# Multi-Party Functional Encryption

Shweta Agrawal[1⋆], Rishab Goyal[2⋆⋆], and Junichi Tomida[3]

[1] IIT Madras, India
shweta.a@cse.iitm.ac.in
[2] MIT, Cambridge, MA
goyal@utexas.edu
[3] NTT Corporation, Japan
junichi.tomida.vw@hco.ntt.co.jp

**Abstract.** We initiate the study of *multi-party functional encryption* (MPFE) which unifies and abstracts out various notions of functional encryption which support distributed ciphertexts or secret keys, such as multi-input FE, multi-client FE, decentralized multi-client FE, multi-authority FE, dynamic decentralized FE, adhoc multi-input FE and such others. Using our framework, we identify several gaps in the literature and provide some constructions to fill these:

1. **Multi-Authority ABE with Inner Product Computation.** The recent work of Abdalla et al. (ASIACRYPT'20) constructed a novel "composition" of Attribute Based Encryption (ABE) and Inner Product Functional Encryption (IPFE), namely functional encryption schemes that combine the access control functionality of attribute based encryption with the possibility of performing linear operations on the encrypted data. In this work, we extend the access control component to support the much more challenging multi-authority setting, i.e. "lift" the primitive of ABE in their construction to multi-authority ABE for the same class of access control policies (LSSS structures). This yields the first construction of a nontrivial multi-authority FE beyond ABE from simple assumptions on pairings to the best of our knowledge.

   Our techniques can also be used to generalize the decentralized attribute based encryption scheme of Michalevsky and Joye (ESORICS'18) to support inner product computation on the message. While this scheme only supports inner product predicates which is less general than those supported by the Lewko-Waters (EUROCRYPT'11) construction, it supports policy hiding which the latter does not. Our extension inherits these features and is secure based on the $k$-linear assumption, in the random oracle model.

⋆ Research supported by the DST "Swarnajayanti" fellowship, an Indo-French CEFIPRA project and the CCD Centre of Excellence

⋆⋆ Research supported in part by NSF CNS Award #1718161, an IBM-MIT grant, and by the Defense Advanced Research Projects Agency (DARPA) under Contract No. HR00112020023. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the United States Government or DARPA. Work done in part while at the Simons Institute for the Theory of Computing, supported by Simons-Berkeley research fellowship.

2. **Function Hiding DDFE.** The novel primitive of *dynamic* decentralized functional encryption (DDFE) was recently introduced by Chotard et al. (CRYPTO'20), where they also provided the first construction for inner products. However, the primitive of DDFE does not support function hiding, which is a significant limitation for several applications. In this work, we provide a new construction for inner product DDFE which supports function hiding. To achieve our final result, we define and construct the first function hiding *multi-client* functional encryption (MCFE) scheme for inner products, which may be of independent interest.
3. **Distributed Ciphertext-Policy ABE.** We provide a distributed variant of the recent ciphertext-policy attribute based encryption scheme, constructed by Agrawal and Yamada (EUROCRYPT'20). Our construction supports $\mathbf{NC}^1$ access policies, and is secure based on "Learning With Errors" and relies on the generic bilinear group model as well as the random oracle model.

Our new MPFE abstraction predicts meaningful new variants of functional encryption as useful targets for future work.

## 1 Introduction

Functional encryption (FE) [32,14] is a powerful generalization of public key encryption which enables a user to learn a function of the encrypted data. Concretely, in FE, a secret key $\mathsf{SK}_f$ is associated with a function $f$ and the ciphertext $\mathsf{CT}_{\mathbf{x}}$ is associated with a message $\mathbf{x}$ (in the domain of $f$). And, by combining $\mathsf{SK}_f$ with $\mathsf{CT}_{\mathbf{x}}$, the decryptor learns $f(\mathbf{x})$ and nothing else.

The original motivation behind the concept of functional encryption, as discussed in [14], was to put forth a *new broad vision of encryption systems*. Since its introduction, the concept of FE has been massively impactful in several aspects: (i) it helped unify the existing literature on encryption systems (such as identity-based encryption [33,12], attribute-based encryption [32,26], predicate encryption [15,27] and more) and place them under a single umbrella which enabled clear comparisons, (ii) it helped in predicting new natural encryption primitives that had not been studied before, such as partially hiding predicate/functional encryption [25], and (iii) it served as the right abstraction to understand the relationship of this broad concept with other notions in cryptography, such as to indistinguishability obfuscation [9,11].

**Supporting Multiple Users.** Subsequently, many new primitives arose to generalize FE to the multi-user setting – multi-input functional encryption [24], multi-client functional encryption [20], decentralized multi-client functional encryption, adhoc multi-input functional encryption [5], multi-authority attribute based encryption [17], dynamic decentralized functional encryption [22] and such others. Similar to the many special cases of functional encryption, these notions are related yet different and it is often difficult to understand how they compare to one-another, whether they use related techniques, and what is known in terms

of feasibility. Moreover, each new variant that springs up acquires a different name, leading to a plethora of acronyms which clutter the landscape, often adding to confusion rather than clarity.

In this work, we initiate the study of "Multi-Party Functional Encryption" (MPFE) which unifies and abstracts out various notions of multi-user functional encryption, such as those described above. Our starting point is the observation that all above notions of FE support some form of distributed ciphertexts or distributed keys or both. In more detail, we summarize the state of affairs as:

1. **Distributed Ciphertexts.** The primitives of multi-input functional encryption (MIFE) [24] and multi-client functional encryption (MCFE) [20] generalize FE to support distributed inputs. Both notions permit different parties $\mathsf{P}_1, \ldots, \mathsf{P}_n$ each with inputs $\mathbf{x}_1, \ldots, \mathbf{x}_n$ to compute joint functions on their data, namely $f(\mathbf{x}_1, \ldots, \mathbf{x}_n)$. Each party encrypts its input $\mathbf{x}_i$ to obtain $\mathsf{CT}_i$, a key authority holding a master secret $\mathsf{MSK}$ generates a functional key $\mathsf{SK}_f$ and these enable the decryptor to compute $f(\mathbf{x}_1, \ldots, \mathbf{x}_n)$.

   The main difference between these definitions lies in the way the inputs can be combined. In multi-*client* functional encryption (MCFE), inputs $\mathbf{x}_i$ are additionally associated with public "labels" $\mathsf{lab}_i$ and inputs can only be combined with other inputs that share the same label. On the other hand, multi-input functional encryption does not restrict the way that inputs are combined and permits all possible combinations of inputs. Both primitives are defined as *key policy* systems – namely, the access control policy or function is embedded in the secret key rather than the ciphertext.
2. **Distributed Keys.** Distribution or decentralization of keys in the context of FE has also been considered in various works, to achieve two primary objectives (not necessarily simultaneously) – a) handling the *key escrow* problem, so that there is no single entity in the system that holds a powerful master secret against which no security can hold, and b) *better fitting real world* scenarios where different authorities may be responsible for issuing keys corresponding to different attributes of a user, such as offices for passport, drivers license and such others. We summarize some relevant primitives next.
   (a) *Decentralized Attribute Based Encryption with Policy Hiding* (DABE): A decentralized policy-hiding ABE, denoted by DABE [31] was proposed by Michalevsky and Joye to handle the key escrow problem. In a DABE scheme, there are $n$ key authorities, each of which run a local setup to generate their private and public keys. An encryptor encrypts a message $m$ along with a general access structure $C$, while secret keys corresponding to (the same) attribute $\mathbf{x}$ are issued by independent authorities. Decryption recovers $m$ if $C(\mathbf{x}) = 1$. The access policy in the ciphertext is hidden.
   (b) *Multi-Authority Functional Encryption* (MAFE): The notion of Multi-Authority FE/ABE [17,29,16] emerged to address the second objective, i.e. handling the case where different authorities are responsible for different sets of attributes. Since ABE is a special case of FE, we focus on MAFE. A MAFE scheme is defined as a ciphertext-policy scheme, namely the policy/function is embedded in the ciphertext as against the function keys.

In MAFE, $n$ key authorities may independently generate their private and public keys, without any interaction. An encryptor computes a ciphertext for a message $m$ along with a policy $f$ over the various authorities. Any authority $i$, can generate a token for a user $\mathsf{P}$ for attributes $\mathsf{lab}_i$. A decryptor with tokens for $\mathsf{lab}_i$ from authority $i \in [n]$, can decrypt the ciphertext to recover $f(\mathsf{lab}_1, \ldots, \mathsf{lab}_n, m)$.

3. **Distributed Ciphertexts and Keys.** Some primitives allow to distribute both ciphertexts and keys. Some examples below.
   (a) *Decentralized Multi-Client Functional Encryption* (D-MCFE): The notion of decentralized multi-client FE was defined by Chotard et al. [20,2,30] in order to handle the key escrow problem in an MCFE scheme. D-MCFE is defined as a key policy primitive, and adapts MCFE as described above to ensure that there is no single master secret held by any entity – the parties participate in an interactive setup protocol to establish their individual (correlated) master secret keys. In more detail, there are $n$ parties, each holding $\mathsf{MSK}_i$ for $i \in [n]$, that compute ciphertexts for their inputs $(\mathsf{lab}_i, \mathbf{x}_i)$ as well as generate partial decryption keys $\mathsf{SK}_{i,f}$ for a given function $f$. The decryptor can combine the partial secret keys and individual ciphertexts to compute $f(\mathbf{x}_1, \ldots, \mathbf{x}_n)$ if and only if all the labels are equal.
   (b) *Ad Hoc MIFE* (aMIFE): Similar to D-MCFE, this notion was introduced in [5] to handle the key escrow problem in MIFE. This notion is key policy, and offers some additional features as compared to D-MCFE — non-interactive setup and dynamic choice of function arity as well as parties that participate in a computation. This notion does not differentiate between key authorities and users, and lets users generate their own partial decryption keys along with ciphertexts. Thus, for $i \in [n]$, party $i$ computes a ciphertext for $\mathbf{x}_i$ and partial key $\mathsf{SK}_{f,i}$ which can be combined by the decryptor to obtain $f(\mathbf{x}_1, \ldots, \mathbf{x}_n)$.
   (c) *Dynamic Decentralized FE* (DDFE): This primitive was introduced very recently in [22] to further generalize aMIFE – it requires non-interactive, local setup and allows dynamic choice of function arity as in aMIFE, but additionally allows partial decryption keys provided by users to be combined in more general ways than in aMIFE. Also, unlike aMIFE, it supports the public key setting.

### 1.1 Unifying the View: Multi-Party Functional Encryption

While the above notions enable controlled manipulation of encrypted data in increasingly expressive ways, they are too related to warrant independent identities. To unify and extend the above primitives, we propose the notion of multi-party functional encryption (MPFE). All the above examples (and more) can be cast as examples of MPFE with a suitable choice of parameters: this clarifies the connections between these primitives. MPFE allows for both distributed ciphertexts and distributed keys, and specifies how these may be combined

for function evaluation. To avoid bifurcating key-policy and ciphertext-policy schemes, we allow either ciphertext or key inputs to encode functions. To better capture attribute and function hiding, we allow every message or function being encoded to have a public and private part. To support schemes with interactive, independent or centralized setup, we allow the setup algorithm of $\mathsf{MPFE}$ to function in any of these modes.

A bit more formally, let $n_x$ be the number of ciphertext inputs and $n_y$ be the number of key inputs. Let $\mathcal{X} = \mathcal{X}_{\text{pub}} \times \mathcal{X}_{\text{pri}}$ be the space of ciphertext inputs and $\mathcal{Y} = \mathcal{Y}_{\text{pub}} \times \mathcal{Y}_{\text{pri}}$ be the space of key inputs. We define two aggregation functions as $\mathsf{Agg}_x : \mathcal{X}^{n_x} \to \mathcal{X}$, and $\mathsf{Agg}_y : \mathcal{Y}^{n_y} \to \mathcal{Y}$, which specify how these inputs may be combined to capture a given primitive. The definitions of the algorithms that constitute an $\mathsf{MPFE}$ scheme are the same as in all prior work:

- a $\mathsf{Setup}$ algorithm outputs the encryption keys for $n_x$ encryptors and master keys for $n_y$ key authorities. This algorithm[4] may now run in one of three modes ($\mathsf{Central}, \mathsf{Local}, \mathsf{Decentralized}$), which captures centralized setup, local/independent setup or decentralized/interactive setup.
- an $\mathsf{Encrypt}$ algorithm which is run independently by $n_x$ users, each encoding their own message $x_i = (x_{\text{pub},i}, x_{\text{pri},i})$ with their own encryption key $\mathsf{EK}_i$.
- a key-generation algorithm $\mathsf{KeyGen}$ which is run independently by all $n_y$ key authorities, each generating its own partial key for an input $y_j = (y_{\text{pub},j}, y_{\text{pri},j})$ of its choice using its own master secret key $\mathsf{MSK}_j$.
- a decryption algorithm $\mathsf{Decrypt}$, which given input the partial keys $\{\mathsf{SK}_i\}_{i \le n_y}$ and partial ciphertexts $\{\mathsf{CT}_j\}_{j \le n_x}$ can combine them to compute $\mathcal{U}\left(\mathsf{Agg}_x(\{x_i\}), \mathsf{Agg}_y(\{y_j\})\right)$, where $\mathcal{U}$ is the universal circuit.

Note that either $x_i$ or $y_j$ can be descriptions of functions, capturing both key and ciphertext policy schemes. By suitably choosing $n_x$, $n_y$, $\mathsf{Agg}_x$, $\mathsf{Agg}_y$ and the mode of setup, namely ($\mathsf{mode} \in \{\mathsf{Central}, \mathsf{Local}, \mathsf{Decentralized}\}$), the above abstraction lets us specify all the aforementioned primitives in a unified manner, and also allows us to instantiate these parameters in different ways to yield new, meaningful primitives. Please see Section 2 for the formal definition and the full version [6] for details on how the above primitives can be expressed as instances of $\mathsf{MPFE}$.

*Dynamic* $\mathsf{MPFE}$. In the above description, we assume that the number of parties as well as the aggregation functions are input to the setup algorithm. A more powerful definition could support full dynamism, where the parties generate their own keys, join the protocol dynamically without prior agreement, and choose the functionality (in our case $\mathsf{Agg}_x$ and $\mathsf{Agg}_y$) dynamically so that it can change for every instance of the protocol.

[4] If the setup mode is decentralized/interactive, then the description of setup could correspond to an interactive multi-round protocol instead of an algorithm. However, for ease of exposition we abuse the notation and use setup algorithm to refer to the corresponding protocol description.

While dynamism is obviously desirable, it is significantly harder to instantiate since it necessitates a local setup algorithm without any co-ordination between the parties. While there do exist some constructions for dynamic FE supporting multiple users, such as adhoc MIFE [5] and DDFE [22], most constructions in the literature are "static" and rely on centralized or interactive setup [24,4,19,2,30,1,20,21]. Thus, a definition which is inherently dynamic would preclude representation of most constructions in the literature.

For simplicity of notation and ease of workability, we define MPFE with and without dynamism separately. We provide the definition of the static variant in Section 2 and the dynamic variant in the full version. We note that these two variants may be condensed to a single one using additional notation but this makes the definitions harder to work with.

*Feasibility.* In the full version, we provide a general feasibility of MPFE for circuits from the minimal assumption of MIFE for circuits.

### 1.2 Comparison with Prior Work

The notions of D-MCFE, aMIFE and DDFE are most closely related to our work, since they allow combining both ciphertexts and keys simulataneously. However, our notion differs from these in important ways. To begin, the setup algorithms of the above primitives have a fixed format – in D-MCFE, this is interactive, while for aMIFE and DDFE, it is decentralized and non-interactive. Thus, aMIFE and DDFE cannot capture D-MCFE and vice versa. Moreover, neither of these can capture most existing constructions in the literature which have trusted, centralized setup as discussed above. In contrast, we allow setup to have either of these, as well as other formats, allowing us to capture all the above primitives and more. Next, D-MCFE, aMIFE require partial keys to represent the same function. While DDFE does allow partial keys to be combined in expressive ways, it does not support any function hiding. Even the support for partial input hiding in these primitives is less than complete: for instance, aMIFE does not support public input in the ciphertext, and while DDFE allows for some part of the input to be public, this is via a separate empty key $\epsilon$. In contrast, MPFE captures public and private input in both the ciphertext and the function key directly, making it feasible (in the case of function inputs) and simpler (in the case of ciphertext inputs) to capture partial hiding.

The most important feature of MPFE is that is captures existing constructions using a uniform, simple notation, allowing to place all prior work on the same map, making these constructions easier to compare and allowing to identify gaps between these. Using our MPFE framework, we interpolate the space in prior work to predict several new, natural and useful primitives. Then, we provide multiple new constructions from simple, standard assumptions to address these limitations (described next), as well as identify novel new primitives (described in Section 1.5) to be constructed in future work.

### 1.3 New Constructions

We next describe the new constructions we provide in this work.

**Multi-Authority $\mathsf{ABE} \circ \mathsf{IPFE}$.** The recent work of Abdalla et al. [4] ($\mathsf{ACGU20}$) constructed a novel "composition" of $\mathsf{ABE}$ and $\mathsf{IPFE}$, namely functional encryption schemes that combine the access control functionality of attribute based encryption with the possibility of performing linear operations on the encrypted data. In more detail, the message space contains a policy predicate $\phi \in \mathsf{NC}_1$ and a message vector $\mathbf{v} \in \mathbb{Z}_q^\ell$, while decryption keys are jointly associated with an attribute vector $\mathbf{x} \in \{0,1\}^n$ and a key vector $\mathbf{u} \in \mathbb{Z}_q^\ell$. The functionality provided by such a system is that a decryptor recovers the inner product value $\langle \mathbf{u}, \mathbf{v} \rangle$ if $\phi(\mathbf{x}) = 1$. Thus, it provides a fine-grained access control on top of inner product functional encryption ($\mathsf{IPFE}$) capability. For ease of exposition, we denote this primitive, which is called "$\mathsf{IPFE}$ with fine-grained access control" in [4] by $\mathsf{ABE} \circ \mathsf{IPFE}$ in our work[5]. Abdalla et al. [4] provide a construction leveraging state of the art $\mathsf{ABE}$ from pairings to support predicates represented by Linear Secret Sharing Schemes (LSSS) in the above functional encryption scheme.

Seen from the lens of $\mathsf{MPFE}$, the $\mathsf{ACGU20}$ construction has $n_x = n_y = 1$, with $(x_{\text{pub}}, x_{\text{pri}}) = (\phi, \mathbf{v})$, $(y_{\text{pub}}, y_{\text{pri}}) = ((f_{\mathbf{x}}, \mathbf{u}), \perp)$ where $f_{\mathbf{x}}$ is a function that takes as input three arguments $(\phi, \mathbf{v}, \mathbf{u})$ and outputs $\langle \mathbf{u}, \mathbf{v} \rangle$ if $\phi(\mathbf{x}) = 1$. The aggregation functions are trivial as there is only a single encryptor and key generator. In this work, we extend the $\mathsf{ACGU20}$ construction to the multiparty setting. In more detail, we support $n_y = n$ for some fixed, polynomial $n$ and $\mathsf{Local}$ mode of setup algorithm, so that each key generator generates its key components locally and independently. The number of encryptors $n_x$ as well as the $(x_{\text{pub}}, x_{\text{pri}})$ remain unchanged. However, each of the $n$ key generators now has input $(y_{\text{pub}}, y_{\text{pri}}) = ((\mathsf{GID}_i, x_i, \mathbf{u}_i), \perp)$ where $\mathsf{GID}_i \in \{0,1\}^*$ is a global identifier, $x_i \in \{0,1\}$ is an attribute bit, and $\mathbf{u}_i \in \mathbb{Z}_q^\ell$ is the key vector for $i \in [n]$. The $\mathsf{Agg}_x$ function remains trivial as before but the $\mathsf{Agg}_y$ function checks if all the global identifiers match $\mathsf{GID}_1 = \ldots = \mathsf{GID}_n$, key vectors are consistent $\mathbf{u}_1 = \ldots = \mathbf{u}_n$, and sets $(y_{\text{pub}}, y_{\text{pri}}) = ((f_{\mathbf{x}}, \mathbf{u}), \perp)$ if so, where $\mathbf{x} = (x_1, \ldots, x_n)$ and $f_{\mathbf{x}}$ is as above.

The above generalization has been studied in the literature in the context of $\mathsf{ABE}$ under the name *multi-authority* $\mathsf{ABE}$, or $\mathsf{MA\text{-}ABE}$ – here, we extend the access control component of $\mathsf{ACGU20}$ to support the multi-authority setting, i.e. "lift" the primitive of $\mathsf{ABE} \circ \mathsf{IPFE}$ to $\mathsf{MA\text{-}ABE} \circ \mathsf{IPFE}$. Our construction departs significantly from $\mathsf{ACGU20}$ in details – our starting point is the $\mathsf{MA\text{-}ABE}$ construction of Lewko and Waters [29] which we extend to support inner product computation. This yields the first construction of a nontrivial multi-authority $\mathsf{FE}$ beyond $\mathsf{ABE}$ from simple assumptions on pairings to the best of our knowledge.

Using our techniques, we also extend the decentralized attribute based encryption ($\mathsf{DABE}$) scheme of Michalevsky and Joye [31] to support inner product computations. While [31] only supports inner product predicates unlike [29], it supports policy hiding unlike the latter – our extension inherits these features.

[5] We caution the reader that the notation $\mathsf{ABE} \circ \mathsf{IPFE}$ is for readability and does not denote a formal composition.

**Function Hiding** DDFE**.** The novel primitive of *dynamic* decentralized inner product functional encryption (IP-DDFE) was recently introduced by Chotard et al. [22], where they also provided the first construction. As discussed above, DDFE is an instance of dynamic MPFE. Using the notation of MPFE, we have the setup algorithm in the Local mode, so that each party $i$ can dynamically join the system by generating a public key $\mathsf{PK}_i$ and a master secret key $\mathsf{MSK}_i$. For encryption, party $i$ sets $(x_{\text{pub}}, x_{\text{pri}}) = \big((\mathcal{U}_M, \mathsf{lab}_M), \mathbf{x}_i\big)$ where $\mathcal{U}_M$ is the set of parties whose inputs will be combined and $\mathsf{lab}_M$ is a label which imposes a constraint on which values can be aggregated together. For key generation, party $i$ sets $(y_{\text{pub}}, y_{\text{pri}}) = \big((\mathbf{y}_i, \mathcal{U}_K, \mathbf{y}), \perp\big)$ where $\mathcal{U}_K$ is a set of public keys that defines the support of the inner product, and $\mathbf{y}$ is an agreed upon vector $\mathbf{y} = \{\mathbf{y}_i\}_{i \in \mathcal{U}_K}$. The function $\mathsf{Agg}_x$ checks if the public inputs $(\mathcal{U}_M, \mathsf{lab}_M)$ match for all parties and that all the ciphertexts are provided for the set $\mathcal{U}_M$. If so, outputs $(\mathcal{U}_M, \mathbf{x})$ where $\mathbf{x} = (\mathbf{x}_1 \| \dots \| \mathbf{x}_{n_x})$. The function $\mathsf{Agg}_y$ checks that all values $\mathcal{U}_K$ and $\mathbf{y}$ are the same for all parties, and that value $\mathbf{y}_i$ matches with its corresponding component in the agreed vector. If so, it outputs the function $f_{\mathcal{U}_K, \mathbf{y}}$ which takes as input $(\mathcal{U}_M, \mathbf{x})$, checks that $\mathcal{U}_M = \mathcal{U}_K$ and if so, outputs $\langle \mathbf{x}, \mathbf{y} \rangle$.

However, as discussed before, the primitive of DDFE does not support function hiding. We see this as a significant limitation of this notion. Function hiding is a well studied and very useful property with many applications – for instance, it allows parties to securely delegate computation to an untrusted server without the server being able to learn the functionality. In some cases, knowing the functionality and the output (which the server computes in the clear) may leak information about the underlying data. In other cases, the functionality itself may be private and protected by copyright laws. In our work, we provide a new construction for IP-DDFE which supports function hiding. In more detail, the key generator, similar to the encryptor associates a label $\mathsf{lab}_K$ with its vector $\mathbf{y}_i$ and combining partial keys is only possible when their labels match. Importantly, the key vector $\mathbf{y}_i$ may now be *hidden* analogously to the vector $\mathbf{x}_i$ in the ciphertext.

In more detail, for key generation, party $i$ sets $(y_{\text{pub}}, y_{\text{pri}}) = \big((\mathcal{U}_K, \mathsf{lab}_K), \mathbf{y}_i\big)$ where $\mathcal{U}_K, \mathsf{lab}_K$ have the same roles as $\mathcal{U}_M, \mathsf{lab}_M$, respectively. The function $\mathsf{Agg}_y$, analogously to $\mathsf{Agg}_x$ checks that all values $\mathcal{U}_K$ and $\mathsf{lab}_K$ are the same for all parties. If so, it outputs the function $f_{\mathcal{U}_K, \mathbf{y}=(\mathbf{y}_1 \| \dots \| \mathbf{y}_{n_y})}$ which takes as input $(\mathcal{U}_M, \mathbf{x})$, checks that $\mathcal{U}_M = \mathcal{U}_K$ and if so, outputs $\langle \mathbf{x}, \mathbf{y} \rangle$.

To achieve our final result, we define and construct the first *function hiding* MCFE scheme for inner products, which may be of independent interest.

**Ciphertext-Policy** ABE **with Distributed Key Generation.** We provide a multiparty variant of the recent ciphertext-policy attribute based encryption scheme, constructed by Agrawal and Yamada [8]. In our scheme, the setup algorithm is run in the Local mode and key generation is distributed amongst $n_y = n$ parties for any polynomial $n$. As in single-party ABE, we have $n_x = 1$ (hence $\mathsf{Agg}_x$ is trivial) where $(x_{\text{pub}}, x_{\text{pri}}) = (C, m)$ where $C$ is a circuit in $\mathsf{NC}_1$ and $m$ is a hidden bit. For key generation, the $i^{th}$ party produces a key for $(y_{\text{pub}}, y_{\text{pri}}) = ((\mathbf{y}, \mathsf{GID}, y_i), \perp)$ where GID is a global identifier, and $\mathbf{y}$ is an agreed

upon vector $\mathbf{y} = (y_1, \ldots, y_n)$. The aggregation function $\mathsf{Agg}_y$ checks if all the values $\mathsf{GID}$ and $\mathbf{y}$ are the same, and that value $y_i$ matches with its corresponding component in the agreed vector $\mathbf{y}$. It then outputs a function $f_{\mathbf{y}}$ which takes as input a circuit $C$ and message $m$ and outputs $m$ if $C(\mathbf{y}) = 1$. Our construction is secure based on "Learning With Errors" and relies on the generic bilinear group model as well as the random oracle model. We show that as long as at least one authority is honest, the scheme remains secure.

### 1.4 Technical Overview

In this section, we provide an overview of the techniques used for our constructions. We begin with our two constructions that extend multi-authority schemes [29,31] to support inner products.

**Multi-Authority $\mathsf{ABE} \circ \mathsf{IPFE}$ for LSSS Access Structures.** We described the functionality of $\mathsf{MA\text{-}ABE} \circ \mathsf{IPFE}$ in Section 1.3. Security is defined in a multifold setting where: (1) adversary is allowed to corrupt the key authorities, (2) make key queries that do not satisfy the challenge policy predicate $\phi^*$, and (3) also make key queries that satisfy the challenge policy predicate $\phi^*$ but decrypt to the same value for both challenge vectors (that is, $\langle \mathbf{u}, \mathbf{v}_0^* \rangle = \langle \mathbf{u}, \mathbf{v}_1^* \rangle$).

A natural first line of attack is to consider whether such a scheme can generically be built from combining these two primitives. As it turns out, any such generic construction suffers from the common problem of mix and match attacks, that is, we must prevent an authorized $\mathsf{MA\text{-}ABE}$ portion of the key from being used along with an $\mathsf{IPFE}$ portion of an unauthorized key. Another idea is to extend the $\mathsf{ABE} \circ \mathsf{IPFE}$ construction of [4] to support multiple authorities. However, this work relies on the predicate encoding framework which is not suitable as-is for our application. Instead, our approach is to start with the multi-authority ABE construction by Lewko and Waters [29] for LSSS access structures, and show how to leverage it's intrinsic algebraic structure to add an inner product functionality "on top" of the multi-authority ABE construction.

To begin, we provide an informal sketch of a simplified version of our construction. Recall that an access policy corresponding to a linear secret sharing scheme access structure contains a share generating matrix $\mathbf{A}$ and a row index to party index mapping function $\rho$.

$\mathsf{LSetup}$ : The $i$-th authority samples a length $\ell$ masking vector $\boldsymbol{\alpha}_i$ as its secret key, and publishes its encoding $[\boldsymbol{\alpha}_i]_T$ in the target group as the public key.

$\mathsf{KeyGen}$ : To generate a secret key for key vector $\mathbf{u}$, the $i$-th authority projects $\boldsymbol{\alpha}_i$ on the vector space defined by key vector $\mathbf{u}$. That is, if the attribute bit $x_i$ is 1[6], then the partial decryption key is simply $[\langle \boldsymbol{\alpha}_i, \mathbf{u} \rangle]$.

$\mathsf{Enc}$ : For encrypting a message vector $\mathbf{v}$ under an access policy $(\mathbf{A}, \rho)$, the encryptor first secret shares the message vector $\mathbf{v}$ using the access policy $\mathbf{A}$ into a share matrix $\mathbf{S}_{\mathbf{v}}$. That is, $\mathbf{S}_{\mathbf{v}}$ is a random matrix with the property that for each accepting attribute $\mathbf{x}$ there exists a reconstruction vector $\mathbf{z}_{\mathbf{x}}$

[6] As in prior ABE schemes based on bilinear maps, the key is empty when $x_i = 0$.

such that $\mathbf{z}_{\mathbf{x}}^{\top} \cdot \mathbf{S}_{\mathbf{v}} = \mathbf{v}^{\top}$. It next arranges the authority public keys $[\boldsymbol{\alpha}_i]_T$ row-wise in a matrix $\boldsymbol{\Delta}$ as per the function $\rho$, that is $i$-th row on $\boldsymbol{\Delta}$ is $\rho(i)$-th public key $[\boldsymbol{\alpha}_{\rho(i)}]_T$. Finally, it output the ciphertext as the following matrix

$$\mathsf{CT}_0 = \left[\mathbf{S}_{\mathbf{v}} + \boldsymbol{\Delta} \odot (\mathbf{r} \otimes \mathbf{1}^{\top})\right]_T, \quad \mathsf{CT}_1 = [\mathbf{r}],$$

where $\mathbf{r}$ is a random vector of appropriate dimension and $\odot$ denotes the component-wise multiplications between two matrices of same dimensions.

Dec : A decryptor then simply left-multiplies $\mathsf{CT}_0$ with the reconstruction vector $\mathbf{z}_{\mathbf{x}}$ and right-multiplies with the key vector $\mathbf{u}$ to compute the following:

$$\begin{aligned}
\mathbf{z}_{\mathbf{x}}^{\top} \cdot \mathsf{CT}_0 \cdot \mathbf{u} &= \mathbf{z}_{\mathbf{x}}^{\top} \cdot \left[\mathbf{S}_{\mathbf{v}} + \boldsymbol{\Delta} \odot (\mathbf{r} \otimes \mathbf{1}^{\top})\right]_T \cdot \mathbf{u} \\
&= \left[\mathbf{v}^{\top} \cdot \mathbf{u} + \mathbf{z}_{\mathbf{x}}^{\top} \cdot (\boldsymbol{\Delta} \odot (\mathbf{r} \otimes \mathbf{1}^{\top})) \cdot \mathbf{u}\right]_T \\
&= \left[\mathbf{v}^{\top} \cdot \mathbf{u} + \mathbf{z}_{\mathbf{x}}^{\top} \cdot (\boldsymbol{\Delta} \odot (\mathbf{r} \otimes \mathbf{u}^{\top}))\right]_T
\end{aligned}$$

It next arranges the partial decryption keys $\left[\langle \boldsymbol{\alpha}_i, \mathbf{u} \rangle\right]$ row-wise in a vector $\mathbf{K}$ as per the function $\rho$, that is $i$-th element of $\mathbf{K}$ is $\rho(i)$-th decryption key $\left[\langle \boldsymbol{\alpha}_{\rho(i)}, \mathbf{u} \rangle\right]$. It performs the component pairing between $\mathbf{K}$ and $\mathsf{CT}_1$, and then takes the linear combination as specified by $\mathbf{z}_{\mathbf{x}}$ which can be simplified as follows:

$$\mathbf{z}_{\mathbf{x}}^{\top} \cdot e\left(\mathbf{K}, \mathsf{CT}_1\right) = \left[\mathbf{z}_{\mathbf{x}}^{\top} \cdot (\boldsymbol{\Delta} \odot (\mathbf{r} \otimes \mathbf{u}^{\top}))\right]_T$$

Finally, it can recover $\left[\mathbf{v}^{\top} \cdot \mathbf{u}\right]_T$ from the above two terms, and learn the exponent value by brute force search.

Now in the above sketch we ignored the global identifier $\mathsf{GID}$ that is necessary for tying together the partial decryption keys provided by each authority, and we also ignore the modifications necessary for proving security under standard bilinear assumptions. At a very high level, for proving security we rely on ideas from the dual system paradigm [34] as in the multi-authority ABE scheme of [29]. However, we must deal with several new challenges to adapt this paradigm to our setting, as we describe next.

In the dual system paradigm, the intuition is that the reduction algorithm first switches all the secret keys to semi-functional keys, and thereafter it also switches the challenge ciphertext to a semi-functional ciphertext, and after both these changes security follows directly from the property that semi-functional secret keys and ciphertexts are not compatible for decryption. In $\mathsf{IPFE}$, we cannot hope to execute the same strategy directly since now we cannot switch all secret keys to semi-functional keys since some secret keys might allow decrypting the challenge ciphertext (but they still would not help in distinguishing by admissibility constraints on the attacker). At this point, we define the concept of *partial* semi-functional ciphertexts such that (at a high level) we first switch all the rejecting secret keys to semi-functional while leaving the accepting keys as is, and thereafter we switch the challenge ciphertext to be a "partial" semi-functional ciphertext such that this hides the non-trivial information about the encrypted message vectors.

Although this intuition seems to work at a high level, it is still insufficient since it is unclear how to switch the entire ciphertext to semi-functional in the standard model. To that end, our idea is to switch all the accepting secret keys (including the ones for satisfying predicates) to their semi-functional counterparts as well, but now ensure that the challenge ciphertext components that the accepting keys interact with are only *nominally* semi-functional. Here the difference between a regular ciphertext, a nominally semi-functional, and a standard semi-functional ciphertext is that – regular ciphertexts lie in a special subgroup with no special blinding terms; while nominally semi-functional ciphertexts have structured blinding factors outside the special subgroup but it does not affect decryption irrespective of the type of secret key being used; and a standard semi-functional ciphertext has unstructured blinding factors outside the special subgroup such that it affects decryption when using semi-functional keys. Now switching portions of the challenge ciphertext as nominally semi-functional is necessary because of two reasons: first, making the entire challenge ciphertext semi-functional will affect decryption w.r.t. accepting keys which will be distinguishable for the adversary; second, it is unclear how to sample the challenge ciphertext in which only one component is semi-functional while other are regular sub-encryptions due to the fact that these different ciphertext components are significantly correlated. Thus, we get around this barrier by ensuring that the challenge ciphertext is sampled as what we call a partial semi-functional ciphertext (which has nominally semi-functional components along with a standard semi-functional component).

Please see Section 3 for the formal construction and proof. Our construction relies on standard assumptions over composite-order bilinear groups, but could be also be easily adapted to prime-order groups with a security proof in the generic group model as in [29].

**$\mathsf{DABE} \circ \mathsf{IPFE}$ for Inner Product Predicates, with Policy Hiding.** Next, we extend the construction of decentralized attribute based encryption by Michalevsky and Joye [31] to incorporate inner product functional encryption. Observe that [31] supports only inner product predicates but allows for hiding the policy in the ciphertext. While our extension to the scheme of [31] also yields a multi-authority $\mathsf{ABE}$ extended to support inner products as above, the details of the transformation are quite different. We observe that the algebraic structure of [31] makes it amenable to incorporating the $\mathsf{IPFE}$ functionality using ideas developed in the literature for constructing $\mathsf{IPFE}$ generically from public-key encryption which have special structural and homomorphic properties [3,7,10]. We proceed to describe this transformation next.

In an overly simplified version of the Michalevsky-Joye construction, one could interpret the $i$-th key authority as simply sampling a pair of secret exponents $\delta_i, w_i \leftarrow \mathbb{Z}_p$, where $\delta_i$ is regarded as the partial message masking term, while $w_i$ is considered the $i$-th attribute bit binding term. Now each authority's public key is simply set as the group encodings $[\delta_i]$ and $[w_i]$. Implicitly, the scheme uses the linear combination of partial message masking terms $\delta = \sum_i \delta_i$ to derive the main message masking term (used for deriving the secret key encapsulating the message, or the KEM key in short).

To encrypt a message $m$ under attribute $\mathbf{x}$, the user chooses randomness $r \leftarrow \mathbb{Z}_p$ and computes $\left[r\delta\right]_T$ to be used as the KEM key, and binds each attribute bit to a ciphertext component as $\left[(x_i\alpha + w_i)r\right]$ (where $\left[\alpha\right]$ is taken from the CRS). It sets the ciphertext to be $C_0 = \left[r\right]$, $C_m = m \cdot \left[r\delta\right]_T$, and $C_i = \left[(x_i\alpha + w_i)r\right]$ for $i \in [n]$. While a partial secret key for policy vector $\mathbf{y}$ for user $\mathsf{GID}$ is simply generated as $K_{i,\mathbf{y}} = \left[\delta_i - y_i w_i h\right]$ where $\left[h\right]$ is computed as $H(\mathsf{GID})$ so as to bind the different authorities' secret keys. The decryption can be simply performed given the bilinear operation as:

$$\begin{aligned}\mathsf{Dec}(\{K_{i,\mathbf{y}}\}_i, \mathsf{CT}) &= \frac{C_m}{\prod_i e(C_i, H(\mathsf{GID})^{y_i}) \cdot \prod_i e(K_{i,\mathbf{y}}, C_0)} \\ &= \frac{m \cdot \left[r\delta\right]_T}{\left[\langle \mathbf{x}, \mathbf{y}\rangle \alpha r h + \cancel{\sum_i w_i h y_i r}\right]_T \cdot \left[\delta r - \cancel{\sum_i y_i w_i h r}\right]_T} = \frac{m}{\left[\langle \mathbf{x}, \mathbf{y}\rangle \alpha r h\right]_T}\end{aligned}$$

As discussed above, we upgrade the [31] construction using ideas from [3,7,10] as follows. During key generation, each authority now samples a vector of partial masking terms instead of a single element, i.e. $\boldsymbol{\delta}_i \leftarrow \mathbb{Z}_p^\ell$, and appropriately sets the public key too. Implicitly, the main message masking term is now set as $\boldsymbol{\delta} = \sum_i \boldsymbol{\delta}_i$. To encrypt a message vector $\mathbf{u}$ under attribute vector $\mathbf{x}$, the user chooses randomness $r \leftarrow \mathbb{Z}_p$ and computes $\left[r\boldsymbol{\delta}\right]_T$ to be used as the KEM key for encrypting $\mathbf{u}$ index-by-index, and binds the attribute bit as before. Thus, only the message binding ciphertext component changes to $C_m = \left[r\boldsymbol{\delta} + \mathbf{u}\right]_T$. Looking ahead, it will be decryptor's job to first homomorphically take an inner product between the $C_m$ vector and the inner product key vector $\mathbf{v}$. Next, a partial secret key for policy vector $\mathbf{y}$ and inner-product vector $\mathbf{v}$ for user $\mathsf{GID}$ is generated as $K_{i,\mathbf{y},\mathbf{v}} = \left[\sum_j \delta_{i,j} v_j - y_i w_i h\right]$. In words, the idea here is that the partial secret key now uses a linear combination of its partial (un)masking term $\sum_j \delta_{i,j} v_j$ depending on the underlying inner-product vector $\mathbf{v}$. The decryption can be naturally extended by performing inner products via the bilinear operations.

As in the case of our first construction, the proof techniques in [31] do not apply directly as they were specially designed for ABE which is an all-or-nothing encryption primitive, and do not translate directly to $\mathsf{IPFE}$. Again, we solve this issue by a careful analysis in the dual system paradigm [34]. We refer the reader to the full version for more details.

**Function-Hiding $\mathsf{DDFE}$ for Inner Products.** In this section, we describe the main ideas in the construction of our function hiding $\mathsf{DDFE}$ for inner products. The functionality of $\mathsf{IP\text{-}DDFE}$ was discussed in Section 1.3. Informally, the security of $\mathsf{DDFE}$ requires that the adversary cannot distinguish two sets $\{\mathsf{CT}_i^0\}$ and $\{\mathsf{CT}_i^1\}$ of ciphertexts even given a set $\{\mathsf{SK}_i\}$ of secret keys and a set $\{\mathsf{MSK}_i\}$ of master secret keys of corrupted parties as long as two sets of values are the same that are legitimately obtained from $\{\mathsf{CT}_i^0\}$ and $\{\mathsf{CT}_i^1\}$ using $\{\mathsf{SK}_i\}$ and $\{\mathsf{MSK}_i\}$. Let us recall dynamic decentralized inner product functional encryption ($\mathsf{IP\text{-}DDFE}$) by Chotard et al. [22].

The starting point of the $\mathsf{IP\text{-}DDFE}$ scheme of [22] is the multi-client inner product functional encryption ($\mathsf{IP\text{-}MCFE}$) scheme in [20], where participants

$\{1, ..., n\}$ in the system are a priori fixed, and there is an authority who generates encryption keys $\mathsf{mcEK}_i$ for each party and a master secret key $\mathsf{mcMSK}$, which is used to generate secret keys $\mathsf{mcSK}$. Here, $\mathsf{mcMSK} = \{\mathsf{mcMSK}_i\}_{i\in[n]}$ and $\mathsf{mcEK}_i = \mathsf{mcMSK}_i$ (and we denote an encryption key for $i$ by $\mathsf{mcMSK}_i$ in what follows). We also recall that in $\mathsf{MCFE}$, only a set of ciphertexts with the same label can be decrypted. Chotard et al. [22] lifted the $\mathsf{IP\text{-}MCFE}$ scheme to an $\mathsf{IP\text{-}DDFE}$ scheme via following steps. First, each party joins the system dynamically by generating a key $\mathsf{K}_i$ of a pseudorandom function (PRF) as a master secret key $\mathsf{MSK}_i$. In encryption and key generation for party set $\mathcal{U}$, party $i \in \mathcal{U}$ generates $\mathsf{mcMSK}_{i,\mathcal{U}}$ on the fly, which corresponds to $\mathsf{mcMSK}_i$ of the $\mathsf{IP\text{-}MCFE}$ scheme for participants $\mathcal{U}$. Then, it can generate $\mathsf{mcCT}_{i,\mathcal{U}}$ and $\mathsf{mcSK}_{i,\mathcal{U}}$ with $\mathsf{mcMSK}_{i,\mathcal{U}}$, which corresponds to $\mathsf{CT}_i$ and $\mathsf{SK}_i$ of the $\mathsf{IP\text{-}DDFE}$ scheme. Second, they introduce a class of $\mathsf{DDFE}$ called $\mathsf{DSum}$, which allows a decryptor to securely obtain $\mathsf{mcSK}_{\mathcal{U}} = \sum_{i\in\mathcal{U}} \mathsf{mcSK}_{i,\mathcal{U}}$ from encryption of partial secret keys $\{\mathsf{mcSK}_{i,\mathcal{U}}\}_{i\in\mathcal{U}}$. Then, the decryptor can compute $\mathsf{mcDec}(\mathsf{mcSK}_{\mathcal{U}}, \{\mathsf{mcCT}_{i,\mathcal{U}}\}_{i\in\mathcal{U}})$. $\mathsf{DSum}$ also plays a role in preventing combination of partial secret keys for which the agreed vectors are inconsistent.

*Our Function-Hiding* $\mathsf{IP\text{-}DDFE}$. Our approach is to lift function-hiding $\mathsf{IP\text{-}MCFE}$ to function-hiding $\mathsf{IP\text{-}DDFE}$ following their blueprint. Unfortunately, there are no function-hiding $\mathsf{IP\text{-}MCFE}$ schemes, and we need to start with constructing this. Our first idea is to leverage the recent conversion by Abdalla et al. [1] from $\mathsf{IPFE}$ to $\mathsf{IP\text{-}MCFE}$. However, this idea does not work since all parties share the same encryption key of an $\mathsf{IPFE}$ scheme in their converted schemes, and once a single party is corrupted, the adversary can learn the entire function (or vector) in secret keys. Thus, we could not achieve a function-hiding $\mathsf{IP\text{-}MCFE}$ scheme even if we apply the conversion to a function-hiding $\mathsf{IPFE}$ scheme.

To address this challenge, we devise a new technique to convert function-hiding $\mathsf{IPFE}$ to function-hiding $\mathsf{IP\text{-}MCFE}$, which is inspired by the function-hiding multi-input $\mathsf{IPFE}$ scheme by Datta et al. [23]. In their scheme, each party $i$ has a master secret key $\mathsf{iMSK}_i$ of a function-hiding $\mathsf{IPFE}$ scheme, the ciphertext $\mathsf{miCT}_i$ of $\mathbf{x}_i$ is $\mathsf{iCT}_i[(\mathbf{x}_i, 1)]$, and the secret key $\mathsf{miSK}$ of $\{\mathbf{y}_i\}_{i\in[n]}$ is $\{\mathsf{iSK}_i[(\mathbf{y}_i, r_i)]\}_{i\in[n]}$ where $r_i$ are randomly chosen so that $\sum_{i\in[n]} r_i = 0$. $\mathsf{iCT}_i[\mathbf{x}]$ and $\mathsf{iSK}_i[\mathbf{y}]$ denotes the ciphertext of $\mathbf{x}$ and the secret key of $\mathbf{y}$ in the function-hiding $\mathsf{IPFE}$ scheme, respectively. To lift their $\mathsf{MIFE}$ to $\mathsf{MCFE}$, we need to add the label checking mechanism and security against corruption of parties. Fortunately, we can achieve the latter almost for free since each party uses independent master secret key and corruption of a party does not affect other parties' ciphertexts and secret keys. We can achieve the former by changing $\mathsf{miCT}_i$ to $\mathsf{iCT}_i[(\mathbf{x}_i, t_i)]$ where $t_i = H(\mathsf{lab})$ is a hash of a label. Then, a decryptor can learn $\sum(\langle \mathbf{x}_i, \mathbf{y}_i\rangle + t_i r_i)$, which reveals $\sum\langle \mathbf{x}_i, \mathbf{y}_i\rangle$ only when $t_1 = \cdots = t_n$. We can prove the masking term $t_i r_i$ hides $\langle \mathbf{x}_i, \mathbf{y}_i\rangle$ under the SXDH assumption in the random oracle model.

Our next step is to lift function-hiding $\mathsf{IP\text{-}MCFE}$ to function-hiding $\mathsf{IP\text{-}DDFE}$. To do so, we must address additional technical challenges as described next. In the original definition of $\mathsf{IP\text{-}DDFE}$, recall that each secret key is associated with $(\mathbf{y}_i, \mathcal{U}, \mathbf{y} = \{\mathbf{y}_{i'}\}_{i'\in\mathcal{U}})$ where the first element $\mathbf{y}_i$ is a vector for a linear

function while the third element $\mathbf{y}$ is an agreed vector that controls combination of partial secret keys. More precisely, a decryptor can combine partial secret keys to obtain a full secret key for $\mathbf{y}$ only when it has $\{\mathsf{SK}_i\}_{i\in\mathcal{U}}$ associated with $\mathbf{y}$. However, $\mathbf{y}$ cannot be hidden in the blueprint by Chotard et al. To tackle this, we observe that the role of the agreed vector is analogous to a label in the ciphertext, controlling combination of partial secret keys. Thus, we alternatively use an independent label $\mathsf{lab}_K$ to create a natural symmetry between inputs for encryption and key generation. Now, since the vector $\mathbf{y}_i$ for a linear function can be hidden by function-hiding IP-MCFE, we obtain function-hiding IP-DDFE.

Another deviation from their blueprint arises in the part that securely generates $\mathsf{mcSK}_{\mathcal{U}}$ from $\mathsf{mcSK}_{i,\mathcal{U}}$. In our IP-MCFE construction, $\mathsf{mcSK}_{i,\mathcal{U}} = \mathsf{iSK}_i[(\mathbf{y}_i, r_i)]$ and $\mathsf{mcSK}_{\mathcal{U}} = \{\mathsf{iSK}_i[(\mathbf{y}_i, r_i)]\}_{i\in\mathcal{U}}$. Thus, we do not need to sum up $\mathsf{mcSK}_{i,\mathcal{U}}$ to obtain $\mathsf{mcSK}_{\mathcal{U}}$, instead, each party has to somehow generate a secret share $r_i$ without interaction such that $\sum_{i\in[\mathcal{U}]} r_i = 0$ only when all $\mathsf{mcSK}_{i,\mathcal{U}}$ are generated on behalf of the same label. To handle this issue, we employ a technique by Chase and Chow [18] to generate such shares via pseudorandom function. Please see Section 4 for more details.

**Distributed Ciphertext-Policy ABE** The recent construction of Agrawal-Yamada [8] proposed a succinct ciphertext-policy ABE for log-depth circuits provably secure under LWE in the bilinear generic group model. In our work, we extend the setup and key generation in [8] among a polynomial number of authorities that are working completely *non-interactively* and asynchronously. We start by describing the syntax of a distributed CP-ABE scheme. In a fully distributed setting, the authorities run their local setup algorithms individually to generate a fresh master public-secret key pair $(\mathsf{PK}, \mathsf{MSK})$ per authority such that given a sequence of, say $N$, master public keys $\{\mathsf{PK}_i\}_{i\in[N]}$, an encryptor could encrypt a message $\mu$ for a predicate circuit $F$ of its choice. Such ciphertexts can be decrypted after obtaining a partial predicate key from all $N$ authorities for a consistent identifier $\mathsf{GID}$, and attribute vector $\mathbf{x}$ such that $F(\mathbf{x}) = 1$. Note that here the key generation algorithm is run locally (and independently) by each authority, which on input its master key $\mathsf{MSK}_i$ along with $\mathsf{GID}$ and attribute $\mathbf{x}$, computes a partial key $\mathsf{SK}_{i,\mathsf{GID},\mathbf{x}}$. While correctness is natural, security must be defined carefully.

In this work, we consider the strongest form of corruption, where we allow the adversary to pick the key parameters for all corrupt authorities, and also allow it to query honest authorities on identifier-attribute pairs $(\mathsf{GID}, \mathbf{x})$ such that $F^*(\mathbf{x}) = 1$ (where $F^*$ is the challenge predicate circuit) as long as there is at least one honest authority to which the adversary did not query the pair $(\mathsf{GID}, \mathbf{x})$. All other queries are unconstrained since if $F^*(\mathbf{x}) = 0$, then such keys should not be useful for decryption to begin with. The intuition behind allowing the queries to honest authorities such that $F^*(\mathbf{x}) = 1$ is that we want to prevent partial secret keys for two distinct accepting attributes provided by two distinct authorities to be usable for decryption.

To describe our construction, we recall the high level structure of the Agrawal-Yamada scheme [8], which in turn uses the $\mathsf{BGG}^+$ ABE construction [13]. Roughly,

a $\mathsf{BGG}^+$ ciphertext is sampled in two steps — first, it samples a sequence of $2\ell$ encodings $\{\psi_{i,b}\}_{i,b}$; second, depending upon the attribute $\mathbf{x}$ the final ciphertext consists of $\ell$ encodings $\{\psi_{i,x_i}\}_i$. (Note that $\mathsf{BGG}^+$ is a key-policy scheme, whereas we are building a ciphertext-policy system.) The main idea behind the ciphertext-policy $\mathsf{ABE}$ of [8] is as follows:

$\mathsf{Setup}$ : Sample $2\ell$ random exponents $w_{i,b}$, store it as master secret key, and give its encoding $\{[w_{i,b}]_1\}_{i,b}$ as the public key.

$\mathsf{KeyGen}$ : To generate a secret key for attribute $\mathbf{x} \in \{0,1\}^\ell$, first sample a random exponent $\delta$ and then given out $[\delta/w_{i,x_i}]_2$ for $i \in [\ell]$ as the secret key.

$\mathsf{Enc}$ : To encrypt under predicate $F$, the encryptor samples all $2\ell$ $\mathsf{BGG}^+$ encodings $\{\psi_{i,b}\}_{i,b}$, and also samples a random exponent $\gamma$. It then gives out the ciphertext as a $\mathsf{BGG}^+$ secret key for predicate $C$ along with encodings $[\gamma w_{i,b}\psi_{i,b}]_1$ for $i \in [\ell], b \in \{0,1\}$.

$\mathsf{Dec}$ : A decryptor pairs the encodings $[\gamma w_{i,x_i}\psi_{i,x_i}]_1$ with $[\delta/w_{i,x_i}]_2$ to learn $[\gamma\delta\psi_{i,x_i}]_T$, and then it performs the $\mathsf{BGG}^+$ decryption in the exponent to learn the plaintext.

For the multi-authority extension, each authority samples its own sequence of $2\ell$ random exponents $w_{i,b}^{(j)}$ for $j \in [N]$. Then during encryption, the encryptor $N$-out-of-$N$ (additively) secret shares the $\mathsf{BGG}^+$ encodings $\{\psi_{i,b}\}_{i,b}$ into $\{\phi_{i,b}^{(j)}\}_{i,b}$ for $j \in [N]$. Now it encodes each sequence of $\{\phi_{i,b}^{(j)}\}_{i,b}$ terms under the corresponding authority's master public key as above. During decryption, a decryptor will first recover $\{\phi_{i,x_i}^{(j)}\}_i$ for all $j$ in the exponent, then add them to reconstruct the actual $\mathsf{BGG}^+$ ciphertext $\{\psi_{i,x_i}\}_i$ which it can decrypt as before. In order to let multiple independent authorities sample the same $\delta$, we rely on a hash function which we model as a ROM, and set $[\delta]_2 = H(\mathsf{GID})$.

Although our multi-authority transformation is natural, the proof does not follow trivially from [8]. This is primarily due to the fact that in the distributed setting, the adversary could potentially make key queries on accepting attributes as long as there is at least one honest party that does not receive the same query. Such queries did not exist in the single-authority setting. However, we can extend the single-authority proof to the multi-authority setting by a careful analysis of the additional "bad" zero-test queries that an adversary can make. Please see the full version for more details.

### 1.5 Predicting New and Useful Primitives via MPFE

One of the most exciting benefits of $\mathsf{MPFE}$ is that it provides the right framework to pose new, compelling questions that have not been studied before. For example, a very interesting question is what new kinds of dynamic key accumulation are possible, namely how to combine keys of different users chosen dynamically. So far, most existing literature on FE systems that enable aggregation of multiple decryption keys still consider very restricted scenarios: (i) each partial decryption

key corresponds to a portion of a much larger decryption key of a single user (e.g., distributed/decentralized/multi-authority FE etc), (ii) each partial decryption key corresponds to a function and many such keys may be combined if they each encode the *same* function (e.g. adhoc MIFE, D-MCFE).

However, the ability to combine keys in much more creative ways can enable several cool new applications. As an example, consider the following notion of "reputation point based encryption" – in this setting, each user key is associated with a subject tag $T$ (say math, history etc) and a reputation value $v$ (that is, a point score denoted as an integer). Now an encryptor specifies a tag $T'$ along with a threshold reputation value $w$, and hides its message $m$ under it. That is, $\mathsf{CT}(T', w, m)$ denotes such a ciphertext, and we require the functionality that such a ciphertext should be decryptable by any sequence of user keys $\mathsf{SK}(T_1, v_1), \ldots \mathsf{SK}(T_\ell, v_\ell)$ where all the subject tags match ($T' = T_1 \cdots = T_\ell$) and the combined reputation value of the group $\sum_{i \leq \ell} v_i$ is greater than threshold $w$. For example, an encryption of a message under subject 'math' and minimum reputation value of 1000 points can be decrypted by not only a single user with 1000 reputation points in 'math' but also by say a group of three users with 400, 250, 350 reputation points (respectively) in 'math', but not by a group of users who satisfy either the subject check or the reputation point check *but not both*. To the best of our knowledge, such an encryption framework has not been studied before, but our MPFE framework enables expressing and introducing such an encryption functionality.

## 2 Multi-Party Functional Encryption

In this section, we define our notion of multi-party functional encryption (MPFE). Let $n_x$ be the number of ciphertext inputs and $n_y$ be the number of key inputs. Let $\mathcal{X} = \mathcal{X}_{\text{pub}} \times \mathcal{X}_{\text{pri}}$ be the space of ciphertext inputs and $\mathcal{Y} = \mathcal{Y}_{\text{pub}} \times \mathcal{Y}_{\text{pri}}$ be the space of key inputs. We define two aggregation functions as $\mathsf{Agg}_x : \mathcal{X}^{n_x} \to \mathcal{X}^*$, and $\mathsf{Agg}_y : \mathcal{Y}^{n_y} \to \mathcal{Y}^*$.

An MPFE scheme is defined as a tuple of 4 algorithms/protocols $\mathsf{MPFE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt})$. To suitably capture existing primitives, we define our Setup algorithm/protocol to run in three modes, described next.

**Setup modes.** The Setup algorithm/protocol can be run in different modes: central, local, or interactive. For $\mathsf{mode} \in \{\mathsf{Central}, \mathsf{Local}, \mathsf{Interactive}\}$, consider the following.

Central : Here the Setup algorithm is run by one trusted third party which outputs the master secret keys and encryption keys for all users in the system.

Local : Here it is run independently by different parties without any interaction, and each party outputs its own encryption key and/or master secret key.

Interactive : Here it is an interactive protocol run by a set of users, at the end of which, each user has its encryption key and/or master secret key. We note that these keys may be correlated across multiple users.

A multi-party functional encryption (MPFE) consists of the following:

$\mathsf{Setup}\left(1^\lambda, n_x, n_y, \mathsf{Agg}_x, \mathsf{Agg}_y\right)$: This algorithm/protocol can be executed in any one of the three modes described above.[7] Given input the security parameter, number of ciphertext inputs $n_x$, number of key inputs $n_y$ and two aggregation functions $\mathsf{Agg}_x$, $\mathsf{Agg}_y$ as defined above, this algorithm outputs a set of encryption keys $\{\mathsf{EK}_i\}_{i\le n_x}$, master secret keys $\{\mathsf{MSK}_i\}_{i\le n_y}$ and public key $\mathsf{PK}$.

$\mathsf{KeyGen}\left(\mathsf{PK}, \mathsf{MSK}, j, y = (y_{\text{pub}}, y_{\text{pri}})\right)$: Given input the public key $\mathsf{PK}$, a master secret key $\mathsf{MSK}$, user index $j \in [n_y]$ and a function input $y = (y_{\text{pub}}, y_{\text{pri}})$, this algorithm outputs a secret key $\mathsf{SK}_y$.

$\mathsf{Encrypt}\left(\mathsf{PK}, \mathsf{EK}, i, x = (x_{\text{pub}}, x_{\text{pri}})\right)$: Given input the public key $\mathsf{PK}$, an encryption key $\mathsf{EK}$, user index $i \in [n_x]$, an input $x = (x_{\text{pub}}, x_{\text{pri}})$, this algorithm outputs a ciphertext $\mathsf{CT}_x$.

$\mathsf{Decrypt}\left(\mathsf{PK}, \{\mathsf{SK}_j\}_{j\le n_y}, \{\mathsf{CT}_i\}_{i\le n_x}\right)$: Given input the public key $\mathsf{PK}$, a set of secret keys $\{\mathsf{SK}_j\}_{j\le n_y}$ and a set of ciphertexts $\{\mathsf{CT}_i\}_{i\le n_x}$, this algorithm outputs a value $z$ or $\perp$.

We remark that in the *local* setup mode, it will be helpful to separate the setup algorithm into a global setup, denoted by $\mathsf{GSetup}$ along with a local setup, denoted by $\mathsf{LSetup}$, where the former is used only to generate common parameters of the system, such as group descriptions and such.

*Correctness.* We say that an $\mathsf{MPFE}$ scheme is *correct* if, $\forall (n_x, n_y) \in \mathbb{N}^2$, ciphertext inputs $x_i \in \mathcal{X}$ for $i \in [n_x]$, key inputs $y_j \in \mathcal{Y}$ for $j \in [n_y]$, message and function aggregation circuits $\mathsf{Agg}_x$ and $\mathsf{Agg}_y$, it holds that:

$$\Pr\left[\begin{array}{l} (\mathsf{PK}, \{\mathsf{EK}_i\}, \{\mathsf{MSK}_j\}) \leftarrow \mathsf{Setup}(1^\lambda, n_x, n_y, \mathsf{Agg}_x, \mathsf{Agg}_y) \\ \mathsf{CT}_i \leftarrow \mathsf{Encrypt}(\mathsf{PK}, \mathsf{EK}_i, i, x_i)\ \forall i \in [n_x] \\ z = z' : \mathsf{SK}_j \leftarrow \mathsf{KeyGen}(\mathsf{PK}, \mathsf{MSK}_j, j, y_j)\ \forall j \in [n_y] \\ z \leftarrow \mathsf{Decrypt}\left(\mathsf{PK}, \{\mathsf{SK}_j\}_{j\le n_y}, \{\mathsf{CT}_i\}_{i\le n_x}\right) \\ z' = \mathcal{U}\left(\mathsf{Agg}_x(\{x_i\}), \mathsf{Agg}_y(\{y_j\})\right) \end{array}\right] = 1.$$

Recall that $\mathcal{U}$ is the universal circuit with appropriate input and output size.

*Indistinguishability based security.* Next, we define security of $\mathsf{MPFE}$. The security definition is modelled in a similar fashion to MIFE security [24, §2.2] while taking into account corruption queries.

For any choice of parameters $\lambda, n_x, n_y$, aggregation functions $\mathsf{Agg}_x, \mathsf{Agg}_y$, and master keys $\mathsf{K} = (\mathsf{PK}, \{\mathsf{EK}_i\}_{i\in[n_x]}, \{\mathsf{MSK}_j\}_{j\in[n_y]}) \leftarrow \mathsf{Setup}(1^\lambda, n_x, n_y, \mathsf{Agg}_x, \mathsf{Agg}_y)$, we define the following list of oracles:

$\mathsf{Corrupt}^{\mathsf{K}}(\cdot)$, upon a call to this oracle for any $i \in [n_x]$ or $j \in [n_y]$, the adversary gets the corresponding encryption key $\mathsf{EK}_i$ or master secret key $\mathsf{MSK}_j$. In the case of a local setup, the adversary could instead also supply the oracle with adversarially generated keys for the corresponding user; whereas in case of an interactive setup, the adversary could simulate the behavior of the queried user

[7] We omit specifying the mode in the syntax for notational brevity.

index in the setup protocol. (Let $\mathcal{S}_x \subseteq [n_x]$ and $\mathcal{S}_y \subseteq [n_y]$ denote the set of user indices for which the corresponding encryption and master keys have been corrupted.)[8]

$\mathsf{Key}^{\mathsf{K},\beta}(\cdot,\cdot)$, upon a call to this oracle for an honest user index $j \in [n_y]$, function inputs $(y_j^{k,0}, y_j^{k,1})$ (where $y_j^{k,b} = \left(y_{j,\mathrm{pub}}^{k,b}, y_{j,\mathrm{pri}}^{k,b}\right)$ for $b \in \{0,1\}$), the challenger first checks whether the user $j$ was already corrupted or not. That is, if $j \in \mathcal{S}_y$, then it sends nothing, otherwise it samples a decryption key for function input $y_j^{k,\beta}$ using key $\mathsf{MSK}_j$ and sends it to the adversary. (Here $\beta$ is the challenge bit chosen at the start of the experiment.)

$\mathsf{Enc}^{\mathsf{K},\beta}(\cdot,\cdot)$, upon a call to this oracle for an honest user index $i \in [n_x]$, message inputs $(x_i^{\ell,0}, x_i^{\ell,1})$ (where $x_i^{\ell,b} = \left(x_{i,\mathrm{pub}}^{\ell,b}, x_{i,\mathrm{pri}}^{\ell,b}\right)$ for $b \in \{0,1\}$), the challenger first checks whether the user $i$ was already corrupted or not. That is, if $i \in \mathcal{S}_x$, then it sends nothing, otherwise it samples a ciphertext for input $x_i^{\ell,\beta}$ using key $\mathsf{EK}_i$ and sends it to the adversary.

We let $Q_x$ and $Q_y$ be the number of encryption and key generation queries (respectively) that had non-empty responses. Let $\mathcal{Q}_x = \{(i, (x^{\ell,0}, x^{\ell,1}))\}_{\ell \in [Q_x]}$ be the set of ciphertext queries and $\mathcal{Q}_y = \{(j, (y_j^{k,0}, y_j^{k,1}))\}_{k \in [Q_y]}$ be the set of key queries.

We say that an adversary $\mathcal{A}$ is *admissible* if:

1. For each of the encryption and key challenges, the public components of the two challenges are equal, namely $x_{\mathrm{pub}}^{\ell,0} = x_{\mathrm{pub}}^{\ell,1}$ for all $\ell \in [Q_x]$, and $y_{\mathrm{pub}}^{k,0} = y_{\mathrm{pub}}^{k,1}$ for all $k \in [Q_y]$.
2. For each of the encryption and key challenges, the *private* components of the two challenges are also equal, namely $x_{\mathrm{pri}}^{\ell,0} = x_{\mathrm{pri}}^{\ell,1}$ for all $\ell \in [Q_x]$ whenever $(i, (x^{\ell,0}, x^{\ell,1})) \in \mathcal{Q}_x$ and $i \in \mathcal{S}_x$, and $y_{\mathrm{pri}}^{k,0} = y_{\mathrm{pri}}^{k,1}$ for all $k \in [Q_y]$ whenever $(j, (y^{\ell,0}, y^{\ell,1})) \in \mathcal{Q}_y$ and $j \in \mathcal{S}_y$. That is, the private components must be the same as well if the user index $i$ or $j$, that the query was made for, was corrupted during the execution.
3. There do not exist two sequences $(\overrightarrow{x}^0, \overrightarrow{y}^0)$ and $(\overrightarrow{x}^1, \overrightarrow{y}^1)$ such that:

$$\mathcal{U}\left(\mathsf{Agg}_x(\{x_i^0\}), \mathsf{Agg}_y(\{y_j^0\})\right) \neq \mathcal{U}\left(\mathsf{Agg}_x(\{x_i^1\}), \mathsf{Agg}_y(\{y_j^1\})\right)$$

and i) for every $i \in [n_x]$, either $x_i^b$ was queried or $\mathsf{EK}_i$ was corrupted, and ii) for every $j \in [n_y]$, either $y_j^b$ was queried or $\mathsf{MSK}_j$ was corrupted, and

[8] Note that in case $\mathsf{EK}_i$ is completely contained in some $\mathsf{MSK}_j$ then make a master secret corruption query for $j$ will also add the corresponding index $i$ to $\mathcal{S}_x$, and vice versa. At a very high level, although having separate aggregation functions for partial secret key and ciphertexts as part of the framework allows us to capture a highly expressive class of encryption scheme; defining the most general notion of security for MPFE that captures all different types of setup and key distribution settings could be very dense. To that end, here we provide a clean security game which captures the existing encryption primitives. Capturing security for each setup mode and corruption model individually would be more precise in certain settings.

iii) at least one of inputs $\left\{x_i^b\right\}, \left\{y_j^b\right\}$ were queried and indices $i, j$ were not corrupted. (Note that if $i \in [n_x]$ or $j \in [n_y]$ were queried to the $\mathsf{Corrupt}$ oracle, the adversary can generate partial keys or ciphertexts for any value of its choice.)

An $\mathsf{MPFE}$ scheme $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt})$ is said to be $\mathsf{IND}$ secure if for any *admissible* $\mathsf{PPT}$ adversary $\mathcal{A}$, all length parameters $n_x, n_y \in \mathbb{N}$, and aggregation functions $\mathsf{Agg}_x, \mathsf{Agg}_y$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, the following holds

$$\Pr\left[\mathcal{A}^{\mathsf{Corrupt}^{\mathsf{K}}(\cdot),\mathsf{Key}^{\mathsf{K},\beta}(\cdot),\mathsf{Enc}^{\mathsf{K},\beta}(\cdot)}(1^\lambda,\mathsf{PK})=\beta : \begin{array}{c}\mathsf{K}\leftarrow\mathsf{Setup}(1^\lambda,n_x,n_y,\mathsf{Agg}_x,\mathsf{Agg}_y),\\ \mathsf{K}=(\mathsf{PK},\{\mathsf{EK}_i\}_i,\{\mathsf{MSK}_j\}_j),\\ \beta\leftarrow\{0,1\}\end{array}\right]\leq\frac{1}{2}+\mathsf{negl}(\lambda).$$

*Remark 2.1 (Weaker notions of security).* We say the scheme is selective $\mathsf{IND}$ secure if the adversary outputs the challenge message and function pairs at the very beginning of the game, before it makes any queries or receives the $\mathsf{PK}$. One may also consider the semi-honest setting, where the $\mathsf{Corrupt}$ oracle is not provided, or the case of static corruptions where the adversary provides all its corruptions once and for all at the start of the game.

Due to space constraints, we provide our definition of dynamic $\mathsf{MPFE}$ in the full version, and also provide a general feasibility of $\mathsf{MPFE}$ for circuits from the minimal assumption of $\mathsf{MIFE}$ for circuits.

## 3 Multi-Authority $\mathsf{ABE} \circ \mathsf{IPFE}$ for LSSS Access Structures

In this section, extend the construction of Abdalla et al. [4] ($\mathsf{ACGU20}$) to the multiparty setting. As discussed in Section 1, we support $n_y = n$ for some fixed, polynomial $n$ and $\mathsf{Local}$ mode of setup algorithm, so that each key generator generates its key components locally and independently. The number of encryptors $n_x = 1$ and public, private input $(\phi, \mathbf{v})$. Each of the $n$ key generators has public inputs $(\mathsf{GID}_i, x_i, \mathbf{u}_i)$ where $x_i \in \{0,1\}$ and $\mathbf{u}_i \in \mathbb{Z}_q^\ell$ for $i \in [n]$. The ciphertext aggregation function remains trivial but the key aggregation function checks if $\mathsf{GID}_1 = \mathsf{GID}_2 = \ldots = \mathsf{GID}_n$, $\mathbf{u}_1 = \mathbf{u}_2 = \ldots = \mathbf{u}_n$, and outputs $(f_{\mathbf{x}}, \mathbf{u})$ if so, where $\mathbf{x} = (x_1, \ldots, x_n)$ and $f_{\mathbf{x}}$ is a function that takes as input three arguments $(\phi, \mathbf{v}, \mathbf{u})$ and outputs $\langle \mathbf{u}, \mathbf{v} \rangle$ if $\phi(\mathbf{x}) = 1$.

In other words, we build a multi-authority attribute-based inner product functional encryption ($\mathsf{MA\text{-}AB\text{-}IPFE}$) scheme for linear secret sharing schemes (LSSS) access structures. We rely on simple asumptions over bilinear maps.

### 3.1 Specializing the MPFE Syntax

Since our framework of MPFE described in Section 2 is general enough to capture a large family of functionalities, using the general syntax as-is would result in a cumbersome definition in which multiple parameters are non-functional. Hence, we specialize the general framework to the specific functionality of interest here for ease of exposition.

*Syntax.* A MA-AB-IPFE scheme for predicate class $\mathcal{C} = \{\mathcal{C}_n : \mathcal{X}_n \to \{0,1\}\}_{n\in\mathbb{N}}$ and inner product message space $\mathcal{U} = \{\mathcal{U}_\ell\}_{\ell\in\mathbb{N}}$ consists of the following PPT algorithms:

$\mathsf{GSetup}(1^\lambda) \to \mathsf{PP}$. On input the security parameter $\lambda$, the setup algorithm outputs public parameters $\mathsf{PP}$.

$\mathsf{LSetup}(\mathsf{PP}, 1^n, 1^\ell, i) \to (\mathsf{PK}, \mathsf{MSK})$. On input the public parameters $\mathsf{PP}$, attribute length $n$, message space index $\ell$, and authority's index $i \in [n]$, the authority setup algorithm outputs a pair of master public-secret key $(\mathsf{PK}, \mathsf{MSK})$ for the $i$-th authority.

$\mathsf{KeyGen}(\mathsf{MSK}_j, \mathsf{GID}, b, \mathbf{u}) \to \mathsf{SK}_{j,\mathsf{GID},b,\mathbf{u}}$. The key generation algorithm takes as input the authority master secret key $\mathsf{MSK}_j$, global identifier $\mathsf{GID}$, an attribute bit $b \in \{0,1\}$, and key vector $\mathbf{u} \in \mathcal{U}_\ell$. It outputs a partial secret key $\mathsf{SK}_{j,\mathsf{GID},b,\mathbf{u}}$.

$\mathsf{Enc}(\{\mathsf{PK}_i\}_{i\in[n]}, C, \mathbf{v}) \to \mathsf{CT}$. The encryption algorithm takes as input the list of public keys $\{\mathsf{PK}_i\}_i$, predicate circuit $C$, and a message vector $\mathbf{v} \in \mathcal{U}_\ell$, and outputs a ciphertext $\mathsf{CT}$.

$\mathsf{Dec}(\{\mathsf{SK}_{i,\mathsf{GID},\mathbf{x},\mathbf{u}}\}_{i\in[n]}, \mathsf{CT}) \to m/\perp$. On input a list of $n$ partial secret keys $\{\mathsf{SK}_{i,\mathsf{GID},\mathbf{x},\mathbf{u}}\}_i$ and a ciphertext $\mathsf{CT}$, the decryption algorithm either outputs a message $m$ (corresponding to the inner product value) or a special string $\perp$ (to denote decryption failure).

*Correctness.* A MA-AB-IPFE scheme is said to be correct if for all $\lambda, n, \ell \in \mathbb{N}$, $C \in \mathcal{C}_n$, $\mathbf{u}, \mathbf{v} \in \mathcal{U}_\ell$, $\mathbf{x} \in \mathcal{X}_n, \mathsf{GID}$, if $C(\mathbf{x}) = 1$, the following holds:

$$\Pr\left[\mathsf{Dec}(\mathsf{SK}, \mathsf{CT}) = \langle \mathbf{u}, \mathbf{v}\rangle : \begin{array}{l} \mathsf{PP} \leftarrow \mathsf{GSetup}(1^\lambda) \\ \forall i \in [n] : (\mathsf{PK}_i, \mathsf{MSK}_i) \leftarrow \mathsf{LSetup}(\mathsf{PP}, 1^n, 1^\ell, i) \\ \forall j \in [n] : \mathsf{SK}_{j,\mathsf{GID},x_j,\mathbf{u}} \leftarrow \mathsf{KeyGen}(\{\mathsf{PK}_i\}_i, \mathsf{MSK}_j, \mathsf{GID}, x_j, \mathbf{u}) \\ \mathsf{CT} \leftarrow \mathsf{Enc}(\{\mathsf{PK}_i\}_i, C, \mathbf{v}), \mathsf{SK} = \{\mathsf{SK}_{i,\mathsf{GID},x_i,\mathbf{u}}\}_i \end{array}\right] = 1.$$

*Security.* In terms of security, a MA-AB-IPFE provides powerful notion of encrypted message vector indistinguishability where the adversary is allowed to corrupt the key generation authorities and also make key queries for message vector distinguishing key vectors (as long as the attribute does not satisfy the encrypted predicate). Below we provide the selective security variant of the corresponding property.[9]

**Definition 3.1 (Selective MA-AB-IPFE security with static corruptions).** *A MA-AB-IPFE scheme is selectively secure with static corruptions if for every stateful admissible PPT adversary $\mathcal{A}$, there exists a negligible function* $\mathsf{negl}(\cdot)$

[9] In this work, we only focus on standard semantic security, but one could also amplify to its CCA counterpart by relying on the generic CPA-to-CCA amplification techniques [28].

*such that for all* $\lambda \in \mathbb{N}$*, the following holds*

$$\Pr\left[\mathcal{A}^{O(\mathsf{key},\cdot,\cdot,\cdot)}(\{\mathsf{PK}_i\}_{i\in[n]\setminus S^*}, \mathsf{CT}) = b : \begin{array}{l} \mathsf{PP} \leftarrow \mathsf{GSetup}(1^\lambda) \\ (1^n, 1^\ell, S^*, C, (\mathbf{v}_0, \mathbf{v}_1), \{\mathsf{PK}_i\}_{i\in S^*}) \leftarrow \mathcal{A}(1^\lambda, \mathsf{PP}) \\ \forall i \in [n] \setminus S^* : (\mathsf{PK}_i, \mathsf{MSK}_i) \leftarrow \mathsf{LSetup}(\mathsf{PP}, 1^n, 1^\ell, i) \\ b \leftarrow \{0,1\}, \mathsf{CT} \leftarrow \mathsf{Enc}(\{\mathsf{PK}_i\}_{i\in[n]}, C, \mathbf{v}_b) \\ \mathsf{key} = \{(\mathsf{PK}_i, \mathsf{MSK}_i)\}_{i\in[n]\setminus S^*} \end{array}\right]$$
$$\leq \frac{1}{2} + \mathsf{negl}(\lambda),$$

*where the oracle* $O(\mathsf{key}, \cdot, \cdot, \cdot)$ *has the master key for honest authorities hardwired. The oracle on input a tuple of a global identifier* $\mathsf{GID}$*, an authority index* $j \in [n] \setminus S^*$*, and an attribute-key vector pair* $(b, \mathbf{u})$*, responds with a partial secret key computed as* $\mathsf{SK}_{j,\mathsf{GID},b,\mathbf{u}} \leftarrow \mathsf{KeyGen}(\mathsf{MSK}_j, \mathsf{GID}, b, \mathbf{u})$*. Note that the adversary is only allowed to submit key queries for non-corrupt authorities (i.e.,* $j \notin S^*$*). Also, the adversary* $\mathcal{A}$ *is admissible as long as every secret key query made by* $\mathcal{A}$ *to the key generation oracle* $O$ *satisfies the condition that — (1) either* $\langle \mathbf{u}, \mathbf{v}_0 \rangle = \langle \mathbf{u}, \mathbf{v}_1 \rangle$*, or (2)* $C$ *does not accept any input* $\mathbf{x}$ *such that* $x_j = b$ *for* $(b, j) \in Q_{\mathsf{GID}}$ *where* $Q_{\mathsf{GID}}$ *contains the attribute bits queries for* $\mathsf{GID}$[10]*.*

### 3.2 Construction

Let $\mathsf{Gen}$ be a composite-order bilinear group generator. Also, let $\mathbb{G}$ and $\mathbb{G}_T$ be the source and target groups, respectively. Additionally, we rely on a hash function $H : \{0,1\}^* \to \mathbb{G}$ that maps global identities $\mathsf{GID}$ to elements of $\mathbb{G}$ and we later model it as a random oracle in the proof. Below we provide our MA-AB-IPFE scheme based on composite-order bilinear maps for the predicates described as an access policy for a linear secret sharing scheme.

$\mathsf{GSetup}(1^\lambda) \to \mathsf{PP}$. The setup algorithm samples a bilinear group as follows

$$(p_1, p_2, p_3, \mathbb{G}, \mathbb{G}_T, e(\cdot,\cdot)) \leftarrow \mathsf{Gen}(1^\lambda, 3).$$

It samples a random generator $g_1 \in \mathbb{G}_1$, and sets the global public parameters as $\mathsf{PP} = (g_1, N = p_1p_2p_3, \mathbb{G}, \mathbb{G}_T, e(\cdot,\cdot))$.

(**Notation.** Here and throughout, we use the 'bracket' notation for representing group elements. Where $[1]_1 := g_1$, and $[1]_{T,1} := e(g_1, g_1)$.)

$\mathsf{LSetup}(\mathsf{PP}, 1^n, 1^\ell, i) \to (\mathsf{PK}, \mathsf{MSK})$. The algorithm samples two random vectors $\boldsymbol{\alpha}, \mathbf{w} \leftarrow \mathbb{Z}_N^\ell$, and sets the authority public-secret key pair as $\mathsf{PK} = (\mathsf{PP}, [\boldsymbol{\alpha}]_{T,1}, [\mathbf{w}]_1)$ and $\mathsf{MSK} = (\boldsymbol{\alpha}, \mathbf{w})$. (Here and throughout, note that $[\mathbf{w}]_1$ and similar terms can be computed as $g_1^{\mathbf{w}}$.)

$\mathsf{KeyGen}(\mathsf{MSK}_j, \mathsf{GID}, b, \mathbf{u}) \to \mathsf{SK}_{j,\mathsf{GID},b,\mathbf{u}}$. It parses the authority key as described above. If $b = 0$, it sets the secret key as empty string. Otherwise, it first

[10] Note that in general this could be a non-falsifiable condition to check if $S^*$ is $\omega(\log \lambda)$ and the predicate class contains general non-monotonic functions.

hashes the $\mathsf{GID}$ to create a masking term $\left[\mu\right] \in \mathbb{G}$ as $\left[\mu\right] = H(\mathsf{GID})$. It then outputs the secret key as

$$\mathsf{SK}_{j,\mathsf{GID},b,\mathbf{u}} = \left[\langle \boldsymbol{\alpha}, \mathbf{u} \rangle\right]_1 \cdot \left[\mu \cdot \langle \mathbf{w}, \mathbf{u} \rangle\right].$$

Note that since the vectors $\mathbf{u}, \mathbf{w}, \boldsymbol{\alpha}$ are known to the algorithm in the clear, thus the above key term can be computed efficiently.

$\mathsf{Enc}(\{\mathsf{PK}_i\}_{i\in[n]}, (\mathbf{A}, \rho), \mathbf{v}) \to \mathsf{CT}$. The encryption algorithm first parses the keys $\mathsf{PK}_i$ as $(\mathsf{PP}, \left[\boldsymbol{\alpha}_i\right]_{T,1}, \left[\mathbf{w}_i\right]_1)$, and the predicate contains an $m_1 \times m_2$ access matrix $\mathbf{A}$ with function $\rho$ mapping the rows to the attribute positions. It samples a $m_2 \times \ell$ matrix $\mathbf{S}$ and $(m_2 - 1) \times \ell$ matrix $\mathbf{T}'$ uniformly at random as $\mathbf{S} \leftarrow \mathbb{Z}_N^{m_2 \times \ell}$ and $\mathbf{T}' \leftarrow \mathbb{Z}_N^{(m_2-1)\times \ell}$. It sets a $m_2 \times \ell$ matrix $\mathbf{T}$, and arranges two $m_1 \times \ell$ matrices $\boldsymbol{\Delta}$ and $\boldsymbol{\Gamma}$ as

$$\mathbf{T} = \begin{pmatrix} \mathbf{0}^\top \\ \mathbf{T}' \end{pmatrix}, \qquad \boldsymbol{\Delta} = \begin{pmatrix} \boldsymbol{\alpha}_{\rho(1)}^\top \\ \vdots \\ \boldsymbol{\alpha}_{\rho(m_1)}^\top \end{pmatrix}, \qquad \boldsymbol{\Gamma} = \begin{pmatrix} \mathbf{w}_{\rho(1)}^\top \\ \vdots \\ \mathbf{w}_{\rho(m_1)}^\top \end{pmatrix}.$$

That is, the matrix $\mathbf{T}$ contains all zeros in the first row and is random otherwise. It also samples a random vector as $\mathbf{r} \leftarrow \mathbb{Z}_N^{m_1}$, and computes the ciphertext $\mathsf{CT} = (C_0, C_1, C_2, C_3)$ as:

$$C_0 = \left[\mathbf{s}_1 + \mathbf{v}\right]_{T,1}, \qquad C_1 = \left[\mathbf{A} \cdot \mathbf{S} + \boldsymbol{\Delta} \odot (\mathbf{r} \otimes \mathbf{1}^\top)\right]_{T,1},$$
$$C_2 = \left[\mathbf{r}\right]_1, \qquad C_3 = \left[\mathbf{A} \cdot \mathbf{T} + \boldsymbol{\Gamma} \odot (\mathbf{r} \otimes \mathbf{1}^\top)\right]_1.$$

Here the vector $\mathbf{s}_1$ is the first column vector of matrix $\mathbf{S}^\top$ (that is, $\mathbf{s}_1 = \mathbf{S}^\top \cdot \mathbf{e}_1$ where $\mathbf{e}_1$ is the first fundamental basis vector of $\mathbb{Z}_N^{m_2}$).

$\mathsf{Dec}(\{\mathsf{SK}_{i,\mathsf{GID},x_i,\mathbf{u}}\}_{i\in[n]}, \mathsf{CT}) \to M$. It parses the secret key and ciphertext as described above. Let $(\mathbf{A}, \rho)$ be the access policy associated with the ciphertext, and $\mathbf{u}$ be the key vector associated with the partial secret keys. (This could either be explicitly addded to the ciphertext and secret keys above, or passed as an auxiliary input.)

The decryptor first computes the LSSS reconstruction vector $\mathbf{z}$ such that $\mathbf{z}^\top \cdot \mathbf{A} = \mathbf{e}_1^\top = (1, 0, \ldots, 0)$. The decryptor then arranges the key terms as

$$K = \begin{pmatrix} \mathsf{SK}_{\rho(1),\mathsf{GID},x_{\rho(1)},\mathbf{u}} \\ \vdots \\ \mathsf{SK}_{\rho(m_1),\mathsf{GID},x_{\rho(m_1)},\mathbf{u}} \end{pmatrix}$$

and recovers the inner product message value $M$ by computing the discrete log of the following the following:

$$\left[M\right]_{T,1} = \frac{\langle C_0, \mathbf{u} \rangle}{(\mathbf{z}^\top \cdot C_1 \cdot \mathbf{u})} \cdot \frac{\mathbf{z}^\top \cdot e\left(K, C_2\right)}{e\left(H(\mathsf{GID}), \mathbf{z}^\top \cdot C_3 \cdot \mathbf{u}\right)}$$

where the matrix vector operations involving group elements and exponents are performed by first raising the exponent of each term (component-by-component) for performing multiplication in the exponent, and then followed

by multiplication of the resulting encodings to simulate addition being performed in the exponent. Also, the operation $e(K, C_2)$ performs the pairing operation element-by-element for each element of the vector.

### 3.3 Correctness and Security

Due to space constraints, the proof is provided in the full version [6].

## 4 Function-Hiding DDFE for Inner Products

In this section, we present our function-hiding decentralized dynamic inner product functional encryption (IP-DDFE) scheme. As described in Section 1, we have the setup algorithm in the Local mode, so that each party $i$ can dynamically join the system by generating a public key $\mathsf{PK}_i$ and a master secret key $\mathsf{MSK}_i$. For encryption, party $i$ sets $(x_{\text{pub}}, x_{\text{pri}}) = \big((\mathcal{U}_M, \mathsf{lab}_M), \mathbf{x}_i\big)$ where $\mathcal{U}_M$ is the set of parties whose inputs will be combined and $\mathsf{lab}_M$ is a label which imposes a constraint on which values can be aggregated together. For key generation, party $i$ sets $(y_{\text{pub}}, y_{\text{pri}}) = \big((\mathcal{U}_K, \mathsf{lab}_K), \mathbf{y}_i\big)$ where $\mathcal{U}_K, \mathsf{lab}_K$ have the same roles as $\mathcal{U}_M, \mathsf{lab}_M$, respectively. The function $\mathsf{Agg}_x$ checks if the public inputs $(\mathcal{U}_M, \mathsf{lab}_M)$ match for all parties and that all the ciphertexts are provided for the set $\mathcal{U}_M$. If so, outputs $(\mathcal{U}_M, \mathbf{x})$ where $\mathbf{x} = (\mathbf{x}_1 \| \dots \| \mathbf{x}_{n_x})$. The function $\mathsf{Agg}_y$ checks that all values $\mathcal{U}_K$ and $\mathsf{lab}_K$ are the same for all parties. If so, it outputs the function $f_{\mathcal{U}_K, \mathbf{y}=(\mathbf{y}_1 \| \dots \| \mathbf{y}_{n_y})}$ which takes as input $(\mathcal{U}_M, \mathbf{x})$, checks that $\mathcal{U}_M = \mathcal{U}_K$ and if so, outputs $\langle \mathbf{x}, \mathbf{y} \rangle$.

As discussed in the introduction, we first obtain a function-hiding multi-client inner product functional encryption (IP-MCFE) scheme, and then lift it to a function-hiding IP-DDFE scheme in a non-black box manner. We first define necessary notions to describe our IP-MCFE and IP-DDFE scheme. As before, we will specialize the MPFE syntax for ease of exposition.

### 4.1 Specializing the MPFE Syntax

*Syntax of* MCFE. Let $\mathcal{F}$ be a function family such that, for all $f \in \mathcal{F}$, $f : \mathcal{M}_1 \times \cdots \times \mathcal{M}_n \to \mathcal{Z}$. Let $\mathcal{L}$ be a label space. An MCFE scheme for $\mathcal{F}$ and $\mathcal{L}$ consists of four algorithms.

$\mathsf{Setup}(1^\lambda, 1^n)$**:** It takes a security parameter $1^\lambda$ and a number $1^n$ of slots, and outputs a public parameter $\mathsf{PK}$, encryption keys $\{\mathsf{EK}_i\}_{i \in [n]}$, a master secret key $\mathsf{MSK}$. The other algorithms implicitly take $\mathsf{PK}$.

$\mathsf{KeyGen}(\mathsf{MSK}, f)$**:** It takes $\mathsf{MSK}$ and $f \in \mathcal{F}$, and outputs a secret key $\mathsf{SK}$.

$\mathsf{Enc}(i, \mathsf{EK}_i, x_i, \mathsf{lab})$**:** It takes $\mathsf{MSK}$, an index $i \in [n]$, $x_i \in \mathcal{M}_i$, and a label $\mathsf{lab}$ and outputs a ciphertext $\mathsf{CT}_i$.

$\mathsf{Dec}(\mathsf{CT}_1, ..., \mathsf{CT}_n, \mathsf{SK})$**:** It takes $\mathsf{CT}_1, ..., \mathsf{CT}_n$ and $\mathsf{SK}$, and outputs a decryption value $d \in \mathcal{Z}$ or a symbol $\perp$.

*Correctness.* An $\mathsf{MCFE}$ scheme is correct if it satisfies the following condition. For all $\lambda, n \in \mathbb{N}$, $(x_1, ..., x_n) \in \mathcal{M}_1 \times \cdots \times \mathcal{M}_n$, $f \in \mathcal{F}$, $\mathsf{lab} \in \mathcal{L}$, we have

$$\Pr\left[d = f(x_1, ..., x_n) : \begin{array}{l}(\mathsf{PK}, \{\mathsf{EK}_i\}, \mathsf{MSK}) \leftarrow \mathsf{Setup}(1^\lambda, 1^n) \\ \mathsf{CT}_i \leftarrow \mathsf{Enc}(i, \mathsf{EK}_i, x_i, \mathsf{lab}) \\ \mathsf{SK} \leftarrow \mathsf{KeyGen}(\mathsf{MSK}, f) \\ d = \mathsf{Dec}(\mathsf{CT}_1, ..., , \mathsf{CT}_n, \mathsf{SK})\end{array}\right] = 1.$$

*Security.* We basically adopt the security definition for $\mathsf{MCFE}$ in [2] and extend it to function-hiding security. We also introduce a selective vatiant because our final goal is $\mathsf{IP\text{-}DDFE}$ with selective security, and selectively secure $\mathsf{IP\text{-}MCFE}$ is sufficient for the security analysis of our $\mathsf{IP\text{-}DDFE}$ scheme.

**Definition 4.1 (Function-hiding security of $\mathsf{MCFE}$).** An $\mathsf{MCFE}$ scheme is $\mathsf{Leak}_y$-xx-yy-function-hiding (xx $\in$ {sel, sta, adt}, yy $\in$ {any, pos}) if for every stateful PPT adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda, n \in \mathbb{N}$, the following holds

$$\Pr\left[\beta \leftarrow \mathcal{A}^{\mathsf{QCor}(),\mathsf{QEnc}^\beta(),\mathsf{QKeyGen}^\beta()}(\mathsf{PK}) : \begin{array}{l}\beta \leftarrow \{0,1\} \\ (\mathsf{PK}, \{\mathsf{EK}_i\}, \mathsf{MSK}) \leftarrow \mathsf{Setup}(1^\lambda, 1^n)\end{array}\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda)$$

where $\mathsf{QCor}(i)$ outputs $\mathsf{EK}_i$, $\mathsf{QEnc}^\beta(i, x_i^0, x_i^1, \mathsf{lab})$ outputs $\mathsf{Enc}(i, \mathsf{EK}_i, x_i^\beta, \mathsf{lab})$, and $\mathsf{QKeyGen}^\beta(f^0, f^1)$ outputs $\mathsf{KeyGen}(\mathsf{MSK}, f^\beta)$. Let $q_{c,i,\mathsf{lab}}$ be the numbers of queries of the forms of $\mathsf{QEnc}^\beta(i, *, *, \mathsf{lab})$. Let $\mathcal{HS}$ be the set of parties on which the adversary has not queried $\mathsf{QCor}$ at the end of the game, and $\mathcal{CS} = [n] \backslash \mathcal{HS}$. Then, the adversary's queries must satisfy the following conditions.

- For $i \in \mathcal{CS}$, the queries $\mathsf{QEnc}^\beta(i, x_i^0, x_i^1, \mathsf{lab})$ and $\mathsf{QKeyGen}^\beta(f^0, f^1)$ must satisfy $x_i^0 = x_i^1$ and $\mathsf{Leak}_y(i, f^0) = \mathsf{Leak}_y(i, f^1)$, respectively.[11]
- There are no sequences $(x_1^0, ..., x_n^0, f^0, \mathsf{lab})$ and $(x_1^1, ..., x_n^1, f^1, \mathsf{lab})$ that satisfy all the conditions:
  - For all $i \in [n]$, [$\mathsf{QEnc}^\beta(i, x_i^0, x_i^1, \mathsf{lab})$ is queried and $i \in \mathcal{HS}$] or [$x_i^0 = x_i^1 \in \mathcal{M}_i$ and $i \in \mathcal{CS}$].
  - $\mathsf{QKeyGen}^\beta(f^0, f^1)$ are queried.
  - $f^0(x_1^0, ..., x_n^0) \neq f^1(x_1^1, ..., x_n^1)$.
- When xx = sta: the adversary cannot query $\mathsf{QCor}$ after querying $\mathsf{QEnc}$ or $\mathsf{QKeyGen}$ even once.
- When xx = sel: the adversary must make all queries in one shot. That is, first it outputs $(\mathcal{CS}, \{i, x_i^0, x_i^1, \mathsf{lab}\}, \{f^0, f^1\})$ and obtains the response: $(\{\mathsf{EK}_i\}_{i \in \mathcal{CS}}, \{\mathsf{Enc}(i, \mathsf{EK}_i, x_i^\beta, \mathsf{lab})\}, \{\mathsf{KeyGen}(\mathsf{MSK}, f^\beta)\})$.
- When yy = pos: for each $\mathsf{lab} \in \mathcal{L}$, either $q_{c,i,\mathsf{lab}} > 0$ for all $i \in \mathcal{HS}$ or $q_{c,i,\mathsf{lab}} = 0$ for all $i \in \mathcal{HS}$.

[11] The leakage function captures information that $\mathsf{EK}_i$ reveals from $\mathsf{SK}$.

*Syntax of* DDFE. We define the syntax of DDFE. Note that we use an identifier $i \in \mathcal{ID}$ to specify each party while they use PK for identifier in the original definition [22], since it allows more precise indexing than the indexing by PK[12]. We assume that the correspondence between id $i$ and public key $\mathsf{PK}_i$ is publicly known, or it could be supplied as an input to the local setup algorithm. We describe the syntax of DDFE in the context of MPFE and change some expressions from the original definition. For instance, we use MSK instead of SK for secret keys of each party, public/private inputs for Enc and KeyGen instead of using empty keys, and so on.

Let $\mathcal{ID}, \mathcal{K}, \mathcal{M}$ be an ID space, a key space, and a message space, respectively. $\mathcal{K}, \mathcal{M}$ consist of a public part and a private part, that is, $\mathcal{K} = \mathcal{K}_{\mathrm{pri}} \times \mathcal{K}_{\mathrm{pub}}, \mathcal{M} = \mathcal{M}_{\mathrm{pri}} \times \mathcal{M}_{\mathrm{pub}}$. Let $f$ be a function such that $f : \bigcup_{i\in\mathbb{N}}(\mathcal{ID} \times \mathcal{K})^i \times \bigcup_{i\in\mathbb{N}}(\mathcal{ID} \times \mathcal{M})^i \to \mathcal{Z}$. A DDFE scheme for $f$ consists of five algorithms.

$\mathsf{GSetup}(1^\lambda)$**:** It takes a security parameter $1^\lambda$ and outputs a public parameter PP. The other algorithms implicitly take PP.

$\mathsf{LSetup}(\mathsf{PP})$**:** It takes PP and outputs local public parameter $\mathsf{PK}_i$ and a master secret key $\mathsf{MSK}_i$. The following three algorithms implicitly take $\mathsf{PK}_i$.

$\mathsf{KeyGen}(\mathsf{MSK}_i, k = (k_{\mathrm{pri}}, k_{\mathrm{pub}}))$**:** It takes $\mathsf{MSK}_i$ and $k \in \mathcal{K}$, and outputs a secret key $\mathsf{SK}_i$.

$\mathsf{Enc}(\mathsf{MSK}_i, m = (m_{\mathrm{pri}}, m_{\mathrm{pub}}))$**:** It takes $\mathsf{MSK}_i$ and $m \in \mathcal{M}$, and outputs a ciphertext $\mathsf{CT}_i$.

$\mathsf{Dec}(\{\mathsf{SK}_i\}_{i\in\mathcal{U}_K}, \{\mathsf{CT}_i\}_{i\in\mathcal{U}_M})$**:** It takes $\{\mathsf{SK}_i\}_{i\in\mathcal{U}_K}, \{\mathsf{CT}_i\}_{i\in\mathcal{U}_M}$ and outputs a decryption value $d \in \mathcal{Z}$ or a symbol $\perp$ where $\mathcal{U}_K \subseteq \mathcal{ID}$ and $\mathcal{U}_M \subseteq \mathcal{ID}$ are any sets.

*Correctness.* An DDFE scheme for $f$ is correct if it satisfies the following condition. For all $\lambda \in \mathbb{N}$, $\mathcal{U}_K \subseteq \mathcal{ID}$, $\mathcal{U}_M \subseteq \mathcal{ID}$, $\{i, k_i\}_{i\in\mathcal{U}_K} \in \bigcup_{i\in\mathbb{N}}(\mathcal{ID} \times \mathcal{K})^i$, $\{i, m_i\}_{i\in\mathcal{U}_M} \in \bigcup_{i\in\mathbb{N}}(\mathcal{ID} \times \mathcal{M})^i$, we have

$$\Pr\left[ d = f(\{i,k_i\}_{i\in\mathcal{U}_K}, \{i,m_i\}_{i\in\mathcal{U}_M}) : \begin{array}{l} \mathsf{PP} \leftarrow \mathsf{GSetup}(1^\lambda) \\ \mathsf{PK}_i, \mathsf{MSK}_i \leftarrow \mathsf{LSetup}(\mathsf{PP}) \\ \mathsf{CT}_i \leftarrow \mathsf{Enc}(\mathsf{MSK}_i, m_i) \\ \mathsf{SK}_i \leftarrow \mathsf{KeyGen}(\mathsf{MSK}_i, k_i) \\ d = \mathsf{Dec}(\{\mathsf{SK}_i\}_{i\in\mathcal{U}_K}, \{\mathsf{CT}_i\}_{i\in\mathcal{U}_M}) \end{array} \right] = 1.$$

Note that we can consider the case where $\mathcal{U}_K$ and $\mathcal{U}_M$ are multisets as in the original definition in [22]. However, we do not consider the case here since it induces ambiguity that can be also found in [22][13]. We assume that $\mathbb{N}$ contains 0 here and $(\mathcal{ID} \times \mathcal{K})^0 = \{i, k_i\}_{i\in\emptyset} = \emptyset$. That is, $\mathcal{U}_K$ and $\mathcal{U}_M$ can be an empty set, which corresponds to the case where Dec does not take secret keys/ciphertexts as input.

[12] In [22], some definitions have ambiguity that seems to stem from the indexing by pk. For instance, correctness of DDFE in Definition 1 implicitly assumes that $\mathsf{sk}_{\mathsf{pk}}$ is uniquely decided by pk, while the syntax does not require such a condition. Another example is the IP-DDFE construction in [22, § 7.2].

[13] Concretely, when $\mathcal{U}_K$ is a multiset, and $i' \in \mathcal{U}_K$ has multiplicity 2, how to treat $k_{i'} \in \{k_i\}_{i\in\mathcal{U}_K}$ is unclear.

*Security.* We naturally extend the security definition for DDFE in [22] to the function-hiding setting as follows.

**Definition 4.2 (Function-hiding security of DDFE).** An DDFE scheme is xx-yy-function-hiding (xx ∈ {sel, adt}, yy ∈ {sym, asym}) if for every stateful PPT adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, the following holds

$$\Pr\left[\beta \leftarrow \mathcal{A}^{\mathsf{QHonestGen}(),\mathsf{QCor}(),\mathsf{QEnc}^{\beta}(),\mathsf{QKeyGen}^{\beta}()}(\mathsf{PP}) : \begin{array}{l}\beta \leftarrow \{0,1\} \\ \mathsf{PP} \leftarrow \mathsf{GSetup}(1^{\lambda})\end{array}\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda).$$

Each oracle works as follows. For $i \in \mathcal{ID}$, $\mathsf{QHonestGen}(i)$ runs $(\mathsf{PK}_i, \mathsf{MSK}_i) \leftarrow \mathsf{LSetup}(\mathsf{PP})$ and returns $\mathsf{PK}_i$. For $i$ such that $\mathsf{QHonestGen}(i)$ was queried, the adversary can make the following queries: $\mathsf{QCor}(i)$ outputs $\mathsf{MSK}_i$, $\mathsf{QEnc}^{\beta}(i, m^0, m^1)$ outputs $\mathsf{Enc}(\mathsf{MSK}_i, m^{\beta})$, and $\mathsf{QKeyGen}^{\beta}(i, k^0, k^1)$ outputs $\mathsf{KeyGen}(\mathsf{MSK}_i, k^{\beta})$. Note that $k^{\beta}$ and $m^{\beta}$ consist of the private elements $k^{\beta}_{\mathrm{pri}}, m^{\beta}_{\mathrm{pri}}$ and the public elements $k_{\mathrm{pub}}, m_{\mathrm{pub}}$, respectively (we always require that $k^0_{\mathrm{pub}} = k^1_{\mathrm{pub}} = k_{\mathrm{pub}}$ and $m^0_{\mathrm{pub}} = m^1_{\mathrm{pub}} = m_{\mathrm{pub}}$ as the public elements are not hidden in SK or CT). Let $\mathcal{S}$ be the set of parties on which $\mathsf{QHonestGen}(i)$ is queried, $\mathcal{HS}$ be the set of parties on which the adversary has not queried QCor at the end of the game, and $\mathcal{CS} = \mathcal{S} \backslash \mathcal{CS}$. Then, the adversary's queries must satisfy the following conditions.

- There are no sequences $(\{i, k^0_i\}_{i \in \mathcal{U}_K}, \{i, m^0_i\}_{i \in \mathcal{U}_M})$ and $(\{i, k^1_i\}_{i \in \mathcal{U}_K}, \{i, m^1_i\}_{i \in \mathcal{U}_M})$ that satisfy all the conditions:
  - For all $i \in \mathcal{U}_K$, [$\mathsf{QKeyGen}^{\beta}(i, k^0_i, k^1_i)$ is queried and $i \in \mathcal{HS}$] or [$k^0_i = k^1_i \in \mathcal{K}$ and $i \in \mathcal{CS}$].
  - For all $i \in \mathcal{U}_M$, [$\mathsf{QEnc}^{\beta}(i, m^0_i, m^1_i)$ is queried and $i \in \mathcal{HS}$] or [$m^0_i = m^1_i \in \mathcal{M}$ and $i \in \mathcal{CS}$].
  - $f(\{i, k^0_i\}_{i \in \mathcal{U}_K}, \{i, m^0_i\}_{i \in \mathcal{U}_M}) \neq f(\{i, k^1_i\}_{i \in \mathcal{U}_K}, \{i, m^1_i\}_{i \in \mathcal{U}_M})$.
- When xx = sel: the adversary first generates a set $\mathcal{S}$ of honest users in one shot. After that it makes the corruption, key generation, encryption queries in one shot to obtain $\{\mathsf{MSK}_i\}, \{\mathsf{KeyGen}(\mathsf{MSK}_i, k^{\beta})\}, \{\mathsf{Enc}(\mathsf{EK}_i, m^{\beta})\}$.
- When yy = sym: for $i \in \mathcal{CS}$, the queries $\mathsf{QKeyGen}^{\beta}(i, k^0, k^1)$ and $\mathsf{QEnc}^{\beta}(i, m^0, m^1)$ must satisfy $k^0 = k^1$ and $m^0 = m^1$, respectively[14].

**Definition 4.3 (Inner Product Functional Encryption (IPFE)).** Let $\Pi = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2)$ be bilinear groups. IPFE for $\Pi$ is a class of FE where $\mathcal{M} = \mathbb{G}_1^{\mathsf{N}}$, and function $f \in \mathcal{F}$ is represented by $[\mathbf{y}]_2 \in \mathbb{G}_2^{\mathsf{N}}$ where $\mathbf{y} \in \mathbb{Z}_p^{\mathsf{N}}$ and defined as $f([\mathbf{x}]_1) = [\langle \mathbf{x}, \mathbf{y} \rangle]_T$. We say IPFE is function-hiding if it has both message and function privacy.

**Definition 4.4 (IP-MCFE).** Let $B \in \mathbb{N}$ be a bound of the infinity norm of vectors. IP-MCFE is a class of MCFE where $\mathcal{M}_i = [-B, B]^{\mathsf{N}}$, $\mathcal{Z} = \mathbb{Z}$, and $\mathcal{L} = \{0,1\}^*$. The function $f$ is represented by $\mathbf{y} \in [-B, B]^{n\mathsf{N}}$ and defined as $f(\mathbf{x}_1, ..., \mathbf{x}_n) = \langle (\mathbf{x}_1 || ... || \mathbf{x}_n), \mathbf{y} \rangle$.

[14] The symmetric setting captures the case where $\mathsf{MSK}_i$ can be used to not only encrypt/key generation but also decryption/decoding of $\mathsf{CT}_i/\mathsf{SK}_i$.

**Definition 4.5** (IP-DDFE). Let $B \in \mathbb{N}$ be a bound of the infinity norm of vectors. IP-DDFE is a class of DDFE where $\mathcal{ID} = \{0,1\}^*$, $\mathcal{K}_{\mathrm{pri}} = \mathcal{M}_{\mathrm{pri}} = [-B, B]^{\mathsf{N}}$, $\mathcal{K}_{\mathrm{pub}} = \mathcal{M}_{\mathrm{pub}} = 2^{\mathcal{ID}} \times \mathcal{L}$, $\mathcal{Z} = \mathbb{Z}$ for label space $\mathcal{L} = \{0,1\}^*$. The function $f$ is defined as, for $\{k_i = (\mathbf{y}_i, \mathcal{U}_{K,i}, \mathsf{lab}_{K,i})\}_{i \in \mathcal{U}'_K}$ and $\{m_i = (\mathbf{x}_i, \mathcal{U}_{M,i}, \mathsf{lab}_{M,i})\}_{i \in \mathcal{U}'_M}$,

$$f(\{i, k_i\}_{i \in \mathcal{U}'_K}, \{i, m_i\}_{i \in \mathcal{U}'_M}) = \begin{cases} \sum_{i \in \mathcal{U}'_K} \langle \mathbf{x}_i, \mathbf{y}_i \rangle & \text{the condition below is satisfied} \\ \bot & \text{otherwise} \end{cases}$$

- $\mathcal{U}'_K = \mathcal{U}'_M$, and $\forall i \in \mathcal{U}'_K, \mathcal{U}_{K,i} = \mathcal{U}_{M,i} = \mathcal{U}'_K$.
- $\exists (\mathsf{lab}_K, \mathsf{lab}_M) \in \mathcal{L}^2, \forall i \in \mathcal{U}'_K, \mathsf{lab}_{K,i} = \mathsf{lab}_K, \mathsf{lab}_{M,i} = \mathsf{lab}_M$.

**Definition 4.6 (One key-label restriction for IP-DDFE).** We define an additional restriction for the adversary in the security game for IP-DDFE. We say an IP-DDFE scheme is xx-yy-function-hiding under the one key-label restriction if it satisfies Definition 4.2 where the adversary's queries additionally satisfy the following condition: QKeyGen with respect to user $i \in \mathcal{ID}$ and label $\mathsf{lab}_K \in \mathcal{L}$ (the query of the form of $\mathsf{QKeyGen}(i, *, *, *, \mathsf{lab}_K)$) can be made only once for each pair $(i, \mathsf{lab}_K)$.

**Definition 4.7 (All-or-nothing encryption (AoNE)).** AoNE is a class of DDFE where $\mathcal{ID} = \{0,1\}^*$, $\mathcal{M}_{\mathrm{pri}} = \{0,1\}^L$ for some $L \in \mathbb{N}$, $\mathcal{M}_{\mathrm{pub}} = 2^{\mathcal{ID}} \times \mathcal{L}$, $\mathcal{K} = \emptyset$, $\mathcal{Z} = \{0,1\}^*$. The function $f$ is defined as, for $\mathcal{U}'_K \in 2^{\mathcal{ID}}$ and $\{m_i = (x_i, \mathcal{U}_{M,i}, \mathsf{lab}_{M,i})\}_{i \in \mathcal{U}'_M}$,

$$f(\{i\}_{i \in \mathcal{U}'_K}, \{i, m_i\}_{i \in \mathcal{U}'_M}) = \begin{cases} \{x_i\}_{i \in \mathcal{U}'_M} & \text{the condition below is satisfied} \\ \bot & \text{otherwise} \end{cases}$$

- $\forall i \in \mathcal{U}'_M, \mathcal{U}'_M = \mathcal{U}_{M,i}$.
- $\exists \mathsf{lab}_M \in \mathcal{L}, \forall i \in \mathcal{U}'_M, \mathsf{lab}_{M,i} = \mathsf{lab}_M$.

This means that KeyGen is unnecessary, and Dec works without taking secret keys as input in AoNE (recall that $\mathcal{U}'_K$ can be an empty set).

Chotard et al. showed that sel-sym-IND-secure AoNE can be generically constructed from identity-based encryption [22][15]. We also use pseudorandom functions and non-interactive key exchange with quite simple requirements, which can be realized by the original Diffie-Hellman key exchange. We formally define it in the full version.

### 4.2 Construction of Function-Hiding IP-MCFE

We first construct a function-hiding IP-MCFE scheme as a step to a function-hiding IP-DDFE scheme. Let $\Pi = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2)$ be bilinear groups.

[15] In AoNE, there are no secret keys and thus the IND-security defined in [22] is exactly the same as function-hiding security in our paper.

Let $\mathsf{iFE} = (\mathsf{iSetup}, \mathsf{iKeyGen}, \mathsf{iEnc}, \mathsf{iDec})$ be a function-hiding IPFE scheme (recall that $\mathsf{iKeyGen}, \mathsf{iEnc}$ take a group-element vector as input instead of a $\mathbb{Z}_p$-element vector (see Definition 4.3)) and $H : \mathcal{L} \to \mathbb{G}_1$ be a hash function modeled as a random oracle. The construction of function hiding IP-MCFE for vector length $\mathsf{N}$ is provided in Figure 4.1.

$\mathsf{Setup}(1^\lambda, 1^n)$: On input the security parameter $1^\lambda$, the number of slots $1^n$, the setup algorithm outputs $(\mathsf{PK}, \mathsf{EK}_i, \mathsf{MSK})$ as follows.

$$\{\mathsf{iMSK}_i\}_{i\in[n]} \leftarrow \mathsf{iSetup}(1^\lambda, 1^{2\mathsf{N}+2})$$
$$\mathsf{PK} = \Pi,\ \mathsf{EK}_i = \mathsf{iMSK}_i,\ \mathsf{MSK} = \{\mathsf{EK}_i\}_{i\in[n]}.$$

$\mathsf{KeyGen}(\mathsf{MSK}, \{\mathbf{y}_i\}_{i\in[n]})$: The key generation algorithm takes as input the master secret key $\mathsf{MSK}$, and vectors $\{\mathbf{y}_i\}_{i\in[n]}$ and outputs $\mathsf{SK}$ as follows. It randomly chooses $r_i \in \mathbb{Z}_p$ so that $\sum_{i\in[n]} r_i = 0$ and compute

$$\widehat{\mathbf{y}}_i = (\mathbf{y}_i, 0^\mathsf{N}, r_i, 0),\ \mathsf{iSK}_i \leftarrow \mathsf{iKeyGen}(\mathsf{iMSK}_i, \widehat{\mathbf{y}}_i),\ \mathsf{SK} = \{\mathsf{iSK}_i\}_{i\in[n]}.$$

$\mathsf{Enc}(i, \mathsf{EK}_i, \mathbf{x}_i, \mathsf{lab})$: The encryption algorithm takes as input user index $i \in [n]$, an encryption key $\mathsf{EK}_i$, an input vector $\mathbf{x}_i$, a label $\mathsf{lab}$ and outputs $\mathsf{CT}_i$ as follows.

$$[t_\mathsf{lab}]_1 = H(\mathsf{lab}),\ \widehat{\mathbf{x}}_i = (\mathbf{x}_i, 0^\mathsf{N}, t_\mathsf{lab}, 0),\ \mathsf{CT}_i = \mathsf{iCT}_i \leftarrow \mathsf{iEnc}(\mathsf{iMSK}_i, [\widehat{\mathbf{x}}_i]_1).$$

$\mathsf{Dec}(\mathsf{SK}, \mathsf{CT}_1, ..., \mathsf{CT}_n)$: The decryption algorithm takes as input the secret key $\mathsf{SK}$, ciphertexts $\mathsf{CT}_1, ..., \mathsf{CT}_n$ and outputs $d$ as follows.

$$[d]_T = \prod_{i\in[n]} \mathsf{iDec}(\mathsf{iSK}_i, \mathsf{iCT}_i).$$

**Fig. 4.1.** Function-Hiding IP-MCFE

*Correctness and Security.* For correctly generated $(\mathsf{SK}, \mathsf{CT}_1, ..., \mathsf{CT}_n)$ for $\{\mathbf{y}_i, \mathbf{x}_i\}$, we have

$$\prod_{i\in[n]} \mathsf{iDec}(\mathsf{iSK}_i, \mathsf{iCT}_i) = [\sum_{i\in[n]} \langle \widehat{\mathbf{x}}_i, \widehat{\mathbf{y}}_i \rangle]_T = [\sum_{i\in[n]} \langle \mathbf{x}_i, \mathbf{y}_i \rangle]_T.$$

In our scheme, $\mathsf{EK}_i$ has a power to decode both $\mathsf{CT}_i$ and $\mathsf{SK}_i$ since $\mathsf{EK}_i$ is a part of $\mathsf{MSK}$. This is captured as the function $\mathsf{Leak}_y$ below.

**Theorem 4.8.** *If the SXDH assumption holds in $\mathbb{G}_1$ and* iFE *is function-hiding, then our* IP-MCFE *scheme is* $\mathsf{Leak}_y$*-sel-pos-function-hiding in the random oracle model, where* $\mathsf{Leak}_y(i, \{\mathbf{y}_i\}_{i\in[n]}) = \mathbf{y}_i$.

Due to limited space, we present the proof in the full version.

### 4.3 Construction of Function-Hiding IP-DDFE

We next construct our function-hiding IP-DDFE scheme. Intuitively, our IP-DDFE scheme instantiates our IP-MCFE scheme in parallel per each party set via a pseudorandom function in a non-black box manner. Nevertheless, in the security proof, we can delete the information of the challenge bit $\beta$ in a hybrid sequence similarly to the security proof of IP-MCFE.

Let $\mathsf{iFE} = (\mathsf{iSetup}, \mathsf{iKeyGen}, \mathsf{iEnc}, \mathsf{Dec})$ be a function-hiding IPFE scheme with the length of the random tape for $\mathsf{iSetup}(1^\lambda, 1^{2\mathsf{N}+2})$ being $p(\lambda, \mathsf{N})$, $\mathsf{AoNE} = (\mathsf{aGSetup}, \mathsf{aLSetup}, \mathsf{aEnc}, \mathsf{aDec})$ be an all-or-nothing encryption scheme, $\mathsf{NIKE} = (\mathsf{nSetup}, \mathsf{nKeyGen}, \mathsf{nSharedKey})$ be a non-interactive key exchange scheme, $\{\mathsf{PRF}_1^{\mathsf{K}}\} : \mathcal{L} \to \mathbb{Z}_p$, $\{\mathsf{PRF}_2^{\mathsf{K}}\} : 2^{\mathcal{ID}} \to \{0,1\}^{p(\lambda,\mathsf{N})}$ be families of pseudorandom functions where $\mathcal{ID}$ denotes an identity space, and $H : 2^{\mathcal{ID}} \times \mathcal{L} \to \mathbb{G}_1$ is a hash function modeled as a random oracle. Let $\mathcal{K}_1, \mathcal{K}_2$ be key spaces of $\mathsf{PRF}_1, \mathsf{PRF}_2$. We assume that the range of $\mathsf{nSharedKey}$ and the key space for $\mathsf{PRF}_1$ are the same, namely, $\mathcal{K}_1$. Our construction for vector length $\mathsf{N}$ is provided in Figure 4.2.

*Correctness and Security.* Thanks to the correctness of AoNE, we have $\widetilde{\mathsf{iCT}_i} = \mathsf{iCT}_i$, $\widetilde{\mathsf{iSK}_i} = \mathsf{iSK}_i$. For all $\mathsf{lab}_K, \{\mathsf{K}_{i,j,1}\}, \mathcal{U}$, we have

$$\sum_{i\in\mathcal{U}} r_i = \sum_{i\in\mathcal{U}} \sum_{\substack{j\in\mathcal{U} \\ i\neq j}} (-1)^{j<i} \mathsf{PRF}_1^{\mathsf{K}_{i,j,1}}(\mathsf{lab}_K) = 0$$

since $\mathsf{K}_{i,j,1} = \mathsf{K}_{j,i,1}$. For all $i \in \mathcal{U}$, $\mathsf{iSK}_i$ and $\mathsf{iCT}_i$ are generated under the same $\mathsf{iMSK}_i$ since they are generated using the same random tape $\mathsf{PRF}_2^{\mathsf{K}_{i,2}}(\mathcal{U})$. Thus, thanks to the correctness of iFE, we have $\sum_{i\in\mathcal{U}} \mathsf{iDec}(\widetilde{\mathsf{iSK}_i}, \widetilde{\mathsf{iCT}_i}) = [\sum_{i\in\mathcal{U}} \langle \widehat{\mathbf{x}}_i, \widehat{\mathbf{y}}_i \rangle]_T = [\sum_{i\in\mathcal{U}} \langle \mathbf{x}_i, \mathbf{y}_i \rangle]_T$.

We show security via the following theorem.

**Theorem 4.9.** *If* $\{\mathsf{PRF}_1^{\mathsf{K}}\}, \{\mathsf{PRF}_2^{\mathsf{K}}\}$ *are families of pseudorandom functions,* NIKE *is IND-secure,* AoNE *is sel-sym-IND-secure, the SXDH assumption holds in* $\mathbb{G}_1$*, and* iFE *is function-hiding, then our* IP-DDFE *scheme is sel-sym-function-hiding under the one key-label restriction in the random oracle model.*

Due to space constraints, we present the proof in the full version.

## References


1. Abdalla, M., Benhamouda, F., Gay, R.: From single-input to multi-client inner-product functional encryption. In: ASIACRYPT (2019)
2. Abdalla, M., Benhamouda, F., Kohlweiss, M., Waldner, H.: Decentralizing inner-product functional encryption. In: PKC (2019)
3. Abdalla, M., Bourse, F., De Caro, A., Pointcheval, D.: Simple functional encryption schemes for inner products. In: PKC (2015)
4. Abdalla, M., Catalano, D., Gay, R., Ursu, B.: Inner-product functional encryption with fine-grained access control. In: Asiacrypt (2020)

$\mathsf{GSetup}(1^\lambda)$: On input the security parameter $1^\lambda$, the setup algorithm outputs $\mathsf{PK}$ as follows.

$$\Pi = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2) \leftarrow \mathsf{Gen}(1^\lambda)$$
$$\mathsf{aPP} \leftarrow \mathsf{aGSetup}(1^\lambda),\ \mathsf{nPP} \leftarrow \mathsf{nSetup}(1^\lambda),\ \mathsf{PP} = (\Pi, \mathsf{aPP}, \mathsf{nPP}).$$

$\mathsf{LSetup}(\mathsf{PP})$: On input $\mathsf{PP}$, user $i \in \mathcal{ID}$ generates $(\mathsf{PK}_i, \mathsf{MSK}_i)$ via the setup algorithm as follows.

$$(\mathsf{nPK}_i, \mathsf{nSK}_i) \leftarrow \mathsf{nKeyGen}(\mathsf{nPP}),\ (\mathsf{aPK}_i, \mathsf{aMSK}_i) \leftarrow \mathsf{aLSetup}(\mathsf{aPP}),\ \mathsf{K}_{i,2} \leftarrow \mathcal{K}_2$$
$$\mathsf{PK}_i = (\mathsf{nPK}_i, \mathsf{aPK}_i),\ \mathsf{MSK}_i = (\mathsf{nSK}_i, \mathsf{aMSK}_i, \mathsf{K}_{i,2}).$$

$\mathsf{KeyGen}(\mathsf{MSK}_i, k)$: The key generation algorithm takes the master secret key $\mathsf{MSK}_i$, and an input $k = (\mathbf{y}_i, \mathcal{U}_K, \mathsf{lab}_K)$ such that $i \in \mathcal{U}_K$ and outputs $\mathsf{SK}_i$ as follows.

$$\mathsf{rt}_i = \mathsf{PRF}_2^{\mathsf{K}_{i,2}}(\mathcal{U}_K),\ \mathsf{iMSK}_i = \mathsf{iSetup}(1^\lambda, 1^{2\mathsf{N}+2}; \mathsf{rt}_i),\ \mathsf{K}_{i,j,1} \leftarrow \mathsf{nSharedKey}(\mathsf{nSK}_i, \mathsf{nPK}_j)$$
$$r_i = \sum_{\substack{j \in \mathcal{U}_K \\ i \neq j}} (-1)^{j<i} \mathsf{PRF}_1^{\mathsf{K}_{i,j,1}}(\mathsf{lab}_K),\ \widehat{\mathbf{y}}_i = (\mathbf{y}_i, 0^\mathsf{N}, r_i, 0),\ \mathsf{iSK}_i \leftarrow \mathsf{iKeyGen}(\mathsf{iMSK}_i, \widehat{\mathbf{y}}_i) \tag{4.1}$$
$$\mathsf{aCT}_i \leftarrow \mathsf{aEnc}(\mathsf{aMSK}_i, (\mathsf{iSK}_i, \mathcal{U}_K, \mathsf{lab}_K)),\ \mathsf{SK}_i = (\mathsf{aCT}_i, \mathcal{U}_K, \mathsf{lab}_K). \tag{4.2}$$

$\mathsf{Enc}(\mathsf{MSK}_i, m)$: The encryption algorithm takes as input the public parameters $\mathsf{PK}$, the master secret key $\mathsf{MSK}_i$, and an input $m = (\mathbf{x}_i, \mathcal{U}_M, \mathsf{lab}_M)$ such that $i \in \mathcal{U}_M$ and outputs $\mathsf{CT}_i$ as follows.

$$\mathsf{rt}_i = \mathsf{PRF}_2^{\mathsf{K}_{i,2}}(\mathcal{U}_M),\ \mathsf{iMSK}_i = \mathsf{iSetup}(1^\lambda, 1^{2\mathsf{N}+2}; \mathsf{rt}_i),\ [t]_1 = H(\mathcal{U}_M, \mathsf{lab}_M)$$
$$\widehat{\mathbf{x}}_i = (\mathbf{x}_i, 0^\mathsf{N}, t, 0),\ \mathsf{iCT}_i \leftarrow \mathsf{iEnc}(\mathsf{iMSK}_i, [\widehat{\mathbf{x}}_i]_1) \tag{4.3}$$
$$\mathsf{aCT}_i \leftarrow \mathsf{aEnc}(\mathsf{aMSK}_i, (\mathsf{iCT}_i, \mathcal{U}_M, \mathsf{lab}_M)),\ \mathsf{CT}_i = (\mathsf{aCT}_i, \mathcal{U}_M, \mathsf{lab}_M). \tag{4.4}$$

$\mathsf{Dec}(\{\mathsf{SK}_i\}_{i \in \mathcal{U}_K}, \{\mathsf{CT}_i\}_{i \in \mathcal{U}_M})$: The decryption algorithm takes as input the public parameters $\mathsf{PK}$, secret keys $\{\mathsf{SK}_i\}_{i \in \mathcal{U}_K}$, ciphertexts $\{\mathsf{CT}_i\}_{i \in \mathcal{U}_M}$ such that $\mathcal{U} = \mathcal{U}_K = \mathcal{U}_M$ and outputs $d$ as follows. Perse $\mathsf{SK}_i = (\mathsf{aCT}_i, \mathcal{U}_K, \mathsf{lab}_K)$ and $\mathsf{CT}_i = (\mathsf{aCT}'_i, \mathcal{U}_M, \mathsf{lab}_M)$. Compute

$$\widetilde{\mathsf{iSK}_i} = \mathsf{aDec}(\{\mathsf{aCT}_i\}_{i \in \mathcal{U}}),\ \widetilde{\mathsf{iCT}_i} = \mathsf{aDec}(\{\mathsf{aCT}'_i\}_{i \in \mathcal{U}}),\ [d]_T = \prod_{i \in \mathcal{U}} \mathsf{iDec}(\widetilde{\mathsf{iSK}_i}, \widetilde{\mathsf{iCT}_i}).$$

**Fig. 4.2.** Function Hiding IP-DDFE


5. Agrawal, S., Clear, M., Frieder, O., Garg, S., O'Neill, A., Thaler, J.: Ad hoc multi-input functional encryption. In: ITCS 2020 (2020)
6. Agrawal, S., Goyal, R., Tomida, J.: Multi-party functional encryption. Cryptology ePrint Archive, Report 2020/1266 (2020), https://ia.cr/2020/1266
7. Agrawal, S., Libert, B., Stehle, D.: Fully secure functional encryption for linear functions from standard assumptions, and applications. In: Crypto (2016)

8. Agrawal, S., Yamada., S.: Optimal broadcast encryption from pairings and lwe. In: Proc. of EUROCRYPT (2020)
9. Ananth, P., Jain, A.: Indistinguishability obfuscation from compact functional encryption. In: CRYPTO 2015 (2015)
10. Bishop, A., Jain, A., Kowalczyk, L.: Function-hiding inner product encryption. In: Asiacrypt (2015)
11. Bitansky, N., Vaikuntanathan, V.: Indistinguishability obfuscation from functional encryption. FOCS 2015, 163 (2015), http://eprint.iacr.org/2015/163
12. Boneh, D., Franklin, M.K.: Identity-based encryption from the weil pairing. In: CRYPTO. pp. 213–229 (2001)
13. Boneh, D., Gentry, C., Gorbunov, S., Halevi, S., Nikolaenko, V., Segev, G., Vaikuntanathan, V., Vinayagamurthy, D.: Fully key-homomorphic encryption, arithmetic circuit ABE and compact garbled circuits. In: EUROCRYPT (2014)
14. Boneh, D., Sahai, A., Waters, B.: Functional encryption: Definitions and challenges. In: TCC. pp. 253–273 (2011)
15. Boneh, D., Waters, B.: Conjunctive, subset, and range queries on encrypted data. In: Proceedings of the 4th conference on Theory of cryptography. pp. 535–554. TCC'07, Springer-Verlag, Berlin, Heidelberg (2007), http://dl.acm.org/citation.cfm?id=1760749.1760788
16. Brakerski, Z., Chandran, N., Goyal, V., Jain, A., Sahai, A., Segev, G.: Hierarchical functional encryption. In: ITCS 2017 (2017)
17. Chase, M.: Multi-authority attribute based encryption. In: TCC (2007)
18. Chase, M., Chow, S.S.M.: Improving privacy and security in multi-authority attribute-based encryption. In: ACM CCS 2009 (2009)
19. Chen, Y., Zhang, L., Yiu, S.M.: Practical attribute based inner product functional encryption from simple assumptions. Cryptology ePrint Archive (2019)
20. Chotard, J., Dufour-Sans, E., Gay, R., Phan, D.H., Pointcheval, D.: Decentralized multi-client functional encryption for inner product. In: Asiacrypt (2018)
21. Chotard, J., Dufour-Sans, E., Gay, R., Phan, D.H., Pointcheval, D.: Multi-client functional encryption with repetition for inner product. Cryptology ePrint Archive, Report 2018/1021 (2018)
22. Chotard, J., Dufour-Sans, E., Gay, R., Phan, D.H., Pointcheval, D.: Dynamic decentralized functional encryption. In: Crypto (2020)
23. Datta, P., Okamoto, T., Tomida, J.: Full-hiding (unbounded) multi-input inner product functional encryption from the k-linear assumption. PKC (2018)
24. Goldwasser, S., Gordon, S.D., Goyal, V., Jain, A., Katz, J., Liu, F.H., Sahai, A., Shi, E., Zhou, H.S.: Multi-input functional encryption. In: Eurocrypt (2014)
25. Gorbunov, S., Vaikuntanathan, V., Wee, H.: Predicate encryption for circuits from lwe. In: Crypto (2015)
26. Goyal, V., Pandey, O., Sahai, A., Waters, B.: Attribute-based encryption for fine-grained access control of encrypted data. In: CCS (2006)
27. Katz, J., Sahai, A., Waters, B.: Predicate encryption supporting disjunctions, polynomial equations, and inner products. In: EUROCRYPT. pp. 146–162 (2008)
28. Koppula, V., Waters, B.: Realizing chosen ciphertext security generically in attribute-based encryption and predicate encryption. In: CRYPTO 2019 (2019)
29. Lewko, A., Waters, B.: Decentralizing attribute-based encryption. In: Proceedings of EuroCrypt (2011)
30. Libert, B., Ţiţiu, R.: Multi-client functional encryption for linear functions in the standard model from lwe. In: Asiacrypt (2019)
31. Michalevsky, Y., Joye, M.: Decentralized policy-hiding ABE with receiver privacy. In: ESORICS (2018)

32. Sahai, A., Waters, B.: Fuzzy identity-based encryption. In: Eurocrypt (2005)
33. Shamir, A.: Identity-based cryptosystems and signature schemes. In: Workshop on the theory and application of cryptographic techniques (1984)
34. Waters, B.: Dual system encryption: Realizing fully secure ibe and hibe under simple assumptions. In: Annual International Cryptology Conference (2009)