Decentralizing Inner-Product Functional Encryption

Michel Abdalla[1,2], Fabrice Benhamouda[3],
Markulf Kohlweiss[4], and Hendrik Waldner[4]

[1] DIENS, École normale supérieure, CNRS, PSL University, Paris, France
michel.abdalla@ens.fr
[2] INRIA, Paris, France
[3] IBM Research, Yorktown Heights, NY, US
fabrice.benhamouda@normalesup.org
[4] University of Edinburgh, Edinburgh, UK
{mkohlwei,hendrik.waldner}@ed.ac.uk

Abstract. Multi-client functional encryption (MCFE) is a more flexible variant of functional encryption whose functional decryption involves multiple ciphertexts from different parties. Each party holds a different secret key and can independently and adaptively be corrupted by the adversary. We present two compilers for MCFE schemes for the inner-product functionality, both of which support encryption labels. Our first compiler transforms any scheme with a special key-derivation property into a decentralized scheme, as defined by Chotard et al. (ASIACRYPT 2018), thus allowing for a simple distributed way of generating functional decryption keys without a trusted party. Our second compiler allows to lift an unnatural restriction present in existing (decentralized) MCFE schemes, which requires the adversary to ask for a ciphertext from each party. We apply our compilers to the works of Abdalla et al. (CRYPTO 2018) and Chotard et al. (ASIACRYPT 2018) to obtain schemes with hitherto unachieved properties. From Abdalla et al., we obtain instantiations of DMCFE schemes in the standard model (from DDH, Paillier, or LWE) but without labels. From Chotard et al., we obtain a DMCFE scheme with labels still in the random oracle model, but without pairings.

1 Introduction

Functional encryption (FE) [BSW11, O'N10, SW05] is a form of encryption that allows fine-grained access control over encrypted data. Besides the classical encryption and decryption procedures, functional encryption schemes consists of a key derivation algorithm, which allows the owner of a master secret key to derive keys with more restricted capabilities. These derived keys sk_f are called functional decryption keys and are associated with a function f. Using the key sk_f for the decryption of a ciphertext $\mathsf{Enc}(x)$ generates the output $f(x)$. During this decryption procedure no more information is revealed about the underlying plaintext than $f(x)$.

In the case of classical functional encryption, the (functional) decryption procedure takes as input a single ciphertext $\mathsf{Enc}(x)$. A natural extension is the multi-input setting, where the decryption procedure takes as input n different ciphertexts and outputs a function applied on the n corresponding plaintexts. Such a scheme is called multi-input functional encryption (MIFE) scheme [GGG+14]. In a MIFE scheme, each ciphertext can be generated independently (i.e., with completely independent randomness).

An important use case of MIFE considers multiple parties or clients, where each party P_i generates a single ciphertext of the tuple. The ciphertext generated by party P_i is often said to correspond to *position* or *slot* i. In the multi-client setting, it becomes natural to assume that each party has a different secret/encryption key sk_i that can be corrupted by the adversary. We call such a scheme a multi-client functional encryption (MCFE) scheme [CDG+18a, GGG+14].

We remark that the exact terminology varies from paper to paper. Here, a MCFE scheme is always supposed to be secure against corruption of the parties encrypting messages. In a MIFE scheme, on the other hand, all the parties may use the same encryption key and there is no security against corruption.

The multi-input and multi-client settings still require a trusted third party that sets up the encryption keys and holds the master secret key used to derive the functional decryption keys. As a result, the central authority is able to recover every client's private data. This raises the question if it is possible to decentralize the concept of functional encryption and get rid of this trusted entity. In this work, we focus on the notion of decentralized multi-client functional encryption (DMCFE) introduced by Chotard et al. [CDG+18a]. In DMCFE, the key derivation procedure KeyDer is divided into two procedures $\mathsf{KeyDerShare}$ and $\mathsf{KeyDerComb}$. The KeyDer procedure allows each party P_i to generate a share $\mathsf{sk}_{i,f}$ of the functional key sk_f from its secret key sk_i. The $\mathsf{KeyDerComb}$ procedure is then used to combine these n different shares $\mathsf{sk}_{1,f}, \ldots, \mathsf{sk}_{n,f}$ to generate the functional key sk_f. Assuming that the secret key sk_i can also be generated in a distributed way, this makes it possible to get rid of the trusted party and to ensure that every party has complete control over their individual data.

An important property of MIFE and (D)MCFE schemes is whether they are labeled or not. The labeled setting is similar to vanilla multi-input/multi-client functional encryption, but the encryption procedure takes as input a second parameter, a so-called label ℓ. The decryption procedure is restricted in such a way that it is only possible to decrypt ciphertexts that are encrypted under the same label $\mathsf{Enc}(\mathsf{sk}_1, x_1, \ell), \ldots, \mathsf{Enc}(\mathsf{sk}_n, x_n, \ell)$. This setting is sometimes desirable in practice as it allows repeated computations over encrypted data that comes from different sources (for example data mining over encrypted data or multi-client delegation of computation [CDG+18a]).

In the last few years, many multi-input or multi-client functional encryption schemes have been constructed. As noted in [AGRW17], these schemes can be split into two main categories: (1) feasibility for general functionalities, and (2) concrete and efficient realizations for more restricted functionalities. Constructions of the first category [GGG+14, BGJS15, AJ15, BKS18] are based on more unstable assumptions, such as indistinguishable obfuscation or multilinear maps, and tackle the problem of creating schemes for more general functionalities. A few constructions of the second category are provided in the work of Abdalla et al. in [AGRW17, ACF+18] and Chotard et al. in [CDG+18a], which consider different types of secret-key constructions for the inner-product functionality. In these schemes, each function is specified by a collection $\boldsymbol{y}$ of n vectors $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_n$ and takes a collection $\boldsymbol{x}$ of n vectors $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$ as input. Their output is $f_{\boldsymbol{y}}(\boldsymbol{x}) = \sum_{i=1}^{n} \langle \boldsymbol{x}_i, \boldsymbol{y}_i \rangle = \langle \boldsymbol{x}, \boldsymbol{y} \rangle$. As the original single-input inner-product functionality [ABDP15, BJK15, DDM16, ALS16] and their quadratic extensions [BCFG17], multi-input or multi-client

inner-product functionalities can be quite useful for computing statistics or performing data mining on encrypted databases [AGRW17, CDG+18a].

Currently, the work of Chotard et al. provides the only known DMCFE to our knowledge. However, while their MCFE uses any cyclic group where the Decisional Diffie-Hellman (DDH) assumption holds, their DMCFE scheme requires pairings. Furthermore, the security notion they achieve only guarantees security against an adversary that queries the encryption/challenge oracle for every position i. At first glance, it might seem that more encryption queries would help the adversary, but this does not allow trivial attacks and the adversary is restricted as follows: all functions f for which the adversary has a functional decryption key must evaluate to the same value on all the plaintext tuples queried to the encryption oracle. However, when a position i is not queried, this condition is always satisfied since the function f in principle can never be really evaluated due to a missing input. Hence, requiring the adversary to query the encryption/challenge oracle for every position i actually weakens the achieved security notion.

This leaves open the following problems that we tackle in this paper:

1. Constructing DMCFE schemes without pairings, and even from more general assumptions than discrete-logarithm-based ones.
2. Removing the restriction that the adversary has to query the encryption oracle for every position i.

1.1 Contributions

Our first main contribution is to provide a generic compiler from any MCFE scheme satisfying an extra property called *special key derivation* into an DMCFE scheme. The transformation is purely information-theoretic and does not require any additional assumptions. As the MCFE from Chotard et al. [CDG+18a] satisfies this extra property, we obtain a *labeled* DMCFE scheme secure under the plain DDH assumption without pairings (in the random oracle model). As in [CDG+18a], the version of the scheme without labels is secure in the standard model (i.e., without random oracles).

Furthermore, we show as an additional contribution that the MIFE schemes from Abdalla et al. [ACF+18] are actually MCFE secure against adaptive corruptions (but without labels). This directly yields the first DMCFE scheme without labels from the LWE assumptions and the Paillier assumptions in the standard model.

Our second main contribution is to provide generic compilers to transform any scheme in the weaker model where the adversary is required to query the encryption oracle at every position i, into a scheme without this restriction. We propose two versions of the compiler: one without labels (in the standard model) which only requires an IND-CPA symmetric encryption scheme, and one with labels in the random oracle model.

These two compilers can be used to lift the security of the previously mentioned constructions of DMCFE to the stronger model. The resulting instantiations from DDH, LWE, and Paillier described above rely on the same assumptions.

1.2 Technical Overview

Contribution 1: A MCFE to DMCFE compiler. The DMCFE construction introduced by Chotard et al. [CDG+18a] is based on pairings and proven in the random oracle model.

Our first compiler transforms an MCFE scheme into a DMCFE scheme and does not require pairings. It operates on schemes with the special key derivation property, namely whose master secret key can be split into separate secret keys, one for each input, i.e. $\mathsf{msk} = \{\mathsf{sk}_i\}_{i\in[n]}$, and whose functional decryption keys are derived through a combination of local and linear inner-product computations on i, f, sk_i and pp. That is, the functional decryption key sk_f for the function $f : \boldsymbol{x} \mapsto \sum_{i=1}^{n}\langle \boldsymbol{x}_i, \boldsymbol{y}_i\rangle$ can be written as $\mathsf{sk}_f = (\{s(\mathsf{sk}_i, \boldsymbol{y})\}_{i\in[n]}, \sum_{i=1}^{n}\langle u(\mathsf{sk}_i), \boldsymbol{y}_i\rangle)$, where f is defined by the collection $\boldsymbol{y}$ of the vectors $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_n$ and

takes as input a collection $\boldsymbol{x}$ of n vectors $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$, and where s and u are two public functions.[5] Sums and inner products are computed modulo some integer L which is either prime or hard to factor.

For instance, the MCFE scheme of [CDG+18a], without pairings but supporting labels, has functional decryption keys of the form $\mathsf{sk}_f = (\boldsymbol{y}, \sum_i \mathsf{sk}_i \cdot y_i)$.

Consider first the following straw man compiler. It splits KeyDer into two procedures KeyDerShare and KeyDerComb. The first procedure assumes that each party P_i has access to the i-th share $t_{i,\boldsymbol{y}}$ of a fresh secret sharing of zero $\{t_{i,\boldsymbol{y}}\}_{i\in[n]}$. It then computes $s_{i,\boldsymbol{y}} = s(\mathsf{sk}_i, \boldsymbol{y})$ (which is a local computation) and $\mathsf{dk}_{i,\boldsymbol{y}} = \langle u(\mathsf{sk}_i), \boldsymbol{y}_i\rangle + t_{i,\boldsymbol{y}}$. The output key share is $\mathsf{sk}_{i,\boldsymbol{y}} = (s_{i,\boldsymbol{y}}, \mathsf{dk}_{i,\boldsymbol{y}})$. In the KeyDerComb procedure, the $\mathsf{dk}_{i,\boldsymbol{y}}$ values get summed up to cancel out the $t_{i,\boldsymbol{y}}$ values and to obtain $\sum_i \langle u(\mathsf{sk}_i), \boldsymbol{y}_i\rangle$. The output gets then extended with the values $s(\mathsf{sk}_i, \boldsymbol{y})$ to obtain the complete functional decryption key sk_f. This works but the question on the generation of the fresh secret sharing of zero $\{t_{i,\boldsymbol{y}}\}_{i\in[n]}$ is left out.

One solution consists in generating it as follows: $t_{i,\boldsymbol{y}} = \sum_{j\neq i}(-1)^{j<i} F_{\mathsf{K}_{i,j}}(\boldsymbol{y})$, where $F_{\mathsf{K}_{i,j}}$ is a pseudorandom function with key $\mathsf{K}_{i,j} = \mathsf{K}_{j,i}$ shared between parties P_i and P_j. This yields an DMCFE scheme secure against static corruption. Unfortunately, we do not know how to prove such a scheme secure against adaptive corruptions.

Our full compiler improves this construction in two ways: it allows adaptive corruptions and does not require any pseudorandom function. The procedure KeyDerShare of our full compiler uses masking values $\{\boldsymbol{v_i}\}_{i\in[n]}$, $\boldsymbol{v_i} \in \mathbb{Z}_L^{m\cdot n}$, such that $\boldsymbol{v}_n = -\sum_{i=1}^{n-1} \boldsymbol{v}_i$, to derive the key shares $\mathsf{sk}_{i,f} = \langle u(\mathsf{sk}_i), \boldsymbol{y}_i\rangle + \langle \boldsymbol{v}_i, \boldsymbol{y}\rangle$. Here, $\langle \boldsymbol{v}_i, \boldsymbol{y}\rangle$ acts as a kind of information theoretic pseudorandom function with key $\boldsymbol{v}_i$. To make this work, the queried values need to be linearly independent. This allows us to construct an information-theoretic compiler that provides security against adaptive corruptions (see Section 3 for details).

The masking of values prevents the combination of key shares for different functions $\boldsymbol{y}$. If one computes shares on different $\boldsymbol{y}$'s, then the sum of these shares will not sum up to 0 and the resulting key will be invalid. The encryption and decryption procedures proceed in the same way as in the MCFE setting.

Contribution 2: A compiler enforcing a single ciphertext query for each position. The standard security property of MIFE/MCFE schemes guarantees that an adversary can only learn a function of the inputs when it is in possession of a ciphertext for every input position i. This property is not satisfied by the schemes of [AGRW17, ACF+18, CDG+18a]. Their basic definitions guarantee security *only* when the adversary queries every position at least once. We call a scheme satisfying this property *pos-IND* secure (for positive) while we call the standard property *any-IND* secure.

To overcome this deficiency, Abdalla et al. [AGRW17] constructed a compiler that turns any *pos-IND* secure MIFE scheme into an *any-IND* secure MIFE scheme. The compiler uses a symmetric encryption scheme in addition to their MIFE encryption scheme. In more detail, the setup procedure of the compiler samples a key K for the symmetric encryption scheme and splits it into n shares $\mathsf{k}_1, \ldots, \mathsf{k}_n$, such that $\mathsf{k}_1 \oplus \cdots \oplus \mathsf{k}_n = \mathsf{K}$. Each party P_i receives its MIFE key sk_i, the symmetric encryption key K, as well as its share of the encryption key k_i. To encrypt, every party first runs the encryption procedure of the MIFE scheme to generate $\mathsf{ct}_i \leftarrow \mathsf{Enc}(\mathsf{sk}_i, \boldsymbol{x}_i)$ and then encrypts the output ct_i using the symmetric encryption scheme to get $\mathsf{ct}'_i \leftarrow \mathsf{Enc}_{\mathsf{SE}}(\mathsf{K}, \mathsf{ct}_i)$. The output of the encryption procedure is $(\mathsf{ct}'_i, \mathsf{k}_i)$. This compiler obviously does not work when we allow corruptions as this would allow the adversary to learn K after corrupting any single party and use it to recover ct_i from ct'_i for all positions i. Consequently, the compiler does not work for (D)MCFE schemes.

In this work, we construct an extension of the compiler described above that works in the multi-client setting by individually having a separate symmetric encryption key for each position. Hence, we increase the number of symmetric encryption keys from 1 to n and the number of the corresponding shares from n to n^2. This allows us to ensure that if the adversary does not ask encryption queries in every uncorrupted position, it does not learn any information about the underlying (D)MCFE ciphertexts.

We describe in more detail how the compiler handles the additional keys. In the setup procedure, every party (or a trusted party) generates its own key K_i and corresponding shares $\mathsf{k}_{i,j}$, such that $\mathsf{k}_{i,1} \oplus \cdots \oplus \mathsf{k}_{i,n} =$

[5] We note that our compiler actually is not restricted to the inner-product functionality. The only requirement is the special key derivation property.

K_i. The share $\mathsf{k}_{i,j}$ gets exchanged with Party P_j afterwards. In the encryption procedure every party encrypts its plaintext in the same way as in the MIFE setting, but using its own key K_i instead of the single symmetric encryption key K. The ciphertext corresponding to slot i of Party P_i is $(\mathsf{ct}'_i, \{\mathsf{k}_{j,i}\}_{j\in[n]})$.

If the adversary does not know all of the shares $\{\mathsf{k}_{i,j}\}_{i,j\in[n]}$, then security relies on the security of the symmetric encryption scheme. If the adversary knows all of the different symmetric encryption keys $\mathsf{K}_i, i \in [n]$, it relies on the security of the multi-client scheme. All of the key shares are only released if an encryption query has been made in every uncorrupted position.

This first compiler is, however, restricted to (D)MCFE schemes without labels. To add support for labels, the first idea is to use fresh keys $\mathsf{k}_{i,j,\ell}$ for each label ℓ. These keys can be locally derived from $\mathsf{k}_{i,j}$ using a pseudorandom function. Unfortunately, we do not know how to prove the security of such a scheme except in a very restricted setting (selective and static corruptions), where the adversary needs to output all its corruption and encryption queries at the beginning of the security experiment. We show how to achieve the standard (adaptive) security notion when the above pseudorandom function as well as the symmetric encryption scheme is implemented using hash functions that can be modeled as random oracles. The use of random oracles allows us to show that the adversary learns absolutely nothing about the inner ciphertexts ct_i until it has queried all the positions (for each given label) and we can then program the random oracles to properly "explain" the previously generated ciphertexts ct'_i.

1.3 Additional Related Work

In [FT18], Fan and Tang proposed a new notion of distributed public key functional encryption, in which the key generation procedure generates n different shares $\{\mathsf{sk}_i^f\}_{i\in[n]}$ instead of a single functional decryption key sk_f. The decryption of a ciphertext ct (a encryption of a message m) under a function f requires first the decryption under the functional key shares $s_i \leftarrow \mathsf{Dec}(\mathsf{sk}_i^f, \mathsf{ct})$ for all $i \in [n]$. These shares $\{s_i\}_{i\in[n]}$ are then used to reconstruct $f(m)$. In this setting, a trusted third party is still needed to set up the public parameters and to generate the functional keys, which makes it not really decentralized.

In Private Stream Aggregation (PSA), a weighted sum $f(\boldsymbol{x}) \mapsto \sum_i^n x_i$ gets computed. This is similar to DMCFE for the inner-product functionality $f(\boldsymbol{x}) \mapsto \langle \boldsymbol{x}, \boldsymbol{y} \rangle$. PSA was introduced by Shi et al. [SCR+11] and allows a set of users to compute the sum of their encrypted data for different time periods. Compared to DMCFE, PSA is more restricted. It only allows computation of simple sums, whereas in principle DMCFE allows the computation of different functions on the input data. Furthermore, research on PSA has mainly focused on achieving new properties or better efficiency [BJL16, CSS12, Emu17, JL13, LC12], instead of providing new functionalities.

1.4 Concurrent Work

Concurrently and independently of our work, Chotard et al. [CDG+18b] proposed new constructions of MCFE schemes for inner products both in the centralized and decentralized settings. Their paper contains three main contributions: (1) A pairing-based compiler that turns any *pos-IND* secure MCFE scheme into an *any-IND* secure MCFE scheme, secure under the decisional Bilinear Diffie-Hellman problem in the random oracle model; (2) A second compiler that turns a *one-IND* secure MCFE scheme into a *pos-IND* secure MCFE scheme; and (3) A compiler that transforms a class of MCFE schemes for inner products into a corresponding DMCFE scheme, based on either the CDH assumption in the random-oracle model or the DDH assumption in the standard model.

While contribution (2) is unrelated and complementary to our work, contributions (1) and (3) are related to our contributions in Section 4 and Section 3, respectively. Regarding (1), their compiler from *pos-IND* to *any-IND* security produces constant-size ciphertexts, but it requires pairings and random oracles. Our compiler in Section 4, on the other hand, avoids pairings, requiring either symmetric encryption when applied to schemes without labels or random oracles for schemes with labels, but ciphertext sizes are linear in the number of inputs. Regarding (3), their compiler is similar to the straw man compiler described above. It

is based on the DDH assumption and proven secure with respect to static corruptions. Our compiler in Section 3, on the other hand, is information-theoretic and achieves adaptive security.

1.5 Organization

The paper is organized as follows. In Section 2, we recall classical definitions as well as the definition of MCFE and DMCFE. Section 3 presents our first main contribution: the compiler from MCFE to DMCFE. Our second main contribution, namely our compilers from pos-IND security to any-IND security, is shown in Section 4. We conclude our paper by the proof that the MIFE scheme of Abdalla et al. [ACF+18] is actually an MCFE scheme that is secure under adaptive corruptions.

2 Definitions and Security Models

Notation. We use $[n]$ to denote the set $\{1, \ldots, n\}$. We write $\boldsymbol{x}$ for vectors and x_i for the i-th element. For security parameter λ and additional parameters n, we denote the winning probability of an adversary $\mathcal{A}$ in a game or experiment G as $\mathsf{Win}^{\mathsf{G}}_{\mathcal{A}}(\lambda, n)$, which is $\Pr[\mathsf{G}(\lambda, n, \mathcal{A}) = 1]$. The probability is taken over the random coins of G and $\mathcal{A}$. We define the distinguishing advantage between games G_0 and G_1 of an adversary $\mathcal{A}$ in the following way: $\mathsf{Adv}^{\mathsf{G}}_{\mathcal{A}}(\lambda, n) = \left|\mathsf{Win}^{\mathsf{G}_0}_{\mathcal{A}}(\lambda, n) - \mathsf{Win}^{\mathsf{G}_1}_{\mathcal{A}}(\lambda, n)\right|$.

2.1 Multi-Client Functional Encryption

In this section, we define the notion of MCFE [GGG+14].

Definition 2.1. ***(Multi-Client Functional Encryption)*** *Let $\mathcal{F} = \{\mathcal{F}_\rho\}_\rho$ be a family (indexed by ρ) of sets $\mathcal{F}_\rho$ of functions $f \colon \mathcal{X}_{\rho,1} \times \cdots \times \mathcal{X}_{\rho,n_\rho} \to \mathcal{Y}_\rho$.*[6] *Let $\mathsf{Labels} = \{0,1\}^*$ or $\{\bot\}$ be a set of labels. A multi-client functional encryption scheme (MCFE) for the function family $\mathcal{F}$ and the label set Labels is a tuple of five algorithms $\mathsf{MCFE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{KeyDer}, \mathsf{Enc}, \mathsf{Dec})$:*

$\mathsf{Setup}(1^\lambda, 1^n)$**:** *Takes as input a security parameter λ and the number of parties n, and generates public parameters pp. The public parameters implicitly define an index ρ corresponding to a set $\mathcal{F}_\rho$ of n-ary functions (i.e., $n = n_\rho$).*

$\mathsf{KeyGen}(\mathsf{pp})$**:** *Takes as input the public parameters pp and outputs n secret keys $\{\mathsf{sk}_i\}_{i\in[n]}$ and a master secret key msk.*

$\mathsf{KeyDer}(\mathsf{pp}, \mathsf{msk}, f)$**:** *Takes as input the public parameters pp, the master secret key msk and a function $f \in \mathcal{F}_\rho$, and outputs a functional decryption key sk_f.*

$\mathsf{Enc}(\mathsf{pp}, \mathsf{sk}_i, x_i, \ell)$**:** *Takes as input the public parameters pp, a secret key sk_i, a message $x_i \in \mathcal{X}_{\rho,i}$ to encrypt, a label $\ell \in \mathsf{Labels}$, and outputs ciphertext $\mathsf{ct}_{i,\ell}$.*

$\mathsf{Dec}(\mathsf{pp}, \mathsf{sk}_f, \mathsf{ct}_{1,\ell}, \ldots, \mathsf{ct}_{n,\ell})$**:** *Takes as input the public parameters pp, a functional key sk_f and n ciphertexts under the same label ℓ and outputs a value $y \in \mathcal{Y}_\rho$.*

A scheme MCFE is correct, if for all $\lambda, n \in \mathbb{N}$, $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, 1^n)$, $f \in \mathcal{F}_\rho$, $\ell \in \mathsf{Labels}$, $x_i \in \mathcal{X}_{\rho,i}$, when $(\{\mathsf{sk}_i\}_{i\in[n]}, \mathsf{msk}) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$ and $\mathsf{sk}_f \leftarrow \mathsf{KeyDer}(\mathsf{pp}, \mathsf{msk}, f)$, we have

$$\Pr\left[\mathsf{Dec}(\mathsf{pp}, \mathsf{sk}_f, \mathsf{Enc}(\mathsf{pp}, \mathsf{sk}_1, x_1, \ell), \ldots, \mathsf{Enc}(\mathsf{pp}, \mathsf{sk}_n, x_n, \ell)) = f(x_1, \ldots, x_n)\right] = 1 \ .$$

When ρ is clear from context, the index ρ is omitted. When $\mathsf{Labels} = \{0,1\}^*$, we say that the scheme is *labeled* or *with labels*. When $\mathsf{Labels} = \{\bot\}$, we say that the scheme is *without labels*, and we often omit ℓ.

[6] All the functions inside the same set $\mathcal{F}_\rho$ have the same domain and the same range.

Remark 2.2. We note that contrary to most definitions, the algorithm Setup only generates public parameters that determine the set of functions for which functional decryption keys can be created. The secret/encryption keys and the master secret keys are generated by another algorithm KeyGen, while the functional decryption keys are generated by KeyDer. This separation between Setup and KeyGen is especially useful when combining multiple MCFE/MIFE schemes as in [ACF+18] to ensure that all the MCFE/MIFE instances are using the same modulus. Note that this separation prevents for example the functionality to consist of inner products modulo some RSA modulus $N = pq$ and the master secret key to contain the factorization of N (except if the factorization of the modulus N is public).

As noted in [CDG+18a, GGG+14], the security model of multi-client functional encryption is similar to the security model of standard multi-input functional encryption, except that instead of a single master secret key msk for encryption, each slot i has a different secret key sk_i and the keys sk_i can be individually corrupted. In addition, one also needs to consider corruptions to handle possible collusions between different parties. In the following, we define security as adaptive left-or-right indistinguishability under both static (sta), and adaptive (adt) corruption. We also consider three variants of these notions $(\text{one}, \text{any}, \text{pos})$ related to the number of encryption queries asked by the adversary for each slot.

Definition 2.3. ***(Security of MCFE)*** *Let* MCFE *be an MCFE scheme,* $\mathcal{F} = \{\mathcal{F}_\rho\}_\rho$ *a function family indexed by* ρ *and* Labels *a label set. For* $\text{xx} \in \{\text{sta}, \text{adt}\}$, $\text{yy} \in \{\text{one}, \text{any}, \text{pos}\}$, *and* $\beta \in \{0,1\}$, *we define the experiment* xx-yy-$IND^{\mathsf{MCFE}}_\beta$ *in Fig. 1, where the oracles are defined as:*

Corruption oracle $\mathsf{QCor}(i)$**:** *Outputs the encryption key* sk_i *of slot* i. *We denote by* $\mathcal{CS}$ *the set of corrupted slots at the end of the experiment.*

Encryption oracle $\mathsf{QEnc}(i, x_i^0, x_i^1, \ell)$**:** *Outputs* $\mathsf{ct}_{i,\ell} = \mathsf{Enc}(\mathsf{pp}, \mathsf{sk}_i, x_i^\beta, \ell)$ *on a query* (i, x_i^0, x_i^1, ℓ). *We denote by* $Q_{i,\ell}$ *the number of queries of the form* $\mathsf{QEnc}(i, \cdot, \cdot, \ell)$.

Key derivation oracle $\mathsf{QKeyD}(f)$**:** *Outputs* $\mathsf{sk}_f = \mathsf{KeyDer}(\mathsf{pp}, \mathsf{msk}, f)$.

and where Condition (*) *holds if all the following conditions hold:*

- *If* $i \in \mathcal{CS}$ *(i.e., slot* i *is corrupted): for any query* $\mathsf{QEnc}(i, x_i^0, x_i^1, \ell)$, $x_i^0 = x_i^1$.
- *For any label* $\ell \in \mathsf{Labels}$, *for any family of queries* $\{\mathsf{QEnc}(i, x_i^0, x_i^1, \ell)\}_{i \in [n] \setminus \mathcal{CS}}$, *for any family of inputs* $\{x_i \in \mathcal{X}_{\rho,i}\}_{i \in \mathcal{CS}}$, *for any query* $\mathsf{QKeyD}(f)$, *we define* $x_i^0 = x_i^1 = x_i$ *for any slot* $i \in \mathcal{CS}$, $\boldsymbol{x}^b = (x_1^b, \ldots, x_n^b)$ *for* $b \in \{0,1\}$, *and we require that:*
$$f(\boldsymbol{x}^0) = f(\boldsymbol{x}^1) \ .$$
We insist that if one index $i \notin \mathcal{CS}$ *is not queried for the label* ℓ, *there is no restriction.*
- *When* $\text{yy} = \text{one}$*: for any slot* $i \in [n]$ *and* $\ell \in \mathsf{Labels}$, $Q_{i,\ell} \in \{0,1\}$, *and if* $Q_{i,\ell} = 1$, *then for any slot* $j \in [n] \setminus \mathcal{CS}$, $Q_{j,\ell} = 1$. *In other words, for any label, either the adversary makes no encryption query or makes exactly one encryption query for each* $i \in [n] \setminus \mathcal{CS}$.
- *When* $\text{yy} = \text{pos}$*: for any slot* $i \in [n]$ *and* $\ell \in \mathsf{Labels}$, *if* $Q_{i,\ell} > 0$, *then for any slot* $j \in [n] \setminus \mathcal{CS}$, $Q_{j,\ell} > 0$. *In other words, for any label, either the adversary makes no encryption query or makes at least one encryption query for each slot* $i \in [n] \setminus \mathcal{CS}$.

We define the advantage of an adversary $\mathcal{A}$ *in the following way:*

$$\mathsf{Adv}^{\text{xx-yy-}IND}_{\mathsf{MCFE},\mathcal{A}}(\lambda, n) = \big| \Pr[\text{xx-yy-IND}^{\mathsf{MCFE}}_0(\lambda, n, \mathcal{A}) = 1] - \Pr[\text{xx-yy-IND}^{\mathsf{MCFE}}_1(\lambda, n, \mathcal{A}) = 1] \big| \ .$$

A multi-client functional encryption scheme MCFE *is* xx-yy-*IND secure, if for any* n, *for any polynomial-time adversary* $\mathcal{A}$, *there exists a negligible function* negl *such that:* $\mathsf{Adv}^{\text{xx-yy-}IND}_{\mathsf{MCFE},\mathcal{A}}(\lambda, n) \leq \text{negl}(\lambda)$.

We omit n when it is clear from the context. We also often omit $\mathcal{A}$ from the parameter of experiments or games when it is clear from context.

$\textbf{sta-yy-IND}^{\mathsf{MCFE}}_{\beta}(\lambda, n, \mathcal{A})$	$\textbf{adt-yy-IND}^{\mathsf{MCFE}}_{\beta}(\lambda, n, \mathcal{A})$
$\mathcal{CS} \leftarrow \mathcal{A}(1^\lambda, 1^n)$	$\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, 1^n)$
$\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, 1^n)$	$(\{\mathsf{sk}_i\}_{i\in[n]}, \mathsf{msk}) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$
$(\{\mathsf{sk}_i\}_{i\in[n]}, \mathsf{msk}) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$	$\alpha \leftarrow \mathcal{A}^{\mathsf{QCor}(\cdot),\mathsf{QEnc}(\cdot,\cdot,\cdot,\cdot),\mathsf{QKeyD}(\cdot)}(\mathsf{pp})$
$\alpha \leftarrow \mathcal{A}^{\mathsf{QEnc}(\cdot,\cdot,\cdot,\cdot),\mathsf{QKeyD}(\cdot)}(\mathsf{pp}, \{\mathsf{sk}_i\}_{i\in\mathcal{CS}})$	**Output:** α if Condition (*) is satisfied, or a uniform bit otherwise
Output: α if Condition (*) is satisfied, or a uniform bit otherwise	

Fig. 1. Security games for MCFE

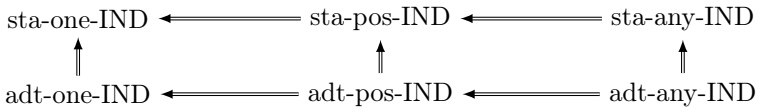

Fig. 2. Relations between the MCFE security notions (arrows indicate implication or being "a stronger security notion than")

We summarize the relations between the six security notions in Fig. 2. Multi-input functional encryption (MIFE) and functional encryption (FE) are special cases of MCFE. MIFE is MCFE without corruption, and FE is the special case of $n = 1$ (in which case, MIFE and MCFE coincide as there is no non-trivial corruption). Therefore, for single-input FE schemes, sta-any-IND = adt-any-IND = any-IND corresponds to the secret-key version of the standard adaptive indistinguishability notion used in [ALS16]. The security notions considered in [CDG+18a] are actually xx-pos-IND and so are the MIFE notions of [ACF+18]. An xx-one-IND MCFE is also called a one-time secure scheme.

2.2 Decentralized Multi-Client Functional Encryption

Now, we introduce the definition of decentralized multi-client functional encryption (DMCFE) [CDG+18a]. As for our definition of MCFE, we separate the algorithm Setup which generates public parameters defining in particular the set of functions, from the algorithm KeyGen (see Remark 2.2).

Definition 2.4. ***(Decentralized Multi-Client Functional Encryption)*** *Let $\mathcal{F} = \{\mathcal{F}_\rho\}_\rho$ be a family (indexed by ρ) of sets $\mathcal{F}_\rho$ of functions $f\colon \mathcal{X}_{\rho,1} \times \cdots \times \mathcal{X}_{\rho,n_\rho} \to \mathcal{Y}_\rho$. Let $\mathsf{Labels} = \{0,1\}^*$ or $\{\bot\}$ be a set of labels. A decentralized multi-client functional encryption scheme (DMCFE) for the function family $\mathcal{F}$ and the label set Labels is a tuple of six algorithms $\mathsf{DMCFE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{KeyDerShare}, \mathsf{KeyDerComb}, \mathsf{Enc}, \mathsf{Dec})$:*

$\mathsf{Setup}(1^\lambda, 1^n)$ *is defined as for MCFE in Definition 2.1.*
$\mathsf{KeyGen}(\mathsf{pp})$**:** *Takes as input the public parameters pp and outputs n secret keys $\{\mathsf{sk}_i\}_{i\in[n]}$.*
$\mathsf{KeyDerShare}(\mathsf{pp}, \mathsf{sk}_i, f)$**:** *Takes as input the public parameters pp, a secret key sk_i from position i and a function $f \in \mathcal{F}_\rho$, and outputs a partial functional decryption key $\mathsf{sk}_{i,f}$.*
$\mathsf{KeyDerComb}(\mathsf{pp}, \mathsf{sk}_{1,f}, \ldots, \mathsf{sk}_{n,f})$**:** *Takes as input the public parameters pp, n partial functional decryption keys $\mathsf{sk}_{1,f}, \ldots, \mathsf{sk}_{n,f}$ and outputs the functional decryption key sk_f.*
$\mathsf{Enc}(\mathsf{pp}, \mathsf{sk}_i, x_i, \ell)$ *is defined as for MCFE in Definition 2.1.*
$\mathsf{Dec}(\mathsf{pp}, \mathsf{sk}_f, \mathsf{ct}_{1,\ell}, \ldots, \mathsf{ct}_{n,\ell})$ *is defined as for MCFE in Definition 2.1.*

A scheme DMCFE is correct, if for all $\lambda, n \in \mathbb{N}$, $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, 1^n)$, $f \in \mathcal{F}_\rho$, $\ell \in \mathsf{Labels}$, $x_i \in \mathcal{X}_{\rho,i}$, when $\{\mathsf{sk}_i\}_{i\in[n]} \leftarrow \mathsf{KeyGen}(\mathsf{pp})$, $\mathsf{sk}_{i,f} \leftarrow \mathsf{KeyDerShare}(\mathsf{sk}_i, f)$ for $i \in [n]$, and $\mathsf{sk}_f \leftarrow \mathsf{KeyDerComb}(\mathsf{pp}, \mathsf{sk}_{1,f}, \ldots, \mathsf{sk}_{n,f})$, we have

$$\Pr\left[\mathsf{Dec}(\mathsf{pp}, \mathsf{sk}_f, \mathsf{Enc}(\mathsf{pp}, \mathsf{sk}_1, x_1, \ell), \ldots, \mathsf{Enc}(\mathsf{pp}, \mathsf{sk}_n, x_n, \ell)) = f(x_1, \ldots, x_n)\right] = 1 \ .$$

We remark that there is no master secret key msk. Furthermore, similarly to [CDG+18a], our definition does not explicitly ask the setup to be decentralized. However, all our constructions allow for the setup to be easily decentralized, at least assuming that the original schemes have such a property in the case of our compilers.

We consider a similar security definition for the decentralized multi-client scheme. We point out that contrary to [CDG+18a], we do not differentiate encryption keys from secret keys. This is without loss of generality, as corruptions in [CDG+18a] only allow to corrupt both keys at the same time.

Definition 2.5. ***(Security of DMCFE)*** *The* xx-yy-*IND security notion of an DMCFE scheme (*$\mathrm{xx} \in \{\mathrm{sta}, \mathrm{adt}\}$ *and* $\mathrm{yy} \in \{\mathrm{one}, \mathrm{any}, \mathrm{pos}\}$*) is similar to the one of an MCFE (Definition 2.3), except that there is no master secret key* msk *and the key derivation oracle is now defined as:*

Key derivation oracle $\mathsf{QKeyD}(f)$**:** *Computes* $\mathsf{sk}_{i,f} := \mathsf{KeyDerShare}(\mathsf{pp}, \mathsf{sk}_i, f)$ *for* $i \in [n]$ *and outputs* $\{\mathsf{sk}_{i,f}\}_{i\in[n]}$.

2.3 Inner-Product Functionality

We describe the functionalities supported by the constructions in this paper, by considering the index ρ of $\mathcal{F}$ in more detail.

The index of the family is defined as $\rho = (\mathcal{R}, n, m, X, Y)$ where $\mathcal{R}$ is either $\mathbb{Z}$ or $\mathbb{Z}_L$ for some integer L, and n, m, X, Y are positive integers. If X, Y are omitted, then $X = Y = L$ is used (i.e., no constraint).

This defines $\mathcal{F}_\rho = \{f_{\boldsymbol{y}_1,\ldots,\boldsymbol{y}_n} : (\mathcal{R}^m)^n \to \mathcal{R}\}$ where

$$f_{\boldsymbol{y}_1,\ldots,\boldsymbol{y}_n}(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n) = \sum_{i=1}^{n} \langle \boldsymbol{x}_i, \boldsymbol{y}_i \rangle = \langle \boldsymbol{x}, \boldsymbol{y} \rangle \ ,$$

where the vectors satisfy the following bounds: $\|\boldsymbol{x}_i\|_\infty < X, \|\boldsymbol{y}_i\|_\infty < Y$ for $i \in [n]$, and where $\boldsymbol{x} \in \mathcal{R}^{mn}$ and $\boldsymbol{y} \in \mathcal{R}^{mn}$ are the vectors corresponding to the concatenation of the n vectors $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$ and $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_n$ respectively.

2.4 Symmetric Encryption

For our second compiler (Section 4.1), we make use of a symmetric encryption scheme $\mathsf{SE} = (\mathsf{Enc}_{\mathsf{SE}}, \mathsf{Dec}_{\mathsf{SE}})$ that is indistinguishable secure under chosen plaintext attacks (IND-CPA) and whose keys are uniform strings in $\{0,1\}^\lambda$ as defined by [BDJR97].

$\mathsf{Enc}_{\mathsf{SE}}(\mathsf{K}, x)$**:** Takes as input a key $\mathsf{K} \in \{0,1\}^\lambda$ and a message x to encrypt, and outputs the ciphertext ct.
$\mathsf{Dec}_{\mathsf{SE}}(\mathsf{K}, \mathsf{ct})$**:** Takes as input a key K and a ciphertext ct to decrypt, and outputs a message x.

We denote with $\mathsf{Adv}^{\text{IND-CPA}}_{\mathsf{SE},\mathcal{A}}(\lambda)$ the advantage of an adversary guessing β in the following game: the challenger picks $\mathsf{K} \leftarrow \{0,1\}^\lambda$ and gives $\mathcal{A}$ access to an encryption oracle $\mathsf{QEnc}(x_i^0, x_i^1)$ that outputs $\mathsf{ct} = \mathsf{Enc}_{\mathsf{SE}}(\mathsf{K}, x_i^\beta)$ on a query (x^0, x^1).

3 From MCFE to DMCFE

In this section, we describe our first compiler which allows the decentralization of MCFE schemes that satisfy an additional property, called special key derivation. We start by defining this property and showing that existing schemes from [ACF+18, CDG+18a] satisfy it. Next, we describe the compiler and prove its security when the underlying modulus of the special key derivation property is prime. Finally, we extend the proof to the case where this modulus is a hard-to-factor composite number.

3.1 Special Key Derivation Property

Definition 3.1 (MCFE with Special Key Derivation). *An MCFE scheme* $\mathsf{MCFE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{KeyDer}, \mathsf{Enc}, \mathsf{Dec})$ *for a family of functions* $\mathcal{F}$ *and a set of labels* Labels *has the special key derivation property modulo* L *if:*[7]

- *Secret keys* sk_i *generated by* KeyGen *have the following form:* $\mathsf{sk}_i = (i, s_i, \{\boldsymbol{u}_i^k\}_{k\in[\kappa]})$*, where* $s_i \in \{0,1\}^*$*, and* $\boldsymbol{u}_i^k \in \mathbb{Z}_L^m$*, and* κ *and* m *are positive integers implicitly depending on the public parameters* pp*.*
- $\mathsf{sk}_f \leftarrow \mathsf{KeyDer}(\mathsf{pp}, \mathsf{msk}, f)$ *outputs* $\mathsf{sk}_f = (\{s_{i,f}\}_{i\in[n]}, \{\mathsf{dk}_f^k\}_{k\in[\kappa]})$*, where* $s_{i,f}$ *is a (polynomial-time) function of* pp*,* i*,* s_i*, and* f*, while:*

$$\mathsf{dk}_f^k = \sum_{i=1}^{n} \langle \boldsymbol{u}_i^k, \boldsymbol{y}_{i,f}^k \rangle = \langle \boldsymbol{u}^k, \boldsymbol{y}_f^k \rangle \ ,$$

where $\boldsymbol{y}_{i,f}^k \in \mathbb{Z}_L^m$ *is a (polynomial-time) function of* pp*,* i*, and* f*, and* $\boldsymbol{u}^k$ *and* $\boldsymbol{y}_f^k$ *are the vectors in* $\mathbb{Z}_L^{mn}$ *corresponding to the concatenation of the vectors* $\{\boldsymbol{u}_i^k\}_{i\in[n]}$ *and* $\{\boldsymbol{y}_{i,f}^k\}_{i\in[n]}$ *respectively.*

Without loss of generality for MCFE with the special key derivation property, we can suppose that $\mathsf{msk} = \{\mathsf{sk}_i\}_{i\in[n]}$. We also remark that we do not require any property of the family of functions $\mathcal{F}$ and that our compiler could be applicable to more general MCFE than inner-product ones.

3.2 Instantiations

The MCFE construction of Chotard et al. [CDG+18a, Section 4] satisfies the special key derivation property modulo $L = p$ (the order of the cyclic group), with $\kappa = 2$ and $\boldsymbol{y}_f^k = \boldsymbol{y}$, when $f : \boldsymbol{x} \mapsto \langle \boldsymbol{x}, \boldsymbol{y} \rangle$.

The generic constructions of Abdalla et al. [ACF+18, Section 3] (both over $\mathbb{Z}$ and $\mathbb{Z}_L$, see also Section 5) satisfy the special key derivation property modulo L (where L is the modulo used for the information-theoretic MIFE/MCFE with one-time security) with $\boldsymbol{y}_f^k = \boldsymbol{y}$. The instantiations from MDDH, LWE, and Paillier ([ACF+18, Section 4]) use $L = p$ the prime order of the cyclic group, $L = q$ the prime modulo for LWE (we need $L = q$ to be prime for our compiler), $L = N = pq$ the modulus used for Paillier respectively.

3.3 Compiler for Prime Moduli

We start by presenting our compiler from MCFE schemes with the special key derivation property modulo a prime L in Fig. 3. Correctness follows directly from the fact that:

$$\begin{aligned}\sum_{i=1}^{n} \mathsf{dk}_{i,f}^k &= \sum_{i=1}^{n} \langle \boldsymbol{u}_i^k, \boldsymbol{y}_{i,f}^k \rangle + \sum_{i=1}^{n} \langle \boldsymbol{v}_i^k, \boldsymbol{y}_f^k \rangle \\ &= \mathsf{dk}_f^k + \langle \sum_{i=1}^{n} \boldsymbol{v}_i^k, \boldsymbol{y}_f^k \rangle = \mathsf{dk}_f^k + \langle \boldsymbol{0}, \boldsymbol{y}_f^k \rangle = \mathsf{dk}_f^k \ .\end{aligned}$$

We insist on the fact that while vectors $\boldsymbol{u}_i^k$ and $\boldsymbol{y}_{i,f}^k$ are m-dimensional, vectors $\boldsymbol{v}_i^k$ and $\boldsymbol{y}_f^k$ are (mn)-dimensional.

We have the following security theorem.

Theorem 3.2. *Let* $\mathsf{MCFE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{KeyDer}, \mathsf{Enc}, \mathsf{Dec})$ *be an MCFE construction for a family of functions* $\mathcal{F}$ *and a set of labels* Labels*. We suppose that* MCFE *has the special key derivation property modulo a prime* L*. For any* $\mathrm{xx} \in \{\mathrm{sta}, \mathrm{adt}\}$ *and any* $\mathrm{yy} \in \{\mathrm{one}, \mathrm{pos}, \mathrm{any}\}$*, if* MCFE *is an* xx-yy*-IND-secure MCFE scheme, then the scheme* DMCFE' *depicted in Fig. 3 is an* xx-yy*-IND-secure DMCFE scheme. Namely, for any PPT adversary* $\mathcal{A}$*, there exist a PPT adversary* $\mathcal{B}$ *such that:*

$$\mathsf{Adv}^{\text{xx-yy-IND}}_{\mathsf{DMCFE}',\mathcal{A}}(\lambda, n) \leq \mathsf{Adv}^{\text{xx-yy-IND}}_{\mathsf{MCFE},\mathcal{B}}(\lambda, n) \ .$$

[7] The integer L can depend on the public parameters pp.

$\underline{\mathsf{Setup}'(1^\lambda, 1^n) :}$

Return $\mathsf{Setup}(1^\lambda, 1^n)$

$\underline{\mathsf{KeyGen}'(\mathsf{pp}) :}$

$(\{\mathsf{sk}_i\}_{i\in[n]}, \mathsf{msk}) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$

Recall that $\mathsf{sk}_i = (i, s_i, \{\boldsymbol{u}_i^k\}_{k\in[\kappa]})$

For $k \in [\kappa]$:

For $i \in [n-1]$, $\boldsymbol{v}_i^k \leftarrow \mathbb{Z}_L^M$

$\boldsymbol{v}_n^k := -\sum_{i=1}^{n-1} \boldsymbol{v}_i^k \bmod L$

Return $\{\mathsf{sk}'_i = (\mathsf{sk}_i, \{\boldsymbol{v}_i^k\}_{k\in[\kappa]})\}_{i\in[n]}$

$\underline{\mathsf{Enc}'(\mathsf{pp}, \mathsf{sk}'_i, x_i, \ell) :}$

Parse $\mathsf{sk}'_i = (\mathsf{sk}_i, \{\boldsymbol{v}_i^k\}_{k\in[\kappa]})$

Return $\mathsf{ct}_{i,\ell} \leftarrow \mathsf{Enc}(\mathsf{pp}, \mathsf{sk}_i, x_i, \ell)$

$\underline{\mathsf{KeyDerShare}'(\mathsf{pp}, \mathsf{sk}'_i, f) :}$

Parse $\mathsf{sk}'_i = (\mathsf{sk}_i, \{\boldsymbol{v}_i^k\}_{k\in[\kappa]})$

For $k \in [\kappa]$, $\mathsf{dk}_{i,f}^k := \langle \boldsymbol{u}_i^k, \boldsymbol{y}_{i,f}^k \rangle + \langle \boldsymbol{v}_i^k, \boldsymbol{y}_f^k \rangle$

Return $\mathsf{sk}'_{i,f} := (s_{i,f}, \{\mathsf{dk}_{i,f}^k\}_{k\in[\kappa]})$

$\underline{\mathsf{KeyDerComb}'(\mathsf{pp}, \{\mathsf{sk}'_{i,f}\}_{i\in[n]}) :}$

Parse $\{\mathsf{sk}'_{i,f} = (s_{i,f}, \{\mathsf{dk}_{i,f}^k\}_{k\in[\kappa]})\}_{i\in[n]}$

For $k \in [\kappa]$, $\mathsf{dk}_f^k := \sum_{i=1}^{n} \mathsf{dk}_{i,f}^k$

Return $\mathsf{sk}'_f = (\{s_{i,f}\}_{i\in[n]}, \{\mathsf{dk}_f^k\}_{k\in[\kappa]})$

$\underline{\mathsf{Dec}'(\mathsf{pp}, \mathsf{sk}'_f, \{\mathsf{ct}_{i,\ell}\}_{i\in[n]}) :}$

Return $\mathsf{Dec}(\mathsf{pp}, \mathsf{sk}'_f, \{\mathsf{ct}_{i,\ell}\}_{i\in[n]})$

Fig. 3. Compiler from MCFE to DMCFE′: $s_{i,f}$ is a function of pp, i, s_i, f and $\boldsymbol{y}_{i,f}^k$ is a function of pp, i, f, and k. $M = mn$.

Below, we provide a proof sketch of the theorem. The formal proof is in Appendix A.1.

Proof (Theorem 3.2 — sketch). In this sketch, we focus on a setting without corruption and where L is a prime number. For the sake of simplicity, we also suppose that $\kappa = 1$ and $s_{i,f}$ is an empty string, so that we can omit the superscript k and we have $\mathsf{sk}'_{i,f} = \mathsf{dk}_{i,f} = \langle \boldsymbol{u}_i, \boldsymbol{y}_{i,f} \rangle + \langle \boldsymbol{v}_i, \boldsymbol{y}_f \rangle$. We can define $\boldsymbol{u}'_i \in \mathbb{Z}_L$ to be $\boldsymbol{u}_i$ "padded with 0" so that we can write: $\langle \boldsymbol{u}_i, \boldsymbol{y}_{i,f} \rangle = \langle \boldsymbol{u}'_i, \boldsymbol{y}_f \rangle$ (recall that $\boldsymbol{y}_f$ is just the concatenation of the vectors $\boldsymbol{y}_{i,f}$ for $i \in [n]$). Thus we have:

$$\mathsf{sk}'_{i,f} = \mathsf{dk}_{i,f} = \langle \boldsymbol{u}'_i, \boldsymbol{y}_f \rangle + \langle \boldsymbol{v}_i, \boldsymbol{y}_f \rangle = \langle \boldsymbol{u}'_i + \boldsymbol{v}_i, \boldsymbol{y}_f \rangle \ .$$

Now, we remark that from keys $\mathsf{dk}_{i,g}$ for $g \in \{f_1, \ldots, f_q\}$, one can compute the key $\mathsf{dk}_{i,f}$ for any f such that $\boldsymbol{y}_f$ is in the subspace generated by $\boldsymbol{y}_{f_1}, \ldots, \boldsymbol{y}_{f_q}$. Indeed, if $\boldsymbol{y} = \sum_{j=1}^{q} \mu_j \cdot \boldsymbol{y}_{f_j}$, for some $\mu_1, \ldots, \mu_q \in \mathbb{Z}_L$, then: $\mathsf{dk}_{i,f} = \sum_{j=1}^{q} \mu_j \cdot \mathsf{dk}_{i,f_j}$.

Let S be the set of functions f queried to QKeyD such that the family of vector $\{\boldsymbol{y}_f\}_{f\in S}$ is linearly independent. We compute the $\mathsf{dk}_{i,f}$ of linearly dependent functions as outlined above. We now look at linearly independent functions. As the vectors $\boldsymbol{v}_i$ are uniformly distributed under the constraints $\sum_{i=1}^{n} \boldsymbol{v}_i = \boldsymbol{0}$ (by definition of Setup′), linear algebra ensures that the values $\{\langle \boldsymbol{v}_i, \boldsymbol{y}_f \rangle\}_{i\in[n], f\in S}$ are distributed uniformly under the constraints $\sum_{i=1}^{n} \langle \boldsymbol{v}_i, \boldsymbol{y}_f \rangle = 0$ for $f \in S$. Thus, from Section 3.3, we get that for any $f \in S$, $\{\mathsf{dk}_{i,f}\}_{i\in[n]}$ is a fresh additive secret sharing of

$$\sum_{i=1}^{n} \mathsf{dk}_{i,f} = \sum_{i=1}^{n} \langle \boldsymbol{u}'_i, \boldsymbol{y}_f \rangle = \sum_{i=1}^{n} \langle \boldsymbol{u}_i, \boldsymbol{y}_{i,f} \rangle = \mathsf{dk}_f \ ,$$

and hence can be simulated knowing only $\mathsf{dk}_f = \mathsf{KeyDer}(\mathsf{msk}, f)$ (but not the vectors $\boldsymbol{u}_i$ themselves, which are parts of the secret keys sk_i). In other words queries to the oracle $\mathsf{QKeyD}(f)$ in the security game of DMCFE′ can be simulated just from $\mathsf{KeyDer}(\mathsf{pp}, \mathsf{msk}, f)$ (or equivalently just from queries to the oracle $\mathsf{QKeyD}(f)$ in the security game of MCFE).

Thus, we have a perfect reduction from the security of DMCFE′ to the security of MCFE. □

3.4 Extension to Hard-to-Factor Moduli

We can extend the previous scheme to moduli L which are hard to factor. This is required for the Paillier instantiation from [ACF+18, Section 4.3].

Let us provide formal details.

Definition 3.3 (Factorization). *Let* GenL *be a PPT algorithm taking as input the security parameter* 1^λ *and outputing a number* $L \geq 2$*. We define the experiment* $\mathsf{Factor}_{\mathsf{GenL}}(\lambda, \mathcal{A})$ *for an adversary* $\mathcal{A}$ *as follows: it outputs* 1 *if on input* $L \leftarrow \mathsf{GenL}(1^\lambda)$*, the adversary outputs two integers* $L_1, L_2 \geq 2$*, such that* $L_1 \cdot L_2 = L$*. The advantage of* $\mathcal{A}$ *is* $\mathsf{Adv}^{\mathsf{Factor}}_{\mathsf{GenL},\mathcal{A}}(\lambda) = \Pr[\mathsf{Factor}_{\mathsf{GenL}}(\lambda, \mathcal{A})]$*. Factorization is hard for* GenL *if the advantage of any PPT adversary* $\mathcal{A}$ *is negligible in* λ*.*

We have the following security theorem proven in Appendix A.2.

Theorem 3.4. *Let* $\mathsf{MCFE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{KeyDer}, \mathsf{Enc}, \mathsf{Dec})$ *be an MCFE construction for an ensemble of functions* $\mathcal{F}$ *and a set of labels* Labels*. We suppose that* MCFE *has the special key derivation property modulo an integer* L*, which is part of the public parameter* pp *and generated as* $L \leftarrow \mathsf{GenL}(1^\lambda)$ *in the setup, for some polynomial-time algorithm. We assume that factorization is hard for* GenL*. For any* $\mathrm{xx} \in \{\mathrm{sta}, \mathrm{adt}\}$ *and any* $\mathrm{yy} \in \{\mathrm{one}, \mathrm{pos}, \mathrm{any}\}$*, if* MCFE *is an* xx-yy*-IND-secure MCFE scheme, then the scheme* DMCFE' *depicted in Fig. 3 is an* xx-yy*-IND-secure DMCFE scheme. Namely, for any PPT adversary* $\mathcal{A}$*, there exist two PPT adversaries* $\mathcal{B}$ *and* $\mathcal{B}'$ *such that:*

$$\mathsf{Adv}^{\text{xx-yy-IND}}_{\mathsf{DMCFE}',\mathcal{A}}(\lambda, n) \leq \mathsf{Adv}^{\text{xx-yy-IND}}_{\mathsf{MCFE},\mathcal{B}}(\lambda, n) + 2 \cdot \mathsf{Adv}^{\mathsf{Factor}}_{\mathsf{GenL},\mathcal{B}'}(\lambda) \ .$$

4 From xx-pos-IND to xx-any-IND Security

We present two compilers transforming pos-IND-secure MIFE, MCFE, and DMCFE schemes into any-IND schemes. These compilers essentially force the adversary to ask for at least one ciphertext per position i (and per label, for labeled schemes).

The first compiler works for sta-pos-IND and adt-pos-IND-secure schemes without labels ($\mathsf{Labels} = \{\bot\}$) and only requires an IND-CPA symmetric encryption scheme to work. We prove it for the adt-pos-IND case as the proof for sta-pos-IND is simpler. The second compiler supports labeled schemes, but is in the random oracle model. Although our presentation is for DMCFE, the compilers can be adapted to work for MCFE schemes in a straightforward way.

Regarding efficiency, both compilers add $2n-1$ symmetric keys (i.e., λ-bit strings) to each secret key sk_i, and n symmetric keys to each ciphertext ct_i (plus the overhead due to symmetric encryption, which can be as low as λ bits using stream ciphers for example). (Partial) functional decryption keys and public parameters are unchanged. For the first compiler, the computational complexity overhead essentially consists in one symmetric encryption of the original ciphertext for functional encryption, and n symmetric decryptions for functional decryption. The second compiler uses a specific encryption scheme based on hash functions (modeled as random oracles) which requires $2n-1$ hash function evaluations in addition to the encryption algorithm.

4.1 Compiler for DMCFE Schemes without Labels

The compiler without labels is described in Fig. 4. where SE is an IND-CPA symmetric-key encryption scheme. We show the following security theorem.

Theorem 4.1. *Let* $\mathsf{DMCFE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{KeyDerShare}, \mathsf{KeyDerComb}, \mathsf{Enc}, \mathsf{Dec})$ *be an* adt-pos*-IND-secure DMCFE scheme without labels (*$\mathsf{Labels} = \{\bot\}$*) for a family of functions* $\mathcal{F}$*. Let* $\mathsf{SE} = (\mathsf{Enc}_{\mathsf{SE}}, \mathsf{Dec}_{\mathsf{SE}})$ *be an IND-CPA symmetric-key encryption scheme. Then the DMCFE scheme* $\mathsf{DMCFE}' = (\mathsf{Setup}', \mathsf{KeyGen}', \mathsf{KeyDerShare}', \mathsf{KeyDerComb}', \mathsf{Enc}', \mathsf{Dec}')$ *described in Fig. 4 is an* adt-any*-IND-secure DMCFE scheme. Namely, for any PPT adversary* $\mathcal{A}$*, there exist PPT adversaries* $\mathcal{B}$ *and* $\mathcal{B}'$ *such that:*

$$\mathsf{Adv}^{\text{adt-any-IND}}_{\mathsf{DMCFE}',\mathcal{A}}(\lambda, n) \leq \mathsf{Adv}^{\text{adt-pos-IND}}_{\mathsf{DMCFE},\mathcal{B}}(\lambda, n) + n \cdot \mathsf{Adv}^{\text{IND-CPA}}_{\mathsf{SE},\mathcal{B}'}(\lambda) \ .$$

$\underline{\mathsf{Setup}'(1^\lambda, 1^n):}$

Return $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, 1^n)$

$\underline{\mathsf{KeyGen}'(\mathsf{pp}):}$

$\{\mathsf{sk}_i\}_{i\in[n]} \leftarrow \mathsf{KeyGen}(\mathsf{pp})$

For $i \in [n]$:

$\quad \mathsf{k}_{i,1}, \ldots, \mathsf{k}_{i,n} \leftarrow \{0,1\}^\lambda$

$\quad \mathsf{K}_i = \oplus_{j\in[n]} \mathsf{k}_{i,j}$

Return $\{\mathsf{sk}'_i = (\mathsf{sk}_i, \mathsf{K}_i, \{\mathsf{k}_{i,j}, \mathsf{k}_{j,i}\}_{j\in[n]})\}_{i\in[n]}$

$\underline{\mathsf{Enc}'(\mathsf{pp}, \mathsf{sk}'_i, x_i):}$

Parse $\mathsf{sk}'_i = (\mathsf{sk}_i, \mathsf{K}_i, \{\mathsf{k}_{i,j}, \mathsf{k}_{j,i}\}_{j\in[n]})$

$\mathsf{ct}_i \leftarrow \mathsf{Enc}(\mathsf{pp}, \mathsf{sk}_i, x_i)$

$\mathsf{ct}'_i \leftarrow \mathsf{Enc}_{\mathsf{SE}}(\mathsf{K}_i, \mathsf{ct}_i)$

Return $(\mathsf{ct}'_i, \{\mathsf{k}_{j,i}\}_{j\in[n]})$

$\underline{\mathsf{KeyDerShare}'(\mathsf{pp}, \mathsf{sk}'_i, f):}$

Parse $\mathsf{sk}'_i = (\mathsf{sk}_i, \mathsf{K}_i, \{\mathsf{k}_{i,j}, \mathsf{k}_{j,i}\}_{j\in[n]})$

Return $\mathsf{sk}'_{i,f} \leftarrow \mathsf{KeyDerShare}(\mathsf{sk}_i, f)$

$\underline{\mathsf{KeyDerComb}'(\mathsf{pp}, \{\mathsf{sk}'_{i,f}\}_{i\in[n]}):}$

$\mathsf{sk}_f := \mathsf{KeyDerComb}(\mathsf{pp}, \{\mathsf{sk}'_{i,f}\}_{i\in[n]})$

Return sk_f

$\underline{\mathsf{Dec}'(\mathsf{pp}, \mathsf{sk}_f, \mathsf{ct}''_1, \ldots, \mathsf{ct}''_n):}$

Parse $\{\mathsf{ct}''_i = (\mathsf{ct}'_i, \{\mathsf{k}_{j,i}\}_{j\in[n]})\}_{i\in[n]}$

For $i \in [n]$:

$\quad \mathsf{K}_i = \oplus_{j\in[n]} \mathsf{k}_{i,j}$

$\quad \mathsf{ct}_i \leftarrow \mathsf{Dec}_{\mathsf{SE}}(\mathsf{K}_i, \mathsf{ct}'_i)$

Return $\mathsf{Dec}(\mathsf{pp}, \mathsf{sk}_f, \mathsf{ct}_1, \ldots, \mathsf{ct}_n)$.

Fig. 4. Compiler from an xx-pos-IND DMCFE DMCFE without labels into an xx-any-IND DMCFE DMCFE' using an IND-CPA symmetric-key encryption scheme SE

Proof. An encryption query on the i-th slot is denoted as (x_i^0, x_i^1).

In the proof we need to consider two different cases:

1. In all uncorrupted positions $i \notin \mathcal{CS}$, at least one query has been made, $Q_i \geq 1$.
2. In an uncorrupted position $i \notin \mathcal{CS}$, zero queries have been made, $Q_i = 0$.

We begin our proof by considering the first point.

Lemma 4.2. *Let* $\mathsf{DMCFE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{KeyDerShare}, \mathsf{KeyDerComb}, \mathsf{Enc}, \mathsf{Dec})$ *be an* adt-pos-*IND-secure DMCFE construction without labels (*$\mathsf{Labels} = \{\perp\}$*) for a family of functions* $\mathcal{F}$*. Let* $\mathsf{SE} = (\mathsf{Enc}_{\mathsf{SE}}, \mathsf{Dec}_{\mathsf{SE}})$ *be a symmetric-key encryption scheme. Then the DMCFE scheme* $\mathsf{DMCFE}' = (\mathsf{Setup}', \mathsf{KeyGen}', \mathsf{KeyDerShare}', \mathsf{KeyDerComb}', \mathsf{Enc}', \mathsf{Dec}')$ *described in Fig. 4 is* adt-any-*IND secure. Namely, for any PPT adversary* $\mathcal{A}$ *restricted to make* $Q_i \geq 1$ *for all* $i \notin \mathcal{CS}$ *there exist a PPT adversary* $\mathcal{B}$ *such that:*

$$\mathsf{Adv}^{\text{adt-any-IND}}_{\mathsf{DMCFE}',\mathcal{A}}(\lambda, n) \leq \mathsf{Adv}^{\text{adt-pos-IND}}_{\mathsf{DMCFE},\mathcal{B}}(\lambda, n) \ .$$

Proof. We construct an adversary $\mathcal{B}$ against the adt-any-IND security of the scheme DMCFE'. $\mathcal{B}$ generates $\mathsf{k}_{i,1}, \ldots, \mathsf{k}_{i,n}$ and K_i for every $i \in [n]$.

If $\mathcal{A}$ aks a query $\mathsf{QCor}'(i)$, $\mathcal{B}$ asks a query $\mathsf{QCor}(i)$ to its own corruption oracle to obtain the key sk_i and uses it to create sk'_i, which gets forwarded to $\mathcal{A}$.

When the adversary $\mathcal{A}$ asks a query $\mathsf{QEnc}'(i, x_i^0, x_i^1)$, $\mathcal{B}$ directly forwards it to its own encryption oracle. It receives $\mathsf{ct}_i \leftarrow \mathsf{Enc}(\mathsf{pp}, \mathsf{sk}_i, x_i^\beta)$ as a result and uses K_i to generate $\mathsf{ct}'_i \leftarrow \mathsf{Enc}_{\mathsf{SE}}(\mathsf{K}_i, \mathsf{ct}_i)$. This ciphertext gets concatenated with the key shares of the symmetric encryption scheme $\{\mathsf{k}_{j,i}\}_{j\in[n]}$ and sent to $\mathcal{A}$ as an answer to the encryption query.

If $\mathcal{A}$ asks a query $\mathsf{QKeyD}'(f)$, $\mathcal{B}$ forwards it to its own oracle to receive $\mathsf{sk}_{i,f}$, which gets forwarded to $\mathcal{A}$.

It is straightforward to see that the adversary $\mathcal{B}$ perfectly simulates the security game for DMCFE' to $\mathcal{A}$. Hence, we have:

$$\mathsf{Adv}^{\text{adt-any-IND}}_{\mathsf{DMCFE}',\mathcal{A}}(\lambda, n) \leq \mathsf{Adv}^{\text{adt-pos-IND}}_{\mathsf{DMCFE},\mathcal{B}}(\lambda, n) \ .$$

□

We continue with the consideration of the second point.

Lemma 4.3. *Let* $\mathsf{DMCFE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{KeyDerShare}, \mathsf{KeyDerComb}, \mathsf{Enc}, \mathsf{Dec})$ *be a DMCFE construction without labels (*$\mathsf{Labels} = \{\bot\}$*) for a family of functions* $\mathcal{F}$*. Let* $\mathsf{SE} = (\mathsf{Enc}_{\mathsf{SE}}, \mathsf{Dec}_{\mathsf{SE}})$ *be an IND-CPA symmetric-key encryption scheme and let* $Q_i = 0$ *for at least one* $i \notin \mathcal{CS}$*. Then the DMCFE scheme* $\mathsf{DMCFE}' = (\mathsf{Setup}', \mathsf{KeyGen}', \mathsf{KeyDerShare}', \mathsf{KeyDerComb}', \mathsf{Enc}', \mathsf{Dec}')$ *described in Fig. 4 is* adt-any-*IND-secure. Namely, for any PPT adversary* $\mathcal{A}$*, there exists an adversary* $\mathcal{B}'$ *such that:*

$$\mathsf{Adv}^{\text{adt-any-IND}}_{\mathsf{DMCFE}',\mathcal{A}}(\lambda, n) \leq n \cdot \mathsf{Adv}^{\text{IND-CPA}}_{\mathsf{SE},\mathcal{B}'}(\lambda) \ .$$

Proof. We prove this part by using a hybrid argument. We define the games $\mathsf{G}_1, \ldots, \mathsf{G}_n$ in Fig. 5.

$\underline{\mathsf{G}_t(\lambda, n, \mathcal{A}) :}$
$\mathcal{ES} = \{\}$
$\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, 1^n)$
$(\mathsf{sk}_i, \mathsf{K}_i, \{\mathsf{k}_{i,j}, \mathsf{k}_{j,i}\}_{j\in[n]})_{i\in[n]} \leftarrow \mathsf{KeyGen}'(\mathsf{pp})$
$\alpha \leftarrow \mathcal{A}^{\mathsf{QEnc}'(\cdot,\cdot,\cdot),\mathsf{QKeyD}'(\cdot),\mathsf{QCor}'(\cdot)}(\mathsf{pp})$
Output: α

$\mathsf{QEnc}'(i, x_i^0, x_i^1)$
Add i to $\mathcal{ES}$
If $i \notin (\mathcal{CS} \cup \mathcal{ES})$, $\mathsf{k}_{j,i} \leftarrow_R \{0,1\}^\lambda$ for all $j \in [n] \setminus \mathcal{CS}$
If $i \leq t$, return$(\mathsf{Enc}_{\mathsf{SE}}(\mathsf{K}_i, \mathsf{Enc}(\mathsf{pp}, \mathsf{sk}'_i, x_i^0)), \{\mathsf{k}_{j,i}\}_{j\in[n]})$
If $i > t$, return$(\mathsf{Enc}_{\mathsf{SE}}(\mathsf{K}_i, \mathsf{Enc}(\mathsf{pp}, \mathsf{sk}'_i, x_i^1)), \{\mathsf{k}_{j,i}\}_{j\in[n]})$

$\mathsf{QKeyD}'(y)$
Return $\{\mathsf{sk}'_{i,f} \leftarrow \mathsf{KeyDerShare}(\mathsf{pp}, \mathsf{sk}_i, f)\}_{i\in[n]}$

$\mathsf{QCor}'(i)$
If $i \notin \mathcal{CS}$
$\mathsf{k}_{i,j} \leftarrow_R \{0,1\}^\lambda$ for all $j \in [n] \setminus (\mathcal{CS} \cup \mathcal{ES})$, s.t. $\mathsf{K}_i = \oplus_{j\in[n]}\mathsf{k}_{i,j}$
If $i \notin \mathcal{ES}$
$\mathsf{k}_{j,i} \leftarrow_R \{0,1\}^\lambda$ for all $j \in [n] \setminus (\mathcal{CS} \cup \{i\})$
Return $(\mathsf{sk}_i, \mathsf{K}_i, \{\mathsf{k}_{i,j}, \mathsf{k}_{j,i}\}_{j\in[n]})$

Fig. 5. The description of the hybrid used for the reduction to the symmetric-key encryption scheme in Lemma 4.3.

Due to the definition of the game it holds that: Game G_0 corresponds to the experiment adt-any-$\text{IND}^{\mathsf{DMCFE}'}_\beta$ for $\beta = 1$ and G_n to the experiment adt-any-$\text{IND}^{\mathsf{DMCFE}'}_\beta$ for $\beta = 1$ therefore using the triangular inequality, we get:

$$\mathsf{Adv}^{\text{adt-any-IND}}_{\mathsf{DMCFE}',\mathcal{A}}(\lambda, n) \leq \sum_{t=1}^{n} |\mathsf{Win}^{\mathsf{G}_{t-1}}_{\mathcal{A}}(\lambda, n) - \mathsf{Win}^{\mathsf{G}_t}_{\mathcal{A}}(\lambda, n)|.$$

We then conclude by showing that for any t, there exists an adversary $\mathcal{B}_t$ such that

$$|\mathsf{Win}^{\mathsf{G}_{t-1}}_{\mathcal{A}}(\lambda, n) - \mathsf{Win}^{\mathsf{G}_t}_{\mathcal{A}}(\lambda, n)| \leq \mathsf{Adv}^{\text{IND-CPA}}_{\mathsf{SE},\mathcal{B}_t}(\lambda).$$

The adversary $\mathcal{B}'$ of the statement then just picks $t \in [n]$ and simulates $\mathcal{B}_t$. The standard details are omitted here. The adversary $\mathcal{B}_t$ against the IND-CPA security of the symmetric encryption scheme behaves in the following way:

In the first step, $\mathcal{B}_t$ generates the keys K_i and also samples sk_i for all $i \in [n] \setminus \{t\}$ by running the key generation algorithm of DMCFE.

We denote by $\mathcal{ES}$ the set of positions i in which encryption queries have been made.

If $\mathcal{A}$ corrupts a position $i \neq t$, the adversary $\mathcal{B}_t$ samples random values $\mathsf{k}_{i,j}$ for all $j \in [n] \setminus (\mathcal{CS} \cup \mathcal{ES})$ such that $\mathsf{K}_i = \oplus_{j \in [n]} \mathsf{k}_{i,j}$. If the position i has not been corrupted before and if no encryption query has been asked in this position (i.e. $i \notin \mathcal{CS} \cup \mathcal{ES}$), then $\mathcal{B}_t$ samples random values $\mathsf{k}_{j,i}$ for all $j \in [n] \setminus (\mathcal{CS} \cup \{i\})$. If the adversary $\mathcal{A}$ asks a corruption query $\mathsf{QCor}'(t)$, the adversary $\mathcal{B}_t$ directly outputs a random value $r \leftarrow \{0,1\}$. This is due to the fact that, if party t is corrupted the games G_{t-1} and G_t are the same. This results in an advantage equal to 0.

Whenever $\mathcal{A}$ asks a query $\mathsf{QEnc}'(i, x_i^0, x_i^1)$ we consider three different cases. In the first case, $\mathcal{A}$ queries the encryption oracle for $i < t$, then $\mathcal{B}_t$ generates $\mathsf{Enc}_{\mathsf{SE}}(\mathsf{K}_i, \mathsf{Enc}(\mathsf{pp}, \mathsf{sk}_i, x_i^0))$ using the key K_i. The same happens for queries with $i > t$, but with x_i^1 instead of x_i^0, i.e. $\mathsf{Enc}_{\mathsf{SE}}(\mathsf{K}_i, \mathsf{Enc}(\mathsf{pp}, \mathsf{sk}_i, x_i^1))$. In the case that $\mathcal{A}$ asks a query $\mathsf{QEnc}'(t, x_t^0, x_t^1)$, $\mathcal{B}_t$ generates $(\mathsf{Enc}(\mathsf{pp}, \mathsf{sk}_t, x_t^0), \mathsf{Enc}(\mathsf{pp}, \mathsf{sk}_t, x_t^1))$ and sends it to its own encryption oracle to receive $\mathsf{Enc}_{\mathsf{SE}}(\mathsf{Enc}(\mathsf{pp}, \mathsf{sk}_t, x_t^\beta))$. If no encryption has been asked in the position i before and if i is not corrupted (i.e., $i \notin (\mathcal{CS} \cup \mathcal{ES})$) then we sample $\mathsf{k}_{j,i}$ for all $j \in [n] \setminus \mathcal{CS}$. If $i \in (\mathcal{CS} \cup \mathcal{ES})$ then the values $\mathsf{k}_{j,i}$ have already been sampled for all $j \in [n]$. The ciphertext $\mathsf{Enc}_{\mathsf{SE}}(\mathsf{K}_i, \mathsf{Enc}(\mathsf{pp}, \mathsf{sk}_i, x_i^\beta))$ together with $\mathsf{k}_{j,i}, \forall j \in [n]$ are then sent to $\mathcal{A}$ in the last step.

If $\mathcal{A}$ asks a key derivation query $\mathsf{QKeyD}'(f)$, $\mathcal{B}_t$ uses the public parameters pp and the keys $\{\mathsf{sk}_i, f\}_{i \in [n]}$ to generate $\{\mathsf{sk}'_{i,f} \leftarrow \mathsf{KeyDerShare}(\mathsf{pp}, \mathsf{sk}_i, f)\}_{i \in [n]}$ as a response for $\mathcal{A}$.

The reduction shows that for all $t \in [n]$:

$$|\mathsf{Win}_{\mathcal{A}}^{\mathsf{G}_{t-1}}(\lambda, n) - \mathsf{Win}_{\mathcal{A}}^{\mathsf{G}_t}(\lambda, n)| \leq \mathsf{Adv}_{\mathsf{SE}, \mathcal{B}_t}^{\text{IND-CPA}}(\lambda) \ .$$

This results in:

$$\sum_{t=1}^{n} |\mathsf{Win}_{\mathcal{A}}^{\mathsf{G}_{t-1}}(\lambda, n) - \mathsf{Win}_{\mathcal{A}}^{\mathsf{G}_t}(\lambda, n)| \leq \sum_{t=1}^{n} \mathsf{Adv}_{\mathsf{SE}, \mathcal{B}_t}^{\text{IND-CPA}}(\lambda) \ .$$

□

Theorem 4.1 follow from the two above lemmas. □

4.2 Compiler for Labeled DMCFE Schemes

We now present the compiler supporting labels in Fig. 6, where $\mathsf{H}_1 : \{0,1\}^* \to \{0,1\}^\lambda$ and $\mathsf{H}_2 : \{0,1\}^* \to \{0,1\}^{|\mathsf{ct}_i|}$ are two hash functions modeled as random oracles in the security proof. We formally prove the following security theorem in Appendix B.

Theorem 4.4. *Let* $\mathsf{DMCFE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{KeyDerShare}, \mathsf{KeyDerComb}, \mathsf{Enc}, \mathsf{Dec})$ *be an* adt-pos-*IND-secure DMCFE scheme for an ensemble of functions* $\mathcal{F}$ *and set of labels* Labels. *Then the DMCFE scheme* $\mathsf{DMCFE}' = (\mathsf{Setup}', \mathsf{KeyGen}', \mathsf{KeyDerShare}', \mathsf{KeyDerComb}', \mathsf{Enc}', \mathsf{Dec}')$ *described in Fig. 6 is an* adt-any-*IND-secure scheme. Namely, when the hash functions* H_1 *and* H_2 *are modeled as random oracles, for any PPT adversary* $\mathcal{A}$ *there exist a PPT adversary* $\mathcal{B}$ *such that:*

$$\mathsf{Adv}_{\mathsf{DMCFE}', \mathcal{A}}^{\text{adt-any-IND}}(\lambda, n) \leq \mathsf{Adv}_{\mathsf{DMCFE}, \mathcal{B}}^{\text{adt-pos-IND}}(\lambda, n) + \frac{2q_{\mathsf{H}_1} + (2n+1) \cdot (q_{\mathsf{H}_2} q_{\mathsf{QEnc}} + q_{\mathsf{QEnc}}^2)}{2^\lambda} ,$$

where q_{H_1}, q_{H_2}, *and* q_{QEnc} *are the numbers of queries to the oracles* H_1, H_2, *and* QEnc *respectively.*

A high-level overview of the proof of this theorem can be found in Section 1.2.

$\underline{\mathsf{Setup}'(1^\lambda, 1^n):}$

Return $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, 1^n)$

$\underline{\mathsf{KeyGen}'(\mathsf{pp}):}$

$\{\mathsf{sk}_i\}_{i\in[n]} \leftarrow \mathsf{KeyGen}(\mathsf{pp})$

For $i \in [n]$:

$\quad \mathsf{k}_{i,1}, \ldots, \mathsf{k}_{i,n} \leftarrow \{0,1\}^\lambda$

Return $\{\mathsf{sk}'_i = (\mathsf{sk}_i, \{\mathsf{k}_{i,j}, \mathsf{k}_{j,i}\}_{j\in[n]})\}_{i\in[n]}$

$\underline{\mathsf{Enc}'(\mathsf{pp}, \mathsf{sk}'_i, x_i, \ell):}$

Parse $\mathsf{sk}'_i = (\mathsf{sk}_i, \{\mathsf{k}_{i,j}, \mathsf{k}_{j,i}\}_{j\in[n]})$

$\mathsf{ct}_i \leftarrow \mathsf{Enc}(\mathsf{pp}, \mathsf{sk}_i, x_i)$

For $j \in [n]$:

$\quad \mathsf{k}_{i,j,\ell} := \mathsf{H}_1(\mathsf{k}_{i,j} \| i \| j \| \ell)$

$\quad \mathsf{k}_{j,i,\ell} := \mathsf{H}_1(\mathsf{k}_{j,i} \| j \| i \| \ell)$

$\mathsf{K}_{i,\ell} := \oplus_{j\in[n]} \mathsf{k}_{i,j,\ell}$

$r_i \leftarrow \{0,1\}^\lambda$; $\mathsf{ct}'_i := \mathsf{ct}_i \oplus \mathsf{H}_2(\mathsf{K}_{i,\ell} \| r_i)$

Return $(\mathsf{ct}'_i, r_i, \{\mathsf{k}_{j,i,\ell}\}_{j\in[n]})$

$\underline{\mathsf{KeyDerShare}'(\mathsf{pp}, \mathsf{sk}'_i, f):}$

Parse $\mathsf{sk}'_i = (\mathsf{sk}_i, \{\mathsf{k}_{i,j}, \mathsf{k}_{j,i}\}_{j\in[n]})$

Return $\mathsf{sk}'_{i,f} \leftarrow \mathsf{KeyDerShare}(\mathsf{pp}, \mathsf{sk}'_i, f)$

$\underline{\mathsf{KeyDerComb}'(\mathsf{pp}, \{\mathsf{sk}'_{i,f}\}_{i\in[n]}):}$

$\mathsf{sk}_f := \mathsf{KeyDerComb}(\mathsf{pp}, \{\mathsf{sk}_{i,f}\}_{i\in[n]})$

Return sk_f

$\underline{\mathsf{Dec}'(\mathsf{pp}, \mathsf{sk}_f, \mathsf{ct}''_1, \ldots, \mathsf{ct}''_n):}$

Parse $\{\mathsf{ct}''_i = (\mathsf{ct}'_i, r_i, \{\mathsf{k}_{j,i,\ell}\}_{j\in[n]})\}_{i\in[n]}$

For $i \in [n]$:

$\quad \mathsf{K}_{i,\ell} = \oplus_{j\in[n]} \mathsf{k}_{i,j,\ell}$

$\quad \mathsf{ct}_i = \mathsf{ct}'_i \oplus \mathsf{H}_2(\mathsf{K}_{i,\ell} \| r_i)$

Return $\mathsf{Dec}(\mathsf{pp}, \mathsf{sk}_f, \mathsf{ct}_1, \ldots, \mathsf{ct}_n)$.

Fig. 6. Compiler from an xx-pos-IND DMCFE DMCFE with labels into an xx-any-IND DMCFE DMCFE' with labels, where $\mathsf{H}_1 : \{0,1\}^* \to \{0,1\}^\lambda$ and $\mathsf{H}_2 : \{0,1\}^* \to \{0,1\}^{|\mathsf{ct}_i|}$ are two hash functions modeled as random oracles in the security proof.

5 Security of the MCFE from Abdalla et al. against Adaptive Corruptions

In this section, we prove that the MIFE scheme by Abdalla et al. [ACF+18] is also secure against adaptive corruptions, when their unique encryption and secret key is split into individual secret keys for each party in a natural way[8], as described in Fig. 7 and Fig. 9.

For simplicity, we focus here on the bounded-norm MIFE case since the construction over $\mathbb{Z}_L$ can be easily adapted from it. Towards this goal, Section 5.1 first recalls the definition of FE with two-step decryption and linear encryption. Next, Section 5.2 recalls the other building block, an *sta-one-IND-secure* MCFE scheme for $\mathcal{F}_\rho, \rho = (\mathbb{Z}_L, n, m, L, L)$. Finally, Section 5.3 recalls the MCFE construction from [ACF+18].

5.1 Inner-Product FE with Two-Step Decryption and Linear Encryption

The [ACF+18] construction extends a one-time secure MIFE scheme over $\mathbb{Z}_L$ to a many-time secure MIFE scheme over $\mathbb{Z}$. This extension relies on a single-input FE scheme for $\mathcal{F}_\rho, \rho = (\mathbb{Z}, 1, m, X, Y)$ satisfying two properties, called *two-step decryption* and *linear encryption* [ACF+18]. As indicated in [ACF+18], the *two-step decryption* property informally says that the FE decryption algorithm can be broken in two steps: one step that uses the secret key to return an encoding of the result and the other step that returns the actual result $\langle \boldsymbol{x}, \boldsymbol{y} \rangle$ as long as the bounds $||\boldsymbol{x}||_\infty < X, ||\boldsymbol{y}||_\infty < Y$ hold. The *linear encryption* property, on the other hand, informally states that the FE encryption algorithm is additively homomorphic. We now recall these definitions more formally.

[8] Note that the schemes in [ACF+18] were presented as a MIFE scheme with a unique encryption and secret key. It is however straightforward to split the encryption key and secret key into a key sk_i for each party.

Definition 5.1 (Two-step decryption [ACF+18]). *A secret-key FE scheme* $\mathsf{FE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{KeyDer}, \mathsf{Enc}, \mathsf{Dec})$ *for the function ensemble* $\mathcal{F}_\rho, \rho = (\mathbb{Z}, 1, m, X, Y)$ *satisfies the* two-step decryption *property if it admits PPT algorithms* $\mathsf{Setup}^\star$, $\mathsf{Dec}_1, \mathsf{Dec}_2$ *and an encoding function* $\mathcal{E}$ *such that:*

1. *For all* $\lambda \in \mathbb{N}, \mathsf{Setup}^\star(1^\lambda, 1^n)$ *outputs* pp *where* pp *includes* $\rho = (\mathbb{Z}, 1, m, X, Y)$ *and a bound* $B \in \mathbb{N}$*, as well as the description of a group* $\mathbb{G}$ *(with group law* $\circ$*) of order* $L > 2 \cdot n \cdot m \cdot X \cdot Y$*, which defines the encoding function* $\mathcal{E} : \mathbb{Z}_L \times \mathbb{Z} \to \mathbb{G}$*.*
2. *For all* $\mathsf{msk} \leftarrow \mathsf{KeyGen}(\mathsf{pp}), \boldsymbol{x} \in \mathbb{Z}^m, \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pp}, \mathsf{msk}, \boldsymbol{x}), \boldsymbol{y} \in \mathbb{Z}^m$*, and* $\mathsf{sk} \leftarrow \mathsf{KeyDer}(\mathsf{msk}, \boldsymbol{y})$*, we have*

$$\mathsf{Dec}_1(\mathsf{pp}, \mathsf{sk}, \mathsf{ct}) = \mathcal{E}(\langle \boldsymbol{x}, \boldsymbol{y} \rangle \bmod L, \mathsf{noise}) \ ,$$

   for some $\mathsf{noise} \in \mathbb{N}$ *that depends on* ct *and* sk*. Furthermore, it holds that* $Pr[\mathsf{noise} < B] = 1 - \mathrm{negl}(\lambda)$*, where the probability is taken over the random coins of* KeyGen *and* KeyDer*. Note that there is no restriction on the norm of* $\langle \boldsymbol{x}, \boldsymbol{y} \rangle$ *here.*
3. *Given any* $\gamma \in \mathbb{Z}_L$*, and* pp*, one can efficiently compute* $\mathcal{E}(\gamma, 0)$*.*
4. *The encoding* $\mathcal{E}$ *is linear, that is: for all* $\gamma, \gamma' \in \mathbb{Z}_L, \mathsf{noise}, \mathsf{noise}' \in \mathbb{Z}$*, we have*

$$\mathcal{E}(\gamma, \mathsf{noise}) \circ \mathcal{E}(\gamma', \mathsf{noise}') = \mathcal{E}(\gamma + \gamma' \bmod L, \mathsf{noise} + \mathsf{noise}') \ .$$

5. *For all* $\gamma < 2 \cdot n \cdot m \cdot X \cdot Y$*, and* $\mathsf{noise} < n \cdot B, \mathsf{Dec}_2(\mathsf{pp}, \mathcal{E}(\gamma, \mathsf{noise})) = \gamma$*.*

Definition 5.2 (Linear encryption [ACF+18]). *A secret-key FE scheme* $\mathsf{FE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{KeyDer}, \mathsf{Enc}, \mathsf{Dec})$ *is said to satisfy the linear encryption property if there exists a deterministic algorithm* Add *that takes as input a ciphertext and a message, such that for all* $\boldsymbol{x}, \boldsymbol{x}' \in \mathbb{Z}^m$*, the following are identically distributed:*

$$\mathsf{Add}(\mathsf{Enc}(\mathsf{pp}, \mathsf{msk}, \boldsymbol{x}), \boldsymbol{x}'), \quad \textit{and} \quad \mathsf{Enc}\big(\mathsf{pp}, \mathsf{msk}, (\boldsymbol{x} + \boldsymbol{x}' \bmod L)\big) \ .$$

Recall that the value $L \in \mathbb{N}$ *is defined as part of the output of the algorithm* $\mathsf{Setup}^\star$ *(see the two-step decryption property above).*

5.2 One-Time Inner-Product MCFE over $\mathbb{Z}_L$

We recap the one-time secure scheme provided by Abdalla et al. [ACF+18] in Fig. 7, to which we made the following modifications. First, our description does not need a setup procedure $\mathsf{Setup}^{\mathsf{ot}}$, which now simply defines (n, m, L). Second, the steps of the original $\mathsf{Setup}^{\mathsf{ot}}$ in Abdalla et al. [ACF+18] are now defined in the $\mathsf{KeyGen}^{\mathsf{ot}}$ procedure. When doing so, we also split their unique secret key into individual secret keys for each party. Since these modifications do not impact the correctness of the scheme, we refer to [ACF+18] for a proof of correctness. As for its security with respect to adaptive corruptions, we need to modify the proof of Abdalla et al. [ACF+18] to account for corruption queries.

Theorem 5.3. *The* $\mathsf{MCFE}^{\mathsf{ot}}$ *scheme in Fig. 7 is* adt-one-IND *secure. Namely, for any adversary* $\mathcal{A}$*,* $\mathsf{Adv}^{\text{adt-one-IND}}_{\mathsf{MCFE}^{\mathsf{ot}}, \mathcal{A}}(\lambda) = 0$

Proof. Let $\mathcal{A}$ be an adversary against the adt-one-IND security of the $\mathsf{MCFE}^{\mathsf{ot}}$ scheme with advantage $\mathsf{Adv}^{\text{adt-one-IND}}_{\mathsf{MCFE}^{\mathsf{ot}}, \mathcal{A}}(\lambda)$. Let sta-one-sel-IND$^{\mathsf{MCFE}^{\mathsf{ot}}}_\beta(\lambda, n, \mathcal{B})$ be a variant of the sta-one-IND$^{\mathsf{MCFE}^{\mathsf{ot}}}_\beta(\lambda, n, \mathcal{B})$ experiment in which the selective adversary $\mathcal{B}$ additionally specifies the encryption challenges $\{\boldsymbol{x}_i^b\}_{i \in [n], b \in \{0,1\}}$ together with the corrupted set at the beginning of the experiment. (Recall that there is a single challenge per slot.)

We use complexity leveraging to transform $\mathcal{A}$ into a selective adversary $\mathcal{B}$ such that:

$$\mathsf{Adv}^{\text{adt-one-IND}}_{\mathsf{MCFE}^{\mathsf{ot}}, \mathcal{A}}(\lambda) \leq 2^{-n} \cdot (2X)^{-2nm} \cdot \mathsf{Adv}^{\text{sta-one-sel-IND}}_{\mathsf{MCFE}^{\mathsf{ot}}, \mathcal{B}}(\lambda) \ .$$

After adversary $\mathcal{B}$ made its guesses $\{\boldsymbol{x}_i^b\}_{i \in [n], b \in \{0,1\}}$ and determined the set of corrupted parties, it simulates $\mathcal{A}$'s experiment using its own static and selective experiment. When $\mathcal{B}$ receives a challenge or corruption query

Define $\mathsf{pp}_{\mathsf{ot}} = (n, m, L)$

$\underline{\mathsf{KeyGen}^{\mathsf{ot}}(\mathsf{pp}_{\mathsf{ot}}):}$

$\{\boldsymbol{u}_i\}_{i\in[n]} \leftarrow (\mathbb{Z}_L^m)^n$

Return $\mathsf{msk} := \{\mathsf{msk}_i\}_{i\in[n]} = \{\boldsymbol{u}_i\}_{i\in[n]}$

$\underline{\mathsf{Enc}^{\mathsf{ot}}(\mathsf{pp}_{\mathsf{ot}}, \mathsf{msk}_i, \boldsymbol{x}_i):}$

Parse $\mathsf{msk}_i = \boldsymbol{u}_i$

Return $\mathsf{ct}_i := \boldsymbol{u}_i + \boldsymbol{x}_i \bmod L$

$\underline{\mathsf{KeyDer}^{\mathsf{ot}}(\mathsf{pp}_{\mathsf{ot}}, \mathsf{msk}, \boldsymbol{y}):}$

Parse $\mathsf{msk} = \{\boldsymbol{u}_i\}_{i\in[n]}, \boldsymbol{y} = (\boldsymbol{y}_1, \ldots, \boldsymbol{y}_n)$

Return $\mathsf{dk}_{\boldsymbol{y}} := \sum_{i\in[n]} \langle \boldsymbol{u}_i, \boldsymbol{y}_i \rangle$

$\underline{\mathsf{Dec}^{\mathsf{ot}}(\mathsf{pp}_{\mathsf{ot}}, \mathsf{dk}_{\boldsymbol{y}}, \boldsymbol{y}, \{\mathsf{ct}_i\}_{i\in[n]}):}$

Parse $\boldsymbol{y} = (\boldsymbol{y}_1, \ldots, \boldsymbol{y}_n)$

Return $\sum_{i\in[n]} \langle \mathsf{ct}_i, \boldsymbol{y}_i \rangle - \mathsf{dk}_{\boldsymbol{y}} \bmod L$

Fig. 7. One-Time Inner-Product MCFE over $\mathbb{Z}_L$ (for $\mathcal{F}_{L,n}^m$)

$\mathcal{H}_\beta(1^\lambda, \mathcal{B})$

$(\mathcal{CS}, \{\boldsymbol{x}_i^b\}_{i\in[n], b\in\{0,1\}}) \leftarrow \mathcal{B}(1^\lambda, 1^n)$

For $i \in [n]$,

$\quad \boldsymbol{u}_i \leftarrow \mathbb{Z}_L^m; \mathsf{ct}_i \leftarrow \boldsymbol{u}_i$

$\alpha \leftarrow \mathcal{B}^{\mathcal{O}_{\mathsf{K}}(\cdot)}(\{\boldsymbol{u}_i\}_{i\in\mathcal{CS}}, \{\mathsf{ct}_i\}_{i\in[n]})$

Output α

$\mathcal{O}_{\mathsf{K}}(i, \boldsymbol{y})$

Parse $\boldsymbol{y} = (\boldsymbol{y}_1, \ldots, \boldsymbol{y}_n)$

$\mathsf{sk}_{\boldsymbol{y}} = \sum_{i\in[n]} \langle \boldsymbol{u}_i - \boldsymbol{x}_i^\beta, \boldsymbol{y}_i \rangle$

Return $\mathsf{sk}_{\boldsymbol{y}}$

Fig. 8. Hybrid experiments for the proof of Theorem 5.3.

from $\mathcal{A}$, it checks if the guess was successful: if it was, it continues simulating $\mathcal{A}$'s experiment, otherwise, it returns 0. When the guess is successful, $\mathcal{B}$ perfectly simulates $\mathcal{A}$'s view.

Hence, to prove that $\mathsf{MCFE}^{\mathsf{ot}}$ satisfies perfect adt-one-IND security, we just need to prove that it satisfies perfect sta-one-sel-IND security. In order to prove $\mathsf{MCFE}^{\mathsf{ot}}$ satisfies perfect sta-one-sel-IND security (i.e., $\mathsf{Adv}^{\text{sta-one-sel-IND}}_{\mathsf{MCFE}^{\mathsf{ot}},\mathcal{B}}(\lambda) = 0$), we introduce hybrid games $\mathcal{H}_\beta(1^\lambda, \mathcal{B})$, described in Fig. 8.

We prove that for all $\beta \in \{0, 1\}$, the hybrid $\mathcal{H}_\beta(1^\lambda, \mathcal{B})$ is identical to the experiment $\text{sta-one-sel-IND}_\beta^{\mathsf{MCFE}^{\mathsf{ot}}}(\lambda, n, \mathcal{B})$. This can be seen by using the fact that, in the selective security game, all $\{\boldsymbol{x}_i^\beta \in \mathbb{Z}^m\}_{i\in[n]}$ have identical distributions: $\{\boldsymbol{u}_i \bmod L\}_{i\in[n]}$ and $\{\boldsymbol{u}_i - \boldsymbol{x}_i^\beta \bmod L\}$, with $\boldsymbol{u}_i \leftarrow_R \mathbb{Z}_L^m$. This also holds for the corrupted positions $i \in \mathcal{CS}$, because in this case it holds that $\boldsymbol{x}_i^0 = \boldsymbol{x}_i^1$.

Finally, we show that $\mathcal{B}$'s view in $\mathcal{H}_\beta(1^\lambda, \mathcal{B})$ is independent of β. Indeed, the only information about β that leaks in the experiment is $\langle \boldsymbol{x}_i^\beta, \boldsymbol{y}_i \rangle$, which is independent of β by the definition of the security game. □

5.3 Inner-Product MCFE over $\mathbb{Z}$

In Fig. 9, we recall the construction of [ACF+18] of a pos-IND-secure scheme $\mathsf{MCFE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{KeyDer}, \mathsf{Enc}, \mathsf{Dec})$ from the (one-IND-secure) MCFE scheme $\mathsf{MCFE}^{\mathsf{ot}} = (\mathsf{KeyGen}^{\mathsf{ot}}, \mathsf{KeyDer}^{\mathsf{ot}}, \mathsf{Enc}^{\mathsf{ot}}, \mathsf{Dec}^{\mathsf{ot}})$ described in Section 5.2 and from any any-IND-secure scheme $\mathsf{FE} = (\mathsf{Setup}^{\mathsf{si}}, \mathsf{KeyGen}^{\mathsf{si}}, \mathsf{KeyDer}^{\mathsf{si}}, \mathsf{Enc}^{\mathsf{si}}, \mathsf{Dec}^{\mathsf{si}})$ for a single input. As for the one-time scheme in Section 5.2, we also modified the KeyGen procedure in [ACF+18] in order to split their unique secret key into individual secret keys for each party. Since these modifications do not impact the correctness of the scheme, we refer to [ACF+18] for a proof of the latter. In the following, we show that this construction allows for adaptive corruption.

Lemma 5.4. *Assume that the single-input scheme* FE *is any-IND-secure and that the multi-client scheme* $\mathsf{MCFE}^{\mathsf{ot}}$ *is adt-one-IND-secure. Then the multi-client scheme* MCFE *is adt-pos-IND-secure. Namely, for any*

$\underline{\mathsf{Setup}(1^\lambda, 1^n) :}$

$\mathsf{pp}_{\mathsf{si}} \leftarrow \mathsf{Setup}^{\mathsf{si}}(1^\lambda, 1^n)$

Set $\mathsf{pp}_{\mathsf{ot}} := (n, m, L)$, with $\rho_{\mathsf{si}} = (\mathbb{Z}, 1, m, 3X, Y)$ and L implicitly defined from $\mathsf{pp}_{\mathsf{si}}$

Return $\mathsf{pp} = (\mathsf{pp}_{\mathsf{si}}, \mathsf{pp}_{\mathsf{ot}})$

$\underline{\mathsf{KeyGen}(\mathsf{pp}) :}$

$\{\boldsymbol{u}_i\}_{i\in[n]} \leftarrow \mathsf{KeyGen}^{\mathsf{ot}}(\mathsf{pp}_{\mathsf{ot}})$

For $i \in [n]$, $\mathsf{msk}_i^{\mathsf{si}} \leftarrow \mathsf{KeyGen}^{\mathsf{si}}(\mathsf{pp}_{\mathsf{si}})$, $\mathsf{sk}_i := (\mathsf{msk}_i^{\mathsf{si}}, \boldsymbol{u}_i)$

Return $\{\mathsf{sk}_i\}_{i\in[n]}$

$\underline{\mathsf{Enc}(\mathsf{pp}, \mathsf{sk}_i, \boldsymbol{x}_i) :}$

Parse $\mathsf{sk}_i = (\mathsf{msk}_i^{\mathsf{si}}, \boldsymbol{u}_i)$ and return $\mathsf{ct}_i := \mathsf{Enc}^{\mathsf{si}}(\mathsf{pp}_{\mathsf{si}}, \mathsf{msk}_i^{\mathsf{si}}, \mathsf{Enc}^{\mathsf{ot}}(\mathsf{pp}_{\mathsf{ot}}, \boldsymbol{u}_i, \boldsymbol{x}_i))$

$\underline{\mathsf{KeyDer}(\mathsf{pp}, \mathsf{msk}, \boldsymbol{y}) :}$

Parse $\mathsf{msk} = \{\mathsf{msk}_i^{\mathsf{si}}, \boldsymbol{u}_i\}_{i\in[n]}, \boldsymbol{y} = (\boldsymbol{y}_1, \ldots, \boldsymbol{y}_n)$

For $i \in [n]$, $\mathsf{sk}_{i,\boldsymbol{y}} \leftarrow \mathsf{KeyDer}^{\mathsf{si}}(\mathsf{pp}_{\mathsf{si}}, \mathsf{msk}_i^{\mathsf{si}}, \boldsymbol{y}_i)$

$\mathsf{dk}_{\boldsymbol{y}} := \mathsf{KeyDer}^{\mathsf{ot}}(\mathsf{pp}_{\mathsf{ot}}, \{\boldsymbol{u}_i\}_{i\in[n]}, \boldsymbol{y})$

Return $\mathsf{sk}_{\boldsymbol{y}} := (\{\mathsf{sk}_{i,\boldsymbol{y}}\}_{i\in[n]}, \mathsf{dk}_{\boldsymbol{y}})$

$\underline{\mathsf{Dec}(\mathsf{pp}, \mathsf{sk}_{\boldsymbol{y}}, \{\mathsf{ct}_i\}_{i\in[n]}) :}$

Parse $\mathsf{sk}_{\boldsymbol{y}} = (\{\mathsf{sk}_{i,\boldsymbol{y}}\}_{i\in[n]}, \mathsf{dk}_{\boldsymbol{y}})$

For $i \in [n]$, $\mathcal{E}(\langle \boldsymbol{u}_i + \boldsymbol{x}_i, \boldsymbol{y}_i \rangle \bmod L, \mathsf{noise}_i) \leftarrow \mathsf{Dec}_1^{\mathsf{si}}(\mathsf{pp}_{\mathsf{si}}, \mathsf{sk}_{i,\boldsymbol{y}}, \mathsf{ct}_i)$

Return $\mathsf{Dec}_2^{\mathsf{si}}(\mathsf{pp}_{\mathsf{si}}, \mathcal{E}(\langle \boldsymbol{u}_1 + \boldsymbol{x}_1, \boldsymbol{y}_1 \rangle \bmod L, \mathsf{noise}_1)) \circ \ldots$
$\circ\, \mathcal{E}(\langle \boldsymbol{u}_n + \boldsymbol{x}_n, \boldsymbol{y}_n \rangle \bmod L, \mathsf{noise}_n) \circ \mathcal{E}(-\mathsf{dk}_{\boldsymbol{y}}, 0))$

Fig. 9. Inner-Product for $\mathcal{F}_\rho, \rho = (\mathbb{Z}, n, m, X, Y)$ built from $\mathsf{MCFE}^{\mathsf{ot}}$ for $\mathcal{F}_{\rho_{\mathsf{ot}}}, \rho_{\mathsf{ot}} = (\mathbb{Z}_L, n, m, L, L)$ and FE for $\mathcal{F}_{\rho_{\mathsf{si}}}, \rho_{\mathsf{si}} = (\mathbb{Z}, 1, m, 3X, Y)$

Game	ct_i^j	justification/remark
G_0	$\mathsf{Enc}(\mathsf{pp}, \mathsf{sk}_i, \boldsymbol{x}_i^{0,j} - \boldsymbol{x}_i^{0,1} + \boldsymbol{x}_i^{0,1})$	
G_1	$\mathsf{Enc}(\mathsf{pp}, \mathsf{sk}_i, \boldsymbol{x}_i^{0,j} - \boldsymbol{x}_i^{0,1} + \boxed{\boldsymbol{x}_i^{1,1}})$	adt-one-IND of $\mathsf{MCFE}^{\mathsf{ot}}$
$\mathsf{G}_{1.k}$	$\mathsf{Enc}(\mathsf{pp}, \mathsf{sk}_i, \boxed{\boldsymbol{x}_i^{1,j} - \boldsymbol{x}_i^{1,1}} + \boldsymbol{x}_i^{1,1})$, for $i \leq k$ $\mathsf{Enc}(\mathsf{pp}, \mathsf{sk}_i, \boldsymbol{x}_i^{0,j} - \boldsymbol{x}_i^{0,1} + \boldsymbol{x}_i^{1,1})$, for $i > k$	any-IND of FE
G_2	$\mathsf{Enc}(\mathsf{pp}', \mathsf{sk}_i, \boxed{\boldsymbol{x}_i^{1,j}})$	$\mathsf{G}_2 = \mathsf{G}_{1,n}$

Fig. 10. Overview of the games to prove the security of the MCFE scheme.

PPT adversary $\mathcal{A}$, there exist PPT adversaries $\mathcal{B}$ and $\mathcal{B}'$ such that

$$\mathsf{Adv}^{\text{adt-pos-IND}}_{\mathsf{MCFE},\mathcal{A}}(\lambda, n) \leq \mathsf{Adv}^{\text{adt-one-IND}}_{\mathsf{MCFE}^{\mathsf{ot}},\mathcal{B}}(\lambda, n) + n \cdot \mathsf{Adv}^{\text{any-IND}}_{\mathsf{FE},\mathcal{B}'}(\lambda, n).$$

Proof. To prove the security of the multi-client inner-product functional encryption scheme, we define a sequence of games, where G_0 is the adt-pos-IND$_0^{\mathsf{MCFE}}(\lambda, n, \mathcal{A})$ game and G_2 the adt-pos-IND$_1^{\mathsf{MCFE}}(\lambda, n, \mathcal{A})$ game. A description of all the different games can be found in Fig. 10. We denote the winning probability of an adversary $\mathcal{A}$ in a game G_i as $\mathsf{Win}^{\mathsf{G}_i}_{\mathcal{A}}(\lambda, n)$, which is $Pr[\mathsf{G}_i(\lambda, n, \mathcal{A}) = 1]$. The probability is taken over the random coins of G_i and $\mathcal{A}$. The encryption query j on the i-th slot is denoted as $(\boldsymbol{x}_i^{0,j}, \boldsymbol{x}_i^{1,j})$.

We start our proof by considering the games G_0 and G_1

Lemma 5.5. *For any PPT $\mathcal{A}$, there exists a PPT adversary $\mathcal{B}$ such that*

$$|\mathsf{Win}^{\mathsf{G}_0}_{\mathcal{A}}(\lambda, n) - \mathsf{Win}^{\mathsf{G}_1}_{\mathcal{A}}(\lambda, n)| \leq \mathsf{Adv}^{\text{adt-one-IND}}_{\mathsf{MCFE}^{\mathsf{ot}},\mathcal{B}}(\lambda, n) \ .$$

Proof. Compared to G_0, G_1 replaces the encryptions of $\boldsymbol{x}_i^{0,j} - \boldsymbol{x}_i^{0,1} + \boldsymbol{x}_i^{0,1}$ with the encryptions of $\boldsymbol{x}_i^{0,j} - \boldsymbol{x}_i^{0,1} + \boldsymbol{x}_i^{1,1}$ for all of the slots i under adaptive corruptions. This mirrors directly the distribution of the challenge ciphertexts in G_β.

The adversary $\mathcal{B}$ simulates G_β to $\mathcal{A}$ using the adt-one-IND$_\beta^{\mathsf{MCFE}}$ experiment. In the beginning $\mathcal{B}$ generates the parameters $\mathsf{pp} = (\mathsf{pp}_{\mathsf{si}}, \mathsf{pp}_{\mathsf{ot}}) \leftarrow \mathsf{Setup}(1^\lambda, 1^n)$ and the keys $\mathsf{msk}_i^{\mathsf{si}} \leftarrow \mathsf{KeyGen}(\mathsf{pp}_{\mathsf{si}})$ for all the positions $i \in [n]$. Whenever $\mathcal{A}$ asks a query $\mathsf{QKeyD}'(\boldsymbol{y} = (\boldsymbol{y}_1, \ldots, \boldsymbol{y}_n))$, $\mathcal{B}$ uses its own key derivation oracle to get $\mathsf{dk}_{\boldsymbol{y}} = \sum_{i \in [n]} \langle \boldsymbol{u}_i, \boldsymbol{y}_i \rangle$ and computes the keys $\mathsf{sk}_{i,\boldsymbol{y}} \leftarrow \mathsf{KeyDer}^{\mathsf{si}}(\mathsf{pp}_{\mathsf{si}}, \mathsf{msk}_i^{\mathsf{si}}, \boldsymbol{y}_i)$ for all the positions $i \in [n]$ on its own and sends them to $\mathcal{A}$.

For each position $i \in [n]$, the first encryption query $\mathsf{QEnc}'(i, \boldsymbol{x}_i^{0,1}, \boldsymbol{x}_i^{1,1})$ by $\mathcal{A}$ gets forwarded to the challenger. $\mathcal{B}$ receives $\mathsf{ct}_{i,\mathsf{ot}} = \boldsymbol{u}_i + \boldsymbol{x}_i^{\beta,1}$ as an answer, computes $\mathsf{ct}_i^1 = \mathsf{Enc}^{\mathsf{si}}(\mathsf{pp}_{\mathsf{si}}, \mathsf{msk}_i^{\mathsf{si}}, \boldsymbol{u}_i + \boldsymbol{x}_i^{\beta,1})$, and returns it to $\mathcal{A}$. For all further queries ($j > 1$), $\mathcal{B}$ produces ct_i^j by encrypting $(\boldsymbol{x}_i^{0,j} - \boldsymbol{x}_i^{0,1} + \mathsf{ct}_{i,\mathsf{ot}}) \bmod L$.

When $\mathcal{A}$ asks a query $\mathsf{QCor}'(i)$, it is necessary that $\boldsymbol{x}_i^{0,j} = \boldsymbol{x}_i^{1,j}$ holds for all the corruption queries that $\mathcal{A}$ has asked before. In this case, $\mathcal{B}$ computes $\boldsymbol{u}_i = \mathsf{ct}_{i,\mathsf{ot}} - \boldsymbol{x}_i^{0,1}$ and sends $(\mathsf{msk}_i^{\mathsf{si}}, \mathsf{mpk}_i^{\mathsf{si}}, \boldsymbol{u}_i)$ to $\mathcal{A}$.

Finally, $\mathcal{B}$ outputs 1, if and only if $\mathcal{A}$ outputs 1. By the reasoning above, we can conclude that:

$$|\mathsf{Win}^{\mathsf{G}_0}_{\mathcal{A}}(\lambda, n) - \mathsf{Win}^{\mathsf{G}_1}_{\mathcal{A}}(\lambda, n)| \leq \mathsf{Adv}^{\text{adt-one-IND}}_{\mathsf{MCFE}^{\mathsf{ot}},\mathcal{B}}(\lambda, n) \ .$$

□

In the next step we consider game G_2. In this game, we change the encryption from $\mathsf{Enc}^{\mathsf{si}}(\mathsf{pp}_{\mathsf{si}}, \mathsf{msk}_i^{\mathsf{si}}, \boldsymbol{x}_i^{0,j} - \boldsymbol{x}_i^{0,1} + \boldsymbol{u}_i + \boldsymbol{x}_i^{1,1})$ to $\mathsf{Enc}^{\mathsf{si}}(\mathsf{pp}_{\mathsf{si}}, \mathsf{msk}_i^{\mathsf{si}}, \boldsymbol{x}_i^{1,j} - \boldsymbol{x}_i^{1,1} + \boldsymbol{u}_i + \boldsymbol{x}_i^{1,1})$ for all slots i and all queries j.

To prove that G_1 is indistinguishable from G_2 we need to apply a hybrid argument over the n slots, using the security of the single input FE scheme.

Using the definition of the games in Fig. 11, we can see that

$$|\mathsf{Win}^{\mathsf{G}_1}_{\mathcal{A}}(\lambda, n) - \mathsf{Win}^{\mathsf{G}_2}_{\mathcal{A}}(\lambda, n)| = \sum_{k=1}^{n} |\mathsf{Win}^{\mathsf{G}_{1.k-1}}_{\mathcal{A}}(\lambda, n) - \mathsf{Win}^{\mathsf{G}_{1.k}}_{\mathcal{A}}(\lambda, n)| \ ,$$

where G_1 corresponds to game $\mathsf{G}_{1.0}$ and whereas G_2 is identical to game $\mathsf{G}_{1.n}$.

Now, we can bound the difference between each consecutive pair of games for every k:

Lemma 5.6. *For every $k \in [n]$, there exists a PPT adversary $\mathcal{B}_k$ against the* any-IND *security of the single-input scheme* FE *such that*

$$|\mathsf{Win}_{\mathcal{A}}^{\mathsf{G}_{1.k-1}}(\lambda, n) - \mathsf{Win}_{\mathcal{A}}^{\mathsf{G}_{1.k}}(\lambda, n)| \leq \mathsf{Adv}_{\mathsf{FE},\mathcal{B}_k}^{\text{any-IND}}(\lambda, n) \ .$$

Proof. $\mathsf{G}_{1.k}$ replaces the encryption of $\boldsymbol{x}_i^{0,j} - \boldsymbol{x}_i^{0,1} + \boldsymbol{x}_i^{1,1}$ with encryptions of $\boldsymbol{x}_i^{1,j} - \boldsymbol{x}_i^{1,1} + \boldsymbol{x}_i^{1,1}$ in all slots, for $i \leq k$. As already described in the preliminaries, it must hold that $\langle \boldsymbol{x}_i^{0,j} - \boldsymbol{x}_i^{0,1}, \boldsymbol{y}_i \rangle = \langle \boldsymbol{x}_i^{1,j} - \boldsymbol{x}_i^{1,1}, \boldsymbol{y}_i \rangle$ for all queries. Hence $\langle \boldsymbol{x}_i^{0,j} - \boldsymbol{x}_i^{0,1} + \boldsymbol{x}_i^{1,1}, \boldsymbol{y}_i \rangle = \langle \boldsymbol{x}_i^{1,j} - \boldsymbol{x}_i^{1,1} + \boldsymbol{x}_i^{1,1}, \boldsymbol{y}_i \rangle$, and since $\|\boldsymbol{x}_i^{0,j} - \boldsymbol{x}_i^{0,1} + \boldsymbol{x}_i^{1,1}\|_\infty < 3X$ and $\|\boldsymbol{x}_i^{1,j} - \boldsymbol{x}_i^{1,1} + \boldsymbol{x}_i^{1,1}\|_\infty < 3X$, using the linear encryption property, we can reduce the difference in the winning probability of an adversary $\mathcal{A}$ in games $\mathsf{G}_{1.k-1}$ and $\mathsf{G}_{1.k}$ to the any-IND security of the single-input scheme FE.

More precisely, we build an adversary $\mathcal{B}_k$ that simulates $\mathsf{G}_{1.k-1+\beta}$ to $\mathcal{A}$ when interacting with the underlying $\text{any-IND}_\beta^{\mathsf{FE}}$ experiment. In the beginning of the reduction, $\mathcal{B}_k$ receives the public parameters from the experiment. The received key from the challenge is set to be $\mathsf{mpk}_k^{\mathsf{si}}$, corresponding to the k-th encryption instance. In the next step, $\mathcal{B}_k$ randomly chooses $\boldsymbol{u}_i \in \mathbb{Z}_L^m$ for all $i \in [n]$ and runs the KeyGen procedure to get $\mathsf{msk}_i^{\mathsf{si}}$ for all $i \neq k$.

Whenever $\mathcal{A}$ asks a query $\mathsf{QKeyD}'(\boldsymbol{y})$, $\mathcal{B}_k$ computes $\mathsf{dk}_y = \sum_{i\in[n]} \langle \boldsymbol{u}_i, \boldsymbol{y}_i \rangle$ on its own and generates $\mathsf{sk}_{i,\boldsymbol{y}} \leftarrow \mathsf{KeyGen}^{\mathsf{si}}(\mathsf{pp}_{\mathsf{si}}, \mathsf{msk}_i^{\mathsf{si}}, \boldsymbol{y}_i)$ for all $i \neq k$. To get the functional key $\mathsf{sk}_{k,\boldsymbol{y}}$, $\mathcal{B}_k$ queries its own key derivation oracle on $\boldsymbol{y}_i$ and outputs $(\{\mathsf{sk}_{i,\boldsymbol{y}}\}_{i\in[n]}, \mathsf{dk}_y)$ to $\mathcal{A}$.

For the encryption queries $\mathsf{QEnc}(i, \boldsymbol{x}_i^{0,j}, \boldsymbol{x}_i^{1,j})$, $\mathcal{B}_k$ proceeds in the following way:

- If $i < k$ it computes $\mathsf{Enc}^{\mathsf{si}}(\mathsf{pp}_{\mathsf{si}}, \mathsf{msk}_i^{\mathsf{si}}, \boldsymbol{u}_i + \boldsymbol{x}_i^{1,j})$.
- If $i > k$ it computes $\mathsf{Enc}^{\mathsf{si}}(\mathsf{pp}_{\mathsf{si}}, \mathsf{msk}_i^{\mathsf{si}}, \boldsymbol{x}_i^{0,j} - \boldsymbol{x}_i^{0,1} + \boldsymbol{u}_i + \boldsymbol{x}_i^{1,1})$.
- If $i = k$, $\mathcal{B}_k$ queries the encryption oracle on input $(\boldsymbol{x}_k^{0,j} - \boldsymbol{x}_k^{0,1} + \boldsymbol{x}_k^{1,1}, \boldsymbol{x}_k^{1,j} - \boldsymbol{x}_k^{1,1} + \boldsymbol{x}_k^{1,1})$ to get back the ciphertext $\mathsf{ct}_*^j := \mathsf{Enc}^{\mathsf{si}}(\mathsf{pp}_{\mathsf{si}}, \mathsf{msk}_k^{\mathsf{si}}, \boldsymbol{x}_k^{\beta,j} - \boldsymbol{x}_k^{\beta,1} + \boldsymbol{x}_k^{1,1})$ from the $\text{any-IND}_\beta^{\mathsf{FE}}$ experiment.[9] Then, $\mathcal{B}_k$ computes the ciphertext $\mathsf{ct}_k^j := \mathsf{Add}(\mathsf{ct}_*^j, \boldsymbol{u}_k)$ and forwards it to $\mathcal{A}$.

As in the security proof of the MIFE scheme in [ACF+18], we remark that by the two-step property Definition 5.2, ct_k^j is identically distributed to $\mathsf{Enc}^{\mathsf{si}}\big(\mathsf{pp}_{\mathsf{si}}, \mathsf{msk}_k^{\mathsf{si}}, \boldsymbol{x}_k^{j,\beta} - \boldsymbol{x}_k^{1,\beta} + \boldsymbol{x}_k^{1,1} + \boldsymbol{u}_k \mod L\big)$, which is itself equal to $\mathsf{Enc}^{\mathsf{si}}\big(\mathsf{pp}_{\mathsf{si}}, \mathsf{msk}_k^{\mathsf{si}}, \mathsf{Enc}^{\mathsf{ot}}(\boldsymbol{x}_k^{j,\beta} - \boldsymbol{x}_k^{1,\beta} + \boldsymbol{x}_k^{1,1})\big)$.

In the case that the adversary $\mathcal{A}$ asks a corruption query $\mathsf{QCor}'(k)$ for position k at any time, the adversary $\mathcal{B}_k$ directly outputs a random value $\alpha \leftarrow \{0,1\}$. This is due to the fact that, if position k is corrupted, then the games $\mathsf{G}_{1.k-1}$ and $\mathsf{G}_{1.k}$ are identical given that $\boldsymbol{x}_k^{1,0} = \boldsymbol{x}_k^{1,1}$. This results in an advantage equal to 0 and Lemma 5.6 trivially holds in this case.

In the case that the adversary $\mathcal{A}$ asks a corruption query $\mathsf{QCor}'(i)$ for $i \neq k$, $\mathcal{B}_k$ simply returns $(\mathsf{msk}_i^{\mathsf{si}}, \boldsymbol{u}_i)$ to $\mathcal{A}$.

This covers the simulation of the game $\mathsf{G}_{1.k-1+\beta}$. Finally, $\mathcal{B}_k$ outputs the same bit β' returned by $\mathcal{A}$:

$$|\mathsf{Win}_{\mathcal{A}}^{\mathsf{G}_{1.k-1}}(\lambda, n) - \mathsf{Win}_{\mathcal{A}}^{\mathsf{G}_{1.k}}(\lambda, n)| \leq \mathsf{Adv}_{\mathsf{FE},\mathcal{B}_k}^{\text{any-IND}}(\lambda, n).$$

□

The proof of theorem follows by combining the statements in Lemma 5.5 and Lemma 5.6 and noticing that the adversary $\mathcal{B}'$ in the theorem statement can be obtained by picking $i \in [n]$ and running $\mathcal{B}_i$. The standard details are omitted here. □

[9] As in [ACF+18], note that these vectors have norm less than $3X$, and as such, are a valid input to the encryption oracle. Furthermore, these queries are allowed, since as explained at the beginning of the proof: it holds that $\langle \boldsymbol{x}_i^{0,j} - \boldsymbol{x}_i^{0,1}, \boldsymbol{y}_i \rangle = \langle \boldsymbol{x}_i^{1,j} - \boldsymbol{x}_i^{1,1}, \boldsymbol{y}_i \rangle$.

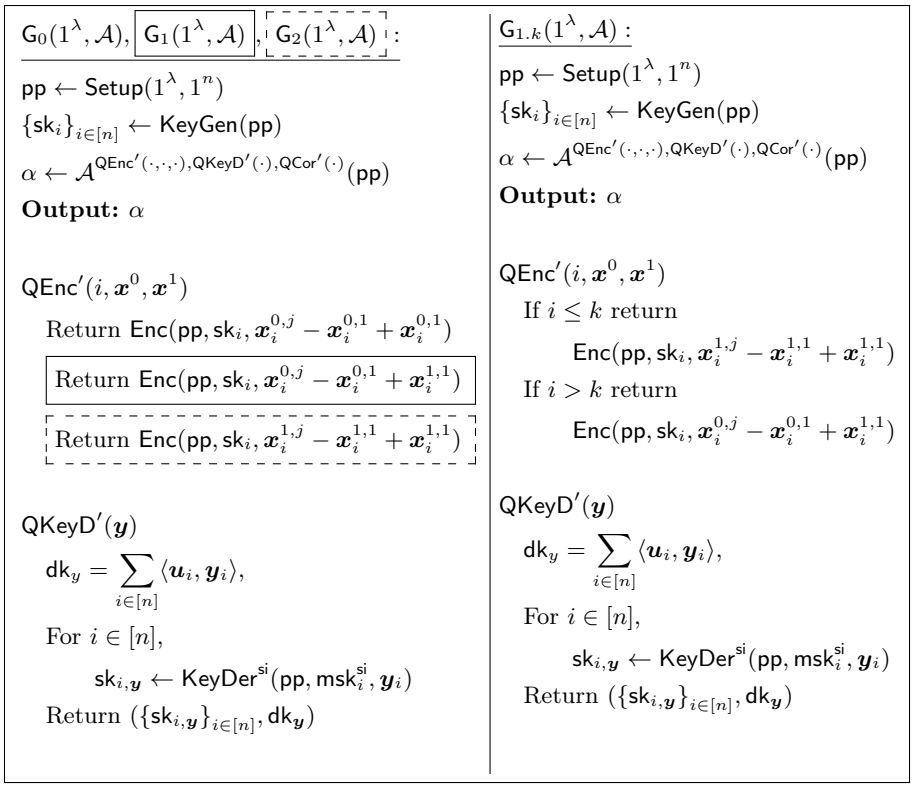

Fig. 11. A more detailed description of how the games work.

Acknowledgments. This work was supported in part by the European Union's Horizon 2020 Research and Innovation Programme under grant agreement 780108 (FENTEC), by the ERC Project aSCEND (H2020 639554), by the French *Programme d'Investissement d'Avenir* under national project RISQ P141580, and by the French FUI project ANBLIC.

References

ABDP15. M. Abdalla, F. Bourse, A. De Caro, and D. Pointcheval. Simple functional encryption schemes for inner products. In *PKC 2015*, *LNCS* 9020, pages 733–751. Springer, Heidelberg, March / April 2015.

ACF+18. M. Abdalla, D. Catalano, D. Fiore, R. Gay, and B. Ursu. Multi-input functional encryption for inner products: Function-hiding realizations and constructions without pairings. In *CRYPTO 2018, Part I*, *LNCS* 10991, pages 597–627. Springer, Heidelberg, August 2018.

AGRW17. M. Abdalla, R. Gay, M. Raykova, and H. Wee. Multi-input inner-product functional encryption from pairings. In *EUROCRYPT 2017, Part I*, *LNCS* 10210, pages 601–626. Springer, Heidelberg, April / May 2017.

AJ15. P. Ananth and A. Jain. Indistinguishability obfuscation from compact functional encryption. In *CRYPTO 2015, Part I*, *LNCS* 9215, pages 308–326. Springer, Heidelberg, August 2015.

ALS16. S. Agrawal, B. Libert, and D. Stehlé. Fully secure functional encryption for inner products, from standard assumptions. In *CRYPTO 2016, Part III*, *LNCS* 9816, pages 333–362. Springer, Heidelberg, August 2016.

BCFG17. C. E. Z. Baltico, D. Catalano, D. Fiore, and R. Gay. Practical functional encryption for quadratic functions with applications to predicate encryption. In *CRYPTO 2017, Part I*, *LNCS* 10401, pages 67–98. Springer, Heidelberg, August 2017.

BDJR97. M. Bellare, A. Desai, E. Jokipii, and P. Rogaway. A concrete security treatment of symmetric encryption. In *38th FOCS*, pages 394–403. IEEE Computer Society Press, October 1997.

BGJS15. S. Badrinarayanan, D. Gupta, A. Jain, and A. Sahai. Multi-input functional encryption for unbounded arity functions. In *ASIACRYPT 2015, Part I*, *LNCS* 9452, pages 27–51. Springer, Heidelberg, November / December 2015.

BJK15. A. Bishop, A. Jain, and L. Kowalczyk. Function-hiding inner product encryption. In *ASIACRYPT 2015, Part I*, *LNCS* 9452, pages 470–491. Springer, Heidelberg, November / December 2015.

BJL16. F. Benhamouda, M. Joye, and B. Libert. A new framework for privacy-preserving aggregation of time-series data. *ACM Trans. Inf. Syst. Secur.*, 18(3):10:1–10:21, 2016.

BKS18. Z. Brakerski, I. Komargodski, and G. Segev. Multi-input functional encryption in the private-key setting: Stronger security from weaker assumptions. *Journal of Cryptology*, 31(2):434–520, April 2018.

BSW11. D. Boneh, A. Sahai, and B. Waters. Functional encryption: Definitions and challenges. In *TCC 2011*, *LNCS* 6597, pages 253–273. Springer, Heidelberg, March 2011.

CDG+18a. J. Chotard, E. Dufour Sans, R. Gay, D. H. Phan, and D. Pointcheval. Decentralized multi-client functional encryption for inner product. In *ASIACRYPT 2018, Part II*, *LNCS* 11273, pages 703–732. Springer, Heidelberg, December 2018.

CDG+18b. J. Chotard, E. Dufour Sans, R. Gay, D. H. Phan, and D. Pointcheval. Multi-client functional encryption with repetition for inner product. Cryptology ePrint Archive, Report 2018/1021, 2018. http://eprint.iacr.org/2018/1021.

CSS12. T.-H. H. Chan, E. Shi, and D. Song. Privacy-preserving stream aggregation with fault tolerance. In *FC 2012*, *LNCS* 7397, pages 200–214. Springer, Heidelberg, February / March 2012.

DDM16. P. Datta, R. Dutta, and S. Mukhopadhyay. Functional encryption for inner product with full function privacy. In *PKC 2016, Part I*, *LNCS* 9614, pages 164–195. Springer, Heidelberg, March 2016.

Emu17. K. Emura. Privacy-preserving aggregation of time-series data with public verifiability from simple assumptions. In *ACISP 17, Part II*, *LNCS* 10343, pages 193–213. Springer, Heidelberg, July 2017.

FT18. X. Fan and Q. Tang. Making public key functional encryption function private, distributively. In *PKC 2018, Part II*, *LNCS* 10770, pages 218–244. Springer, Heidelberg, March 2018.

GGG+14. S. Goldwasser, S. D. Gordon, V. Goyal, A. Jain, J. Katz, F.-H. Liu, A. Sahai, E. Shi, and H.-S. Zhou. Multi-input functional encryption. In *EUROCRYPT 2014*, *LNCS* 8441, pages 578–602. Springer, Heidelberg, May 2014.

JL13. M. Joye and B. Libert. A scalable scheme for privacy-preserving aggregation of time-series data. In *FC 2013*, *LNCS* 7859, pages 111–125. Springer, Heidelberg, April 2013.

LC12. Q. Li and G. Cao. Efficient and privacy-preserving data aggregation in mobile sensing. In *20th IEEE International Conference on Network Protocols, ICNP*, pages 1–10, Austin, TX, USA, 2012. IEEE Computer Society.

O'N10. A. O'Neill. Definitional issues in functional encryption. Cryptology ePrint Archive, Report 2010/556, 2010. http://eprint.iacr.org/2010/556.

SCR+11. E. Shi, T.-H. H. Chan, E. G. Rieffel, R. Chow, and D. Song. Privacy-preserving aggregation of time-series data. In *NDSS 2011*. The Internet Society, February 2011.

SW05. A. Sahai and B. R. Waters. Fuzzy identity-based encryption. In *EUROCRYPT 2005*, *LNCS* 3494, pages 457–473. Springer, Heidelberg, May 2005.

A Postponed Proofs for the Compiler from MCFE to DMCFE (Section 3)

A.1 Proof of Theorem 3.2

Before stating the formal proof of this theorem, we need some intermediate results. For any prime L, any positive integer M, any set of vectors $S \subseteq \mathbb{Z}_L^M$, we define $\mathrm{Vect}(\{\boldsymbol{y}\}_{\boldsymbol{y}\in S})$ to be the subspace generated by the vectors $\boldsymbol{y}$ in S.

Lemma A.1. *For any* prime L*, any positive integer* M*, and any vector* $\boldsymbol{u} \in \mathbb{Z}_L^M$*, the games* $\mathrm{IP}_0(L, M, \boldsymbol{u})$ *and* $\mathrm{IP}_1(L, M, \boldsymbol{u})$ *depicted in Fig. 12 are perfectly indistinguishable.*

Proof (Lemma A.1). The proof in the "selective case" where the adversary outputs all its queries at the beginning of the game follows from classical linear algebra. Since the two games are perfectly indistinguishable in this "selective case," we get perfect indistinguishability by a "complexity-leveraging-like" argument, similarly to what is done in the proof of Theorem 5.3. □

Lemma A.2. *For any* prime L*, any positive integer* M*, and any vector* $\boldsymbol{u} \in \mathbb{Z}_L^M$*, the games* $\mathrm{IPCor}_0(L, M, \boldsymbol{u})$ *and* $\mathrm{IPCor}_1(L, M, \boldsymbol{u})$ *depicted in Fig. 14 are perfectly indistinguishable.*

Remark A.3. We have the following straightforward claim.

Claim. At any point in time in $\mathrm{IPCor}_1(L, M, \boldsymbol{u})$ from Fig. 14, S is actually a basis of the vector space V (i.e., vectors in S are linearly independent).

Thus, a way to pick $\boldsymbol{v}$ uniformly under the constraint:

$$\forall \boldsymbol{y} \in S,\ \mathsf{dk}_{\boldsymbol{y}} = \langle \boldsymbol{u}, \boldsymbol{y} \rangle + \langle \boldsymbol{v}, \boldsymbol{y} \rangle$$

is to

1. Choose a set $S' \subseteq \mathbb{Z}_L^M$ such that $S' \cap S = \emptyset$ and $S' \cup S$ is a basis of $\mathbb{Z}_L^M$ (it is possible since S is a base of the subspace V and hence vectors in S are linearly independent).
2. Choose random values for $\mathsf{dk}_{\boldsymbol{y}} \in \mathbb{Z}_L$ for $\boldsymbol{y} \in S'$.
3. Solve the linear system with indeterminate $\boldsymbol{v}$:

$$\forall \boldsymbol{y} \in S \cup S',\ \mathsf{dk}_{\boldsymbol{y}} = \langle \boldsymbol{u}, \boldsymbol{y} \rangle + \langle \boldsymbol{v}, \boldsymbol{y} \rangle \ .$$

Since $S \cup S'$ is a basis of $\mathbb{Z}_L^M$, this system has a unique solution.

$\mathbf{IP}_0(L, M, \boldsymbol{u})$	$\mathbf{IP}_1(L, M, \boldsymbol{u})$
$\boldsymbol{v} \leftarrow \mathbb{Z}_L^M$	$S := \emptyset$. At any time: $V := \mathrm{Vect}(\{\boldsymbol{y}\}_{\boldsymbol{y}\in S})$
Output: $\mathcal{A}^{\mathsf{QIP}(\cdot)}(L, M, \boldsymbol{u})$	**Output:** $\mathcal{A}^{\mathsf{QIP}(\cdot)}(L, M, \boldsymbol{u})$
$\mathsf{QIP}(\boldsymbol{y})$	$\mathsf{QIP}(\boldsymbol{y})$
Set $\mathsf{dk}_{\boldsymbol{y}} := \langle \boldsymbol{u}, \boldsymbol{y} \rangle + \langle \boldsymbol{v}, \boldsymbol{y} \rangle$	If $\boldsymbol{y} \notin V$
Return $\mathsf{dk}_{\boldsymbol{y}}$	Add $\boldsymbol{y}$ to S
	Set and return $\mathsf{dk}_{\boldsymbol{y}} \leftarrow \mathbb{Z}_L^M$
	Else, find $\{\mu_{\boldsymbol{y}'} \in \mathbb{Z}_L\}_{\boldsymbol{y}'\in S}$ s.t. $\boldsymbol{y} = \sum_{\boldsymbol{y}'\in S} \mu_{\boldsymbol{y}'} \cdot \boldsymbol{y}'$,
	Set and return $\mathsf{dk}_{\boldsymbol{y}} := \sum_{\boldsymbol{y}'\in S} \mu_{\boldsymbol{y}'} \cdot \mathsf{dk}_{\boldsymbol{y}'}$

Fig. 12. Games for Lemma A.1

$\mathbf{IPCor}_{0.1}(L, M, \boldsymbol{u})$	$\mathbf{IPCor}_{0.9}(L, M, \boldsymbol{u})$
$\boldsymbol{v} \leftarrow \mathbb{Z}_L^M$	$S := \emptyset$. At any time: $V := \mathrm{Vect}(\{\boldsymbol{y}\}_{\boldsymbol{y} \in S})$
Output: $\mathcal{A}^{\mathsf{QCor}(), \mathsf{QIP}(\cdot)}(L, M, \boldsymbol{u})$	**Output:** $\mathcal{A}^{\mathsf{QCor}(), \mathsf{QIP}(\cdot)}(L, M, \boldsymbol{u})$
$\mathsf{QCor}()$	$\mathsf{QCor}()$
Return $\perp$ if called more than once.	Return $\perp$ if called more than once.
For $i \in [M]$, $\mathsf{dk}_{\boldsymbol{e}_i} \leftarrow \mathsf{QIP}(\boldsymbol{e}_i)$	For $i \in [M]$, $\mathsf{dk}_{\boldsymbol{e}_i} \leftarrow \mathsf{QIP}(\boldsymbol{e}_i)$
$\boldsymbol{w} := (\mathsf{dk}_{\boldsymbol{e}_1}, \ldots, \mathsf{dk}_{\boldsymbol{e}_M}) \in \mathbb{Z}_L^M$	$\boldsymbol{w} := (\mathsf{dk}_{\boldsymbol{e}_1}, \ldots, \mathsf{dk}_{\boldsymbol{e}_M}) \in \mathbb{Z}_L^M$
Return $\boldsymbol{w} - \boldsymbol{u}$	Return $\boldsymbol{w} - \boldsymbol{u}$
$\mathsf{QIP}(\boldsymbol{y})$	$\mathsf{QIP}(\boldsymbol{y})$
Return $\perp$ if called after QCor.	Return $\perp$ if called after QCor.
Set $\mathsf{dk}_{\boldsymbol{y}} := \langle \boldsymbol{u}, \boldsymbol{y} \rangle + \langle \boldsymbol{v}, \boldsymbol{y} \rangle$	If $\boldsymbol{y} \notin V$
Return $\mathsf{dk}_{\boldsymbol{y}}$	Add $\boldsymbol{y}$ to S
	Set and return $\mathsf{dk}_{\boldsymbol{y}} \leftarrow \mathbb{Z}_L^M$
	Else
	Find $\{\mu_{\boldsymbol{y}'} \in \mathbb{Z}_L\}_{\boldsymbol{y}' \in S}$
	s.t. $\boldsymbol{y} = \sum_{\boldsymbol{y}' \in S} \mu_{\boldsymbol{y}'} \cdot \boldsymbol{y}'$
	Set and return $\mathsf{dk}_{\boldsymbol{y}} := \sum_{\boldsymbol{y}' \in S} \mu_{\boldsymbol{y}'} \cdot \mathsf{dk}_{\boldsymbol{y}'}$

Fig. 13. Games for the proof of Lemma A.2

Proof (Lemma A.2). Let $\boldsymbol{e}_i$ be the i-th vector in the canonical basis of $\mathbb{Z}_L^M$. We introduce two additional games $\mathrm{IPCor}_{0.1}(L, M, \boldsymbol{u})$ and $\mathrm{IPCor}_{0.9}(L, M, \boldsymbol{u})$ defined in Fig. 13, where QCor is essentially implemented using calls to the QIP oracle.

The result follows from the three claims below.

Claim. $\mathrm{IPCor}_0(L, M, \boldsymbol{u})$ and $\mathrm{IPCor}_{0.1}(L, M, \boldsymbol{u})$ are perfectly indistinguishable.

Proof. This follows from the fact that since $\mathsf{dk}_{\boldsymbol{e}_i} = \langle \boldsymbol{u}, \boldsymbol{e}_i \rangle + \langle \boldsymbol{v}, \boldsymbol{e}_i \rangle = u_i + v_i$, we have $\boldsymbol{w} = \boldsymbol{u} + \boldsymbol{v}$. □

Claim. $\mathrm{IPCor}_{0.1}(L, M, \boldsymbol{u})$ and $\mathrm{IPCor}_{0.9}(L, M, \boldsymbol{u})$ are perfectly indistinguishable.

Proof. We can indeed perfectly simulate the QIP oracle of $\mathrm{IPCor}_{0.1}(L, M, \boldsymbol{u})$ and $\mathrm{IPCor}_{0.9}(L, M, \boldsymbol{u})$ respectively from the QIP' oracle of $\mathrm{IP}_0(L, M, \boldsymbol{u})$ and $\mathrm{IP}_1(L, M, \boldsymbol{u})$ respectively as follows: $\mathsf{QIP}(\boldsymbol{y})$

- Return $\perp$ if called after QCor.
- Set and return $\mathsf{dk}_{\boldsymbol{y}} := \langle \boldsymbol{u}, \boldsymbol{y} \rangle + \mathsf{QIP}'(\boldsymbol{y})$.

The fact that when QIP' is from $\mathrm{IP}_0(L, M, \boldsymbol{u})$, the result perfectly simulates QIP from $\mathrm{IPCor}_{0.1}(L, M, \boldsymbol{u})$ is straightforward. The fact that when QIP' is from $\mathrm{IP}_1(L, M, \boldsymbol{u})$, the result perfectly simulates QIP from $\mathrm{IPCor}_{0.9}(L, M, \boldsymbol{u})$ comes from the fact that if for all $\boldsymbol{y}' \in S$: $\mathsf{dk}_{\boldsymbol{y}'} = \langle \boldsymbol{u}, \boldsymbol{y}' \rangle + \mathsf{QIP}'(\boldsymbol{y}')$ and if $\boldsymbol{y} = \sum_{\boldsymbol{y}' \in S} \mu_{\boldsymbol{y}'} \cdot \boldsymbol{y}'$ then:

$$\sum_{\boldsymbol{y}' \in S} \mu_{\boldsymbol{y}'} \cdot \mathsf{dk}_{\boldsymbol{y}'} = \langle \boldsymbol{u}, \boldsymbol{y} \rangle + \sum_{\boldsymbol{y}' \in S} \mu_{\boldsymbol{y}'} \cdot \mathsf{QIP}'(\boldsymbol{y}') \ .$$

We conclude the proof using Lemma A.1. □

$\mathbf{IPCor}_0(L, M, \boldsymbol{u})$	$\mathbf{IPCor}_1(L, M, \boldsymbol{u})$
$\boldsymbol{v} \leftarrow \mathbb{Z}_L^M$	$S := \emptyset$. At any time: $V := \mathrm{Vect}(\{\boldsymbol{y}\}_{\boldsymbol{y} \in S})$
Output: $\mathcal{A}^{\mathsf{QCor}(),\mathsf{QIP}(\cdot)}(L, M, \boldsymbol{u})$	**Output:** $\mathcal{A}^{\mathsf{QCor}(),\mathsf{QIP}(\cdot)}(L, M, \boldsymbol{u})$
$\mathsf{QCor}()$	$\mathsf{QCor}()$
Return $\perp$ if called before	Return $\perp$ called before
Return $\boldsymbol{v}$	Pick $\boldsymbol{v}$ uniformly under the constraint:
	$\forall \boldsymbol{y} \in S,\ \mathsf{dk}_{\boldsymbol{y}} = \langle \boldsymbol{u}, \boldsymbol{y} \rangle + \langle \boldsymbol{v}, \boldsymbol{y} \rangle$
	Return $\boldsymbol{v}$
$\mathsf{QIP}(\boldsymbol{y})$	$\mathsf{QIP}(\boldsymbol{y})$
Return $\perp$ if called after QCor	Return $\perp$ if called after QCor
Set $\mathsf{dk}_{\boldsymbol{y}} := \langle \boldsymbol{u}, \boldsymbol{y} \rangle + \langle \boldsymbol{v}, \boldsymbol{y} \rangle$	If $\boldsymbol{y} \notin V$
Return $\mathsf{dk}_{\boldsymbol{y}}$	Add $\boldsymbol{y}$ to S. Set and return $\mathsf{dk}_{\boldsymbol{y}} \leftarrow \mathbb{Z}_L^M$
	Else
	Find $\{\mu_{\boldsymbol{y}'} \in \mathbb{Z}_L\}_{\boldsymbol{y}' \in S}$ s.t. $\boldsymbol{y} = \sum_{\boldsymbol{y}' \in S} \mu_{\boldsymbol{y}'} \cdot \boldsymbol{y}'$,
	Set and return $\mathsf{dk}_{\boldsymbol{y}} := \sum_{\boldsymbol{y}' \in S} \mu_{\boldsymbol{y}'} \cdot \mathsf{dk}_{\boldsymbol{y}'}$

Fig. 14. Games for Lemma A.2

Claim. $\mathrm{IPCor}_{0.9}(L, M, \boldsymbol{u})$ and $\mathrm{IPCor}_1(L, M, \boldsymbol{u})$ are perfectly indistinguishable.

Proof. The proof is essentially a combination of Remark A.3 and the first claim. □

This concludes the proof of Lemma A.2.

□

Proof (Theorem 3.2). We focus on the adaptive case (xx = adt). The static case (xx = sta) is simpler. We can deal with any yy $\in \{\mathrm{one}, \mathrm{pos}, \mathrm{any}\}$, as we are anyway forwarding the encryption queries directly to the MCFE encryption oracle.

Case $\kappa = 1$. We start by proving the case $\kappa = 1$. This allows us to omit the superscript k. We define $M = mn$.

We assume without loss of generality that the oracle QCor is only called once for each i (as its output is deterministic).

Let G_0 and G_3 correspond to the experiments xx-yy-$\mathrm{IND}_\beta^{\mathsf{DMCFE}}$ for $\beta = 0$ and $\beta = 1$, respectively. We prove the security by introducing two hybrid games G_1 and G_2 (see Fig. 15), for which the only difference with G_0 and G_3 respectively is in the definition of the key derivation oracle and corruption oracle.

We abuse notation as the set S and the corresponding subspace V, as well as the set of corrupted parties $\mathcal{CS}$ potentially change at each oracle query QCor and QKeyD. But we still write them as S, V, and $\mathcal{CS}$.

The proof follows from Lemmas A.4 to A.6 below.

Lemma A.4. *Games G_0 and G_1 are perfectly indistinguishable, i.e., for any adversary $\mathcal{A}$, $\mathsf{Win}_{\mathcal{A}}^{\mathsf{G}_0}(\lambda, n) = \mathsf{Win}_{\mathcal{A}}^{\mathsf{G}_1}(\lambda, n)$.*

$\underline{\mathsf{G}_0, \mathsf{G}_3 :}$	$\underline{\mathsf{G}_1, \mathsf{G}_2 :}$
	$S := \emptyset$. Vectors $\boldsymbol{v}_i$ are not initialized. At any point in time: $V = \mathrm{Vect}(\{\boldsymbol{y}_g\}_{g\in S})$ and $\mathcal{CS}$ is the set of corrupted parties.
$\mathsf{QCor}(i)$ Return $\mathsf{sk}'_i := (i, s_i, \boldsymbol{u}_i, \boldsymbol{v}_i)$	$\mathsf{QCor}(i)$ Pick $\boldsymbol{v}_i$ uniformly under the constraint $\forall g \in S,\ \mathsf{dk}_{i,g} = \langle \boldsymbol{u}_i, \boldsymbol{y}_{i,g}\rangle + \langle \boldsymbol{v}_i, \boldsymbol{y}_g\rangle$ Return $\mathsf{sk}'_i := (i, s_i, \boldsymbol{u}_i, \boldsymbol{v}_i)$
$\mathsf{QKeyD}(f)$ For any $i \in [n]$, $\mathsf{dk}_{i,f} := \langle \boldsymbol{u}_i, \boldsymbol{y}_{i,f}\rangle + \langle \boldsymbol{v}_i, \boldsymbol{y}_f\rangle$ $\mathsf{sk}'_{i,f} := (s_{i,f}, \mathsf{dk}_{i,f})$ Return $\{\mathsf{sk}'_{i,f}\}_{i\in[n]}$	$\mathsf{QKeyD}(f)$ $\mathsf{dk}_f := \langle \boldsymbol{u}, \boldsymbol{y}_f\rangle$ For any $i \in \mathcal{CS}$, $\mathsf{dk}_{i,f} := \langle \boldsymbol{u}_i, \boldsymbol{y}_{i,f}\rangle + \langle \boldsymbol{v}_i, \boldsymbol{y}_f\rangle$ If $\boldsymbol{y}_f \notin V$, Pick $\{\mathsf{dk}_{i,f}\}_{i\notin\mathcal{CS}}$ uniformly under the constraint $\sum_{i\in[n]} \mathsf{dk}_{i,f} = \mathsf{dk}_f$ Add $\boldsymbol{y}_f$ to the set S If $\boldsymbol{y}_f \in V$ Find $\{\mu_g \in \mathbb{Z}_L\}_{g\in S}$ s.t. $\boldsymbol{y}_f = \sum_{g\in S} \mu_g \cdot \boldsymbol{y}_g$ Set $\mathsf{dk}_{i,f} := \sum_{g\in S} \mu_g \cdot \mathsf{dk}_{i,g}$ Return $\{\mathsf{sk}'_{i,f} := (s_{i,f}, \mathsf{dk}_{i,f})\}_{i\in[n]}$

Fig. 15. Key derivation and corruption oracles in the games for the proof of Theorem 3.2

Proof. Let us suppose that we know in advance a non-corrupted party P_{i^*}, i.e., $i^* \in [n]\setminus\mathcal{CS}$. We then remark that both in G_0 and G_1, we could compute $\mathsf{dk}_{i^*,f}$ in $\mathsf{QKeyD}(f)$ as:

$$\mathsf{dk}_{i^*,f} = \mathsf{dk}_f - \sum_{i\neq i^*} \mathsf{dk}_{i,f} \ ,$$

where $\mathsf{dk}_f = \langle \boldsymbol{u}, \boldsymbol{y}_f\rangle$.

For any $i \in [n]$, we define $\boldsymbol{u}'_i \in \mathbb{Z}_L^M$ to be the vector corresponding to the concatenation of $\mathbf{0} \in \mathbb{Z}_L^m$, …, $\mathbf{0} \in \mathbb{Z}_L^m$, $\boldsymbol{u}_i$, $\mathbf{0} \in \mathbb{Z}_L^m$, …, $\mathbf{0} \in \mathbb{Z}_L^m$. We have:

$$\langle \boldsymbol{u}_i, \boldsymbol{y}_{i,f}\rangle = \langle \boldsymbol{u}'_i, \boldsymbol{y}_f\rangle \ .$$

We can perfectly simulate the oracles QCor and QKeyD of Games G_0 and G_1 respectively from the QCor^i and QIP^i oracles of $n-1$ instances (for $i \in [n]\setminus\{i^*\}$) of $\mathrm{IPCor}_0(L, M, \boldsymbol{u})$ and $\mathrm{IPCor}_1(L, M, \boldsymbol{u})$ respectively with $\boldsymbol{u} = \boldsymbol{u}'_i$, as follows: $\mathsf{QIP}(\boldsymbol{y})$

- $\mathsf{QCor}(i)$ generate $v_i \leftarrow \mathsf{QCor}^i()$.
- $\mathsf{QKeyD}(f)$ generate $\mathsf{dk}_{i,f} \leftarrow \mathsf{QIP}^i(\boldsymbol{y}_f)$ for $i \in [n]\setminus\{i^*\}$ add $\mathsf{dk}_{i^*,f} = \mathsf{dk}_f - \sum_{i\neq i^*} \mathsf{dk}_{i,f}$.

We conclude the proof by applying Lemma A.2. □

Lemma A.5. *For any PPT adversary $\mathcal{A}$, there exists a PPT adversary $\mathcal{B}$ such that:*

$$|\mathsf{Win}^{\mathsf{G}_1}_{\mathcal{A}}(\lambda, n) - \mathsf{Win}^{\mathsf{G}_2}_{\mathcal{A}}(\lambda, n)| \leq \mathsf{Adv}^{\text{xx-yy-IND}}_{\mathsf{MCFE},\mathcal{B}}(\lambda, n) \ .$$

Proof. We remark that in Games G_1 and G_2, the values $\boldsymbol{u}_i$ are not used directly by $\mathsf{QKeyD}(f)$ as long as slot i is not corrupted. Only the value dk_f is used, which is provided by the key derivation oracle for MCFE (together with $s_{i,f}$), as $\mathsf{sk}_f = (\{s_{i,f}\}_{i\in[n]}, \mathsf{dk}_f)$. When the slot i is corrupted, we need to learn $\boldsymbol{u}_i$, but the corruption oracle of MCFE will give us this value, as $\boldsymbol{u}_i$ is part of sk_i. Thus, the reduction to xx-yy-IND security of MCFE is straightforward. □

Lemma A.6. *G_2 and G_3 are perfectly indistinguishable, i.e., for any adversary $\mathcal{A}$, $\mathsf{Win}^{\mathsf{G}_2}_{\mathcal{A}}(\lambda, n) = \mathsf{Win}^{\mathsf{G}_3}_{\mathcal{A}}(\lambda, n)$.*

Proof. The proof is similar to the one of Lemma A.4. □

This concludes the proof of the case $\kappa = 1$.

General case ($\kappa \geq 1$). The proof is similar, as we can see such a case as κ parallel and independent executions of the case $\kappa = 1$, if we ignore the values which are independent of k, such as $s_{i,f}$. □

A.2 Proof of Theorem 3.4

Before proving Theorem 3.4, let us state and prove the following extension of Lemma A.2:

Corollary A.7. *Let $\mathrm{IPCor}'_b(L, M, \boldsymbol{u})$ be defined as $\mathrm{IPCor}_b(L, M, \boldsymbol{u})$ in Fig. 14 except that $\mathsf{QIP}(\boldsymbol{y})$ aborts if the following condition is satisfied:*

> *$\boldsymbol{y}$ is not in the subspace $V = \{\boldsymbol{y}'\}_{\boldsymbol{y}'\in S}$ generated by the previous queries but when performing the Gaussian elimination over the matrix with rows $\{\boldsymbol{y}'^{\mathsf{T}}\}_{\boldsymbol{y}'\in S} \cup \{\boldsymbol{y}^{\mathsf{T}}\}$, some rows do not have an* invertible *pivot (a.k.a., leading non-zero coefficient).*

(We note that this condition is never satisfied when L is prime.) Then for integer $L \geq 2$, any positive integer M, and any vector $\boldsymbol{u} \in \mathbb{Z}_L^M$, the games $\mathrm{IPCor}'_0(L, M, \boldsymbol{u})$ and $\mathrm{IPCor}'_1(L, M, \boldsymbol{u})$ are perfectly indistinguishable.

Proof. The main difference is that we are now working with modules over a ring $\mathbb{Z}_L$, rather than with vector spaces over a field.

The proof is essentially the same as before, using a similarly modified Lemma A.1 (using similarly modified games $\mathrm{IP}'_b(L, M, \boldsymbol{u})$ where QIP aborts as in $\mathrm{IPCor}'_b(L, M, \boldsymbol{u})$). We remark that that the condition of non-abort implies that, at any point in time, the pivots of Gaussian elimination over the set S of vectors $\boldsymbol{y}$ queried to QIP (and not leading to an abort) are all invertible element of $\mathbb{Z}_L$. Thus, S is not only always a basis of $V = \mathrm{Vect}(S)$, but S can also be extended into a basis $S \cup S'$ of $\mathbb{Z}_L^M$. Furthermore, S' can be chosen as a subset of the canonical basis $\{\boldsymbol{e}_i\}_{i\in[M]}$ of $\mathbb{Z}_L^M$. This is essentially the only property we use in the proof that is specific to vector spaces over fields, rather than modules over rings $\mathbb{Z}_L$.

We point out that S might not even be a basis of $V = \mathrm{Vect}(S)$ without the abort: if 2 divides M and the first query to QIP is $\boldsymbol{y}_1 = (2, \ldots, 2)$, and the second query is $\boldsymbol{y}_2 = (1, \ldots, 1)$, then $\boldsymbol{y_2} \notin \mathrm{Vect}(\boldsymbol{y_1})$. Thus after the second query, $S = \{\boldsymbol{y}_1, \boldsymbol{y}_2\}$, which is not a basis as $\boldsymbol{y}_1$ and $\boldsymbol{y}_2$ are linearly dependent. □

Proof (Theorem 3.4). The proof is similar to the one of Theorem 3.2. We directly consider the general case $\kappa \geq 1$. The main difference is the introduction of two games $\mathsf{G}_{0.5}$ and $\mathsf{G}_{2.5}$ which are similar to G_0 and G_3 respectively, except that QKeyD aborts when:

> $\boldsymbol{y}_f^k$ is linearly dependent of the vectors $\boldsymbol{y}_{f'}^k$ for previous queries $\mathsf{QKeyD}(f')$ but when performing the Gaussian elimination over the matrix with rows $\boldsymbol{y}_{f'}^{k\mathsf{T}}$ corresponding to the previous queries $\mathsf{QKeyD}(f')$ and to $f' = f$, some rows do not have an *invertible* pivot (a.k.a., leading non-zero coefficient).

Using a similar proof as before, just replacing Lemma A.2 by Corollary A.7, we get the following claim.

Claim. For any PPT adversary $\mathcal{A}$, there exists a PPT adversary $\mathcal{B}$ such that:

$$|\mathsf{Win}^{\mathsf{G}_{0.5}}_{\mathcal{A}} - \mathsf{Win}^{\mathsf{G}_{2.5}}_{\mathcal{A}}| \leq \mathsf{Adv}^{\text{xx-yy-IND}}_{\mathsf{MCFE},\mathcal{B}}(\lambda, n) \ .$$

Finally, we have the following claims.

Claim. For any PPT adversary $\mathcal{A}$, there exists a PPT adversary $\mathcal{B}'$ such that:

$$|\mathsf{Win}^{\mathsf{G}_0}_{\mathcal{A}} - \mathsf{Win}^{\mathsf{G}_{0.5}}_{\mathcal{A}}| \leq \mathsf{Adv}^{\mathsf{Factor}}_{\mathsf{GenL},\mathcal{B}'}(\lambda) \ .$$

Proof. The only difference between is when $\mathsf{QKeyD}(f)$ aborts. Let us consider the first aborting query. Gaussian elimination as in the condition above yields a non-invertible non-zero element $\mu \in \mathbb{Z}_M$. The greatest common divisor (gcd) of μ and M yields a non-trivial factor of M. □

Claim. For any PPT adversary $\mathcal{A}$, there exists a PPT adversary $\mathcal{B}''$ such that:

$$|\mathsf{Win}^{\mathsf{G}_{2.5}}_{\mathcal{A}} - \mathsf{Win}^{\mathsf{G}_3}_{\mathcal{A}}| \leq \mathsf{Adv}^{\mathsf{Factor}}_{\mathsf{GenL},\mathcal{B}''}(\lambda) \ .$$

Proof. The proof is similar to the one of the previous claim. □

The theorem follows by combining $\mathcal{B}'$ and $\mathcal{B}''$ into a single adversary flipping a coin b at the beginning to decide whether it will behave as $\mathcal{B}'$ or $\mathcal{B}''$. □

B Proof for the Compiler from pos-IND to any-IND for Labeled DMCFE Schemes (Section 4.2)

Proof (Theorem 4.4). For the sake of simplicity, we suppose that xx = adt. The proof for xx = sta is simpler.

Let G_0 and G_3 correspond to the experiments xx-yy-IND$^{\mathsf{DMCFE}}_{\beta}$ for $\beta = 0$ and $\beta = 1$, respectively (see Fig. 16 for the definition of the random oracles H_1 and H_2). We denote by ν_γ the output length of the oracle H_γ. We prove the security by introducing two hybrid games G_1 and G_2 (see Figs. 16 and 17).

Essentially in G_1 and G_2, the keys $\mathsf{k}_{i,j,\ell}$ and the ciphertexts ct_i are generated uniformly at random on the fly. The random oracles are programmed to explain these values, when the adversary corrupts a new slot i or calls the QEnc oracle for a label ℓ in such a way that for all the i, $Q_{i,\ell} > 0$ after this call. In the latter case, the programming of the random oracles is done by $\mathsf{AdaptEnc}$.

We have the following claims which prove the result. The first claim is a straightforward reduction. The last two claims come from bounding the probability of the event **Abort**, which comes from collisions.

Claim. For any PPT adversary $\mathcal{A}$, there exists a PPT adversary $\mathcal{B}$ such that:

$$|\mathsf{Win}^{\mathsf{G}_1}_{\mathcal{A}}(\lambda, n) - \mathsf{Win}^{\mathsf{G}_2}_{\mathcal{A}}(\lambda, n)| \leq \mathsf{Adv}^{\text{xx-yy-IND}}_{\mathsf{DMCFE},\mathcal{B}}(\lambda, n) \ .$$

Claim.

$$|\mathsf{Win}^{\mathsf{G}_0}_{\mathcal{A}}(\lambda, n) - \mathsf{Win}^{\mathsf{G}_1}_{\mathcal{A}}(\lambda, n)| \leq \frac{2 \cdot q_{\mathsf{H}_1} + (2n+1) \cdot (q_{\mathsf{H}_2} \cdot q_{\mathsf{QEnc}} + q^2_{\mathsf{QEnc}})}{2^\lambda} \ .$$

Proof. Without loss of generality, we assume that the adversary only makes queries such that Condition (*) (defined in Section 2 on page 6) is satisfied.

Let us consider the q-th query $\mathsf{QCor}(i)$ and bound the probability of the event **Abort(1)**. Let $j \notin \mathcal{CS} \backslash \{i\}$, we start by bounding the probability that HT_1 contains a key of the form $\mathsf{k}_{i,j}\|i\|j\|\star$ or $\mathsf{k}_{j,i}\|j\|i\|\star$. Since $\mathsf{k}_{i,j}$ and $\mathsf{k}_{j,i}$ are drawn uniformly random and independently from $\{0,1\}^\lambda$, for any key of HT_1 of the form $\star'\|i\|j\|\star$ or $\star'\|j\|i\|\star$, the probability that $\star' = \mathsf{k}_{i,j}$ or $\star' = \mathsf{k}_{j,i}$ is exactly $1/2^\lambda$. Thus, by union bound, the probability that HT_1 contains a key of the form $\mathsf{k}_{i,j}\|i\|j\|\star$ or $\mathsf{k}_{j,i}\|j\|i\|\star$ is at most:

$$\frac{q_{\mathsf{H}_1,i,j} + q_{\mathsf{H}_1,j,i}}{2^\lambda} \ ,$$

where $q_{\mathsf{H}_1,i,j}$ is the number of queries to H_1 of the form $\star'\|i\|j\|\star$ and $q_{\mathsf{H}_1,j,i}$ the number of queries of the form $\star'\|j\|i\|\star$, respectively. (Even if keys of HT_1 are also added by the challenger and not only when the

$\underline{\mathsf{G}_0, \mathsf{G}_1, \mathsf{G}_2, \mathsf{G}_3 :}$

HT_1 and HT_2 are two empty arrays

$\mathsf{H}_\gamma(z)$ for $\gamma \in \{0,1\}$

If $\mathsf{HT}[z]$ does not exist, $\mathsf{HT}[z] \leftarrow \{0,1\}^{\nu_\gamma}$

Return $\mathsf{HT}[z]$

$\underline{\mathsf{G}_{1+\beta} \text{ for } \beta \in \{0,1\} :}$

$\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, 1^n)$

$\{\mathsf{sk}_i\}_{i\in[n]} \leftarrow \mathsf{KeyGen}(\mathsf{pp})$

$\alpha \leftarrow \mathcal{A}^{\mathsf{QCor}(\cdot),\mathsf{QEnc}(\cdot,\cdot,\cdot,\cdot),\mathsf{QKeyD}(\cdot)}(\mathsf{pp})$

Output: α if Condition (*) is satisfied,
or a uniform bit otherwise

$\mathsf{QCor}(i)$

If already called for the same i, return same answer

Add i to $\mathcal{CS}$

For all $j \notin (\mathcal{CS} \setminus \{i\})$,

Define $\mathsf{k}_{i,j} \leftarrow \{0,1\}^\lambda$ and $\mathsf{k}_{j,i} \leftarrow \{0,1\}^\lambda$

Abort(1) if HT_1 contains a key of the form

$\mathsf{k}_{i,j}\|i\|j\|\star$ or $\mathsf{k}_{j,i}\|j\|i\|\star$ for $j \notin (\mathcal{CS} \setminus \{i\})$

For all previous queries $\mathsf{QEnc}(i,\star,\star,\ell)$ for some ℓ,

If $\mathsf{QEnc}(j,\star,\star,\ell)$ has not been queried for all $j \notin \mathcal{CS}$

Let $(\mathsf{ct}'_i, r_i, \{\mathsf{k}_{j,i,\ell}\}_{j\in[n]})$ be the output of the $\mathsf{QEnc}(i,\star,\star,\ell)$ query

For all $j \notin \mathcal{CS}$ where $\mathsf{QEnc}(j,\star,\star,\ell)$ has not been called,

Define $\mathsf{k}_{i,j,\ell} \leftarrow \{0,1\}^\lambda$

$\mathsf{HT}_1[\mathsf{k}_{i,j}\|i\|j\|\ell] := \mathsf{k}_{i,j,\ell}$

$\mathsf{K}_{i,\ell} := \oplus_{m\in[n]}\mathsf{k}_{i,m,\ell}$

Abort(2) if HT_2 contains a key of the form $\mathsf{K}_{i,\ell}\|r_i$

$\mathsf{ct}_i \leftarrow \mathsf{Enc}(i, x_i^\beta)$

Set $\mathsf{HT}_2[\mathsf{K}_{i,\ell}\|r_i] := \mathsf{ct}'_i \oplus \mathsf{ct}_i$

For all j,ℓ if $\mathsf{k}_{i,j,\ell}$ is defined, set $\mathsf{HT}_1[\mathsf{k}_{i,j}\|i\|j\|\ell] := \mathsf{k}_{i,j,\ell}$

For all j,ℓ if $\mathsf{k}_{j,i,\ell}$ is defined, set $\mathsf{HT}_1[\mathsf{k}_{j,i}\|j\|i\|\ell] := \mathsf{k}_{j,i,\ell}$

For all labels ℓ such that $\mathsf{QEnc}(j,\star,\star,\ell)$ has been queried for all $j \notin \mathcal{CS}$,

Run $\mathsf{AdaptEnc}(\ell)$

Return $\mathsf{sk}'_i = (\mathsf{sk}_i, \{\mathsf{k}_{i,j}, \mathsf{k}_{j,i}\}_{j\in[n]})$

Fig. 16. Corruption queries QCor of the games for the proof of Theorem 4.4

$\underline{\mathsf{G}_{1+\beta} \text{ for } \beta \in \{0,1\} :}$

$\mathsf{QEnc}(i, x_i^0, x_i^1, \ell)$

$r_i \leftarrow \{0,1\}^\lambda$

$\mathsf{ct}'_i \leftarrow \{0,1\}^{|\mathsf{ct}'_i|}$

If $i \in \mathcal{CS}$ or if $\mathsf{QEnc}(j, \star, \star, \ell)$ has already been called for all $j \notin \mathcal{CS}$

If $i \in \mathcal{CS}$ for all $m \in [n]$,

$\mathsf{k}_{i,m,\ell} := \mathsf{H}_1[\mathsf{k}_{i,m} \| i \| m \| \ell]$

$\mathsf{K}_{i,\ell} := \oplus_{m \in [n]} \mathsf{k}_{i,m,\ell}$

Abort(3) if HT_2 contains a key of the form $\mathsf{K}_{i,\ell} \| r_i$

$\mathsf{ct}_i \leftarrow \mathsf{Enc}(i, x_i^\beta)$

Set $\mathsf{HT}_2[\mathsf{K}_{i,\ell} \| r_i] := \mathsf{ct}'_i \oplus \mathsf{ct}_i$

Return $(\mathsf{ct}'_i, r_i, \{\mathsf{k}_{j,i,\ell}\}_{j \in [n]})$

For all $j \notin \mathcal{CS}$, if $\mathsf{k}_{j,i,\ell}$ not defined

Define $\mathsf{k}_{j,i,\ell} \leftarrow \{0,1\}^\lambda$

For all $j \in \mathcal{CS}$,

$\mathsf{k}_{j,i,\ell} := \mathsf{H}_1(\mathsf{k}_{j,m} \| j \| m \| \ell)$

If after this call to QEnc, for all $j \notin \mathcal{CS}$, $\mathsf{QEnc}(j, \star, \star, \ell)$ has been called

Run $\mathsf{AdaptEnc}(\ell)$

Return $(\mathsf{ct}'_i, r_i, \{\mathsf{k}_{j,i,\ell}\}_{j \in [n]})$

$\mathsf{AdaptEnc}(\ell)$

Do nothing if it was already called on the same label ℓ

Remark that when $\mathsf{AdaptEnc}(\ell)$ is called, for all $j \notin \mathcal{CS}$, for all $i \in [n]$,

$\mathsf{k}_{i,j,\ell}$ has been defined by a call $\mathsf{QEnc}(j, \star, \star, \ell)$

For all $j \in \mathcal{CS}$, for all $i \in [n]$,

$\mathsf{k}_{i,j}$ has been defined by $\mathsf{QCor}(j)$, and we set $\mathsf{k}_{i,j,\ell} := \mathsf{H}_1(\mathsf{k}_{i,j} \| i \| j \| \ell)$

For all $i \notin \mathcal{CS}$, $\mathsf{K}_{i,\ell} := \oplus_{m \in [n]} \mathsf{k}_{i,m,\ell}$

Abort(4) if HT_2 contains a key of the form $\mathsf{K}_{i,\ell} \| \star$ for $i \notin \mathcal{CS}$

For all previous queries $\mathsf{QEnc}(i, x_i^0, x_i^1, \ell)$ for some $i \notin \mathcal{CS}, x_i^0, x_i^1, \ell$,

Let $(\mathsf{ct}'_i, r_i, \{\mathsf{k}_{j,i,\ell}\}_{j \in [n]})$ be the output of the QEnc query

$\mathsf{ct}_i \leftarrow \mathsf{Enc}(i, x_i^\beta)$

Set $\mathsf{HT}_2[\mathsf{K}_{i,\ell} \| r_i] := \mathsf{ct}'_i \oplus \mathsf{ct}_i$

Fig. 17. Encryption queries QEnc of the games for the proof of Theorem 4.4

adversary queries H_1, the keys added by the challenger can never create an abort, so we are ignoring them.) By union bound over $j \notin \mathcal{CS} \setminus \{i\}$, we get that the probability that the q-th query aborts is at most:

$$\sum_{j \notin \mathcal{CS} \setminus \{i\}} \frac{q_{\mathsf{H}_1,i,j} + q_{\mathsf{H}_1,j,i}}{2^\lambda} \leq \frac{q_{\mathsf{H}_1,i} + q_{\mathsf{H}_1,i}}{2^\lambda} = \frac{2 \cdot q_{\mathsf{H}_1,i}}{2^\lambda} \ ,$$

where $q_{\mathsf{H}_1,i,j}$ is the number of queries to H_1 of the form $\star' \| i \| \star'' \| \star$ and $q_{\mathsf{H}_1,j,i}$ the number of queries of the form $\star' \| \star'' \| i \| \star$, respectively.

By remarking that only the first query $\mathsf{QCor}(i)$ for a given i might abort (if the query happens for an already queried i, the same result gets returned) and by union bound, the probability for the execution of **Abort(1)** is at most:

$$\sum_{i \in [n]} \frac{2 \cdot q_{\mathsf{H}_1,i}}{2^\lambda} = \frac{2 \cdot q_{\mathsf{H}_1}}{2^\lambda} \ .$$

To determine the probability for **Abort(2)-(4)**, we introduce $q'_{\mathsf{H}_2} = q_{\mathsf{H}_2} + q_{\mathsf{QEnc}}$, which is an upper bound on the number of (table) keys set in the table HT_2, either generated by the adversary directly querying the oracle H_2, or added by QCor, QEnc, and $\mathsf{AdaptEnc}$. The number of the second kind of keys is bounded by the number of queries to QEnc, because keys added by QCor and $\mathsf{AdaptEnc}$ correspond to queries $\mathsf{QEnc}(i, \star, \star, \star)$ where i was not corrupted before and hence no key was added to HT_1 at the time QEnc was called (furthermore if $\mathsf{AdaptEnc}(\ell)$ is called on a label ℓ, no more keys related to $\mathsf{QEnc}(\star, \star, \star, \ell)$ can be added to HT_2 by QEnc). Recall that we did not need to introduce such quantity for H_1 because there was no risk of collisions between keys added by QCor and QEnc and keys that might produce **Abort(1)**.

In the next step, we consider the probability of **Abort(2)** in the q-th query of $\mathsf{QCor}(i)$. Let $i \notin \mathcal{CS}$, we start by bounding the probability that HT_2 contains a key of the form $\mathsf{K}_{i,\ell} \| r_i$ for a fixed r_i. Since $\mathsf{K}_{i,\ell}$ is constructed by taking the XOR of the sampled keys $\mathsf{k}_{i,m,\ell}$, because we sample at least one new random key (namely, $\mathsf{k}_{i,i,\ell}$) in the corruption query, $\mathsf{K}_{i,\ell}$ is also a random value. This results in the probability of $1/2^\lambda$ for any key of HT_2 to be of the form $\star \| r_i$ with $\star = \mathsf{K}_{i,\ell}$. Thus, by union bound, the probability that HT_2 contains a key of the form $\mathsf{K}_{i,\ell} \| r_i$ is at most $q'_{\mathsf{H}_2}/2^\lambda$. By remarking again that only the first query $\mathsf{QCor}(i)$ for a given i might abort and by union bound, the probability of **Abort(2)** is at most:

$$\frac{n \cdot q_{\mathsf{QEnc}} \cdot q'_{\mathsf{H}_2}}{2^\lambda} \ .$$

We consider the probability of the first abort in the encryption procedure QEnc, **Abort(3)**. We start by bounding the probability that HT_2 contains a key of the form $\mathsf{K}_{i,\ell} \| r_i$. Because we sample a value r_i uniformly random and independently in every encryption query QEnc, we get a probability of $1/2^\lambda$ for any key of HT_2 to be of the form $\mathsf{K}_{i,\ell} \| \star$ with $\star = r_i$. Thus, by union bound, the probability that HT_2 contains a key of the form $\mathsf{K}_{i,\ell} \| r_i$ is at most $q'_{\mathsf{H}_2}/2^\lambda$. By taking the union bound over the encryption queries for all the different labels, we get that the probability of the event **Abort(3)** is at most:

$$\frac{q'_{\mathsf{H}_2} \cdot q_{\mathsf{QEnc}}}{2^\lambda} \ .$$

In the last step, we consider the q-th query of the $\mathsf{AdaptEnc}$ procedure and determine the abort probability for **Abort(4)**. Without loss of generality, we suppose that the q-th query of $\mathsf{AdaptEnc}$ is for a label ℓ, such that $\mathsf{AdaptEnc}(\ell)$ was never called before (as otherwise, $\mathsf{AdaptEnc}$ does nothing and in particular does not abort). Let us also consider a slot $i^* \notin \mathcal{CS}$. We start by bounding the probability that HT_2 contains a key of the form $\mathsf{K}_{i^*,\ell}$, where $\mathsf{K}_{i^*,\ell} = \oplus_{m \in [n]} \mathsf{k}_{i^*,m,\ell}$. We will show that $\mathsf{K}_{i^*,\ell}$ is uniformly random and independent of all previous keys in HT_2, by showing that at least one share $\mathsf{k}_{i^*,m,\ell}$ is uniformly random and independent of everything else.

We consider two cases:

Case 1: $\mathsf{AdaptEnc}(\ell)$ was called by $\mathsf{QEnc}(i, \star, \star, \ell)$. In this case, necessarily $i \notin \mathcal{CS}$ (otherwise $\mathsf{AdaptEnc}$ would not have been called) and $\mathsf{k}_{i^*,i,\ell}$ is freshly sampled during the execution of $\mathsf{QEnc}(i, \star, \star, \ell)$. This implies that $\mathsf{K}_{i^*,\ell}$ is uniformly random and independent of all previous keys in HT_2.

Case 2: $\mathsf{AdaptEnc}(\ell)$ was called by $\mathsf{QCor}(i)$. In this case, $\mathsf{QEnc}(i, \star, \star, \ell)$ was never called before (otherwise $\mathsf{AdaptEnc}(\ell)$ would have been called before). This implies that no key shares $\mathsf{k}_{j,i,\ell}$ for $j \notin \mathcal{CS}$ under label ℓ have been defined so far, which also means that $\mathsf{HT}_1[\mathsf{k}_{j,i} \| j \| i \| \ell]$ does not exist for all $j \notin \mathcal{CS}$ (as otherwise **Abort(1)** in the $\mathsf{QCor}(i)$ query would have been triggered). Thus, in the $\mathsf{AdaptEnc}(\ell)$ procedure, $\mathsf{k}_{j,i,\ell}$ gets set for all $j \notin \mathcal{CS}$, as $\mathsf{k}_{j,i',\ell} := \mathsf{H}_1(\mathsf{k}_{j,i'} \| j \| i' \| \ell)$, and hence is freshly sampled uniformly at random. In particular, $\mathsf{k}_{i^*,i,\ell}$ is uniformly random and independent of all previous keys in HT_2, and so is $K_{i^*,\ell}$.

Hence in both case, the probability that HT_2 contains a key of the form $\mathsf{K}_{i^*,\ell} \| \star$ is at most $1/2^\lambda$. By union bound over $i^* \in \mathcal{CS} \subseteq [n]$, in each call $\mathsf{AdaptEnc}(\ell)$ for a label on which it was never called, **Abort(4)** happens with probability at most $n \cdot q'_{\mathsf{H}_2}/2^\lambda$. By union bound over all the labels queried by the adversary, since there are at most q_{QEnc} such labels, the probability that **Abort(4)** happens is at most:

$$\frac{n \cdot q_{\mathsf{QEnc}} \cdot q'_{\mathsf{H}_2}}{2^\lambda} \ .$$

This concludes the proof of the claim. □

This concludes the proof of Theorem 4.4. □