applied sciences

MDPI

Article

Decentralized Inner-Product Encryption with Constant-Size Ciphertext

Yi-Fan Tseng * and Shih-Jie Gao

Department of Computer Science, National Chengchi University, Taipei 11605, Taiwan; j23793276@gmail.com
* Correspondence: yftseng@cs.nccu.edu.tw

Abstract: With the rise of technology in recent years, more people are studying distributed system architecture, such as the e-government system. The advantage of this architecture is that when a single point of failure occurs, it does not cause the system to be invaded by other attackers, making the entire system more secure. On the other hand, inner product encryption (IPE) provides fine-grained access control, and can be used as a fundamental tool to construct other cryptographic primitives. Lots of studies for IPE have been proposed recently. The first and only existing decentralized IPE was proposed by Michalevsky and Joye in 2018. However, some restrictions in their scheme may make it impractical. First, the ciphertext size is linear to the length of the corresponding attribute vector; second, the number of authorities should be the same as the length of predicate vector. To cope with the aforementioned issues, we design the first decentralized IPE with constant-size ciphertext. The security of our scheme is proven under the ℓ-DBDHE assumption in the random oracle model. Compared with Michalevsky and Joye's work, ours achieves better efficiency in ciphertext length and encryption/decryption cost.

Keywords: inner product encryption; decentralized inner product encryption; constant-size ciphertext

Citation: Tseng, Y.-F.; Gao, S.-J. Decentralized Inner-Product Encryption with Constant-Size Ciphertext. *Appl. Sci.* **2022**, *12*, 636. https://doi.org/10.3390/app12020636

Academic Editor: Gianluca Lax

Received: 5 October 2021
Accepted: 6 January 2022
Published: 10 January 2022

1. Introduction

Identity-based encryption (IBE) was first introduced by Shamir [1] in 1985, which allows a sender to use the recipient's identity to encrypt a message. An identity is a unique string directly linking to a user, e.g., an email address, a student ID number, an employee ID, etc. The first IBE scheme was proposed by Boneh and Franklin [2] in 2001. Though IBE reduces the management cost for traditional public key infrastructures, a drawback of IBE is that an encrypted datum can be only shared at a coarse-grained control level. This may not be suitable in the real world because the sender should know the particular recipient in advance. In a system, there may be a lot of users, and the identities of recipients may be uncertain when a message is encrypted. To solve the issue, Katz, Sahai and Waters [3] conceptualized inner product encryption (IPE) in 2008. In an IPE scheme, each ciphertext is associated with an attribute vector $\vec{Y}$ that can be decrypted by a private key associated with a predicate vector $\vec{X}$ if and only if the inner product of $\vec{X}$ and $\vec{Y}$ is zero, denoted by $< \vec{X}, \vec{Y} >= 0$. IPE can be viewed as the generalization for several cryptographic primitives. For example, given two identities, ID, ID', we can encode it into two vectors, $\vec{X} = (ID, 1), \vec{Y} = (-1, ID')$, and we have

$$ID = ID' \Leftrightarrow < \vec{X}, \vec{Y} >= 0.$$

Thus, we are able to represent the functionality of IBE using IPE. Since then, lots of IPE scheme have been proposed [4–11]. In additional to its theoretical value, IPE provides lots applications in fine-grained access control as well. Using the encoding technique, IPE can be converted into many types of one-to-many encryption, such as broadcast encryption [12–14], attribute-based encryption [15–17] and subset predicate encryption [18–20]. Therefore, by adopting IPE, one can realize multiple kinds of flexible access control using only a single cryptographic primitive. Recently, more applications for

IPE have been developed, e.g., privacy-preserving video streaming [21], access control for WBAN [22], secure keyword searching [23] and outsourced data integration [24]. It shows the possibility for the application of IPE in various environments.

Traditionally, IPE is a centralized architecture, which needs a trusted server to issue private keys for all users. However, a centralized paradigm may not be practical in a real-world environment. In practice, the privileges of a user are usually given by different authorities. In addition, a centralized architecture would suffer from the problem of a single point of failure. To cope with these problems, Michalevsky and Joye gave the first Decentralized IPE (DIPE) scheme [25] in 2018. In a DIPE scheme, there are multiple authorities. For a user, each authority will output a partial private key for this user, without interaction with each other.

After studying the DIPE scheme of Michalevsky and Joye, we found two problems. One problem is the large ciphertext size. In their scheme, the ciphertext size is $\mathcal{O}(nk)$ group elements, where n is the length of attribute/predicate vector and k is the parameter of k-linear assumption. Since k can be viewed as a part of the security parameter, which is a constant, the ciphertext size is linear to the length of attribute/predicate vector. Another problem is that, in their scheme, each authority is responsible for issuing a private key for only an element in the user's predicate vector. This setting brings two disadvantages. First, unlike to decentralized attribute-based encryption [26–28], where the attributes of a user is independent to each other, the elements in a predicate vector for a user are usually closely bonded. Second, since each authority issues a partial private key for one element in a predicate vector, the number of authorities must equal to the length of predicate vector, which may not be practical, i.e., in the scheme of [25], an authority cannot responsible for multiple attributes, which is common in practice.

1.1. Contribution

In this manuscript, we propose a novel DIPE scheme with constant-size ciphertexts, and we give a formal security proof for the selective IND-CPA security under q-DBDHE assumption. We also modify the way an authority produces private keys from predicate vectors due to the aforementioned issue. In addition, we implement our construction in Python with Charm-Crypto library and C with PBC library to evaluate the performance.

1.2. Organization

In Section 2, we introduce the notations and complexity assumption used in our manuscript, and the definition of decentralized inner product encryption. The security of DIPE is defined in Section 2, as well. In Section 3, we describe our proposed scheme in detail and show the correctness. In Section 4, we give the formal security proof for our scheme. In Section 5, we show the comparison results between our scheme and the DIPE scheme in [25]. Finally, we conclude our work in Section 6.

2. Preliminaries

In this section, we introduce the definition and security requirements of decentralized inner product encryption. In addition, we demonstrate the notation and complexity assumption used in our work.

2.1. Notation

Given a set S, "randomly choose an element x from the set S" is denoted as $x \xleftarrow{\$} S$. For algorithm A, we write $x \leftarrow A$ to denote "x is the output by running A". The symbol "$\perp$" means a failed decryption that recovers the certain message unsuccessfully. "PPT" algorithm means "probabilistic polynomial time" algorithm that can run in polynomial-bounded time.

2.2. Bilinear Maps and Complexity Assumption

Let $\mathbb{G}$ and $\mathbb{G}_T$ be two multiplicative cyclic groups with prime order p. A map e is called a bilinear map if the following properties hold:

1. **Bilinearity**: For $u, v \in \mathbb{G}$, and $a, b \in \mathbb{Z}_p$, the equation $e(u^a, v^b) = e(u, v)^{ab}$ holds.
2. **Non-Degeneracy**: Assume g is the generator of $\mathbb{G}$, then, $e(g, g) \neq 1$.
3. **Computability**: For $u, v \in \mathbb{G}$, there exists an efficient algorithm to compute $e(u, v)$.

Next, we show the complexity assumption, the ℓ-decisional bilinear Diffie–Hellman exponent (ℓ-DBDHE) assumption [29,30], which the security of our scheme based on.

Definition 1 (The ℓ-Decisional Bilinear Diffie–Hellman Exponent Problem)**.** *Let $\mathbb{G}$ be a group. g is a generator of $\mathbb{G}$, and $\gamma, s \xleftarrow{\$} \mathbb{Z}_p$ are two integers. Given a tuple:*

$$(g, g^{\gamma}, g^{\gamma^2}, \ldots, g^{\gamma^{\ell}}, g^{\gamma^{\ell+2}}, \ldots, g^{\gamma^{2\ell}}, g^{s}, T),$$

decide if $T = e(g, g)^{\gamma^{\ell+1} s}$ or $T \xleftarrow{\$} \mathbb{G}_T$ is a random element of $\mathbb{G}_T$.

Let $T_0 = (g, g^{\gamma}, g^{\gamma^2}, \ldots, g^{\gamma^{\ell}}, g^{\gamma^{\ell+2}}, \ldots, g^{\gamma^{2\ell}}, g^{s})$. For an algorithm $\mathcal{A}$, the advantage of $\mathcal{A}$ in solving the ℓ-DBDHE problem is defined as:

$$\mathsf{Adv}_{\mathcal{A}}^{\ell\text{-DBDHE}} = \left| \Pr[\mathcal{A}(T_0, T = e(g, g)^{\gamma^{\ell+1} s}) = 1] - \Pr[\mathcal{A}(T_0, T \xleftarrow{\$} \mathbb{G}_T) = 1] \right|.$$

Definition 2 (The ℓ-Decisional Bilinear Diffie–Hellman Exponent Assumption)**.** *We say that the ℓ-decisional bilinear Diffie–Hellman exponent assumption holds if for all PPT algorithms, $\mathsf{Adv}_{\mathcal{A}}^{\ell\text{-DBDHE}}$ is negligible.*

2.3. Definition of Decentralized Inner Product Encryption

The difference between DIPE and IPE is that a private key of DIPE is generated by multiple authorities, while a private key of IPE is generated by a centralized authority.

2.3.1. System Model

A DIPE scheme contains three roles, i.e., sender, receiver and authorities. A sender is a participant of the system who transfers the encrypted data to the receiver. The data are encrypted by an attribute vector before delivered to receiver. Authorities are responsible for issuing partial keys for receivers who make a request to obtain partial keys. The authorities will issue partial keys according to the predicate vector of the receiver. A receiver is a participant who wants to receive encrypted data. After a receiver receives all the partial keys from the authorities, the receiver will perform a decryption procedure to recover the data.

2.3.2. Definition of DIPE

A decentralized inner product encryption scheme consists of five PPT algorithms: **Setup**, **AuthSetup**, $\mathbf{KeyGen}_{\mathbf{A_i}}$, **Encrypt** and **Decrypt**. Unlike the single authority construction, in DIPE, the private key of a user is generated by multiple authorities. Each authority $\mathbf{A}_i$ computes a "partial key sk_i" of a user using its master secret key and the user's predicate vector. The full private key of a user is $\{sk_i\}_{i=1,\ldots,n}$, where n is the number of authorities:

- $Setup(1^{\lambda})$. An authority in the system or a third party will run the algorithm. Taking as input a security parameter 1^{λ}, the algorithm outputs a public parameter pp.
- $AuthSetup(pp, i)$. All authorities will run the algorithm. Taking as inputs a public parameter pp, and a number i, the algorithm outputs a master secret key MSK_i and a public key PK_i of each authority, where i is the index of authority.
- $KeyGen_{A_i}(pp, MSK_i, GID, \vec{X})$. All authorities will run the algorithm. Taking as inputs a public parameter pp, a master secret key MSK_i, a global identity GID and a predicate vector $\vec{X}$, the algorithm outputs a partial key of the private key associated with $\vec{X}$ generated by i_{th} authority. Note that the description of $\vec{X}$ will be included in the partial keys.
- $Encrypt(pp, \{PK_i\}_{i=1,\ldots,n}, M, \vec{Y})$. A sender will run the algorithm. Taking as inputs a public parameter pp, all the public keys of each authority $\{PK_i\}_{i=1,\ldots,n}$, a message

M and an attribute vector $\vec{Y}$, the algorithm outputs a ciphertext C associated with $\vec{Y}$. Note that the description of $\vec{Y}$ will be included in the ciphertext.

- $Decrypt(\{sk_i\}_{i=1,\ldots,n}, C)$. A receiver will run the algorithm. Taking as inputs all the partial key of private keys of each authority $\{sk_i\}_{i=1,\ldots,n}$, a ciphertext C and an attribute vector $\vec{Y}$, the algorithm outputs a message M or $\perp$.
  Correctness. For $pp \leftarrow Setup(1^\lambda)$, $(PK_i, MSK_i) \leftarrow AuthSetup(pp, i)$, $sk_i \leftarrow KeyGen_{A_i}(pp, MSK_i, GID, \vec{X})$, $C \leftarrow Encrypt(pp, \{PK_i\}_{i=1,\ldots,n}, M, \vec{Y})$, where $i = 1, \ldots, n$, we have that:
  - If $\langle \vec{X}, \vec{Y} \rangle = 0$, then $Decrypt(\{sk_i\}_{i=1,\ldots,n}, C) = M$.
  - If $\langle \vec{X}, \vec{Y} \rangle \neq 0$, then $Decrypt(\{sk_i\}_{i=1,\ldots,n}, C) = \perp$.

2.3.3. Security Model

The security definition used in our manuscript is the security against indistinguishability under selective chosen-plaintext attacks (sIND-CPA). "Indistinguishability" means that given a ciphertext, which is the encryption of one of two messages chosen by an adversary, the adversary tries to tell which of the two messages is encrypted. In addition, "chosen-plaintext attacks" means that an adversary is allowed to obtain the ciphertext for the plaintext of its choice. Finally, "selective" means that an adversary chooses a target vector and submits to the challenger before Setup phase.

Definition 3 (The sIND-CPA Security)**.** *Let $\mathcal{A}$ be a probabilistic polynomial-time adversary. We define our security via the following interactive game between $\mathcal{A}$ and a challenger $\mathcal{C}$:*

- ***Initialization***.
  $\mathcal{A}$ chooses an attribute vector $\vec{Y}^* = (y_1^*, y_2^*, \ldots, y_\ell^*)$ and sends $\vec{Y}^*$ to $\mathcal{C}$.
- ***Setup***.
  $\mathcal{C}$ runs the Setup algorithm to generate PK_i and MSK_i, where $1 \leq i \leq n$, is the index of authority. $\mathcal{C}$ sends $PK_1, \ldots, PK_n$ and $MSK_1, \ldots, MSK_{n-1}$ to $\mathcal{A}$.
- ***Phase1***.
  $\mathcal{A}$ can make polynomially times queries of the following oracle.
  - KeyExtract oracle: $\mathcal{A}$ sends a predicate vector $\vec{X}$ and a global identity GID to $\mathcal{C}$, and $\mathcal{C}$ returns the private key of $\vec{X}$. There is a restriction, that is, $\langle \vec{X}, \vec{Y}^* \rangle \neq 0$.
- ***Challenge***.
  $\mathcal{A}$ submits two distinct messages M_0, M_1 of the same length to $\mathcal{C}$. $\mathcal{C}$ then randomly chooses $\beta \in \{0, 1\}$ and generates ciphertexts $C^* = Encrypt(pp, \{PK_i\}_{i=1,\ldots,n}, M_\beta, \vec{Y}^*)$. Then, $\mathcal{C}$ sends C^* to $\mathcal{A}$.
- ***Phase2***.
  Same as Phase1.
- ***Guess***.
  $\mathcal{A}$ will output a bit $\beta' \in \{0, 1\}$ and win the game if $\beta' = \beta$.
  The advantage of $\mathcal{A}$ winning the game is defined as:

$$\mathbf{Adv}_{\mathcal{A}}^{\mathsf{sIND\text{-}CPA}} = \left| Pr[\beta' = \beta] - \frac{1}{2} \right|.$$

A DIPE scheme is sIND-CPA secure if for all PPT adversaries $\mathcal{A}$, $\mathbf{Adv}_{\mathcal{A}}^{\mathsf{sIND\text{-}CPA}}$ is negligible.

3. The Proposed Scheme

In this section, we present our decentralized inner product encryption scheme with constant-size ciphertexts. The notations used in the proposed scheme are defined in Table 1.

Table 1. Notations.

Notation	Description
$\mathbb{G}$	a bilinear group with prime order p
$\mathbb{G}_T$	a bilinear group by pairing of the element of $\mathbb{G}$
e	a bilinear mapping; $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$
g	a generator of $\mathbb{G}$
n	total number of authorities
ℓ	the length of predicate/attribute vector
A_i	ith authority
pp	public parameter
PK_i	public key of authority i
MSK_i	master secret key of authority i
$\vec{X}$	a predicate vector
$\vec{Y}$	an attribute vector
GID	an identity of a receiver
M	a message

Setup(1^λ)

The algorithm performs the following steps:

1. Randomly choose bilinear groups $\mathbb{G}, \mathbb{G}_T$ of prime order p with a generator $g \xleftarrow{\$} \mathbb{G}$;
2. Choose an one-way hash function, $H : \{0,1\}^* \times \mathbb{Z}_p^\ell \rightarrow \mathbb{G}$;
3. Output the public parameter $pp = \{g, H\}$.

AuthSetup(pp, i)

Each authority A_i in the system performs the following steps to generate its public key and its master secret key:

1. Choose $\overline{\alpha}_i \xleftarrow{\$} \mathbb{Z}_p$;
2. Choose $\alpha_{0,i} \xleftarrow{\$} \mathbb{Z}_p$;
3. Choose $\alpha_{1,i}, \alpha_{2,i}, \ldots, \alpha_{\ell,i} \xleftarrow{\$} \mathbb{Z}_p$;
4. Output a public key of authority i, $PK_i = \{g^{\alpha_{0,i}}, g^{\alpha_{1,i}}, \ldots, g^{\alpha_{\ell,i}}, Z_i = e(g,g)^{\overline{\alpha}_i}\}$;
5. Output a master secret key of authority A_i, $MSK_i = \{g^{\overline{\alpha}_i}, \alpha_{0,i}, \alpha_{1,i}, \ldots, \alpha_{\ell,i}\}$.

KeyGen$_{\mathbf{A_i}}(pp, MSK_i, GID, \vec{X} = (x_1, \cdots, x_\ell))$

Each authority A_i in the system performs the following steps to generate a part of private key for receivers in the system"

1. Return failure symbol $\perp$ if $x_1 = 0$;
2. Output the private key $sk_i = \{D_0, D_{1,i}, \{K_{j,i}\}_{j=2,\ldots,\ell}\}$, where

$$D_0 = H(GID, \vec{X})$$
$$D_{1,i} = g^{\overline{\alpha}_i} \cdot H(GID, \vec{X})^{\alpha_{0,i}}$$
$$\{K_{j,i} = H(GID, \vec{X})^{-\alpha_{1,i}\frac{x_j}{x_1}} \cdot H(GID, \vec{X})^{\alpha_{j,i}}\}_{j=2,\ldots,\ell}.$$

Unlike the **KeyGen** algorithm in [25], we use the entire predicate vector $\vec{X}$ in **KeyGen**$_{\mathbf{A_i}}$ performed by a single authority A_i.

Encrypt$(pp, \{PK_i\}_{i=1,\ldots,n}, M, \vec{Y} = (y_1, \ldots, y_\ell))$

A sender computes the ciphertext for a message $M \in \mathbb{G}_T$ and an attribute vector $\vec{Y} = (y_1, \ldots, y_\ell)$ by the following steps:

1. Choose $s \xleftarrow{\$} \mathbb{Z}_p$;
2. Output the ciphertexts as $C = \{E_0, E_1, E_2\}$, where

$$E_0 = M \cdot (\prod_{i=1}^{n} Z_i)^s$$
$$E_1 = ((\prod_{i=1}^{n} g^{\alpha_{0,i}}) \cdot (\prod_{i=1}^{n} g^{\alpha_{1,i}})^{y_1} \cdot \ldots \cdot (\prod_{i=1}^{n} g^{\alpha_{\ell,i}})^{y_\ell})^s$$
$$E_2 = g^s.$$

Decrypt$(\{sk_i\}_{i=1,\ldots,n}, C)$

To decrypt, a receiver uses the private key $\{sk_i\}_{i=1,\ldots,n}$ to recover the message M from a ciphertext C as follows:

1. If $\langle \vec{X}, \vec{Y} \rangle = 0$, perform the following computation; otherwise, return $\perp$;
2. Compute

$$(\prod_{i=1}^{n} Z_i)^s = \left(\prod_{i=1}^{n} e(g,g)^{\overline{\alpha}_i}\right)^s = \frac{e((\prod_{i=1}^{n} D_{1,i}) \cdot (\prod_{i=1}^{n} K_{2,i})^{y_2} \ldots (\prod_{i=1}^{n} K_{\ell,i})^{y_\ell}, E_2)}{e(E_1, D_0)}.$$

3. Compute $M = E_0 / (\prod_{i=1}^{n} Z_i)^s$.

Correctness.

The correctness of the decryption algorithm is described as follows. For convenience, let $\overline{\alpha} = \sum_{i=1}^{n} \overline{\alpha}_i, \alpha_j = \sum_{i=1}^{n} \alpha_{j,i}$, for $j = 0, \ldots, \ell$. It is enough to show that

$$e(g,g)^{\overline{\alpha}_0 s} = \left(\prod_{i=1}^{n} e(g,g)^{\overline{\alpha}_i}\right)^s = \frac{e((\prod_{i=1}^{n} D_{1,i}) \cdot (\prod_{i=1}^{n} K_{2,i})^{y_2} \ldots (\prod_{i=1}^{n} K_{\ell,i})^{y_\ell}, E_2)}{e(E_1, D_0)}.$$

We first take a look at the numerator:

$$\begin{aligned}
\prod_{i=1}^{n} D_{1,i} &= \prod_{i=1}^{n} g^{\overline{\alpha}_i} \cdot H(GID, \vec{X})^{\alpha_{0,i}} = g^{\sum_{i=1}^{n} \overline{\alpha}_i} \cdot H(GID, \vec{X})^{\sum_{i=1}^{n} \alpha_{0,i}} = g^{\overline{\alpha}} \cdot H(GID, \vec{X})^{\alpha_0} \\
\prod_{i=1}^{n} K_{j,i} &= \prod_{i=1}^{n} H(GID, \vec{X})^{-\alpha_{1,i} \frac{x_j}{x_1}} \cdot H(GID, \vec{X})^{\alpha_{j,i}} = H(GID, \vec{X})^{\frac{-x_j}{x_1} \sum_{i=1}^{n} \alpha_{1,i} + \sum_{i=1}^{n} \alpha_{j,i}} \\
&= H(GID, \vec{X})^{\alpha_1 (\frac{-x_j}{x_1}) + \alpha_j},
\end{aligned}$$

where $j = 2, \ldots, \ell$. Using the fact that

$$\langle \vec{X}, \vec{Y} \rangle = 0 \Leftrightarrow \sum_{i=1}^{\ell} x_i y_i = 0 \Leftrightarrow y_1 = \frac{\sum_{i=2}^{\ell} (-x_i y_i)}{x_1},$$

we have:

$$\begin{aligned}
& (\prod_{i=1}^{n} D_{1,i}) \cdot (\prod_{i=1}^{n} K_{2,i})^{y_2} \ldots (\prod_{i=1}^{n} K_{\ell,i})^{y_\ell} \\
= \; & g^{\overline{\alpha}} \cdot H(GID, \vec{X})^{\alpha_0} \cdot (H(GID, \vec{X})^{\alpha_1 (\frac{-x_2}{x_1}) + \alpha_2})^{y_2} \cdot \ldots \cdot (H(GID, \vec{X})^{\alpha_1 (\frac{-x_\ell}{x_1}) + \alpha_\ell})^{y_\ell} \\
= \; & g^{\overline{\alpha}} \cdot H(GID, \vec{X})^{\alpha_0} \cdot (H(GID, \vec{X})^{\alpha_1 \frac{\sum_{i=2}^{\ell} (-x_i y_i)}{x_1} + \sum_{i=2}^{\ell} \alpha_i y_i} \\
= \; & g^{\overline{\alpha}} \cdot H(GID, \vec{X})^{\alpha_0} \cdot H(GID, \vec{X})^{\alpha_1 y_1 + \sum_{i=2}^{\ell} \alpha_i y_i} \\
= \; & g^{\overline{\alpha}} \cdot H(GID, \vec{X})^{\alpha_0} \cdot H(GID, \vec{X})^{\sum_{i=1}^{\ell} \alpha_i y_i}.
\end{aligned}$$

Thus, the numerator is:

$$\begin{aligned}
& e(g^{\overline{\alpha}} \cdot H(GID, \vec{X})^{\alpha_0} \cdot H(GID, \vec{X})^{\sum_{i=1}^{\ell} \alpha_i y_i}, E_2) \\
= \; & e(g^{\overline{\alpha}} \cdot H(GID, \vec{X})^{\alpha_0} \cdot H(GID, \vec{X})^{\sum_{i=1}^{\ell} \alpha_i y_i}, g^s) \\
= \; & e(g,g)^{\overline{\alpha} s} \cdot e(H(GID, \vec{X}), g)^{(\alpha_0 + \sum_{i=1}^{\ell} \alpha_i y_i) s}.
\end{aligned}$$

In addition, the denominator is:

$$\begin{aligned} & e(E_1, D_0) \\ = \; & e(((\prod_{i=1}^{n} g^{\alpha_{0,i}}) \cdot (\prod_{i=1}^{n} g^{\alpha_{1,i}})^{y_1} \cdot \ldots \cdot (\prod_{i=1}^{n} g^{\alpha_{\ell,i}})^{y_\ell})^s, H(GID, \vec{X})) \\ = \; & e(g^{\sum_{i=1}^{n} \alpha_{0,j}} \cdot g^{y_1 \sum_{i=1}^{n} \alpha_{1,j}} \cdot \ldots \cdot g^{y_\ell \sum_{i=1}^{n} \alpha_{\ell,j}}, H(GID, \vec{X}))^s \\ = \; & e(g^{\alpha_0} \cdot g^{\alpha_1 y_1} \cdot \ldots \cdot g^{\alpha_\ell y_\ell}, H(GID, \vec{X}))^s \\ = \; & e(g^{\alpha_0 + \sum_{i=1}^{\ell} \alpha_i y_i}, H(GID, \vec{X}))^s \\ = \; & e(H(GID, \vec{X}), g)^{(\alpha_0 + \sum_{i=1}^{\ell} \alpha_i y_i)s}. \end{aligned}$$

Finally, we have:

$$\frac{\text{numerator}}{\text{denominator}} = \frac{e(g,g)^{\overline{\alpha} s} \cdot e(H(GID, \vec{X}), g)^{(\alpha_0 + \sum_{i=1}^{\ell} \alpha_i y_i)s}}{e(H(GID, \vec{X}), g)^{(\alpha_0 + \sum_{i=1}^{\ell} \alpha_i y_i)s}} = e(g,g)^{\overline{\alpha} s}.$$

4. Security Proof

In this section, we will prove the sIND-CPA security for the proposed under the ℓ-DBDHE assumption in the random oracle model.

Theorem 1. *The proposed DIPE scheme is sIND-CPA secure if the q-DBDHE assumption holds.*

Proof. Assume there is a polynomial-time adversary that can win the sIND-CPA game with a non-negligible advantage. Then, we construct a PPT challenger $\mathcal{C}$ able to solve the ℓ-DBDHE problem as follows:

First of all, $\mathcal{C}$ is given an instance of the q-DBDHE problem, that is,

$$\left(g, g^{\gamma}, g^{\gamma^2}, \ldots, g^{\gamma^{\ell}}, g^{\gamma^{\ell+2}}, \ldots, g^{\gamma^{2\ell}}, g^s, T\right),$$

where T is $e(g,g)^{\gamma^{\ell+1} s}$ or a random element of $\mathbb{G}_T$. Then, $\mathcal{C}$ interacts with $\mathcal{A}$ in the game as follows.

Initialization.

$\mathcal{A}$ first sends the target vector $\vec{Y}^* = (y_1^*, y_2^*, \ldots, y_\ell^*)$ to $\mathcal{C}$.

Setup.

Without loss of generality, we may assume that $\mathcal{A}$ can obtain the first $n-1$ master secret keys MSK_i of authorities, where $i = 1, \ldots, n-1$:

1. Set $(g^{\alpha_{1,n}}, g^{\alpha_{2,n}}, \ldots, g^{\alpha_{\ell,n}}) = (g^{\gamma}, g^{\gamma^2}, \ldots, g^{\gamma^{\ell}})$. Define $\vec{\alpha}_n = \langle \alpha_{1,n}, \alpha_{2,n}, \ldots, \alpha_{\ell,n} \rangle$;
2. Choose $\delta \xleftarrow{\$} \mathbb{Z}_p$;
3. Compute $Z_n = e(g,g)^{\overline{\alpha}_n} = e(g^{\gamma}, g^{\gamma^{\ell}})$ and $g^{\alpha_{0,n}} = \left((g^{\gamma})^{y_1^*} (g^{\gamma^2})^{y_2^*} \ldots (g^{\gamma^{\ell}})^{y_\ell^*}\right)^{-1} \cdot g^{\delta}$;
4. For $i = 1, \ldots, n-1$, $\mathcal{C}$, compute PK_i and MSK_i following the **AuthSetup**(pp, i) shown in Section 3;
5. Send to $\mathcal{A}$ the public keys $\{PK_i\}_{i=1,\ldots,n} = \{g^{\alpha_{0,i}}, g^{\alpha_{1,i}}, \ldots, g^{\alpha_{\ell,i}}, Z_i = e(g,g)^{\overline{\alpha}_i}\}_{i=1,\ldots,n}$, and the master secret key s $\{MSK_i\}_{i=1,\ldots,n-1} = \{g^{\overline{\alpha}_i}, \alpha_{0,i}, \alpha_{1,i}, \ldots, \alpha_{\ell,i}\}_{i=1,\ldots,n-1}$.

Here, we implicitly set

$$\overline{\alpha}_n = \gamma^{\ell+1}, \qquad \alpha_{0,n} = -\langle \vec{\alpha}_n, \vec{Y}^* \rangle + \delta, \qquad \{\alpha_{j,n} = \gamma^j\}_{j=1\ldots,\ell}.$$

Phase1.

$\mathcal{C}$ maintains a hash list, H-list, to store the mapping result of $H(GID, \vec{X})$. Then, $\mathcal{A}$ is allowed to query the following oracles:

- Hash oracle:

This oracle takes $\vec{X} \in \mathbb{Z}_p^{\ell}$ and $GID \in \{0,1\}^*$(global identity) as input and outputs an element of $\mathbb{G}$. If there exists a record $(GID, \vec{X}, v_k, V_k)$ in the H-list, return V_k. Otherwise, the oracle performs the following steps:

1. If $\langle \vec{X}, \vec{Y}^* \rangle = 0$, then randomly choose $V_k \xleftarrow{\$} \mathbb{G}$ and return V_k to $\mathcal{A}$;
2. Choose $v_k \xleftarrow{\$} \mathbb{Z}_p^*$;
3. Implicitly set

$$t = \frac{x_1\gamma^{\ell} + x_2\gamma^{\ell-1} + \ldots + x_{\ell}\gamma}{\langle \vec{X}, \vec{Y}^* \rangle} + v_k$$

by computing

$$V_k = g^t = (g^{\gamma})^{\frac{x_{\ell}}{\langle \vec{X}, \vec{Y}^* \rangle}} \cdot \ldots \cdot (g^{\gamma^{\ell-1}})^{\frac{x_2}{\langle \vec{X}, \vec{Y}^* \rangle}} \cdot (g^{\gamma^{\ell}})^{\frac{x_1}{\langle \vec{X}, \vec{Y}^* \rangle}} \cdot g^{v_k}.$$

This can be efficiently computed with the instance of q-DBDHE problem;
4. Return $H(GID, \vec{X}) = V_k$ to $\mathcal{A}$ and store $(GID, \vec{X}, v_k, V_k)$ into the H-list.

- KeyExtract oracle:

Upon receiving a vector $\vec{X} = (x_1, x_2, \ldots, x_{\ell})$ and a global identity GID from $\mathcal{A}$, where $\langle \vec{X}, \vec{Y}^* \rangle \neq 0$ (As shown in Definition 3, $\mathcal{A}$ is not allowed to make a KeyExtract query with $\langle \vec{X}, \vec{Y}^* \rangle = 0$, otherwise $\mathcal{A}$ can break the security trivially.) $\mathcal{C}$ performs as follows. For $i = 1, \ldots, n-1$, sk_i can be easily computed using the algorithm $\mathbf{KeyGen}_{\mathbf{A_i}}(pp, MSK_i, GID, \vec{X})$ shown in Section 3 since $\mathcal{C}$ knows MSK_i. As for sk_n, it can be computed from the instance of the ℓ-DBDHE problem by the following steps:

1. Query $V_k = H(GID, \vec{X})$ and set $D_0 = V_k$. Let $D_0 = g^t$, where

$$t = \frac{x_1\gamma^{\ell} + x_2\gamma^{\ell-1} + \ldots + x_{\ell}\gamma}{\langle \vec{X}, \vec{Y}^* \rangle} + v_k.$$

Note that v_k can be found in the H-list;
2. For $j = 2, \ldots, \ell$, compute

$$K_{j,n} = H(GID, \vec{X})^{-\alpha_{1,n}\frac{x_j}{x_1}} \cdot H(GID, \vec{X})^{\alpha_{j,n}} = (g^{-\alpha_{1,n}\frac{x_j}{x_1}} g^{\alpha_{j,n}})^t.$$

One can note that, in the exponent of $K_{j,n}$,

$$(-\alpha_{1,n}\frac{x_j}{x_1} + \alpha_{j,n})t = (-\gamma\frac{x_j}{x_1} + \gamma^j) \cdot (\frac{x_1\gamma^{\ell} + x_2\gamma^{\ell-1} + \ldots + x_{\ell}\gamma}{\langle \vec{X}, \vec{Y}^* \rangle} + v_k),$$

the only unknown term is $\gamma^{\ell+1}$. However, the coefficient of $\gamma^{\ell+1}$ is

$$-\frac{x_j}{x_1} \cdot \frac{x_1}{\langle \vec{X}, \vec{Y}^* \rangle} + \frac{x_j}{\langle \vec{X}, \vec{Y}^* \rangle} = 0, j = 2, \ldots, \ell.$$

Thus, $K_{j,n}$ can be easily computed using the knowledge of $\vec{X}, \vec{Y}^*$ and the instance $(g, g^{\gamma}, g^{\gamma^2}, \ldots, g^{\gamma^{\ell}}, g^{\gamma^{\ell+2}}, \ldots, g^{\gamma^{2\ell}})$ of the ℓ-DBDHE problem;
3. Compute

$$D_{1,n} = g^{\overline{\alpha}_n} \cdot H(GID, \vec{X})^{\alpha_{0,n}} = g^{\overline{\alpha}_n + \alpha_{0,n}t}.$$

One can note that the exponent of $D_{1,n}$ is

$$
\begin{aligned}
&\overline{\alpha}_n + \alpha_{0,n} t \\
=&\gamma^{\ell+1} + (-\langle \vec{\alpha}_n, \vec{Y}^* \rangle + \delta) \cdot \ t \\
=&\gamma^{\ell+1} + (-\langle \vec{\alpha}_n, \vec{Y}^* \rangle) \cdot (\frac{x_1\gamma^{\ell} + x_2\gamma^{\ell-1} + \ldots + x_\ell\gamma}{\langle \vec{X}, \vec{Y}^* \rangle} + v_k) + \delta\ t \\
=&\gamma^{\ell+1} - \frac{(\alpha_1 y_1^* + \alpha_2 y_2^* + \ldots + \alpha_\ell y_\ell^*)(x_1\gamma^{\ell} + x_2\gamma^{\ell-1} + \ldots + x_\ell\gamma)}{\langle \vec{X}, \vec{Y}^* \rangle} - \langle \vec{\alpha}_n, \vec{Y}^* \rangle \cdot v_k + \delta\ t \\
=&\gamma^{\ell+1} - \frac{(\gamma y_1^* + \gamma^2 y_2^* + \ldots + \gamma^\ell y_\ell^*)(x_1\gamma^{\ell} + x_2\gamma^{\ell-1} + \ldots + x_\ell\gamma)}{\langle \vec{X}, \vec{Y}^* \rangle} - \langle \vec{\alpha}_n, \vec{Y}^* \rangle \cdot v_k + \delta t
\end{aligned}
$$

Again, the coefficient of the unknown term $\gamma^{\ell+1}$ is

$$1 - \frac{x_1 y_1^* + x_2 y_2^* + \ldots + x_\ell y_\ell^*}{\langle \vec{X}, \vec{Y}^* \rangle} = 1 - \frac{\langle \vec{X}, \vec{Y}^* \rangle}{\langle \vec{X}, \vec{Y}^* \rangle} = 0.$$

Therefore, $D_{1,n}$ can be also computed using the knowledge of $\vec{X}, \vec{Y}^*$ and the instance $(g, g^{\gamma}, g^{\gamma^2}, \ldots, g^{\gamma^{\ell}}, g^{\gamma^{\ell+2}}, \ldots, g^{\gamma^{2\ell}})$ of the ℓ-DBDHE problem.

Challenge.

$\mathcal{A}$ submits two message M_0 and M_1 of the same length, and $\mathcal{C}$ computes the challenge ciphertext as follows:

1. Choose $\beta \xleftarrow{\$} \{0, 1\}$;
2. Set $E_2 = g^s$;
3. Compute

$$
\begin{aligned}
E_0 &= M_\beta \cdot (\textstyle\prod_{i=1}^{n-1} Z_i)^s \cdot \ T \\
&= M_\beta \cdot Z_1^s \cdot \ldots \cdot Z_{n-1}^s \cdot \ T \\
&= M_\beta \cdot e(g^s, g^{\overline{\alpha}_1}) \cdot \ldots \cdot e(g^s, g^{\overline{\alpha}_{n-1}}) \cdot T;
\end{aligned}
$$

4. Compute

$$
\begin{aligned}
E_1 &= \left((\textstyle\prod_{i=1}^{n} g^{\alpha_{0,i}}) \cdot (\prod_{i=1}^{n} g^{\alpha_{1,i}})^{y_1^*} \cdot \ldots \cdot (\prod_{i=1}^{n} g^{\alpha_{\ell,i}})^{y_\ell^*} \right)^s \\
&= \left((g^{\alpha_{0,1}} \cdot \ldots \cdot g^{\alpha_{0,n-1}}) \cdot (g^{\alpha_{1,1}} \cdot \ldots \cdot g^{\alpha_{1,n-1}})^{y_1^*} \cdot \ldots \cdot (g^{\alpha_{\ell,1}} \cdot \ldots \cdot g^{\alpha_{\ell,n-1}})^{y_\ell^*} \right)^s \\
&\cdot \left(g^{\alpha_{0,n}} \cdot (g^{\alpha_{1,n}})^{y_1^*} \cdot \ldots \cdot (g^{\alpha_{\ell,n}})^{y_\ell^*} \right)^s \\
&= (g^s)^{\sum_{i=1}^{n-1} \alpha_{0,i} + y_1^* \sum_{i=1}^{n-1} \alpha_{1,i} + \ldots + y_\ell^* \sum_{i=1}^{n-1} \alpha_{\ell,i}} \cdot \left(g^{-\langle \vec{\alpha}_n, \vec{Y}^* \rangle + \delta} \cdot g^{\langle \vec{\alpha}_n, \vec{Y}^* \rangle} \right)^s \\
&= (g^s)^{\sum_{i=1}^{n-1} \alpha_{0,i} + y_1^* \sum_{i=1}^{n-1} \alpha_{1,i} + \ldots + y_\ell^* \sum_{i=1}^{n-1} \alpha_{\ell,i}} \cdot (g^s)^{\delta};
\end{aligned}
$$

5. Output $C^* = (E_0, E_1, E_2)$ to $\mathcal{A}$.

Phase2.

Same as Phase1.

Guess.

$\mathcal{A}$ outputs a bit $\beta' \in \{0, 1\}$. $\mathcal{C}$ outputs 1 if $\beta' = \beta$; otherwise, $\mathcal{C}$ outputs 0.

If $T = e(g, g)^{\gamma^{\ell+1} s}$, then:

$$
\begin{aligned}
E_0 &= M_\beta \textstyle\prod_{i=1}^{n-1} Z_i^s \cdot \ T \\
&= M_\beta \textstyle\prod_{i=1}^{n-1} Z_i^s \cdot e(g, g)^{\gamma^{\ell+1} s} \\
&= M_\beta \textstyle\prod_{i=1}^{n-1} Z_i^s \cdot Z_n^s \\
&= M_\beta \textstyle\prod_{i=1}^{n} Z_i^s,
\end{aligned}
$$

and hence $C^* = (E_0, E_1, E_2)$ is a valid ciphertext. Thus, we have:

$$\mathbf{Adv}_{\mathcal{A}}^{\mathsf{sIND\text{-}CPA}} = \left|\Pr[\beta' = \beta] - \frac{1}{2}\right|,$$

and

$$\Pr[\mathcal{C}(g, g^{\gamma}, g^{\gamma^2}, \ldots, g^{\gamma^{\ell}}, g^{\gamma^{\ell+2}}, \ldots, g^{\gamma^{2\ell}}, g^s, T = e(g,g)^{\gamma^{\ell+1}s}) = 1] = \Pr[\beta' = \beta]$$
$$= \mathbf{Adv}_{\mathcal{A}}^{\mathsf{sIND\text{-}CPA}} + \frac{1}{2}.$$

If T is a random element from $\mathbb{G}_T$, then the message M_β is completely hidden from the adversary's view, since E_0, E_1 and E_2 are all independently random elements. Therefore, the advantage of the adversary is:

$$\mathbf{Adv}_{\mathcal{A}}^{\mathsf{sIND\text{-}CPA}} = \left|Pr[\beta' = \beta] - \frac{1}{2}\right| = 0,$$

and

$$\Pr[\mathcal{C}(g, g^{\gamma}, g^{\gamma^2}, \ldots, g^{\gamma^{\ell}}, g^{\gamma^{\ell+2}}, \ldots, g^{\gamma^{2\ell}}, g^s, T = e(g,g)^{\gamma^{\ell+1}s}) = 1] = \frac{1}{2}.$$

Finally, the advantage of $\mathcal{C}$ in solving the ℓ-DBDHE problem is:

$$\begin{aligned}
&\mathbf{Adv}_{\mathcal{C}}^{\ell\text{-DBDHE}} \\
=&\Big|Pr[\mathcal{C}(g, g^{\gamma}, g^{\gamma^2}, \ldots, g^{\gamma^{\ell}}, g^{\gamma^{\ell+2}}, \ldots, g^{\gamma^{2\ell}}, g^s, T = e(g,g)^{\gamma^{\ell+1}s}) = 1] \\
&-Pr[\mathcal{C}(g, g^{\gamma}, g^{\gamma^2}, \ldots, g^{\gamma^{\ell}}, g^{\gamma^{\ell+2}}, \ldots, g^{\gamma^{2\ell}}, g^s, T \xleftarrow{\$} \mathbb{G}_T) = 1]\Big| \\
=&\left|(\mathbf{Adv}_{\mathcal{A}}^{\mathsf{sIND\text{-}CPA}} + \frac{1}{2}) - \frac{1}{2}\right| \\
=&\mathbf{Adv}_{\mathcal{A}}^{\mathsf{sIND\text{-}CPA}}.
\end{aligned}$$

Therefore, if there is an adversary that wins the sIND-CPA game with a non-negligible advantage, then we can construct an algorithm $\mathcal{C}$ to solve the ℓ-DBDHE problem with a non-negligible advantage in polynomial time. □

5. Comparison

In this section, we compare our scheme with [3,5,8,11,25] in time complexity, space complexity and other security features. Among these works, [3,5,8,11] are IPE schemes and [25] is a DIPE scheme. In addition, we implement our scheme and the scheme of [25] in Python and C, and compare the execution time of our algorithms with theirs.

5.1. Asymptotic Comparisons

In Table 2, we show the encryption cost and decryption cost of each scheme. For encryption, the exponentiation computation cost is linear with the vector size, which is better than others, except [5]. In addition, we only need ℓ times exponentiation computations plus two pairing computations in decryption. Though our efficiency is not the best among [3,5,8,11], our scheme achieves decentralization while others do not. In [25], they need n times exponentiation computations plus $\mathcal{O}(k)$ pairing computations, where $k \leq 2$. Thus, both of the cost for our scheme in encryption and decryption algorithm is more efficient.

Table 2. Comparison of time complexity.

	Encryption Cost	Decryption Cost
[3]	$(4\ell+1)T_e$	$(2\ell+1)T_p$
[8]	$(2\ell+2)T_e$	$\ell T_e + 3T_p$
[5]	$(\ell+3)T_e$	$\ell T_e + 2T_p$
[11]	$(2\ell+2)T_e$	$(\ell+2)T_e + T_p$
[25]	$[(2n+1)k^2 + (2n+2)k]T_e$	$nT_e + (2k+2)T_p$
Ours	$(\ell+3)T_e$	$\ell T_e + 2T_p$

T_e: The cost of an exponentiation in multiplicative groups. T_p: The cost of pairing computation. n: The total number of authorities. ℓ: The length of predicate/attribute vector. k: The parameter of k-linear assumption. ($k \geq 2$)

The length of ciphertexts and private keys are shown in Table 3. Due to decentralization, it is normal that the private key length of DIPE is larger than that of IPE. In addition, though, we can see that [25] needs about $\mathcal{O}(nk)$ elements in $\mathbb{G}$ for a private key. Indeed, the value of k can be small in their work. However, in our work, the vector size could be large in reality. Therefore, our private key length is larger than others, which may need more storage. Nevertheless, if the value of k is greater or equal than our vector size. Then, we only need less storage than [25] in storing the private key. Note that the work of [11] achieves constant private key size. As a trade-off, their ciphertext size is $\mathcal{O}(\ell)|\mathbb{G}_T|$, which might be longer then others in the respect of the curve used in implementation.

In the comparison of ciphertext length, both our work and [5] have the least ciphertext length and only needs two elements in $\mathbb{G}$ plus an element in $\mathbb{G}_T$. It means that our ciphertext length is independent with the vector size and the number of the authorities. It can reduce the burden of connection between sender and receiver for transmitting ciphertext. However, the ciphertext length of [25] dependent on n and k. To the best of our knowledge, our work is the first DIPE scheme achieving a constant-size ciphertext.

Table 3. Comparison with the previous schemes in space complexity.

	Ciphertext Length	Private Key Length
[3]	$(2\ell+1)\|\mathbb{G}\|$	$(2\ell+1)\|\mathbb{G}\|$
[8]	$(\ell+2)\|\mathbb{G}\|$	$3\|\mathbb{G}\| + \|\mathbb{Z}_p^\ell\|$
[5]	$2\|\mathbb{G}\| + \|\mathbb{G}_T\|$	$(\ell+1)\|\mathbb{G}\|$
[11]	$1\|\mathbb{G}\| + (\ell+1)\|\mathbb{G}_T\|$	$1\|\mathbb{G}\| + \|\mathbb{Z}_p\|$
[25]	$(nk+n+k+1)\|\mathbb{G}\| + \|\mathbb{G}_T\|$	$n(2k+2)\|\mathbb{G}\|$
Ours	$2\|\mathbb{G}\| + \|\mathbb{G}_T\|$	$n(\ell+1)\|\mathbb{G}\|$

$|\mathbb{G}|$: The length of an element in $\mathbb{G}$. $|\mathbb{G}_T|$: The length of an element in $\mathbb{G}_T$. $|\mathbb{Z}_p^\ell|$: The length of an element in $\mathbb{Z}_p^\ell$. n: The total number of authorities. ℓ: The length of predicate/attribute vector. k: The parameter of k-linear assumption. ($k \geq 2$).

In Table 4, only our work, as well as [25], achieves a decentralized framework. In order to avoid collusion between users, a GID and a predicate (or an attribute) vector $\vec{X}$ are mapped to a value by a random oracle. Therefore, the security of ours and [25]'s are both proven in the random oracle model. As far as we know, there is no standard model for DIPE currently. In addition, although ours and [25]'s are both CPA secure, the latter achieves adaptive security, which is stronger than our selective model. Though all the works in Table 4 achieve CPA security, we should note that [11]'s security is proven in a relatively less used model, called a co-selective model, where an adversary outputs several vectors for querying the Key-Extract oracle in Phase 1 before seeing the system parameter. Although selective security and co-selective security are both weaker than full security, both notions are incomparable in general by definition.

Table 4. Property comparison.

	Decentralization	Confidentiality	Security Model	Group Order	Complexity Assumption
[3]	No	CPA	STD	Composite	SD
[8]	No	CPA	STD	Prime	$\mathcal{P}$-DBDH
[5]	No	CPA	STD	Prime	ℓ-DBDHE
[11]	No	CPA*	STD	Prime	M-DDH$_{\mathbb{G}_T}$
[25]	Yes	CPA	ROM	Prime	k-Lin
Ours	Yes	CPA	ROM	Prime	ℓ-DBDHE

CPA: Chosen-plaintext attack. CPA*: CPA in coselective model. STD: Standard model. ROM: Random oracle model. SD: Subgroup decision problem.

5.2. Experimental Result

In this section, we show the experimental results of our construction and the construction of [25] via Python and C languages, and analyze the execution time of the five algorithms.

Table 5 shows the system configuration and the chosen pairing group of Python. We implement our construction by Charm-Crypto library in Python. In our implementation, the pairing group is a symmetric pairing curve with a 512-bit-based field. The experiment is executed on Intel(R) Core(TM) i7-10875H CPU at 3.60GHz processor, 4 GB memory size and under the Ubuntu-16.04 operating system. In addition, we also implement our scheme and [25] in C with the pbc library, where a Type a1 pairing group is used. Table 6 shows the details for the system configuration of our C implementation.

Table 5. System configuration and elliptic curve for Python.

CPU	Intel(R) Core(TM) i7-10875H CPU @ 2.30GHz
Memory	4GB
OS	Ubuntu-16.04 (64-bit)
Package	Python Charm-Crypto (v0.43) library
Pairing group	SS512

Table 6. System configuration and elliptic curve for C.

CPU	Intel(R) Core(TM) i5-8257U CPU @ 1.40GHz
Memory	2GB
OS	Docker:Debian10
Package	pbc-0.5.14
Pairing group	Type a1

We analyze the time cost of each algorithm in our DIPE scheme below. In our experiment, the length of GID (global identity) is set to 10 bits for convenience. However, note that the length of a GID can be arbitrarily long since it is a input of the hash function. In [25], since each authority generates a partial key for an element of the predicate vector, therefore, the length of th vector size should be the same as the total number of authorities, ranging from 1 to 25 in our implementation. In addition, the value of k in [25] is set to one to minimize the cost of their work. The value of each point on the figure is obtained by executing the algorithm 1000 times and obtaining the value of the average execution time.

For the implementation using Python, Figure 1b shows that the time spent by [25] on the **AuthSetup** algorithm is more time-consuming than ours. In Figure 1d,e, we can note that the **Encrypt** and **Decrypt** algorithms are both growing linearly in two schemes when

the number of authorities increases. However, ours has better performance than theirs. Then, Figure 1c exhibits that **KeyGen** is the most time-consuming algorithm due to the decentralized network. Nevertheless, we have relatively poorer performance than [25]. Since our decentralization is different from [25], in our scheme, each authority generates a partial key for a whole predicate vector instead of only an element. Therefore, the execution time of **KeyGen** is longer. Finally, in Figure 1a, the **Setup** algorithm only generates some generator of $\mathbb{G}$, some elements of $\mathbb{G}$ and the description of a hash function in both schemes. Thus, execution time is independent of the total number of authorities and vector size. In addition, our scheme has one more advantage, that is, the length of the predicate vector does not need to bind with the total number of authorities with same value.

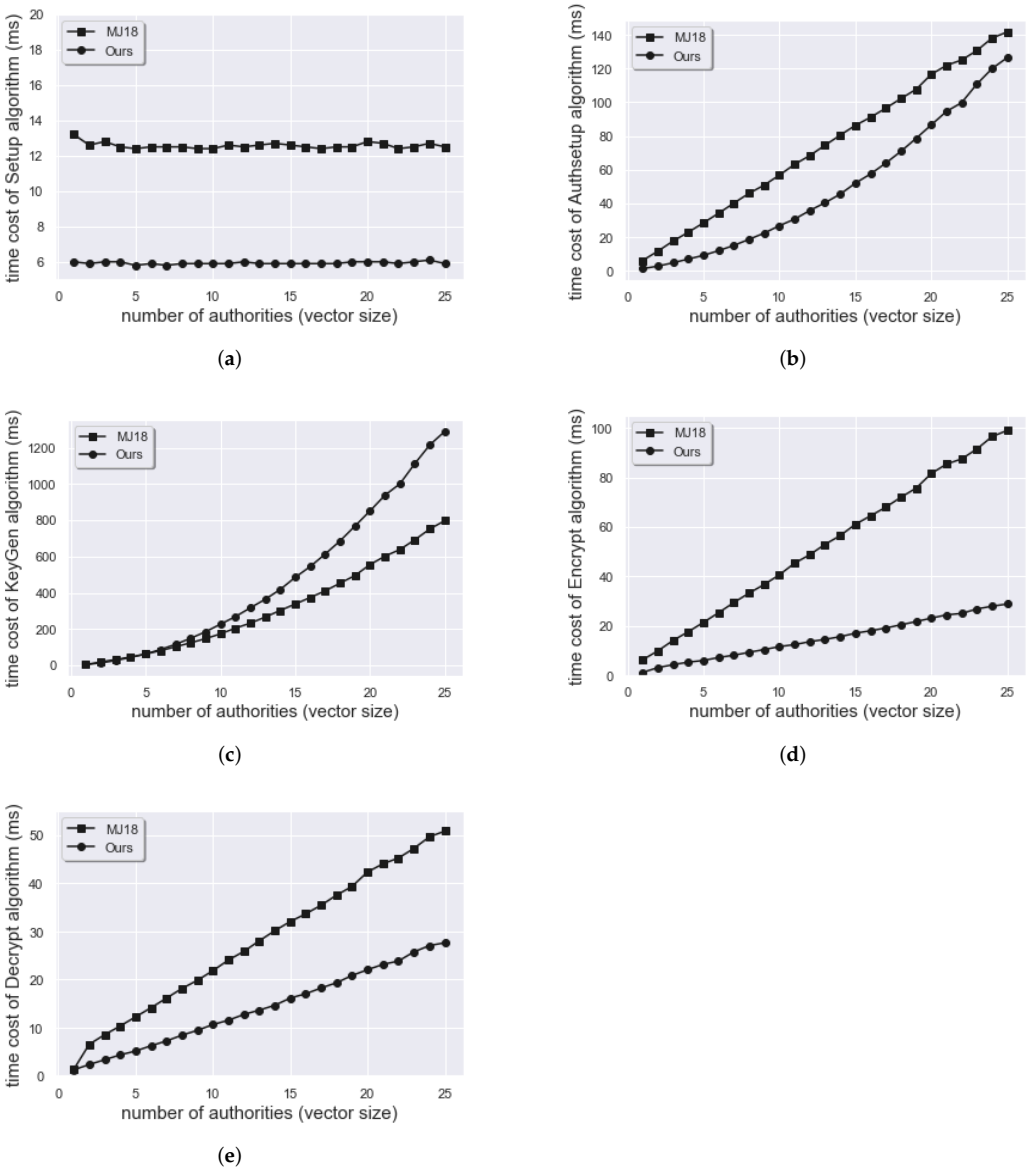

Figure 1. The time cost for Python Implementation of (**a**) Setup, (**b**) Authsetup, (**c**) KeyGen, (**d**) Encrypt, (**e**) Decrypt algorithm. (Pairing group: SS512, $|GID|$=10, # of authorities = $|\vec{X}| = |\vec{Y}| = [1,\ldots,25]$, $k = 1$).

In addition, Figure 2 shows the time cost of our scheme and [25] using C. Similar to the results using Python, Figure 2a,b show that in the comparison of the time costs of **Setup** and **AuthSetup** algorithms, our scheme is more efficient than [25]. As shown in Figure 2c,e, the costs for **KeyGen** and **Decryption** of ours are pretty close to those of [25].

Interestingly, the result of **Encryption** in C is opposite to that in Python. Figure 2d shows that the **Encrypt** algorithm of [25] is faster than ours. The reason for this might be due to the system configuration or the language. We will keep figuring out more details that may be inspired from this difference.

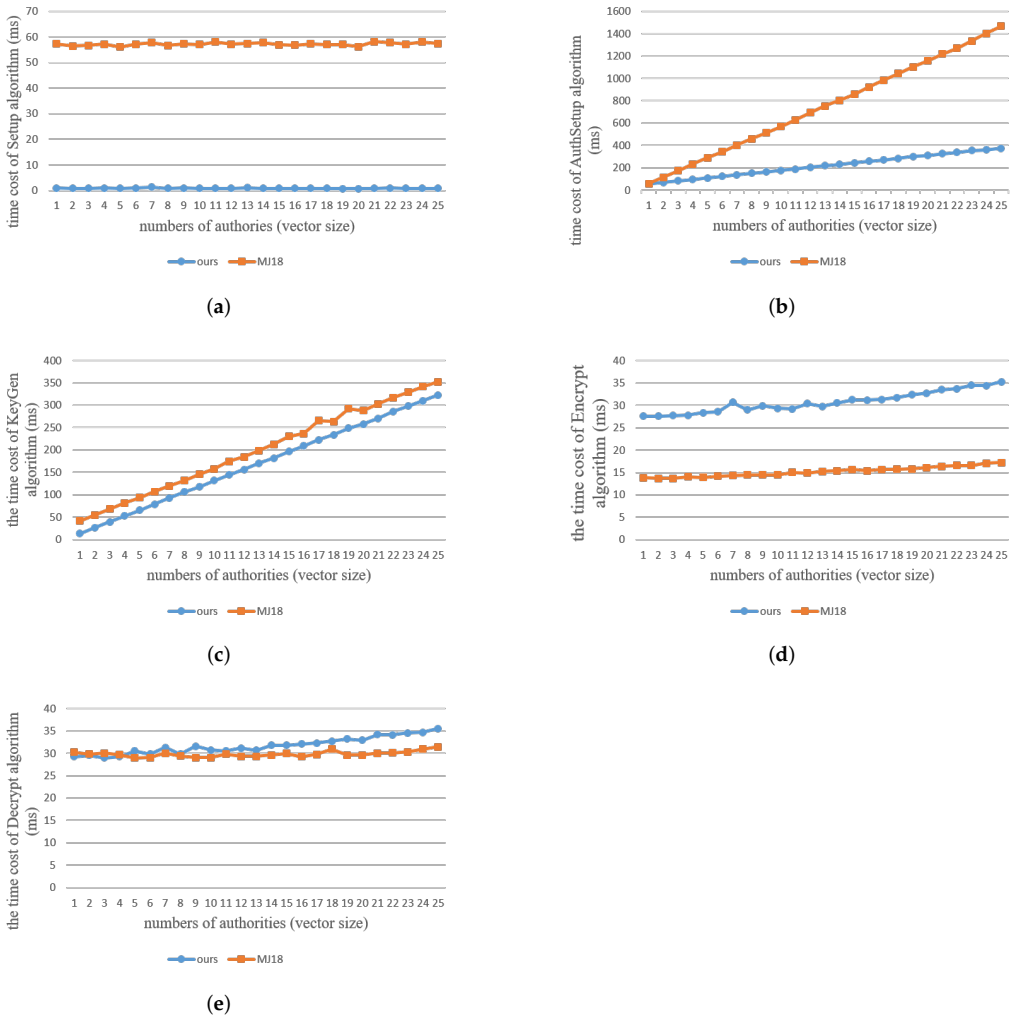

Figure 2. The time cost for C Implementation of (**a**) Setup, (**b**) Authsetup, (**c**) KeyGen, (**d**) Encrypt, (**e**) Decrypt algorithm. (Pairing group: Type a1, $|GID| = 10$, # of authorities = $|\vec{X}| = |\vec{Y}| = [1,\ldots,25]$, $k = 1$).

6. Conclusions

Thus far, there is only one decentralized inner product encryption, proposed by Michalevsky et al. in 2018. In their scheme, however, the length of ciphertexts are dependent on the number of authorities, which may become a bottleneck in the system. Therefore, we would like to solve this problem. In this manuscript, we present a novel decentralized inner product encryption which achieves constant-size ciphertexts. In addition, our scheme is proven to be selectively secure under the ℓ-DBDHE assumption. We further implement our scheme and the scheme of [25] to analyze the execution time. Except for the KeyGen algorithm, our work has better performance in the remaining four algorithms (Setup, AuthSetup, Encrypt, Decrypt). Yet, our scheme is the first DIPE scheme achieving constant-size ciphertext, and there are several potential improvements. One direction could be to upgrade the security to chosen-ciphertext security. Several generic methods [31–35] have been proposed in the literature, however, constructing a DIPE scheme with direct chosen-ciphertext security is an open problem. In addition, the security of our scheme is proven under the random oracle model. How to construct a DIPE scheme that is secure in the standard model is also a worth-fighting goal.

Author Contributions: Conceptualization, Y.-F.T. and S.-J.G.; methodology, Y.-F.T.; formal analysis, Y.-F.T.; investigation, Y.-F.T. and S.-J.G.; writing—original draft preparation, S.-J.G.; writing—review and editing, Y.-F.T.; supervision, Y.-F.T.; project administration, Y.-F.T.; funding acquisition, Y.-F.T. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the Ministry of Science and Technology, Taiwan (ROC), under grant numbers MOST 110-2221-E-004 -003-, MOST 110-2218-E-004-001-MBK, MOST 109-2221-E-004 -011 -MY3 and MOST 109-3111-8-004-001.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Shamir, A. Identity-Based Cryptosystems and Signature Schemes. In *Advances in Cryptology*; Blakley, G.R., Chaum, D., Eds.; Springer: Berlin/Heidelberg, Germany, 1985; pp. 47–53.
2. Boneh, D.; Franklin, M. Identity-Based Encryption from the Weil Pairing. In *Advances in Cryptology—CRYPTO 2001*; Kilian, J., Ed.; Springer: Berlin/Heidelberg, Germany, 2001; pp. 213–229.
3. Katz, J.; Sahai, A.; Waters, B. Predicate Encryption Supporting Disjunctions, Polynomial Equations, and Inner Products. In *Advances in Cryptology–EUROCRYPT 2008*; Smart, N., Ed.; Springer: Berlin/Heidelberg, Germany, 2008; pp. 146–162.
4. Lewko, A.; Okamoto, T.; Sahai, A.; Takashima, K.; Waters, B. Fully Secure Functional Encryption: Attribute-Based Encryption and (Hierarchical) Inner Product Encryption. In *Advances in Cryptology–EUROCRYPT 2010*; Gilbert, H., Ed.; Springer: Berlin/Heidelberg, Germany, 2010; pp. 62–91.
5. Attrapadung, N.; Libert, B. Functional Encryption for Inner Product: Achieving Constant-Size Ciphertexts with Adaptive Security or Support for Negation. In *Public Key Cryptography–PKC 2010*; Nguyen, P.Q., Pointcheval, D., Eds.; Springer: Berlin/Heidelberg, Germany, 2010; pp. 384–402.
6. Park, J.H. Inner-product encryption under standard assumptions. *Des. Codes Cryptogr.* **2011**, *58*, 235–257.
7. Tan, Z.; Zhang, W. A Predicate Encryption Scheme Supporting Multiparty Cloud Computation. In Proceedings of the 2015 International Conference on Intelligent Networking and Collaborative Systems, Taipei, Taiwan, 2–4 September 2015; pp. 252–256.
8. Kim, I.; Hwang, S.O.; Park, J.H.; Park, C. An Efficient Predicate Encryption with Constant Pairing Computations and Minimum Costs. *IEEE Trans. Comput.* **2016**, *65*, 2947–2958.
9. Zhang, Y.; Li, Y.; Wang, Y. Efficient inner product encryption for mobile clients with constrained computation capacity. *Int. J. Innov. Comput. Inf. Control* **2019**, *15*, 209–226.
10. Soroush, N.; Iovino, V.; Rial, A.; Roenne, P.B.; Ryan, P.Y.A. Verifiable Inner Product Encryption Scheme. In *Public-Key Cryptography–PKC 2020*; Kiayias, A., Kohlweiss, M., Wallden, P., Zikas, V., Eds.; Springer International Publishing: Cham, Switzerland, 2020; pp. 65–94.
11. Tseng, Y.F.; Liu, Z.Y.; Tso, R. Practical Inner Product Encryption with Constant Private Key. *Appl. Sci.* **2020**, *10*, 8669.
12. Fiat, A.; Naor, M. Broadcast Encryption. In *Advances in Cryptology—CRYPTO' 93*; Stinson, D.R., Ed.; Springer: Berlin/Heidelberg, Germany, 1994; pp. 480–491.
13. Acharya, K. Secure and efficient public key multi-channel broadcast encryption schemes. *J. Inf. Secur. Appl.* **2020**, *51*, 102436, https://doi.org/10.1016/j.jisa.2019.102436.
14. Chen, L.; Li, J.; Zhang, Y. Anonymous Certificate-Based Broadcast Encryption With Personalized Messages. *IEEE Trans. Broadcast.* **2020**, *66*, 867–881, https://doi.org/10.1109/TBC.2020.2984974.
15. Li, J.; Wang, S.; Li, Y.; Wang, H.; Wang, H.; Wang, H.; Chen, J.; You, Z. An Efficient Attribute-Based Encryption Scheme With Policy Update and File Update in Cloud Computing. *IEEE Trans. Ind. Inform.* **2019**, *15*, 6500–6509, https://doi.org/10.1109/TII.2019.2931156.
16. Xue, L.; Yu, Y.; Li, Y.; Au, M.H.; Du, X.; Yang, B. Efficient attribute-based encryption with attribute revocation for assured data deletion. *Inf. Sci.* **2019**, *479*, 640–650, https://doi.org/10.1016/j.ins.2018.02.015.
17. Li, J.; Zhang, Y.; Ning, J.; Huang, X.; Poh, G.S.; Wang, D. Attribute Based Encryption with Privacy Protection and Accountability for CloudIoT. *IEEE Trans. Cloud Comput.* **2020**, 1, https://doi.org/10.1109/TCC.2020.2975184.
18. Katz, J.; Maffei, M.; Malavolta, G.; Schröder, D. Subset Predicate Encryption and Its Applications. In *Cryptology and Network Security*; Capkun, S., Chow, S.S.M., Eds.; Springer International Publishing: Cham, Switzerland, 2018; pp. 115–134.
19. Chatterjee, S.; Mukherjee, S. Large Universe Subset Predicate Encryption Based on Static Assumption (Without Random Oracle). In *Topics in Cryptology–CT-RSA 2019*; Matsui, M., Ed.; Springer International Publishing: Cham, Switzerland, 2019; pp. 62–82.
20. Tseng, Y.F.; Gao, S.J. Efficient Subset Predicate Encryption for Internet of Things. In Proceedings of the 2021 IEEE Conference on Dependable and Secure Computing (DSC), Edinburgh, UK, 22–24 June 2021; pp. 1–2, https://doi.org/10.1109/DSC49826.2021.9346245.
21. Rajan, M.; Varghese, A.; Narendra, N.; Singh, M.; Shivraj, V.; Chandra, G.; Balamuralidhar, P. Security and Privacy for Real Time Video Streaming Using Hierarchical Inner Product Encryption Based Publish-Subscribe Architecture. In Proceedings of the 2016

30th International Conference on Advanced Information Networking and Applications Workshops (WAINA), Crans-Montana, Switzerland, 23–25 March 2016; pp. 373–380, https://doi.org/10.1109/WAINA.2016.101.
22. Xiong, H.; Yang, M.; Yao, T.; Chen, J.; Kumari, S. Efficient Unbounded Fully Attribute Hiding Inner Product Encryption in Cloud-Aided WBANs. *IEEE Syst. J.* **2021**, 1–9, https://doi.org/10.1109/JSYST.2021.3125455.
23. Zhang, L.; Wang, Z.; Mu, Y.; Hu, Y. Fully Secure Hierarchical Inner Product Encryption for Privacy Preserving Keyword Searching in Cloud. In Proceedings of the 2015 10th International Conference on P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC), Krakow, Poland, 4–6 November 2015; pp. 449–453, https://doi.org/10.1109/3PGCIC.2015.63.
24. Huang, K.C.; Chen, Y.C. Privacy Preserving Outsourced Data Integration from Inner Product Encryption. In Proceedings of the 2021 International Symposium on Intelligent Signal Processing and Communication Systems (ISPACS), Hualien City, Taiwan, 16–19 November 2021; pp. 1–2, https://doi.org/10.1109/ISPACS51563.2021.9651006.
25. Michalevsky, Y.; Joye, M. Decentralized Policy-Hiding ABE with Receiver Privacy. In Proceedings of the 23rd European Symposium on Research in Computer Security, ESORICS 2018, Barcelona, Spain, 3–7 September 2018; pp. 548–567.
26. Chase, M. Multi-authority Attribute Based Encryption. In *Theory of Cryptography*; Vadhan, S.P., Ed.; Springer: Berlin/Heidelberg, Germany, 2007; pp. 515–534.
27. Lewko, A.; Waters, B. Decentralizing Attribute-Based Encryption. In *Advances in Cryptology–EUROCRYPT 2011*; Paterson, K.G., Ed.; Springer: Berlin/Heidelberg, Germany, 2011; pp. 568–588.
28. Zhang, L.; Gao, X.; Kang, L.; Liang, P.; Mu, Y. Distributed Ciphertext-Policy Attribute-Based Encryption With Enhanced Collusion Resilience and Privacy Preservation. *IEEE Syst. J.* **2021**, 1–12. doi:10.1109/JSYST.2021.3072793.
29. Boneh, D.; Gentry, C.; Waters, B. Collusion Resistant Broadcast Encryption with Short Ciphertexts and Private Keys. In *Advances in Cryptology–CRYPTO 2005*; Shoup, V., Ed.; Springer: Berlin/Heidelberg, Germany, 2005; pp. 258–275.
30. Boneh, D.; Hamburg, M. *Generalized Identity Based and Broadcast Encryption Schemes*; In Proceedings of the International Conference on the Theory and Application of Cryptology and Information Security, Melbourne, Australia, 7–11 December 2008.
31. Canetti, R.; Halevi, S.; Katz, J. Chosen-Ciphertext Security from Identity-Based Encryption. In *Advances in Cryptology–EUROCRYPT 2004*; Cachin, C., Camenisch, J.L., Eds.; Springer: Berlin/Heidelberg, Germany, 2004; pp. 207–222.
32. Fujisaki, E.; Okamoto, T. Secure Integration of Asymmetric and Symmetric Encryption Schemes. In *Advances in Cryptology—CRYPTO' 99*; Wiener, M., Ed.; Springer: Berlin/Heidelberg, Germany, 1999; pp. 537–554.
33. Fujisaki, E.; Okamoto, T. Secure Integration of Asymmetric and Symmetric Encryption Schemes. *J. Cryptol.* **2011**, *26*, 80–101.
34. Koppula, V.; Waters, B. Realizing Chosen Ciphertext Security Generically in Attribute-Based Encryption and Predicate Encryption. In *Advances in Cryptology–CRYPTO 2019*; Boldyreva, A., Micciancio, D., Eds.; Springer International Publishing: Cham, Switzerland, 2019; pp. 671–700.
35. Yamada, S.; Attrapadung, N.; Hanaoka, G.; Kunihiro, N. Generic Constructions for Chosen-Ciphertext Secure Attribute Based Encryption. In *Public Key Cryptography—PKC 2011*; Catalano, D., Fazio, N., Gennaro, R., Nicolosi, A., Eds.; Springer: Berlin/Heidelberg, Germany, 2011; pp. 71–89.