begin{aligned} K_h^* &= (\prod_{i=1}^{n} h_i^{y_i^*} h)^{t^*} \mathcal{H}^b \\ &= (\prod_{i=1}^{n} (g^{k_i y_i^* x} g^{h'_i y_i^*})(g^x)^{\sum_{j=1}^{n} -k_j y_j^*})^{t^*} \cdot (g^x)^y \\ &= (g^{t^*})^{h'_i y_i} g^{xy} \\ &= (K_t^*)^{h'_i y_i} g^{xy}. \end{aligned}$$

Therefore, if the adversary can selectively collude to forge a valid private key, then the challenger can solve the CDH assumption with non-negligible advantage. Theorem 2 holds true.

5. Conclusion

In this paper, we propose a novel IPE against PCA. Our scheme is constructed with bilinear map and is proven secure under the sv-CPA security model. Furthermore, our scheme remains secure when multiple corrupted users still *fail to* form a new private key even when given as many private keys as they need. Our proof is adapted to the selective security framework; however, one can simply modify our construction to achieve the fully security by introducing the dual system encryption technique [13] [14].

References

[1] Dan Boneh, Amit Sahai, and Brent Waters. Functional encryption: Definitions and challenges. In *TCC*, volume 6597 of *Lecture Notes in Computer Science*, pages 253–273. Springer, 2011.

[2] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova 0001, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *FOCS*, pages 40–49. IEEE Computer Society, 2013.

[3] Nuttapong Attrapadung. Dual system encryption via doubly selective security: Framework, fully secure functional encryption for regular languages, and more. In *EUROCRYPT*, volume 8441 of *Lecture Notes in Computer Science*, pages 557–577. Springer, 2014.

[4] Brent Waters. A punctured programming approach to adaptively secure functional encryption. In *CRYPTO*, volume 9216 of *Lecture Notes in Computer Science*, pages 678–697. Springer, 2015.

[5] Gilad Asharov and Gil Segev. Limits on the power of indistinguishability obfuscation and functional encryption. In Venkatesan Guruswami, editor, *FOCS*, pages 191–209. IEEE Computer Society, 2015.

[6] Michel Abdalla, Florian Bourse, Angelo De Caro, and David Pointcheval. Simple functional encryption schemes for inner products. In Jonathan Katz, editor, *Public Key Cryptography*, volume 9020 of *Lecture Notes in Computer Science*, pages 733–751. Springer, 2015.

[7] Michel Abdalla, Florian Bourse, Angelo De Caro, and David Pointcheval. Better security for functional encryption for inner product evaluations. Cryptology ePrint Archive, Report 2016/011, 2016.

[8] Allison Bishop, Abhishek Jain, and Lucas Kowalczyk. Function-hiding inner product encryption. In *ASIACRYPT*, volume 9452 of *Lecture Notes in Computer Science*, pages 470–491. Springer, 2015.

[9] Pratish Datta, Ratna Dutta, and Sourav Mukhopadhyay. Functional encryption for inner product with full function privacy. In *Public Key Cryptography*, volume 9614 of *Lecture Notes in Computer Science*, pages 164–195. Springer, 2016.

[10] Somindu C. Ramanna. More efficient constructions for inner-product encryption. In *ACNS*, volume 9696 of *Lecture Notes in Computer Science*, pages 231–248. Springer, 2016.

[11] Tatsuaki Okamoto and Katsuyuki Takashima. Fully secure functional encryption with general relations from the decisional linear assumption. *IACR Cryptology ePrint Archive*, 2010:563, 2010.

[12] W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(5):644–654, November 1976.

[13] Allison B. Lewko, Tatsuaki Okamoto, Amit Sahai, Katsuyuki Takashima, and Brent Waters. Fully secure functional encryption: Attribute-based encryption and (hierarchical) inner product encryption.

In *EUROCRYPT*, volume 6110 of *Lecture Notes in Computer Science*, pages 62–91. Springer, 2010.

[14] Brent Waters. Dual system encryption: Realizing fully secure IBE and HIBE under simple assumptions. In *CRYPTO*, volume 5677 of *Lecture Notes in Computer Science*, pages 619–636. Springer, 2009.