Multi-Party Functional Encryption

Shweta Agrawal
IIT Madras*

Rishab Goyal
MIT†

Fabrice Mouhartem
IIT Madras‡

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 ABE, multi-authority FE and such others. We provide a unifying framework to capture existing multi-party FE primitives, define new, natural primitives that emerge from the framework, provide a feasibility result assuming general multi-input functional encryption (MIFE), and also provide constructions for restricted function families from standard assumptions.

In more detail, we provide the following new constructions:

1. *Two Input Quadratic Functional Encryption*: We provide the first two input functional encryption scheme for *quadratic* functions from the SXDH assumption on bilinear groups. To the best of our knowledge, this is the first construction of MIFE from standard assumptions that goes beyond the inner product functionality.

2. *Decentralized Inner Product Functional Encryption*: We provide the first decentralized version of an inner product functional encryption scheme, generalizing the recent work of Michalevsky and Joye [MJ18]. Our construction supports access policies C that are representable as inner product predicates, and is secure based on the k-linear assumption, in the random oracle model.

3. *Distributed Ciphertext-Policy Attribute Based Encryption.* We provide a decentralized variant of the recent ciphertext-policy attribute based encryption scheme, constructed by Agrawal and Yamada [AY20]. 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 abstraction predicts meaningful new primitives for multi-party functional encryption which we describe but do not instantiate — these may be constructed in 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 and an Indo-French CEFIPRA grant.

†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: fabrice.mouhartem@gmail.com.

Contents

1 Introduction

Functional encryption (FE) [SW05, BSW11] is a paradigm that enables fine-grained access control on encrypted data. In more detail, functional encryption allows deriving private keys corresponding to *functions* rather than users, namely 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. When SK_f is used to decrypt ciphertext $\mathsf{CT}_{\mathbf{x}}$, the decryptor learns $f(x)$ and nothing else.

Functional Encryption as an abstract notion was introduced in 2011, by Boneh, Sahai, and Waters [BSW11] to unify and abstract out various notions of encryption with restricted decryption capabilities, such as identity-based encryption [Sha84, BF01], predicate encryption [KSW08], attribute-based encryption [GPSW06] and such others. This abstraction helped unify the view on the many related constructions and primitives that had sprung up in the literature, helped to predict new and useful primitives that had not been studied till then (such as partially hiding predicate/functional encryption [GVW12, GVW15]), and also served as the right abstraction to understand the relationship of this broad area to other notions in cryptography (such as to indistinguishability obfuscation [AJ15, BV15]). Several interesting constructions of FE for various functionalities from standard assumptions have been obtained [Wee17, DOT18, CZY19, AGW20, ACGU20].

Supporting Multiple Users. Subsequently, many new primitives arose to generalize FE to the multi-user setting – multi-input functional encryption [GGG+14a], multi-client functional encryption [CSG+18a], decentralized multi-client functional encryption, adhoc multi-input functional encryption [ACF+20], multi-authority attribute based encryption [Cha07] and such others. These notions are related yet different, similar to the many variants of functional encryption, and it is often difficult to understand how they compare to one-another and what is known in terms of feasibility.

In this work, we initiate the study of multi-party functional encryption which similarly unifies and abstracts out various notions of *multi-party* functional encryption, such as the above. Our starting point observation is that all the 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+14a] and multi-client functional encryption (MCFE) [CSG+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) — i) 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 ii) *better fitting real world* scenarios where different authorities may be responsible for issuing keys corresponding to different attributes to a user, such as passport office, drivers license, university 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 embedded 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 Multi-Authority Functional Encryption (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_i 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 decentralize both ciphertexts and keys, as follows.

(a) *Decentralized Multi-Client Functional Encryption* (D-MCFE): The notion of decentralized multi-client FE was defined by Chotard et al. [CSG+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. Each party's master key is sufficient for it to produce its own ciphertexts and function keys but does not compromise the security of any other user in the system. 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 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): Similar to aMIFE, this primitive was introduced very recently in [CSG+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 also supports the public key setting.

The Right Abstraction? While the above notions are powerful and enable controlled manipulation of encrypted data in ever-more powerful ways, they seem i) too related to warrant independent identities, and ii) too specialized and thereby under-representing their full scope. Establishing each new variant of distributed FE as a fresh new primitive clutters the landscape and obscures the connections between them. Additionally, these primitives hint at other, equally meaningful primitives which have not been studied this far. For concreteness, let us consider the following examples to motivate meaningful, new primitives:

Distributed Ciphertexts: 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. However, neither MIFE nor MCFE capture such combinations – MIFE does not support labels and MCFE only allows ciphertexts with matching labels to be combined. 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. The notion of partially hiding multi-input FE is also a generalization of the well studied partially hiding functional encryption [GVW12, AJL+19] which is restricted to the single user setting.

Distributed Keys: Similarly, even though distributed keys have been considered in the literature before, it is useful to consider more general notions. As an example, let us say that there are multiple key authorities that wish to specify different filtering criteria on the encrypted data. For instance, in the context of an investigation, a hospital, a police station and a bank may wish to filter suspects in an encrypted database based on different criteria, for which they issue different keys. Previous works have considered the combination of partial keys when these refer to the same function, but in the current setting, we desire to compose the functions specified by each partial key. As another example, consider a ciphertext containing a missile access code which should only be opened if sufficient number of officials in a certain group combine their partial decryption keys. While the traditional notion of security for FE does not permit collusion of users' keys, in some cases it is desirable to permit certain controlled collusions to achieve new and meaningful functionalities.

1.1 Our Notion: Multi-Party Functional Encryption

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. 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 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 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 3.1 for details on how the above primitives can be expressed as instances of MPFE.

In Appendix B, we provide a feasibility result for our general notion of MPFE by leveraging ideas from the recent work on adhoc MIFE [ACF+20]. Since our general feasibility result must support all aggregation functions, it relies on the minimal assumption of MIFE for circuits. In Section 7, we suggest new and meaningful primitives for multi-party functional encryption which we do not instantiate – these may be constructed in future work.

1.2 Related Work

Most closely related to our abstraction of MPFE is the very recent notion of "Dynamic Decentralized Functional Encryption" (DDFE) introduced by Chotard et al. [CSG+20]. The notion of DDFE also unifies some notions of functional encryption supporting multiple users, but is more restricted than ours in the following ways:

1. DDFE does not support any function hiding and always assumes that the input in the ciphertext is entirely hidden. In contrast, MPFE allows for a public and private component in both the ciphertext and the function key. This allows our notion to easily capture function hiding and partial function hiding in the secret keys as well as partial attribute hiding which can naturally capture "public index" FE schemes such as attribute based encryption, partially hiding predicate encryption [GVW15] or partially hiding functional encryption [GVW12, AJL+19].

2. DDFE does not differentiate between encryption keys and master secret keys – this forces the encryptors and the key generators to be the same users. In contrast, in MPFE (as in MIFE) these users can be different.

3. MPFE allows the "mode" of the setup algorithm to be specified – central, local or interactive, which denotes a central trusted setup, a completely decentralized non-interactive setup or an interactive one time setup between parties, respectively. On the other hand, DDFE supports only a decentralized non-interactive setup, which makes it *too strong* to capture several existing constructions in the literature such as constructions of D-MCFE which have interactive setup [CSG+18a], or even constructions of MIFE [GGG+14b] which were built using centralized, trusted setup (which may generate possibly correlated encryption keys and a master secret key which are later distributed to encryptors and key generator respectively).

1.3 New Constructions

After defining a unified notion and exploring general feasibility, we also provide new constructions for multi-party functional encryption schemes. Our focus is on constructions for interesting classes of multi-party primitives, that can be constructed from standard assumptions (excepting our feasibility result). To this end, we provide the following new constructions:

1. *Two Input Quadratic Functional Encryption*: We provide the first two input functional encryption scheme for quadratic functions from the SXDH assumption on bilinear groups. Our construction builds upon the single user quadratic functional encryption scheme by Lin [Lin17]. Our scheme has a centralized setup algorithm which generates encryption keys EK_1 and EK_2 for both users, using which the two encryptors can generate ciphertexts independently for any inputs $\mathbf{u}_1$ under EK_1, and $\mathbf{u}_2$ under EK_2. The key generator provides a key for a quadratic polynomial $\mathbf{c}$, and the decryptor, given the two ciphertexts and secret key may decrypt to recover $\langle \mathbf{u}_1 \otimes \mathbf{u}_2,\ \mathbf{c} \rangle$

2. *Decentralized, Attribute Based Inner Product Functional Encryption*: We provide the first decentralized version of an inner product functional encryption scheme, generalizing the recent work of Michalevsky and Joye [MJ18]. Our scheme allows multiple authorities to generate their public and master keys independently, without any communication. A ciphertext is associated with a policy C and vector

$\mathbf{v}$, while the i^{th} secret key for user GID, for $i \in [n]$ is associated with an attribute x_i and a vector $\mathbf{u}$ (same for all authorities). Decryption succeeds to recover $\langle \mathbf{u}, \mathbf{v} \rangle$ if and only if $C(x_1, \ldots, x_n) = 1$. Our construction supports access policies C that are representable as inner product predicates, and is secure based on the k-linear assumption, in the random oracle model.

3. *Distributed Ciphertext-Policy Attribute Based Encryption.* We provide a decentralized variant of the recent ciphertext-policy attribute based encryption scheme, constructed by Agrawal and Yamada [AY20]. In more detail, we have n key authorities, all of which run an independent, local setup algorithm, and issue partial keys for a given user GID for a given attribute $\mathbf{x}$. The ciphertext is associated with an NC_1 circuit C (public) and a private message m. The decryptor may obtain partial decryption keys from the n authorities and recover m if and only if $C(\mathbf{x}) = 1$. 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 now provide a brief technical overview of each of our three constructions.

Two Input Quadratic Functional Encryption

The starting point of our two-input FE scheme for quadratic functions is the single input quadratic FE scheme by Lin [Lin17]. Similarly to her scheme, our two input FE for quadratic functions is made up of the following ingredients:

1. A function hiding inner product encryption scheme cIPE which has a *canonical* form. In particular, its ciphertexts and secret keys encode the input and function vectors in different source groups $\mathbb{G}_1$, $\mathbb{G}_2$ of the bilinear map, and decryption simply uses pairing to produce an encoding of the output inner product in the target group $\mathbb{G}_T$. Such a function hiding IPE was constructed by [LV16, Lin17].
2. The inner product encryption scheme IPE provided by Abdalla et al. [ABDCP15].

These schemes are interleaved in a *non black-box* way to obtain the final result. We proceed to describe the main ideas of the construction.

Review of the ABDP Inner Product Scheme. Recall that in an inner product functional encryption scheme (IPFE), a ciphertext is associated with a vector $\mathbf{u} \in \mathbb{Z}_p^{\mathsf{N}}$, a function key is associated with some vector $\mathbf{v} \in \mathbb{Z}_p^{\mathsf{N}}$ and decryption reveals $\langle \mathbf{u}, \mathbf{v} \rangle$ and nothing else. In the ABDP IPFE scheme, the master key is a randomly chosen vector $\mathbf{s} \in \mathbb{Z}_p^{\mathsf{N}}$ while the public key is $g^{\mathbf{s}}$, which we denote by $[\mathbf{s}]$. The ciphertext for a vector $\mathbf{u}$ is computed as $\mathsf{iCT} = (\ [-r], [r\mathbf{s} + \mathbf{u}]\ )$ where r is a random scalar. The secret key for vector $\mathbf{v}$ is computed as $\mathsf{iSK} = (\langle \mathbf{v}, \mathbf{s} \rangle,\ \mathbf{v})$. To decrypt, the user simply computes the inner product of the ciphertext and the secret key to obtain $[z] = [\langle \mathbf{u},\ \mathbf{v} \rangle]$ and then recovers z by discrete log, if z is polynomially bounded.

Review of Lin's Quadratic FE Scheme. Lin [Lin17] designed a clever quadratic FE scheme using the above ingredients, which achieves ciphertext size linear in the length of the underlying vectors (ignoring polynomial factors of the security parameter). Her scheme uses an "inner" function hiding IPE scheme (cIPE) which enjoys canonical form as described above, so that decryption of the inner cIPE yields ciphertexts under the "outer" ABDP IPFE scheme. The ciphertexts of the outer IPFE scheme may then be decrypted with the secret keys of the outer IPFE scheme, which are provided by the key generator, to recover the output.

In more detail, let $\mathbf{s} = \mathbf{s}_1 \otimes \mathbf{s}_2 \in \mathbb{Z}_p^{\mathsf{N}^2}$. Then, the encryptor provides cIPE ciphertexts for the vector $(u_i \| s_{1,i})$ as well as cIPE secret keys for the vector $(u_j \| r s_{2,j})$ for $i, j \in [\mathsf{N}]$. Since cIPE is function hiding, the messages are hidden both in the ciphertext and the secret key. Moreover, since cIPE has canonical form, decryption yields $[\mathbf{u} \otimes \mathbf{u} \| r\ \mathbf{s}_1 \otimes \mathbf{s}_2]$, which is exactly the ciphertext of ABDP IPFE under master secret key $\mathbf{s}_1 \otimes \mathbf{s}_2$. When decrypted with an ABDP secret key for a function $\mathbf{c} \in \mathbb{Z}_p^{\mathsf{N}^2}$, this yields the quadratic function $\langle \mathbf{c},\ \mathbf{u} \otimes \mathbf{u} \rangle$ as desired.

The above informal description correctly conveys the high level intuition but is wrong in practically all the details. The scheme must undergo several modifications before it admits a (complicated) security proof from SXDH. As an example, one issue that arises is that $\mathbf{s}_1 \otimes \mathbf{s}_2 \in \mathbb{Z}_p^{\mathsf{N}^2}$ is not a kosher ABDP master key, since it is not a uniformly distributed vector over $\mathbb{Z}_p^{\mathsf{N}^2}$. To overcome this, Lin also lifts the outer IPE function key to the exponent, and relies on SXDH to argue that $[\mathbf{s}_1 \otimes \mathbf{s}_2]$ is indistinguishable from $[\mathbf{s}_{1,2}]$ where $\mathbf{s}_{1,2} \leftarrow \mathbb{Z}_p^{\mathsf{N}^2}$. This modification appears to create a problem for correctness of the outer ABDP scheme, but this can be resolved – please see [Lin17] for details.

Our Two-Input Scheme for Quadratic Functions. Our starting point observation is that the ciphertext of Lin's single input quadratic scheme (denoted henceforth as QFE) is more powerful than required for the single input setting. In more detail, recall that the QFE ciphertext consists of cIPE ciphertexts and secret keys for vectors $(u_i \| s_{1,i})$ and $(u_j \| rs_{2,j})$ respectively, for $i, j \in [\mathsf{N}]$. However, cIPE being an inner product FE scheme, allows its ciphertexts and secret keys to be generated independently – this suggests that we may be able to distribute encryption among two parties, each holding input $\mathbf{u}_\beta$ for $\beta \in \{1, 2\}$ so that cIPE decryption produces a ciphertext for $\mathbf{u}_1 \otimes \mathbf{u}_2$ under the outer ABDP scheme. Now, the key generator can provide a key for a quadratic function $\mathbf{c}$ as in [Lin17], and decryption can recover $\langle \mathbf{c},\ \mathbf{u}_1 \otimes \mathbf{u}_2, \rangle$ as desired. In more detail, we let party 1 with encryption key $\mathbf{s}_1$ and input $\mathbf{u}_1$ generate cIPE ciphertext for $(u_{1,i} \| s_{1,i})$, $i \in [\mathsf{N}]$ and party 2 with encryption key $\mathbf{s}_2$ and input $\mathbf{u}_2$ generate cIPE secret keys for $(u_{2,j} \| rs_{2,j})$, $j \in [\mathsf{N}]$. Now, given IPE function key for $\mathbf{c}$, cIPE decryption followed by IPE decryption as before can let us compute $\langle \mathbf{c},\ \mathbf{u}_1 \otimes \mathbf{u}_2 \rangle$ as desired.

As in the case of [Lin17], we will run into several difficulties before this intuition can be made to work. In particular, we now need to randomize each user's ciphertext with a fresh randomness – user β chooses r_β, for $\beta \in \{1, 2\}$ and we use the pairing to compute $[r_1 r_2]$ as part of the ciphertext for the outer IPFE. One difficulty that this creates is that we can no longer lift $\mathbf{s}_1 \otimes \mathbf{s}_2$ to the exponent as in [Lin17], since we have already "used up" the pairing operation in computing $[r_1 r_2]$ and we would need another level of multilinearity to compute $[r_1 r_2 \langle \mathbf{c},\ \mathbf{s}_1 \otimes \mathbf{s}_2 \rangle]$ as required for IPE decryption. Note that [Lin17] did not run into this issue since there is only one encryptor in her setting. We overcome this by expanding the outer IPFE master secret key to a random vector of length N^2, thereby sacrificing succinctness. Note that in the two input case, even a non-succinct construction is non-trivial, as it cannot be obtained by simple linearization, unlike the single input case.

The proof of our construction departs significantly from [Lin17]. The security definition for multi-input FE is much more complex than that for single input FE – in particular, given multiple (say Q) ciphertexts in each slot, we must now account for combinations of ciphertexts across the two slots and cannot naively change a ciphertext pair $(\mathbf{u}_1^k, \mathbf{u}_2^k)$ for $b = 0$ to $(\mathbf{v}_1^k, \mathbf{v}_2^k)$ for $b = 1$ one pair at a time, for $k \in [Q]$. Such an approach would lead to an immediate distinguishing attack, since there is no guarantee that $\langle \mathbf{c}, \mathbf{u}_1^k \otimes \mathbf{u}_2^{k+1} \rangle = \langle \mathbf{c}, \mathbf{v}_1^k \otimes \mathbf{u}_2^{k+1} \rangle$. To account for this, we switch the ciphertext for $b = 0$ to ciphertext for $b = 1$, one input *combination* at a time. It is not clear that this can work, since there are Q^2 combinations, but only $O(Q)$ space in the hybrid. We overcome this by designing a careful sequence of hybrids which tracks all input combinations and changes them one at a time, by relying on trapdoor techniques. Please see Section 4 for details.

Decentralized Attribute Based Inner Product Functional Encryption

Recently Michalevsky and Joye [MJ18] gave positive results for the notion of decentralized policy-hiding ABE systems. The centerpiece of their construction is a decentralized ABE scheme for inner product predicates which they extended in a black-box way to support conjunctions, disjunctions, and threshold policies. Since theirs is a regular ABE scheme, thus it provides the standard all-or-nothing access to the encrypted plaintext. Here we are interested in the question whether one could improve it to more expressive decryption functionality such as inner products. We answer this in the affirmative by constructing a decentralized attribute-based IPFE scheme for inner product predicates.

Let us start by recalling the broad structure of the decentralized ABE scheme from [MJ18]. In their

system, the i-th authority controls the master secret key for the i-th bit of the attribute vector. In an overly simplified version of their 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 KEM key).

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

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

A closer inspection of the above basic scheme reveals that it has useful linear homomorphic properties over both the ciphertext and key space. Thus, a natural idea to extend to provide inner product decryption functionality would be to use ideas developed in the literature for building inner-product encryption generically from public-key encryption with special structural and homomorphic properties [ABDCP15]. At a high level, we follow a similar approach. Suppose the message vectors be of length ℓ. We modify the above skeleton 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.

At a high level, our attribute-based IPFE scheme structurally follows the above paradigm, but for proving security we need to make quite a few changes. Very briefly, the [MJ18] proof techniques were specially designed for ABE since ABE is an all-or-nothing encryption primitive, and the ideas do not seem to translate directly to inner-product FE since, in a decentralized attribute-based IPFE system, the adversary is allowed to query on accepting attribute vectors as long as the challenge messages have identical inner product with the underlying inner-product vector (embedded inside the key). Concretely, it turns out in [MJ18], one could rely on the dual system paradigm [Wat09] for proving security where 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 do not work at all. In IPFE, we can not hope to execute the same strategy (at least) directly since now we can not 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 informally 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. A similar technical issue was also noted in [ACGU20] in the context of single-authority

setting, whereas here we are working in a decentralized setting. We refer the reader to Section 5 for more details.

Distributed Ciphertext-Policy Attribute Based Encryption

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. Since their system is defined in the single-authority setting, it suffers from the key escrow problem. Motivated by our premise of decentralization, here we show that we could distribute the setup and key generation in [AY20] among a polynomial number of authorities that are working completely *non-interactively* and asynchronously. Our transformation is a natural extension of the scheme of [AY20], and we prove security under the same assumptions, except we also need to rely on the Random Oracle Model (ROM) for sampling the global user identifiers GID in order to tie together the secret key components of all the authorities.

Let us 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 order for it to be useful and interesting.

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[1].

To describe our construction, we recall the high level structure of the Agrawal-Yamada scheme [AY20] below. But first, we need to recollect some basic facts about 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 to do the following:

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.

[1] As an example, consider two identifier-attribute pairs $(\mathsf{GID}_b, \mathbf{x}_b)$ for $b \in \{0,1\}$ such that $\mathbf{x}_0 \neq \mathbf{x}_1$ but $F^*(\mathbf{x}_b) = 1$ for both $b \in \{0,1\}$. Suppose there are exactly two honest authorities (and rest are all corrupt). Ideally in a distributed ABE scheme, if an attacker requests a partial key on $(\mathsf{GID}_0, \mathbf{x}_0)$ from the first honest authority but not the second one, and it also requests a partial key on $(\mathsf{GID}_1, \mathbf{x}_1)$ from second honest authority but not the first one; then the semantic security of the underlying system must still hold.

Our multi-authority extension is natural and easy to describe. First, 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 $j \in [N]$. (That is, we have that $\sum_j \phi_{i,b}^{(j)} = \psi_{i,b}$ for all i, b.) Now it encodes each sequence of $\{\phi_{i,b}^{(j)}\}_{i,b}$ terms under the corresponding authority's master public key as above. The intuition is that 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.

The above idea seems to work except for the following caveat: during key generation, in [AY20], the key generator samples a random exponent δ per secret key which in the single-authority setting could be easily sampled by the central authority. However, in our distributed multi-authority setting, it is not clear how to sample it without interaction. To this end, we rely on a hash function which we model as a ROM for proof purposes, and implicitly set $[\delta]_2 = H(\mathsf{GID})$. Since all authorities have access to H, thus this resolves the sampling fresh δ per key problem.

This completes the main idea behind our distributed $\mathsf{CP\text{-}ABE}$ scheme. Although the 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 query that does not receive the same query. Such queries did not exist in the single-authority setting. However, we are able to extend the single authority proof to the multi-authority setting by careful analysis. Along the way, we provide an alternate proof for showing hardness of making certain "bad" queries than that provided in [AY20]. Lastly, we remark that our scheme enjoys useful features such as: (1) an authority can adaptively join and leave the system at any point, (2) an encryptor can choose the set of authorities under which it wants to encrypt, and (3) different authorities can be run totally asynchronously. Please see Section 6 for more details.

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 Multi-Input Functional Encryption

An *n-input FE scheme* [GGG+14b] 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 $\lambda, \in \mathbb{N}$, $\mathbf{x}_1 \dots \mathbf{x}_n \in \mathcal{M}_\lambda$ and $f \in \mathcal{F}_\lambda$

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

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 2.1.

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

$$|\Pr\left[\mathbf{IND}_{\mathcal{A}}^{\mathsf{MIFE}}(\cdot) \text{ outputs } 1 \right] - 1/2| = \mathsf{negl}(\lambda) \, .$$

2.2 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").

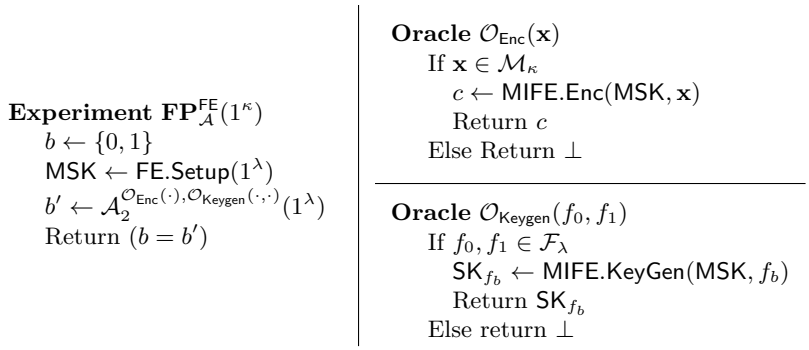

Figure 2.2: Experiment for FP-security of FE.

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 2.2.

We call $\mathcal{A}$ *legitimate* if for all $\lambda \in \mathbb{N}$, in all transcripts $\mathbf{FP}^{\mathsf{FE}}_{\mathcal{A}}(1^\kappa)$ 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 FP-secure if for any adversary $\mathcal{A}$

$$|\Pr\left[\mathbf{FP}^{\mathsf{FE}}_{\mathcal{A}}(\cdot) \text{ outputs } 1\right] - 1/2| = \mathsf{negl}(\cdot) \ .$$

2.3 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 2.1. 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 2.3. 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 $\mathbf{REAL}^{\mathsf{MPC}}_{\mathcal{A}}(\cdot)$ and $\mathbf{IDEAL}^{\mathsf{MPC}}_{\mathcal{A},SS}(\cdot)$ are computationally indistinguishable. Note that simulation of the first-round protocol messages for the honest

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

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

(**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 2.3: Experiments for SIM-security of two-round MPC.

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 2.3 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 $\mathcal{A}_2$.

2.4 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 2.2 (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 2.3 ($\mathsf{Ada\text{-}IND}$ 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 2.4 ($\mathsf{Ada\text{-}INDr}$ 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 2.5 (Sel-IND security for ABE)**.** We define Sel-IND security game as Ada-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 Sel-INDr security (or simply *selective security*) if the quantity is negligible.

Definition 2.6 (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

$$R^{\mathsf{CP}}_\lambda(C, \mathbf{x}) = \neg C(\mathbf{x}).^{2}$$

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

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

2.5 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 2.7 (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

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

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

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. We follow the presentation by Tsabary [Tsa19], but with different parameters.

Lemma 2.9 (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.

2.6 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 2.9). 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

$$\psi_1 := \mathbf{s}^\top \mathbf{u} + e_1 + \mu\lceil q/2\rceil \in \mathbf{Z}_q,\ \psi_2^\top := \mathbf{s}^\top \mathbf{A} + \mathbf{e}_2^\top \in \mathbf{Z}_q^m,\ \psi_{i,b}^\top := \mathbf{s}^\top(\mathbf{B} - x_i\mathbf{G}) + \mathbf{e}_{i,b}^\top \in \mathbf{Z}_q^m$$

   for all $i \in [\ell]$ and $b \in \{0,1\}$.
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 $\perp$.
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 2.10. 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 2.11 (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 $\mathsf{Sel\text{-}INDr}$ security. Assuming *subexponential* hardness of $\mathsf{LWE}(\mathsf{n}, \mathsf{m}, \mathsf{q}, \chi)$ with the same parameters, the above scheme satisfies $\mathsf{Ada\text{-}INDr}$ security with respect to the ciphertext space $\mathcal{CT} := \mathbf{Z}_q^{m(\ell+1)+1}$

2.7 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. A 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 2.12 (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, 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. 2.4. 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

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 2.4: 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 $\mathcal{D}$.

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.

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 to run in three modes, described next.

Setup modes. The Setup algorithm 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 can be executed in any one of the three modes described above.[3] 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 $\{\mathsf{EK}_i\}_{i \le n_x}$, master secret keys $\{\mathsf{MSK}_i\}_{i \le n_y}$ and public key PK.

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

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

Remark 3.1 (Dynamic MPFE)**.** The notions of adhoc MIFE [ACF+20] and DDFE place a strong emphasis on *dynamism* – namely the ability for parties to i) generate their own keys (local setup), and join the protocol dynamically without prior agreement via a centralized or interactive setup, and ii) choose the functionality (in our case Agg_x and Agg_y) to be computed dynamically; in particular the functionality may change for every instance of the protocol. Note that dynamism necessitates a local setup algorithm, since centralized or interactive setup is not meaningful when even the number of participating paties is not known. Since a majority of constructions in the literature (from standard assumptions) do not support dynamism, we use the simpler fixed-party setting for our main definition. In Appendix A, we capture the dynamic setting as well.

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

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:

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

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+14a, §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. (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.)

$\mathsf{Key}^{\mathsf{K},\beta}(\cdot,\cdot)$, upon a call to this oracle for an honest user index $j \in [n_y]$, function inputs $(y_j^{k,0}, y_j^{k,1})$ (where $y_j^{k,b} = \left(y_{j,\mathrm{pub}}^{k,b}, y_{j,\mathrm{pri}}^{k,b}\right)$ for $b \in \{0,1\}$), the challenger first checks whether the user j was already corrupted or not. That is, if $j \in \mathcal{S}_y$, then it sends nothing, otheriwse 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 $x_i^{\ell,b} = \left(x_{i,\mathrm{pub}}^{\ell,b}, x_{i,\mathrm{pri}}^{\ell,b}\right)$ for $b \in \{0,1\}$), the challenger first checks whether the user i was already corrupted or not. That is, if $i \in \mathcal{S}_x$, then it sends nothing, otheriwse 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 $x_{\mathrm{pub}}^{\ell,0} = x_{\mathrm{pub}}^{\ell,1}$ for all $\ell \in [Q_x]$, and $y_{\mathrm{pub}}^{k,0} = y_{\mathrm{pub}}^{k,1}$ for all $k \in [Q_y]$.

2. For each of the encryption and key challenges, the *private* components of the two challenges are also equal, namely $x_{\mathrm{pri}}^{\ell,0} = x_{\mathrm{pri}}^{\ell,1}$ for all $\ell \in [Q_x]$ whenever $(i, (x^{\ell,0}, x^{\ell,1})) \in \mathcal{Q}_x$ and $i \in \mathcal{S}_x$, and $y_{\mathrm{pri}}^{k,0} = y_{\mathrm{pri}}^{k,1}$ for all $k \in [Q_y]$ whenever $(j, (x^{\ell,0}, x^{\ell,1})) \in \mathcal{Q}_y$ 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 $\left\{x_i^b\right\}, \left\{y_j^b\right\}$ were queried and indices i, j were not corrupted. (Note that if $i \in [n_x]$ or $j \in [n_y]$ were queried to the $\mathsf{Corrupt}$ oracle, the adversary can generate partial keys or ciphertexts for any value of its choice.)

An 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 3.2 (Handling non-challenge queries)**.** Note that in the above security experiment, the adversary is not allowed to make any non-challenge queries directly as every key or encryption query must contain a pair of message/function inputs. We want to point out that this is not a restriction since an adversary can simply send its non-challenge queried input as a challenge query with both the challenge inputs being the same queried input.

Remark 3.3 (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 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.

Feasibility of MPFE for General Circuits. In Appendix B, we provide a general feasibility of MPFE for circuits.

3.1 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+14a], 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 $(n_x, n_y) = (n, 1)$, $y = (f, \perp)$, $x_i = (x_{\mathrm{pub},i}, x_{\mathrm{pri},i}) = (\perp, z_i)$. We also set $\mathsf{Agg}_x(x_1, \ldots, x_n) = (x_{\mathrm{pri},1}, \ldots, x_{\mathrm{pri},n_x}) = (z_1, \ldots, z_n)$. Finally, Agg_y outputs $y_{\mathrm{pub}} = f$ 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+14a, CSG+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 $(n_x, n_y) = (n, 1)$, $y = (f, \perp)$, $x_i = (x_{\mathrm{pub},i}, x_{\mathrm{pri},i}) = (\mathsf{lab}_i, z_i)$. The function $\mathsf{Agg}_x(x_1, \ldots, x_n)$ checks that $x_{\mathrm{pub},1} = \ldots = x_{\mathrm{pub},n}$. If this verification passes, it outputs $(x_{\mathrm{pri},1}, \ldots, x_{\mathrm{pri},n_x}) = (z_1, \ldots, z_n)$. Finally, Agg_y outputs $y_{\mathrm{pub}} = f$ 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 [CSG+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) 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 $(n_x, n_y) = (1, n)$, $x = (f, z)$, $y_i = (\mathsf{lab}_i, \perp)$. The aggregation function Agg_y outputs $(\mathsf{lab}_1, \ldots, \mathsf{lab}_n)$. The aggregation function Agg_x returns (f', z) where $f'(z, \cdot) = z$ 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 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 $y_{\text{pub},1} = f_i = \ldots = y_{\text{pub},n} = f_n$ 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 [CSG+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, 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 CT. The key generator chooses a key space object k and computes a corresponding decryption key DK_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 $(\mathsf{DK}_{k_i})_i$ and ciphertexts $(\mathsf{CT}_i)_i$, and outputs $F((k_i)_i, (m_i)_i)$.

To formally capture DDFE, we require the generalized dynamic MPFE, which is discussed in Appendix A, 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 $y_i = ((F_i, k_i), \perp)$, $x_i = (\perp, m_i)$. The function Agg_x outputs $(m_1, \ldots, m_n)$, and Agg_y checks that $F_1 = F_2 \ldots = F_n$ is the same in all y_i and outputs $(F, k_1, \ldots, k_n)$ if so ($\perp$ otherwise). Upon decryption this yields $\mathcal{U}\big((m_1, \ldots, m_n), (F, (k_1, \ldots, k_n))\big) = F\big((k_i)_i, (m_i)_i\big)$ as desired.

3.2 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 B, 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 B for details.

4 Two Input Quadratic Functional Encryption

In this section, we provide a construction for two party functional encryption for quadratic functions. In more detail, if party 1 encrypts $\mathbf{x}_1 \in R^{\mathsf{N}}$, party 2 encrypts $\mathbf{x}_2 \in R^{\mathsf{N}}$ and the key generator provides a key for $\mathbf{c} \in R^{\mathsf{N}^2}$, the decryption produces $\langle \mathbf{x}_0 \otimes \mathbf{x}_1,\ \mathbf{c} \rangle \in R$. Here R is the ring in which computation is performed.

While our general definition (Section 3) is clearly powerful enough to capture this simple functionality (by setting $\mathsf{mode} = \mathsf{Central}$, $n_x = 2$, $n_y = 1$, $\mathsf{Agg}_x(\mathbf{x}_1, \mathbf{x}_2) = \mathbf{x}_1 \otimes \mathbf{x}_2$, $\mathsf{Agg}_y(\mathbf{c}) = \langle \mathbf{c}, \cdot \rangle$), we provide the restricted definition again here for ease of exposition and verification. For simple functionalities (which may admit constructions from standard assumptions), it is often easier to work with restricted definitions. However, by having a general framework in place, these special cases may be connected back to the general case to chart out the landscape of feasibility results and connections between them.

4.1 Definition of Two Input Quadratic FE

$\mathsf{Setup}(1^\lambda, 1^{\mathsf{N}})$: On input the security parameter 1^λ and the length 1^{N} of the vectors supported by the scheme, the setup algorithm outputs a set of public parameters PK, a master secret key MSK and two encryption keys EK_1 and EK_2.

$\mathsf{KeyGen}(\mathsf{PK}, \mathsf{MSK}, \mathbf{c})$: The key generation algorithm takes as input the public parameters PK, the master secret key MSK, and a quadratic function $\mathbf{c} \in R^{\mathsf{N}^2}$ and outputs a secret key $\mathsf{SK}_{\mathbf{c}}$

$\mathsf{Encrypt}(\mathsf{PK}, \beta, \mathsf{EK}_\beta, \mathbf{x}_\beta)$: The encryption algorithm takes as input the public parameters PK, user index $\beta \in \{1, 2\}$, an encryption key EK_β, an input vector $\mathbf{x}_\beta \in R^{\mathsf{N}}$ and outputs a ciphertext $\mathsf{CT}_{\mathbf{x}}$.

$\mathsf{Decrypt}(\mathsf{PK}, \mathsf{SK}_{\mathbf{c}}, \mathsf{CT}_1, \mathsf{CT}_2)$: The decryption algorithm takes as input the public parameters PK, the secret key $\mathsf{SK}_{\mathbf{c}}$, two ciphertexts CT_1 and CT_2 and outputs a value $y \in R$.

Correctness. We say that a $\mathsf{2QFE}$ scheme is *correct* if for all message $\mathbf{x}_0, \mathbf{x}_1 \in R^{\mathsf{N}}$, function inputs $\mathbf{c} \in R^{\mathsf{N}^2}$, and $\beta \in \{1, 2\}$, it holds that:

$$\Pr\left[z = \langle \mathbf{x}_0 \otimes \mathbf{x}_1,\ \mathbf{c} \rangle : \begin{array}{l} (\mathsf{EK}_1, \mathsf{EK}_2, \mathsf{PK}, \mathsf{MSK}) \leftarrow \mathsf{Setup}(1^\lambda, 1^{\mathsf{N}}) \\ \mathsf{CT}_\beta \leftarrow \mathsf{Encrypt}(\mathsf{PK}, \beta, \mathsf{EK}_\beta, \mathbf{x}_\beta) \\ \mathsf{SK}_{\mathbf{c}} \leftarrow \mathsf{KeyGen}(\mathsf{PK}, \mathsf{MSK}, \mathbf{c}) \\ z \leftarrow \mathsf{Decrypt}\left(\mathsf{PK}, \mathsf{SK}_{\mathbf{c}}, \mathsf{CT}_1, \mathsf{CT}_2\right) \end{array} \right] = 1 \ .$$

Indistinguishability-Based Security. For a 2-input quadratic FE scheme $\mathsf{2QFE}$ as above and adversary $\mathcal{A}$, consider the experiment in Figure 4.1.

Experiment $\mathbf{IND}_{\mathcal{A}}^{\mathsf{2QFE}}(1^\lambda)$
$b \leftarrow \{0, 1\}$
$(\mathsf{EK}_1, \mathsf{EK}_2, \mathsf{MSK}, \mathsf{PK}) \leftarrow \mathsf{Setup}(1^\lambda)$
$b' \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{Enc}}(\cdot,\cdot,\cdot), \mathcal{O}_{\mathsf{Keygen}}(\cdot)}(\mathsf{PK})$
Return $(b = b')$

Oracle $\mathcal{O}_{\mathsf{Enc}}(\beta, \mathbf{x}_0, \mathbf{x}_1)$
$\mathsf{CT}_b \leftarrow \mathsf{Enc}(\mathsf{PK}, \beta, \mathsf{EK}_\beta, \mathbf{x}_b)$
Return CT_b

Oracle $\mathcal{O}_{\mathsf{Keygen}}(\mathbf{c})$
$\mathsf{SK}_{\mathbf{c}} \leftarrow \mathsf{KeyGen}(\mathsf{MSK}, \mathbf{c})$
Return $\mathsf{SK}_{\mathbf{c}}$

Figure 4.1: Experiment for IND-security of two-input quadratic FE (Semi-Honest Setting).

We call $\mathcal{A}$ *legitimate* if it holds that for every key generation query $\mathbf{c}$ and all ciphertext queries $\{\mathbf{u}_1^\ell\}_{\ell\in[Q]}$ to party 1 and $\{\mathbf{u}_2^k\}_{k\in[Q]}$ to party 2 when $b=0$, and $\{\mathbf{v}_1^\ell\}_{\ell\in[Q]}$ to party 1 and $\{\mathbf{v}_2^k\}_{k\in[Q]}$ to party 2 when $b=1$, it holds that:

$$\langle \mathbf{u}_1^\ell \otimes \mathbf{u}_2^k,\ \mathbf{c}\rangle = \langle \mathbf{v}_1^\ell \otimes \mathbf{v}_2^k,\ \mathbf{c}\rangle \ \ \forall \ell, k \in [Q]$$

We assume adversaries are legitimate unless otherwise stated. We call $\mathcal{A}$ *selective* if $\mathcal{A}$ outputs its challenge before it sees the public parameters. We say that $\mathsf{2QFE}$ is IND-secure if for any adversary $\mathcal{A}$

$$\Pr\left(\mathbf{IND}_{\mathcal{A}}^{\mathsf{2QFE}}(\cdot)=1\right) \le 1/2 + \mathsf{negl}(\lambda).$$

4.2 Inner Product Functional Encryption

In this section, we define the primitive of Inner Product Functional Encryption (IPFE) [ABDCP15, ALS16] that we crucially rely on in our core construction. An inner-product functional encryption (IPFE) scheme consists of 4 efficient algorithms:

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

$\mathsf{KeyGen}(\mathsf{MSK}, \mathbf{v}) \to \mathsf{SK}_\mathbf{v}$: The key generation algorithm takes as input the master secret key and a function vector $\mathbf{v} \in \mathbb{Z}_p^\mathsf{N}$, and outputs a secret key $\mathsf{SK}_\mathbf{v}$.

$\mathsf{Encrypt}(\mathsf{PK}, \mathbf{u}) \to \mathsf{CT}_\mathbf{u}$: The encrypt algorithm takes input the master public key and a message vector $\mathbf{u} \in \mathbb{Z}_p^\mathsf{N}$ and outputs a ciphertext $\mathsf{CT}_\mathbf{u}$.

$\mathsf{Decrypt}(\mathsf{SK}_\mathbf{v}, \mathsf{CT}_\mathbf{u}) \to z \vee \perp$: The decrypt algorithm takes as input a secret key $\mathsf{SK}_\mathbf{v}$ and a ciphertext $\mathsf{CT}_\mathbf{u}$, and outputs an element z or $\perp$.

The above primitive can equivalently be defined in the symmetric key setting, where the encryption algorithm also takes the master secret key as input.

Correctness. We say the inner-product functional encryption scheme satisfies decryption correctness if for all $\lambda \in \mathbb{N}$ and all vectors $\mathbf{u}, \mathbf{v} \in \mathbb{Z}_p^\mathsf{N}$,

$$\Pr\left[\mathsf{Decrypt}(\mathsf{SK}_\mathbf{v}, \mathsf{CT}_\mathbf{u}) = \langle \mathbf{u},\ \mathbf{v}\rangle : \begin{array}{r} \mathsf{MSK} \leftarrow \mathsf{Setup}(1^\lambda, 1^\mathsf{N}) \\ \mathsf{SK}_\mathbf{v} \leftarrow \mathsf{KeyGen}(\mathsf{MSK}, \mathbf{v}) \\ \mathsf{CT}_\mathbf{u} \leftarrow \mathsf{Encrypt}(\mathsf{PK}, \mathbf{u}) \end{array}\right] = 1\ .$$

Security. Next, we define security of IPFE. Security captures message hiding in the public key setting and both message and function hiding in the symmetric key setting. Let $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt})$ be an IPFE scheme. Consider the following experiment $\mathsf{Exp}_{\mathsf{FH}}^b$ for $b \in \{0,1\}$.

1. **Setup**: Let $(\mathsf{PK}, \mathsf{MSK}) \leftarrow \mathsf{Setup}(1^\lambda, 1^\mathsf{N})$ and return PK to Adv.
2. **Challenge**: Repeat the following for arbitrarily many rounds determined by Adv: In each round, Adv has 2 options:
   - *Message Hiding.* Adv chooses $\mathbf{u}_i^0, \mathbf{u}_i^1 \in \mathbb{Z}_p^\mathsf{N}$ and submits them for a ciphertext. Upon receiving this, compute $\mathsf{CT}_i \leftarrow \mathsf{Encrypt}(\mathsf{MSK}, \mathbf{u}_i^b)$ and return this to Adv.
   - *Function Hiding.* Adv chooses $\mathbf{v}_j^0, \mathbf{v}_j^1 \in \mathbb{Z}_p^\mathsf{N}$ and submits them for a secret key. Upon receiving this, compute $\mathsf{SK}_j \leftarrow \mathsf{KeyGen}(\mathsf{MSK}, \mathbf{v}_j^b)$ and return this to Adv.

3. **Guess**: Adv outputs its guess b'.

The outcome of the experiment is defined as b' if $\langle \mathbf{u}_i^0,\ \mathbf{v}_j^0\rangle = \langle \mathbf{u}_i^1,\ \mathbf{v}_j^1\rangle$ for all i, j.

The scheme is IND secure with function hiding in the symmetric key setting if $\mathsf{Exp}^0_{\mathsf{FH}} \approx \mathsf{Exp}^1_{\mathsf{FH}}$. In the public key setting, the adversary does not make function hiding queries, i.e. $\mathbf{v}_j^0 = \mathbf{v}_j^1$ for all key queries j. In the selective game, the adversary declares the challenge messages in the very first step, before receiving the public parameters.

Canonical Form. The works of [LV16, Lin17] constructed function hiding inner product encryption based on the SXDH assumption that satisfy the following canonical form:

1. Its ciphertext or secret key consist of only encodings in group $\mathbb{G}_1$ or $\mathbb{G}_2$ respectively of ring elements that depend linearly in the encoded vector $\mathbf{v}$.
2. The setup, key generation, and encryption algorithms do not use pairing nor the target group $\mathbb{G}_T$.
3. The decryption algorithm homomorphically evaluates a degree 2 polynomial (namely inner product), on the encodings in the secret key and ciphertext, and then zero-tests the output encoding.

Review of the ABDP IPFE scheme [ABDCP15]. Next, we review the DDH based, selectively secure IPFE scheme constructed by Abdalla et al. [ABDCP15]. Let $\mathbb{G}$ be a group of prime order p on which DDH is hard. The ABDP scheme may be summarized as:

$\mathsf{Setup}(1^\lambda, 1^{\mathsf{N}}) \to (\mathsf{PK}, \mathsf{MSK})$: The setup algorithm takes as input the security parameter and the length of the supported vectors, samples $\mathbf{s} \leftarrow \mathbb{Z}_p^{\mathsf{N}}$ and computes $[\mathbf{s}]$. It outputs $\mathsf{PK} = [\mathbf{s}]$ and $\mathsf{MSK} = \mathbf{s}$.

$\mathsf{KeyGen}(\mathsf{MSK}, \mathbf{v}) \to \mathsf{SK}_{\mathbf{v}}$: The key generation algorithm takes as input the master secret key and a function vector $\mathbf{v} \in \mathbb{Z}_p^{\mathsf{N}}$, and outputs a secret key $\mathsf{SK}_{\mathbf{v}} = (\langle \mathbf{v}, \mathbf{s}\rangle,\ \mathbf{v})$.

$\mathsf{Encrypt}(\mathsf{PK}, \mathbf{u}) \to \mathsf{CT}_{\mathbf{u}}$: The encrypt algorithm takes input the master public key $[\mathbf{s}]$ and a message vector $\mathbf{u} \in \mathbb{Z}_p^{\mathsf{N}}$ and does the following:

1. It samples random $r \leftarrow \mathbb{Z}_p$.
2. It computes a ciphertext $\mathsf{CT}_{\mathbf{u}} = ([-r], [r \cdot \mathbf{s} + \mathbf{u}])$ and outputs it.

$\mathsf{Decrypt}(\mathsf{SK}_{\mathbf{v}}, \mathsf{CT}_{\mathbf{u}}) \to z \vee \perp$: The decrypt algorithm takes as input a secret key $\mathsf{SK}_{\mathbf{v}}$ and a ciphertext $\mathsf{CT}_{\mathbf{u}}$, and computes $[z] = \langle \mathsf{SK}_{\mathbf{v}},\ \mathsf{CT}_{\mathbf{u}}\rangle$. It then recovers z from the exponent by brute force discrete log and outputs it.

We will use the algebraic structure of the above scheme crucially in our construction of two party quadratic FE.

4.3 Construction

Our two input FE for quadratic functions is made up of the following ingredients:

1. A function hiding inner product encryption scheme cIPE which has a *canonical* form. In particular, its ciphertexts and secret keys encode the input and function vectors in different source groups $\mathbb{G}_1$, $\mathbb{G}_2$ of the bilinear map, and decryption simply uses pairing to produce an encoding of the output inner product in the target group $\mathbb{G}_T$. Such a function hiding IPE was constructed by [LV16, Lin17]. Please see Section 4.2 for the formal definitions.

2. The inner product encryption scheme IPE provided by [ABDCP15]. Please see Section 4.2 for a review.

These schemes are interleaved in a *non black-box* way to obtain the final result.

Since the intuition was discussed in Section 1, we proceed directly with the construction.

$\mathsf{Setup}(1^\lambda, 1^\mathsf{N})$: On input the security parameter 1^λ and the length 1^N of the vectors supported by the scheme, the setup algorithm proceeds as follows.

1. Sample two "inner" function hiding inner product schemes with canonical form as:

$$(\mathsf{cPK}_1, \mathsf{cMSK}_1) \leftarrow \mathsf{cIPE.Setup}(1^\lambda, 1^{5\cdot\mathsf{N}}), \quad (\mathsf{cPK}_2, \mathsf{cMSK}_2) \leftarrow \mathsf{cIPE.Setup}(1^\lambda, 1^2)$$

2. Sample an "outer" IPE scheme as: $(\mathsf{iPK}, \mathsf{iMSK}) \leftarrow \mathsf{IPE.Setup}(1^\lambda, 1^{\mathsf{N}^2})$
   Let $\mathbf{S} = \mathsf{iMSK} \in R^{\mathsf{N}\times\mathsf{N}}$ be secret shared as $\mathbf{S} = \mathbf{S}_1 \cdot \mathbf{S}_2$. Let $\mathbf{s}_{1,2} \in R^{\mathsf{N}^2}$ be the vectorized form of $\mathbf{S}$.
3. Output the public and master secret keys as

$$\mathsf{2QFE.PK} = (\mathsf{iPK}, \mathsf{cPK}_1, \mathsf{cPK}_2), \quad \mathsf{2QFE.MSK} = \mathbf{S}$$
$$\mathsf{2QFE.EK}_\beta = (\mathbf{S}_\beta, \mathsf{cMSK}_1, \mathsf{cMSK}_2) \ \ \forall \beta \in \{1, 2\}$$

$\mathsf{KeyGen}(\mathsf{PK}, \mathsf{MSK}, \mathbf{c})$: The key generation algorithm takes as input the public parameters PK, the master secret key MSK, and a quadratic function $\mathbf{c} \in R^{\mathsf{N}^2}$ and proceeds as follows.

1. Vectorize $\mathbf{S}$ to make it a vector $\mathbf{s}_{1,2}$ of length N^2.
2. Output $\mathsf{iSK}_\mathbf{c} = \big(\mathbf{c},\ \langle \mathbf{c},\ \mathbf{s}_{1,2}\rangle\big)$.

$\mathsf{Encrypt}(\mathsf{PK}, \beta, \mathsf{EK}_\beta, \mathbf{x}_\beta)$: The encryption algorithm takes as input the public parameters PK, user index $\beta \in \{1, 2\}$, an encryption key EK_β, an input vector $\mathbf{x} \in R^\mathsf{N}$ and proceeds as follows.

1. **Encode the input vector.**
   (a) If $\beta = 1$, encode $\mathbf{x}_1$ into N vectors of size N so that the i^{th} vector has the i^{th} co-ordinate of $\mathbf{x}_1$ repeated N times. Formally, let $\hat{\mathbf{x}}_{1,i} = (\mathbf{x}_1[i], \ldots, \mathbf{x}_1[i])$.
   (b) If $\beta = 2$, encode $\mathbf{x}_2$ into N vectors of size N so that the i^{th} vector has $\mathbf{x}_2[i]$ as the i^{th} co-ordinate and 0 elsewhere. Formally, let $\hat{\mathbf{x}}_{2,i} = (0, \ldots, \mathbf{x}_2[i], 0, \ldots, 0)$.
2. **Compute cIPE encodings.**
   (a) For $\beta \in \{1, 2\}$, sample a fresh random scalar $r_\beta \in \mathbb{Z}_p$.
   (b) For $\beta \in \{1, 2\}$, vectorize $\mathbf{S}_\beta$ into a vector $\mathbf{s}_\beta = (\mathbf{s}_{\beta,1}, \ldots, \mathbf{s}_{\beta,\mathsf{N}})$ such that $\mathbf{S}[i, j] = \langle \mathbf{s}_{1,i}, \mathbf{s}_{2,j}\rangle$ for $i, j \in [\mathsf{N}]$.
   (c) Let $\mathbf{0}$ be the all-zeros vector of length $3 \cdot \mathsf{N}$[4]. For $\beta \in \{1, 2\}$, $i, j \in [\mathsf{N}]$, set the inner message vector as:
$$\chi_{\beta,i} = (\hat{\mathbf{x}}_{\beta,i} \| r_\beta \cdot \mathbf{s}_{\beta,i}), \quad \text{let} \quad \boldsymbol{\chi}_\beta = (\chi_{\beta,i})_{i\in[\mathsf{N}]}$$
   (d) For $i \in [\mathsf{N}]$, if $\beta = 1$, compute the inner cIPE encoding as:
$$\mathsf{cCT}_{1,i} = \mathsf{cIPE.Encrypt}(\mathsf{cMSK}_1, \chi_{1,i} \| \mathbf{0}), \quad \mathsf{cCT}_2 = \mathsf{cIPE.Encrypt}(\mathsf{cMSK}_2, r_1 \| 0)$$
   (e) For $i \in [\mathsf{N}]$, if $\beta = 2$, compute the inner cIPE encoding as:
$$\mathsf{cSK}_{1,i} = \mathsf{cIPE.KeyGen}(\mathsf{cMSK}_1, \chi_{2,i} \| \mathbf{0}), \quad \mathsf{cSK}_2 = \mathsf{cIPE.KeyGen}(\mathsf{cMSK}_2, r_2 \| 0)$$
3. **Output.** If $\beta = 1$, output $\Big(\{\mathsf{cCT}_{1,i}\}_{i\in[\mathsf{N}]}, \mathsf{cCT}_2\Big)$, else if $\beta = 2$, output $\Big(\{\mathsf{cSK}_{1,i}\}_{i\in[\mathsf{N}]}, \mathsf{cSK}_2\Big)$.

[4]This is required for the proof and plays no role in the real world.

$\mathsf{Decrypt}(\mathsf{PK}, \mathsf{iSK}_{\mathbf{c}}, \mathsf{CT}_1, \mathsf{CT}_2)$: The decryption algorithm takes as input the public parameters PK, the secret key $\mathsf{SK}_{\mathbf{c}}$, two ciphertexts CT_0 and CT_1 and proceeds as follows.

1. Parse $\mathsf{CT}_1 = \left(\left\{\mathsf{cCT}_{1,i}\right\}_{i\in[\mathsf{N}]}, \mathsf{cCT}_2\right)$ and $\mathsf{CT}_2 = \left(\left\{\mathsf{cSK}_{1,i}\right\}_{i\in[\mathsf{N}]}, \mathsf{cSK}_2\right)$.
2. For $i, j \in [\mathsf{N}]$, compute all pairwise cIPE decryptions:
$$\mathsf{iCT}_{1,i,j} = \mathsf{cIPE.Decrypt}(\mathsf{cCT}_{1,i}, \mathsf{cSK}_{1,j})$$
Let $\mathsf{iCT}_1 = (\mathsf{iCT}_{i,j})_{i,j\in[\mathsf{N}]}$.
3. Compute $\mathsf{iCT}_2 = \mathsf{cIPE.Decrypt}(\mathsf{cCT}_2, \mathsf{cSK}_2)$. Let $\mathsf{iCT} = (\mathsf{iCT}_1, \mathsf{iCT}_2)$.
4. Compute decryption of the outer IPE as $\mathsf{IPE.Decrypt}(\mathsf{iCT}, \mathsf{iSK}_{\mathbf{c}})$ and output it.

Correctness. We now establish that the scheme is correct. To begin, note that by correctness and the canonical form of cIPE, we have that:

$$\mathsf{iCT}_{1,i,j} = \left[\langle \hat{\mathbf{x}}_{1,i},\ \hat{\mathbf{x}}_{2,j}\rangle + r_1 r_2 \langle \mathbf{s}_{1,i},\ \mathbf{s}_{2,j}\rangle\right]_T = \left[\mathbf{x}_1[i]\mathbf{x}_2[j] + r_1 r_2 \mathbf{S}[i,j]\right]_T$$

Let $\mathbf{s}_{1,2} \in R^{\mathsf{N}^2}$ be the vectorized form of $\mathbf{S} = \mathbf{S}_1 \cdot \mathbf{S}_2$. Then, we have:

$$\mathsf{iCT}_1 = [\mathbf{x}_1 \otimes \mathbf{x}_2 + r_1 r_2\ \mathbf{s}_{1,2}]_T, \qquad \mathsf{iCT}_2 = [r_1 r_2]_T$$

Next, we perform the outer IPE decryption. Given, $\mathbf{c}$ and $\langle \mathbf{c},\ \mathbf{s}_{1,2}\rangle$, we compute:

$$[\langle \mathbf{c},\ \mathbf{x}_1 \otimes \mathbf{x}_2\rangle + r_1 r_2\ \langle \mathbf{c},\ \mathbf{s}_{1,2}\rangle]_T - [r_1 r_2 \cdot \langle \mathbf{c},\ \mathbf{s}_{1,2}\rangle]_T = [\langle \mathbf{c},\ \mathbf{x}_1 \otimes \mathbf{x}_2\rangle]_T$$

If the resultant function value is polynomially bound, then by taking discrete log, we recover $\langle \mathbf{c},\ \mathbf{x}_1 \otimes \mathbf{x}_2\rangle$ as desired.

4.4 Security

Theorem 4.1. Assume that cIPE is a function hiding inner product encryption scheme and that IPE is an inner product encryption scheme. Then the scheme $\mathsf{2QFE}$ is a two input functional encryption scheme for quadratic functions satisfying selective indistinguishability based security (Definition 4.1).

Proof. We assume that the adversary makes Q ciphertext queries to party 1 and party 2. We denote these by $\mathbf{u}_1^1, \ldots \mathbf{u}_1^Q \in R^{\mathsf{N}}$ and $\mathbf{u}_2^1, \ldots \mathbf{u}_2^Q \in R^{\mathsf{N}}$ for challenge bit $b = 0$ and $\mathbf{v}_1^1, \ldots \mathbf{v}_1^Q \in R^{\mathsf{N}}$ and $\mathbf{v}_2^1, \ldots \mathbf{v}_2^Q \in R^{\mathsf{N}}$ for challenge bit $b = 1$. Note that for an admissible adversary, for any requested key $\mathbf{c} \in R^{\mathsf{N}^2}$, and for all $\ell, k \in [Q]$, we have that

$$\langle \mathbf{c},\ \mathbf{u}_1^\ell \otimes \mathbf{u}_2^k\rangle = \langle \mathbf{c},\ \mathbf{v}_1^\ell \otimes \mathbf{v}_2^k\rangle \tag{4.1}$$

Let the corresponding random scalars chosen by the two parties in Step 2a as $r_1^1, \ldots, r_1^Q$ and $r_2^1, \ldots, r_2^Q$. Let the inner encoded messages corresponding to Step 2c for parties 1 and 2 be $\boldsymbol{\mu}_1^1, \ldots \boldsymbol{\mu}_1^Q$ and $\boldsymbol{\mu}_2^1, \ldots, \boldsymbol{\mu}_2^Q$ for challenge bit $b = 0$ and $\boldsymbol{\nu}_1^1, \ldots \boldsymbol{\nu}_1^Q$ and $\boldsymbol{\nu}_2^1, \ldots, \boldsymbol{\nu}_2^Q$ for challenge bit $b = 1$. Here each $\boldsymbol{\mu}_\beta^\ell$, $\boldsymbol{\nu}_\beta^k$ encodes a vector of length N for $\ell, k \in [Q]$.

We denote by $\mathsf{iCT}_b^{\ell,k}$ the outer IPE ciphertext that is constructed by combining the ℓ^{th} ciphertext of party 1 with the k^{th} ciphertext of party 2 when challenge bit is b. In more detail, $\mathsf{iCT}_0^{\ell,k}$ is constructed by pairing $\mathsf{cCT}(\boldsymbol{\mu}_{1,i}^\ell)$ and $\mathsf{cSK}(\boldsymbol{\mu}_{2,j}^k)$ while $\mathsf{iCT}_1^{\ell,k}$ is constructed by pairing $\mathsf{cCT}(\boldsymbol{\nu}_{1,i}^\ell)$ and $\mathsf{cSK}(\boldsymbol{\nu}_{2,j}^k)$ for all $i, j \in [\mathsf{N}]$.

Due to the structure of the IPE decryption equation, Equation 4.1 also implies that for all $\ell, k \in [Q]$:

$$\langle \mathsf{iSK}_{\mathbf{c}},\ \mathsf{iCT}_0^{\ell,k}\rangle = \langle \mathsf{iSK}_{\mathbf{c}},\ \mathsf{iCT}_1^{\ell,k}\rangle \tag{4.2}$$

For notational brevity we will refer to the pairing of $\mathsf{cCT}(\boldsymbol{\mu}_{1,i}^\ell)$ and $\mathsf{cSK}(\boldsymbol{\mu}_{2,j}^k)$ for all i, j as simply the pairing of $\mathsf{cCT}(\boldsymbol{\mu}_1^\ell)$ and $\mathsf{cSK}(\boldsymbol{\mu}_2^k)$ (analogously $\mathsf{cCT}(\boldsymbol{\nu}_1^\ell)$ and $\mathsf{cSK}(\boldsymbol{\nu}_2^k)$).

Hybrids. Our proof is structured as a sequence of hybrids that progress from the real world with challenge bit $b = 0$ encoding the Q^2 pairs

$$\mathsf{cCT}(\boldsymbol{\mu}_1^1)\big(\mathsf{cSK}(\boldsymbol{\mu}_2^1), \ldots, \mathsf{cSK}(\boldsymbol{\mu}_2^Q)\big), \ldots \mathsf{cCT}(\boldsymbol{\mu}_1^Q)\big(\mathsf{cSK}(\boldsymbol{\mu}_2^1), \ldots, \mathsf{cSK}(\boldsymbol{\mu}_2^Q)\big)$$

to the real world with challenge bit $b = 1$ encoding the Q^2 pairs

$$\mathsf{cCT}(\boldsymbol{\nu}_1^1)\big(\mathsf{cSK}(\boldsymbol{\nu}_2^1), \ldots, \mathsf{cSK}(\boldsymbol{\nu}_2^Q)\big), \ldots \mathsf{cCT}(\boldsymbol{\nu}_1^Q)\big(\mathsf{cSK}(\boldsymbol{\nu}_2^1), \ldots, \mathsf{cSK}(\boldsymbol{\nu}_2^Q)\big)$$

For notational brevity, we drop the notation cCT and cSK above and denote the vector $\Big(\mathsf{cSK}(\boldsymbol{\mu}_2^1), \ldots, \mathsf{cSK}(\boldsymbol{\mu}_2^Q)\Big)$ as $\overrightarrow{\boldsymbol{\mu}}_2 = \Big(\boldsymbol{\mu}_2^1, \ldots, \boldsymbol{\mu}_2^Q\Big)$, and analogously $\overrightarrow{\boldsymbol{\nu}}_2 = \Big(\boldsymbol{\nu}_2^1, \ldots, \boldsymbol{\nu}_2^Q\Big)$. We design $2Q$ hybrids, H_b^ρ for $(\rho, b) \in [Q] \times \{0, 1\}$ as:

In Hybrid H_0^ρ, the Q^2 cross terms are:

$$\boldsymbol{\nu}_1^1 \overrightarrow{\boldsymbol{\nu}}_2, \ldots, \boldsymbol{\nu}_1^{\rho-1} \overrightarrow{\boldsymbol{\nu}}_2, \ \underline{\boldsymbol{\mu}_1^\rho \overrightarrow{\boldsymbol{\mu}}_2}, \ \boldsymbol{\mu}_1^{\rho+1} \overrightarrow{\boldsymbol{\mu}}_2, \ldots \boldsymbol{\mu}_1^Q \overrightarrow{\boldsymbol{\mu}}_2 \tag{4.3}$$

In Hybrid H_1^ρ, the Q^2 cross terms are:

$$\boldsymbol{\nu}_1^1 \overrightarrow{\boldsymbol{\nu}}_2, \ldots, \boldsymbol{\nu}_1^{\rho-1} \overrightarrow{\boldsymbol{\nu}}_2, \ \underline{\boldsymbol{\nu}_1^\rho \overrightarrow{\boldsymbol{\nu}}_2}, \ \boldsymbol{\mu}_1^{\rho+1} \overrightarrow{\boldsymbol{\mu}}_2, \ldots \boldsymbol{\mu}_1^Q \overrightarrow{\boldsymbol{\mu}}_2 \tag{4.4}$$

To achieve this, design Hybrid H_b^ρ as follows.
The Q CTs of party 1 for $\ell \in [Q]$:

$$\begin{aligned}
&[r_1^\ell \parallel 0], \ \{ \ [\ \mathbf{0} \parallel \boldsymbol{\nu}_{1,i}^\ell \parallel 0 \parallel \ldots, \parallel 0 \ ] \ \}_{i \in [\mathsf{N}]} && \text{when} \quad \ell < \rho \\
&[r_1^\ell \parallel 0], \ \{ \ [\ \boldsymbol{\mu}_{1,i}^\ell \parallel \mathbf{0} \parallel 0 \parallel \ldots, \parallel 0 \ ] \ \}_{i \in [\mathsf{N}]} && \text{when} \quad \ell > \rho \\
&[0 \parallel 1], \{ \ [\ \mathbf{0} \parallel \mathbf{0} \parallel 0 \parallel \ldots \parallel 1 \parallel \ldots \parallel 0 \ ] \ \}_{i \in [\mathsf{N}]} && \text{when} \quad \ell = \rho
\end{aligned}$$

Above, when $\ell = \rho$, the 1 is in the i^{th} position of the last N co-ordinates (i.e. the last N co-ordinates form the i^{th} basis vector).

The Q CTs of party 2 for $k \in [Q]$:

$$\begin{aligned}
&[r_2^k \parallel r_1^\rho r_2^k], \ \Big\{ \ \Big[\boldsymbol{\mu}_{2,j}^k \parallel \boldsymbol{\nu}_{2,j}^k \parallel \langle \boldsymbol{\mu}_{1,1}^\rho, \ \boldsymbol{\mu}_{2,j}^k \rangle \| \ldots, \| \langle \boldsymbol{\mu}_{1,\mathsf{N}}^\rho, \ \boldsymbol{\mu}_{2,j}^k \rangle \ \Big] \ \Big\}_{j \in [\mathsf{N}]} && \text{when} \quad b = 0 \\
&[r_2^k \parallel r_1^\rho r_2^k], \ \Big\{ \ \Big[\boldsymbol{\mu}_{2,j}^k \parallel \boldsymbol{\nu}_{2,j}^k \parallel \langle \boldsymbol{\nu}_{1,1}^\rho, \ \boldsymbol{\nu}_{2,j}^k \rangle \| \ldots, \| \langle \boldsymbol{\nu}_{1,\mathsf{N}}^\rho, \ \boldsymbol{\nu}_{2,j}^k \rangle \ \Big] \ \Big\}_{j \in [\mathsf{N}]} && \text{when} \quad b = 1
\end{aligned}$$

It is easy to verify that the constraints imposed by equations 4.3 and 4.4 are satisfied. In particular, when $\ell = \rho$, the ℓ^{th} ciphertext of party 1 computes $\boldsymbol{\mu}_1^\rho \boldsymbol{\mu}_2^k$ for all $k \in [Q]$, i.e. $\boldsymbol{\mu}_1^\rho \overrightarrow{\boldsymbol{\mu}}_2$ when $b = 0$ and $\boldsymbol{\nu}_1^\rho \overrightarrow{\boldsymbol{\nu}}_2$ when $b = 1$ as desired.

Indistinguishability of Hybrids. It is easy to see that H_1^ρ and $\mathsf{H}_0^{\rho+1}$ are indistinguishable by cIPE security, for $\rho \in [Q]$. To see this, observe that by Equations 4.3 and 4.4, the inner products generated by the Q^2 cross terms are exactly the same, namely:

$$\boldsymbol{\nu}_1^1 \overrightarrow{\boldsymbol{\nu}}_2, \ldots, \boldsymbol{\nu}_1^{\rho-1} \overrightarrow{\boldsymbol{\nu}}_2, \ \underline{\boldsymbol{\nu}_1^\rho \overrightarrow{\boldsymbol{\nu}}_2}, \ \boldsymbol{\mu}_1^{\rho+1} \overrightarrow{\boldsymbol{\mu}}_2, \ldots \boldsymbol{\mu}_1^Q \overrightarrow{\boldsymbol{\mu}}_2$$

In more detail, we show that:

Claim 4.2. If cIPE_1 and cIPE_2 satisfy function hiding security (Definition 4.2) then H_1^ρ and $\mathsf{H}_0^{\rho+1}$ are indistinguishable.

Proof. The proof proceeds in two steps, via a reduction to cIPE_1 and cIPE_2. We provide the first reduction for cIPE_1, the case of cIPE_2 is analogous. In more detail, assume there exists an adversary Adv who can distinguish between H_1^ρ and $\mathsf{H}_0^{\rho+1}$. Then, we construct an adversary $\mathcal{B}$ who can break the function hiding property of cIPE_1. $\mathcal{B}$ does the following:

1. It samples the public and master key of the outer IPE scheme, namely $[\mathbf{S}], \mathbf{S}$ and uses this to simulate IPE secret keys to Adv.
2. It samples the public and master key of the scheme cIPE_2 and uses these to generate cCT_2 and cSK_2 as in the previous hybrid.
3. From the cIPE challenger, it makes ciphertext queries to obtain :

$$\begin{aligned} &\left\{ \mathsf{cCT} \left[\mathbf{0} \parallel \boldsymbol{\nu}_{1,i}^{\ell} \parallel 0 \parallel \ldots, \parallel 0 \right] \right\}_{i \in [\mathsf{N}]} \quad \text{when} \quad \ell < \rho \\ &\left\{ \mathsf{cCT} \left[\boldsymbol{\mu}_{1,i}^{\ell} \parallel \mathbf{0} \parallel 0 \parallel \ldots, \parallel 0 \right] \right\}_{i \in [\mathsf{N}]} \quad \text{when} \quad \ell > \rho \end{aligned}$$

4. For the challenge, for $i, j \in [\mathsf{N}]$, $k \in [Q]$ it provides:
   (a) For $b = 0$, ciphertext vectors are:

$$\{[\mathbf{0} \parallel \mathbf{0} \parallel 0 \parallel \ldots \parallel 1_i \parallel \ldots \parallel 0]\}_i,$$

   and function vectors are:

$$\{[\boldsymbol{\mu}_{2,j}^k \parallel \boldsymbol{\nu}_{2,j}^k \parallel \langle \boldsymbol{\nu}_{1,1}^\rho, \boldsymbol{\nu}_{2,j}^k \rangle \parallel \ldots, \parallel \langle \boldsymbol{\nu}_{1,\mathsf{N}}^\rho, \boldsymbol{\nu}_{2,j}^k \rangle]\}_j$$

   (b) For $b = 1$, ciphertext vectors are:

$$\{[\mathbf{0} \parallel \boldsymbol{\nu}_{1,i}^\rho \parallel \ 0 \parallel \ldots \parallel 0_i \parallel \ldots \parallel 0 \ ]\}_i,$$

   and function vectors are:

$$\{ [\boldsymbol{\mu}_{2,j}^k \parallel \boldsymbol{\nu}_{2,j}^k \parallel \langle \boldsymbol{\mu}_{1,1}^\rho, \boldsymbol{\mu}_{2,j}^k \rangle \parallel \ldots, \parallel \langle \boldsymbol{\mu}_{1,\mathsf{N}}^\rho, \boldsymbol{\mu}_{2,j}^k \rangle]\}_j$$

   Since the inner products are the same, the reduction is an admissible cIPE adversary. If it receives cIPE ciphertext and key for the former it is in the first hybrid H_1^ρ, else it is in the second hybrid $\mathsf{H}_0^{\rho+1}$. Thus, if the adversary can distinguish between these two hybrids, the reduction can break cIPE security.

□

Also, note that the real world with the challenge bit $b = 0$ is indistinguishable from H_0^1 by the above argument.

Showing that H_0^ρ and H_1^ρ are indistinguishable is more subtle, and we rely on the outer IPE security for this.

Claim 4.3. If the ABDP public key IPE scheme satisfies selective IND security (Definition 4.2) then H_0^ρ and H_1^ρ are indistinguishable.

Proof. To argue this claim, we provide two intermediate hybrids, where we replace the terms $[r_1^\rho r_2^1], \ldots, [r_1^\rho r_2^Q]$ by random terms $[r_{12}^{\rho,1}], \ldots, [r_{12}^{\rho,Q}]$ by the SXDH assumption. We then reduce to the security of the ABDP scheme. A technical hurdle that arises is that the ABDP challenger only returns $[\mathbf{S}]$ as part of the IPE public key, but the reduction must simulate ciphertexts for both parties, which involve elements containing only $[\mathbf{S}_1]$ and $[\mathbf{S}_2]$ individually. To handle this, the reduction samples a dummy $\mathbf{T}_1$ to play the role of $\mathbf{S}_1$ and uses knowledge of $\mathbf{T}_1$ and $[\mathbf{S} = \mathbf{S}_1\mathbf{S}_2]$ to solve for $[\mathbf{T}_2]$ that satisfies all the requisite relations for decryption.

Hybrid $\mathsf{H}_0^\rho[1]$. In this hybrid, we replace the terms $[r_1^\rho r_2^1], \ldots, [r_1^\rho r_2^Q]$ by random terms $[r_{12}^{\rho,1}], \ldots, [r_{12}^{\rho,Q}]$ by the SXDH assumption. The reduction receives the SXDH challenge:

$$\left\{[r_1^\rho], [r_2^1], \ldots [r_2^Q], [r_1^\rho r_1^1], \ldots, [r_1^\rho r_2^Q]\right\} \quad \text{or} \quad \left\{[r_1^\rho], [r_2^1], \ldots [r_2^Q], [r_{12}^{\rho,1}], \ldots, [r_{12}^{\rho,Q}]\right\}$$

Note that r_1^ρ is not used in the generation of party 1's ciphertexts and these are generated as in the previous hybrid. The terms that involve the SXDH challenge are in party 2's ciphertexts:

$$[r_2^\ell \parallel \underline{r_1^\rho r_2^\ell}], \left\{ \left[\boldsymbol{\mu}_{2,j}^\ell \parallel \boldsymbol{\nu}_{2,j}^\ell \parallel \underline{\langle \boldsymbol{\mu}_{1,1}^\rho, \boldsymbol{\mu}_{2,j}^\ell \rangle} \| \ldots, \| \underline{\langle \boldsymbol{\mu}_{1,\mathsf{N}}^\rho, \boldsymbol{\mu}_{2,j}^\ell \rangle} \right] \right\}_{j \in [\mathsf{N}]}$$

Here, $\langle \boldsymbol{\mu}_{1,i}^\rho, \boldsymbol{\mu}_{2,j}^\ell \rangle = \hat{\mathbf{u}}_1^\rho[i]\ \hat{\mathbf{u}}_2^\ell[j] + \underline{r_1^\rho r_2^\ell} \cdot \mathbf{S}[i,j]$. The reduction uses the SXDH challenge to set these terms. In more detail, for $j \in [\mathsf{N}]$, we have the ciphertext components as:

$$\begin{aligned}
&\left[\boldsymbol{\mu}_{2,j}^\ell \parallel \boldsymbol{\nu}_{2,j}^\ell \parallel \hat{\mathbf{u}}_1^\rho[1]\ \hat{\mathbf{u}}_2^\ell[j] + \underline{r_{12}^{\rho,\ell}} \cdot \mathbf{S}[1,j] \| \ldots, \| \hat{\mathbf{u}}_1^\rho[\mathsf{N}]\ \hat{\mathbf{u}}_2^\ell[j] + \underline{r_{12}^{\rho,\ell}} \cdot \mathbf{S}[\mathsf{N},j] \right] \\
&= \left[\boldsymbol{\mu}_{2,j}^\ell \parallel \boldsymbol{\nu}_{2,j}^\ell \parallel \hat{\mathbf{u}}_1^\rho[1]\ \hat{\mathbf{u}}_2^\ell[j] + \underline{r_{12}^{\rho,\ell}} \cdot \langle \mathbf{s}_{1,1}, \mathbf{s}_{2,j} \rangle \| \ldots, \| \hat{\mathbf{u}}_1^\rho[\mathsf{N}]\ \hat{\mathbf{u}}_2^\ell[j] + \underline{r_{12}^{\rho,\ell}} \cdot \langle \mathbf{s}_{1,\mathsf{N}}, \mathbf{s}_{2,j} \rangle \right] \qquad (4.5)
\end{aligned}$$

It follows that if an adversary can distinguish between Hybrid H_0^ρ and Hybrid $\mathsf{H}_0^\rho[1]$, the reduction can break the SXDH challenge.

Hybrid $\mathsf{H}_0^\rho[2]$. This hybrid analogous to Hybrid $\mathsf{H}_0^\rho[1]$, except that the Q ciphertexts for party 2 now have:

$$[\ r_2^\ell \parallel \underline{r_{12}^{\rho,\ell}}\ ], \left\{ \left[\boldsymbol{\mu}_{2,j}^\ell \parallel \boldsymbol{\nu}_{2,j}^\ell \parallel \langle \boldsymbol{\nu}_{1,1}^\rho, \boldsymbol{\nu}_{2,j}^\ell \rangle \| \ldots, \| \langle \boldsymbol{\nu}_{1,\mathsf{N}}^\rho, \boldsymbol{\nu}_{2,j}^\ell \rangle \right] \right\}_{j \in [\mathsf{N}]}$$

The second term can be written as:

$$\left\{ \left[\boldsymbol{\mu}_{2,j}^\ell \parallel \boldsymbol{\nu}_{2,j}^\ell \parallel \hat{\mathbf{v}}_1^\rho[1]\ \hat{\mathbf{v}}_2^\ell[j] + \underline{r_{12}^{\rho,\ell}} \cdot \mathbf{S}[1,j] \| \ldots, \| \hat{\mathbf{v}}_1^\rho[\mathsf{N}]\ \hat{\mathbf{v}}_2^\ell[j] + \underline{r_{12}^{\rho,\ell}} \cdot \mathbf{S}[\mathsf{N},j] \right] \right\}_{j \in [\mathsf{N}]}$$

It is easy to see that Hybrid $\mathsf{H}_0^\rho[2]$ is indistinguishable from Hybrid H_1^ρ by the SXDH using the exact same argument as above.

Hence, it suffices to show that Hybrid $\mathsf{H}_0^\rho[1]$ is indistinguishable from Hybrid $\mathsf{H}_0^\rho[2]$. We do so by relying on the security of IPE. In more detail, we construct a reduction $\mathcal{B}$ that plays against the ABDP IPE challenger, and succeeds in breaking IPE security if there exists an adversary $\mathcal{A}$ who can distinguish between Hybrids $\mathsf{H}_0^\rho[1]$ and $\mathsf{H}_0^\rho[2]$.

The reduction $\mathcal{B}$ proceeds as follows:

1. $\mathcal{B}$ samples the cIPE schemes to obtain cMSK_1 and cMSK_2 for constructing the CTs for party 1 and 2.
2. To compute the ciphertexts for party 1, do the following:
   (a) Generate a dummy $\mathbf{T}_1$ to substitute for $\mathbf{S}_1$, from the same distribution as $\mathbf{S}_1$.
   (b) Sample r_1^ℓ for $\ell \in [Q]$, $\ell \neq \rho$.

(c) Compute party 1's ciphertexts using the dummy $\mathbf{T}_1$ as before, namely for $\ell \in [Q]$:

$$\begin{aligned}
&[r_1^\ell \,\|\, 0],\ \{\ [\ \mathbf{0} \,\|\, \boldsymbol{\nu}_{1,i}^\ell \,\|\, 0 \,\|\, \ldots, \|\, 0\ ]\ \}_{i\in[\mathsf{N}]} && \text{when} \quad \ell < \rho \\
&[r_1^\ell \,\|\, 0],\ \{\ [\ \boldsymbol{\mu}_{1,i}^\ell \,\|\, \mathbf{0} \,\|\, 0 \,\|\, \ldots, \|\, 0\ ]\ \}_{i\in[\mathsf{N}]} && \text{when} \quad \ell > \rho \\
&[0 \,\|\, 1], \{\ [\ \mathbf{0} \,\|\, \mathbf{0} \,\|\, 0 \,\|\ldots\| \,1 \,\|\ldots\| \,0\ ]\ \}_{i\in[\mathsf{N}]} && \text{when} \quad \ell = \rho
\end{aligned}$$

3. To compute party 2's ciphertexts, do the following:
   (a) Sample random scalars $r_2^1, \ldots, r_2^Q$. Using these, compute $[r_2^\ell]$ for $\ell \in [Q]$.
   (b) To compute $[\boldsymbol{\mu}_2^\ell]$ and $[\boldsymbol{\nu}_2^\ell]$ for $\ell \in [Q]$, do the following:
      i. Invoke the IPE challenger to obtain the public key $[\mathbf{S}] = [\mathbf{S}_1 \cdot \mathbf{S}_2]$.
      ii. Let $[\mathbf{T}_2] = [\mathbf{T}_1^{-1} \cdot \mathbf{S}]$. Note that $\mathbf{T}_1$ is known in the clear and hence this term may be computed.
      iii. Compute $[\boldsymbol{\mu}_2^\ell]$ and $[\boldsymbol{\nu}_2^\ell]$ for $\ell \in [Q]$ where

$$\boldsymbol{\mu}_2^\ell = (\hat{\mathbf{u}}_{2,i}^\ell \| r_2^\ell \cdot \mathbf{t}_{2,i})_{i\in[\mathsf{N}]}$$

$$\boldsymbol{\nu}_2^\ell = (\hat{\mathbf{v}}_{2,i}^\ell \| r_2^\ell \cdot \mathbf{t}_{2,i})_{i\in[\mathsf{N}]}$$

   (c) We rely on the IPE challenge ciphertext to compute the remainder of the ciphertext. In more detail, $\mathcal{B}$ submits the challenge messages as $(\mathbf{u}_1^\rho \otimes \mathbf{u}_2^\ell)_{\ell\in[Q]}$ for $b = 0$ and $(\mathbf{v}_1^\rho \otimes \mathbf{v}_2^\ell)_{\ell\in[Q]}$ for $b = 1$. By the construction of IPE, if $b = 0$, it obtains for $j \in [\mathsf{N}]$:

$$[\ r_{12}^{\rho,\ell}\ ],\ \left\{\ \left[\ \mathbf{u}_1^\rho[1]\ \mathbf{u}_2^\ell[j] + r_{12}^{\rho,\ell} \cdot \mathbf{S}[1,j] \,\|\ldots,\|\, \mathbf{u}_1^\rho[\mathsf{N}]\ \mathbf{u}_2^\ell[j] + r_{12}^{\rho,\ell} \cdot \mathbf{S}[\mathsf{N},j]\ \right]\ \right\} \tag{4.6}$$

   and if $b = 1$, it obtains for $j \in [\mathsf{N}]$:

$$[\ r_{12}^{\rho,\ell}\ ],\ \left\{\ \left[\ \mathbf{v}_1^\rho[1]\ \mathbf{v}_2^\ell[j] + r_{12}^{\rho,\ell} \cdot \mathbf{S}[1,j] \,\|\ldots,\|\, \mathbf{v}_1^\rho[\mathsf{N}]\ \mathbf{v}_2^\ell[j] + r_{12}^{\rho,\ell} \cdot \mathbf{S}[\mathsf{N},j]\ \right]\ \right\} \tag{4.7}$$

Next, we rewrite Equations 4.6 and 4.7 by making the following substitutions:

(a) First, note that since the IPE MSK $\mathbf{S} = \mathbf{S}_1 \cdot \mathbf{S}_2$, we have that $\mathbf{S}[i,j] = \langle \mathbf{s}_{1,i},\ \mathbf{s}_{2,j} \rangle$. By construction, $\mathbf{T}_2 = \mathbf{T}_1^{-1} \cdot \mathbf{S}$, so we have that $\mathbf{S}[i,j] = \langle \mathbf{t}_{1,i},\ \mathbf{t}_{2,j} \rangle$.

(b) Next, we use the encoding described in Step 1 to write:

$$\mathbf{u}_1^\rho[1]\ \mathbf{u}_2^\ell[j] = \langle \hat{\mathbf{u}}_{1,1}^\rho\ \hat{\mathbf{u}}_{2,j}^\ell \rangle$$

Thus, if $b = 0$, it obtains for $j \in [\mathsf{N}]$:

$$[\ r_{12}^{\rho,\ell}\ ],\ \left\{\ \left[\ \langle \hat{\mathbf{u}}_{1,1}^\rho\ \hat{\mathbf{u}}_{2,j}^\ell \rangle + r_{12}^{\rho,\ell} \cdot \langle \mathbf{t}_{1,1},\ \mathbf{t}_{2,j} \rangle \,\|\ldots,\|\, \langle \hat{\mathbf{u}}_{1,\mathsf{N}}^\rho\ \hat{\mathbf{u}}_{2,j}^\ell \rangle + r_{12}^{\rho,\ell} \cdot \langle \mathbf{t}_{1,\mathsf{N}},\ \mathbf{t}_{2,j} \rangle\ \right]\ \right\} \tag{4.8}$$

and if $b = 1$, it obtains for $j \in [\mathsf{N}]$:

$$[\ r_{12}^{\rho,\ell}\ ],\ \left\{\ \left[\ \langle \hat{\mathbf{v}}_{1,1}^\rho\ \hat{\mathbf{v}}_{2,j}^\ell \rangle + r_{12}^{\rho,\ell} \cdot \langle \mathbf{t}_{1,1},\ \mathbf{t}_{2,j} \rangle \,\|\ldots\|\, \langle \hat{\mathbf{v}}_{1,\mathsf{N}}^\rho\ \hat{\mathbf{v}}_{2,j}^\ell \rangle + r_{12}^{\rho,\ell} \cdot \langle \mathbf{t}_{1,\mathsf{N}},\ \mathbf{t}_{2,j} \rangle\ \right]\ \right\} \tag{4.9}$$

We claim that if $b = 0$ then the adversary sees the distribution of Hybrid $\mathsf{H}_0^\rho[1]$ while if $b = 1$, it sees the distribution of Hybrid $\mathsf{H}_0^\rho[2]$.

By equation 4.5, we are required to construct the ciphertext as the following, for $j \in [\mathsf{N}]$, $\ell \in [Q]$, $b = 0$ (the case of $b = 1$ simply substitutes $\mathbf{u}$ by $\mathbf{v}$):

$$\begin{aligned}
[r_2^\ell \| r_{12}^{\rho,\ell}], \Big[\boldsymbol{\mu}_{2,j}^\ell \,\|\, \boldsymbol{\nu}_{2,j}^\ell \,\|\, \langle \hat{\mathbf{u}}_{1,1}^\rho\ \hat{\mathbf{u}}_{2,j}^\ell \rangle + r_{12}^{\rho,\ell} \cdot \langle \mathbf{t}_{1,1},\ \mathbf{t}_{2,j} \rangle \| \\
\ldots \| \langle \hat{\mathbf{u}}_{1,\mathsf{N}}^\rho\ \hat{\mathbf{u}}_{2,j}^\ell \rangle + r_{12}^{\rho,\ell} \cdot \langle \mathbf{t}_{1,\mathsf{N}},\ \mathbf{t}_{2,j} \rangle \Big]
\end{aligned} \tag{4.10}$$

Note that, in step 3 defined above, we showed how to construct, for $\ell \in [Q]$:

$$[r_2^\ell], \quad [\boldsymbol{\mu}_2^\ell], \quad [\boldsymbol{\nu}_2^\ell]$$

using master secret as $\mathbf{T}_1$ and $\mathbf{T}_2$. The reduction $\mathcal{B}$ also constructs the terms described in equation 4.8 if $b = 0$ and equation 4.9 if $b = 1$. Thus, the reduction computes all the components required by Equation 4.10 correctly.

□

□

5 Decentralized Attribute-Based Inner Product FE

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 (IPE), which is suggested by casting both existing primitives as special cases of our model.

As in Section 4, while our general definition (Section 3) is clearly powerful enough to capture this simple functionality, we provide the restricted definition again here for ease of exposition and verification.

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].

5.1 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 $\lambda, n, \ell \in \mathbb{N}$, $C \in \mathcal{C}_n$, $\mathbf{u}, \mathbf{v} \in \mathcal{U}_\ell$, $\mathbf{x} \in \mathcal{X}_n$, 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.[5]

Definition 5.1 (Selective decentralized AB-IPFE security in presence of corrupt authorities)**.** A decentralized AB-IPFE scheme is selectively 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

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

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 $C^*(\mathbf{x}) = 0$, or (2) $\langle \mathbf{u}, \mathbf{v}_0^*\rangle = \langle \mathbf{u}, \mathbf{v}_1^*\rangle$, or (3) adversary does not query at least one non-corrupt authority for the same global identifier, attribute-key vector tuple $(\mathsf{GID}, \mathbf{x}, \mathbf{u})$.

5.2 Construction

Let Gen be a bilinear group generator with assymetric 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

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

[5] 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].

$\mathsf{KeyGen}(\{\mathsf{PK}_i\}_{i\in[n]}, \mathsf{MSK}_j, \mathsf{GID}, \mathbf{x}, \mathbf{u}) \to \mathsf{SK}_{j,\mathsf{GID},\mathbf{x},\mathbf{u}}$. 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

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

$\mathsf{Enc}(\{\mathsf{PK}_i\}_{i\in[n]}, \mathbf{y}, \mathbf{v}) \to \mathsf{CT}$. 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

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

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

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

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.

5.3 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 $\mathsf{SK}_{i,\mathsf{GID},\mathbf{x},\mathbf{u}} = ([\boldsymbol{\Delta}_i\mathbf{u} - x_i\mathbf{W}_i\mathbf{h} + \boldsymbol{\mu}_i]_2, \mathbf{x}, \mathbf{u})$ 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:

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

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

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

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

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

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

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

Therefore, whenever $\langle \mathbf{x}, \mathbf{y} \rangle = \mathbf{x}^\top \mathbf{y} = 0$, the above gets simplified to $[\mathbf{u}^\top \mathbf{v}]_T$ 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 5.2. If the k-linear assumption (assumption 2.12) 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 for inner product testing predicates as per Definition 5.1 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). 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[6], 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 $[\mathbf{h}]_2$ 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 $\langle \mathbf{x}, \mathbf{y}^* \rangle = 0$. 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 $\langle \mathbf{x}, \mathbf{y}^* \rangle = 0$ 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.

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

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

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

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

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 $[\mathbf{h}]_2 = [\mathbf{Br}]_2$.

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 $\langle \mathbf{x}, \mathbf{y}^* \rangle = 0$ or not. If it is equal to 0, then it computes the key K honestly as

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

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

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

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

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

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

except if the predicate is satisfied (that is, $\langle \mathbf{x}, \mathbf{y}^* \rangle = 0$), then it computes h as in Game 2, i.e. by picking a random r and computing $[\mathbf{h}]_2 = [\mathbf{Br}]_2$; 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 $[\mathbf{h}]_2 = [\mathbf{Br} + \mathbf{a}^\perp \widehat{r}]_2$.

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

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

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

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 $\langle \mathbf{x}, \mathbf{y}^* \rangle = 0$ or not. If it is equal to 0, then it computes the key K honestly as

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

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

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

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

Game $3.q.3$**.** This is same as the previous game, except when it answers the q-th query, then it computes h as $[\mathbf{h}]_2 = [\mathbf{Br}]_2$ 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:

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

where β is the random challenge bit.

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 5.3. For any (potentially unbounded) 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, for any invertible matrix $\mathbf{R} \in \mathbb{Z}_p^{\ell \times \ell}$, the following distributions are identical.

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

□

Lemma 5.4. If k-linear assumption (assumption 2.12) holds in $\mathbb{G}_2$ 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 identical to that of [MJ18, Lemma 1]. □

Lemma 5.5. If k-linear assumption (assumption 2.12) 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 $\mathsf{Adv}_{3.(q-1).3}^{\mathcal{A}}(\lambda) - \mathsf{Adv}_{3.q.1}^{\mathcal{A}}(\lambda) \leq \mathsf{negl}(\lambda)$ for some negligible function $\mathsf{negl}(\cdot)$.

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

Lemma 5.6. For any (potentially unbounded) adversary $\mathcal{A}$ and $q \in \{1, \ldots, Q\}$, we have that $\mathsf{Adv}_{3.q.1}^{\mathcal{A}}(\lambda) - \mathsf{Adv}_{3.q.2}^{\mathcal{A}}(\lambda) \leq \mathsf{negl}(\lambda)$ for some negligible function $\mathsf{negl}(\cdot)$.

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

Lemma 5.7. If k-linear assumption (assumption 2.12) 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 $\mathsf{Adv}_{3.q.2}^{\mathcal{A}}(\lambda) - \mathsf{Adv}_{3.q.3}^{\mathcal{A}}(\lambda) \leq \mathsf{negl}(\lambda)$ for some negligible function $\mathsf{negl}(\cdot)$.

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

Lemma 5.8. If k-linear assumption (assumption 2.12) holds in $\mathbb{G}_1$ over the group generator Gen, then for any PPT adversary $\mathcal{A}$, we have that $\mathsf{Adv}_{3.Q.3}^{\mathcal{A}}(\lambda) - \mathsf{Adv}_4^{\mathcal{A}}(\lambda) \leq \mathsf{negl}(\lambda)$ for some negligible function $\mathsf{negl}(\cdot)$.

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

Lemma 5.9. For any (potentially unbounded) adversary $\mathcal{A}$, we have that $\mathsf{Adv}_4^{\mathcal{A}}(\lambda) \leq \mathsf{negl}(\lambda)$ 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:

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

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

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

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 $\langle \mathbf{x}, \mathbf{y}^* \rangle = 0$, 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 $\langle \mathbf{x}, \mathbf{y}^* \rangle \neq 0$, 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. □

□

6 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]. We show that, by incurring a linear overhead in the parameter sizes, we could distribute the ABE construction by obtaining a non-interactive distributed setup and key generation such that as long as at least one authority is still honest, the scheme remains secure.

6.1 Defining distributed CP-ABE

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 6.1 (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 6.2 (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.

6.2 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 assymetric 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 Sections 2.5 and 2.7.

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 2.6. 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
$$\phi_0 = 1 \in \mathbf{Z}_q, \quad \phi_1 = 1 \in \mathbf{Z}_q, \quad \phi_2 := \mathbf{1} \in \mathbf{Z}_q^m,$$
$$\phi_{i,b} := \begin{cases} \mathbf{1} \in \mathbf{Z}_q^m & \text{if } b = x_i \\ \mathbf{0} \in \mathbf{Z}_q^m & \text{if } b \neq x_i \end{cases} \quad \text{for } i \in [\ell] \text{ and } b \in \{0,1\}. \tag{6.1}$$
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 $\mathbf{B} = (\mathbf{B}_1, \dots, \mathbf{B}_\ell) \leftarrow (\mathbf{Z}_q^{n\times m})^\ell$ 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
$$\psi_0 := 1 \in \mathbf{Z}_q, \quad \psi_1 := \mathbf{s}^\top \mathbf{u} + e_1 + \mu\lceil q/2 \rceil \in \mathbf{Z}_q,$$
$$\psi_2^\top := \mathbf{s}^\top \mathbf{A} + \mathbf{e}_2^\top \in \mathbf{Z}_q^m,$$
$$\psi_{i,b}^\top := \mathbf{s}^\top (\mathbf{B}_i - b\mathbf{G}) + \mathbf{e}_{i,b}^\top \in \mathbf{Z}_q^m \quad \text{for } i \in [\ell] \text{ and } b \in \{0,1\}. \tag{6.2}$$

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 6.3. 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 6.1) 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 6.1. 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 $[\delta_{\mathsf{GID},\mathbf{x}}]_1$. (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 6.4. 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 6.5. 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 6.6. 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 6.7. For any adversary $\mathcal{A}$, we have that $\mathsf{Adv}_3^{\mathcal{A}}(\lambda) - \mathsf{Adv}_4^{\mathcal{A}}(\lambda) \leq Q_{\mathtt{zt}}(N \cdot L + 3)^2/q$.

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 6.8. 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).[7] Therefore, the lemma follows by combining these. □

Lemma 6.9. 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 6.10. 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 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

$$f(\{W_i^{(j)}\}_{j \notin S^*, i \in [L]}, \{\Delta_{\mathsf{GID},\mathbf{x}}\}_{\mathsf{GID},\mathbf{x}}, \Gamma, \{\widetilde{C}_i^{(j)}\}_{j \in [N], i \in [L]}) = \sum_{Z \in S_{T,2}} a_Z Z \tag{6.3}$$

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

$$f(\{W_i^{(j)}\}_{j \notin S^*, i \in [L]}, \{\Delta_{\mathsf{GID},\mathbf{x}}\}_{\mathsf{GID},\mathbf{x}}, \Gamma, \{\widetilde{c}_i^{(j)}\}_{j \in [N], i \in [L]}) = 0. \tag{6.4}$$

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

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

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 6.11. 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 6.12. 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 6.1 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 simulateÂăhalf 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.

□

7 Conclusions and Open Problems

In this section, we define new primitives that can be seen as special cases of multi-party functional encryption and have natural, compelling applications. As discussed in 1, we may generalize MIFE, MCFE and PHFE so as to enforce a more complex policy on the public labels of multiple parties' ciphertexts, before the private inputs are combined. We note that in some scenarios, it will make sense to separate the computation on the public labels and the private inputs, that is, given encryptions of $(\mathsf{lab}_i, \mathbf{x}_i)$ for $i \in [n]$ and a function key for $f = (f_1, f_2)$, if $f_1(\mathsf{lab}_1, \ldots, \mathsf{lab}_n) = 1$ then output $f_2(\mathbf{x}_1, \ldots, \mathbf{x}_n)$. We may permit more meaningful ways of combining keys than those that have been studied before. Some special cases are also interesting in the single

input setting, which have not been considered. For instance, single input, ciphertext policy PHFE has not been considered before, to the best of our knowledge. Another meaningful primitive to consider may be to have $\mathsf{CT}(\mathbf{x}, f_1)$ and $\mathsf{SK}(f_2, \mathbf{y})$ be combined to compute $f_1(\mathbf{y})$ if $f_2(\mathbf{x}) = 1$ (and vice versa).

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.

[ABKW19] Michel Abdalla, Fabrice Benhamouda, Markulf Kohlweiss, and Hendrik Waldner. Decentralizing inner-product functional encryption. In *IACR International Workshop on Public Key Cryptography*, pages 128–157. Springer, 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. *IACR Cryptol. ePrint Arch.*, 2020.

[AGW20] Michel Abdalla, Junqing Gong, and Hoeteck Wee. Functional encryption for attribute-weighted sums from k-lin. In *Annual International Cryptology Conference*, 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 *International Conference on the Theory and Application of Cryptology and Information Security*, 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.

[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.

[CSG+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.

[CSG+18b] JÃľrÃľmy 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.

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

[CZY19] Yuechen Chen, Linru Zhang, and Siu-Ming Yiu. Practical attribute based inner product functional encryption from simple assumptions. Cryptology ePrint Archive, Report 2019/846, 2019.

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

[GGG+14a] 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.

[GGG+14b] 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.

[Lin17] Huijia Lin. Indistinguishability obfuscation from sxdh on 5-linear maps and locality-5 prgs. In *Crypto*, 2017.

[LŢ19] Benoît Libert and Radu Ţiţiu. Multi-client functional encryption for linear functions in the standard model from lwe. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 520–551. Springer, 2019.

[LV16] Huijia Lin and Vinod Vaikuntanathan. Indistinguishability obfuscation from ddh-like assumptions on constant-degree graded encodings. In *FOCS*, 2016.

[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*, pages 256–266. Springer-Verlag, 1997.

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

[Tsa19] Rotem Tsabary. Fully secure attribute-based encryption for t-cnf from LWE. In *CRYPTO*, 2019.

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

[Wee17] Hoeteck Wee. Attribute-Hiding Predicate Encryption in Bilinear Groups, Revisited. In *TCC*, 2017.

Appendix

A Dynamic Multi-Party Functional Encryption

In this section, we define the dynamic notion of multi-party functional encryption (MPFE). As mentioned in Remark 3.1, 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. 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 pair of encryption key EK and a master secret key MSK.

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

$\mathsf{Encrypt}\left(\mathsf{EK}, i, x = (x_{\text{pub}}, x_{\text{pri}}), \mathsf{Agg}_x\right)$: 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 $\mathsf{Agg}_x : \mathcal{X}^{n_x} \to \mathcal{X}^*$ (for some $n_x \in \mathbb{N}$), this algorithm outputs a ciphertext $\mathsf{CT}_{i,x}$.

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

Correctness. We say that an MPFE scheme is *correct* if, $\forall (N, n) \in \mathbb{N}^2$, ciphertext inputs $x_i \in \mathcal{X}$ for $i \in [n]$, key inputs $y_j \in \mathcal{Y}$ for $j \in [n]$, message and function aggregation circuits Agg_x and Agg_y, and indexing function $\text{index} : [n] \to [N]$, it holds that:

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

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

A.1 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[8]:

$\mathsf{HonestGen}()$, upon a call to this oracle, the challenger samples a fresh pair of encryption and master key $(\mathsf{EK}, \mathsf{MSK})$, and stores them alongside the query number i in a list $Ł_{\mathsf{setup}}$. It does not send any key to the adversary. (Note that the adversary can record the query number itself too, thus challenger does not need to send it. Note that if the scheme is a public key scheme, then the challenger sends the encryption key EK to the adversary.)

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

$\mathsf{Corrupt}(\cdot)$, upon a call to this oracle for an honest user index i, the challenger first checks whether the list $Ł_{\mathsf{setup}}$ contains a key pair associated with index i. If there is such a key pair $(\mathsf{EK}_i, \mathsf{MSK}_i)$, then it removes this entry from the list and sends it to the adversary. Otherwise, it sends nothing.

$\mathsf{Key}^{\beta}(\cdot,\cdot,\cdot,\cdot)$, upon a call to this oracle for an honest user index i, function inputs $(y_j^{k,0}, y_j^{k,1})$ (where $y_j^{k,b} = \left(\left(y_{j,\text{pub}}^{k,b}, y_{j,\text{pri}}^{k,b}\right)\right)$ 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 index i. If there is such a key pair $(\mathsf{EK}_i, \mathsf{MSK}_i)$, then it samples a decryption key for function input $y_j^{k,\beta}$ using key MSK_i 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 index i, inputs $(x_j^{\ell,0}, x_j^{\ell,1})$ (where $x_j^{\ell,b} = \left(\left(x_{j,\text{pub}}^{\ell,b}, x_{j,\text{pri}}^{\ell,b}\right)\right)$ 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 index i. If there is such a key pair $(\mathsf{EK}_i, \mathsf{MSK}_i)$, then it samples a ciphertext for input $x_j^{\ell,\beta}$ using key EK_i 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 $\mathcal{Q}_x = \{(i, (x_j^{\ell,0}, x_j^{\ell,1}), j, \mathsf{Agg}_{x,j}^{\ell})\}_{\ell \in [Q_x]}$ be the set of ciphertext challenge queries and $\mathcal{Q}_y = \{(i, (y_j^{k,0}, y_j^{k,1}), j, \mathsf{Agg}_{y,j}^{k})\}_{k \in [Q_y]}$ be the set of key challenge queries.

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

1. For each of the encryption and key challenges, the public components of the two challenges are equal, namely $x_{j,\text{pub}}^{\ell,0} = x_{j,\text{pub}}^{\ell,1}$ for all $\ell \in [Q_x]$, and $y_{j,\text{pub}}^{k,0} = y_{j,\text{pub}}^{k,1}$ for all $k \in [Q_y]$.
2. For each of the encryption and key challenges, the *private* components of the two challenges are also equal, namely $x_{j,\text{pri}}^{\ell,0} = x_{j,\text{pri}}^{\ell,1}$ for all $\ell \in [Q_x]$, and $y_{j,\text{pri}}^{k,0} = y_{j,\text{pri}}^{k,1}$ for all $k \in [Q_y]$ if the user index i, that the query was made for, was corrupted during the execution.
3. There do not exist two sequences $(\overrightarrow{x}^0, \overrightarrow{y}^0) \neq (\overrightarrow{x}^1, \overrightarrow{y}^1)$ and aggregation functions $\mathsf{Agg}_x, \mathsf{Agg}_y$ such that:
$$\mathcal{U}\left(\mathsf{Agg}_x(\{x_j^0\}), \mathsf{Agg}_y(\{y_j^0\})\right) \neq \mathcal{U}\left(\mathsf{Agg}_x(\{x_j^1\}), \mathsf{Agg}_y(\{y_j^1\})\right)$$
and i) x_j^b was queried for aggregation function Agg_x and index j, and ii) y_j^b was queried for aggregation function Agg_y and index j, and iii) at least one of inputs $\left\{x_j^b\right\}, \left\{y_j^b\right\}$ were queried and index j was not corrupted. Note that if some x_j^b or y_j^b 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.

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 A.1 (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.

Capturing Dynamic Decentralized Functional Encryption (DDFE): A dynamic decentralized FE scheme, denoted by DDFE [CSG+20], can be viewed as a special case of MPFE with a local setup in the dynamic setting. In an DDFE scheme for functionality F, the setup algorithm 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 CT. The key generator chooses a key space object k and computes a corresponding decryption key dk_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 $(\mathsf{dk}_{k_i})_i$ and ciphertexts $(\mathsf{CT}_i)_i$, and outputs $F((k_i)_i, (m_i)_i)$.

In our notation, DDFE is captured by setting $y_i = ((F_i, k_i), \perp)$, $x_i = (\perp, m_i)$. The function Agg_x outputs $(m_1, \ldots, m_n)$, and Agg_y checks that $F_1 = F_2 \ldots = F_n$ is the same in all y_i and outputs $(F, k_1, \ldots, k_n)$ if so ($\perp$ otherwise). Upon decryption this yields $\mathcal{U}\big((m_1, \ldots, m_n), (F, (k_1, \ldots, k_n))\big) = F\big((k_i)_i, (m_i)_i\big)$ as desired.

B Feasibility for MPFE for General Circuits

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 $\left(\mathsf{MIFE.MSK}, \mathsf{MIFE.EK}_1, \ldots, \mathsf{MIFE.EK}_{\mathsf{n_x+n_y}}\right) \leftarrow \mathsf{MIFE.Setup}(1^\lambda, 1^{n_x+n_y})$.
   (b) Using an N out of N secret sharing scheme, compute shares $\mathsf{MIFE.MSK}_i$ of the $\mathsf{MIFE.MSK}$ 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 $\mathsf{MSK}_i = (\mathsf{MIFE.EK}_{\mathsf{n_x+i}}, \mathsf{MIFE.MSK_i}, \mathfrak{s}_\mathsf{i})$ 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 $\mathsf{MIFE.MSK}$ 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[9].

4. To decrypt, the decryptor computes $\mathsf{MPC.ComputeResult}$ 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.

[9] 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.

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 $\mathsf{MIFE.SK}$ 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.