Multi-Party Functional Encryption

Shweta Agrawal
IIT Madras*

Rishab Goyal
MIT†

Junichi Tomida
NTT Corporation‡

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.

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.

*Email: shweta.a@cse.iitm.ac.in. Work done in part while at the Simons Institute for the Theory of Computing. Also supported by the Swarnajayanti Fellowship, an Indo-French CEFIPRA grant and the CCD Centre of Excellence.

†Email: goyal@utexas.edu. 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.

‡Email: junichi.tomida.vw@hco.ntt.co.jp.

Contents

1 Introduction

Functional encryption (FE) [SW05, BSW11] 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 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 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 [BSW11], 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 [Sha84, BF01], attribute-based encryption [SW05, GPSW06], predicate encryption [KSW08] 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 [GVW15], 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 [AJ15, BV15].

Supporting Multiple Users. Subsequently, many new primitives arose to generalize FE to the multi-user setting – multi-input functional encryption [GGG+14], multi-client functional encryption [CDSG+18a], decentralized multi-client functional encryption, adhoc multi-input functional encryption [ACF+20], multi-authority attribute based encryption [Cha07], dynamic decentralized functional encryption [CDSG+20] 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) [GGG+14] and multi-client functional encryption (MCFE) [CDSG+18a] 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 CT_i, a key authority holding a master secret MSK generates a functional key 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" 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 [MJ18] 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 [Cha07, LW11, BCG+17] 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 P for attributes lab_i. A decryptor with tokens for 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. [CDSG+18a, ABKW19, LT19] 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 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 [ACF+20] 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 [CDSG+20] 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 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 MPFE scheme are the same as in all prior work:

- a Setup algorithm outputs the encryption keys for n_x encryptors and master keys for n_y key authorities. This algorithm[1] 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 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 EK_i.
- a key-generation algorithm 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 MSK_j.
- a decryption algorithm 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, Agg_x, 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 3 for the formal definition and Section 4 for details on how the above primitives can be expressed as instances of MPFE.

Dynamic 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 Agg_x and Agg_y) dynamically so that it can change for every instance of the protocol.

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 [ACF+20] and DDFE [CDSG+20], most constructions in the literature are "static" and rely on centralized or interactive setup [GGG+14, ACGU20, CZY19, ABKW19, LT19, ABG19, CDSG+18a, CDSG+18b]. 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 3 and the dynamic variant in Section 3.1. 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 Appendix A, 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

[1] 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.

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 (see Section 4). 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 ϵ. 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 ABE $\circ$ IPFE. The recent work of Abdalla et al. [ACGU20] (ACGU20) constructed a novel "composition" of ABE and 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 (IPFE) capability. For ease of exposition, we denote this primitive, which is called "IPFE with fine-grained access control" in [ACGU20] by ABE $\circ$ IPFE in our work[2]. Abdalla et al. [ACGU20] provide a construction leveraging state of the art ABE from pairings to support predicates represented by Linear Secret Sharing Schemes (LSSS) in the above functional encryption scheme.

Seen from the lens of MPFE, the 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 ACGU20 construction to the multiparty setting. In more detail, we support $n_y = n$ for some fixed, polynomial n and 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 Agg_x function remains trivial as before but the 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 ABE under the name *multi-authority* ABE, or MA-ABE – here, we extend the access control component of ACGU20 to support the multi-authority setting, i.e. "lift" the primitive of ABE $\circ$ IPFE to MA-ABE $\circ$ IPFE. Our construction departs significantly from ACGU20 in details – our starting point is the MA-ABE construction of Lewko and

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

Waters [LW11] which we extend to support inner product computation. This yields the first construction of a nontrivial multi-authority FE beyond ABE from simple assumptions on pairings to the best of our knowledge.

Using our techniques, we also extend the decentralized attribute based encryption (DABE) scheme of Michalevsky and Joye [MJ18] to support inner product computations. While [MJ18] only supports inner product predicates unlike [LW11], it supports policy hiding unlike the latter – our extension inherits these features.

Function Hiding DDFE. The novel primitive of *dynamic* decentralized inner product functional encryption (IP-DDFE) was recently introduced by Chotard et al. [CDSG+20], 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 PK_i and a master secret key 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 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 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 $\mathbf{x} = (\mathbf{x}_1 \| \ldots \| \mathbf{x}_{n_x})$. The function 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 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 Agg_y, analogously to Agg_x checks that all values $\mathcal{U}_K$ and lab_K are the same for all parties. If so, it outputs the function $f_{\mathcal{U}_K, \mathbf{y} = (\mathbf{y}_1 \| \ldots \| \mathbf{y}_{n_y})}$ 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 [AY20]. 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 Agg_x is trivial) where $(x_{\text{pub}}, x_{\text{pri}}) = (C, m)$ where C is a circuit in 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 Agg_y checks if all the values 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 [LW11, MJ18] to support inner products.

Multi-Authority ABE∘IPFE for LSSS Access Structures. We described the functionality of MA-ABE∘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 ϕ^*, and (3) also make key queries that satisfy the challenge policy predicate ϕ^* 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 MA-ABE portion of the key from being used along with an IPFE portion of an unauthorized key. Another idea is to extend the ABE∘IPFE construction of [ACGU20] 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 [LW11] 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 ρ.

LSetup : The i-th authority samples a length ℓ 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.

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[3], then the partial decryption key is simply $[\langle \boldsymbol{\alpha}_i, \mathbf{u} \rangle]$.

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 $\mathbf{z}_\mathbf{x}$ 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 ρ, 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 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 $[\langle \boldsymbol{\alpha}_i, \mathbf{u} \rangle]$ row-wise in a vector $\mathbf{K}$ as per the function ρ, that is i-th element of $\mathbf{K}$ is $\rho(i)$-th decryption key $[\langle \boldsymbol{\alpha}_{\rho(i)}, \mathbf{u} \rangle]$. It performs the component pairing between $\mathbf{K}$ and CT_1, and then takes the linear combination as specified by $\mathbf{z}_\mathbf{x}$ which can be simplified as follows:

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

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

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 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 [Wat09] as in the multi-authority ABE scheme of [LW11]. 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 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 5 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 [LW11].

DABE ∘ IPFE for Inner Product Predicates, with Policy Hiding. Next, we extend the construction of decentralized attribute based encryption by Michalevsky and Joye [MJ18] to incorporate inner product functional encryption. Observe that [MJ18] supports only inner product predicates but allows for hiding the policy in the ciphertext. While our extension to the scheme of [MJ18] also yields a multi-authority ABE extended to support inner products as above, the details of the transformation are quite different. We observe that the algebraic structure of [MJ18] makes it amenable to incorporating the IPFE functionality using ideas developed in the literature for constructing IPFE generically from public-key encryption which have special structural and homomorphic properties [ABDCP15, ALS16, BJK15]. 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 δ_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 $[r\delta]_T$ to be used as the KEM key, and binds each attribute bit to a ciphertext component as $[(x_i\alpha + w_i)r]$ (where $[\alpha]$ is taken from the CRS). It sets the ciphertext to be $C_0 = [r]$, $C_m = m \cdot [r\delta]_T$, and $C_i = [(x_i\alpha + w_i)r]$ for $i \in [n]$. While a partial secret key for policy vector $\mathbf{y}$ for user GID is simply generated as $K_{i,\mathbf{y}} = [\delta_i - y_i w_i h]$ where $[h]$ 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:

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

As discussed above, we upgrade the [MJ18] construction using ideas from [ABDCP15, ALS16, BJK15] 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 $[r\boldsymbol{\delta}]_T$ 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 $C_m = [r\boldsymbol{\delta} + \mathbf{u}]_T$. 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 GID is generated as $K_{i,\mathbf{y},\mathbf{v}} = [\sum_j \delta_{i,j} v_j - y_i w_i h]$. 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 [MJ18] 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 IPFE. Again, we solve this issue by a careful analysis in the dual system paradigm [Wat09]. We refer the reader to Appendix B for more details.

Function-Hiding DDFE for Inner Products. In this section, we describe the main ideas in the construction of our function hiding DDFE for inner products. The functionality of IP-DDFE was discussed in Section 1.3. Informally, the security of 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 (IP-DDFE) by Chotard et al. [CDSG+20].

The starting point of the IP-DDFE scheme of [CDSG+20] is the multi-client inner product functional encryption (IP-MCFE) scheme in [CDSG+18a], where participants $\{1, ..., n\}$ in the system are a priori fixed, and there is an authority who generates encryption keys mcEK_i for each party and a master secret key mcMSK, which is used to generate secret keys 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 mcMSK_i in what follows). We also recall that in MCFE, only a set of ciphertexts with the same label can be decrypted. Chotard et al. [CDSG+20] lifted the IP-MCFE scheme to an IP-DDFE scheme via following steps. First, each party joins the system dynamically by generating a key K_i of a pseudorandom function (PRF) as a master secret key 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 mcMSK_i of the IP-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 CT_i and SK_i of the IP-DDFE scheme. Second, they introduce a class of DDFE called 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 $\mathsf{mcDec}(\mathsf{mcSK}_{\mathcal{U}}, \{\mathsf{mcCT}_{i,\mathcal{U}}\}_{i\in\mathcal{U}})$. DSum also plays a role in preventing combination of partial secret keys for which the agreed vectors are inconsistent.

Our Function-Hiding IP-DDFE. Our approach is to lift function-hiding IP-MCFE to function-hiding IP-DDFE following their blueprint. Unfortunately, there are no function-hiding IP-MCFE schemes, and we need to start with constructing this. Our first idea is to leverage the recent conversion by Abdalla et al. [ABG19] from IPFE to IP-MCFE. However, this idea does not work since all parties share the same encryption key of an IPFE scheme in their converted schemes, and once a single party is corrupted, the adversary can completely decode the entire secret keys. Thus, their conversion is not applicable to the function-hiding setting.

To address this challenge, we devise a new technique to convert function-hiding IPFE to function-hiding IP-MCFE, which is inspired by the function-hiding multi-input IPFE scheme by Datta et al. [DOT18]. In their scheme, each party i has a master secret key iMSK_i of a function-hiding IPFE scheme, the ciphertext miCT_i of $\mathbf{x}_i$ is $\mathsf{iCT}_i[(\mathbf{x}_i, 1)]$, and the secret key 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 IPFE scheme, respectively. To lift their MIFE to 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 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 IP-MCFE to function-hiding IP-DDFE. To do so, we must address additional technical challenges as described next. In the original definition of IP-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 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 [CC09] to generate such shares via pseudorandom function. Please see Section 6 for more details.

Distributed Ciphertext-Policy ABE The recent construction of Agrawal-Yamada [AY20] 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 [AY20] 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 μ 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 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 MSK_i along with 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 [AY20], which in turn uses the BGG^+ ABE construction [BGG+14]. Roughly, a BGG^+ ciphertext is sampled in two steps — first, it samples a sequence of 2ℓ encodings $\{\psi_{i,b}\}_{i,b}$; second, depending upon the attribute $\mathbf{x}$ the final ciphertext consists of ℓ encodings $\{\psi_{i,x_i}\}_i$. (Note that BGG^+ is a key-policy scheme, whereas we are building a ciphertext-policy system.) The main idea behind the ciphertext-policy ABE of [AY20] is as follows:

Setup : Sample 2ℓ 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.

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

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

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 BGG^+ decryption in the exponent to learn the plaintext.

For the multi-authority extension, each authority samples its own sequence of 2ℓ random exponents $w_{i,b}^{(j)}$ for $j \in [N]$. Then during encryption, the encryptor N-out-of-N (additively) secret shares the 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 BGG^+ ciphertext $\{\psi_{i,x_i}\}_i$ which it can decrypt as before. In order to let multiple independent authorities sample the same δ, 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 [AY20]. 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 Section 7 for more details.

1.5 Predicting New and Useful Primitives via MPFE

One of the most exciting benefits of 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, $\mathsf{D\text{-}MCFE}$).

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.

To supplement this new abstraction, we sketch a generic construction of this new multi-key FE primitive from a multi-authority ABE scheme in Section 8. We also identify other useful new primitives that have not been studied before as tagets for future work.

Acknowledgements. We thank Fabrice Mouhartem for helpful discussions and collaboration during early stages of this work. We are grateful to Edouard Dufour Sans, David Pointcheval and Romain Gay for insightful discussions about DDFE which led to refinements in the definition of dynamic MPFE.

2 Preliminaries

Notation. We begin by defining the notation that we will use throughout the paper. We use bold letters to denote vectors and the notation $[a, b]$ to denote the set of integers $\{k \in \mathbb{N} \mid a \leq k \leq b\}$. We use $[n]$ to denote the set $[1, n]$. Concatenation is denoted by the symbol $\|$.

Throughout the paper, we use λ to denote the security parameter. We say a function $f(\lambda)$ is *negligible* if it is $O(\lambda^{-c})$ for all $c > 0$, and we use $\mathsf{negl}(\lambda)$ to denote a negligible function of λ. We say $f(\lambda)$ is *polynomial* if it is $O(\lambda^c)$ for some constant $c > 0$, and we use $\mathsf{poly}(\lambda)$ to denote a polynomial function of λ. We use the abbreviation PPT for probabilistic polynomial-time. The function $\log x$ is the base 2 logarithm of x.

2.1 Access Structures and Linear Secret-Sharing Schemes

We recall the concepts of access structures and linear secret-sharing schemes (LSSS). We follow the notation from prior works [GPSW06, LW11].

Definition 2.1 (Access Structures)**.** Let $\{P_i\}_{i \in [n]}$ be a set of parties. A collection $\mathbb{A} \subseteq 2^{\{P_1, \ldots, P_n\}}$ is monotone if $\forall B, C$: if $B \in \mathbb{A}$ and $B \subseteq C$, then $C \in \mathbb{A}$. An access structure (respectively, monotone access structure) is a collection (respectively, monotone collection) $\mathbb{A}$ of non-empty subsets of $\{P_i\}_{i \in [n]}$. The sets in $\mathbb{A}$ are called the authorized sets, and the sets not in $\mathbb{A}$ are called the unauthorized sets.

As in prior work in the bilinear setting, attributes will play the role of parties and we will only consider monotone access structures. We observe that more general access structures can be (inefficiently) realized with our techniques by letting the negation of an attribute be a separate attribute (this doubles the total number of attributes).

Definition 2.2 (Linear Secret-Sharing Schemes (LSSS))**.** A secret sharing scheme Π over a set of parties $\mathcal{P}$ is called linear (over $\mathbb{Z}_p$) if:

1. The shares for each party form a vector over $\mathbb{Z}_p$.
2. There exists a $\ell \times m$ matrix $\mathbf{A}$ called the share-generating matrix for Π such that the x-th row of $\mathbf{A}$ is labeled by a party $\rho(x)$ (ρ is a function from row index to party index). When we consider the column vector $\mathbf{v} = (s, r_2, \ldots, r_n)^\top$, where $s \in \mathbb{Z}_p$ is the secret to be shared and $r_2, \ldots, r_n \in \mathbb{Z}_p$ are randomly

chosen, then $\mathbf{A} \cdot \mathbf{v}$ is the vector of ℓ shares of the secret s according to Π. The share $(\mathbf{A} \cdot \mathbf{v})_x$ belongs to party $\rho(x)$.

Linear reconstruction property: Let S denote an authorized set, and define $I \subseteq [\ell]$ as $I = \{x : \rho(x) \in S\}$. Then the first basis vector of canonical basis of $\mathbb{Z}_p^m$, that is $(1, 0, \ldots, 0)^\top$ is in the span of rows of $\mathbf{A}$ indexed by I, and there exist constants $\{w_x \in \mathbb{Z}_p\}_{x \in I}$ such that, for any valid shares $\{\mathsf{sh}_x\}_x$ of a secret s according to Π, we have: $\sum_{x \in I} w_x \mathsf{sh}_x = s$. And, these constants can be found in polynomial time with respect to the size of the share-generating matrix $\mathbf{A}$.

2.2 Bilinear Map Preliminaries

Here, we introduce our notation for bilinear maps and the bilinear generic group model following Baltico et.al [BCFG17], who specializes the framework by Barthe [BFF+14] for defining generic k-linear groups to the bilinear group settings. The definition closely follows that of Maurer [Mau05], which is equivalent to the alternative formulation by Shoup [Sho97].

Notation on Bilinear Maps. In this work, we use bilinear groups for our various MPFE constructions. Some of our constructions rely on composite order bilinear groups, while other rely on prime order asymmetric bilinear groups. Below we define our notations for both.

Asymmetric prime order bilinear groups. An asymmetric bilinear group generator takes as input 1^λ and outputs a group description
$\mathbb{G} = (q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2)$, where q is a prime of $\Theta(\lambda)$ bits, $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{G}_T$ are cyclic groups of order q, $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is a non-degenerate bilinear map, and g_1 and g_2 are generators of $\mathbb{G}_1$ and $\mathbb{G}_2$, respectively. In more detail, we have:

- Bilinearity: $\forall g_1 \in \mathbb{G}_1, g_2, \in \mathbb{G}_2, a, b \in \mathbb{Z}_p, e(g_1^a, g_2^b) = e(g_1, g_2)^{ab}$,
- Non-Degeneracy: $e(g_1, g_2) \neq 1_{\mathbb{G}_T}$ for $g_1 \neq 1_{\mathbb{G}_1}$ and $g_2 \neq 1_{\mathbb{G}_2}$, where $1_{\mathbb{G}_1}, 1_{\mathbb{G}_2}$ and $1_{\mathbb{G}_T}$ are the identity elements of groups $\mathbb{G}_1, \mathbb{G}_2$ and $\mathbb{G}_T$ respectively.

We require that the group operations in $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{G}_T$ as well as the bilinear map e can be efficiently computed. We employ the implicit representation of group elements: for a matrix $\mathbf{A}$ over $\mathbf{Z}_q$, we define $[\mathbf{A}]_1 := g_1^{\mathbf{A}}$, $[\mathbf{A}]_2 := g_2^{\mathbf{A}}$, $[\mathbf{A}]_T := g_T^{\mathbf{A}}$, where exponentiation is carried out component-wise.

We also use the following less standard notations. For vectors $\mathbf{w} = (w_1, \ldots, w_\ell)^\top \in \mathbf{Z}_q^\ell$ and $\mathbf{v} = (w_1, \ldots, w_\ell)^\top \in \mathbf{Z}_q^\ell$ of the same length, $\mathbf{w} \odot \mathbf{v}$ denotes the vector that is obtained by component-wise multiplications. Namely, $\mathbf{v} \odot \mathbf{w} = (v_1 w_1, \ldots, v_\ell w_\ell)^\top$. When $\mathbf{w} \in (\mathbf{Z}_q^*)^\ell$, $\mathbf{v} \oslash \mathbf{w}$ denotes the vector $\mathbf{v} \oslash \mathbf{w} = (v_1/w_1, \ldots, v_\ell/w_\ell)^\top$. It is easy to verify that for vectors $\mathbf{c}, \mathbf{d} \in \mathbf{Z}_q^\ell$ and $\mathbf{w} \in (\mathbf{Z}_q^*)^\ell$, we have $(\mathbf{c} \odot \mathbf{w}) \odot (\mathbf{d} \oslash \mathbf{w}) = \mathbf{c} \odot \mathbf{d}$. For group elements $[\mathbf{v}]_1 \in \mathbb{G}_1^\ell$ and $[\mathbf{w}]_1 \in \mathbb{G}_2^\ell$, $[\mathbf{v}]_1 \odot [\mathbf{w}]_2$ denotes $([v_1 w_1]_T, \ldots, [v_\ell w_\ell]_T)^\top$, which is efficiently computable from $[\mathbf{v}]_1$ and $[\mathbf{w}]_2$ using the bilinear map e.

We now recall the k-linear assumption on prime order bilinear groups over source group $\mathbb{G}_1$. It can be analogously define for group $\mathbb{G}_2$ as well.

Assumption 2.3 (k-linear in $\mathbb{G}_1$)**.** For every PPT adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

$$\Pr\left[\mathcal{A}\left(\Pi, g_1, g_2, g_1^{\mathbf{a}}, g_1^{\mathbf{z}_\beta}\right) = b : \begin{array}{c} \Pi = (q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e(\cdot,\cdot)) \leftarrow \mathsf{Gen}(1^\lambda) \\ g_1 \leftarrow \mathbb{G}_1, g_2 \leftarrow \mathbb{G}_2, \beta \leftarrow \{0,1\} \\ \mathbf{a} \leftarrow \mathbb{Z}_q^k, \mathbf{b} \leftarrow \mathbb{Z}_q^k, \mathbf{z}_1 \leftarrow \mathbb{Z}_q^{k+1} \\ \mathbf{z}_0 = (a_1 b_1, \ldots, a_k b_k, \sum_i b_i) \end{array}\right] \leq 1/2 + \mathsf{negl}(\lambda).$$

Generic Bilinear Group Model. Let $\mathbb{G} = (q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2)$ be a bilinear group setting, L_1, L_2, and L_T be lists of group elements in $\mathbf{G}_1$, $\mathbf{G}_2$, and $\mathbf{G}_T$ respectively, and let $\mathcal{D}$ be a distribution over L_1,

State: Lists L_1, L_2, L_T over $\mathbf{G}_1$, $\mathbf{G}_2$, $\mathbf{G}_T$ respectively.

Initializations: Lists L_1, L_2, L_T sampled according to distribution $\mathcal{D}$.

Oracles: The oracles provide black-box access to the group operations, the bilinear map, and equalities.

- For all $s \in \{1, 2, T\}$: $\mathsf{add}_s(h_1, h_2)$ appends $L_s[h_1] + L_s[h_2]$ to L_s and returns its handle $(s, |L_s|)$.
- For all $s \in \{1, 2, T\}$: $\mathsf{neg}_s(h_1, h_2)$ appends $L_s[h_1] - L_s[h_2]$ to L_s and returns its handle $(s, |L_s|)$.
- $\mathsf{map}_e(h_1, h_2)$ appends $e(L_1[h_1], L_2[h_2])$ to L_T and returns its handle $(T, |L_T|)$.
- $\mathsf{zt}_T(h)$ returns 1 if $L_T[h] = 0$ and 0 otherwise.

All oracles return $\perp$ when given invalid indices.

Figure 2.1: Generic group model for bilinear group setting $\mathbf{G} = (q, \mathbf{G}_1, \mathbf{G}_2, \mathbf{G}_T, e, g_1, g_2)$ and distribution $\mathcal{D}$.

L_2, and L_T. The generic group model for a bilinear group setting $\mathbf{G}$ and a distribution $\mathcal{D}$ is described in Fig. 2.1. In this model, the challenger first initializes the lists L_1, L_2, and L_T by sampling the group elements according to $\mathcal{D}$, and the adversary receives handles for the elements in the lists. For $s \in \{1, 2, T\}$, $L_s[h]$ denotes the h-th element in the list L_s.

The handle to this element is simply the pair (s, h). An adversary running in the generic bilinear group model can apply group operations and bilinear maps to the elements in the lists. To do this, the adversary has to call the appropriate oracle specifying handles for the input elements. The challenger computes the result of a query, stores it in the corresponding list, and returns to the adversary its (newly created) handle. Handles are not unique (i.e., the same group element may appear more than once in a list under different handles).

We remark that we slightly simplify the generic group model of Baltico et. al [BCFG17]. Whereas they allow the adversary to access the equality test oracle, which is given two handles (s, h_1) and (s, h_2) and returns 1 if $L_s[h_1] = L_s[h_2]$ and 0 otherwise for all $s \in \{1, 2, T\}$, we replace this oracle with the zero-test oracle, which is given a handle (s, h) and returns 1 if $L_s[h] = 0$ and 0 otherwise only for the case of $s = T$. We claim that even with this modification, the model is equivalent to the original one. This is because we can perform the equality test for (s, h_1) and (s, h_2) using our restricted oracles as follows. Let us first consider the case of $s = T$. In this case, we can get the handle (T, h') corresponding to $L_T[h_1] - L_T[h_2]$ by calling neg_T and add_T. We then make a zero-test query for (T, h'). Clearly, we get 1 if $L_s[h_1] = L_s[h_2]$ and 0 otherwise. We next consider the case of $s \in \{1, 2\}$. This case can be reduced to the case of $s = T$ by lifting the group elements corresponding to h_1 and h_2 to the group elements in $\mathbf{G}_T$ by taking bilinear maps with an arbitrary non-unit group element in $\mathbf{G}_{3-s}$, which is possible by calling map_e.

Symbolic Group Model. The symbolic group model for a bilinear group setting $\mathbf{G}$ and a distribution $\mathcal{D}_P$ gives to the adversary the same interface as the corresponding generic group model, except that internally the challenger stores lists of element in the field $\mathbf{Z}_p(X_1, \ldots, X_n)$ instead of lists of group elements. The oracles add_s, neg_s, map, and zt computes addition, negation, multiplication, and equality in the field. In our work, we will use the subring $\mathbf{Z}_p[X_1, \ldots, X_n, 1/X_1, \ldots, 1/X_n]$ of the entire field $\mathbf{Z}_p(X_1, \ldots, X_n)$. Note that any element f in $\mathbf{Z}_p[X_1, \ldots, X_n, 1/X_1, \ldots, 1/X_n]$ can be represented as

$$f(X_1, \ldots, X_n) = \sum_{(c_1, \ldots, c_n) \in \mathbf{Z}^n} a_{c_1, \ldots, c_n} X_1^{c_1} \cdots X_n^{c_n}$$

using $\{a_{c_1, \ldots, c_n} \in \mathbf{Z}_p\}_{(c_1, \ldots, c_n) \in \mathbf{Z}^n}$, where we have $a_{c_1, \ldots, c_n} = 0$ for all but finite $(c_1, \ldots, c_n) \in \mathbf{Z}^n$. Note that this expression is unique.

Composite order bilinear groups. A composite order bilinear group generator takes as input 1^λ and number of prime-order subgroups k, and outputs a group description $((p_1, p_2, \ldots, p_k), (N, \mathbb{G}, \mathbb{G}_T, e(\cdot,\cdot)))$, where p_i's are primes such that $p_i \in \{2^{\lambda-1}, \ldots, 2^\lambda - 1\}$ for all $i \in [k]$, $N = \prod_i p_i$, $\mathbb{G}$ and $\mathbb{G}_T$ are groups of order N and e is an efficiently computable function computing a non-degenerate bilinear map taking two group elements of $\mathbb{G}$ to a group element in $\mathbb{G}_T$. For any set $S \subseteq [k]$, let $\mathbb{G}_S \subseteq \mathbb{G}$ denote the (unique) subgroup of order $p_S = \prod_{i \in S} p_i$. Observe that for any $S \subseteq [k]$, given $\{p_i\}_{i \in S}$, one can sample a uniformly random element from $\mathbb{G}_S$.

We now state the assumptions we make on composite order bilinear groups. All our assumptions belong in the family of subgroup decision assumptions introduced by Bellare, Waters, and Yilek [BWY11]. These same assumptions appear in the work of Lewko and Waters [LW11].

Assumption 2.4. We say that the assumption holds with respect to Gen for three primes if for any stateful PPT adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

$$\Pr\left[\mathcal{A}(\Pi, g_1, T_b) = b : \begin{array}{c}((p_1,p_2,p_3),\Pi) \leftarrow \mathsf{Gen}(1^\lambda, 3);\ \Pi = (N, \mathbb{G}, \mathbb{G}_T, e) \\ g_1 \leftarrow \mathbb{G}_1; T_0 \leftarrow \mathbb{G};\ T_1 \leftarrow \mathbb{G}_1;\ b \leftarrow \{0,1\}\end{array}\right] \leq 1/2 + \mathsf{negl}(\lambda).$$

Assumption 2.5. We say that the assumption holds with respect to Gen for three primes if for any stateful PPT adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

$$\Pr\left[\mathcal{A}(\Pi, g_1, g_3, h_1h_2, T_b) = b : \begin{array}{c}((p_1,p_2,p_3),\Pi) \leftarrow \mathsf{Gen}(1^\lambda, 3);\ \Pi = (N, \mathbb{G}, \mathbb{G}_T, e) \\ g_1, h_1 \leftarrow \mathbb{G}_1; h_2 \leftarrow \mathbb{G}_2; g_3 \leftarrow \mathbb{G}_3 \\ T_0 \leftarrow \mathbb{G}_1;\ T_1 \leftarrow \mathbb{G}_{\{1,2\}};\ b \leftarrow \{0,1\}\end{array}\right] \leq 1/2 + \mathsf{negl}(\lambda).$$

Assumption 2.6. We say that the assumption holds with respect to Gen for three primes if for any stateful PPT adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

$$\Pr\left[\mathcal{A}(\Pi, g_1, h_1g_3, g_2h_3, T_b) = b : \begin{array}{c}((p_1,p_2,p_3),\Pi) \leftarrow \mathsf{Gen}(1^\lambda, 3);\ \Pi = (N, \mathbb{G}, \mathbb{G}_T, e) \\ g_1, h_1 \leftarrow \mathbb{G}_1; g_2 \leftarrow \mathbb{G}_2; g_3, h_3 \leftarrow \mathbb{G}_3 \\ T_0 \leftarrow \mathbb{G}_{\{1,2\}};\ T_1 \leftarrow \mathbb{G}_{\{1,3\}};\ b \leftarrow \{0,1\}\end{array}\right] \leq 1/2 + \mathsf{negl}(\lambda).$$

Assumption 2.7. We say that the assumption holds with respect to Gen for three primes if for any stateful PPT adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

$$\Pr\left[\mathcal{A}\left(\begin{array}{c}\Pi, \{g_i\}_{i\in[3]}, T_b, \\ g_1^a, g_1^b g_3^b, g_1^c, g_1^{ac} g_3^d\end{array}\right) = b : \begin{array}{c}((p_1,p_2,p_3),\Pi) \leftarrow \mathsf{Gen}(1^\lambda, 3);\ \Pi = (N, \mathbb{G}, \mathbb{G}_T, e) \\ g_1 \leftarrow \mathbb{G}_1; g_2 \leftarrow \mathbb{G}_2; g_3 \leftarrow \mathbb{G}_3 \\ a,b,c,d \leftarrow \mathbb{Z}_N,\ T_0 \leftarrow e(g_1,g_1)^{abc};\ T_1 \leftarrow \mathbb{G}_T;\ b \leftarrow \{0,1\}\end{array}\right] \leq 1/2 + \mathsf{negl}(\lambda).$$

2.3 Basic Tools

Pseudorandom Functions. A pseudorandom function (PRF) family $\mathcal{F} = \{\mathsf{PRF}^\mathsf{K}\}_{\mathsf{K} \in \mathcal{K}}$ with a key space $\mathcal{K}$, a domain $\mathcal{X}$, and a range $\mathcal{Y}$ is a function family that consists of functions $\mathsf{PRF}^\mathsf{K} : \mathcal{X} \to \mathcal{Y}$.

Definition 2.8. Let $\mathcal{R}$ be a set of functions consisting of all functions whose domain and range are $\mathcal{X}$ and $\mathcal{Y}$ respectively. A PRF family $\mathcal{F}$ is said to be secure if for any PPT adversary $\mathcal{A}$, the following condition holds,

$$|\Pr[\mathcal{A}^{\mathsf{PRF}^\mathsf{K}(\cdot)}(1^\lambda) = 1] - \Pr[\mathcal{A}^{R(\cdot)}(1^\lambda) = 1]| \leq \mathsf{negl}(\lambda),$$

where $K \leftarrow \mathcal{K}$ and $R \leftarrow \mathcal{R}$.

Non-interactive key exchange (NIKE). A NIKE scheme for shared key space $\mathcal{K}$ consists of the three algorithms.

$\mathsf{Setup}(1^\lambda) \to \mathsf{PP}$. It takes a security parameter 1^λ and outputs a public parameter PP.

$\mathsf{KeyGen}(\mathsf{PP}) \to (\mathsf{PK}, \mathsf{SK})$. It takes PP and outputs a public key PK and the corresponding secret key SK.

$\mathsf{SharedKey}(\mathsf{PK}, \mathsf{SK}) \to \mathsf{K}$. It takes PK and SK and deterministically outputs a shared key $\mathsf{K} \in \mathcal{K}$.

Correctness. A NIKE scheme is correct if, for all $\lambda \in \mathbb{N}$, we have

$$\Pr\left[\mathsf{K}_{i,j} = \mathsf{K}_{j,i} : \begin{array}{l} \mathsf{PP} \leftarrow \mathsf{Setup}(1^\lambda) \\ (\mathsf{PK}_i, \mathsf{SK}_i), (\mathsf{PK}_j, \mathsf{SK}_j) \leftarrow \mathsf{KeyGen}(\mathsf{PP}) \\ \mathsf{K}_{i,j} = \mathsf{SharedKey}(\mathsf{PK}_i, \mathsf{SK}_j) \\ \mathsf{K}_{j,i} = \mathsf{SharedKey}(\mathsf{PK}_j, \mathsf{SK}_i) \end{array}\right] = 1.$$

Security. We say a NIKE scheme is IND-secure if, for all stateful PPT adversaries $\mathcal{A}$, we have

$$\Pr\left[\beta = \beta' : \begin{array}{l} \beta \leftarrow \{0,1\},\ \mathsf{PP} \leftarrow \mathsf{Setup}(1^\lambda) \\ \mathcal{S} \leftarrow \mathcal{A}(\mathsf{PP}) \\ (\mathsf{PK}_i, \mathsf{SK}_i) \leftarrow \mathsf{KeyGen}(\mathsf{PP}) \\ \mathcal{CS}, (i', j') \leftarrow \mathcal{A}(\{\mathsf{PK}_i\}_{i \in \mathcal{S}}) \text{ where } i', j' \in \mathcal{S} \backslash \mathcal{CS} \text{ and } i' \neq j' \\ \mathsf{K}^0_{i',j'} = \mathsf{SharedKey}(\mathsf{PK}_{i'}, \mathsf{SK}_{j'}),\ \mathsf{K}^1_{i',j'} \leftarrow \mathcal{K} \\ \beta' \leftarrow \mathcal{A}(\{\mathsf{SK}_i\}_{i \in \mathcal{CS}}, \mathsf{K}^\beta_{i',j'}) \end{array}\right] \leq 1/2 + \mathsf{negl}(\lambda).$$

3 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.[4] Given input the security parameter, number of ciphertext inputs n_x, number of key inputs n_y and two aggregation functions Agg_x, Agg_y as defined above, this algorithm outputs a set of encryption keys $\{\mathsf{EK}_i\}_{i \leq n_x}$, master secret keys $\{\mathsf{MSK}_i\}_{i \leq n_y}$ and public key PK.

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

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

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

$\mathsf{Decrypt}\left(\mathsf{PK}, \{\mathsf{SK}_j\}_{j\le n_y}, \{\mathsf{CT}_i\}_{i\le n_x}\right)$: Given input the public key 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 GSetup along with a local setup, denoted by 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 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 Agg_x and 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 MPFE. The security definition is modelled in a similar fashion to MIFE security [GGG+14, §2.2] while taking into account corruption queries.

For any choice of parameters λ, 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 EK_i or master secret key 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 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.)[5]

$\mathsf{Key}^{\mathsf{K},\beta}(\cdot,\cdot)$, 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 $y_j^{k,b} = \left(y_{j,\text{pub}}^{k,b}, y_{j,\text{pri}}^{k,b}\right)$ 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 MSK_j and sends it to the adversary. (Here β 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 $x_i^{\ell,b} = \left(x_{i,\text{pub}}^{\ell,b}, x_{i,\text{pri}}^{\ell,b}\right)$ 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 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 $x_{\text{pub}}^{\ell,0} = x_{\text{pub}}^{\ell,1}$ for all $\ell \in [Q_x]$, and $y_{\text{pub}}^{k,0} = y_{\text{pub}}^{k,1}$ for all $k \in [Q_y]$.

[5] Note that in case EK_i is completely contained in some 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.

2. For each of the encryption and key challenges, the *private* components of the two challenges are also equal, namely $x_{\text{pri}}^{\ell,0} = x_{\text{pri}}^{\ell,1}$ 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 $y_{\text{pri}}^{k,0} = y_{\text{pri}}^{k,1}$ 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 EK_i was corrupted, and ii) for every $j \in [n_y]$, either y_j^b was queried or MSK_j was corrupted, and iii) at least one of inputs $\{x_i^b\}, \{y_j^b\}$ 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 MPFE scheme $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt})$ is said to be IND secure if for any *admissible* 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 3.1 (Weaker notions of security)**.** We say the scheme is selective 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 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.

3.1 Dynamic Multi-Party Functional Encryption

In this section, we define the dynamic notion of multi-party functional encryption (MPFE). We consider the fully dynamic setting where the number of key/ciphertext inputs is unspecified during setup time, and the aggregation functions are also specified only during key generation and encryption times. In the dynamic setting, an interactive or centalized setup is not meaningful since the number of parties is itself not known during setup time, hence we restrict ourselves to the local setup mode for simplicity.

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. Also, let $\mathcal{PK}$ be the space to which each local public key belongs. A dynamic multi-party functional encryption scheme (MPFE) with local setup is defined as a tuple of 5 algorithms/protocols $\mathsf{MPFE} = (\mathsf{GSetup}, \mathsf{LSetup}, \mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt})$ with the following syntax:

$\mathsf{GSetup}(1^\lambda)$: On input the security parameter, the global setup algorithm samples a globally shared set of public parameters PP.

$\mathsf{LSetup}(\mathsf{PP})$: Given input the public parameters, the local setup algorithm outputs a tuple consisting of local public key PK, an encryption key EK, and a master secret key MSK. (Here the local public key is just regarded as a public identifier for the user, and not given as explicit input to other algorithms since it could always be added to the encryption and/or master secret key.)

$\mathsf{KeyGen}\left(\mathsf{MSK}, j, y = (y_{\text{pub}}, y_{\text{pri}}), \mathsf{Agg}_y\right)$: Given input a master secret key MSK, user index $j \in [n_y]$[6], a function input $y = (y_{\text{pub}}, y_{\text{pri}})$, and an aggregation function $\mathsf{Agg}_y : (\mathcal{PK} \times \mathcal{Y})^{n_y} \to \mathcal{Y}^*$ (for some $n_y \in \mathbb{N}$), this algorithm outputs a secret key SK_j.

[6]Note that both the key generator and encryptor take as input the user index as well. This is not an additional requirement as in our current definition we are modeling the aggregation functions as a circuit, and thus both the key generator and encryptor could explicitly provide at which slot in the aggregation circuit do they want their inputs to be read. As we remark later in Remark 3.2, if we model the aggregation function in the uniform computation model, then we could avoid providing the user index as an input to the key generation and encryption algorithms.

$\mathsf{Encrypt}\left(\mathsf{EK}, i, x = (x_{\text{pub}}, x_{\text{pri}}), \mathsf{Agg}_x\right)$: Given input an encryption key EK, user index $i \in [n_x]$, an input $x = (x_{\text{pub}}, x_{\text{pri}})$, and an aggregation function $\mathsf{Agg}_x : (\mathcal{PK} \times \mathcal{X})^{n_x} \to \mathcal{X}^*$ (for some $n_x \in \mathbb{N}$), this algorithm outputs a ciphertext CT_i.

$\mathsf{Decrypt}\left((\mathsf{SK}_j)_j, (\mathsf{CT}_i)_i\right)$: Given input a sequence of secret keys $(\mathsf{SK}_j)_j$ and a sequence of ciphertexts $(\mathsf{CT}_i)_i$, this algorithm outputs a value z or $\perp$.

Correctness. We say that an MPFE scheme is *correct* if, $\forall (N, n_x, n_y) \in \mathbb{N}^3$, 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 Agg_x and Agg_y, and indexing functions $\text{index}_x : [n_x] \to [N]$, $\text{index}_y : [n_y] \to [N]$ it holds that:

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

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

Indistinguishability based security. Here we extend the security experiment for multi-party functional encryption that we provided in Section 3 to the dynamic user setting in the local setup mode. Since we are working in the dynamic setting, we need to define the following oracles[7]:

$\mathsf{HonestGen}()$, upon a call to this oracle, the challenger samples a fresh tuple of local public key, encryption key, and master key $(\mathsf{PK}, \mathsf{EK}, \mathsf{MSK})$, and stores them in a list $Ł_{\mathsf{setup}}$. It sends PK to the adversary. (Note that if the scheme is a public key scheme, then the challenger sends the encryption key EK to the adversary.)

$\mathsf{Corrupt}(\cdot, \cdot)$, upon a call to this oracle for an honest user local public key PK and key type $\mathsf{type} \in \{\mathsf{enc}, \mathsf{master}\}$, the challenger first checks whether the list $Ł_{\mathsf{setup}}$ contains a key pair associated with PK. If there is such a key pair $(\mathsf{PK}, \mathsf{EK}, \mathsf{MSK})$, then it sends either the EK or MSK depending on the type queried. Otherwise, it sends nothing.[8]

$\mathsf{Key}^\beta(\cdot, \cdot, \cdot, \cdot)$, upon a call to this oracle for an honest user local public key PK, function inputs $(y_j^{k,0}, y_j^{k,1})$ (where $y_j^{k,b} = \left(y_{j,\text{pub}}^{k,b}, y_{j,\text{pri}}^{k,b}\right)$ for $b \in \{0,1\}$), index j, aggregation function $\mathsf{Agg}_{y,j}^k$, the challenger first checks whether the list $Ł_{\mathsf{setup}}$ contains a key pair associated with PK. If there is such a key pair $(\mathsf{PK}, \mathsf{EK}, \mathsf{MSK})$, then it samples a decryption key for function input $y_j^{k,\beta}$ using key MSK and sends it to the adversary. Otherwise, it sends nothing. (Here β is the challenge bit chosen at the start of the experiment.)

$\mathsf{Enc}^\beta(\cdot, \cdot, \cdot, \cdot)$, upon a call to this oracle for an honest user local public key PK, inputs $(x_j^{\ell,0}, x_j^{\ell,1})$ (where $x_j^{\ell,b} = \left(x_{j,\text{pub}}^{\ell,b}, x_{j,\text{pri}}^{\ell,b}\right)$ for $b \in \{0,1\}$), index j, aggregation function $\mathsf{Agg}_{x,j}^\ell$, the challenger first checks whether the list $Ł_{\mathsf{setup}}$ contains a key pair associated with PK. If there is such a key pair $(\mathsf{PK}, \mathsf{EK}, \mathsf{MSK})$, then it samples a ciphertext for input $x_j^{\ell,\beta}$ using key EK and sends it to the adversary. Otherwise, it sends nothing. (Here β is the challenge bit chosen at the start of the experiment.)

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

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

[7] Note that all these oracles have access to all other oracle's state.

[8] As we point out in the static setting, in case EK_i is completely contained in some MSK_i (or vice versa), then making a master secret corruption query for i will also imply that encryption key for i has been corrupted too (and vice versa).

1. For each of the encryption and key challenges, the public components of the two challenges are equal, namely $x^{\ell,0}_{j,\text{pub}} = x^{\ell,1}_{j,\text{pub}}$ for all $\ell \in [Q_x]$, and $y^{k,0}_{j,\text{pub}} = y^{k,1}_{j,\text{pub}}$ 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 $x^{\ell,0}_{j,\text{pri}} = x^{\ell,1}_{j,\text{pri}}$ for all $\ell \in [Q_x]$, and $y^{k,0}_{j,\text{pri}} = y^{k,1}_{j,\text{pri}}$ for all $k \in [Q_y]$ if the encryption key EK^ℓ or the master secret key MSK^k, that the query was made for, was corrupted during the execution (respectively).

3. There do not exist two sequences $((\overrightarrow{\mathsf{PK}}_x, \overrightarrow{x}^0), (\overrightarrow{\mathsf{PK}}_y, \overrightarrow{y}^0)) \neq ((\overrightarrow{\mathsf{PK}}_x, \overrightarrow{x}^1), (\overrightarrow{\mathsf{PK}}_y, \overrightarrow{y}^1))$ and aggregation functions $\mathsf{Agg}_x, \mathsf{Agg}_y$ such that:

$$\mathcal{U}\left(\mathsf{Agg}_x\left((\mathsf{PK}_{x,i}, x^0_i)_i\right), \mathsf{Agg}_y\left((\mathsf{PK}_{y,j}, y^0_j)_j\right)\right) \neq \mathcal{U}\left(\mathsf{Agg}_x\left((\mathsf{PK}_{x,i}, x^1_i)_i\right), \mathsf{Agg}_y\left((\mathsf{PK}_{y,j}, y^1_j)_j\right)\right)$$

   and i) x^b_i was queried for aggregation function Agg_x, index i and public key $\mathsf{PK}_{x,i}$, and ii) y^b_j was queried for aggregation function Agg_y, index j and public key $\mathsf{PK}_{y,j}$, and iii) at least one of inputs $\{x^b_i\}, \{y^b_j\}$ were queried and public key $\mathsf{PK}_{x,i}, \mathsf{PK}_{y,j}$ was not corrupted. Note that if some x^b_i or y^b_j was not queried by the adversary, then it can generate partial keys or ciphertexts for any value of its choice by performing a fresh key generation since this is a fully dynamic system, however that samples a fresh public as well.

An MPFE scheme $(\mathsf{GSetup}, \mathsf{LSetup}, \mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt})$ is said to be IND secure if for any *admissible* PPT adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, the following holds

$$\Pr\left[\mathcal{A}^{\mathsf{HonestGen}(),\ \mathsf{Corrupt}(),\ \mathsf{Key}^\beta(),\ \mathsf{Enc}^\beta()}(1^\lambda) = \beta : \beta \leftarrow \{0,1\}\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda).$$

Remark 3.2 (Potential variations)**.** The above multi-party function encryption system that we define allows the users to dynamically join the system in the permissionless model, where each incoming user only needs to know the public parameters and not interact with any authority. A slightly weaker setting could be a permissioned model in which users can still dynamically join the system but they need to contact the global authority (which sampled the public parameters) either for some identification tokens or its encryption and master secret key pair in order to prevent totally unrestricted computation which happens in the permissionless model.

Also, we want to point out that in our current framework we let the users select the aggregation functions during individual functional key and ciphertext generation to allow for more flexibility. This could be relaxed even further by letting the aggregation functions be either be described in a uniform computation model, or using an ensemble of non-uniform functions. Also, one could instead restrict the flexibility in aggregation by asking each user to choose their aggregation functions at setup time. Such flexibilities will be important in capturing the notion of DDFE described in Section 4.

4 Capturing Existing Primitives as MPFE

We now show that our model is general enough to capture several existing notions in the literature as special cases. To simplify notation, we assume that every input is associated with an index which indicates its position in the set of arguments to the universal circuit $\mathcal{U}$. Here, we assume that the input to $\mathcal{U}$ which represents a circuit that must be emulated, is always associated with index 1 in either the first argument or the second argument.

Multi-Input Functional Encryption (MIFE)**:** A Multi-Input Functional Encryption scheme, denoted by MIFE [GGG+14], can be viewed as a special case of MPFE with a *centralized* setup. In an MIFE scheme, the setup algorithm outputs n encryption keys $\mathsf{EK}_1, \ldots, \mathsf{EK}_n$ and a single MSK. Each encryptor $i \in [n]$, chooses its message z_i and computes a ciphertext of z_i using EK_i. Here, the entire z_i is private. The key generator chooses a function f and computes a corresponding secret key SK_f using its MSK. Here, f is public.

In our notation, MIFE is captured by setting $(n_x, n_y) = (n, 1)$, $y = (f, \perp)$, $x_i = (x_{\text{pub},i}, x_{\text{pri},i}) = (\perp, z_i)$. We also set $\mathsf{Agg}_x(x_1, \ldots, x_n) = (x_{\text{pri},1}, \ldots, x_{\text{pri},n_x}) = (z_1, \ldots, z_n)$. Finally, Agg_y outputs $y_{\text{pub}} = f$ and upon decryption this yields $\mathcal{U}((z_1, \ldots, z_n), f) = f(z_1, \ldots, z_n)$ as desired.

Multi-Client Functional Encryption (MCFE)**:** A Multi-Client Functional Encryption scheme, denoted by MCFE [GKL+13, GGG+14, CDSG+18b] can be viewed as a special case of MPFE with a *centralized* setup. In an MCFE scheme, the setup algorithm outputs n encryption keys $\mathsf{EK}_1, \ldots, \mathsf{EK}_n$ and a single MSK. Each encryptor $i \in [n]$, chooses its message z_i and *a public label* lab_i and computes a ciphertext for (lab_i, z_i) using EK_i. Here, lab_i is public but z_i is private. The key generator chooses a function f and computes a corresponding secret key SK_f using its MSK. Here, f is public.

In our notation, set $(n_x, n_y) = (n, 1)$, $y = (f, \perp)$, $x_i = (x_{\text{pub},i}, x_{\text{pri},i}) = (\mathsf{lab}_i, z_i)$. The function $\mathsf{Agg}_x(x_1, \ldots, x_n)$ checks that $x_{\text{pub},1} = \ldots = x_{\text{pub},n}$. If this verification passes, it outputs $(x_{\text{pri},1}, \ldots, x_{\text{pri},n_x}) = (z_1, \ldots, z_n)$. Finally, Agg_y outputs $y_{\text{pub}} = f$ and upon decryption this yields $\mathcal{U}((z_1, \ldots, z_n), f) = f(z_1, \ldots, z_n)$ as desired.

Distributed Multi-Client Functional Encryption (D-MCFE)**:** The decentralized version of MCFE, denoted by D-MCFE [CDSG+18a], can be viewed as a special case of MPFE with an *interactive* setup. This primitive removes the need for a centralized authority by shifting the task of generating functional secret keys to the clients themselves. In D-MCFE, during the setup phase, the users generate public parameters and their individual master secret keys by running an interactive protocol. No further interaction is needed among clients for subsequent generation of functional keys. When an evaluator wishes to obtain a functional key for some function f, it requests each user for a partial decryption key corresponding to f. Ciphertexts for inputs (lab_i, z_i) are generated by each user independently as in MCFE. The decryptor, upon receiving ciphertexts and partial decryption keys from all the parties can compute $f(z_1, \ldots, z_n)$ as in MCFE, so long as all the labels match.

In our notation, set $(n_x, n_y) = (n, n)$ and $y_i = (f, \perp)$, $x_i = (x_{\text{pub},i}, x_{\text{pri},i}) = (\mathsf{lab}_i, z_i), \forall i \in [n]$. As in MCFE, $\mathsf{Agg}_x(x_1, \ldots, x_n)$ checks that $x_{\text{pub},1} = \ldots = x_{\text{pub},n}$. If this verification passes, it outputs $(x_{\text{pri},1}, \ldots, x_{\text{pri},n_x}) = (z_1, \ldots, z_n)$. Next, Agg_y first checks that $y_{\text{pub},i} = f, \forall i \in [n]$, and if so, outputs the corresponding value $y_{\text{pub},1} = f$. Upon decryption, this yields $f(z_1, \ldots, z_n)$ as in MCFE. In this primitive, $\mathsf{MSK}_i = \mathsf{EK}_i$ for all $i \in [n]$.

Multi-Authority Functional or Attribute-Based Encryption (MAFE **or** MA-ABE)**:** A Multi-Authority Functional Encryption, denoted by MAFE (or its special case Multi-Authority Attribute Based Encryption, MA-ABE) [Cha07, LW11, BCG+17] can be viewed as an instance of MPFE with a *local* setup. In MAFE, there are n key authorities who may independently generate their private and public keys, without even being aware of the existence of other authorities. MAFE is a ciphertext-policy scheme, so an encryptor encrypts a message z along with a policy f over the various authorities. Here, z is private but f is public. Any authority i (say), should be able to generate a token for a user with a global identifier GID for attributes lab_i. A user with tokens for attribute-indentifier pair $(\mathsf{lab}_i, \mathsf{GID}_i$ from authority $i \in [n]$, should be able to decrypt the ciphertext to recover $f(z, \mathsf{lab}_1, \ldots, \mathsf{lab}_n)$ as long as all the n identifiers are the same.

In our notation, we set $(n_x, n_y) = (1, n)$, $x = (f, z)$, $y_i = (y_{\text{pub},i}, y_{\text{pri},i}) = ((\mathsf{lab}_i, \mathsf{GID}_i), \perp)$. The function Agg_x simply outputs the pair (f, z). The aggregation function Agg_y checks that $\mathsf{GID}_1 = \ldots = \mathsf{GID}_n$. If this verification passes, it outputs $(\mathsf{lab}_1, \ldots, \mathsf{lab}_n)$. Decryption computes $\mathcal{U}(f, z, (\mathsf{lab}_1, \ldots, \mathsf{lab}_n)) = f(z, \mathsf{lab}_1, \ldots, \mathsf{lab}_n)$ as desired.

Decentralized Policy-Hiding Attribute Based Encryption (DABE)**:** A decentralized policy-hiding ABE, denoted by DABE [MJ18] proposed by Michalevsky and Joye can also be captured by an MPFE scheme with local setup. In a DABE scheme, there are n key authorities, each of which run a local setup to generate their private and public keys. They do so without communicating with each other, although they are required to obtain some global parameters of the system (such as group generators and such), which are generated by

a one-time trusted setup. This is a ciphertext-policy scheme, in which an encryptor can compute a ciphertext under a general access structure, while the corresponding secret keys are issued by independent authorities as in MAFE. However, in this system, the access policy embedded in the ciphertext is hidden.

In our notation, we set $(n_x, n_y) = (1, n)$, $x = (\perp, (f, z))$, $y_i = (\mathsf{lab}_i, \perp)$. The aggregation function Agg_y outputs $(\mathsf{lab}_1, \ldots, \mathsf{lab}_n)$. The aggregation function Agg_x returns (f', z) where $f'(z, \cdot) = (f, z)$ iff $f(\mathsf{lab}_1, \ldots, \mathsf{lab}_n) = 1$ and $\perp$ otherwise. The decryptor computes $\mathcal{U}(f', z, \mathsf{lab}_1, \ldots, \mathsf{lab}_n) = f'(z, \mathsf{lab}_1, \ldots, \mathsf{lab}_n)$ which checks if $f(\mathsf{lab}_1, \ldots, \mathsf{lab}_n) = 1$ and outputs (f, z) if and only if this is the case, as desired.

Partially Hiding Functional/Predicate Encryption (PHFE and PHPE): Partially Hiding Functional/Predicate Encryption (PHFE or PHPE) [GVW12, GVW15] can be seen as a special case of MPFE with a centralized setup algorithm in the public-key setting. This is a single user scheme, in which the setup outputs a PK and MSK. The encryptor, given the public key PK chooses a public label lab and a private input z and computes a ciphertext for the pair (lab, z). The key generator, given the master secret key, computes a secret key for some function f, which is public. The decryptor computes $f(\mathsf{lab}, z)$.

This scheme can be expressed as an MPFE scheme with $(n_x, n_y) = (1, 1)$, $x = (\mathsf{lab}, z)$, $y = (f, \perp)$. The Agg_x and Agg_y functions are trivial in this setting, since it is a single user scheme, and simply output their inputs. The decryptor computes $\mathcal{U}(f, \mathsf{lab}, z) = f(\mathsf{lab}, z)$ as desired.

Ad Hoc Multi-Input Functional Encryption (aMIFE): The primitive of adhoc multi-input functional encryption (aMIFE) recently proposed by Agrawal et al. [ACF+20] can be instantiated as an MPFE scheme with local setup. Adhoc MIFE enables multiple parties to generate their own private master keys as well as corresponding public keys. To encrypt data, each party (say i) uses its private master key to compute a ciphertext for any message of its choice (say z_i), and to issue function specific keys, the party uses the same master key to compute a partial function key corresponding to some function (say f). These ciphertexts and partial keys are sent to the decryptor who can aggregate them and compute $f(z_1, \ldots, z_n)$ as long as all the partial decryption keys correspond to the same function f.

In our notation, we set $(n_x, n_y) = (n, n)$, $x_i = (\perp, z_i)$, $y_i = (f_i, \perp)$. The function Agg_x outputs $(z_1, \ldots, z_n)$, the function Agg_y checks that all $y_{\mathrm{pub},1} = f_i = \ldots = y_{\mathrm{pub},n} = f_n$ are equal to the same value f, which it outputs. The decryptor computes $\mathcal{U}(f, z_1, \ldots, z_n) = f(z_1, \ldots, z_n)$ as desired.

Dynamic Decentralized Functional Encryption (DDFE): A dynamic decentralized FE scheme, denoted by DDFE [CDSG+20], can be viewed as a special case of MPFE with a local setup in the dynamic setting. In a DDFE scheme for functionality F and empty key ϵ, the setup outputs the public parameters PP, and each party samples its own public and secret key pair PK, SK. Each encryptor on input a public key PK (and the secret key SK if the DDFE scheme is secret key) and a message m, outputs a ciphertext $\mathsf{CT}_{\mathsf{PK}}$. The key generator chooses a key space object k and computes a corresponding decryption key $\mathsf{DK}_{\mathsf{PK},k}$ using a user master secret key SK. Here, key space object k is public. The decryption algorithm takes as input a list of decryption keys $(\mathsf{DK}_{\mathsf{PK}_j,k_j})_j$ and ciphertexts $(\mathsf{CT}_{\mathsf{PK}'_i})_i$, and outputs $F((\mathsf{PK}_j, k_j)_j, (\mathsf{PK}'_i, m_i)_i)$. Also, the system enables learning the empty function on a possibly non-singleton list of ciphertexts as: $F(\epsilon, (\mathsf{PK}'_i, m_i)_i)$.

To formally capture DDFE, we require the generalized dynamic MPFE, which is discussed in Section 3.1, which allows to specify (n_x, n_y) as well as $(\mathsf{Agg}_x, \mathsf{Agg}_y)$ dynamically. To set the remaining parameters in our notation, we let $y_j = ((F, k_j), \perp)$, $x_i = (\perp, m_i)$. The function Agg_x outputs $((\mathsf{PK}'_1, m_1), \ldots, (\mathsf{PK}'_{n_x}, m_{n_x}))$ for some $n_x \in \mathbb{N}$, and Agg_y checks that $F_1 = \cdots = F_{n_y} = F$ is the same in all y_j and outputs $(F, (\mathsf{PK}_1, k_1), \ldots, (\mathsf{PK}_{n_y}, k_{n_y}))$ if so ($\perp$ otherwise) for some $n_y \in \mathbb{N}$. (Note that since the aggregator functions in dynamic setting also take as input the corresponding public keys, thus they have access to PK and PK′ to include as part of the output.) Upon decryption this yields $\mathcal{U}\big(((\mathsf{PK}'_1, m_1), \ldots, (\mathsf{PK}'_{n_x}, m_{n_x})), (F, (\mathsf{PK}_1, k_1), \ldots, (\mathsf{PK}_{n_y}, k_{n_y}))\big) = F((\mathsf{PK}_j, k_j)_j, (\mathsf{PK}'_i, m_i)_i)$ as desired. And, note that if we consider the aggregation functions to be described as TMs or an ensemble of non-uniform functions Remark 3.2, then the key generator and does not need to specify the user indices during the key

generation and encryption phase as desired in DDFE. Lastly, to enable learning the empty function ϵ, each user during their local setup also samples a secret key for input $y_j^{\epsilon} = ((F, \epsilon), \perp)$ and keeps it part of the public key and the ciphertext, and thus the decryption algorithm recovers the $\mathcal{U}\big(((\mathsf{PK}'_1, m_1), \ldots, (\mathsf{PK}'_{n_x}, m_{n_x})), (F, \epsilon, \ldots, \epsilon)\big)$ $= F(\epsilon, (\mathsf{PK}'_i, m_i)_i)$ as desired.

4.1 Feasibility of MPFE for General Circuits

In this section we discuss the feasibility of general MPFE. Since the framework of MPFE is very general and supports all existing constructions in the literature (that we are aware of), feasibility varies widely depending on the properties desired, such as whether the setup algorithm must be local, centralized or interactive, whether and encryption keys must be public or private and such others. Since the case of centralized setup has been most widely studied in the literature and it is evident that feasibility results for local and interactive setup also apply for a centralized setup, we focus on these below.

Local Setup. In multi-party FE schemes, local setup is highly desirable, since it overcomes the key escrow problem which is one of the main drawbacks of present day FE constructions. In this setting, each user i runs the setup algorithm independently, and obtains her own public key PK_i and master secret key MSK_i. No communication or co-ordination is required between the users. In the symmetric key setting, the user encrypts her input using her master key MSK_i, while in the public key setting, anyone may encrypt using the public key(s) PK_i.

Constructions of multi-party FE with local setup have been notoriously hard to build. For general circuits and in the symmetric key setting, the primitive of adhoc multi-input functional encryption (aMIFE) recently proposed by Agrawal et al. [ACF+20] comes close to a general feasibility result in our model. Although there are important differences between aMIFE and MPFE, we show in Appendix A, that the construction of aMIFE provided in [ACF+20] is general enough to provide a feasibility result for MPFE in the symmetric key setting, with local setup.

Interactive Setup. A standard MIFE scheme can be modified to support distributed keys by replacing the setup algorithm with an interactive protocol. This makes use of a function delayed, rerunnable two round MPC protocol similar to the construction of aMIFE provided by [ACF+20]. Note that the limitation of interactive setup is mitigated by the fact that it must only be run once. This transformation also works in the public key setting. Please see Appendix A for details.

5 Multi-Authority ABE ∘ IPFE for LSSS Access Structures

In this section, extend the construction of Abdalla et al. [ACGU20] (ACGU20) to the multiparty setting. As discussed in Section 1, we support $n_y = n$ for some fixed, polynomial n and 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 (MA-ABIPFE) scheme for linear secret sharing schemes (LSSS) access structures. We rely on simple asumptions over bilinear maps.

5.1 Specializing the MPFE Syntax

Since our framework of MPFE described in Section 3 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 λ, the setup algorithm outputs public parameters PP.

$\mathsf{LSetup}(\mathsf{PP}, 1^n, 1^\ell, i) \to (\mathsf{PK}, \mathsf{MSK})$. On input the public parameters PP, attribute length n, message space index ℓ, 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 MSK_j, global identifier 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 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 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 5.1 (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 $\mathsf{negl}(\cdot)$ 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 GID, an authority index $j \in [n] \setminus S^*$, and an attribute-key vector pair $(b, \mathbf{u})$, responds

[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 [KW19].

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_{GID} contains the attribute bits queries for GID[10].

5.2 Construction

Let 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 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 $\mathsf{PK} = (\mathsf{PP}, \left[\boldsymbol{\alpha}\right]_{T,1}, \left[\mathbf{w}\right]_1)$ and $\mathsf{MSK} = (\boldsymbol{\alpha}, \mathbf{w})$. (Here and throughout, note that $\left[\mathbf{w}\right]_1$ 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 hashes the 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 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 ρ 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:

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

[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.

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:

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

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.

5.3 Correctness and Security

In this section, we provide the proofs of correctness and security. Our security proof consists of a description of a sequence of hybrid games, and prove indistinguishability of successive games.

Correctness. The proof of correctness follows from the correctness of the LSSS scheme, and underlying algebraic structure. Below we briefly highlight the main points.

First, note that for an access policy $(\mathbf{A}, \rho)$, if an attribute vector $\mathbf{x} \in \{0,1\}^n$ satisfies it, then there exists an LSSS reconstruction vector $\mathbf{z}$ as defined in decryption that satisfies the property that $\mathbf{z}^\top \cdot \mathbf{A} = \mathbf{e}_1^\top = (1, 0, \ldots, 0)$. Now consider the partial secret keys associated with attribute vector $\mathbf{x}$ and key vector $\mathbf{u}$ for identifier GID, $\mathsf{SK}_{i,\mathsf{GID},x_i,\mathbf{u}}$ for $i \in [n]$. Observe that $\mathsf{SK}_{i,\mathsf{GID},x_i,\mathbf{u}}$ is empty whenever $x_i = 0$, otherwise it contains the group element $\big[\langle \boldsymbol{\alpha}_i, \mathbf{u}\rangle\big]_1 \cdot \big[\mu \cdot \langle \mathbf{w}_i, \mathbf{u}\rangle\big]$ where $\big[\mu\big] = H(\mathsf{GID})$.

Now let us look at the term $\big[M\big]_{T,1}$, the decryptor computes. It contains the following terms which can be simplified as follows–

$$\begin{aligned}
\langle C_0, \mathbf{u}\rangle &= \langle \big[\mathbf{s}_1 + \mathbf{v}\big]_{T,1}, \mathbf{u}\rangle = \big[\langle \mathbf{s}_1 + \mathbf{v}, \mathbf{u}\rangle\big]_{T,1}, \\
\mathbf{z}^\top \cdot C_1 \cdot \mathbf{u} &= \mathbf{z}^\top \cdot \big[\mathbf{A} \cdot \mathbf{S} + \boldsymbol{\Delta} \odot (\mathbf{r} \otimes \mathbf{1}^\top)\big]_{T,1} \cdot \mathbf{u} \\
&= \big[\mathbf{z}^\top \cdot \mathbf{A} \cdot \mathbf{S} \cdot \mathbf{u} + \mathbf{z}^\top \cdot \big(\boldsymbol{\Delta} \odot (\mathbf{r} \otimes \mathbf{1}^\top)\big) \cdot \mathbf{u}\big]_{T,1} \\
&= \big[\mathbf{e}_1^\top \cdot \mathbf{S} \cdot \mathbf{u} + \mathbf{z}^\top \cdot \big(\boldsymbol{\Delta} \odot (\mathbf{r} \otimes \mathbf{1}^\top)\big) \cdot \mathbf{u}\big]_{T,1} \\
&= \big[\mathbf{e}_1^\top \cdot \mathbf{S} \cdot \mathbf{u} + \mathbf{z}^\top \cdot ((\boldsymbol{\Delta} \cdot \mathbf{u}) \odot \mathbf{r})\,\big]_{T,1}
\end{aligned}$$

$$
\begin{aligned}
\mathbf{z}^\top \cdot e\left(K, C_2\right) &= \mathbf{z}^\top \cdot \begin{pmatrix} e\left(\mathsf{SK}_{\rho(1),\mathsf{GID},x_{\rho(1)},\mathbf{u}}, \left[r_1\right]_1\right) \\ \vdots \\ e\left(\mathsf{SK}_{\rho(m_1),\mathsf{GID},x_{\rho(m_1)},\mathbf{u}}, \left[r_{m_1}\right]_1\right) \end{pmatrix} \\
&= \mathbf{z}^\top \cdot \begin{pmatrix} e\left(\left[\langle \boldsymbol{\alpha}_{\rho(1)}, \mathbf{u}\rangle\right]_1 \cdot \left[\mu \cdot \langle \mathbf{w}_{\rho(1)}, \mathbf{u}\rangle\right], \left[r_1\right]_1\right) \\ \vdots \\ e\left(\left[\langle \boldsymbol{\alpha}_{\rho(m_1)}, \mathbf{u}\rangle\right]_1 \cdot \left[\mu \cdot \langle \mathbf{w}_{\rho(m_1)}, \mathbf{u}\rangle\right], \left[r_{m_1}\right]_1\right) \end{pmatrix} \\
&= \mathbf{z}^\top \cdot \left[\left(\boldsymbol{\Delta} \cdot \mathbf{u} + \mu \cdot \boldsymbol{\Gamma} \cdot \mathbf{u}\right) \odot \mathbf{r}\right]_{T,1} \\
&= \left[\mathbf{z}^\top \cdot \left(\left(\boldsymbol{\Delta} \cdot \mathbf{u}\right) \odot \mathbf{r}\right) + \mathbf{z}^\top \cdot \left(\left(\mu \cdot \boldsymbol{\Gamma} \cdot \mathbf{u}\right) \odot \mathbf{r}\right)\right]_{T,1}
\end{aligned}
$$

$$
\begin{aligned}
\mathbf{z}^\top \cdot C_3 \cdot \mathbf{u} &= \mathbf{z}^\top \cdot \left[\mathbf{A} \cdot \mathbf{T} + \boldsymbol{\Gamma} \odot \left(\mathbf{r} \otimes \mathbf{1}^\top\right)\right]_1 \cdot \mathbf{u} \\
&= \left[\mathbf{e}_1^\top \cdot \mathbf{T} \cdot \mathbf{u} + \mathbf{z}^\top \cdot \left(\left(\boldsymbol{\Gamma} \cdot \mathbf{u}\right) \odot \mathbf{r}\right)\right]_1 \\
e\left(H(\mathsf{GID}), \mathbf{z}^\top \cdot C_3 \cdot \mathbf{u}\right) &= \left[\mu \cdot \mathbf{e}_1^\top \cdot \mathbf{T} \cdot \mathbf{u} + \mu \cdot \mathbf{z}^\top \cdot \left(\left(\boldsymbol{\Gamma} \cdot \mathbf{u}\right) \odot \mathbf{r}\right)\right]_{T,1}
\end{aligned}
$$

Combining all these together, we get that

$$
\begin{aligned}
\left[M\right]_{T,1} &= \frac{\left[\langle \mathbf{s}_1 + \mathbf{v}, \mathbf{u}\rangle\right]_{T,1}}{\left[\mathbf{e}_1^\top \cdot \mathbf{S} \cdot \mathbf{u} + \mathbf{z}^\top \cdot \left(\left(\boldsymbol{\Delta} \cdot \mathbf{u}\right) \odot \mathbf{r}\right)\right]_{T,1}} \cdot \frac{\left[\mathbf{z}^\top \cdot \left(\left(\boldsymbol{\Delta} \cdot \mathbf{u}\right) \odot \mathbf{r}\right) + \mathbf{z}^\top \cdot \left(\left(\mu \cdot \boldsymbol{\Gamma} \cdot \mathbf{u}\right) \odot \mathbf{r}\right)\right]_{T,1}}{\left[\mu \cdot \mathbf{e}_1^\top \cdot \mathbf{T} \cdot \mathbf{u} + \mu \cdot \mathbf{z}^\top \cdot \left(\left(\boldsymbol{\Gamma} \cdot \mathbf{u}\right) \odot \mathbf{r}\right)\right]_{T,1}} \\
&= \left[\langle \mathbf{s}_1 + \mathbf{v}, \mathbf{u}\rangle - \mathbf{e}_1^\top \cdot \mathbf{S} \cdot \mathbf{u} - \mu \cdot \mathbf{e}_1^\top \cdot \mathbf{T} \cdot \mathbf{u}\right]_{T,1} \\
&= \left[\langle \mathbf{v}, \mathbf{u}\rangle\right]_{T,1}
\end{aligned}
$$

where the last equality follows from the facts that $\mathbf{e}_1^\top \cdot \mathbf{S} = \mathbf{s}_1^\top$, and $\mathbf{e}_1^\top \cdot \mathbf{T} = \mathbf{0}^\top$. Therefore, whenever the policy $(\mathbf{A}, \rho)$ is satisfied, then $\left[M\right]_{T,1}$ gets simplified to $\left[\mathbf{u}^\top \mathbf{v}\right]_{T,1}$ which can be recovered by computing the discrete log. Thus, correctness follows.

Security. Below we provide a proof of security under standard assumptions over composite-order bilinear groups.

Theorem 5.2. If the assumptions 2.4, 2.5, 2.6, and 2.7 hold over the bilinear group generator Gen, then the scheme described above is a selectively secure MA-AB-IPFE for LSSS access structures as per Definition 5.1.

As discussed in Section 1.3, in addition of the type of key queries allowed under MA-ABE, we need to answer key queries for attribute-key vector pairs $(j, \mathsf{GID}, b, \mathbf{u})$ such that $\langle \mathbf{u}, \mathbf{v}_0 - \mathbf{v}_1\rangle = 0$. At a high level, our idea is to sample the authority secret keys $\boldsymbol{\alpha}_i, \mathbf{w}_i$ such that the reduction could honestly compute the secret keys for all attribute-key vector pairs $(j, \mathsf{GID}, b, \mathbf{u})$ such that $\mathbf{A}$ accepts the list of attribute bits queried for a given identifier GID and $\langle \mathbf{u}, \mathbf{v}_0\rangle = \langle \mathbf{u}, \mathbf{v}_1\rangle$. On the other hand, to answer key queries for attribute-key vector pairs $(j, \mathsf{GID}, b, \mathbf{u})$ such that $\mathbf{A}$ does *not* accept the list of attribute bits queried for a given identifier GID, we switch such secret keys to their semi-functional counterparts.

Although this intuition seems to work at a high level, it does not work as described above. To that end, we switch all the accepting secret keys to their semi-functional counterparts as well, but ensure that the challenge ciphertext components that they interact with are only nominally semi-functional. Here the last part 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. We get around this barrier by ensuring that the challenge ciphertext is sampled as what we call a "partial semi-functional ciphertext" (which has full-nominally semi-functional components along with standard semi-functional components).

Proof. We start by sketching the sequence of games where the first game corresponds to the original MA-AB-IPFE security game. First, we switch the way we sample the honest authority's public-secret keys wherein instead of sampling the random vectors $\boldsymbol{\alpha}_i$ and $\mathbf{w}_i$ for authority i directly, we sample two random vectors $\widetilde{\boldsymbol{\alpha}}_i$ and $\widetilde{\mathbf{w}}_i$ of same dimensions and implicitly set $\boldsymbol{\alpha}_i^\top = \widetilde{\boldsymbol{\alpha}}_i^\top \mathbf{F}$ and $\mathbf{w}_i^\top = \widetilde{\mathbf{w}}^\top \mathbf{F}$ where $\mathbf{F} \in \mathbb{Z}_N^{\ell \times \ell}$ is a random full rank matrix such that $\mathbf{F}(\mathbf{v}_0 - \mathbf{v}_1) = \mathbf{e}_1$ where $\mathbf{e}_1$ is the first vector in the canonical basis of $\mathbb{Z}_N^\ell$.

Next, we switch the way the random oracle queries are answered wherein we map identities GID to random elements of $\mathbb{G}_{p_1}$ instead of $\mathbb{G}$. This is followed by switching the challenge ciphertext to come from a special ciphertext distribution which we refer to as a *partial* semi-functional ciphertext, where the intuition is that only the projection of the ℓ-dimension message vector onto a one-dimensional subspace is encrypted under a semi-functional mode while the remaining $(\ell - 1)$-dimensional projection are encrypted under the full-nominally semi-functional encryption mode. Basically, we split the challenge message vector such that now we encrypt the vector $\beta(\mathbf{v}_0 - \mathbf{v}_1)$ as a semi-functional ciphertext (for random challenge bit β) and vector $\mathbf{v}_1$ as a full-nominally semi-functional ciphertext and homomorphically combines both of them to create the final challenge ciphertext. Finally, we switch the partial secret keys one-by-one to be semi-functional irrespective of whether $\mathbf{A}$ accept the list of attribute bits queried for a given identifier GID or not. An important property of the full-nominally semi-functional ciphertexts is that they can be decrypted by all types of semi-functional keys, while having the capability to be jointly sampled with a semi-functional ciphertext with correlated randomness.

The idea here is that since the adversary never receives any secret key for attribute-key vector pairs $(j, \mathsf{GID}, b, \mathbf{u})$ where both $\mathbf{A}$ accepts the list of attribute bits queried for a given identifier GID and $\langle \mathbf{u}, \mathbf{v}_0 - \mathbf{v}_1 \rangle \neq 0$, thus the reduction algorithm can perfectly simulate the above games using the dual system encryption paradigm [Wat09] as used in [LW11]. We want to point out that as in [LW11], we will assume a one-use restriction on the attributes throughout the proof, that is the row labeling function ρ of the challenge ciphertext is injective.

Description of games

Game 0**.** This corresponds to the original MA-AB-IPFE security game. Let Q denote the total number of key queries made by adversary.

Game 1**.** This is same as the previous game, except for each non-corrupt authority, i.e. $i \notin S^*$, the challenger:

— samples a uniformly random vectors $\widetilde{\boldsymbol{\alpha}}_i, \widetilde{\mathbf{w}}_i \leftarrow \mathbb{Z}_N^\ell$,

— samples a random *orthogonal* matrix $\mathbf{F} \in \mathbb{Z}_N^{\ell \times \ell}$ subject to the constraint that $\mathbf{F}(\mathbf{v}_0 - \mathbf{v}_1) = \mathbf{e}_1$, where $\mathbf{v}_0, \mathbf{v}_1$ are the challenge message vectors and $\mathbf{e}_1 = (1, 0, \ldots, 0)^\top$ (i.e., the first canonical basis vector of $\mathbb{Z}_N^\ell$), and

— sets $\boldsymbol{\alpha}_i = \mathbf{F}^\top \widetilde{\boldsymbol{\alpha}}_i, \mathbf{w}_i = \mathbf{F}^\top \widetilde{\mathbf{w}}_i$ instead of sampling it uniformly at random.

Re-writing the keys and challenge ciphertext. With the above change in sampling the parameters, we below re-write how all the keys and ciphertexts in the scheme can be rewrittten.

$$\mathsf{PK}_i = (\mathsf{PP}, \mathbf{F}^\top \cdot \left[\widetilde{\boldsymbol{\alpha}}_i\right]_{T,1}, \mathbf{F}^\top \cdot \left[\widetilde{\mathbf{w}}_i\right]_1), \quad \mathsf{SK}_{i,\mathsf{GID},1,\mathbf{u}} = \left[\langle \widetilde{\boldsymbol{\alpha}}_i, \mathbf{F} \cdot \mathbf{u} \rangle\right]_1 \cdot \left[\mu \cdot \langle \widetilde{\mathbf{w}}_i, \mathbf{F} \cdot \mathbf{u} \rangle\right].$$

$$\widetilde{\mathbf{S}} \leftarrow \mathbb{Z}_N^{m_2 \times \ell}, \qquad \widetilde{\mathbf{T}} = \begin{pmatrix} \mathbf{0}^\top \\ \widetilde{\mathbf{T}}' \leftarrow \mathbb{Z}_N^{(m_2-1)\times \ell} \end{pmatrix}, \qquad \widetilde{\boldsymbol{\Delta}} = \begin{pmatrix} \widetilde{\boldsymbol{\alpha}}_{\rho(1)}^\top \\ \vdots \\ \widetilde{\boldsymbol{\alpha}}_{\rho(m_1)}^\top \end{pmatrix}, \qquad \widetilde{\boldsymbol{\Gamma}} = \begin{pmatrix} \widetilde{\mathbf{w}}_{\rho(1)}^\top \\ \vdots \\ \widetilde{\mathbf{w}}_{\rho(m_1)}^\top \end{pmatrix}.$$

$$\mathbf{S} = \widetilde{\mathbf{S}} \cdot \mathbf{F}, \qquad \mathbf{T} = \widetilde{\mathbf{T}} \cdot \mathbf{F}, \qquad \boldsymbol{\Delta} = \widetilde{\boldsymbol{\Delta}} \cdot \mathbf{F}, \qquad \boldsymbol{\Gamma} = \widetilde{\boldsymbol{\Gamma}} \cdot \mathbf{F}.$$

$$
\begin{aligned}
C_0 &= \left[\mathbf{F}^\top \widetilde{\mathbf{S}}^\top \mathbf{e}_1 + \beta(\mathbf{v}_1 - \mathbf{v}_0) + \mathbf{v}_0\right]_{T,1}, \quad C_1 = \left[\mathbf{A}\cdot\widetilde{\mathbf{S}}\cdot\mathbf{F} + \widetilde{\boldsymbol{\Delta}}\cdot\mathbf{F}\odot(\mathbf{r}\otimes\mathbf{1}^\top)\right]_{T,1},\\
&= \mathbf{F}^\top\cdot\left[\widetilde{\mathbf{S}}^\top\mathbf{e}_1 - \beta\mathbf{e}_1 + \mathbf{F}\cdot\mathbf{v}_0\right]_{T,1}, \qquad = \left[\mathbf{A}\cdot\widetilde{\mathbf{S}} + \widetilde{\boldsymbol{\Delta}}\odot(\mathbf{r}\otimes\mathbf{1}^\top)\right]_{T,1}\cdot\mathbf{F},\\
C_2 &= \left[\mathbf{r}\right]_1, \qquad\qquad\qquad\qquad C_3 = \left[\mathbf{A}\cdot\widetilde{\mathbf{T}}\cdot\mathbf{F} + \widetilde{\boldsymbol{\Gamma}}\cdot\mathbf{F}\odot(\mathbf{r}\otimes\mathbf{1}^\top)\right]_1,\\
&\qquad\qquad\qquad\qquad\qquad\quad = \left[\mathbf{A}\cdot\widetilde{\mathbf{T}} + \widetilde{\boldsymbol{\Gamma}}\odot(\mathbf{r}\otimes\mathbf{1}^\top)\right]_1\cdot\mathbf{F},
\end{aligned}
$$

The above re-writing of the keys and ciphertext will be crucial later in the proof when dividing the type of queries and moving the ciphertext to *partial* semi-functional.

Game 2. This is same as the previous game, except the challenger answers the random oracle queries as follows:

— for fresh random oracle query, it samples a random vector $\mu_{\mathsf{GID}} \leftarrow \mathbb{Z}_N$ and responding with $\left[\mu_{\mathsf{GID}}\right]_1 = g_1^{\mu_{\mathsf{GID}}}$. Therefore, the partial secret keys can be simplified as:

$$\mathsf{SK}_{i,\mathsf{GID},1,\mathbf{u}} = \left[\langle\widetilde{\boldsymbol{\alpha}}_i + \mu_{\mathsf{GID}}\cdot\widetilde{\mathbf{w}}_i, \mathbf{F}\cdot\mathbf{u}\rangle\right]_1.$$

Defining partial semi-functional ciphertexts and secret keys. Similar to [LW11], we define the space of semi-functional ciphertexts and keys for our scheme. The main differences are that rather than turning the entire ciphertext to a semi-functional ciphertext, we only switch one dimensional subspace out of the ℓ dimensions of the ciphertext to be semi-functional and remaining $\ell - 1$ dimensional projection of the ciphertext space are turned full-nominally semi-functional, and these components are then algebraically manipulated given the vector space transformation matrix $\mathbf{F}$ by performing group operations on the encodings. Looking ahead, a full-nominally semi-functional ciphertext is more general than a nominally semi-functional ciphertext in that the latter can not be decrypted by one type of semi-functional keys whereas a full-nominally semi-functional ciphertext can be decrypted by all types of semi-functional keys. The difference between a regular and a full-nominally semi-functional ciphertext is only in that a regular ciphertext has no blinding factors, whereas a full-nominally semi-functional ciphertext has blinding factors in all remaining subgroups but they are structured such that decryption is unaffected. At a high level, the reason for defining a full-nominally semi-functional ciphertext is that algebraical manipulation of a semi-functional ciphertext and regular ciphertext is not possible due to lack of blinding factors in a regular ciphertext. Therefore, introducing structured blinding factors in a regular ciphertext make it decryptable with respect to all semi-functional keys but they also contain blinding factors thereby making algebraic manipulation possible.

For the remainder of the proof and descriptions of all types of semi-functional ciphertexts and keys, we use the following notation. Let $\mathsf{bad} \subset [m_1]$ denote the set of rows for which the corresponding attributes belong to corrupted authorities. That is, $\mathsf{bad} = \{i \in [m_1] : \rho(i) \in S^*\}$. Also, let $\mathbb{I}^* \in \{0,1\}^{m_1}$ be the vector such that i-th component $\mathbb{I}^*_i = 0$ if $i \in \mathsf{bad}$, otherwise $\mathbb{I}^*_i = 1$.

Partial semi-functional ciphertexts. The C_0 and C_1 components are sampled exactly as before. Here we switch how the C_2 and C_3 components are sampled. The procedure is described in detail in Fig. 5.1.

Semi-functional keys. When we refer to the key for identity GID, we regard as the set of all keys $\mathsf{SK}_{i,\mathsf{GID},1,\mathbf{u}}$ for the attributes i belonging to non-corrupt authorities requested by the attacker throughout the game. Recall that the adversary is also allowed to make key queries for accepting attributes for a fixed GID given that $\langle\mathbf{u}, \mathbf{v}_0 - \mathbf{v}_1\rangle = 0$. Now we sample all the secret keys as semi-functional keys irrespective of whether $\langle\mathbf{u}, \mathbf{v}_0 - \mathbf{v}_1\rangle = 0$ or not. One might think that this will be problematic since a semi-functional key for an accepting attribute will no longer decrypt the semi-functional ciphertext from above. However, this is precisely avoided by making the ciphertext to be full-nominally semi-functional, and ensuring that the accepting secret keys only interact with the nominally semi-functional component of the ciphertext. Recall that regular keys are sampled as

$$H(\mathsf{GID}) = \left[\mu_{\mathsf{GID}}\right]_1, \quad \mathsf{SK}_{i,\mathsf{GID},1,\mathbf{u}} = \left[\langle\widetilde{\boldsymbol{\alpha}}_i, \mathbf{F}\cdot\mathbf{u}\rangle\right]_1 \cdot \left[\langle\mu_{\mathsf{GID}}\cdot\widetilde{\mathbf{w}}_i, \mathbf{F}\cdot\mathbf{u}\rangle\right]_1. \tag{5.2}$$

Figure 5.1: Sampling partial semi-functional ciphertexts.

The challenger samples the following ciphertext matrices

$$\widetilde{C}_2 = \left[\mathbf{r}\right]_1, \quad \widetilde{C}_3 = \left[\mathbf{A}\cdot\widetilde{\mathbf{T}} + \widetilde{\mathbf{\Gamma}} \odot (\mathbf{r}\otimes\mathbf{1}^\top)\right]_1,$$

where $\mathbf{r}$ and $\widetilde{\mathbf{T}}$ are sampled as described previously. It then samples two random matrices $\mathbf{T}^*, \widehat{\mathbf{T}}^* \leftarrow \mathbb{Z}_N^{m_2\times\ell}$, and computes the semi-functional term of C_3 as

$$\widetilde{C}_3^* = \left[\mathbf{A}\cdot\mathbf{T}^*\right]_2 \odot \left[\mathbf{A}\cdot\widehat{\mathbf{T}}^*\right]_3 \odot \left[\widetilde{\mathbf{\Gamma}} \odot ((\mathbf{r}\odot\mathbb{I}^*)\otimes\mathbf{1}^\top)\right]_{\{2,3\}}$$

where encodings in subgroups 2 and 3 are computed using a fixed generator of these subgroups sampled at setup time. Finally, it sets the C_2 and C_3 components of the challenge ciphertext as

$$C_2 = \widetilde{C}_2 \odot \left[\mathbf{r}\odot\mathbb{I}^*\right]_{\{2,3\}}, \quad C_3 = \left(\widetilde{C}_3 \odot \widetilde{C}_3^*\right)\cdot\mathbf{F}. \tag{5.1}$$

We say the ciphertext is full-nominally semi-functional for column $j \in [\ell]$ if the first row of matrices $\mathbf{T}^*$ and $\widehat{\mathbf{T}}^*$ has 0 at the j-th index (that is, $T^*_{1,j} = \widehat{T}^*_{1,j} = 0$). In other words, this means that the secret being shared in column j is 0.

And, we say that the ciphertext is full-nominally semi-functional it is full-nominally semi-functional for all but the first column. That is, the first rows of matrices $\mathbf{T}^*$ and $\widehat{\mathbf{T}}^*$ is the first basis vector in the canonical basis of $\mathbb{Z}_N^\ell$.

The important aspect of these keys are that they *do not* depend on the first component of vectors $\widetilde{\boldsymbol{\alpha}}_i$ and $\widetilde{\mathbf{w}}_i$, that is $\widetilde{\alpha}_{i,1}$ and $\widetilde{w}_{i,1}$ are not needed for generating $\mathsf{SK}_{i,\mathsf{GID},1,\mathbf{u}}$ whenever the key corresponds to accepting attributes. This is because whenever $\langle\mathbf{u}, \mathbf{v}_0 - \mathbf{v}_1\rangle = 0$ (that is, the condition which must be true for accepting key queries), we get that $\mathbf{F}\cdot\mathbf{u}$ is orthogonal to the first basis vector in the canonical basis of $\mathbb{Z}_N^\ell$. That is, $\mathbf{F}\cdot\mathbf{u}$ is of the form $(0, *, \ldots, *)^\top$.

Now we sample the secret keys one of two ways. We refer to these as type 1 and 2 semi-functional keys as in [LW11]. A type-1 secret key contains blinding factors in the second subgroup, whereas type-2 secret key contains blinding factors in third subgroup such that they do not a nominally semi-functional ciphertext component does not get affected by type-1 keys.

A type-1 semi-functional secret key is computed as follows

$$H(\mathsf{GID}) = \left[\mu_{\mathsf{GID}}\right]_{\{1,2\}}, \quad \mathsf{SK}_{i,\mathsf{GID},1,\mathbf{u}} = \left[\langle\widetilde{\boldsymbol{\alpha}}_i, \mathbf{F}\cdot\mathbf{u}\rangle\right]_1 \cdot \left[\langle\mu_{\mathsf{GID}}\cdot\widetilde{\mathbf{w}}_i, \mathbf{F}\cdot\mathbf{u}\rangle\right]_{\{1,2\}}. \tag{5.3}$$

Similarly, a type-2 semi-functional secret key is computed as follows

$$H(\mathsf{GID}) = \left[\mu_{\mathsf{GID}}\right]_{\{1,3\}}, \quad \mathsf{SK}_{i,\mathsf{GID},1,\mathbf{u}} = \left[\langle\widetilde{\boldsymbol{\alpha}}_i, \mathbf{F}\cdot\mathbf{u}\rangle\right]_1 \cdot \left[\langle\mu_{\mathsf{GID}}\cdot\widetilde{\mathbf{w}}_i, \mathbf{F}\cdot\mathbf{u}\rangle\right]_{\{1,3\}}. \tag{5.4}$$

Game 3. This is the same as the previous game, except that the challenge ciphertext is partial semi-functional as described in Fig. 5.1. That is, the challenger samples the first two components of the challenge ciphertext C_0 and C_1 as before that is

$$C_0 = \mathbf{F}^\top\cdot\left[\widetilde{\mathbf{S}}^\top\mathbf{e}_1 - \beta\mathbf{e}_1 + \mathbf{F}\cdot\mathbf{v}_0\right]_{T,1}, \quad C_1 = \left[\mathbf{A}\cdot\widetilde{\mathbf{S}} + \widetilde{\mathbf{\Delta}} \odot (\mathbf{r}\otimes\mathbf{1}^\top)\right]_{T,1}\cdot\mathbf{F}$$

where β is the random challenge bit. Whereas the last two components C_2 and C_3 are sampled as in Eq. (5.1).

Note. The above Game 3 is same as Game 4.0.2 that we describe below.

Game $4.q.1$**.** This is same as Game 3, except the challenger answers the key queries as follows:

— for the first $q-1$ key queries, on attribute-key vector pairs $(j, \mathsf{GID}, b, \mathbf{u})$, it computes the partial key as a semi-functional key of 'type-2' as in Eq. (5.4).

— for the q-th key query, on attribute-key vector pair $(j, \mathsf{GID}, b, \mathbf{u})$, it computes the partial key as a semi-functional key of 'type-1' as in Eq. (5.3).

— remaining key queries are answered exactly as in Game 3. That is, it computes the partial key honestly as in Eq. (5.2).

Game $4.q.2$**.** This is same as Game $4.q.1$, except the challenger answers the key queries as follows:

— for the q-th key query, on attribute-key vector pair $(j, \mathsf{GID}, b, \mathbf{u})$, it computes the partial key as a semi-functional key of 'type-2' as in Eq. (5.4). (That is, the q-th key is switched from type 1 to type 2 as well. All keys in this game are either a type-2 semi-functional key, or honestly sampled keys. Also, note that all the accepting partial keys do not depend on the first component of the key vectors $\widetilde{\boldsymbol{\alpha}}_i$ and $\widetilde{\mathbf{w}}_i$.)

Game 5**.** This is the same as Game $4.Q.2$, except that the challenge ciphertext is a partial semi-functional ciphertext where the C_0 component encrypts a random vector along the direction $\mathbf{v}_1 - \mathbf{v}_0$. That is,

$$C_0 = \mathbf{F}^\top \cdot \left[\widetilde{\mathbf{S}}^\top \mathbf{e}_1 - \kappa \mathbf{e}'_1 + \mathbf{F} \cdot \mathbf{v}_0\right]_{T,1}$$

where κ is a random exponent, and $\mathbf{e}_1, \mathbf{e}'_1$ are first basis vectors in the canonical basis of $\mathbb{Z}_N^{m_2}$ and $\mathbb{Z}_N^{\ell}$, respectively.

Note that in the last hybrid game, the challenger's responses are independent of the challenge bit β, thus to complete the proof we only need to show that each adjacent game is indistinguishable.

Indistinguishability of games

To complete the proof we need to show that adjacent games are indistinguishable. The high level strategy behind the reductions is similar to [LW11] which is to rely on appropriate type of subgroup hiding assumption coupled with the one-use restriction on the attributes so that information-theoretic security of the secret sharing scheme could be used to move from nominally semi-functional ciphertexts to semi-functional ciphertexts. However, due to the fact that we can not make all subspaces of the challenge ciphertext semi-functional, thus we turn only the first subspace to be semi-functional while remaining subspaces are made nominally semi-functional. The precise reductions are similar to those provided in [LW11], thus we only provide a high level sketch below.

For any adversary $\mathcal{A}$ and game X, we denote by $\mathsf{Adv}_s^{\mathcal{A}}(\lambda)$, the probability that $\mathcal{A}$ wins in game s.

Lemma 5.3. For any PPT adversary $\mathcal{A}$, we have that $\mathsf{Adv}_0^{\mathcal{A}}(\lambda) - \mathsf{Adv}_1^{\mathcal{A}}(\lambda) \leq \mathsf{negl}(\lambda)$ for some negligible function $\mathsf{negl}(\cdot)$.

Proof. This follows directly from the fact that, for any invertible matrix $\mathbf{F} \in \mathbb{Z}_N^{\ell \times \ell}$, the following distributions are identical.

$$\left\{\{\boldsymbol{\alpha}_i, \mathbf{w}_i\}_{i\in[n]} \ : \ \forall i \in [n], \boldsymbol{\alpha}_i, \mathbf{w}_i \leftarrow \mathbb{Z}_N^{\ell}\right\} \equiv \left\{\{\mathbf{F}^\top \cdot \widetilde{\boldsymbol{\alpha}}_i, \mathbf{F}^\top \cdot \widetilde{\mathbf{w}}_i\}_{i\in[n]} \ : \ \forall i \in [n], \widetilde{\boldsymbol{\alpha}}_i, \widetilde{\mathbf{w}}_i \leftarrow \mathbb{Z}_N^{\ell}\right\}.$$

Since N is a composite with large factors, thus with all but negligible probability $\mathbf{F}$ will be invertible as otherwise we could factor N thereby breaking security of the composite-order bilinear maps. And, since $\mathbf{F}$ is invertible with all but negligible probability, thus the lemma follows. □

Lemma 5.4. If assumption 2.4 holds over the group generator Gen, then for any PPT adversary $\mathcal{A}$, we have that $\mathsf{Adv}_1^{\mathcal{A}}(\lambda) - \mathsf{Adv}_2^{\mathcal{A}}(\lambda) \leq \mathsf{negl}(\lambda)$ for some negligible function $\mathsf{negl}(\cdot)$.

Proof. The proof of this lemma is similar to that of [LW11, Lemma 7]. That is, the reduction algorithm simply uses the challenge group element T, which lies either in $\mathbb{G}$ or $\mathbb{G}_1$, to answer all the random oracle queries (made directly or indirectly via the partial key queries). If $T \in \mathbb{G}$, then the reduction simulates Game 1, otherwise it simulates Game 2. Thus, the lemma follows. □

Lemma 5.5. If assumption 2.4 holds over the group generator Gen, then for any PPT adversary $\mathcal{A}$, we have that $\mathsf{Adv}_2^{\mathcal{A}}(\lambda) - \mathsf{Adv}_3^{\mathcal{A}}(\lambda) \leq \mathsf{negl}(\lambda)$ for some negligible function $\mathsf{negl}(\cdot)$.

Proof. The proof of this lemma is similar to that of [LW11, Lemma 8], except now the reduction algorithm needs to generate a much larger ciphertext (ℓ [LW11]-like ciphertexts but non-trivially correlated). Ideally, if the correlations between the ℓ sub-ciphertexts were mild, and could be solved using reusability of correlated components then the proof would be more straightforward. But since one of these ℓ sub-ciphertexts must not be decryptable by the queried secret keys, while all other ciphertexts must be, thus the correlations are significant.

However, despite the technical challenges, subgroup decision assumption is still sufficient for our purposes. At a high level, the reduction algorithm simply uses the challenge group element T, which lies either in $\mathbb{G}$ or $\mathbb{G}_1$, to sample the C_2 and C_3 components of the ciphertext corresponding to rows which are non-corrupt, while for corrupt rows, only C_3 depends on the challenge T. This portion of reduction can be carried out similar to [LW11, Lemma 8] by sampling the vectors and matrices of appropriate dimensions directly over $\mathbb{Z}_N$ and implicitly setting certain exponents such as $\mathbf{r}$. The main difference appears in the step that unlike [LW11, Lemma 8], where the authors directly argued semi-functionality of the ciphertext due to the fact that all keys are rejecting, we can not rely on the same strategy as we allow the adversary to make accepting queries as well as long as $\langle \mathbf{u}, \mathbf{v}_0 - \mathbf{v}_1 \rangle = 0$. But we note that by sampling the subspace transformation matrix $\mathbf{F}$ and applying homomorphically on top of the challenge ciphertext, we could pinpoint a single subspace (which is along the first canonical basis vector in our case) where we switch the ciphertext to be semi-functional from the adversary's perspective, while all other sub-ciphertext components (that is, in the subspace orthogonal to the first canonical basis vector) are nominally semi-functional from the adversary's perspective as well. In a little more detail, the information-theoretic security argument for arguing that the secret shares are well distributed is only applied for the first subspace, but not the rest. □

Lemma 5.6. If assumption 2.5 holds over the group generator Gen, then for any PPT adversary $\mathcal{A}$ and $q \in \{1, \ldots, Q\}$, we have that $\mathsf{Adv}_{4.(q-1).2}^{\mathcal{A}}(\lambda) - \mathsf{Adv}_{4.q.1}^{\mathcal{A}}(\lambda) \leq \mathsf{negl}(\lambda)$ for some negligible function $\mathsf{negl}(\cdot)$.

Proof. The proof of this lemma is similar to that of [LW11, Lemma 9] but with changes similar to that for Lemma 5.5. Here again we rely on the fact that neither the reduction or the attacker can distinguish the type-1 semi-functional key from a regular key. This is because for all but the first subspace the ciphertext is still nominally semi-functional both from the reduction and attacker's perspective; whereas the first subspace appears like a semi-functional ciphertext to $\mathcal{A}$ but for the challenger it is still nominally semi-functional. Now as in the previous lemma, we only apply the information-theoretic secret sharing guarantee in the first sub-ciphertext component. □

Lemma 5.7. If assumption 2.6 holds over the group generator Gen, then for any PPT adversary $\mathcal{A}$ and $q \in \{1, \ldots, Q\}$, we have that $\mathsf{Adv}_{4.q.1}^{\mathcal{A}}(\lambda) - \mathsf{Adv}_{4.q.2}^{\mathcal{A}}(\lambda) \leq \mathsf{negl}(\lambda)$ for some negligible function $\mathsf{negl}(\cdot)$.

Proof. The proof of this lemma is identical to that of [LW11, Lemma 10]. □

Lemma 5.8. If assumption 2.7 holds over the group generator Gen, then for any PPT adversary $\mathcal{A}$, we have that $\mathsf{Adv}_{4.Q.2}^{\mathcal{A}}(\lambda) - \mathsf{Adv}_5^{\mathcal{A}}(\lambda) \leq \mathsf{negl}(\lambda)$ for some negligible function $\mathsf{negl}(\cdot)$.

Proof. The proof of this lemma is similar to that of [LW11, Lemma 11], except now the strategy is to switch from encryption of bit β to a random element in the first subspace of the challenge ciphertext, while keeping the remaining sub-ciphertext portions to be identically distributed. Since the reduction algorithm receives the generators for all subgroups from the challenger, thus it could simulate the distribution of nominally semi-functional ciphertext components honestly. And , it uses the remaining components of the bilinear challenge to simulate the first sub-ciphertext component by using the strategy similar to in [LW11, Lemma 11]. This completes the proof. □

□

6 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 PK_i and a master secret key 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 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 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 Agg_y checks that all values $\mathcal{U}_K$ and 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.

6.1 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^λ and a number 1^n of slots, and outputs a public parameter PK, encryption keys $\{\mathsf{EK}_i\}_{i \in [n]}$, a master secret key MSK. The other algorithms implicitly take PK.

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

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

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

Correctness. An 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 MCFE in [ABKW19] and extend it to function-hiding security. We also introduce a selective vatiant because our final goal is IP-DDFE with selective security, and selectively secure IP-MCFE is sufficient for the security analysis of our IP-DDFE scheme.

Definition 6.1 (Function-hiding security of MCFE)**.** An MCFE scheme is 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 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 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 QCor after querying 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 $i \in \mathcal{HS}$.

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 [CDSG+20], since it allows more precise indexing than the indexing by PK[12]. We assume that the correspondence between id i and public key 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^λ and outputs a public parameter PP. The other algorithms implicitly take PP.

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

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

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

[11] The leakage function captures information that EK_i reveals from SK.

[12] In [CDSG+20], 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 [CDSG+20, § 7.2].

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 [CDSG+20]. However, we do not consider the case here since it induces ambiguity that can be also found in [CDSG+20][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.

Security. We naturally extend the security definition for DDFE in [CDSG+20] to the function-hiding setting as follows.

Definition 6.2 (Function-hiding security of DDFE)**.** An DDFE scheme is xx-yy-function-hiding (xx $\in$ {sel, adt}, yy $\in$ {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

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

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 PK_i. For i such that $\mathsf{QHonestGen}(i)$ was queried, the adversary can make the following queries: $\mathsf{QCor}(i)$ outputs 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^β and m^β 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 6.3 (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 Π 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 as per Section A.1.2.

[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.

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

Definition 6.4 (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 $f(\mathbf{x}_1, ..., \mathbf{x}_n) = \langle (\mathbf{x}_1 || ... || \mathbf{x}_n), \mathbf{y} \rangle$.

Definition 6.5 (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 $\{m_i = (\mathbf{x}_i, \mathcal{U}_{M,i}, \mathsf{lab}_{M,i})\}_{i \in \mathcal{U}'_M}$,

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

- $\mathcal{U}'_K = \mathcal{U}'_M$, 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 6.6 (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 6.2 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, lab_K).

Definition 6.7 (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 $\{m_i = (x_i, \mathcal{U}_{M,i}, \mathsf{lab}_{M,i})\}_{i \in \mathcal{U}'_M}$,

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

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

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 [CDSG+20][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 Section 2.3.

6.2 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. Let $\mathsf{iFE} = (\mathsf{iSetup}, \mathsf{iKeyGen}, \mathsf{iEnc}, \mathsf{iDec})$ be a function-hiding IPFE scheme (recall that iKeyGen, iEnc take a group-element vector as input instead of a $\mathbb{Z}_p$-element vector (see Definition 6.3)) 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 N is provided in Figure 6.1.

[15] In AoNE, there are no secret keys and thus the IND-security defined in [CDSG+20] is exactly the same as function-hiding security in our paper.

$\mathsf{Setup}(1^\lambda, 1^n)$: On input the security parameter 1^λ, 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 MSK, and vectors $\{\mathbf{y}_i\}_{i\in[n]}$ and outputs 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 EK_i, an input vector $\mathbf{x}_i$, a label lab and outputs 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 SK, ciphertexts $\mathsf{CT}_1, ..., \mathsf{CT}_n$ and outputs d as follows.

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

Figure 6.1: 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, EK_i has a power to decode both CT_i and SK_i since EK_i is a part of MSK. This is captured as the function Leak_y below.

Theorem 6.8. If the SXDH assumption holds in $\mathbb{G}_1$ and iFE is function-hiding, then our IP-MCFE scheme is Leak_y-sel-pos-function-hiding in the random oracle model, where $\mathsf{Leak}_y(i, \{\mathbf{y}_i\}_{i\in[n]}) = \mathbf{y}_i$.

Proof. Let $\mathcal{HS} \subseteq [n]$ be the set of uncorrupted parties and $\mathcal{CS} = [n] \backslash \mathcal{HS}$. Let $\mathcal{LS} \subset \mathcal{L}$ be a set of labels queried by the adversary. Let q_k be the maximum number of key generation queries. We prove the theorem via a series of hybrids, which are defined as follows. Basically we denote a variable v used in the ℓ-th encryption query for label lab in slot i by $v^\ell_{\mathsf{lab},i}$ but often omit ℓ and lab when we uniformly handle the variable in all queries. We use similar omission for key generation query.

G^β : The original game. Especially, the challenger sets $\widehat{\mathbf{x}}_i = (\mathbf{x}^\beta_i, 0^\mathsf{N}, t_\mathsf{lab}, 0)$, $\widehat{\mathbf{y}}_i = (\mathbf{y}^\beta_i, 0^\mathsf{N}, r_i, 0)$ for the reply to the encryption query and the key generation query, respectively. Note that when the adversary or the encryption oracle evaluate H, it queries random oracle to obtain its function values.

H^β_0 : This hybrid is the same as G^β except that the challenger sets $\widehat{\mathbf{x}}_i = (\mathbf{x}^\beta_i, \mathbf{x}^0_i, t_\mathsf{lab}, 0)$ for the reply to $\mathsf{QEnc}(i, \mathbf{x}^0_i, \mathbf{x}^1_i, \mathsf{lab})$ with $i \in \mathcal{HS}$. The indistinguishability between G^β and H^β_0 directly follows from the message privacy of iFE, since $\langle \widehat{\mathbf{x}}_i, \widehat{\mathbf{y}}_i \rangle$ in G^β and that in H^β_0 are the same for all queries of $i \in \mathcal{HS}$.

$\mathsf{H}^\beta_j (j \in [q_k])$: In this hybrid, the challenger changes $\widehat{\mathbf{y}}_i$ in the reply to $\mathsf{QKeyGen}(\{\mathbf{y}^0_i\}, \{\mathbf{y}^1_i\})$ from H^β_0. That is, for $i \in \mathcal{HS}$, it defines $\widehat{\mathbf{y}}_i = (0^\mathsf{N}, \mathbf{y}^0_i, r_i, 0)$ for the first j queries and $\widehat{\mathbf{y}}_i = (\mathbf{y}^\beta_i, 0^\mathsf{N}, r_i, 0)$ for the last $q_k - j$ queries. The indistinguishability between H^β_{j-1} and H^β_j for $j \in [q_k]$ is proven in Lemma 6.9.

H_f^β : This hybrid is the same as $\mathsf{H}_{q_k}^\beta$ except that the challenger set $\widehat{\mathbf{x}}_i = (0^\mathsf{N}, \mathbf{x}_i^0, t_\mathsf{lab}, 0)$ instead of $(\mathbf{x}_i^\beta, \mathbf{x}_i^0, t_\mathsf{lab}, 0)$ for all encryption queries with $i \in \mathcal{HS}$. The indistinguishability between $\mathsf{H}_{q_k}^\beta$ and H_f^β directly follows from the message privacy of iFE, since $\langle \widehat{\mathbf{x}}_i, \widehat{\mathbf{y}}_i \rangle$ in $\mathsf{H}_{q_k}^\beta$ and that in H_f^β are the same for all queries with $i \in \mathcal{HS}$. In this hybrid, the adversary's advantage is 0 since its view is independent of β (recall that $x_i^0 = x_i^1$ and $y_i^0 = y_i^1$ for all $i \in \mathcal{CS}$).

□

Lemma 6.9. If the SXDH assumption holds in $\mathbb{G}_1$ and iFE is function-hiding, then H_{j-1}^β and H_j^β are indistinguishable for $j \in [q_k]$.

Proof. We introduce intermediate hybrids to prove the lemma as follows.

$\mathsf{H}_{j-1,1}^\beta$: This hybrids is the same as H_{j-1}^β except that, for $i \in \mathcal{HS}$, the challenger sets

$$\widehat{\mathbf{x}}_i = (\mathbf{x}_i^\beta, \mathbf{x}_i^0, t_\mathsf{lab}, \underline{r_i^j t_\mathsf{lab} + \langle \mathbf{x}_i^\beta, \mathbf{y}_i^{j,\beta} \rangle}),\ \widehat{\mathbf{y}}_i = \begin{cases} (0^\mathsf{N}, \mathbf{y}_i^0, r_i, 0) & \text{(the first } j-1 \text{ queries)} \\ (\underline{0^\mathsf{N}, 0^\mathsf{N}, 0, 1}) & \text{(the } j\text{-th query)} \\ (\mathbf{y}_i^\beta, 0^\mathsf{N}, r_i, 0) & \text{(the last } q_k - j \text{ queries)} \end{cases}$$

where r_i^j are the random elements chosen in the j-th key generation query. The indistinguishability between H_{j-1}^β and $\mathsf{H}_{j-1,1}^\beta$ directly follows from the security of the function-hiding property of iFE, since $\langle \widehat{\mathbf{x}}_i, \widehat{\mathbf{y}}_i \rangle$ in H_{j-1}^β and that in $\mathsf{H}_{j-1,1}^\beta$ are the same for all queries with $i \in \mathcal{HS}$.

$\mathsf{H}_{j-1,2}^\beta$: This hybrids is the same as $\mathsf{H}_{j-1,1}^\beta$ except that, for all $\mathsf{lab} \in \mathcal{LS}$, and $i \in \mathcal{HS}$, the challenger first randomly choose $\widehat{t}_{\mathsf{lab},i} \in \mathbb{Z}_p$ so that $\sum_{i \in \mathcal{HS}} \widehat{t}_{\mathsf{lab},i} = -t_\mathsf{lab} \sum_{i \in \mathcal{CS}} r_i^j$. Then it sets, for $i \in \mathcal{HS}$,

$$\widehat{\mathbf{x}}_i = (\mathbf{x}_i^\beta, \mathbf{x}_i^0, t_\mathsf{lab}, \underline{\widehat{t}_{\mathsf{lab},i}} + \langle \mathbf{x}_i^\beta, \mathbf{y}_i^{j,\beta} \rangle).$$

The indistinguishability between $\mathsf{H}_{j-1,1}^\beta$ and $\mathsf{H}_{j-1,2}^\beta$ can be proven under SXDH as follows. The random self-reducibility of SXDH implies $[\{a_\mu, a_\mu \mathbf{b}\}_{\mu \in [\ell]}]_1 \approx_c [\{a_\mu, \mathbf{u}_\mu\}_{\mu \in [\ell]}]_1$ where $a_\mu \leftarrow \mathbb{Z}_p, \mathbf{b} = (b_1, ..., b_h) \leftarrow \mathbb{Z}_p^h, \mathbf{u}_\mu = (u_{\mu,1}, ..., u_{\mu,h}) \leftarrow \mathbb{Z}_p^h$, $\ell = |\mathcal{LS}|$ and $h = |\mathcal{HS}| - 1$. Without loss of generality, we can assume $\mathcal{HS} = \{1, ..., h+1\}$ and $\mathcal{CS} = \{h+2, ..., n\}$. We implicitly define $t_\mathsf{lab} = a_{\mathsf{L}(\mathsf{lab})}$, $r_i^j = b_i$ (for $i \in [h]$), $r_{h+1}^j = -\sum_{i \in [h]} b_i - \sum_{i \in \mathcal{CS}} r_i^j$, $\widehat{t}_{\mathsf{lab},i} = u_{\mathsf{L}(\mathsf{lab}),i}$, where $\mathsf{L} : \mathcal{LS} \to [\ell]$ is some bijective function. Then, we can construct a reduction that chooses $r_i^j \leftarrow \mathbb{Z}_p$ for $i \in \mathcal{CS}$ and simulates vectors in queries as

$$\widehat{\mathbf{x}}_i = \begin{cases} (\mathbf{x}_i^\beta, \mathbf{x}_i^0, a_{\mathsf{L}(\mathsf{lab})}, z_{\beta,\mathsf{L}(\mathsf{lab}),i} + \langle \mathbf{x}_i^\beta, \mathbf{y}_i^{j,\beta} \rangle) & (i \in [h]) \\ (\mathbf{x}_i^\beta, \mathbf{x}_i^0, a_{\mathsf{L}(\mathsf{lab})}, -\sum_{i' \in [h]} z_{\beta,\mathsf{L}(\mathsf{lab}),i'} - a_{\mathsf{L}(\mathsf{lab})} \sum_{i' \in \mathcal{CS}} r_{i'}^j + \langle \mathbf{x}_i^\beta, \mathbf{y}_i^{j,\beta} \rangle) & (i = h+1) \\ (\mathbf{x}_i^\beta, 0, a_{\mathsf{L}(\mathsf{lab})}, 0) & (i \in \mathcal{CS}) \end{cases}$$

$$\widehat{\mathbf{y}}_i = \begin{cases} (0^\mathsf{N}, \mathbf{y}_i^0, r_i, 0) & \text{(the first } j-1 \text{ queries for } i \in \mathcal{HS}) \\ (0^\mathsf{N}, 0^\mathsf{N}, 0, 1) & \text{(the } j\text{-th query for } i \in \mathcal{HS}) \\ (\mathbf{y}_i^\beta, 0^\mathsf{N}, r_i^j, 0) & \text{(the } j\text{-th query for } i \in \mathcal{CS}) \\ (\mathbf{y}_i^\beta, 0^\mathsf{N}, r_i, 0) & \text{(the last } q_k - j \text{ queries for } i \in \mathcal{HS} \text{ and all for } i \in \mathcal{CS}) \end{cases}$$

where $z_{0,\mu,i} = a_\mu b_i$ and $z_{1,\mu,i} = u_{\mu,i}$.

$\mathsf{H}_{j-1,3}^\beta$: This hybrids is the same as $\mathsf{H}_{j-1,2}^\beta$ except that, for all $i \in \mathcal{HS}$, the challenger sets $\widehat{\mathbf{x}}_i$ in encryption queries as

$$\widehat{\mathbf{x}}_i = (\mathbf{x}_i^\beta, \mathbf{x}_i^0, t_\mathsf{lab}, \widehat{t}_{\mathsf{lab},i} + \underline{\langle \mathbf{x}_i^0, \mathbf{y}_i^{j,0} \rangle}).$$

This change is information-theoretic, which can be proven as follows. Let $(i, \mathbf{x}^{\ell,0}_{\mathsf{lab},i}, \mathbf{x}^{\ell,1}_{\mathsf{lab},i}, \mathsf{lab})$ be the ℓ-th encryption query of the form of $(i, *, *, \mathsf{lab})$. Due to the query condition, we have

$$\Delta^{\beta}_{\mathsf{lab},i} = \mathbf{x}^{\ell,\beta}_{\mathsf{lab},i}\mathbf{y}^{j,\beta}_i - \mathbf{x}^{\ell,0}_{\mathsf{lab},i}\mathbf{y}^{j,0}_i = \mathbf{x}^{1,\beta}_{\mathsf{lab},i}\mathbf{y}^{j,\beta}_i - \mathbf{x}^{1,0}_{\mathsf{lab},i}\mathbf{y}^{j,0}_i \quad \text{for all } \ell.$$

Thus, we can define $\widehat{t}_{\mathsf{lab},i} = \widehat{t}'_{\mathsf{lab},i} - \Delta^{\beta}_{\mathsf{lab},i}$ and obtain

$$\widehat{\mathbf{x}}_i = (\mathbf{x}^{\beta}_i, \mathbf{x}^0_i, t_{\mathsf{lab}}, \widehat{t}_{\mathsf{lab},i} + \langle \mathbf{x}^{\ell,\beta}_{\mathsf{lab},i}, \mathbf{y}^{j,\beta}_i \rangle) = (\mathbf{x}^{\beta}_i, \mathbf{x}^0_i, t_{\mathsf{lab}}, \widehat{t}'_{\mathsf{lab},i} + \langle \mathbf{x}^{\ell,0}_{\mathsf{lab},i}, \mathbf{y}^{j,0}_i \rangle).$$

Observe that both $\{\widehat{t}_{\mathsf{lab},i}\}_{i\in\mathcal{HS}}$ and $\{\widehat{t}'_{\mathsf{lab},i}\}_{i\in\mathcal{HS}}$ are secret shares of $-\widehat{t}_{\mathsf{lab}}\sum_{i\in\mathcal{CS}} r^j_i$ since $\sum_{i\in\mathcal{HS}}\Delta^{\beta}_{\mathsf{lab},i} = 0$ due to the query condition. Hence, they are identically distributed. The indistinguishability between $\mathsf{H}^{\beta}_{j-1,3}$ and H^{β}_j can be proven similarly to that between H^{β}_{j-1} and $\mathsf{H}^{\beta}_{j-1,2}$.

□

6.3 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 β 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, $\{\mathsf{PRF}^{\mathsf{K}}_1\} : \mathcal{L} \to \mathbb{Z}_p$, $\{\mathsf{PRF}^{\mathsf{K}}_2\} : 2^{\mathcal{ID}} \to \{0,1\}^{p(\lambda,\mathsf{N})}$ 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 PRF_1 are the same, namely, $\mathcal{K}_1$. Our construction for vector length N is provided in Figure 6.2.

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

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

since $\mathsf{K}_{i,j,1} = \mathsf{K}_{j,i,1}$. For all $i \in \mathcal{U}$, iSK_i and iCT_i are generated under the same iMSK_i since they are generated using the same random tape $\mathsf{PRF}^{\mathsf{K}_{i,2}}_2(\mathcal{U})$. 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 6.10. If $\{\mathsf{PRF}^{\mathsf{K}}_1\}, \{\mathsf{PRF}^{\mathsf{K}}_2\}$ 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.

Proof. Let $\mathcal{S}$ be the set of parties generated by honest-party generation queries. Let $\mathcal{HS} \subseteq \mathcal{S}$ be the set of uncorrupted parties and $\mathcal{CS} = \mathcal{S}\backslash\mathcal{HS}$. Let $\mathcal{LS} \subset \mathcal{L}$ be a set of labels queried by the adversary. We prove the theorem via a series of hybrids, which are defined as follows.

G^{β} : The original game. Especially, the challenger sets $\widehat{\mathbf{x}}_i = (\mathbf{x}^{\beta}_i, 0^{\mathsf{N}}, t, 0)$, $\widehat{\mathbf{y}}_i = (\mathbf{y}^{\beta}_i, 0^{\mathsf{N}}, r_i, 0)$ for the reply to the encryption query and the key generation query, respectively (the procedures in eq. 6.1 and 6.3). Note that when the adversary or the encryption oracle evaluates H, it queries random oracle to obtain its function values.

$\mathsf{GSetup}(1^\lambda)$: On input the security parameter 1^λ, the setup algorithm outputs 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 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 MSK_i, and an input $k = (\mathbf{y}_i, \mathcal{U}_K, \mathsf{lab}_K)$ such that $i \in \mathcal{U}_K$ and outputs SK_i as follows.

$$\mathsf{rt}_i = \mathsf{PRF}_2^{\mathsf{K}_{i,2}}(\mathcal{U}_K),\ \mathsf{iMSK}_i = \mathsf{iSetup}(1^\lambda, 1^{2\mathsf{N}+2}; \mathsf{rt}_i),\ \mathsf{K}_{i,j,1} \leftarrow \mathsf{nSharedKey}(\mathsf{nSK}_i, \mathsf{nPK}_j)$$
$$r_i = \sum_{\substack{j \in \mathcal{U}_K \\ i \neq j}} (-1)^{j<i} \mathsf{PRF}_1^{\mathsf{K}_{i,j,1}}(\mathsf{lab}_K),\ \widehat{\mathbf{y}}_i = (\mathbf{y}_i, 0^\mathsf{N}, r_i, 0),\ \mathsf{iSK}_i \leftarrow \mathsf{iKeyGen}(\mathsf{iMSK}_i, \widehat{\mathbf{y}}_i) \tag{6.1}$$
$$\mathsf{aCT}_i \leftarrow \mathsf{aEnc}(\mathsf{aMSK}_i, (\mathsf{iSK}_i, \mathcal{U}_K, \mathsf{lab}_K)),\ \mathsf{SK}_i = (\mathsf{aCT}_i, \mathcal{U}_K, \mathsf{lab}_K). \tag{6.2}$$

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

$$\mathsf{rt}_i = \mathsf{PRF}_2^{\mathsf{K}_{i,2}}(\mathcal{U}_M),\ \mathsf{iMSK}_i = \mathsf{iSetup}(1^\lambda, 1^{2\mathsf{N}+2}; \mathsf{rt}_i),\ [t]_1 = H(\mathcal{U}_M, \mathsf{lab}_M)$$
$$\widehat{\mathbf{x}}_i = (\mathbf{x}_i, 0^\mathsf{N}, t, 0),\ \mathsf{iCT}_i \leftarrow \mathsf{iEnc}(\mathsf{iMSK}_i, [\widehat{\mathbf{x}}_i]_1) \tag{6.3}$$
$$\mathsf{aCT}_i \leftarrow \mathsf{aEnc}(\mathsf{aMSK}_i, (\mathsf{iCT}_i, \mathcal{U}_M, \mathsf{lab}_M)),\ \mathsf{CT}_i = (\mathsf{aCT}_i, \mathcal{U}_M, \mathsf{lab}_M). \tag{6.4}$$

$\mathsf{Dec}(\{\mathsf{SK}_i\}_{i \in \mathcal{U}_K}, \{\mathsf{CT}_i\}_{i \in \mathcal{U}_M})$: The decryption algorithm takes as input the public parameters 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

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

Figure 6.2: Function Hiding IP-DDFE

H_1^β : In this hybrid, the challenger uses random functions $R_{i,2}$ instead of $\mathsf{PRF}_2^{\mathsf{K}_{i,2}}$ for $i \in \mathcal{HS}$ in key generation and encryption queries. The indistinguishability directly follows from the security of the pseudorandom function.

H_2^β : We say a key generation query on $\widehat{k} = (i, \mathbf{y}_i^0, \mathbf{y}_i^1, \mathcal{U}_K, \mathsf{lab}_K)$ is incomplete if there exists $i' \in \mathcal{U}_K$ such that $i' \in \mathcal{HS}$ and the query of the form $\widehat{k} = (i', *, *, \mathcal{U}_K, \mathsf{lab}_K)$ is not made. In this hybrid, for all incomplete key generation queries, aCT_i is changed to the encryption of $(0, \mathcal{U}_K, \mathsf{lab}_K)$ (the procedure in eq. 6.2). The indistinguishability directly follows from the security of AoNE.

H_3^β : We say an encryption query on $\widehat{m} = (i, \mathbf{x}_i^0, \mathbf{x}_i^1, \mathcal{U}_M, \mathsf{lab}_M)$ is incomplete if there exists $i' \in \mathcal{U}_M$ such that $i' \in \mathcal{HS}$ and the query of the form $\widehat{m} = (i', *, *, \mathcal{U}_M, \mathsf{lab}_M)$ is not made. In this hybrid, for all incomplete encryption queries, aCT_i is changed to the encryption of $(0, \mathcal{U}_M, \mathsf{lab}_M)$ (the procedure in eq. 6.4). The indistinguishability directly follows from the security of AoNE.

H_f^β : In this hybrid, for all complete queries, the challenger sets $\widehat{\mathbf{x}}_i = (0^\mathsf{N}, \mathbf{x}_i^0, t, 0)$, $\widehat{\mathbf{y}}_i = (0^\mathsf{N}, \mathbf{y}_i^0, r_i, 0)$ for the reply to the encryption query and the key generation query on $i \in \mathcal{HS}$, respectively (the procedures in eq. 6.1 and 6.3). In this hybrid, the adversary's advantage is 0 since its view is independent of β (recall

that $x_i^0 = x_i^1$ and $y_i^0 = y_i^1$ for all $i \in \mathcal{CS}$). The indistinguishability from H_3^β is shown in Lemma 6.11.

□

Lemma 6.11. If $\{\mathsf{PRF}_1^\mathsf{K}\}$ are families of pseudorandom functions, the SXDH assumption holds in $\mathbb{G}_1$, iFE is function-hiding, and NIKE is IND-secure, then H_3^β and H_f^β are indistinguishable in the random oracle model.

Proof. We say $\mathcal{U} \in 2^{\mathcal{ID}}$ is a complete ID set if $\mathcal{U}$ is queried in either complete key generation queries or complete encryption query. Let q_u be the number of complete ID sets in the adversary's queries. We always put the complete ID sets in some order as $\mathcal{U}_1, ..., \mathcal{U}_{q_u}$, e.g., in ascending order with respect to the size of ID set. To prove the lemma, we introduce hybrids $\widehat{\mathsf{H}}_j^\beta$ for $j \in [q'_u]$ between H_3^β and H_f^β as follows, where q'_u is the upper bound of q_u.

$\widehat{\mathsf{H}}_j^\beta$ $(j \in [q'_u])$: This hybrid is the same as H_3^β except that, for complete queries of the form $\widehat{m} = (i, \mathbf{x}_i^0, \mathbf{x}_i^1, \mathcal{U}_M, \mathsf{lab}_M)$ and $\widehat{k} = (i, \mathbf{y}_i^0, \mathbf{y}_i^1, \mathcal{U}_K, \mathsf{lab}_K)$ such that $i \in \mathcal{HS}$, the challenger set

$$\widehat{\mathbf{x}}_i = \begin{cases} (\underline{0^\mathsf{N}, \mathbf{x}_i^0}, t, 0) & (\mathcal{U}_M \in \{\mathcal{U}_1, ..., \mathcal{U}_j\}) \\ (\underline{\mathbf{x}_i^\beta, 0^\mathsf{N}}, t, 0) & (\mathcal{U}_M \in \{\mathcal{U}_{j+1}, ..., \mathcal{U}_{q_u}\}) \end{cases}, \ \widehat{\mathbf{y}}_i = \begin{cases} (\underline{0^\mathsf{N}, \mathbf{y}_i^0}, r_i, 0) & (\mathcal{U}_K \in \{\mathcal{U}_1, ..., \mathcal{U}_j\}) \\ (\underline{\mathbf{y}_i^\beta, 0^\mathsf{N}}, r_i, 0) & (\mathcal{U}_K \in \{\mathcal{U}_{j+1}, ..., \mathcal{U}_{q_u}\}) \end{cases}$$

respectively, where $\mathcal{U}_j = \{\perp\}$ for $j > q_u$.

We can see that $\widehat{\mathsf{H}}_{q'_u}^\beta = \mathsf{H}_f^\beta$. Thus, what we need to do is to prove $\widehat{\mathsf{H}}_{j-1}^\beta$ and $\widehat{\mathsf{H}}_j^\beta$ are indistinguishable for $j \in [q'_u]$, where we define $\widehat{\mathsf{H}}_0^\beta = \mathsf{H}_3^\beta$.

For the purpose, we define intermediate hybrids $\widehat{\mathsf{H}}_{j-1,\kappa}^\beta$ for $\kappa \in [q_k]$ where q_k is the maximum number of key generation queries, and which is defined as follows.

$\widehat{\mathsf{H}}_{j-1,\kappa}^\beta$ $(\kappa \in [q_k])$: Let $\{\mathsf{lab}_{K,\mathcal{U}_K}^1, ..., \mathsf{lab}_{K,\mathcal{U}_K}^v\}$ denote a set of labels used in the complete key query of the form $(*, *, *, \mathcal{U}_K, *)$. This hybrid is the same as $\widehat{\mathsf{H}}_{j-1}^\beta$ except that, for complete queries of the form $\widehat{m} = (i, \mathbf{x}_i^0, \mathbf{x}_i^1, \mathcal{U}_M, \mathsf{lab}_M)$ and $\widehat{k} = (i, \mathbf{y}_i^0, \mathbf{y}_i^1, \mathcal{U}_K, \mathsf{lab}_{K,\mathcal{U}_K}^\ell)$ such that $i \in \mathcal{HS}$, the challenger set

$$\widehat{\mathbf{x}}_i = \begin{cases} (0^\mathsf{N}, \mathbf{x}_i^0, t, 0) & (\mathcal{U}_M \in \{\mathcal{U}_1, ..., \mathcal{U}_{j-1}\}) \\ (\underline{\mathbf{x}_i^\beta, \mathbf{x}_i^0}, t, 0) & (\mathcal{U}_M = \mathcal{U}_j) \\ (\underline{\mathbf{x}_i^\beta, 0^\mathsf{N}}, t, 0) & (\mathcal{U}_M \in \{\mathcal{U}_{j+1}, ..., \mathcal{U}_{q_u}\}) \end{cases}$$

$$\widehat{\mathbf{y}}_i = \begin{cases} (\underline{0^\mathsf{N}, \mathbf{y}_i^0}, r_i, 0) & (\mathcal{U}_K \in \{\mathcal{U}_1, ..., \mathcal{U}_{j-1}\} \text{ or } (\mathcal{U}_K = \mathcal{U}_j \text{ and } \ell \leq \kappa)) \\ (\underline{\mathbf{y}_i^\beta, 0^\mathsf{N}}, r_i, 0) & (\mathcal{U}_K \in \{\mathcal{U}_{j+1}, ..., \mathcal{U}_{q_u}\} \text{ or } (\mathcal{U}_K = \mathcal{U}_j \text{ and } \ell > \kappa)) \end{cases}$$

respectively, where $\mathcal{U}_j = \{\perp\}$ for $j > q_u$.

We can prove $\widehat{\mathsf{H}}_{j-1}^\beta \approx_c \widehat{\mathsf{H}}_{j-1,0}^\beta$, $\widehat{\mathsf{H}}_{j-1,\kappa-1}^\beta \approx_c \widehat{\mathsf{H}}_{j-1,\kappa}^\beta$, $\widehat{\mathsf{H}}_{j-1,q_k}^\beta \approx_c \widehat{\mathsf{H}}_j^\beta$ similarly to Lemma 6.9 where q_k is the maximum number of key generation queries. However, we need a more careful analysis to prove $\widehat{\mathsf{H}}_{j-1,\kappa-1}^\beta \approx_c \widehat{\mathsf{H}}_{j-1,\kappa}^\beta$ since we additionally utilize NIKE and PRF_1 to prove them following the strategy in Lemma 6.9.

In the following, we focus only on complete encryption queries of the form $(*, *, *, \mathcal{U}_j, *)$ and complete key generation queries of the form $(*, *, *, \mathcal{U}_j, \mathsf{lab}_{K,\mathcal{U}_j}^\kappa)$, since other queries are not changed between $\widehat{\mathsf{H}}_{j-1,\kappa-1}^\beta$ and $\widehat{\mathsf{H}}_{j-1,\kappa}^\beta$. Let $\mathcal{U}_j \cap \mathcal{HS} = \{u_1, ..., u_w\}$ and w' be an upper bound of w. We then define intermediate hybrid $\overline{\mathsf{H}}_\eta^\beta$ between $\widehat{\mathsf{H}}_{j-1,\kappa-1}^\beta$ and $\widehat{\mathsf{H}}_{j-1,\kappa}^\beta$.

$\overline{\mathsf{H}}_\eta^\beta$ $(\eta \in [w'])$: This hybrid is the same as $\widehat{\mathsf{H}}_{j-1,\kappa-1}^\beta$ except that, for complete queries of the form $\widehat{m} = (u_i, \mathbf{x}_{\mathsf{lab}_M, u_i}^{\ell,0}, \mathbf{x}_{\mathsf{lab}_M, u_i}^{\ell,1}, \mathcal{U}_j, \mathsf{lab}_M)$ and $\widehat{k} = (u_i, \mathbf{y}_{u_i}^0, \mathbf{y}_{u_i}^1, \mathcal{U}_j, \mathsf{lab}_{K,\mathcal{U}_j}^\kappa)$ such that $i \in \mathcal{U}_j \cap \mathcal{HS}$, the challenger

set

$$\widehat{\mathbf{x}}_{u_i} = \begin{cases} (\mathbf{x}^{\ell,\beta}_{\mathsf{lab}_M,u_i}, \mathbf{x}^{\ell,0}_{\mathsf{lab}_M,u_i}, t, 0) & (i < w) \\ (\mathbf{x}^{\ell,\beta}_{\mathsf{lab}_M,u_i}, \mathbf{x}^{\ell,0}_{\mathsf{lab}_M,u_i}, t, \underline{\sum_{i' \in [\eta]} \Delta^{\beta}_{\mathsf{lab}_M,u_{i'}}}) & (i = w) \end{cases}$$

$$\widehat{\mathbf{y}}_{u_i} = \begin{cases} (\underline{0^{\mathsf{N}}, \mathbf{y}^0_{u_i}}, r_{u_i}, 0) & (i \le \eta) \\ (\mathbf{y}^{\beta}_{u_i}, 0^{\mathsf{N}}, r_{u_i}, 0) & (\eta < i < w) \\ (\mathbf{y}^{\beta}_{u_i}, 0^{\mathsf{N}}, r_{u_i}, \underline{1}) & (i = w) \end{cases}$$

respectively, where $(\mathbf{x}^{\ell,0}_{\mathsf{lab}_M,u_i}, \mathbf{x}^{\ell,1}_{\mathsf{lab}_M,u_i})$ are the vectors in the ℓ-th encryption query of the form $(u_i, *, *, \mathcal{U}_j, \mathsf{lab}_M)$ and $\Delta^{\beta}_{\mathsf{lab}_M,u_i} = \mathbf{x}^{1,\beta}_{\mathsf{lab}_M,u_i} \mathbf{y}^{\beta}_{u_i} - \mathbf{x}^{1,0}_{\mathsf{lab}_M,u_i} \mathbf{y}^0_{u_i}$.

Due to the query condition of the adversary, we have

$$\Delta^{\beta}_{\mathsf{lab}_M,u_i} = \mathbf{x}^{\ell,\beta}_{\mathsf{lab}_M,u_i} \mathbf{y}^{\beta}_{u_i} - \mathbf{x}^{\ell,0}_{\mathsf{lab}_M,u_i} \mathbf{y}^0_{u_i} \text{ for all } \ell \tag{6.5}$$

$$\sum_{i \in [w']} \Delta^{\beta}_{\mathsf{lab}_M,u_i} = 0 \tag{6.6}$$

We can easily prove that $\widehat{\mathsf{H}}^{\beta}_{j-1,\kappa-1} \approx_c \overline{\mathsf{H}}^{\beta}_0$ and $\overline{\mathsf{H}}^{\beta}_{w'} \approx_c \widehat{\mathsf{H}}^{\beta}_{j-1,\kappa}$ by the function-hiding security of iFE and eq. 6.6. Thus the remaining task is to prove $\overline{\mathsf{H}}^{\beta}_{\eta-1} \approx_c \overline{\mathsf{H}}^{\beta}_{\eta}$ for $\eta \in [w']$. To prove this, we first change the way of choosing $\mathsf{K}_{u_\eta,u_w,1}$ as $\mathsf{K}_{u_\eta,u_w,1} \leftarrow \mathcal{K}_1$ instead of $\mathsf{K}_{u_\eta,u_w,1} \leftarrow \mathsf{nSharedKey}(\mathsf{nSK}_{u_\eta}, \mathsf{nPK}_{u_w})$, which directly follows from IND-security of NIKE. We next change

$$r_{u_\eta} = \sum_{\substack{i \in \mathcal{U}_j \cap \mathcal{HS} \\ i \neq u_\eta}} (-1)^{i < u_\eta} \mathsf{PRF}_1^{\mathsf{K}_{u_\eta,i,1}}(\mathsf{lab}^{\kappa}_{K,\mathcal{U}_j})$$

to

$$r_{u_\eta} = \sum_{\substack{i \in \mathcal{U}_j \cap \mathcal{HS} \\ i \notin \{u_\eta,u_w\}}} (-1)^{i < u_\eta} \mathsf{PRF}_1^{\mathsf{K}_{u_\eta,i,1}}(\mathsf{lab}^{\kappa}_{K,\mathcal{U}_j}) - s_{u_\eta,u_w}$$

and

$$r_{u_w} = \sum_{\substack{i \in \mathcal{U}_j \cap \mathcal{HS} \\ i \neq u_w}} (-1)^{i < u_w} \mathsf{PRF}_1^{\mathsf{K}_{u_w,i,1}}(\mathsf{lab}^{\kappa}_{K,\mathcal{U}_j})$$

to

$$r_{u_w} = \sum_{\substack{i \in \mathcal{U}_j \cap \mathcal{HS} \\ i \notin \{u_\eta,u_w\}}} (-1)^{i < u_w} \mathsf{PRF}_1^{\mathsf{K}_{u_w,i,1}}(\mathsf{lab}^{\kappa}_{K,\mathcal{U}_j}) + s_{u_\eta,u_w}$$

where $s_{u_\eta,u_w} \leftarrow \mathbb{Z}_p$. This indistinguishability directly follows from the security of pseudorandom function PRF_1. Then, we change encrypted vectors using the function-hiding security of iFE as follows:

$$\widehat{\mathbf{x}}_{u_i} = \begin{cases} (\mathbf{x}^{\ell,\beta}_{\mathsf{lab}_M,u_i}, \mathbf{x}^{\ell,0}_{\mathsf{lab}_M,u_i}, t, \underline{-t s_{u_\eta,u_w}}) & (i = \eta) \\ (\mathbf{x}^{\ell,\beta}_{\mathsf{lab}_M,u_i}, \mathbf{x}^{\ell,0}_{\mathsf{lab}_M,u_i}, t, \underline{\sum_{i' \in [\eta-1]} \Delta^{\beta}_{\mathsf{lab}_M,u_{i'}} + t s_{u_\eta,u_w}}) & (i = w) \end{cases}$$

$$\widehat{\mathbf{y}}_{u_i} = \begin{cases} (\mathbf{y}^{\beta}_{u_i}, 0^{\mathsf{N}}, \underline{r_{u_i} + s_{u_\eta,u_w}}, 1) & (i = \eta) \\ (\mathbf{y}^{\beta}_{u_i}, 0^{\mathsf{N}}, \underline{r_{u_i} - s_{u_\eta,u_w}}, 1) & (i = w) \end{cases}.$$

The remaining things are the similar to Lemma 6.9:

1. change $t s_{u_\eta,u_w}$ to a random element $\widehat{s}_{u_\eta,u_w}$ by the SXDH assumption;

2. implicitly define $\widehat{s}_{u_\eta,u_w} = \widehat{s}'_{u_\eta,u_w} + \Delta^{\beta}_{\mathsf{lab}_M,u_\eta}$;

3. change $\widehat{\mathbf{y}}_{u_\eta} = (\mathbf{y}^{\beta}_{u_\eta}, 0^{\mathsf{N}}, r_{u_\eta} + s_{u_\eta,u_w}, 1)$ to $\widehat{\mathbf{y}}_{u_\eta} = (0^{\mathsf{N}}, \mathbf{y}^{0}_{u_\eta}, r_{u_\eta} + s_{u_\eta,u_w}, 1)$ by using eq. 6.5 and function-hiding security of iFE;

4. go to $\overline{\mathsf{H}}^{\beta}_{\eta}$ by rewinding changes of vectors by function-hiding security, the SXDH assumption, PRF, and NIKE.

□

7 Distributed Ciphertext Policy ABE

In this section, we show how to distribute the recent construction of succinct ciphertext policy ABE by Agrawal and Yamada [AY20]. As discussed in Section 1, the setup algorithm is run in the Local mode and key generation is distributed amongst $n_y = n$ parties. As in [AY20], $n_x = 1$ (hence Agg_x is trivial) and $(x_{\mathrm{pub}}, x_{\mathrm{pri}}) = (C, m)$ where C is a circuit in NC_1 and m is a hidden bit. For key generation, the i^{th} party produces a key for $(y_{\mathrm{pub}}, y_{\mathrm{pri}}) = ((\mathbf{y}, \mathsf{GID}, y_i), \perp)$. The aggregation function Agg_y checks if all the values of GID and $\mathbf{y}$ are the same and all the attribute vector bits y_i are consistent with $\mathbf{y}$ and if so, it 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.

7.1 Specializing the MPFE Syntax

In this section, we define the notion of a distributed ciphertext-policy attribute-based encryption. First, we provide the syntax, and later describe the security definition.

Syntax. A distributed attribute-based encryption for predicate class $\mathcal{C} = \{\mathcal{C}_\ell : \{0,1\}^\ell \to \{0,1\}\}_{\ell \in \mathbb{N}}$ and 1-bit message space consists of the following PPT algorithms:

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

$\mathsf{LSetup}(\mathsf{PP}, 1^n, 1^\ell) \to (\mathsf{PK}, \mathsf{MSK})$. On input the public parameters PP, number of authorities n, and attribute length ℓ, the authority setup algorithm outputs a pair of master public-secret key $(\mathsf{PK}, \mathsf{MSK})$.

$\mathsf{KeyGen}(\mathsf{MSK}_i, \mathsf{GID}, \mathbf{x}) \to \mathsf{SK}_{i,\mathsf{GID},\mathbf{x}}$. The key generation algorithm takes as input an authority master secret key MSK_i, global identifier GID, and an attribute $\mathbf{x} \in \{0,1\}^\ell$. It outputs a partial secret key $\mathsf{SK}_{i,\mathsf{GID},\mathbf{x}}$.

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

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

We require such an ABE scheme to satisfy the following properties.

Correctness. A distributed ABE scheme is said to be correct if for all $\lambda, n, \ell \in \mathbb{N}$, $C \in \mathcal{C}_\ell$, $\mathbf{x} \in \{0,1\}^\ell$, $\mu \in \{0,1\}$, if $C(\mathbf{x}) = 1$ then the following holds:

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

Security. For security, we consider standard semantic security but in presence of corrupt authorities. Here we consider a much stronger adversary which can even choose the corrupt authorities' keys on its own.

Definition 7.1 (Distributed ABE security in presence of corrupt authorities with unknown corrupt keys)**.** A distributed ABE scheme is secure with corrupt authorities if for every stateful admissible PPT adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, the following holds

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

where the oracle $O(\mathsf{key}, \cdot, \cdot, \cdot)$ has key material hardwired, and on input a tuple of a global identifier GID, an authority index j, and an attribute $\mathbf{x}$, and responds with a partial secret key computed as $\mathsf{SK}_{j,\mathsf{GID},\mathbf{x}} \leftarrow \mathsf{KeyGen}(\mathsf{MSK}_j, \mathsf{GID}, \mathbf{x})$. 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 $C^*(\mathbf{x}) = 0$, or (2) adversary does not query at least one non-corrupt authority for the same global identifier, attribute tuple $(\mathsf{GID}, \mathbf{x})$.

Remark 7.2 (Comparing with decentralized ABE)**.** The notion of distributed ABE differs from the notion of decentralized ABE in the sense that in a distributed ABE scheme each authority controls essentially an almost identical secret share of the master key, whereas in a decentralized (or multi-authority) ABE scheme each authority only controls the master key component for the attributes under its control.

7.2 Preliminaries

In this section, we define some preliminaries that we require for our constructions. Some other relevant preliminaries are provided in Section 2.

7.2.1 Attribute Based Encryption

Let $R = \{R_\lambda : A_\lambda \times B_\lambda \to \{0,1\}\}_\lambda$ be a relation where A_λ and B_λ denote "ciphertext attribute" and "key attribute" spaces. An attribute-based encryption (ABE) scheme for R is defined by the following PPT algorithms:

$\mathsf{Setup}(1^\lambda) \to (\mathsf{PK}, \mathsf{MSK})$: The setup algorithm takes as input the unary representation of the security parameter λ and outputs a master public key PK and a master secret key MSK.

$\mathsf{Enc}(\mathsf{PK}, X, \mu) \to \mathsf{CT}$: The encryption algorithm takes as input a master public key PK, a ciphertext attribute $X \in A_\lambda$, and a message bit μ. It outputs a ciphertext CT.

$\mathsf{KeyGen}(\mathsf{PK}, \mathsf{MSK}, Y) \to \mathsf{SK}_Y$: The key generation algorithm takes as input the master public key PK, the master secret key MSK, and a key attribute $Y \in B_\lambda$. It outputs a private key SK_Y.

$\mathsf{Dec}(\mathsf{PK}, \mathsf{CT}, X, \mathsf{SK}_Y, Y) \to \mu$ or $\perp$: We assume that the decryption algorithm is deterministic. The decryption algorithm takes as input the master public key PK, a ciphertext CT, ciphertext attribute $X \in A_\lambda$, a private key SK_Y, and private key attribute $Y \in B_\lambda$. It outputs the message μ or $\perp$ which represents that the ciphertext is not in a valid form.

Definition 7.3 (Correctness)**.** An ABE scheme for relation family R is correct if for all $\lambda \in \mathbb{N}$, $X \in A_\lambda, Y \in B_\lambda$ such that $R(X,Y) = 1$, and for all messages $\mu \in \mathsf{msg}$,

$$\Pr\left[\begin{array}{l}(\mathsf{PK},\mathsf{MSK}) \leftarrow \mathsf{Setup}(1^\lambda),\\ \mathsf{SK}_Y \leftarrow \mathsf{KeyGen}(\mathsf{PK},\mathsf{MSK},Y),\\ \mathsf{CT} \leftarrow \mathsf{Enc}(\mathsf{PK},X,\mu):\\ \mathsf{Decrypt}\Big(\mathsf{PK},\mathsf{SK}_Y,Y,\mathsf{CT},X\Big) \neq \mu\end{array}\right] = \mathsf{negl}(\lambda)$$

where the probability is taken over the coins of Setup, KeyGen, and Enc.

Definition 7.4 ($\mathsf{Ada\text{-}IND}$ security for ABE)**.** For an ABE scheme $\mathsf{ABE} = \{\mathsf{Setup},\mathsf{Enc},\mathsf{KeyGen},\mathsf{Decrypt}\}$ for a relation family $R = \{R_\lambda : A_\lambda \times B_\lambda \to \{0,1\}\}_\lambda$ and a message space $\{\mathsf{msg}_\lambda\}_{\lambda\in\mathbb{N}}$ and an adversary A, let us define $\mathsf{Ada\text{-}IND}$ security game as follows.

1. **Setup phase:** On input 1^λ, the challenger samples $(\mathsf{PK},\mathsf{MSK}) \leftarrow \mathsf{Setup}(1^\lambda)$ and gives PK to A.
2. **Query phase:** During the game, A adaptively makes the following queries, in an arbitrary order. A can make unbounded many key queries, but can make only single challenge query.
   (a) **Key Queries:** A chooses an input $Y \in B_\lambda$. For each such query, the challenger replies with $\mathsf{SK}_Y \leftarrow \mathsf{KeyGen}(\mathsf{PK},\mathsf{MSK},Y)$.
   (b) **Challenge Query:** At some point, A submits a pair of equal length messages $(\mu_0,\mu_1) \in (\mathsf{msg})^2$ and the target $X^\star \in A_\lambda$ to the challenger. The challenger samples a random bit $b \leftarrow \{0,1\}$ and replies to A with $\mathsf{CT} \leftarrow \mathsf{Enc}(\mathsf{PK},X^\star,\mu_b)$.

   We require that $R(X^\star,Y) = 0$ holds for any Y such that A makes a key query for Y in order to avoid trivial attacks.
3. **Output phase:** A outputs a guess bit b' as the output of the experiment.

We define the advantage $\mathsf{Adv}^{\mathsf{Ada\text{-}IND}}_{\mathsf{ABE},\mathsf{A}}(1^\lambda)$ of A in the above game as

$$\mathsf{Adv}^{\mathsf{Ada\text{-}IND}}_{\mathsf{ABE},\mathsf{A}}(1^\lambda) := \left|\Pr[\mathsf{Exp}_{\mathsf{ABE},\mathsf{A}}(1^\lambda) = 1 | b = 0] - \Pr[\mathsf{Exp}_{\mathsf{ABE},\mathsf{A}}(1^\lambda) = 1 | b = 1]\right|.$$

The ABE scheme ABE is said to satisfy $\mathsf{Ada\text{-}IND}$ security (or simply *adaptive security*) if for any stateful PPT adversary A, there exists a negligible function $\mathsf{negl}(\cdot)$ such that $\mathsf{Adv}^{\mathsf{Ada\text{-}IND}}_{\mathsf{ABE},\mathsf{A}}(1^\lambda) \neq \mathsf{negl}(\lambda)$.

We can consider the following stronger version of the security where we require the ciphertext to be pseudorandom.

Definition 7.5 ($\mathsf{Ada\text{-}INDr}$ security for ABE)**.** We define $\mathsf{Ada\text{-}INDr}$ security game similarly to $\mathsf{Ada\text{-}IND}$ security game except that the adversary A chooses single message μ instead of (μ_0,μ_1) at the challenge phase and the challenger returns $\mathsf{CT} \leftarrow \mathsf{Enc}(\mathsf{PK},X^\star,\mu)$ if $b = 0$ and a random ciphertext $\mathsf{CT} \leftarrow \mathcal{CT}$ from a ciphertext space $\mathcal{CT}$ if $b = 1$. We define the advantage $\mathsf{Adv}^{\mathsf{Ada\text{-}INDr}}_{\mathsf{ABE},\mathsf{A}}(1^\lambda)$ of the adversary A accordingly and say that the scheme satisfies $\mathsf{Ada\text{-}INDr}$ security if the quantity is negligible.

We also consider (weaker) selective versions of the above notions, where A specifies its target $X^\star$ at the beginning of the game.

Definition 7.6 ($\mathsf{Sel\text{-}IND}$ security for ABE)**.** We define $\mathsf{Sel\text{-}IND}$ security game as $\mathsf{Ada\text{-}IND}$ security game with the exception that the adversary A has to choose the challenge ciphertext attribute $X^\star$ before the setup phase but key queries $Y_1, Y_2, \ldots$ and choice of (μ_0,μ_1) can still be adaptive. We define the advantage $\mathsf{Adv}^{\mathsf{Sel\text{-}IND}}_{\mathsf{ABE},\mathsf{A}}(1^\lambda)$ of the adversary A accordingly and say that the scheme satisfies $\mathsf{Sel\text{-}INDr}$ security (or simply *selective security*) if the quantity is negligible.

Definition 7.7 (Sel-INDr security for ABE)**.** We define Sel-INDr security game as Ada-INDr security game with the exception that the adversary A has to choose the challenge ciphertext attribute $X^\star$ before the setup phase but key queries $Y_1, Y_2, \ldots$ and choice of μ can still be adaptive. We define the advantage $\mathsf{Adv}^{\mathsf{Sel\text{-}INDr}}_{\mathsf{ABE},\mathsf{A}}(1^\lambda)$ of the adversary A accordingly and say that the scheme satisfies Sel-INDr security if the quantity is negligible.

In the following, we recall definitions of various ABEs by specifying the relation. We start with the standard notions of ciphertext-policy attribute-based encryption (CP-ABE) and key-policy attribute-based encryption (KP-ABE).

CP-ABE for circuits. We define CP-ABE for circuit class $\{\mathcal{C}_\lambda\}_\lambda$ by specifying the relation. Here, $\mathcal{C}_\lambda$ is a set of circuits with input length $\ell(\lambda)$ and binary output. We define $A^{\mathsf{CP}}_\lambda = \mathcal{C}_\lambda$ and $B^{\mathsf{CP}}_\lambda = \{0,1\}^\ell$. Furthermore, we define the relation R^{CP}_λ as

$$R^{\mathsf{CP}}_\lambda(C, \mathbf{x}) = \neg C(\mathbf{x}).^{16}$$

KP-ABE for circuits. To define KP-ABE for circuits, we simply swap key and ciphertext attributes in CP-ABE for circuits. More formally, to define KP-ABE for circuits, we define $A^{\mathsf{KP}}_\lambda = \{0,1\}^\ell$ and $B^{\mathsf{KP}}_\lambda = \mathcal{C}_\lambda$. We also define $R^{\mathsf{KP}}_\lambda : A^{\mathsf{KP}}_\lambda \times B^{\mathsf{KP}}_\lambda \to \{0,1\}$ as

$$R^{\mathsf{KP}}_\lambda(\mathbf{x}, C) = \neg C(\mathbf{x}).$$

7.2.2 Lattice Preliminaries

Here, we recall some facts on lattices that are needed for the exposition of our construction. Throughout this section, n, m, and q are integers such that $n = \mathsf{poly}(\lambda)$ and $m \geq n\lceil \log q \rceil$. In the following, let $\mathsf{SampZ}(\gamma)$ be a sampling algorithm for the truncated discrete Gaussian distribution over $\mathbf{Z}$ with parameter $\gamma > 0$ whose support is restricted to $z \in \mathbf{Z}$ such that $|z| \leq \sqrt{n}\gamma$.

Learning with Errors. We the introduce then learning with errors (LWE) problem.

Definition 7.8 (The LWE Assumption)**.** Let $n = n(\lambda)$, $m = m(\lambda)$, and $q = q(\lambda) > 2$ be integers and $\chi = \chi(\lambda)$ be a distribution over $\mathbf{Z}_q$. We say that the $\mathsf{LWE}(\mathsf{n}, \mathsf{m}, \mathsf{q}, \chi)$ hardness assumption holds if for any PPT adversary A we have

$$|\Pr[\mathsf{A}(\mathbf{A}, \mathbf{s}^\top \mathbf{A} + \mathbf{x}^\top) \to 1] - \Pr[\mathsf{A}(\mathbf{A}, \mathbf{v}^\top) \to 1]| \leq \mathsf{negl}(\lambda)$$

where the probability is taken over the choice of the random coins by the adversary A and $\mathbf{A} \leftarrow \mathbf{Z}_q^{n \times m}$, $\mathbf{s} \leftarrow \mathbf{Z}_q^n$, $\mathbf{x} \leftarrow \chi^m$, and $\mathbf{v} \leftarrow \mathbf{Z}_q^m$. We also say that $\mathsf{LWE}(\mathsf{n}, \mathsf{m}, \mathsf{q}, \chi)$ problem is subexponentially hard if the above probability is bounded by $2^{-n^\epsilon} \cdot \mathsf{negl}(\lambda)$ for some constant $0 < \epsilon < 1$ for all PPT A.

As shown by previous works [Reg09, BLP+13], if we set $\chi = \mathsf{SampZ}(\gamma)$, the $\mathsf{LWE}(\mathsf{n}, \mathsf{m}, \mathsf{q}, \chi)$ problem is as hard as solving worst case lattice problems such as gapSVP and SIVP with approximation factor $\mathsf{poly}(n) \cdot (q/\gamma)$ for some $\mathsf{poly}(n)$. Since the best known algorithms for 2^k-approximation of gapSVP and SIVP run in time $2^{\tilde{O}(n/k)}$, it follows that the above $\mathsf{LWE}(\mathsf{n}, \mathsf{m}, \mathsf{q}, \chi)$ with noise-to-modulus ratio 2^{-n^ϵ} is likely to be (subexponentially) hard for some constant ϵ.

Trapdoors. Let us consider a matrix $\mathbf{A} \in \mathbf{Z}_q^{n \times m}$. For all $\mathbf{V} \in \mathbf{Z}_q^{n \times m'}$, we let $\mathbf{A}_\gamma^{-1}(\mathbf{V})$ be an output distribution of $\mathsf{SampZ}(\gamma)^{m \times m'}$ conditioned on $\mathbf{A} \cdot \mathbf{A}_\gamma^{-1}(\mathbf{V}) = \mathbf{V}$. A γ-trapdoor for $\mathbf{A}$ is a trapdoor that enables one to sample from the distribution $\mathbf{A}_\gamma^{-1}(\mathbf{V})$ in time $\mathsf{poly}(n, m, m', \log q)$ for any $\mathbf{V}$. We slightly overload notation and denote a γ-trapdoor for $\mathbf{A}$ by $\mathbf{A}_\gamma^{-1}$. We also define the special gadget matrix $\mathbf{G} \in \mathbf{Z}_q^{n \times m}$ as the matrix obtained by padding $\mathbf{I}_n \otimes (1, 2, 4, 8, \ldots, 2^{\lceil \log q \rceil})$ with zero-columns. The following properties had been established in a long sequence of works [GPV08, CHKP10, ABB10a, ABB10b, MP12, BLP+13].

Lemma 7.9 (Properties of Trapdoors)**.** Lattice trapdoors exhibit the following properties.

[16] Here, we follow the standard convention in lattice-based cryptography where the decryption succeeds when $C(\mathbf{x}) = 0$ rather than $C(\mathbf{x}) = 1$.

1. Given $\mathbf{A}_\tau^{-1}$, one can obtain $\mathbf{A}_{\tau'}^{-1}$ for any $\tau' \geq \tau$.
2. Given $\mathbf{A}_\tau^{-1}$, one can obtain $[\mathbf{A}\|\mathbf{B}]_\tau^{-1}$ and $[\mathbf{B}\|\mathbf{A}]_\tau^{-1}$ for any $\mathbf{B}$.
3. There exists an efficient procedure $\mathsf{TrapGen}(1^n, 1^m, q)$ that outputs $(\mathbf{A}, \mathbf{A}_{\tau_0}^{-1})$ where $\mathbf{A} \in \mathbf{Z}_q^{n\times m}$ for some $m = O(n \log q)$ and is 2^{-n}-close to uniform, where $\tau_0 = \omega(\sqrt{n \log q \log m})$.

Lattice Evaluation. The following is an abstraction of the evaluation procedure in previous LWE based FHE and ABE schemes.

Lemma 7.10 (Fully Homomorphic Computation [GV15])**.** There exists a pair of deterministic algorithms $(\mathsf{EvalF}, \mathsf{EvalFX})$ with the following properties.

- $\mathsf{EvalF}(\mathbf{B}, F) \to \mathbf{H}_F$. Here, $\mathbf{B} \in \mathbf{Z}_q^{n\times m\ell}$ and $F : \{0,1\}^\ell \to \{0,1\}$ is a circuit.
- $\mathsf{EvalFX}(F, \mathbf{x}, \mathbf{B}) \to \widehat{\mathbf{H}}_{F,\mathbf{x}}$. Here, $\mathbf{x} \in \{0,1\}^\ell$ and $F : \{0,1\}^\ell \to \{0,1\}$ is a circuit with depth d. We have

$$[\mathbf{B} - \mathbf{x} \otimes \mathbf{G}]\widehat{\mathbf{H}}_{F,\mathbf{x}} = \mathbf{B}\mathbf{H}_F - F(\mathbf{x})\mathbf{G} \mod q,$$

  where we denote $[x_1\mathbf{G}\| \cdots \|x_k\mathbf{G}]$ by $\mathbf{x} \otimes \mathbf{G}$. Furthermore, we have

$$\|\mathbf{H}_F\|_\infty \leq m \cdot 2^{O(d)}, \quad \|\widehat{\mathbf{H}}_{F,\mathbf{x}}\|_\infty \leq m \cdot 2^{O(d)}.$$

- The running time of $(\mathsf{EvalF}, \mathsf{EvalFX})$ is bounded by $\mathsf{poly}(n, m, \log q, 2^d)$.

The above algorithms are taken from [GV15], which is a variant of similar algorithms proposed by Boneh et al. [BGG+14]. The algorithms in [BGG+14] work for any polynomial-sized circuit F, but $\|\mathbf{H}_F\|_\infty$ and $\|\mathbf{H}_{F,\mathbf{x}}\|_\infty$ become super-polynomial even if the depth of the circuit is shallow (i.e., logarithmic depth). On the other hand, the above algorithms run in polynomial time only when F is of logarithmic depth, but $\|\mathbf{H}_F\|_\infty$ and $\|\mathbf{H}_{F,\mathbf{x}}\|_\infty$ can be polynomially bounded. The latter property is crucial for our purpose.

7.2.3 KP-ABE Scheme by Boneh et al. [BGG+14].

We will use a variant of the KP-ABE scheme proposed by Boneh et al. [BGG+14] as a building block of our construction of CP-ABE. We call the scheme BGG^+ and provide the description of the scheme in the following. We focus on the case where the policies associated with secret keys are limited to circuits with logarithmic depth rather than arbitrary polynomially bounded depth, so that we can use the evaluation algorithm due to Gorbunov and Vinayagamurthy [GV15] (see Lemma 7.10). This allows us to bound the noise growth during the decryption by a polynomial factor, which is crucial for our application.

The scheme supports the circuit class $\mathcal{C}_{\ell(\lambda),d(\lambda)}$, which is a set of all circuits with input length $\ell(\lambda)$ and depth at most $d(\lambda)$ with arbitrary $\ell(\lambda) = \mathsf{poly}(\lambda)$ and $d(\lambda) = O(\log \lambda)$.

$\mathsf{Setup}(1^\lambda)$: On input 1^λ, the setup algorithm defines the parameters $n = n(\lambda)$, $m = m(\lambda)$, noise distribution noise over $\mathbf{Z}$, τ_0, τ, and $B = B(\lambda)$ as specified later. It then proceeds as follows.

1. Sample $(\mathbf{A}, \mathbf{A}_{\tau_0}^{-1}) \leftarrow \mathsf{TrapGen}(1^n, 1^m, q)$ such that $\mathbf{A} \in \mathbf{Z}_q^{n\times m}$.
2. Sample random matrix $\mathbf{B} = (\mathbf{B}_1, \ldots, \mathbf{B}_\ell) \leftarrow (\mathbf{Z}_q^{n\times m})^\ell$ and a random vector $\mathbf{u} \leftarrow \mathbf{Z}_q^n$.
3. Output the master public key $\mathsf{PK} = (\mathbf{A}, \mathbf{B}, \mathbf{u})$ and the master secret key $\mathsf{MSK} = \mathbf{A}_{\tau_0}^{-1}$.

$\mathsf{KeyGen}(\mathsf{PK}, \mathsf{MSK}, F)$: The key generation algorithm takes as input the master public key PK, the master secret key MSK, and a circuit $F \in \mathcal{F}_\lambda$ and proceeds as follows.

1. Compute $\mathbf{H}_F = \mathsf{EvalF}(\mathbf{B}, F)$ and $\mathbf{B}_F = \mathbf{B}\mathbf{H}_F$.
2. Compute $[\mathbf{A}\|\mathbf{B}_F]_\tau^{-1}$ from $\mathbf{A}_{\tau_0}^{-1}$ and sample $\mathbf{r} \in \mathbf{Z}^{2m}$ as $\mathbf{r} \leftarrow [\mathbf{A}\|\mathbf{B}_F]_\tau^{-1}(\mathbf{u})$.

3. Output the secret key $\mathsf{SK}_F := \mathbf{r}$.

$\mathsf{Enc}(\mathsf{PK}, \mathbf{x}, \mu)$: The encryption algorithm takes as input the master public key PK, an attribute $\mathbf{x} \in \{0,1\}^\ell$, and a message $\mu \in \{0,1\}$ and proceeds as follows.

1. Sample $\mathbf{s} \leftarrow \mathbf{Z}_q^n$, $e_1 \leftarrow \mathsf{noise}$, $\mathbf{e}_2 \leftarrow \mathsf{noise}^m$, and $\mathbf{S}_{i,b} \leftarrow \{-1,1\}^{m\times m}$ for $i \in [\ell]$ and $b \in \{0,1\}$. Then, set $\mathbf{e}_{i,b} := \mathbf{S}_{i,b}^\top \mathbf{e}_2$ for $i \in [\ell]$ and $b \in \{0,1\}$.
2. Compute
$$\psi_1 := \mathbf{s}^\top \mathbf{u} + e_1 + \mu\lceil q/2\rceil \in \mathbf{Z}_q,\ \psi_2^\top := \mathbf{s}^\top \mathbf{A} + \mathbf{e}_2^\top \in \mathbf{Z}_q^m,$$
$$\psi_{i,b}^\top := \mathbf{s}^\top(\mathbf{B} - x_i\mathbf{G}) + \mathbf{e}_{i,b}^\top \in \mathbf{Z}_q^m \text{ for all } i \in [\ell],\ \ b \in \{0,1\}.$$
3. Output the ciphertext $\mathsf{CT}_\mathbf{x} := (\psi_1, \psi_2, \{\psi_{i,x_i}\}_{i\in[\ell]})$, where x_i is the i-th bit of $\mathbf{x}$.

$\mathsf{Decrypt}(\mathsf{PK}, \mathsf{SK}_\mathbf{x}, \mathbf{x}, F, \mathsf{CT}_F)$: The decryption algorithm takes as input the master public key PK, a secret key SK_F for a circuit F, and a ciphertext $\mathsf{CT}_\mathbf{x}$ for an attribute $\mathbf{x}$ and proceeds as follows.

1. Parse $\mathsf{CT}_\mathbf{x} \to (\psi_1 \in \mathbf{Z}_q, \psi_2 \in \mathbf{Z}_q^m, \{\psi_{i,x_i} \in \mathbf{Z}_q^m\}_{i\in[\ell]})$, and $\mathsf{SK}_F \in \mathbf{Z}^{2m}$. If any of the component is not in the corresponding domain or $F(\mathbf{x}) = 1$, output $\bot$.
2. Concatenate $\{\psi_{i,x_i}\}_{i\in[\ell]}$ to form $\psi_3^\top = (\psi_{1,x_1}^\top, \ldots, \psi_{\ell,x_\ell}^\top)$.
3. Compute
$$\psi' := \psi_1 - [\psi_2^\top \| \psi_3^\top]\mathbf{r}.$$
4. Output 0 if $\psi' \in [-B, B]$ and 1 if $[-B + \lceil q/2\rceil, B + \lceil q/2\rceil]$.

Remark 7.11. We note that the encryption algorithm above computes redundant components $\{\psi_{i,\neg x_i}\}_{i\in[\ell]}$ in the second step, which are discarded in the third step. However, due to this redundancy, the scheme has the following special structure that will be useful for us. Namely, the first and the second steps of the encryption algorithm can be executed without knowing $\mathbf{x}$. Only the third step of the encryption algorithm needs the information of $\mathbf{x}$, where it chooses $\{\psi_{i,x_i}\}_{i\in[\ell]}$ from $\{\psi_{i,b}\}_{i\in[\ell],b\in\{0,1\}}$ depending on each bit of $\mathbf{x}$ and then output the former terms along with ψ_1 and ψ_2.

There, the encryption algorithm, who takes as input a circuit C that specifies the policy and does not know the corresponding input $\mathbf{x}$, executes the first two steps of the above encryption algorithm. This is possible since these two steps do not need the knowledge of $\mathbf{x}$.

Parameters and Security. We choose the parameters for the scheme as follows:

$$m = n^{1.1}\log q, \qquad q = 2^{\Theta(\lambda)}, \qquad \chi = \mathsf{SampZ}(3\sqrt{n}),$$
$$\tau_0 = n\log q \log m, \qquad \tau = m^{3.1}\ell \cdot 2^{O(d)} \qquad B = n^2m^2\tau \cdot 2^{O(d)}.$$

The parameter n will be chosen depending on whether we need $\mathsf{Sel\text{-}INDr}$ security or $\mathsf{Ada\text{-}INDr}$ security for the scheme. If it suffices to have $\mathsf{Sel\text{-}INDr}$ security, we set $n = \lambda^c$ for some constant $c > 1$. If we need $\mathsf{Ada\text{-}INDr}$ security, we have to enlarge the parameter to be $n = (\ell\lambda)^c$ in order to compensate for the security loss caused by the complexity leveraging.

We remark that if we were to use the above ABE scheme stand-alone, we would have been able to set q polynomially bounded as in [GV15]. The reason why we set q exponentially large is that we combine the scheme with bilinear maps of order q to lift the ciphertext components to the exponent so that they are "hidden" as in [AY20]. In order to use the security of the bilinear map, we set the group order q to be exponentially large.

The following theorem summarizes the security and efficiency properties of the construction. There are two parameter settings depending on whether we assume subexponential hardness of LWE or not.

Theorem 7.12 (Adapted from [GV15, BGG+14]). Assuming hardness of $\mathsf{LWE}(\mathsf{n}, \mathsf{m}, \mathsf{q}, \chi)$ with $\chi = \mathsf{SampZ}(3\sqrt{n})$ and $q = O(2^{n^{1/\epsilon}})$ for some constant $\epsilon > 1$, the above scheme satisfies Sel-INDr security. Assuming *subexponential* hardness of $\mathsf{LWE}(\mathsf{n}, \mathsf{m}, \mathsf{q}, \chi)$ with the same parameters, the above scheme satisfies Ada-INDr security with respect to the ciphertext space $\mathcal{CT} := \mathbf{Z}_q^{m(\ell+1)+1}$

7.3 Construction

Here we provide our extension of [AY20] to distribute the setup and key generation process among non-interacting a-priori fixed number of authorities. Below we follow the [AY20] scheme syntactically and describe our modified construction. As in [AY20], we also work over asymmetric bilinear groups. Also, let $\mathbb{G}_1, \mathbb{G}_2$ and $\mathbb{G}_T$ be the source and target groups, respectively. Our construction additionally relies on a hash function $H : \{0,1\}^\lambda \times \{0,1\}^\ell \to \mathbb{G}_2$ that is later modelled as a random oracle. Below we provide our ABE scheme following the notation used in [AY20, Section 3] almost verbatim for presentation purposes. For completeness, we recall the notation used later in Section 7.2.2 and Section 2.2.

Our construction can deal with any circuit class $\mathcal{F} = \{\mathcal{F}_\lambda\}_\lambda$ that is subclass of $\{\mathcal{C}_{\ell(\lambda),d(\lambda)}\}_\lambda$ with arbitrary $\ell(\lambda) \le \mathsf{poly}(\lambda)$ and $d(\lambda) = O(\log \lambda)$, where $\mathcal{C}_{\ell(\lambda),d(\lambda)}$ is a set of circuits with input length $\ell(\lambda)$ and depth at most $d(\lambda)$. Also, all algorithms are provided the public parameters as an additional input. We do not explicitly write for ease of exposition.

$\mathsf{GSetup}(1^\lambda)$: On input 1^λ, the setup algorithm samples a group description $\mathbf{G} = (q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, [1]_1, [1]_2)$, and sets the public parameters as $\mathsf{PP} = \mathbf{G}$.

$\mathsf{LSetup}(\mathsf{PP}, 1^N, 1^\ell)$: On input the bilinear group description $\mathsf{PP} = \mathbf{G}$, number of authorities N, attribute length ℓ, the authority setup algorithm defines the parameters $n = n(\lambda)$, $m = m(\lambda)$, noise distribution noise over $\mathbf{Z}$, τ_0, τ, and $B = B(\lambda)$ as specified in Section 7.2.3. It then sets $L := (2\ell+1)m + 2$ and proceeds as follows.

1. Sample $\mathbf{w} \leftarrow (\mathbf{Z}_q^*)^L$ and compute $[\mathbf{w}]_1$.
2. Output $\mathsf{PK} = [\mathbf{w}]_1$ and $\mathsf{MSK} = \mathbf{w}$.

$\mathsf{KeyGen}(\mathsf{MSK}_j, \mathsf{GID}, \mathbf{x})$: The key generation algorithm takes as input the authority master secret key $\mathsf{MSK}_j = \mathbf{w}^{(j)}$, global identifier GID, and an attribute $\mathbf{x} \in \{0,1\}^\ell$ with $x_1 = 1$ and proceeds as follows.

1. Let $\mathbf{1} := (1, \dots, 1)^\top \in \mathbf{Z}_q^m$ and $\mathbf{0} := (0, \dots, 0)^\top \in \mathbf{Z}_q^m$. Set

$$\phi_0 = 1 \in \mathbf{Z}_q, \quad \phi_1 = 1 \in \mathbf{Z}_q, \quad \phi_2 := \mathbf{1} \in \mathbf{Z}_q^m,$$
$$\phi_{i,b} := \begin{cases} \mathbf{1} \in \mathbf{Z}_q^m & \text{if } b = x_i \\ \mathbf{0} \in \mathbf{Z}_q^m & \text{if } b \neq x_i \end{cases} \quad \text{for } i \in [\ell] \text{ and } b \in \{0,1\}. \tag{7.1}$$

2. Vectorize $(\phi_0, \phi_1, \phi_2, \{\phi_{i,b}\}_{i,b})$ to form a vector $\mathbf{d} \in \mathbf{Z}_q^L$ by concatenating each entry of the vectors in a predetermined order.
3. Compute $h = H(\mathsf{GID}, \mathbf{x}) \in \mathbb{G}_2$.
4. Compute $h^{\mathbf{d} \oslash \mathbf{w}^{(j)}} \in \mathbb{G}_2^L$ using $h \in \mathbb{G}_2$ and $\mathbf{w}^{(j)}$ in the master key.
5. Output $\mathsf{SK}_{j,\mathsf{GID},\mathbf{x}} = h^{\mathbf{d} \oslash \mathbf{w}^{(j)}}$.

$\mathsf{Enc}(\{\mathsf{PK}_i\}_{i\in[N]}, F, \mu)$: The encryption algorithm takes as input a list of authority master public keys PK_i for $i \in [N]$, the circuit F, and a message $\mu \in \{0,1\}$ and proceeds as follows.

1. Sample fresh BGG^+ scheme:
   (a) Sample $(\mathbf{A}, \mathbf{A}_{\tau_0}^{-1}) \leftarrow \mathsf{TrapGen}(1^n, 1^m, q)$ such that $\mathbf{A} \in \mathbf{Z}_q^{n \times m}$.

(b) Sample random matrix $\mathbf{B} = (\mathbf{B}_1, \ldots, \mathbf{B}_\ell) \leftarrow (\mathbf{Z}_q^{n \times m})^\ell$ and a random vector $\mathbf{u} \leftarrow \mathbf{Z}_q^n$.

2. Compute BGG^+ function key for circuit F:
   (a) Compute $\mathbf{H}_F = \mathsf{EvalF}(\mathbf{B}, F)$ and $\mathbf{B}_F = \mathbf{B}\mathbf{H}_F$.
   (b) Compute $[\mathbf{A} \| \mathbf{B}_F]_\tau^{-1}$ from $\mathbf{A}_{\tau_0}^{-1}$ and sample $\mathbf{r} \in \mathbf{Z}^{2m}$ as $\mathbf{r} \leftarrow [\mathbf{A} \| \mathbf{B}_F]_\tau^{-1}(\mathbf{u})$.

3. Compute BGG^+ ciphertext for all possible inputs:
   (a) Sample $\mathbf{s} \leftarrow \mathbf{Z}_q^n$, $e_1 \leftarrow \mathsf{noise}$, $\mathbf{e}_2 \leftarrow \mathsf{noise}^m$, and $\mathbf{S}_{i,b} \leftarrow \{-1, 1\}^{m \times m}$ for $i \in [\ell]$ and $b \in \{0, 1\}$. Then, set $\mathbf{e}_{i,b} := \mathbf{S}_{i,b}^\top \mathbf{e}_2$ for $i \in [\ell]$ and $b \in \{0, 1\}$.
   (b) Compute

$$\begin{aligned} \psi_0 &:= 1 \in \mathbf{Z}_q, \quad \psi_1 := \mathbf{s}^\top \mathbf{u} + e_1 + \mu \lceil q/2 \rceil \in \mathbf{Z}_q, \\ \psi_2^\top &:= \mathbf{s}^\top \mathbf{A} + \mathbf{e}_2^\top \in \mathbf{Z}_q^m, \\ \psi_{i,b}^\top &:= \mathbf{s}^\top (\mathbf{B}_i - b\mathbf{G}) + \mathbf{e}_{i,b}^\top \in \mathbf{Z}_q^m \quad \text{for } i \in [\ell] \text{ and } b \in \{0, 1\}. \end{aligned} \tag{7.2}$$

4. N-out-of-N secret share the BGG^+ ciphertexts:
   (a) Vectorize $(\psi_0, \psi_1, \psi_2, \{\psi_{i,b}\}_{i,b})$ to form a vector $\mathbf{c} \in \mathbf{Z}_q^L$ by concatenating each entry of the vectors in a predetermined order (that aligns with the one used in the key generation algorithm).
   (b) Sample $N - 1$ uniformly random vectors $\widetilde{\mathbf{c}}^{(j)} \leftarrow \mathbf{Z}_q^L$ for $j \in [N - 1]$.
   (c) Compute vector $\widetilde{\mathbf{c}}^{(N)}$ as $\widetilde{\mathbf{c}}^{(N)} = \mathbf{c} + \sum_{j \in [N-1]} \widetilde{\mathbf{c}}^{(j)}$.
5. Encode the secret shared ciphertexts in exponent of bilinear group:
   (a) Sample $\gamma \leftarrow \mathbf{Z}_q^*$.
   (b) For every $j \in [N]$, compute $[\gamma \widetilde{\mathbf{c}}^{(j)} \odot \mathbf{w}^{(j)}]_1 \in \mathbb{G}_1^L$ from γ, $\widetilde{\mathbf{c}}^{(j)}$, and $[\mathbf{w}^{(j)}]_1$ in PK_j.
6. Output $\mathsf{CT}_F = \Big(\mathsf{CT}_0 = (\mathbf{A}, \mathbf{B}), \{\mathsf{CT}_1^{(j)} = [\gamma \widetilde{\mathbf{c}}^{(j)} \odot \mathbf{w}^{(j)}]_1\}_{j \in [N]}, \mathsf{CT}_2 = \mathbf{r}\Big)$.

$\mathsf{Dec}(\{\mathsf{SK}_{j,\mathsf{GID},\mathbf{x}}\}_{j \in [N]}, \mathbf{x}, F, \mathsf{CT}_F)$: The decryption algorithm takes as input the N partial secret keys $\mathsf{SK}_{j,\mathsf{GID},\mathbf{x}}$ for an attribute $\mathbf{x}$ and authority index $j \in [N]$, and the ciphertext CT_F for a circuit F and proceeds as follows.

1. Parse $\mathsf{CT}_F \to (\mathsf{CT}_0 = (\mathbf{A} \in \mathbf{Z}_q^{n \times m}, \mathbf{B} \in \mathbf{Z}_q^{n \times m\ell}), \{\mathsf{CT}_1^{(j)} \in \mathbb{G}_1^L\}_j, \mathsf{CT}_2 \in \mathbf{Z}^{2m})$ and $\mathsf{SK}_\mathbf{x} \in \mathbb{G}_2^L$. If any of the component is not in the corresponding domain or $F(\mathbf{x}) = 1$, output $\perp$.

2. Unmask and reconstruct the secret-shared BGG^+ ciphertexts corresponding to $\mathbf{x}$ by using each partial secret key:
   Compute the partial unmasked ciphertexts $[\mathbf{v}^{(j)}]_T := \mathsf{CT}_1^{(j)} \odot \mathsf{SK}_{j,\mathsf{GID},\mathbf{x}}$, and reconstruct them as $[\mathbf{v}]_T := \prod_{j \in [N]} [\mathbf{v}^{(j)}]_T$. Next, de-vectorize $[\mathbf{v}]_T$ to obtain

$$[v_0]_T \in \mathbf{G}_T, \ [v_1]_T \in \mathbf{G}_T, \ [\mathbf{v}_2]_T \in \mathbf{G}_T^m, \ [\mathbf{v}_{i,b}]_T \in \mathbf{G}_T^m, \quad \text{for } i \in [\ell],\ b \in \{0, 1\}.$$

3. Evaluate circuit F on BGG^+ ciphertexts in the exponent:
   Compute $\widehat{\mathbf{H}}_{F,\mathbf{x}} = \mathsf{EvalF}(F, \mathbf{x}, \mathbf{B})$.

4. Perform BGG^+ decryption in the exponent:
   Form $[\mathbf{v}_\mathbf{x}^\top]_T = [\mathbf{v}_{1,x_1}^\top, \ldots, \mathbf{v}_{\ell,x_\ell}^\top]_T$ and $\mathsf{CT}_2^\top = (\mathbf{r}_1^\top \in \mathbf{Z}_q^m, \mathbf{r}_2^\top \in \mathbf{Z}_q^m)$. Then compute

$$[v']_T := [v_1 - (\mathbf{v}_2^\top \mathbf{r}_1 + \mathbf{v}_\mathbf{x}^\top \widehat{\mathbf{H}}_{F,\mathbf{x}} \mathbf{r}_2)]_T$$

   from $[v_1]_T$, $[\mathbf{v}_2]_T$, $[\mathbf{v}_\mathbf{x}]_T$, $\mathbf{r}_1$, $\mathbf{r}_2$, and $\widehat{\mathbf{H}}_{F,\mathbf{x}}$.

5. Recover exponent via brute force if $F(\mathbf{x}) = 0$:
   Find $\eta \in [-B, B] \cup [-B + \lceil q/2 \rceil, B + \lceil q/2 \rceil]$ such that $[v_0]_T^\eta = [v']_T$ by brute-force search. If there is no such η, output $\perp$. To speed up the operation, one can employ the baby-step giant-step algorithm.

6. Output 0 if $\eta \in [-B, B]$ and 1 if $[-B + \lceil q/2 \rceil, B + \lceil q/2 \rceil]$.

Correctness. The correctness of our scheme follows almost directly from the correctness of the [AY20] CP-ABE scheme. The only difference being that in [AY20] the unmasking step directly reveals the BGG^+ ciphertexts in the exponent, whereas in the above construction the unmasking step reveals the secret shares of the BGG^+ ciphertext, which we first combine to recover the BGG^+ ciphertext and then decrypt as in [AY20]. More concretely, observe that for every $j \in [N]$,

$$\mathsf{CT}_1^{(j)} \odot \mathsf{SK}_{j,\mathsf{GID},\mathbf{x}} = h^{\gamma \cdot \widetilde{\mathbf{c}}^{(j)} \odot \mathbf{d}_\mathbf{x}},$$

where $h = H(\mathsf{GID}, \mathbf{x})$, and $\mathbf{d}_\mathbf{x}$ is as defined in key generation. Thus, we have that

$$\prod_{j \in [N]} \mathsf{CT}_1^{(j)} \odot \mathsf{SK}_{j,\mathsf{GID},\mathbf{x}} = h^{\gamma \cdot \sum_{j \in [N]} (\widetilde{\mathbf{c}}^{(j)} \odot \mathbf{d}_\mathbf{x})} = h^{\gamma \cdot \mathbf{c} \odot \mathbf{d}_\mathbf{x}}.$$

Given this, rest of the correctness proof is identical to that provided in [AY20].

Proof of Security

Here we prove security of our construction. Formally, we prove the following.

Theorem 7.13. If the underlying key-policy ABE scheme BGG^+ satisfies semantic security with pseudorandom ciphertexts, then our distributed ciphertext-policy ABE scheme is semantically secure (Definition 7.1) in the generic group model.

The proof structure is similar to that of [AY20], except now the reduction algorithm needs to additionally answer key queries for GID-attribute vector pairs $(\mathsf{GID}, \mathbf{x})$ for which $F^*(\mathbf{x}) = 1$ (where F^* is the challenge circuit) for some non-corrupt/honest authorities as well. Recall that the security must hold even if the attacker queries all but one honest authorities on such accepting input attribute vectors $\mathbf{x}$. Despite such additional key queries, it turns out the [AY20] proof could be extended to handle them. Intuitively, this is because even given such partial accepting keys one could show that the adversary is not able to make any "bad" zero-test queries as defined in [AY20]. Below we sketch the hybrid games along the lines of [AY20]. We refer the reader to [AY20, Section 4] for more details.

Game 0. This corresponds to the distributed ABE security game as described in Definition 7.1. Let Q_{kq} denote the total number of queries made by the adversary, Q_{zt} denote the number of zero-test queries, and Q_{ro} denote the number of RO queries. Here the challenger simulates both the random oracle as well as generic group oracle. (That is, it gives handles to the group elements each time, and when answering a RO query, it again answers with a handle since hash H maps elements to $\mathbb{G}_2$.)

Game 1. This is same as the previous game, except the way in which the challenger answers/resolves each call to the hash function/random oracle:

— on input $(\mathsf{GID}, \mathbf{x})$, the challenger chooses a random exponent $\delta_{\mathsf{GID},\mathbf{x}} \leftarrow \mathbf{Z}_q$, stores $\delta_{\mathsf{GID},\mathbf{x}}$ for answering future queries on same input, and responds with $\left[\delta_{\mathsf{GID},\mathbf{x}}\right]_1$. (Intuitively, the challenger samples a random exponent per input to simulate the randomness term δ in the [AY20] secret keys.)

Game 2. This is same as the previous game, except the challenger samples the honest authority master keys $\mathbf{w}^{(j)}$ (for $j \notin S^*$, where S^* is the set of corrupt users), random exponents $\delta_{\mathsf{GID},\mathbf{x}}$ (for all key queries $(\mathsf{GID}, \mathbf{x})$, and RO queries too), ciphertext components $\mathbf{A}, \mathbf{B}, \mathbf{u}, \gamma, \mu$, $\mathbf{c}$, and $\widetilde{\mathbf{c}}^{(j)}$ (for $j \in [N]$) at the beginning of the game. Since all these terms are independent of the challenge circuit F^*, thus the game is well defined. *(This is similar to that done in Game* 1 *in [AY20, Section 4].)*

Game 3. This is same as the previous game, except the challenger (partially) switches to the symbolic group model and replaces the terms $\{\mathbf{w}^{(j)}\}_{j\notin S^*}$, $\{\delta_{\mathsf{GID},\mathbf{x}}\}_{\mathsf{GID},\mathbf{x}}$, γ, $\mathbf{c}$ and $\{\widetilde{\mathbf{c}}^{(j)}\}_j$ with their respective formal variables $\{\mathbf{W}^{(j)}\}_{j\notin S^*}$, $\{\Delta_{\mathsf{GID},\mathbf{x}}\}_{\mathsf{GID},\mathbf{x}}$, Γ, $\mathbf{C}$ and $\{\widetilde{\mathbf{C}}^{(j)}\}_j$. Let $\mathbf{W}_j = (W_1^{(j)}, \ldots, W_L^{(j)})$ and $\widetilde{\mathbf{C}}^{(j)} = (\widetilde{C}_1^{(j)}, \ldots, \widetilde{C}_L^{(j)})$. As a result, all handles given to adversary $\mathcal{A}$ refer to elements in the ring

$$\mathbb{T} := \mathbf{Z}_q[\{W_i^{(j)}, 1/W_i^{(j)}\}_{j\notin S^*, i\in[L]}, \{\Delta_{\mathsf{GID},\mathbf{x}}\}_{\mathsf{GID},\mathbf{x}}, \Gamma, \{\widetilde{C}_i^{(j)}\}_{j\in[N],i\in[L]}],$$

where $\{1/W_i^{(j)}\}_{j,i}$ are needed to represent the components in the secret keys. However, whenever it performs a zero-test then it checks if the underlying ring element evaluates to 0 by substituting formal variables with corresponding elements in $\mathbf{Z}_q$. *(This is similar to that done in Game* 2 *in [AY20, Section 4]. However, the only difference is that since the adversary corrupts some of the key authorities, thus it knows/chooses the key vectors* $\mathbf{w}^{(j)}$ *on its own for* $j \in S^*$*. Recall that [AY20] this was not the case.)*

Note. As in [AY20], we extend the definition of sets $S_{T,1}$ and $S_{T,2}$ to our above distributed ABE construction. Briefly, the difference being that now the number of products being unmatching positions increases by a large amount due to cross terms between ciphertext and key components that correspond to different authorities.

Game 4. This is same as the previous game, except the challenger treates $\{\mathbf{W}_j\}_{j\notin S^*}$, $\{\Delta_{\mathsf{GID},\mathbf{x}}\}_{\mathsf{GID},\mathbf{x}}$, and Γ as formal variables rather than elements in $\mathbf{Z}_q$ even when answering zero-test queries. *(This is similar to that done in Game* 3 *in [AY20, Section 4].)*

Game 5. This is same as the previous game, except now the challenger aborts the game and enforces the adversary to output a random bit when there exists $i \in [L]$ and $j \notin S^*$ such that $\widetilde{c}_i^{(j)} = 0$, where $\widetilde{\mathbf{c}}^{(j)} = (\widetilde{c}_1^{(j)}, \ldots, \widetilde{c}_L^{(j)})^\top$ is sampled as in the previous game. *(Note that this departs slightly from Game* 4 *in [AY20, Section 4]. This is due to the fact that [AY20] is a single-authority ABE system, whereas ours is a distributed scheme.)*

Game 6. This is same as the previous game, except now the challenger changes how it answers the zero-test queries slightly. In particular, when adversary makes a zero-test query, then the challenger interprets it as a ring element of the form $\sum_{Z\in S_{T,1}} a_Z Z + \sum_{Z\in S_{T,2}} a_Z Z$ (where sets $S_{T,1}$ and $S_{T,2}$ are defined as appropriate extensions of that in [AY20] as discussed after description of Game 3 above). And, if there exists a $Z \in S_{T,1}$ such that its coefficient $a_Z \neq 0$, then the challenger returns 0, otherwise it answers the query as before. *(This is similar to that done in Game* 5 *in [AY20, Section 4].)*

Game 7. This is same as the previous game, except now the challenger answers *all* the zero-test queries completely over the ring (defined over the formal variables). Namely, when adversary makes a zero-test query for a handle corresponding to a ring element $f \in \mathbb{T}$, then the challenger returns 0 if f is not already a zero element over the ring $\mathbb{T}$, that is $f \neq 0$ over $\mathbb{T}$. In other words, the challenger returns 0 if there exists a $Z \in S_{T,1} \cup S_{T,2}$ such that its coefficient $a_Z \neq 0$. *Note that* $\{\widetilde{\mathbf{c}}^{(j)}\}_{j\notin S^*}$ *is not used in this game, thus the challenger does not have to sample them any more. (This is similar to that done in Game* 6 *in [AY20, Section 4].)*

Indistinguishability of games

We complete the proof by showing that adjacent games are indistinguishable. For any adversary $\mathcal{A}$ and game X, we denote by $\mathsf{Adv}_s^{\mathcal{A}}(\lambda)$, the probability that $\mathcal{A}$ wins in game S.

Lemma 7.14. For any adversary $\mathcal{A}$, we have that $\mathsf{Adv}_0^{\mathcal{A}}(\lambda) = \mathsf{Adv}_1^{\mathcal{A}}(\lambda)$.

Proof. This follows directly from the fact that H is modelled as a programmable random oracle. □

Lemma 7.15. For any adversary $\mathcal{A}$, we have that $\mathsf{Adv}_1^{\mathcal{A}}(\lambda) = \mathsf{Adv}_2^{\mathcal{A}}(\lambda)$.

Proof. Since this is only a conceptual change where the challenger pre-computes ciphertext terms, the lemma immediately follows. □

Lemma 7.16. For any adversary $\mathcal{A}$, we have that $\mathsf{Adv}_2^{\mathcal{A}}(\lambda) = \mathsf{Adv}_3^{\mathcal{A}}(\lambda)$.

Proof. This lemma again follows from the fact that this game just constitutes a syntactic difference, where the only difference is in how queries are answered. But since the terms are sampled identically in both games, thus the distributions are identical. Therefore, the lemma follows. □

Lemma 7.17. For any adversary $\mathcal{A}$, we have that $\mathsf{Adv}_3^{\mathcal{A}}(\lambda) - \mathsf{Adv}_4^{\mathcal{A}}(\lambda) \leq Q_{\mathsf{zt}}(N \cdot L + 3)^2/q$.

Proof. The proof of this lemma is identical to that of [AY20, Lemma 4.4], and follows by an application of Schwartz-Zippel lemma and union bound. The main difference is that the number of authority random variables $\mathbf{W}$ grew from L to $N \cdot L$, thus the degree of the polynomial on which Schwartz-Zippel is applied gets raised by a factor of $N - |S^*| \leq N$, where S^* is the set of corrupt authorities. □

Lemma 7.18. For any adversary $\mathcal{A}$, we have that $\mathsf{Adv}_4^{\mathcal{A}}(\lambda) - \mathsf{Adv}_5^{\mathcal{A}}(\lambda) \leq N \cdot L/q$.

Proof. This is a statistical property, and follows from the combination of following two facts. First, the challenger samples all but the last encoding vector $\widetilde{\mathbf{c}}^{(j)}$ uniformly at random. Thus, by a simple union bound we get that the probability $\widetilde{c}_i^{(j)} \neq 0$ for all $i \in [L]$ and $j \in [N-1]$ is at most $(N-1)L/q$. Now, the vector $\widetilde{\mathbf{c}}^{(N)}$ contains the BGG^+ ciphertext components which are either fixed to be 1 or well distributed over $\mathbf{Z}_q$ (assuming LWE).[17] Therefore, the lemma follows by combining these. □

Lemma 7.19. For any adversary $\mathcal{A}$, we have that $\mathsf{Adv}_5^{\mathcal{A}}(\lambda) = \mathsf{Adv}_6^{\mathcal{A}}(\lambda)$.

Proof. The proof of this lemma is similar to that of [AY20, Lemma 4.6], and intuitively follows from the fact that all monomials in $S_{T,1}$ are distinct (even if one substitutes ciphertext formal variables $\{\widetilde{C}_i^{(j)}\}_{j,i}$ with its actual value $\{\widetilde{c}_i^{(j)}\}_{j,i}$ and ignore the difference between the coefficients of the monomials), thus if $a_Z \neq 0$ for some $Z \in S_{T,1}$, then that term will never get cancelled, thus not lead to a successful zero-test query. Hence, the lemma follows. □

Lemma 7.20. If the underlying key-policy ABE scheme BGG^+ satisfies semantic security with pseudorandom ciphertexts and $\log q = \omega(\log \lambda)$, then for every PPT adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that $\mathsf{Adv}_6^{\mathcal{A}}(\lambda) - \mathsf{Adv}_7^{\mathcal{A}}(\lambda) \leq \mathsf{negl}(\lambda)$.

Proof. In [AY20], the authors proposed a hybrid based approach for proving the single-authority variant of the above lemma. Here we provide a different case based analysis. Our proof structure could also be used in [AY20] to potentially provide a simpler of the same.

The proof of this lemma is based on the ideas used in the proof of [AY20, Lemma 4.7], but there are a few differences. Very briefly, the main difference is that here we could have more than one non-corrupt/honest authorities, thus an adversary could potentially make key queries to all but one honest authorities on GID-attribute pairs $(\mathsf{GID}, \mathbf{x})$ where $F^*(\mathbf{x}) = 1$. *(Note that an adversary is allowed to query some of the*

[17] This part is similar to the proof of [AY20, Lemma 4.5].

honest authorities on accepting attributes, as long as it does not query all the honest authorities thereby simply being able to decrypt the challenge ciphertext, since we want to disallow an adversary utilizing partial secret keys for two different accepting attributes.)

First, observe that Games 6 and 7 differ only when $\mathcal{A}$ makes a zero-test query for a handle corresponding to $f \in \mathbb{T}$ that can be represented as

$$f(\{W_i^{(j)}\}_{j\notin S^*,i\in[L]}, \{\Delta_{\mathsf{GID},\mathbf{x}}\}_{\mathsf{GID},\mathbf{x}}, \Gamma, \{\widetilde{C}_i^{(j)}\}_{j\in[N],i\in[L]}) = \sum_{Z\in S_{T,2}} a_Z Z \tag{7.3}$$

and satisfies $f \neq 0$ over $\mathbb{T}$, but

$$f(\{W_i^{(j)}\}_{j\notin S^*,i\in[L]}, \{\Delta_{\mathsf{GID},\mathbf{x}}\}_{\mathsf{GID},\mathbf{x}}, \Gamma, \{\widetilde{c}_i^{(j)}\}_{j\in[N],i\in[L]}) = 0. \tag{7.4}$$

That is, only substituting the formal variables corresponding to $\widetilde{C}_i^{(j)}$ to their actual values makes the ring element go to 0. Following [AY20], we call such a query *bad*. Now note that the set $S_{T,2}$ contains the following terms:

$$S_{T,2} = \left\{ \Gamma\widetilde{C}_i^{(j)}\Delta_{\mathsf{GID},\mathbf{x}} : \begin{array}{c} (j,\mathsf{GID},\mathbf{x}) \in \mathcal{Q}_{\mathsf{key}},\ i\in[L],\ d_{\mathbf{x},i}=1 \\ \text{where } \mathbf{d}_{\mathbf{x}} = (d_{\mathbf{x},1},\ldots,d_{\mathbf{x},L}) \text{ and} \\ \mathcal{Q}_{\mathsf{key}} \text{ denotes the set of key queries by } \mathcal{A} \end{array} \right\}$$
$$\bigcup \left\{ \Gamma\widetilde{C}_i^{(j)}\Delta_{\mathsf{GID},\mathbf{x}} : \begin{array}{c} j\in S^*, i\in[L], (\mathsf{GID},\mathbf{x}) \in \mathcal{Q}_{\mathsf{ro}}, \\ \text{where } \mathcal{Q}_{\mathsf{ro}} \text{ denotes the set of RO queries by } \mathcal{A} \end{array} \right\}$$

Suppose $\mathcal{A}$ always queries RO on $(\mathsf{GID},\mathbf{x})$ whenever it makes a key query to some honest authority on $(\mathsf{GID},\mathbf{x})$. This can be assumed without loss of generality. Now one could additionally split the above set $S_{T,2}$ as per the GID-attribute pairs $(\mathsf{GID},\mathbf{x})$ as:

$$S_{T,2}^{(\mathsf{GID},\mathbf{x})} = \left\{ \Gamma\widetilde{C}_i^{(j)}\Delta_{\mathsf{GID},\mathbf{x}} : \begin{array}{c} i\in[L],\ d_{\mathbf{x},i}=1,\ \forall j \text{ s.t. } (j,\mathsf{GID},\mathbf{x}) \in \mathcal{Q}_{\mathsf{key}} \\ \text{where } \mathbf{d}_{\mathbf{x}} = (d_{\mathbf{x},1},\ldots,d_{\mathbf{x},L}) \text{ and} \\ \mathcal{Q}_{\mathsf{key}} \text{ denotes the set of key queries by } \mathcal{A} \end{array} \right\}$$
$$\bigcup \left\{ \Gamma\widetilde{C}_i^{(j)}\Delta_{\mathsf{GID},\mathbf{x}} : j\in S^*, i\in[L] \right\}$$

First, note that any *bad* query $f \in \mathbb{T}$ can now be represented as

$$f(\{W_i^{(j)}\}_{j\notin S^*,i\in[L]}, \{\Delta_{\mathsf{GID},\mathbf{x}}\}_{\mathsf{GID},\mathbf{x}}, \Gamma, \{\widetilde{C}_i^{(j)}\}_{j\in[N],i\in[L]}) = \sum_{(\mathsf{GID},\mathbf{x})\in\mathcal{Q}_{\mathsf{ro}}} \sum_{Z\in S_{T,2}^{(\mathsf{GID},\mathbf{x})}} a_Z Z.$$

Now we further classify *bad* queries into "types". We say that f is a Type-$(\mathsf{GID},\mathbf{x})$ bad query if:

$$\sum_{Z\in S_{T,2}^{(\mathsf{GID},\mathbf{x})}} a_Z Z \neq 0 \quad \text{and} \quad \sum_{Z\in S_{T,2}^{(\mathsf{GID},\mathbf{x})}} a_Z Z(\{\widetilde{c}_i^{(j)}\}_{j\in[N],i\in[L]}) = 0,$$

where $Z(\{\widetilde{c}_i^{(j)}\}_{j,i})$ denotes $Z(\{W_i^{(j)}\}_{j,i}, \{\Delta_{\mathsf{GID},\mathbf{x}}\}_{\mathsf{GID},\mathbf{x}}, \Gamma, \{\widetilde{c}_i^{(j)}\}_{j,i}) \in \mathbb{T}$ above.

Next, we claim the following.

Claim 7.21. If $f \in \mathbb{T}$ is a "bad" query, then there must exist $(\mathsf{GID},\mathbf{x}) \in \mathcal{Q}_{\mathsf{ro}}$ such that f is a "Type-$(\mathsf{GID},\mathbf{x})$ bad" query.

Proof. The proof of this claim follows along the lines of indistinguishability proof of hybrids 5.3 and 5.4 in [AY20]. The idea is that if f is a bad query, then there must exist a $(\mathsf{GID},\mathbf{x}) \in \mathcal{Q}_{\mathsf{ro}}$ satisfying $\sum_{Z\in S_{T,2}^{(\mathsf{GID},\mathbf{x})}} a_Z Z \neq 0$. Furthermore, we have that

$$\sum_{Z\in S_{T,2}^{(\mathsf{GID},\mathbf{x})}} a_Z Z(\{\widetilde{c}_i^{(j)}\}_{j\in[N],i\in[L]}) = -\sum_{(\mathsf{GID}',\mathbf{x}')\neq(\mathsf{GID},\mathbf{x})} \sum_{Z\in S_{T,2}^{(\mathsf{GID}',\mathbf{x}')}} a_Z Z(\{\widetilde{c}_i^{(j)}\}_{j\in[N],i\in[L]}).$$

However, the above is impossible unless the left hand side equals to 0 since any monomial in $S_{T,2}^{(\mathsf{GID},\mathbf{x})}$ never appears in $S_{T,2}^{(\mathsf{GID}',\mathbf{x}')}$ for $(\mathsf{GID}',\mathbf{x}') \neq (\mathsf{GID},\mathbf{x})$ even if we replace $\{\widetilde{C}_i^{(j)}\}$ with $\{\widetilde{c}_i^{(j)}\}$ and ignore the difference between the coefficients of the monomials. This is due to the fact that there is a $\delta_{\mathsf{GID},\mathbf{x}}$ variable on the left side that never appears on the right side. Thus, f must be Type-$(\mathsf{GID},\mathbf{x})$ bad query. □

To complete the proof of this lemma, we just need to show that adversary $\mathcal{A}$ never makes any "Type-$(\mathsf{GID},\mathbf{x})$ bad" query. To argue this, we prove the following:

Claim 7.22. For every GID-attribute pair $(\mathsf{GID},\mathbf{x}) \in \mathcal{Q}_{\mathsf{ro}}$, zero-test query number $t \in [Q_{\mathsf{zt}}]$, if BGG^+ satisfies semantic security with pseudorandom ciphertexts and $\log q = \omega(\log \lambda)$, then the probability that adversary $\mathcal{A}$'s first bad query is the t-th zero-test query and it is a "Type-$(\mathsf{GID},\mathbf{x})$ bad" query is at most $\mathsf{negl}'(\lambda)$ for some negligible function $\mathsf{negl}'(\cdot)$.

Proof. Now the GID-attribute pairs $(\mathsf{GID},\mathbf{x})$ queried by $\mathcal{A}$ can be further divided into two categories — (1) $F^*(\mathbf{x}) = 0$ (that is, attribute does not satisfy the predicate), or (2) there exists an index $j \notin S^*$ such that $\mathcal{A}$ does not make make a key generation query of the form $(j,\mathsf{GID},\mathbf{x})$ (that is, it does not query at least one honest authority for its key share for GID-attribute pair $(\mathsf{GID},\mathbf{x})$). (Refer to Definition 7.1 for admissible key queries.)

First, we argue that in case (2) as per above categorization, the claim is correct. This follows from the observation that if $\mathcal{A}$ makes its first bad query on t-th zero-test query and it is a "Type-$(\mathsf{GID},\mathbf{x})$ bad" query, and $\mathcal{A}$ did not make a key query on $(\mathsf{GID},\mathbf{x})$ to some honest authority say j^*, then in that case the every monomial $Z \in S_{T,2}^{(\mathsf{GID},\mathbf{x})}$ does not depend on $\widetilde{\mathbf{c}}^{(j^*)}$. Now since $\widetilde{\mathbf{c}}^{(j)}$ are sampled uniformly at random with the only constraint that $\sum_j \widetilde{\mathbf{c}}^{(j)} = \mathbf{c}$, thus the joint distributions of $\widetilde{\mathbf{c}}^{(j)}$ for $j \neq j^*$ (i.e., excluding $\widetilde{\mathbf{c}}^{(j^*)}$) is identical to that of a random distribution over $\mathbf{Z}_q^L$. By using uniformity of $\widetilde{\mathbf{c}}^{(j)}$ for $j \neq j^*$, we can conclude the argument that this event can happen only with probability at most $1/q$ since f is represented as a linear combination of $\{\Gamma\widetilde{C}_i^{(j)}\Delta_{\mathsf{GID},\mathbf{x}}\}_{i,j\neq j^*}$ and all entries of $\{\widetilde{c}_i^{(j)}\}_{i\in[L],j\neq j^*}$ are chosen uniformly at random. This completes the argument for case (2).

Otherwise, in case (1) as per above categorization, we have that $F^*(\mathbf{x}) = 0$. To argue negligible probability of this event, we define an intermediate hybrid game as follows. (This part of the proof is similar to that of [AY20, Lemma 4.7-4.8].) In the intermediate hybrid game, the challenger samples c_i as in the original scheme only for $i \in [L]$ such that $d_{\mathbf{x},i} = 1$, where $\mathbf{d}_{\mathbf{x}} = (d_{\mathbf{x},1},\ldots,d_{\mathbf{x},L})$ and $\mathbf{d}_{\mathbf{x}}$ is as defined in the construction for attribute $\mathbf{x}$. The game is still well-defined since the only place in the game where we need the information of $\mathbf{c}$ is to check that t-th zero-test query is a "Type-$(\mathsf{GID},\mathbf{x})$ bad" query, and we only need c_i for $i \in [L]$ such that $d_{\mathbf{x},i} = 1$. Basically given these, the reduction can sample all the $\widetilde{c}_i^{(j)}$ terms appropriately. (Here appropriately means that for $j \in S^*$, it samples all of them to be uniformly random, while for $j \notin S^*$ it only needs to simulate half of the $\widetilde{c}_i^{(j)}$ terms depending on $\mathbf{x}$ which are computable given half of the c_i terms.)

We start by claiming that this intermediate hybrid is identically distributed to the previous game. This follows from the fact that $S_{T,2}^{(\mathsf{GID},\mathbf{x})}$ only contains terms $\Gamma\widetilde{C}_i^{(j)}\Delta_{\mathsf{GID},\mathbf{x}}$ for $j \notin S^*, i \in [L], d_{\mathbf{x},i} = 1$. Thus, since c_i is N-out-of-N secret shared into $\widetilde{c}_i^{(j)}$, thus the challenger only needs to sample c_i for $i \in [L]$ such that $d_{\mathbf{x},i} = 1$ in both the original game as well as the intermediate hybrid game. This proves indistinguishability of the intermediate hybrid game with the original game.

Next, once the challenger only needs to sample c_i for $i \in [L]$ such that $d_{\mathbf{x},i} = 1$ and given that $F^*(\mathbf{x}) = 0$, thus we could switch all these c_i terms to be sampled uniformly at random instead by relying semantic security (with pseudorandom ciphertexts) of BGG^+. This reduction is identical to that of [AY20, Lemma 4.8]. Finally, once c_i for $i \in [L]$ such that $d_{\mathbf{x},i} = 1$ are sampled uniformly at random, then as in the proof of case (2) previously, we could show that a bad query happens with probability at most $1/q$. This completes the argument in case (1) too, and hence this completes the proof. □

Finally, using the above claim with a simple union bound over the total number of zero-test queries and key queries, we get that Games 6 and 7 are negligibly far apart. Since, the adversary's advantage is Game 7 is clearly zero as it contains no information about the challenge bit, thus the theorem follows.

□

8 New Primitives Predicted by MPFE

In this section, we define new primitives that can be seen as special cases of multi-party functional encryption and have natural, compelling applications. We also provide a high level sketch of a generic construction of our new reputation point based encryption described in Section 1.5 from a multi-authority ABE scheme.

Reputation Point Based Encryption: Below we sketch the constructions in two different setup modes which leads to different trade-offs in the type of collusion security and aggregation functionality.

Centralized setup with bounded aggregation. In the centralized setup setting, the central authority on input the upper bound, N, on the number of user key aggregations samples N pairs of MA-ABE authority public-secret key pairs $(\mathsf{PK}_i, \mathsf{MSK}_i)$, and sets the the master public and secret keys as the sequence of these N keys that is $(\mathsf{PK}_i)_i, (\mathsf{MSK}_i)_i$. Now to generate a key for tag T and points v, the authority starts by sampling a random label L and generates N MA-ABE partial decryption keys by running the MA-ABE key generation algorithm with respect to each key MSK_i with global identifier $\mathsf{GID} = T$ and partial attribute vector (T, L, v). An encryptor simply hides the message under the policy P which on input a sequence $(T_i, L_i, v_i)_i$ checks that all tags match the encrypted tag T, all labels are distinct, and the user points sum to a value larger than the encrypted threshold w. Note that here we are using randomly sampled labels in each user's key for ensuring that a single user can not combine two components of its own key, while N distinct users can always combine one key from their sequence of N MA-ABE keys with all but negligible probability. At a high level, the above idea seems to work for any MA-ABE scheme but one needs to be careful about the notion of security the underlying MA-ABE scheme must satisfy as now in this system we ask each authority to possibly generate multiple keys for the same global identifier.

Decentralized setup with security in the erasure model. Note that one could try to make the above construction fully decentralized by asking each user to locally sample its own public-secret key pair $(\mathsf{PK}, \mathsf{MSK})$ for a MA-ABE scheme, and then the user locally generates a partial predicate key $\mathsf{SK}_{T,v}$ for attribute (T, v). Now each user must *delete* its master secret key MSK, and it outputs PK as its public key and $\mathsf{SK}_{T,v}$ as its decryption key. An encryptor simply takes as input the list of public keys of all users which it possibly wants to encrypt to, and defines the corresponding policy circuit as follows — either the attribute string is not specified, or it the specified tag matches the encrypted tag T, and the user points sum to a value larger than the encrypted threshold w. To make this work, we need the MA-ABE scheme to support the concept of empty key (or zero-keys) that is the partial decryption key for all zeros attribute is simply the empty string, and the fact that an attacker never corrupts the master secret key of the underlying MA-ABE system for any user (that is, the random coins of every user's setup are erased). We leave further analysis of above approaches for future work.

Other New Primitives for Future Work. Providing more constructions is beyond the scope of the present work, so we outline syntax of some new primitives here and leave instantiation to future work.

1. *Generalizing* MIFE, MCFE *and* PHFE*:* In many cases it is useful to combine ciphertexts from multiple users if a more complex policy is satisfied by their public labels: for instance, we may want to compute on all ciphertexts that were generated during a specified time interval. Here, the labels may be set as the timestamp at which a ciphertext was generated. The above example motivates defining the notion of *partially hiding* multi-input FE, where the input of party P_i is the pair $(\mathsf{lab}_i, \mathbf{x}_i)$ where lab_i is public and $\mathbf{x}_i$ is private. Given encryptions of $(\mathsf{lab}_i, \mathbf{x}_i)$ for $i \in [n]$ and a function key for $f = (f_1, f_2)$, if $f_1(\mathsf{lab}_1, \ldots, \mathsf{lab}_n) = 1$, decryption must output $f_2(\mathbf{x}_1, \ldots, \mathbf{x}_n)$. Note that this notion generalizes MIFE, MCFE as well as partially hiding FE [GVW12, AJL+19].

2. *Generalizing Partial Key Combinations:* We may permit more meaningful ways of combining keys than those that have been studied before. For instance, in the context of an investigation, multiple

key authorities such as hospital, police station and bank may wish to filter suspects in an encrypted database based on different criteria, for which they issue different keys. Let us say that the input $\mathbf{x}$ contains a list of user records. Then, each authority gives a key that takes an input list of records, tests which records satisfy its criteria and erases the rest. Authority i provides a key for function f_i, and given ciphertext for $\mathbf{x}$, decryption returns $f_1 \circ \ldots f_n(\mathbf{x})$.

References

[ABB10a] Shweta Agrawal, Dan Boneh, and Xavier Boyen. Efficient lattice (H)IBE in the standard model. In *EUROCRYPT*, pages 553–572, 2010.

[ABB10b] Shweta Agrawal, Dan Boneh, and Xavier Boyen. Lattice basis delegation in fixed dimension and shorter-ciphertext hierarchical IBE. In *CRYPTO*, pages 98–115, 2010.

[ABDCP15] Michel Abdalla, Florian Bourse, Angelo De Caro, and David Pointcheval. Simple functional encryption schemes for inner products. In *PKC*, 2015.

[ABG19] Michel Abdalla, Fabrice Benhamouda, and Romain Gay. From single-input to multi-client inner-product functional encryption. In *ASIACRYPT*, 2019.

[ABKW19] Michel Abdalla, Fabrice Benhamouda, Markulf Kohlweiss, and Hendrik Waldner. Decentralizing inner-product functional encryption. In *PKC*, 2019.

[ACF+20] Shweta Agrawal, Michael Clear, Ophir Frieder, Sanjam Garg, Adam O'Neill, and Justin Thaler. Ad hoc multi-input functional encryption. In *ITCS 2020*, 2020.

[ACGU20] Michel Abdalla, Dario Catalano, Romain Gay, and Bogdan Ursu. Inner-product functional encryption with fine-grained access control. In *Asiacrypt*, 2020.

[AJ15] Prabhanjan Ananth and Abhishek Jain. Indistinguishability obfuscation from compact functional encryption. In *CRYPTO 2015*, 2015.

[AJL+19] Prabhanjan Ananth, Aayush Jain, Huijia Lin, Christian Matt, and Amit Sahai. Indistinguishability obfuscation without multilinear maps: io from lwe, bilinear maps, and weak pseudorandomness. In *Crypto*, 2019.

[ALS16] Shweta Agrawal, Benoit Libert, and Damien Stehle. Fully secure functional encryption for linear functions from standard assumptions, and applications. In *Crypto*, 2016.

[AY20] Shweta Agrawal and Shota Yamada. Optimal broadcast encryption from pairings and lwe. In *Proc. of EUROCRYPT*, 2020.

[BCFG17] Carmen Elisabetta Zaira Baltico, Dario Catalano, Dario Fiore, and Romain Gay. Practical functional encryption for quadratic functions with applications to predicate encryption. In *CRYPTO*, 2017.

[BCG+17] Zvika Brakerski, Nishanth Chandran, Vipul Goyal, Aayush Jain, Amit Sahai, and Gil Segev. Hierarchical functional encryption. In *ITCS 2017*, 2017.

[BF01] Dan Boneh and Matthew K. Franklin. Identity-based encryption from the weil pairing. In *CRYPTO*, pages 213–229, 2001.

[BFF+14] Gilles Barthe, Edvard Fagerholm, Dario Fiore, John C. Mitchell, Andre Scedrov, and Benedikt Schmidt. Automated analysis of cryptographic assumptions in generic group models. In *CRYPTO*, 2014.

[BGG+14] Dan Boneh, Craig Gentry, Sergey Gorbunov, Shai Halevi, Valeria Nikolaenko, Gil Segev, Vinod Vaikuntanathan, and Dhinakaran Vinayagamurthy. Fully key-homomorphic encryption, arithmetic circuit ABE and compact garbled circuits. In *EUROCRYPT*, 2014.

[BJK15] Allison Bishop, Abhishek Jain, and Lucas Kowalczyk. Function-hiding inner product encryption. In *Asiacrypt*, 2015.

[BLP+13] Zvika Brakerski, Adeline Langlois, Chris Peikert, Oded Regev, and Damien Stehlé. Classical hardness of learning with errors. In *STOC*, STOC '13, 2013.

[BSW11] Dan Boneh, Amit Sahai, and Brent Waters. Functional encryption: Definitions and challenges. In *TCC*, pages 253–273, 2011.

[BV15] Nir Bitansky and Vinod Vaikuntanathan. Indistinguishability obfuscation from functional encryption. *FOCS*, 2015:163, 2015.

[BWY11] Mihir Bellare, Brent Waters, and Scott Yilek. Identity-based encryption secure against selective opening attack. In *Theory of Cryptography Conference*, 2011.

[CC09] Melissa Chase and Sherman S. M. Chow. Improving privacy and security in multi-authority attribute-based encryption. In *ACM CCS 2009*, 2009.

[CDSG+18a] Jérémy Chotard, Edouard Dufour-Sans, Romain Gay, Duong Hieu Phan, and David Pointcheval. Decentralized multi-client functional encryption for inner product. In *Asiacrypt*, 2018.

[CDSG+18b] Jérémy Chotard, Edouard Dufour-Sans, Romain Gay, Duong Hieu Phan, and David Pointcheval. Multi-client functional encryption with repetition for inner product. Cryptology ePrint Archive, Report 2018/1021, 2018.

[CDSG+20] Jérémy Chotard, Edouard Dufour-Sans, Romain Gay, Duong Hieu Phan, and David Pointcheval. Dynamic decentralized functional encryption. In *Crypto*, 2020.

[Cha07] Melissa Chase. Multi-authority attribute based encryption. In *TCC*, 2007.

[CHKP10] David Cash, Dennis Hofheinz, Eike Kiltz, and Chris Peikert. Bonsai trees, or how to delegate a lattice basis. In *EUROCRYPT*, pages 523–552, 2010.

[CZY19] Yuechen Chen, Linru Zhang, and Siu-Ming Yiu. Practical attribute based inner product functional encryption from simple assumptions. Cryptology ePrint Archive, 2019.

[DOT18] Pratish Datta, Tatsuaki Okamoto, and Katsuyuki Takashima. Adaptively simulation-secure attribute-hiding predicate encryption. Cryptology ePrint Archive, Report 2018/1093, 2018.

[GGG+14] Shafi Goldwasser, S. Dov Gordon, Vipul Goyal, Abhishek Jain, Jonathan Katz, Feng-Hao Liu, Amit Sahai, Elaine Shi, and Hong-Sheng Zhou. Multi-input functional encryption. In *Eurocrypt*, 2014.

[GKL+13] S. Dov Gordon, Jonathan Katz, Feng-Hao Liu, Elaine Shi, and Hong-Sheng Zhou. Multi-input functional encryption. Cryptology ePrint Archive, Report 2013/774, 2013.

[GPSW06] Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent Waters. Attribute-based encryption for fine-grained access control of encrypted data. In *CCS*, 2006.

[GPV08] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In *STOC*, pages 197–206, 2008.

[GV15] Sergey Gorbunov and Dhinakaran Vinayagamurthy. Riding on asymmetry: Efficient ABE for branching programs. In *ASIACRYPT*, 2015.

[GVW12] Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Functional encryption with bounded collusions from multiparty computation. In *CRYPTO*, 2012.

[GVW15] Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Predicate encryption for circuits from lwe. In *Crypto*, 2015.

[KSW08] Jonathan Katz, Amit Sahai, and Brent Waters. Predicate encryption supporting disjunctions, polynomial equations, and inner products. In *EUROCRYPT*, pages 146–162, 2008.

[KW19] Venkata Koppula and Brent Waters. Realizing chosen ciphertext security generically in attribute-based encryption and predicate encryption. In *CRYPTO 2019*, 2019.

[LŢ19] Benoît Libert and Radu Ţiţiu. Multi-client functional encryption for linear functions in the standard model from lwe. In *Asiacrypt*, 2019.

[LW11] Allison Lewko and Brent Waters. Decentralizing attribute-based encryption. In *Proceedings of EuroCrypt*, 2011.

[Mau05] Ueli Maurer. Abstract models of computation in cryptography. In *IMA Int. Conf.*, volume 3796 of *Lecture Notes in Computer Science*, pages 1–12. Springer, 2005.

[MJ18] Yan Michalevsky and Marc Joye. Decentralized policy-hiding ABE with receiver privacy. In *ESORICS*, 2018.

[MP12] Daniele Micciancio and Chris Peikert. Trapdoors for lattices: Simpler, tighter, faster, smaller. In *EUROCRYPT*, 2012.

[Reg09] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. *J.ACM*, 56(6), 2009. extended abstract in STOC'05.

[Sha84] Adi Shamir. Identity-based cryptosystems and signature schemes. In *Workshop on the theory and application of cryptographic techniques*, 1984.

[Sho97] Victor Shoup. Lower bounds for discrete logarithms and related problems. In *Eurocrypt*, 1997.

[SW05] Amit Sahai and Brent Waters. Fuzzy identity-based encryption. In *Eurocrypt*, 2005.

[Wat09] Brent Waters. Dual system encryption: Realizing fully secure ibe and hibe under simple assumptions. In *Annual International Cryptology Conference*, 2009.

Appendix

A Feasibility for MPFE for General Circuits

In this section we discuss the feasibility of general MPFE. Since the framework of MPFE is very general and supports all existing constructions in the literature (that we are aware of), feasibility varies widely depending on the properties desired, such as whether the setup algorithm must be local, centralized or interactive, whether and encryption keys must be public or private and such others. Since the case of centralized setup has been most widely studied in the literature and it is evident that feasibility results for local and interactive setup also apply for a centralized setup, we focus on these below.

First, we provide some definitions needed for our constructions.

A.1 Definitions

A.1.1 Multi-Input Functional Encryption

An *n-input FE scheme* [GGG+14] MIFE for a message space $\{\mathcal{M}_\lambda\}_{\lambda\in\mathbb{N}}$ and a functionality $\{\mathcal{F}_\lambda\}_{\lambda\in\mathbb{N}}$, where for each $\lambda \in \mathbb{N}$, each $f \in \mathcal{F}_\lambda$ is a (description of a) function on $(\mathcal{M}_\lambda)^n$, is given by a set of algorithms with the following syntax:

- $\mathsf{MIFE.Setup}(1^\lambda, 1^n)$: A PPT algorithm taking the security parameter λ and number of users n, and outputting the master secret key MSK and encryption keys $(\mathsf{EK}_1, \ldots, \mathsf{EK}_n)$.
- $\mathsf{MIFE.KeyGen}(\mathsf{MSK}, f)$: A PT algorithm taking a master secret key MSK, a function $f \in \mathcal{F}_\lambda$ and outputting a corresponding decryption key SK_f.
- $\mathsf{MIFE.Enc}(\mathsf{EK}, \mathbf{x})$: A PPT algorithm taking an encryption key EK and a message $\mathbf{x} \in \mathcal{M}_\lambda$, and outputting a ciphertext c.
- $\mathsf{MIFE.Dec}(\mathsf{SK}_f, (c_1, \ldots, c_n))$: A PT algorithm taking decryption key SK_f and vector of ciphertexts $(c_1, \ldots, c_n)$, and outputting a string y.

Correctness We say that MIFE is *correct* if for all $\lambda, \in \mathbb{N}$, $\mathbf{x}_1 \ldots \mathbf{x}_n \in \mathcal{M}_\lambda$ and $f \in \mathcal{F}_\lambda$

$$\Pr\left[y = f(\mathbf{x}_1, \cdots, \mathbf{x}_n) : \begin{array}{l} ((\mathsf{EK}_1, \ldots, \mathsf{EK}_n), \mathsf{MSK}) \leftarrow \mathsf{MIFE.Setup}(1^\lambda) \\ c_i \leftarrow \mathsf{MIFE.Enc}(\mathsf{EK}_i, \mathbf{x}_i) \quad \forall i \in [n] \\ \mathsf{SK}_f \leftarrow \mathsf{MIFE.KeyGen}(\mathsf{MSK}, f) \\ y \leftarrow \mathsf{MIFE.Dec}(\mathsf{SK}_f, (c_1, \ldots, c_n)) \end{array} \right] = 1 \, .$$

We remark that our formulation of MIFE assumes that the senders (as in our application an encryptor is referred to as a sender or source) are ordered.

Indistinguishability-Based Security. For an n-input FE scheme MIFE as above and adversary $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1, \mathcal{A}_2)$, consider the experiment in Figure A.1.

Experiment $\mathbf{IND}_{\mathcal{A}}^{\mathsf{MIFE}}(1^\lambda)$
$(I, {}^*) \leftarrow \mathcal{A}_0(1^\lambda)$
$b \leftarrow \{0,1\}$
$((\mathsf{EK}_1, \ldots, \mathsf{EK}_n), \mathsf{MSK})$
$\leftarrow \mathsf{MIFE.Setup}(1^\lambda)$
${}^* \leftarrow \mathcal{A}_1^{\mathcal{O}_{\mathsf{Enc}}(\cdot,\cdot,\cdot), \mathcal{O}_{\mathsf{Keygen}}(\cdot)}({}^*)$
$b' \leftarrow \mathcal{A}_2^{\mathcal{O}_{\mathsf{Enc}}(\cdot,\cdot,\cdot), \mathcal{O}_{\mathsf{Keygen}}(\cdot)}((\mathsf{EK}_i)_{i\in I}, {}^*)$
Return $(b = b')$

Oracle $\mathcal{O}_{\mathsf{Enc}}(i, \mathbf{x}_0, \mathbf{x}_1)$
If $\mathbf{x}_0, \mathbf{x}_1 \in \mathcal{M}_\lambda$ and $|\mathbf{x}_0| = |\mathbf{x}_1|$
$c_b \leftarrow \mathsf{MIFE.Enc}(\mathsf{EK}_i, \mathbf{x}_b)$
Return c_b
Else Return $\perp$

Oracle $\mathcal{O}_{\mathsf{Keygen}}(f)$
If $f \in \mathcal{F}_\lambda$
$\mathsf{SK}_f \leftarrow \mathsf{MIFE.KeyGen}(\mathsf{MSK}, f)$
Return SK_f
Else return $\perp$

Figure A.1: Experiment for IND-security of standard MIFE.

We call $\mathcal{A}$ *legitimate* if for all $\lambda \in \mathbb{N}$, in all transcripts $\mathbf{IND}_{\mathcal{A}}^{\mathsf{MIFE}}(1^\lambda)$ it holds that for every key generation query f there does not exist two sequences $(y_{1,0}, \cdots, y_{n,0})$ and $(y_{1,1}, \cdots, y_{n,1})$ such that

$$f(y_{1,0}, \cdots, y_{n,0}) \neq f(y_{1,1}, \cdots, y_{n,1})$$

and for every $j \in [n]$

- $j \in I$, *i.e.* j is corrupted (so there is no restriction on $y_{j,0}, y_{j,1}$ above), or
- there is an encryption query $(j, \mathbf{x}_0, \mathbf{x}_1)$ such that $y_{j,0} = \mathbf{x}_0$ and $y_{j,1} = \mathbf{x}_1$.

We assume adversaries are legitimate unless otherwise stated. We call $\mathcal{A}$ *passive* if $I = \emptyset$. We call $\mathcal{A}$ *selective* if $\mathcal{A}_2$ makes no queries. We say that MIFE is IND-secure if for any PPT adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

$$\Pr\left[\mathbf{IND}_{\mathcal{A}}^{\mathsf{MIFE}}(1^\lambda) \text{ outputs } 1\right] \leq 1/2 + \mathsf{negl}(\lambda).$$

A.1.2 Functional Encryption

A functional encryption scheme, denoted as FE [BSW11], is a tuple of algorithms $\mathsf{FE} = (\mathsf{FE.Setup}, \mathsf{FE.KeyGen}, \mathsf{FE.Enc}, \mathsf{FE.Dec})$ for a message space $\{\mathcal{M}_\lambda\}_{\lambda\in\mathbb{N}}$ and a functionality $\{\mathcal{F}_\lambda\}_{\kappa\in\mathbb{N}}$, where for each $\lambda \in \mathbb{N}$, each $f \in \mathcal{F}_\lambda$ is a (description of a) function on $\mathcal{M}_\lambda$. The syntax is the same as for a 1-input MIFE scheme where $\mathsf{EK}_1 = \mathsf{MSK}$ and $I = \emptyset$. The correctness requirement remains the same, as well the notion of indistinguishability based security (which we refer to as "message privacy").

Function Privacy. We additionally define the notion of function privacy as follows. For an FE scheme FE as above and adversary $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1, \mathcal{A}_2)$, consider the experiment in Figure A.2.

Experiment $\mathbf{FP}_{\mathcal{A}}^{\mathsf{FE}}(1^\kappa)$
$b \leftarrow \{0,1\}$
$\mathsf{MSK} \leftarrow \mathsf{FE.Setup}(1^\lambda)$
$b' \leftarrow \mathcal{A}_2^{\mathcal{O}_{\mathsf{Enc}}(\cdot),\mathcal{O}_{\mathsf{Keygen}}(\cdot,\cdot)}(1^\lambda)$
Return $(b = b')$

Oracle $\mathcal{O}_{\mathsf{Enc}}(\mathbf{x})$
If $\mathbf{x} \in \mathcal{M}_\kappa$
$c \leftarrow \mathsf{MIFE.Enc}(\mathsf{MSK}, \mathbf{x})$
Return c
Else Return $\perp$

Oracle $\mathcal{O}_{\mathsf{Keygen}}(f_0, f_1)$
If $f_0, f_1 \in \mathcal{F}_\lambda$
$\mathsf{SK}_{f_b} \leftarrow \mathsf{MIFE.KeyGen}(\mathsf{MSK}, f_b)$
Return SK_{f_b}
Else return $\perp$

Figure A.2: Experiment for FP-security of FE.

We call $\mathcal{A}$ *legitimate* if for all $\lambda \in \mathbb{N}$, in all transcripts $\mathbf{FP}_{\mathcal{A}}^{\mathsf{FE}}(1^\kappa)$ it holds that for every key generation query f_0, f_1 there does not exist an encryption query $\mathbf{x} \in \mathcal{M}_\kappa$ such that $f_0(\mathbf{x}) \neq f_1(\mathbf{x})$. We assume adversaries are legitimate unless otherwise stated. We say that FE is function-private FE if for any PPT adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

$$\Pr\left[\mathbf{FP}_{\mathcal{A}}^{\mathsf{FE}}(1^\lambda) \text{ outputs } 1\right] \leq 1/2 + \mathsf{negl}(\lambda).$$

A.1.3 Two-Round MPC

A *2-round MPC protocol* MPC for message-space $\{\mathcal{M}_\lambda\}_{\lambda\in\mathbb{N}}$ and functionality $\{\mathcal{F}_\lambda\}_{\lambda\in\mathbb{N}}$ where for each $\lambda \in \mathbb{N}$ each $f \in \mathcal{F}_\lambda$ is a function on $(\mathcal{M}_\lambda)^n$ for some n, consists of three algorithms with the following syntax:

- $\mathsf{RunRoundOne}(1^\lambda, 1^n, f, i, x)$: A PPT algorithm taking the security parameter λ, number of users n, a (description of a) function $f \in \mathcal{F}_\lambda$ of arity n, an index $i \in [n]$, an input $x \in \mathcal{M}_\lambda$, and outputting a first protocol message $\rho^{(1)}$ and secret $\mathfrak{s}$.
- $\mathsf{RunRoundTwo}(\mathfrak{s}, (\rho_1^{(1)}, \ldots, \rho_n^{(1)}))$: A PPT algorithm taking a secret $\mathfrak{s}$ and the first protocol message for all n parties $\rho_1^{(1)}, \ldots, \rho_n^{(1)}$, and outputting a second protocol message $\rho^{(2)}$.

- $\mathsf{ComputeResult}$: A PT algorithm taking as input the n second-round protocol messages $\rho_1^{(2)}, \ldots, \rho_n^{(2)}$ for each party and outputting a value y.

Correctness. We say that MPC is *correct* if for all $\lambda, n, \in \mathbb{N}$, $\mathbf{x}_1 \ldots \mathbf{x}_n \in \mathcal{M}_\lambda$ and $f \in \mathcal{F}_\lambda$

$$\Pr\left[y = f(\mathbf{x}) : \begin{array}{ll} (\rho_i^{(1)}, \mathfrak{s}_i) \leftarrow \mathsf{RunRoundOne}(1^\lambda, 1^n, f, i, \mathbf{x}_i) & \forall i \in [n] \\ \rho_i^{(2)} \leftarrow \mathsf{RunRoundTwo}(\mathfrak{s}_i, (\rho_1^{(1)}, \ldots, \rho_n^{(1)})) & \forall i \in [n] \\ y \leftarrow \mathsf{ComputeResult}(\rho_1^{(2)}, \ldots, \rho_n^{(2)}) & \end{array} \right] = 1 \, .$$

Remark A.1. The above definition of two-round MPC is without setup (*i.e.*, a CRS). We also consider the case that there is an additional algorithm CRSGen taking 1^λ and outputting a common reference string CRS that is input to the remaining algorithms. We call this two-round MPC *in the CRS model*.

We call MPC *input-rerunnable* (resp. *function-rerunnable*) if it is *input-delayed* (resp. *function-delayed*) and if $\mathsf{RunRoundTwo}$ can be executed multiple times with different input choices (resp. function choices) while still preserving the security properties of the MPC protocol.

Security. Let MPC be a 2-round MPC protocol as above. Let Coins be the coin-space for the protocol. For an adversary $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ and simulator SS, consider the experiments in Figure A.3. We say that $\mathcal{A}$ is *passive* (aka. semi-honest) of $I = \emptyset$. We say that MPC is SIM-secure if for any PPT adversary $\mathcal{A}$ there is a stateful PPT simulator $SS = (\widetilde{\mathsf{CRSGen}}, \widetilde{\mathsf{Extract}}, \widetilde{\mathsf{Sim}})$ such that $\mathbf{REAL}_{\mathcal{A}}^{\mathsf{MPC}}(\cdot)$ and $\mathbf{IDEAL}_{\mathcal{A},SS}^{\mathsf{MPC}}(\cdot)$ are computationally indistinguishable. Note that simulation of the first-round protocol messages for the

Experiment $\mathbf{REAL}_{\mathcal{A}}^{\mathsf{MPC}}(1^\kappa)$

(**Optional**) $\mathsf{crs} \leftarrow \mathsf{CRSGen}(1^\lambda)$
$(1^n, I, f, (x_i)_{i \notin I}) \leftarrow \mathcal{A}_0(1^\kappa)$
  // $f \in \mathcal{F}_\lambda$ of arity n, $x_i \in \mathcal{M}_\lambda$
$((\rho_i^{(1)})_{i \in I}, \mathsf{st}) \leftarrow \mathcal{A}_1(n, I, f)$
For $i \notin I$ do:
  $r_i \leftarrow \mathsf{Coins}(1^\kappa)$
  $(\rho_i^{(1)}, \mathfrak{s}_i)$
  $\leftarrow \mathsf{RunRoundOne}(1^\kappa, 1^n, f, i, x_i; r_i)$
For $i \notin I$ do:
  $\rho_i^{(2)}$
  $\leftarrow \mathsf{RunRoundTwo}(\mathfrak{s}_i, (\rho_1^{(1)}, \ldots, \rho_n^{(1)}); r_i)$
$\alpha \leftarrow \mathcal{A}_2(\mathsf{st}, (\rho_i^{(1)}, \rho_i^{(2)})_{i \notin I})$
Return α

Experiment $\mathbf{IDEAL}_{\mathcal{A},SS}^{\mathsf{MPC}}(1^\kappa)$

(**Optional**) $\mathsf{crs} \leftarrow \widetilde{\mathsf{CRSGen}}(1^\lambda)$
$(1^n, I, f, (x_i)_{i \notin I}) \leftarrow \mathcal{A}_0(1^\kappa)$
  // $f \in \mathcal{F}_\lambda$ of arity n, $x_i \in \mathcal{M}_\lambda$
$((\rho_i^{(1)})_{i \in I}, \mathsf{st}) \leftarrow \mathcal{A}_1(n, I, f)$
$x_i \leftarrow \widetilde{\mathsf{Extract}}(\rho_i^{(1)}) \quad \forall i \in I$
$(\rho_i^{(1)}, \rho_i^{(2)})_{i \notin I}$
$\leftarrow \widetilde{\mathsf{Sim}}((x_i)_{i \in I}, f(x_1, \ldots, x_n))$
$\alpha \leftarrow \mathcal{A}_2(\mathsf{st}, (\rho_i^{(1)}, \rho_i^{(2)})_{i \notin I})$
Return α

Figure A.3: Experiments for SIM-security of two-round MPC.

honest parties are independent of the inputs, so for convenience and ease of presentation we will partition the algorithm $\widetilde{\mathsf{Sim}}$ into two algorithms: $\widetilde{\mathsf{Sim}_1}$ and $\widetilde{\mathsf{Sim}_2}$, defined as follows:

- $\widetilde{\mathsf{Sim}_1}() \mapsto (\rho_i^{(1)})_{i \notin I}$: Outputs the first-round protocol messages for the honest parties.
- $\widetilde{\mathsf{Sim}_2}((x_i)_{i \in I}, y) \mapsto (\rho_i^{(2)})_{i \notin I}$: On input the inputs of the corrupted parties along with the target output value of the protocol, $\widetilde{\mathsf{Sim}_2}$ outputs the second-round protocol messages for the honest parties.

Input/Function-Rerunnability. For simplicity, the definition in Figure A.3 does not capture input/function-rerunnability. It is straightforward to see how the definition can be extended. For example in the case of

function-rerunnability (the situation is analogous for input-rerunnability where inputs and functions are swapped), the changes to the definition are (1) $\mathcal{A}_0$ outputs a set of functions $\{f_i\}$ instead of a single function, (2) in the real experiment $\mathsf{RunRoundTwo}$ is executed for each function, (3) in the ideal experiement $\widetilde{\mathsf{Sim}_2}$ is called for each function, and (4) the complete set of second-round protocol messages for all functions is given to $\mathcal{A}_2$.

A.2 Constructions

Local Setup. In multi-party FE schemes, local setup is highly desirable, since it overcomes the key escrow problem which is one of the main drawbacks of present day FE constructions. In this setting, each user i runs the setup algorithm independently, and obtains her own public key PK_i and master secret key MSK_i. No communication or co-ordination is required between the users. In the symmetric key setting, the user encrypts her input using her master key MSK_i, while in the public key setting, anyone may encrypt using the public key(s) PK_i.

Constructions of multi-party FE with local setup have been notoriously hard to build. For general circuits and in the symmetric key setting, the primitive of adhoc multi-input functional encryption (aMIFE) recently proposed by Agrawal et al. [ACF+20] comes close to a general feasibility result in our model. Adhoc MIFE enables multiple parties to run local, independent setup algorithms, and generate their own private master keys as well as corresponding public keys. To encrypt data, each party (say i) uses its private master key to compute a ciphertext for any message of its choice (say x_i), and to issue function specific keys, the party uses the same master key to compute a partial function key corresponding to some function (say f). These ciphertexts and partial keys are sent to the decryptor who can aggregate them and compute $f(x_1, \ldots, x_n)$ as long as all the partial decryption keys correspond to the same function f. The authors provide a construction of aMIFE for circuits by using the standard notion of MIFE, a single input FE and a special purpose two round MPC.

There are some important differences in the formulation of aMIFE and our MPFE:

1. In aMIFE, users each provide partial keys which may be combined when these keys refer to the *same* function. In contrast, our notion allows different partial keys to refer to different functions which may be combined in diverse ways as codified by the function Agg_y.
2. The encryptors and key authorities are the same in aMIFE, and have the same master key MSK. This is not general enough to capture even regular single user FE, where the key authority and encryptor are different entities that may not share the master key, for instance in the public key setting. To capture these notions, we defined our framework to allow for users and key authorities to have separate encryption and master keys, where the encryption keys may be public.

Below, we provide an MPFE construction using aMIFE as a black-box – this transformation works in the symmetric key setting. The more general construction supporting the public key setting requires an interactive setup, and is discussed next. The construction for symmetric key MPFE with local setup using aMIFE is as follows:

1. Setup in MPFE is run with $\mathsf{mode} = \mathsf{Local}$. Every party in the MPFE protocol, whether a key generator or encryptor runs the aMIFE setup and obtains its own public and private key. If the party is a key generator, it sets this as its MSK, if it is an encryptor, it sets it as EK.
2. Next, each party in MPFE, whether encryptor or key generator, computes an aMIFE *ciphertext* for its input x_i or y_j. Thus, we obtain n_x partial ciphertexts for inputs x_i, $i \in [n_x]$ and n_y partial keys for inputs y_j, where $j \in [n_y]$.
3. Each party, whether encryptor or key generator, also computes an aMIFE partial key for the function $\mathcal{U}(\mathsf{Agg}_x(\cdot), \mathsf{Agg}_y(\cdot))$.
4. The decryptor/aggregator receives the partial keys and ciphertexts and performs aMIFE decryption to compute $\mathcal{U}(\mathsf{Agg}_x(x_1, \ldots, x_{n_x}), \mathsf{Agg}_y(y_1, \ldots, y_{n_y}))$.

Correctness and security follow by those of aMIFE. The decryptor obtains aMIFE ciphertexts for x_i where $i \in [n_x]$ and y_j where $j \in [n_y]$ along with $n_x + n_y$ aMIFE partial keys for the function $\mathcal{U}(\mathsf{Agg}_x(\cdot), \mathsf{Agg}_y(\cdot))$. This enables the decryptor to recover $\mathcal{U}(\mathsf{Agg}_x(x_1, \ldots, x_{n_x}), \mathsf{Agg}_y(y_1, \ldots, y_{n_y})$ as desired. Security follows directly from the security of aMIFE.

Interactive Setup. A standard MIFE scheme can be modified to support distributed keys by replacing the setup algorithm with an interactive protocol. This makes use of a function delayed, rerunnable two round MPC protocol similar to the construction of aMIFE provided by [ACF+20]. Note that the limitation of interactive setup is mitigated by the fact that it must only be run once.

We provide a general feasibility result for the case of interactive setup below.

1. The MPFE setup protocol does the following:
   (a) Invoke $\left(\mathsf{MIFE.MSK}, \mathsf{MIFE.EK}_1, \ldots, \mathsf{MIFE.EK}_{n_x+n_y}\right) \leftarrow \mathsf{MIFE.Setup}(1^\lambda, 1^{n_x+n_y})$.
   (b) Using an N out of N secret sharing scheme, compute shares $\mathsf{MIFE.MSK}_i$ of the MIFE.MSK and output $\mathsf{MIFE.MSK}_i$ to each key authority i for $i \in [n_y]$.
   (c) Using $\mathsf{MIFE.MSK}_i$ as input, run the first round of function-delayed, rerunnable two round MPC protocol as $\mathsf{MPC.RunRoundOne}(1^\lambda, 1^{n_y}, i, \mathsf{MIFE.MSK}_i)$ to obtain first protocol message $\rho_i^{(1)}$ and secret $\mathfrak{s}_i$ for each party $i \in [n_y]$.
   (d) For each key authority $i \in [n_y]$, output $\mathsf{PK}_i = \rho_i^{(1)}$ as the public key and $\mathsf{MSK}_i = (\mathsf{MIFE.EK}_{n_x+i}, \mathsf{MIFE.MSK}_i, \mathfrak{s}_i)$ as the master secret key.
   (e) For each encryptor $j \in [n_x]$, output the encryption key $\mathsf{EK}_j = \mathsf{MIFE.EK}_j$ and $\mathsf{PK}_j = \mathsf{MIFE.PK}$.
2. To encrypt, encryptor j for $j \in [n_x]$ computes a ciphertext $\mathsf{MIFE.CT}_j = \mathsf{MIFE.Encrypt}(\mathsf{EK}_j, x_j)$ for any message x_j of its choice.
3. To generate a partial key for a given function f_i, the i^{th} key authority does the following:
   (a) Compute a ciphertext $\mathsf{MIFE.CT}_{n_x+i} = \mathsf{MIFE.Encrypt}(\mathsf{EK}_{n_x+i}, f_i)$.
   (b) Define a function $\mathcal{U}'$, which upon inputs $\mathsf{MIFE.MSK}_1, \ldots, \mathsf{MIFE.MSK}_{n_y}$ does the following: i) it computes MIFE.MSK using share reconstruction. ii) it computes $\mathsf{MIFE.SK} = \mathsf{MIFE.KeyGen}(\mathsf{MIFE.MSK}, \mathcal{U}(\mathsf{Agg}_x(\cdot), \mathsf{Agg}_y(\cdot))$.
   (c) Compute $\mathsf{MIFE.SK}_{i,\mathcal{U}'} = \mathsf{MPC.RunRoundTwo}(\mathcal{U}', \mathfrak{s}_i, (\rho_1^{(1)}, \ldots, \rho_n^{(1)}))$ for $i \in [n_y]$ and outputs it[18].
4. To decrypt, the decryptor computes MPC.ComputeResult to obtain $\mathcal{U}'(\mathsf{MIFE.MSK}_1, \ldots, \mathsf{MIFE.MSK}_{n_y}) = \mathsf{MIFE.SK}_{\mathcal{U}'}$. It computes $\mathsf{MIFE.Decrypt}\left(\mathsf{MIFE.CT}_1, \ldots, \mathsf{MIFE.CT}_{n_x+n_y}, \mathsf{MIFE.SK}_{\mathcal{U}'}\right)$ and outputs it.

This construction is simpler than that of aMIFE since we permit interactive setup and allow the arity of the function being computed to be fixed. Note that if the MIFE is public key, then so is the above MPFE, since the symmetric/public key property of the above MPFE is inherited from the underlying MIFE. By correctness of MPC, the decryptor recovers MIFE.SK for the function $\mathcal{U}(\mathsf{Agg}_x(\cdot), \mathsf{Agg}_y(\cdot))$. The decryptor also obtains ciphertexts $\mathsf{MIFE.CT}(x_1), \ldots, \mathsf{MIFE.CT}(x_{n_x})$ and $\mathsf{MIFE.CT}(f_1), \ldots, \mathsf{MIFE.CT}(f_{n_y})$. By correctness of MIFE, it therefore obtains $\mathcal{U}(\mathsf{Agg}_x(x_1, \ldots, x_{n_x}), \mathsf{Agg}_y(y_1, \ldots, y_{n_y})$ as desired. Security follows as a special case of aMIFE security, by leveraging security of MPC and MIFE.

[18] We note that all authorities need to agree on the choice of Agg_x and Agg_y for correctness of this step to hold. Also note that the choice of Agg_x and Agg_y may be made during the partial key generation step, and is not required at setup.

B Decentralized ABE ∘ IPFE with Policy Hiding

In this section, we extend the decentralized inner-product predicate encryption by Michalevsky and Joye [MJ18] to provide an inner-product functional encryption capability. We observe that the primitive of decentralized attribute based, inner product FE is a natural and meaningful composition of decentralized attribute based encryption (DABE) and inner product functional encryption (IPFE), which is suggested by casting both existing primitives as special cases of our model.

For the construction, our high-level idea is to exploit the underlying algebraic structure and use the linear ciphertext and key homomorphism properties already satisfied by their construction, reminiscing the inner-product functional encryption schemes from (linearly homomorphic) PKE in [ABDCP15, ALS16, BJK15].

Definition. In this section, we define the notion of decentralized attribute-based inner-product function encryption. First, we recall the syntax, and later describe the security definition.

Syntax. A decentralized attribute-based inner-product function encryption 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 λ, the setup algorithm outputs public parameters PP.

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

$\mathsf{KeyGen}(\{\mathsf{PK}_i\}_{i\in[n]}, \mathsf{MSK}_j, \mathsf{GID}, \mathbf{x}, \mathbf{u}) \to \mathsf{SK}_{j,\mathsf{GID},\mathbf{x},\mathbf{u}}$. The key generation algorithm takes as input the public keys of all the authorities $\{\mathsf{PK}_i\}_i$, an authority master secret key MSK_j, global identifier GID, an attribute $\mathbf{x} \in \mathcal{X}_n$, and key vector $\mathbf{u} \in \mathcal{U}_\ell$. It outputs a partial secret key $\mathsf{SK}_{j,\mathsf{GID},\mathbf{x},\mathbf{u}}$.

$\mathsf{Enc}(\{\mathsf{PK}_i\}_{i\in[n]}, C, \mathbf{v}) \to \mathsf{CT}$. 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 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 CT, the decryption algorithm either outputs a message m (corresponding to the inner product value) or a special string $\perp$ (to denote decryption failure).

We require such an ABE scheme to satisfy the following properties.

Correctness. A decentralized attribute-based inner-product function encryption (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:

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

Security. For security, we consider a combination of semantic security for decentralized ABE systems in presence of corrupt authorities and message vector indistinguishability for inner product functional encryption. Below we provide the selective security variant of the corresponding property.[19]

[19] In this work, we only focus on standard semantic security, but one could also amplify to its CCA counterpart by relying on recently developed generic techniques [KW19].

Definition B.1 (Selective decentralized AB-IPFE security with 1-sided policy hiding in presence of corrupt authorities)**.** A decentralized AB-IPFE scheme is selectively secure with 1-sided policy hiding in presence of corrupt authorities if for every stateful admissible PPT adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, the following holds

$$\Pr\left[\mathcal{A}^{O(\mathsf{key},\cdot,\cdot,\cdot)}(\mathsf{params},\mathsf{CT}) = b : \begin{array}{l}(1^n, 1^\ell, S^*, (C_0^*, C_1^*), (\mathbf{v}_0^*, \mathbf{v}_1^*)) \leftarrow \mathcal{A}(1^\lambda), \mathsf{PP} \leftarrow \mathsf{GSetup}(1^\lambda) \\ \forall i \in [n] : (\mathsf{PK}_i, \mathsf{MSK}_i) \leftarrow \mathsf{LSetup}(\mathsf{PP}, 1^n, 1^\ell, i) \\ b \leftarrow \{0,1\}, \mathsf{CT} \leftarrow \mathsf{Enc}(\{\mathsf{PK}_i\}_{i\in[n]}, C_b^*, \mathbf{v}_b^*) \\ \mathsf{key} = \{(\mathsf{PK}_i, \mathsf{MSK}_i)\}_{i\in[n]} \\ \mathsf{params} = \left(\{\mathsf{PK}_i\}_{i\in[n]}, \{\mathsf{MSK}_i\}_{i\in S^*}\right)\end{array}\right]$$
$$\leq \frac{1}{2} + \mathsf{negl}(\lambda),$$

where the oracle $O(\mathsf{key},\cdot,\cdot,\cdot)$ has key material hardwired, and on input a tuple of a global identifier GID, an authority index j, and an attribute-key vector pair $(\mathbf{x},\mathbf{u})$, and responds with a partial secret key computed as $\mathsf{SK}_{j,\mathsf{GID},\mathbf{x},\mathbf{u}} \leftarrow \mathsf{KeyGen}(\{\mathsf{PK}_i\}_i, \mathsf{MSK}_j, \mathsf{GID}, \mathbf{x}, \mathbf{u})$. 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 $C_0^*(\mathbf{x}) = C_1^*(\mathbf{x}) = 0$, or (2) $C_0^* = C_1^*$ and $\langle \mathbf{u}, \mathbf{v}_0^* \rangle = \langle \mathbf{u}, \mathbf{v}_1^* \rangle$, or (3) $C_0^* = C_1^*$ and adversary does not query at least one non-corrupt authority for the same global identifier, attribute-key vector tuple $(\mathsf{GID}, \mathbf{x}, \mathbf{u})$.

B.1 Construction

Let Gen be a bilinear group generator with asymmetric source groups of prime order p, and k be the parameter used to select the computational hardness assumption as in [MJ18]. Also, let $\mathbb{G}_1, \mathbb{G}_2$ and $\mathbb{G}_T$ be the source and target groups, respectively. Similar to [MJ18], we rely on a hash function $H : \mathbb{G}_2 \times \{0,1\}^\lambda \times \mathbb{Z}_p^n \times \mathbb{Z}_p^\ell \to \mathbb{G}_2^{k+1}$ that is later modelled as a random oracle. Below we provide our decentralized inner-product FE scheme based on bilinear maps. We are using a notation similar to that used in [MJ18, Section 3.2] for ease of exposition.

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

$$\Pi = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e(\cdot,\cdot)) \leftarrow \mathsf{Gen}(1^\lambda).$$

It next samples random generators $g_1 \in \mathbb{G}_1, g_2 \in \mathbb{G}_2$. Additionally it chooses random matrices $\mathbf{A}, \mathbf{U}$ as $\mathbf{A} \leftarrow \mathbb{Z}_p^{(k+1)\times k}$ and $\mathbf{U} \leftarrow \mathbb{Z}_p^{(k+1)\times(k+1)}$, and sets the public parameters as

$$\mathsf{PP} = \left(\Pi, g_1 = [1]_1, g_2 = [1]_2, [\mathbf{A}]_1, [\mathbf{U}^\top \mathbf{A}]_1\right).$$

$\mathsf{LSetup}(\mathsf{PP}, 1^n, 1^\ell, i) \to (\mathsf{PK}, \mathsf{MSK})$. The algorithm samples random matrices $\mathbf{W}_i \leftarrow \mathbb{Z}_p^{(k+1)\times(k+1)}$ and $\mathbf{\Delta}_i \leftarrow \mathbb{Z}_p^{(k+1)\times\ell}$. It also samples a random exponent $\sigma_i \leftarrow \mathbb{Z}_p$, and sets the authority public-secret key pair as

$$\mathsf{PK} = \left(\mathsf{PP}, [\mathbf{W}_i^\top \mathbf{A}]_1, [\mathbf{\Delta}_i^\top \mathbf{A}]_T, y_i = [\sigma_i]_2\right), \qquad \mathsf{MSK} = (\mathbf{W}_i, \mathbf{\Delta}_i, \sigma_i).$$

$\mathsf{KeyGen}(\{\mathsf{PK}_i\}_{i\in[n]}, \mathsf{MSK}_j, \mathsf{GID}, \mathbf{x}, \mathbf{u}) \to \mathsf{SK}_{j,\mathsf{GID},\mathbf{x},\mathbf{u}}$. It parses the authority public keys as described above, and first computes a masking value $\boldsymbol{\mu}_j \in \mathbb{Z}_p^{k+1}$ as

$$[\boldsymbol{\mu}_j]_2 = \prod_{i=1}^{j-1} H([\sigma_i\sigma_j]_2, \mathsf{GID}, \mathbf{x}, \mathbf{u}) \oslash \prod_{i=j+1}^{n} H([\sigma_i\sigma_j]_2, \mathsf{GID}, \mathbf{x}, \mathbf{u}),$$

where the masking value is implicitly sampled. Next, it also implicitly samples $\mathbf{h} \in \mathbb{Z}_p^{k+1}$ as $[\mathbf{h}]_2 = H(g_2, \mathsf{GID}, \mathbf{x}, \mathbf{u})$. Finally, it outputs the secret key as

$$\mathsf{SK}_{j,\mathsf{GID},\mathbf{x},\mathbf{u}} = \left([\mathbf{\Delta}_j \mathbf{u} - x_j \mathbf{W}_j \mathbf{h} + \boldsymbol{\mu}_j]_2, \mathbf{x}, \mathbf{u}\right).$$

$\mathsf{Enc}(\{\mathsf{PK}_i\}_{i\in[n]}, \mathbf{y}, \mathbf{v}) \to \mathsf{CT}$. The encryption algorithm first parses the keys PK_i as defined during setup. It samples a random exponent vector $\mathbf{s} \leftarrow \mathbb{Z}_p^k$, and outputs the ciphertext CT as

$$\mathsf{CT} = \left(C_0 = \left[\mathbf{As}\right]_1, \left\{ C_i = \left[(y_i \mathbf{U}^\top + \mathbf{W}_i^\top)\mathbf{As}\right]_1 \right\}_{i\in[n]}, C_m = \left[\textstyle\sum_{i=1}^n \boldsymbol{\Delta}_i^\top \mathbf{As} + \mathbf{v}\right]_T \right).$$

$\mathsf{Dec}(\{\mathsf{SK}_{i,\mathsf{GID},\mathbf{x},\mathbf{u}}\}_{i\in[n]}, \mathsf{CT}) \to m$. It parses the secret key and ciphertext as described above. The decryptor first combines the partial keys to obtain key term $K = \odot_{i=1}^n \left[\boldsymbol{\Delta}_i \mathbf{u} - x_i \mathbf{W}_i \mathbf{h} + \boldsymbol{\mu}_i\right]_2$. The algorithm then recovers the inner product by computing the discrete log of the following

$$\frac{\langle \mathbf{u}, C_m \rangle}{e\langle C_0, K\rangle \cdot \prod_{i\in[n]} e\langle C_i, \left[x_i \mathbf{h}\right]_2\rangle},$$

where the operation $\langle \mathbf{u}, C_m \rangle$ computes the encoded inner product by first raising $\mathbf{u}$ in the exponent of each term (component-by-component), and then followed by multiplication of the resulting encodings resulting in the inner product being computed in the exponent. Also, the operation $e\langle \cdot, \cdot \rangle$ carries the same inner product operation where the pairing function $e(\cdot, \cdot)$ is being used for computing the product terms instead.

B.2 Correctness and Security

In this section, we provide the correctness proof and a high level description of security games and prove indistinguishability of successive games.

Correctness. The proof of correctness is along the lines of the Michalevsky-Joye [MJ18] scheme. Below we briefly highlight the main points.

Consider n partial secret keys $\mathsf{SK}_{i,\mathsf{GID},\mathbf{x},\mathbf{u}} = \left(\left[\boldsymbol{\Delta}_i \mathbf{u} - x_i \mathbf{W}_i \mathbf{h} + \boldsymbol{\mu}_i\right]_2, \mathbf{x}, \mathbf{u}\right)$ for authority indices $i \in [n]$ and key vectors $\mathbf{x}, \mathbf{u}$. First, note that the key product term K can be simplified as follows:

$$\begin{aligned} K &= \odot_{i=1}^n \left[\boldsymbol{\Delta}_i \mathbf{u} - x_i \mathbf{W}_i \mathbf{h} + \boldsymbol{\mu}_i\right]_2 \\ &= \odot_{i=1}^n \left[\boldsymbol{\Delta}_i \mathbf{u} - x_i \mathbf{W}_i \mathbf{h}\right]_2 \qquad (\text{since } \textstyle\sum_i \boldsymbol{\mu}_i = \mathbf{0}) \\ &= \left[\boldsymbol{\Delta}\mathbf{u} - \textstyle\sum_{i=1}^n x_i \mathbf{W}_i \mathbf{h}\right]_2 \qquad (\text{where } \boldsymbol{\Delta} = \textstyle\sum_i \boldsymbol{\Delta}_i) \end{aligned}$$

Therefore, combining with the fact that $C_0 = \left[\mathbf{As}\right]_1$ we get that

$$e\langle C_0, K\rangle = \left[(\mathbf{u}^\top \boldsymbol{\Delta}^\top - \textstyle\sum_{i=1}^n x_i \mathbf{h}^\top \mathbf{W}_i^\top)\mathbf{As}\right]_T.$$

Also, we can simplify the terms $e\langle C_i, \left[x_i \mathbf{h}\right]_2\rangle$ and $\prod_i e\langle C_i, \left[x_i \mathbf{h}\right]_2\rangle$ as

$$\begin{aligned} e\langle C_i, \left[x_i \mathbf{h}\right]_2\rangle &= \left[(x_i y_i \mathbf{h}^\top \mathbf{U}^\top + x_i \mathbf{h}^\top \mathbf{W}_i^\top)\mathbf{As}\right]_T, \\ \textstyle\prod_{i\in[n]} e\langle C_i, \left[x_i \mathbf{h}\right]_2\rangle &= \left[((\mathbf{x}^\top \mathbf{y})\mathbf{h}^\top \mathbf{U}^\top + \textstyle\sum_{i=1}^n x_i \mathbf{h}^\top \mathbf{W}_i^\top)\mathbf{As}\right]_T. \end{aligned}$$

Lastly, we also get that $\langle \mathbf{u}, C_m \rangle = \left[\mathbf{u}^\top \boldsymbol{\Delta}^\top \mathbf{As} + \mathbf{u}^\top \mathbf{v}\right]_T$ where $\boldsymbol{\Delta} = \sum_i \boldsymbol{\Delta}_i$. Combining all these terms as done during decryption, we get that

$$\frac{\langle \mathbf{u}, C_m \rangle}{e\langle C_0, K\rangle \cdot \prod_{i\in[n]} e\langle C_i, \left[x_i \mathbf{h}\right]_2\rangle} = \left[\mathbf{u}^\top \mathbf{v} - (\mathbf{x}^\top \mathbf{y})\mathbf{h}^\top \mathbf{U}^\top \mathbf{As}\right]_T.$$

Therefore, whenever $\langle \mathbf{x}, \mathbf{y} \rangle = \mathbf{x}^\top \mathbf{y} = 0$, the above gets simplified to $\left[\mathbf{u}^\top \mathbf{v}\right]_T$ which can be recovered by computing the discrete log. Thus, correctness follows.

Security. Below we briefly highlight the main ideas behind the security proof.

Theorem B.2. If the k-linear assumption (assumption 2.3) holds in both the source groups $\mathbb{G}_1$ and $\mathbb{G}_2$ over the group generator Gen, then the scheme described above is a selectively secure decentralized AB-IPFE with 1-sided policy hiding for inner product testing predicates as per Definition B.1 with at least two honest authorities.

The proof structure is similar to that of [MJ18], except now we need additionally to answer key queries for attribute-key vector pairs $(\mathbf{x}, \mathbf{u})$ such that $\langle \mathbf{x}, \mathbf{y}^* \rangle = 0$ where $\mathbf{y}^*, (\mathbf{v}_0^*, \mathbf{v}_1^*)$ are the challenge key and message vectors (respectively).[20] At a high level, the idea is to sample the central secret key term $\boldsymbol{\Delta}$ such that the reduction could honestly compute the secret keys for all attribute-key vector pairs $(\mathbf{x}, \mathbf{u})$ such that $\langle \mathbf{x}, \mathbf{y}^* \rangle = 0$ and $\langle \mathbf{u}, \mathbf{v}_0^* \rangle = \langle \mathbf{u}, \mathbf{v}_1^* \rangle$, while switching the secret keys to their semi-functional counterparts for attribute-key vector pairs $(\mathbf{x}, \mathbf{u})$ where $\langle \mathbf{x}, \mathbf{y}^* \rangle \neq 0$.

For ease of exposition, we prove security in an alternate game where the challenger answers each key query with the full (combined) key term K instead of providing partial key shares (i.e., per authority shares). Similar approach was taken in [MJ18, Appendix B], where the idea was to use the fact that by programming the masking terms $\boldsymbol{\mu}_i$'s appropriately, the security proof could be easily extended to the more general case where the challenger responds with partial shares instead of the full key. Since the same idea can be used for our scheme as well, thus we prove security in the simpler model and refer to [MJ18, Appendix B] for more details.

Proof. We start by sketching the sequence of games where the first game corresponds to the original security game. First, we switch the way we sample the master secret key wherein instead of sampling the random master secret matrix $\boldsymbol{\Delta}$ directly[21], we sample a random matrix $\boldsymbol{\Gamma}$ of same dimensions and implicitly set $\boldsymbol{\Delta} = \boldsymbol{\Gamma}\mathbf{R}$ where $\mathbf{R} \in \mathbb{Z}_p^{\ell \times \ell}$ is a random full rank matrix such that $\mathbf{R}(\mathbf{v}_0^* - \mathbf{v}_1^*) = \mathbf{e}_1$ where $\mathbf{e}_1$ is the first vector in the canonical basis of $\mathbb{Z}_p^{\ell}$. Next, we switch the way the random oracle queries are answered wherein we sample encoding $\left[\mathbf{h}\right]_2$ from a random k-dimensional subspace. This is followed by switching all the secret key queries for key vector pairs $(\mathbf{x}, \mathbf{u})$ to semi-functional whenever $\langle \mathbf{x}, \mathbf{y}_\beta^* \rangle = 0$. Next, we switch the ciphertext to a *partial* semi-functional ciphertext. Here by a partial semi-functional ciphertext we mean that we split the challenge message vector such that now we encrypt the vector $\beta(\mathbf{v}_0^* - \mathbf{v}_1^*)$ as a semi-functional ciphertext (for random challenge bit β) and vector $\mathbf{v}_1^*$ as a normal (non-semi-functional) ciphertext and homomorphically combines both of them to create the final ciphertext. The idea here is that since adversary never receives any secret key for a key vector pair $(\mathbf{x}, \mathbf{u})$ where both $\langle \mathbf{x}, \mathbf{y}_\beta^* \rangle = 0$ and $\langle \mathbf{u}, \mathbf{v}_0^* - \mathbf{v}_1^* \rangle \neq 0$, thus the reduction algorithm can perfectly simulate the above games using the dual system encryption paradigm [Wat09] as used in [MJ18]. Below we provide a high level description of the security games.

Description of games

Game 0. This corresponds to the modified AB-IPFE security game, where the challenger answers key queries will the combined secret keys instead of partial key shares. Let Q denote the total number of key queries made by adversary.

Game 1. This is same as the previous game, except the challenger:

— samples a uniformly random matrix $\boldsymbol{\Gamma} \leftarrow \mathbb{Z}_p^{(k+1)\times \ell}$,

— samples a random *orthogonal* matrix $\mathbf{R} \in \mathbb{Z}_p^{\ell \times \ell}$ subject to the constraint that $\mathbf{R}(\mathbf{v}_0^* - \mathbf{v}_1^*) = \mathbf{e}_1$, where $\mathbf{v}_0^*, \mathbf{v}_1^*$ are the challenge message vectors and $\mathbf{e}_1 = (1, 0, \ldots, 0)^\top$ (i.e., the first canonical basis vector of $\mathbb{Z}_p^{\ell}$), and

— sets $\boldsymbol{\Delta} = \boldsymbol{\Gamma}\mathbf{R}$ instead of sampling it uniformly at random.

[20] Note that in the case, the adversary submits two distinct key vectors $\mathbf{y}_0^*$ and $\mathbf{y}_1^*$ as its challenge, then it must be that $\langle \mathbf{x}, \mathbf{y}_0^* \rangle = \langle \mathbf{x}, \mathbf{y}_1^* \rangle = 0$. In this case, we can use the same proof strategy as [MJ18].

[21] Recall that we are using $\boldsymbol{\Delta}$ to denote the term $\sum_i \boldsymbol{\Delta}_i$ as in the correctness proof.

Game 2. This is same as the previous game, except the challenger answers the random oracle queries as follows:

— during setup, it samples a random matrix $\mathbf{B} \leftarrow \mathbb{Z}_p^{(k+1)\times k}$, and each fresh random oracle query is answered by sampling a random vector $\mathbf{r} \leftarrow \mathbb{Z}_p^k$ and responding with $\big[\mathbf{h}\big]_2 = \big[\mathbf{Br}\big]_2$.

Game 3.q.1. This is same as Game 2, except the challenger samples random matrices $\mathbf{A}, \mathbf{B} \leftarrow \mathbb{Z}_p^{(k+1)\times k}$ along with vectors $\mathbf{a}^\perp, \mathbf{b}^\perp \in \mathbb{Z}_p^{k+1}$ such that $\mathbf{A}^\top \mathbf{a}^\perp = \mathbf{B}^\top \mathbf{b}^\perp = \mathbf{0}$, and a random exponent $\widehat{t} \leftarrow \mathbb{Z}_p$, and the key queries are answered as follows:

— for the first $q-1$ key queries, on key vector pair $(\mathbf{x}, \mathbf{u})$, it checks whether $\langle \mathbf{x}, \mathbf{y}^*_\beta \rangle = 0$ or not. If it is equal to 0, then it computes the key K honestly as

$$K = \big[\boldsymbol{\Delta}\mathbf{u} - \textstyle\sum_{i=1}^n x_i \mathbf{W}_i \mathbf{h}\big]_2.$$

Otherwise, if the predicate is not satisfied, then it computes the key K as

$$K = \big[\boldsymbol{\Delta}\mathbf{u} + \widehat{t}\langle \mathbf{u}, \mathbf{v}^*_0 - \mathbf{v}^*_1 \rangle \mathbf{a}^\perp - \textstyle\sum_{i=1}^n x_i \mathbf{W}_i \mathbf{h}\big]_2,$$

where h in both cases is computed as in Game 2, i.e. by picking a random r and computing $\big[\mathbf{h}\big]_2 = \big[\mathbf{Br}\big]_2$.

— for the q-th key query $(\mathbf{x}, \mathbf{u})$, it again checks whether $\langle \mathbf{x}, \mathbf{y}^*_\beta \rangle = 0$ or not. Now it computes the key K (in both cases) as

$$K = \big[\boldsymbol{\Delta}\mathbf{u} - \textstyle\sum_{i=1}^n x_i \mathbf{W}_i \mathbf{h}\big]_2,$$

except if the predicate is satisfied (that is, $\langle \mathbf{x}, \mathbf{y}^*_\beta \rangle = 0$), then it computes h as in Game 2, i.e. by picking a random r and computing $\big[\mathbf{h}\big]_2 = \big[\mathbf{Br}\big]_2$; whereas if the predicate is not satisfied, then it computes h by sampling random $\mathbf{r} \leftarrow \mathbb{Z}_p^k$ and $\widehat{r} \leftarrow \mathbb{Z}_p$, and setting $\big[\mathbf{h}\big]_2 = \big[\mathbf{Br} + \mathbf{a}^\perp \widehat{r}\big]_2$.

— remaining $Q - q$ key queries are answered exactly as in Game 2. That is, it computes the key K (in both cases) as

$$K = \big[\boldsymbol{\Delta}\mathbf{u} - \textstyle\sum_{i=1}^n x_i \mathbf{W}_i \mathbf{h}\big]_2,$$

where h is computed as $\big[\mathbf{h}\big]_2 = \big[\mathbf{Br}\big]_2$.

In the above game, the random oracle queries are also answered similarly wherein the challenger does not sample the full secret key, but instead it just computes the hash vector $\mathbf{h}$ depending on the query counter and the type of query.

Game 3.q.2. This is same as the previous game, except the way the challenger answers the q-th query:

— for the q-th key query $(\mathbf{x}, \mathbf{u})$, it checks whether $\langle \mathbf{x}, \mathbf{y}^*_\beta \rangle = 0$ or not. If it is equal to 0, then it computes the key K honestly as

$$K = \big[\boldsymbol{\Delta}\mathbf{u} - \textstyle\sum_{i=1}^n x_i \mathbf{W}_i \mathbf{h}\big]_2,$$

where h is computed as $\big[\mathbf{h}\big]_2 = \big[\mathbf{Br}\big]_2$. Otherwise, if the predicate is not satisfied, then it computes the key K as

$$K = \big[\boldsymbol{\Delta}\mathbf{u} + \widehat{t}\langle \mathbf{u}, \mathbf{v}^*_0 - \mathbf{v}^*_1 \rangle \mathbf{a}^\perp - \textstyle\sum_{i=1}^n x_i \mathbf{W}_i \mathbf{h}\big]_2,$$

where h is computed by sampling random $\mathbf{r} \leftarrow \mathbb{Z}_p^k$ and $\widehat{r} \leftarrow \mathbb{Z}_p$, and setting $\big[\mathbf{h}\big]_2 = \big[\mathbf{Br} + \mathbf{a}^\perp \widehat{r}\big]_2$.

Game 3.q.3. This is same as the previous game, except when it answers the q-th query, then it computes h as $\big[\mathbf{h}\big]_2 = \big[\mathbf{Br}\big]_2$ irrespective of whether the predicate is satisfied or not. (Note that Game 3.0.3 is same as Game 2.)

Game 4. This is the same as Game 3.Q.3, except that the challenge ciphertext is semi-functional. That is, the challenger samples a random vector $\mathbf{z} \leftarrow \mathbb{Z}_p^{k+1}$ and computes the challenge ciphertext as:

$$\mathsf{CT} = \left(C_0 = \left[\mathbf{z}\right]_1, \left\{ C_i = \left[(y^*_{\beta,i}\mathbf{U}^\top + \mathbf{W}_i^\top)\mathbf{z}\right]_1 \right\}_{i\in[n]}, C_m = \left[\boldsymbol{\Delta}^\top \mathbf{z} + \beta \mathbf{v}^*_\beta\right]_T \right),$$

where β is the random challenge bit.

Game 5. This is the same as Game 4, except that the challenge ciphertext encrypts a message vector of the form $\mathbf{v}^*_0 + \eta(\mathbf{v}^*_1 - \mathbf{v}^*_0)$ instead of $\beta\mathbf{v}^*_\beta = \mathbf{v}^*_0 + \beta(\mathbf{v}^*_1 - \mathbf{v}^*_0)$, where η is sampled uniformly at random. That is,

$$C_m = \left[\boldsymbol{\Delta}^\top \mathbf{z} + \mathbf{v}^*_0 + \eta(\mathbf{v}^*_1 - \mathbf{v}^*_0)\right]_T,$$

where η is a random exponent.

Indistinguishability of games

We complete the proof by showing that adjacent games are indistinguishable. For any adversary $\mathcal{A}$ and game X, we denote by $\mathsf{Adv}^{\mathcal{A}}_s(\lambda)$, the probability that $\mathcal{A}$ wins in game S.

Lemma B.3. For any (potentially unbounded) adversary $\mathcal{A}$, we have that $\mathsf{Adv}^{\mathcal{A}}_0(\lambda) = \mathsf{Adv}^{\mathcal{A}}_1(\lambda)$.

Proof. This follows directly from the fact that, for any invertible matrix $\mathbf{R} \in \mathbb{Z}_p^{\ell\times\ell}$, the following distributions are identical.

$$\left\{ \boldsymbol{\Delta} \ : \ \boldsymbol{\Delta} \leftarrow \mathbb{Z}_p^{(k+1)\times\ell} \right\} \equiv \left\{ \boldsymbol{\Gamma}\mathbf{R} \ : \ \boldsymbol{\Gamma} \leftarrow \mathbb{Z}_p^{(k+1)\times\ell} \right\}.$$

□

Lemma B.4. If k-linear assumption (assumption 2.3) holds in $\mathbb{G}_2$ over the group generator Gen, then for any PPT adversary $\mathcal{A}$, we have that $\mathsf{Adv}^{\mathcal{A}}_1(\lambda) - \mathsf{Adv}^{\mathcal{A}}_2(\lambda) \leq \mathsf{negl}(\lambda)$ for some negligible function $\mathsf{negl}(\cdot)$.

Proof. The proof of this lemma is identical to that of [MJ18, Lemma 1]. □

Lemma B.5. If k-linear assumption (assumption 2.3) holds in $\mathbb{G}_2$ over the group generator Gen, then for any PPT adversary $\mathcal{A}$ and $q \in \{1,\ldots,Q\}$, we have that $\mathsf{Adv}^{\mathcal{A}}_{3.(q-1).3}(\lambda) - \mathsf{Adv}^{\mathcal{A}}_{3.q.1}(\lambda) \leq \mathsf{negl}(\lambda)$ for some negligible function $\mathsf{negl}(\cdot)$.

Proof. The proof of this lemma is identical to that of [MJ18, Lemma 2]. □

Lemma B.6. For any (potentially unbounded) adversary $\mathcal{A}$ and $q \in \{1,\ldots,Q\}$, we have that $\mathsf{Adv}^{\mathcal{A}}_{3.q.1}(\lambda) - \mathsf{Adv}^{\mathcal{A}}_{3.q.2}(\lambda) \leq \mathsf{negl}(\lambda)$ for some negligible function $\mathsf{negl}(\cdot)$.

Proof. The proof of this lemma is identical to that of [MJ18, Lemma 3]. □

Lemma B.7. If k-linear assumption (assumption 2.3) holds in $\mathbb{G}_2$ over the group generator Gen, then for any PPT adversary $\mathcal{A}$ and $q \in \{1,\ldots,Q\}$, we have that $\mathsf{Adv}^{\mathcal{A}}_{3.q.2}(\lambda) - \mathsf{Adv}^{\mathcal{A}}_{3.q.3}(\lambda) \leq \mathsf{negl}(\lambda)$ for some negligible function $\mathsf{negl}(\cdot)$.

Proof. The proof of this lemma is identical to that of [MJ18, Lemma 4]. □

Lemma B.8. If k-linear assumption (assumption 2.3) holds in $\mathbb{G}_1$ over the group generator Gen, then for any PPT adversary $\mathcal{A}$, we have that $\mathsf{Adv}^{\mathcal{A}}_{3.Q.3}(\lambda) - \mathsf{Adv}^{\mathcal{A}}_4(\lambda) \leq \mathsf{negl}(\lambda)$ for some negligible function $\mathsf{negl}(\cdot)$.

Proof. The proof of this lemma is identical to that of [MJ18, Lemma 5]. □

Lemma B.9. For any (potentially unbounded) adversary $\mathcal{A}$, we have that $\mathsf{Adv}_4^{\mathcal{A}}(\lambda) - \mathsf{Adv}_5^{\mathcal{A}}(\lambda) \leq \mathsf{negl}(\lambda)$ for some negligible function $\mathsf{negl}(\cdot)$.

Proof. The proof of this lemma is similar to that of [MJ18, Lemma 6], except now the secret matrix $\mathbf{\Delta}$ is set in a more intricate way. First, note that in game 4, we can rewrite the message component of the ciphertext (i.e., C_m) as follows:

$$\begin{aligned} C_m &= \left[\mathbf{\Delta}^\top \mathbf{z} + \beta \mathbf{v}_\beta\right]_T \\ &= \left[\mathbf{R}^\top \mathbf{\Gamma}^\top \mathbf{z} + \beta(\mathbf{v}_1^* - \mathbf{v}_0^*) + \mathbf{v}_0^*\right]_T \end{aligned}$$

Since we sampled matrix $\mathbf{R}$ such that it is a random *orthogonal* $(\ell \times \ell)$ matrix with the constraint that $\mathbf{R}(\mathbf{v}_0^* - \mathbf{v}_1^*) = \mathbf{e}_1$, thus we have that $\mathbf{R}^\top \mathbf{e}_1 = \mathbf{v}_0^* - \mathbf{v}_1^*$. So, we can write C_m as

$$C_m = \left[\mathbf{R}^\top \mathbf{\Gamma}^\top \mathbf{z} - \beta \mathbf{R}^\top \mathbf{e}_1 + \mathbf{v}_0^*\right]_T = \left[\mathbf{R}^\top (\mathbf{\Gamma}^\top \mathbf{z} - \beta \mathbf{e}_1) + \mathbf{v}_0^*\right]_T.$$

Now the idea is to sample matrix $\mathbf{\Delta}$ as $\mathbf{\Delta} = \widehat{\mathbf{\Delta}} - (\mathbf{e}_1^\top \otimes \mathbf{a}^\perp)\widehat{t}$ for a random matrix $\widehat{\mathbf{\Delta}}$ and random exponent $\widehat{t}$. Here $\otimes$ denotes the tensoring operation.

With this modification, we can apply a proof strategy similar to that in [MJ18, Lemma 6], where we rely on the fact that the public keys and public parameters can be sampled without the knowledge of $\widehat{t}$ (i.e., only given the matrix $\widehat{\mathbf{\Delta}}$ and other public terms). Next, we can show that for every key query $(\mathbf{x}, \mathbf{u})$ we again do not need $\widehat{t}$, and can only be answered using $\widehat{\mathbf{\Delta}}$. For this, consider two cases.

First, when $\langle \mathbf{x}, \mathbf{y}_\beta^* \rangle = 0$, then it must be the case that $\langle \mathbf{u}, \mathbf{v}_0^* - \mathbf{v}_1^* \rangle = 0$, therefore $\mathbf{u} \in \mathsf{Span}(\{\mathbf{R}^\top \mathbf{e}_i\}_{i \in [n] \setminus \{1\}})$. This gives that $\mathbf{\Delta u} = \mathbf{\Gamma R u} \in \mathbf{\Gamma} \cdot \mathsf{Span}(\{\mathbf{e}_i\}_{i \in [n] \setminus \{1\}})$, that is it does not depend on the first column of $\mathbf{\Gamma}$ which depends on $\widehat{t}$.

Second, when $\langle \mathbf{x}, \mathbf{y}_\beta^* \rangle \neq 0$, then we get that the first column entry of $\mathbf{Ru}$ corresponds to $\langle \mathbf{u}, \mathbf{v}_0^* - \mathbf{v}_1^* \rangle$, and since the key term contains the term $\mathbf{\Delta u} + \widehat{t}\langle \mathbf{u}, \mathbf{v}_0^* - \mathbf{v}_1^* \rangle \mathbf{a}^\perp$, therefore this gets simplified to $\widehat{\mathbf{\Delta}}\mathbf{u}$. Thus, this can also be computed without the knowledge of $\widehat{t}$.

Finally, we notice that setting $\mathbf{z}$ in the challenge ciphertext as $\mathbf{As} + \mathbf{b}^\perp \widehat{s}$ (where $\mathbf{s} \leftarrow \mathbb{Z}_p^k$, $\widehat{s} \leftarrow \mathbb{Z}_p$), we obtain that C_m completely hides the challenge bit β. This completes the proof. □

Lemma B.10. If the k-linear assumption (assumption 2.3) holds over the group generator Gen, then for any PPT adversary $\mathcal{A}$, we have that $\mathsf{Adv}_5^{\mathcal{A}}(\lambda) \leq \mathsf{negl}(\lambda)$ for some negligible function $\mathsf{negl}(\cdot)$.

Proof. If $\mathbf{y}_0^* = \mathbf{y}_1^*$, then the advantage $\mathsf{Adv}_5^{\mathcal{A}}(\lambda) = 0$ since the challenge ciphertext is independent of the challenge bit β. Whereas if $\mathbf{y}_0^* \neq \mathbf{y}_1^*$, then it must be the case that all the adversary's key queries were for rejecting attributes that is $\langle \mathbf{x}, \mathbf{y}_0^* \rangle = \langle \mathbf{x}, \mathbf{y}_1^* \rangle = 0$ for all vectors $\mathbf{x}$ queried by $\mathcal{A}$. In the latter case, we have that all the keys queried have been semi-functional functional above, and thus we can make the entire ciphertext semi-functional as well by using a similar hybrid structure as above, and finally by using an argument similar to that of [MJ18, Lemma 7], we can conclude that the challenge ciphertext encrypts a random vector $\mathbf{y}$ instead of $\mathbf{y}_\beta^*$ thereby completing the proof. □

□