

Decentralizing Inner-Product Functional Encryption

Michel Abdalla^{1,2}[0000–0002–2447–4329], Fabrice
Benhamouda³[0000–0002–8300–1820],
Markulf Kohlweiss⁴, and Hendrik Waldner⁴[0000–0002–9083–5794]

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) [12, 23, 24] 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 $\text{Enc}(x)$ 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 $\text{Enc}(x)$. 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 [20]. 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 [15, 20].

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. [15]. In DMCFE, the key derivation procedure KeyDer is divided into two procedures KeyDerShare and KeyDerComb . The KeyDer procedure allows each party P_i to generate a share $\text{sk}_{i,f}$ of the functional key sk_f from its secret key sk_i . The KeyDerComb procedure is then used to combine these n different shares $\text{sk}_{1,f}, \dots, \text{sk}_{n,f}$ 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 $\text{Enc}(\text{sk}_1, x_1, \ell), \dots, \text{Enc}(\text{sk}_n, x_n, \ell)$. 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 [15]).

In the last few years, many multi-input or multi-client functional encryption schemes have been constructed. As noted in [4], 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 [6, 7, 13, 20] are based on more unstable assumptions, such as indis-

tinguishable 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 [3, 4] and Chotard et al. in [15], which consider different types of secret-key constructions for the inner-product functionality. In these schemes, each function is specified by a collection \mathbf{y} of n vectors $\mathbf{y}_1, \dots, \mathbf{y}_n$ and takes a collection \mathbf{x} of n vectors $\mathbf{x}_1, \dots, \mathbf{x}_n$ as input. Their output is $f_{\mathbf{y}}(\mathbf{x}) = \sum_{i=1}^n \langle \mathbf{x}_i, \mathbf{y}_i \rangle = \langle \mathbf{x}, \mathbf{y} \rangle$. As the original single-input inner-product functionality [2, 5, 11, 17] and their quadratic extensions [8], multi-input or multi-client inner-product functionalities can be quite useful for computing statistics or performing data mining on encrypted databases [4, 15].

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. [15] 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 [15], 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. [3] 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. [15] 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. $\text{msk} = \{\text{sk}_i\}_{i \in [n]}$, 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 $f : \mathbf{x} \mapsto \sum_{i=1}^n \langle \mathbf{x}_i, \mathbf{y}_i \rangle$ can be written as $\text{sk}_f = (\{s(\text{sk}_i, \mathbf{y})\}_{i \in [n]}, \sum_{i=1}^n \langle u(\text{sk}_i), \mathbf{y}_i \rangle)$, where f is defined by the collection \mathbf{y} of the vectors $\mathbf{y}_1, \dots, \mathbf{y}_n$ and takes as input a collection \mathbf{x} of n vectors $\mathbf{x}_1, \dots, \mathbf{x}_n$, 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 [15], without pairings but supporting labels, has functional decryption keys of the form $\text{sk}_f = (\mathbf{y}, \sum_i \text{sk}_i \cdot \mathbf{y}_i)$.

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 $t_{i,\mathbf{y}}$ of a fresh secret sharing of zero $\{t_{i,\mathbf{y}}\}_{i \in [n]}$. It then computes $s_{i,\mathbf{y}} = s(\text{sk}_i, \mathbf{y})$ (which is a local computation) and $\text{dk}_{i,\mathbf{y}} = \langle u(\text{sk}_i), \mathbf{y}_i \rangle + t_{i,\mathbf{y}}$. The output key share is $\text{sk}_{i,\mathbf{y}} = (s_{i,\mathbf{y}}, \text{dk}_{i,\mathbf{y}})$. In the **KeyDerComb** procedure, the $\text{dk}_{i,\mathbf{y}}$ values get summed up to cancel out the $t_{i,\mathbf{y}}$ values and to obtain $\sum_i \langle u(\text{sk}_i), \mathbf{y}_i \rangle$. The output gets then extended with the values $s(\text{sk}_i, \mathbf{y})$ to obtain the complete functional decryption key sk_f . This works but the question on the generation of the fresh secret sharing of zero $\{t_{i,\mathbf{y}}\}_{i \in [n]}$ is left out.

One solution consists in generating it as follows: $t_{i,\mathbf{y}} = \sum_{j \neq i} (-1)^{j < i} F_{\text{K}_{i,j}}(\mathbf{y})$, where $F_{\text{K}_{i,j}}$ is a pseudorandom function with key $\text{K}_{i,j} = \text{K}_{j,i}$ shared between parties P_i and P_j . This yields an DMCFE scheme secure against static corruption.

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

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 $\{\mathbf{v}_i\}_{i \in [n]}$, $\mathbf{v}_i \in \mathbb{Z}_L^{m \cdot n}$, such that $\mathbf{v}_n = -\sum_{i=1}^{n-1} \mathbf{v}_i$, to derive the key shares $\mathbf{sk}_{i,f} = \langle u(\mathbf{sk}_i), \mathbf{y}_i \rangle + \langle \mathbf{v}_i, \mathbf{y} \rangle$. Here, $\langle \mathbf{v}_i, \mathbf{y} \rangle$ acts as a kind of information theoretic pseudorandom function with key \mathbf{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 \mathbf{y} . If one computes shares on different \mathbf{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 [3, 4, 15]. 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. [4] 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 k_1, \dots, k_n , such that $k_1 \oplus \dots \oplus k_n = K$. Each party P_i receives its MIFE key \mathbf{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 $\mathbf{ct}_i \leftarrow \text{Enc}(\mathbf{sk}_i, \mathbf{x}_i)$ and then encrypts the output \mathbf{ct}_i using the symmetric encryption scheme to get $\mathbf{ct}'_i \leftarrow \text{Enc}_{\text{SE}}(K, \mathbf{ct}_i)$. The output of the encryption procedure is (\mathbf{ct}'_i, 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 \mathbf{ct}_i from \mathbf{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 $k_{i,j}$, such that $k_{i,1} \oplus \dots \oplus k_{i,n} = K_i$. The share $k_{i,j}$ 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 $(ct'_i, \{k_{j,i}\}_{j \in [n]})$.

If the adversary does not know all of the shares $\{k_{i,j}\}_{i,j \in [n]}$, then security relies on the security of the symmetric encryption scheme. If the adversary knows all of the different symmetric encryption keys $K_i, i \in [n]$, 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 $k_{i,j,\ell}$ for each label ℓ . These keys can be locally derived from $k_{i,j}$ 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 [19], Fan and Tang proposed a new notion of distributed public key functional encryption, in which the key generation procedure generates n different shares $\{sk_i^f\}_{i \in [n]}$ 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 $s_i \leftarrow \text{Dec}(sk_i^f, ct)$ 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 $f(\mathbf{x}) \mapsto \sum_i^n x_i$ gets computed. This is similar to DMCFE for the inner-product functionality $f(\mathbf{x}) \mapsto \langle \mathbf{x}, \mathbf{y} \rangle$. PSA was introduced by Shi et al. [25] 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 [10, 14, 18, 21, 22], instead of providing new functionalities.

1.4 Concurrent Work

Concurrently and independently of our work, Chotard et al. [16] 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. [3] is actually an MCFE scheme that is secure under adaptive corruptions.

2 Definitions and Security Models

Notation. We use $[n]$ to denote the set $\{1, \dots, n\}$. We write \mathbf{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 $\text{Win}_{\mathcal{A}}^G(\lambda, n)$, which is $\Pr[G(\lambda, n, \mathcal{A}) = 1]$. 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: $\text{Adv}_{\mathcal{A}}^G(\lambda, n) = |\text{Win}_{\mathcal{A}}^{G_0}(\lambda, n) - \text{Win}_{\mathcal{A}}^{G_1}(\lambda, n)|$.

2.1 Multi-Client Functional Encryption

In this section, we define the notion of MCFE [20].

Definition 2.1. (Multi-Client Functional Encryption) Let $\mathcal{F} = \{\mathcal{F}_\rho\}_\rho$ be a family (indexed by ρ) of sets \mathcal{F}_ρ of functions $f: \mathcal{X}_{\rho,1} \times \dots \times \mathcal{X}_{\rho,n_\rho} \rightarrow \mathcal{Y}_\rho$.⁶ Let $\text{Labels} = \{0,1\}^*$ or $\{\perp\}$ 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 $\text{MCFE} = (\text{Setup}, \text{KeyGen}, \text{KeyDer}, \text{Enc}, \text{Dec})$:

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}_ρ of n -ary functions (i.e., $n = n_\rho$).
KeyGen(pp): Takes as input the public parameters pp and outputs n secret keys $\{\text{sk}_i\}_{i \in [n]}$ and a master secret key msk .
KeyDer(pp, 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 .
Enc($\text{pp}, \text{sk}_i, x_i, \ell$): 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 $\ell \in \text{Labels}$, and outputs ciphertext $\text{ct}_{i,\ell}$.
Dec($\text{pp}, \text{sk}_f, \text{ct}_{1,\ell}, \dots, \text{ct}_{n,\ell}$): 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 $\lambda, n \in \mathbb{N}$, $\text{pp} \leftarrow \text{Setup}(1^\lambda, 1^n)$, $f \in \mathcal{F}_\rho$, $\ell \in \text{Labels}$, $x_i \in \mathcal{X}_{\rho,i}$, when $(\{\text{sk}_i\}_{i \in [n]}, \text{msk}) \leftarrow \text{KeyGen}(\text{pp})$ and $\text{sk}_f \leftarrow \text{KeyDer}(\text{pp}, \text{msk}, f)$, we have

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

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

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 [3] 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).

⁶ All the functions inside the same set \mathcal{F}_ρ have the same domain and the same range.

As noted in [15, 20], 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 (one, any, 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 $\text{xx-yy-IND}_\beta^{\text{MCFE}}$ in Fig. 1, where the oracles are defined as:

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

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

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

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

- If $i \in \text{CS}$ (i.e., slot i is corrupted): for any query $\text{QEnc}(i, x_i^0, x_i^1, \ell)$, $x_i^0 = x_i^1$.
- For any label $\ell \in \text{Labels}$, for any family of queries $\{\text{QEnc}(i, x_i^0, x_i^1, \ell)\}_{i \in [n] \setminus \text{CS}}$, for any family of inputs $\{x_i \in \mathcal{X}_{\rho,i}\}_{i \in \text{CS}}$, for any query $\text{QKeyD}(f)$, we define $x_i^0 = x_i^1 = x_i$ for any slot $i \in \text{CS}$, $\mathbf{x}^b = (x_1^b, \dots, x_n^b)$ for $b \in \{0, 1\}$, and we require that:

$$f(\mathbf{x}^0) = f(\mathbf{x}^1) \text{ .}$$

We insist that if one index $i \notin \text{CS}$ is not queried for the label ℓ , there is no restriction.

- When $\text{yy} = \text{one}$: for any slot $i \in [n]$ and $\ell \in \text{Labels}$, $Q_{i,\ell} \in \{0, 1\}$, and if $Q_{i,\ell} = 1$, then for any slot $j \in [n] \setminus \text{CS}$, $Q_{j,\ell} = 1$. In other words, for any label, either the adversary makes no encryption query or makes exactly one encryption query for each $i \in [n] \setminus \text{CS}$.
- When $\text{yy} = \text{pos}$: for any slot $i \in [n]$ and $\ell \in \text{Labels}$, if $Q_{i,\ell} > 0$, then for any slot $j \in [n] \setminus \text{CS}$, $Q_{j,\ell} > 0$. In other words, for any label, either the adversary makes no encryption query or makes at least one encryption query for each slot $i \in [n] \setminus \text{CS}$.

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

$$\begin{aligned} \text{Adv}_{\text{MCFE}, \mathcal{A}}^{\text{xx-yy-IND}}(\lambda, n) = & \left| \Pr[\text{xx-yy-IND}_0^{\text{MCFE}}(\lambda, n, \mathcal{A}) = 1] \right. \\ & \left. - \Pr[\text{xx-yy-IND}_1^{\text{MCFE}}(\lambda, n, \mathcal{A}) = 1] \right| \text{ .} \end{aligned}$$

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

Fig. 1. Security games for MCFE

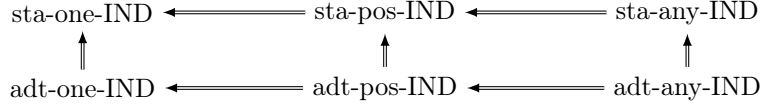


Fig. 2. Relations between the MCFE security notions (arrows indicate implication or being “a stronger security notion than”)

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: $\text{Adv}_{\text{MCFE}, \mathcal{A}}^{\text{xx-yy-IND}}(\lambda, n) \leq \text{negl}(\lambda)$.

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.

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, $\text{sta-any-IND} = \text{adt-any-IND} = \text{any-IND}$ corresponds to the secret-key version of the standard adaptive indistinguishability notion used in [5]. The security notions considered in [15] are actually xx-pos-IND and so are the MIFE notions of [3]. 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) [15]. 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}_ρ of functions $f: \mathcal{X}_{\rho,1} \times \dots \times \mathcal{X}_{\rho,n_\rho} \rightarrow \mathcal{Y}_\rho$. Let $\text{Labels} = \{0, 1\}^*$ or $\{\perp\}$ 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 $\text{DMCFE} = (\text{Setup}, \text{KeyGen}, \text{KeyDerShare}, \text{KeyDerComb}, \text{Enc}, \text{Dec})$:

$\text{Setup}(1^\lambda, 1^n)$ is defined as for MCFE in Definition 2.1.

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

KeyDerShare(pp, 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 $\text{sk}_{i,f}$.

KeyDerComb(pp, $\text{sk}_{1,f}, \dots, \text{sk}_{n,f}$): Takes as input the public parameters pp , n partial functional decryption keys $\text{sk}_{1,f}, \dots, \text{sk}_{n,f}$ and outputs the functional decryption key sk_f .

$\text{Enc}(\text{pp}, \text{sk}_i, x_i, \ell)$ is defined as for MCFE in Definition 2.1.

$\text{Dec}(\text{pp}, \text{sk}_f, \text{ct}_{1,\ell}, \dots, \text{ct}_{n,\ell})$ is defined as for MCFE in Definition 2.1.

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

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

We remark that there is no master secret key msk . Furthermore, similarly to [15], 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 [15], we do not differentiate encryption keys from secret keys. This is without loss of generality, as corruptions in [15] 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 ($\text{xx} \in \{\text{sta}, \text{adt}\}$ and $\text{yy} \in \{\text{one}, \text{any}, \text{pos}\}$) 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 $\text{QKeyD}(f)$: Computes $\text{sk}_{i,f} := \text{KeyDerShare}(\text{pp}, \text{sk}_i, f)$ for $i \in [n]$ and outputs $\{\text{sk}_{i,f}\}_{i \in [n]}$.

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 $\mathcal{F}_\rho = \{f_{\mathbf{y}_1, \dots, \mathbf{y}_n} : (\mathcal{R}^m)^n \rightarrow \mathcal{R}\}$ where

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

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

2.4 Symmetric Encryption

For our second compiler (Section 4.1), we make use of a symmetric encryption scheme $\text{SE} = (\text{Enc}_{\text{SE}}, \text{Dec}_{\text{SE}})$ that is indistinguishable secure under chosen plaintext attacks (IND-CPA) and whose keys are uniform strings in $\{0, 1\}^\lambda$ as defined by [9].

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

$\text{Dec}_{\text{SE}}(\mathbf{K}, \text{ct})$: Takes as input a key \mathbf{K} and a ciphertext ct to decrypt, and outputs a message x .

We denote with $\text{Adv}_{\text{SE}, \mathcal{A}}^{\text{IND-CPA}}(\lambda)$ the advantage of an adversary guessing β in the following game: the challenger picks $\mathbf{K} \leftarrow \{0, 1\}^\lambda$ and gives \mathcal{A} access to an encryption oracle $\text{QEnc}(x_i^0, x_i^1)$ that outputs $\text{ct} = \text{Enc}_{\text{SE}}(\mathbf{K}, x_i^\beta)$ on a query (x_i^0, x_i^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 [3, 15] 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 $\text{MCFE} = (\text{Setup}, \text{KeyGen}, \text{KeyDer}, \text{Enc}, \text{Dec})$ 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: $\text{sk}_i = (i, s_i, \{\mathbf{u}_i^k\}_{k \in [\kappa]}),$ where $s_i \in \{0, 1\}^*$, and $\mathbf{u}_i^k \in \mathbb{Z}_L^m$, and κ and m are positive integers implicitly depending on the public parameters pp .

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

- $\text{sk}_f \leftarrow \text{KeyDer}(\text{pp}, \text{msk}, f)$ outputs $\text{sk}_f = (\{s_{i,f}\}_{i \in [n]}, \{\text{dk}_f^k\}_{k \in [\kappa]})$, where $s_{i,f}$ is a (polynomial-time) function of pp , i , s_i , and f , while:

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

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

Without loss of generality for MCFE with the special key derivation property, we can suppose that $\text{msk} = \{\text{sk}_i\}_{i \in [n]}$. 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. [15, Section 4] satisfies the special key derivation property modulo $L = p$ (the order of the cyclic group), with $\kappa = 2$ and $\mathbf{y}_f^k = \mathbf{y}$, when $f : \mathbf{x} \mapsto \langle \mathbf{x}, \mathbf{y} \rangle$.

The generic constructions of Abdalla et al. [3, 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 $\mathbf{y}_f^k = \mathbf{y}$. The instantiations from MDDH, LWE, and Paillier ([3, 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:

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

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

We have the following security theorem.

$\text{Setup}'(1^\lambda, 1^n) :$ Return $\text{Setup}(1^\lambda, 1^n)$ $\text{KeyGen}'(\text{pp}) :$ $(\{\text{sk}_i\}_{i \in [n]}, \text{msk}) \leftarrow \text{KeyGen}(\text{pp})$ Recall that $\text{sk}_i = (i, s_i, \{\mathbf{u}_i^k\}_{k \in [\kappa]})$ For $k \in [\kappa]$: For $i \in [n-1]$, $\mathbf{v}_i^k \leftarrow \mathbb{Z}_L^M$ $\mathbf{v}_n^k := - \sum_{i=1}^{n-1} \mathbf{v}_i^k \bmod L$ Return $\{\text{sk}'_i = (\text{sk}_i, \{\mathbf{v}_i^k\}_{k \in [\kappa]})\}_{i \in [n]}$ $\text{Enc}'(\text{pp}, \text{sk}'_i, x_i, \ell) :$ Parse $\text{sk}'_i = (\text{sk}_i, \{\mathbf{v}_i^k\}_{k \in [\kappa]})$ Return $\text{ct}_{i,\ell} \leftarrow \text{Enc}(\text{pp}, \text{sk}_i, x_i, \ell)$	$\text{KeyDerShare}'(\text{pp}, \text{sk}'_i, f) :$ Parse $\text{sk}'_i = (\text{sk}_i, \{\mathbf{v}_i^k\}_{k \in [\kappa]})$ For $k \in [\kappa]$, $\text{dk}_{i,f}^k := \langle \mathbf{u}_i^k, \mathbf{y}_{i,f} \rangle + \langle \mathbf{v}_i^k, \mathbf{y}_f \rangle$ Return $\text{sk}'_{i,f} := (s_{i,f}, \{\text{dk}_{i,f}^k\}_{k \in [\kappa]})$ $\text{KeyDerComb}'(\text{pp}, \{\text{sk}'_{i,f}\}_{i \in [n]}) :$ Parse $\{\text{sk}'_{i,f} = (s_{i,f}, \{\text{dk}_{i,f}^k\}_{k \in [\kappa]})\}_{i \in [n]}$ For $k \in [\kappa]$, $\text{dk}_f^k := \sum_{i=1}^n \text{dk}_{i,f}^k$ Return $\text{sk}'_f = (\{s_{i,f}\}_{i \in [n]}, \{\text{dk}_f^k\}_{k \in [\kappa]})$ $\text{Dec}'(\text{pp}, \text{sk}'_f, \{\text{ct}_{i,\ell}\}_{i \in [n]}) :$ Return $\text{Dec}(\text{pp}, \text{sk}'_f, \{\text{ct}_{i,\ell}\}_{i \in [n]})$
--	---

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

Theorem 3.2. *Let $\text{MCFE} = (\text{Setup}, \text{KeyGen}, \text{KeyDer}, \text{Enc}, \text{Dec})$ 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 $\text{xx} \in \{\text{sta}, \text{adt}\}$ and any $\text{yy} \in \{\text{one}, \text{pos}, \text{any}\}$, 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:*

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

Below, we provide a proof sketch of the theorem. The formal proof is in the full version [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 $\text{sk}'_{i,f} = \text{dk}_{i,f} = \langle \mathbf{u}_i, \mathbf{y}_{i,f} \rangle + \langle \mathbf{v}_i, \mathbf{y}_f \rangle$. We can define $\mathbf{u}'_i \in \mathbb{Z}_L$ to be \mathbf{u}_i “padded with 0” so that we can write: $\langle \mathbf{u}_i, \mathbf{y}_{i,f} \rangle = \langle \mathbf{u}'_i, \mathbf{y}_f \rangle$ (recall that \mathbf{y}_f is just the concatenation of the vectors $\mathbf{y}_{i,f}$ for $i \in [n]$). Thus we have:

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

Now, we remark that from keys $\text{dk}_{i,g}$ for $g \in \{f_1, \dots, f_q\}$, one can compute the key $\text{dk}_{i,f}$ for any f such that \mathbf{y}_f is in the subspace generated by $\mathbf{y}_{f_1}, \dots, \mathbf{y}_{f_q}$.

Indeed, if $\mathbf{y} = \sum_{j=1}^q \mu_j \cdot \mathbf{y}_{f_j}$, for some $\mu_1, \dots, \mu_q \in \mathbb{Z}_L$, then: $\text{dk}_{i,f} = \sum_{j=1}^q \mu_j \cdot \text{dk}_{i,f_j}$.

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

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

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

Thus, we have a perfect reduction from the security of DMCFE' to the security of MCFE. \square

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 [3, 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 outputting a number $L \geq 2$. We define the experiment $\text{Factor}_{\text{GenL}}(\lambda, \mathcal{A})$ for an adversary \mathcal{A} as follows: it outputs 1 if on input $L \leftarrow \text{GenL}(1^\lambda)$, 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 $\text{Adv}_{\text{GenL}, \mathcal{A}}^{\text{Factor}}(\lambda) = \Pr[\text{Factor}_{\text{GenL}}(\lambda, \mathcal{A})]$. 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 the full version [1].

Theorem 3.4. Let $\text{MCFE} = (\text{Setup}, \text{KeyGen}, \text{KeyDer}, \text{Enc}, \text{Dec})$ 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 $L \leftarrow \text{GenL}(1^\lambda)$ in the setup, for some polynomial-time algorithm. We assume that factorization is hard for GenL . For any $\text{xx} \in \{\text{sta}, \text{adt}\}$ and any $\text{yy} \in \{\text{one}, \text{pos}, \text{any}\}$, 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:

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

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 ($\text{Labels} = \{\perp\}$) 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 $\text{DMCFE} = (\text{Setup}, \text{KeyGen}, \text{KeyDerShare}, \text{KeyDerComb}, \text{Enc}, \text{Dec})$ be an adt-pos-IND-secure DMCFE scheme without labels ($\text{Labels} = \{\perp\}$) for a family of functions \mathcal{F} . Let $\text{SE} = (\text{Enc}_{\text{SE}}, \text{Dec}_{\text{SE}})$ be an IND-CPA symmetric-key encryption scheme. Then the DMCFE scheme $\text{DMCFE}' = (\text{Setup}', \text{KeyGen}', \text{KeyDerShare}', \text{KeyDerComb}', \text{Enc}', \text{Dec}')$ 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:*

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

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.

$\text{Setup}'(1^\lambda, 1^n) :$ Return $\text{pp} \leftarrow \text{Setup}(1^\lambda, 1^n)$ $\text{KeyGen}'(\text{pp}) :$ $\{\text{sk}_i\}_{i \in [n]} \leftarrow \text{KeyGen}(\text{pp})$ For $i \in [n] :$ $\quad \text{k}_{i,1}, \dots, \text{k}_{i,n} \leftarrow \{0, 1\}^\lambda$ $\quad \text{K}_i = \oplus_{j \in [n]} \text{k}_{i,j}$ Return $\{\text{sk}'_i = (\text{sk}_i, \text{K}_i, \{\text{k}_{i,j}, \text{k}_{j,i}\}_{j \in [n]})\}_{i \in [n]}$ $\text{Enc}'(\text{pp}, \text{sk}'_i, x_i) :$ Parse $\text{sk}'_i = (\text{sk}_i, \text{K}_i, \{\text{k}_{i,j}, \text{k}_{j,i}\}_{j \in [n]})$ $\text{ct}_i \leftarrow \text{Enc}(\text{pp}, \text{sk}_i, x_i)$ $\text{ct}'_i \leftarrow \text{Enc}_{\text{SE}}(\text{K}_i, \text{ct}_i)$ Return $(\text{ct}'_i, \{\text{k}_{j,i}\}_{j \in [n]})$	$\text{KeyDerShare}'(\text{pp}, \text{sk}'_i, f) :$ Parse $\text{sk}'_i = (\text{sk}_i, \text{K}_i, \{\text{k}_{i,j}, \text{k}_{j,i}\}_{j \in [n]})$ Return $\text{sk}'_{i,f} \leftarrow \text{KeyDerShare}(\text{sk}_i, f)$ $\text{KeyDerComb}'(\text{pp}, \{\text{sk}'_{i,f}\}_{i \in [n]}) :$ $\text{sk}_f := \text{KeyDerComb}(\text{pp}, \{\text{sk}'_{i,f}\}_{i \in [n]})$ Return sk_f $\text{Dec}'(\text{pp}, \text{sk}_f, \text{ct}'_1, \dots, \text{ct}'_n) :$ Parse $\{\text{ct}'_i = (\text{ct}'_i, \{\text{k}_{j,i}\}_{j \in [n]})\}_{i \in [n]}$ For $i \in [n] :$ $\quad \text{K}_i = \oplus_{j \in [n]} \text{k}_{i,j}$ $\quad \text{ct}_i \leftarrow \text{Dec}_{\text{SE}}(\text{K}_i, \text{ct}'_i)$ Return $\text{Dec}(\text{pp}, \text{sk}_f, \text{ct}_1, \dots, \text{ct}_n)$
---	---

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

Lemma 4.2. *Let $\text{DMCFE} = (\text{Setup}, \text{KeyGen}, \text{KeyDerShare}, \text{KeyDerComb}, \text{Enc}, \text{Dec})$ be an adt-pos-IND-secure DMCFE construction without labels (Labels = $\{\perp\}$) for a family of functions \mathcal{F} . Let $\text{SE} = (\text{Enc}_{\text{SE}}, \text{Dec}_{\text{SE}})$ be a symmetric-key encryption scheme. Then the DMCFE scheme $\text{DMCFE}' = (\text{Setup}', \text{KeyGen}', \text{KeyDerShare}', \text{KeyDerComb}', \text{Enc}', \text{Dec}')$ 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:*

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

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

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

If \mathcal{A} asks a query $\text{QKeyD}'(f)$, \mathcal{B} forwards it to its own oracle to receive $\text{sk}_{i,f}$, 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:

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

□

We continue with the consideration of the second point.

Lemma 4.3. *Let $\text{DMCFE} = (\text{Setup}, \text{KeyGen}, \text{KeyDerShare}, \text{KeyDerComb}, \text{Enc}, \text{Dec})$ be a DMCFE construction without labels ($\text{Labels} = \{\perp\}$) for a family of functions \mathcal{F} . Let $\text{SE} = (\text{Enc}_{\text{SE}}, \text{Dec}_{\text{SE}})$ 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 $\text{DMCFE}' = (\text{Setup}', \text{KeyGen}', \text{KeyDerShare}', \text{KeyDerComb}', \text{Enc}', \text{Dec}')$ described in Fig. 4 is adt-any-IND -secure. Namely, for any PPT adversary \mathcal{A} , there exists an adversary \mathcal{B}' such that:*

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

Proof. We prove this part by using a hybrid argument. We define the games G_1, \dots, G_n in Fig. 5.

Due to the definition of the game it holds that: Game G_0 corresponds to the experiment $\text{adt-any-IND}_{\beta}^{\text{DMCFE}'}$ for $\beta = 1$ and G_n to the experiment $\text{adt-any-IND}_{\beta}^{\text{DMCFE}'}$ for $\beta = 0$ therefore using the triangular inequality, we get:

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

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

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

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 $k_{i,j}$ for all $j \in [n] \setminus (\mathcal{CS} \cup \mathcal{ES})$ such that $K_i = \bigoplus_{j \in [n]} k_{i,j}$. 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 $k_{j,i}$ for all $j \in [n] \setminus (\mathcal{CS} \cup \{i\})$. If the adversary \mathcal{A} asks a corruption query $\text{QCor}'(t)$, 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 $\text{QEnc}'(i, x_i^0, x_i^1)$ we consider three different cases. In the first case, \mathcal{A} queries the encryption oracle for $i < t$, then \mathcal{B}_t generates

```

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

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

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

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

```

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

$\text{Enc}_{\text{SE}}(\text{K}_i, \text{Enc}(\text{pp}, \text{sk}_i, x_i^0))$ 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. $\text{Enc}_{\text{SE}}(\text{K}_i, \text{Enc}(\text{pp}, \text{sk}_i, x_i^1))$. In the case that \mathcal{A} asks a query $\text{QEnc}'(t, x_t^0, x_t^1)$, \mathcal{B}_t generates $(\text{Enc}(\text{pp}, \text{sk}_t, x_t^0), \text{Enc}(\text{pp}, \text{sk}_t, x_t^1))$ and sends it to its own encryption oracle to receive $\text{Enc}_{\text{SE}}(\text{Enc}(\text{pp}, \text{sk}_t, x_t^\beta))$. 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 $\text{k}_{j,i}$ for all $j \in [n] \setminus \mathcal{CS}$. If $i \in (\mathcal{CS} \cup \mathcal{ES})$ then the values $\text{k}_{j,i}$ have already been sampled for all $j \in [n]$. The ciphertext $\text{Enc}_{\text{SE}}(\text{K}_i, \text{Enc}(\text{pp}, \text{sk}_i, x_i^\beta))$ together with $\text{k}_{j,i}, \forall j \in [n]$ are then sent to \mathcal{A} in the last step.

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

$\text{Setup}'(1^\lambda, 1^n) :$ Return $\text{pp} \leftarrow \text{Setup}(1^\lambda, 1^n)$ $\text{KeyGen}'(\text{pp}) :$ $\{\text{sk}_i\}_{i \in [n]} \leftarrow \text{KeyGen}(\text{pp})$ For $i \in [n] :$ $\mathbf{k}_{i,1}, \dots, \mathbf{k}_{i,n} \leftarrow \{0, 1\}^\lambda$ Return $\{\text{sk}'_i = (\text{sk}_i, \mathbf{K}_i, \{\mathbf{k}_{i,j}, \mathbf{k}_{j,i}\}_{j \in [n]})\}_{i \in [n]}$ $\text{Enc}'(\text{pp}, \text{sk}'_i, x_i, \ell) :$ Parse $\text{sk}'_i = (\text{sk}_i, \mathbf{K}_i, \{\mathbf{k}_{i,j}, \mathbf{k}_{j,i}\}_{j \in [n]})$ $\text{ct}_i \leftarrow \text{Enc}(\text{pp}, \text{sk}_i, x_i)$ For $j \in [n] :$ $\mathbf{k}_{i,j,\ell} := \text{H}_1(\mathbf{k}_{i,j} \ i \ j \ \ell)$ $\mathbf{k}_{j,i,\ell} := \text{H}_1(\mathbf{k}_{j,i} \ j \ i \ \ell)$ $\mathbf{K}_{i,\ell} := \bigoplus_{j \in [n]} \mathbf{k}_{i,j,\ell}$ $r_i \leftarrow \{0, 1\}^\lambda; \text{ct}'_i := \text{ct}_i \oplus \text{H}_2(\mathbf{K}_{i,\ell} \ r_i)$ Return $(\text{ct}'_i, r_i, \{\mathbf{k}_{j,i,\ell}\}_{j \in [n]})$	$\text{KeyDerShare}'(\text{pp}, \text{sk}'_i, f) :$ Parse $\text{sk}'_i = (\text{sk}_i, \mathbf{K}_i, \{\mathbf{k}_{i,j}, \mathbf{k}_{j,i}\}_{j \in [n]})$ Return $\text{sk}'_{i,f} \leftarrow \text{KeyDerShare}(\text{pp}, \text{sk}_i, f)$ $\text{KeyDerComb}'(\text{pp}, \{\text{sk}'_{i,f}\}_{i \in [n]}) :$ $\text{sk}_f := \text{KeyDerComb}(\text{pp}, \{\text{sk}_{i,f}\}_{i \in [n]})$ Return sk_f $\text{Dec}'(\text{pp}, \text{sk}_f, \text{ct}'_1, \dots, \text{ct}'_n) :$ Parse $\{\text{ct}'_i = (\text{ct}'_i, r_i, \{\mathbf{k}_{j,i,\ell}\}_{j \in [n]})\}_{i \in [n]}$ For $i \in [n] :$ $\mathbf{K}_{i,\ell} = \bigoplus_{j \in [n]} \mathbf{k}_{i,j,\ell}$ $\text{ct}_i = \text{ct}'_i \oplus \text{H}_2(\mathbf{K}_{i,\ell} \ r_i)$ Return $\text{Dec}(\text{pp}, \text{sk}_f, \text{ct}_1, \dots, \text{ct}_n)$.
---	---

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

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

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

This results in:

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

□

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 $\text{H}_1 : \{0, 1\}^* \rightarrow \{0, 1\}^\lambda$ and $\text{H}_2 : \{0, 1\}^* \rightarrow \{0, 1\}^{|\text{ct}_i|}$ are two hash functions modeled as random oracles in the security proof. We formally prove the following security theorem in the full version [1].

Theorem 4.4. *Let $\text{DMCFE} = (\text{Setup}, \text{KeyGen}, \text{KeyDerShare}, \text{KeyDerComb}, \text{Enc}, \text{Dec})$ be an adt-pos-IND -secure DMCFE scheme for an ensemble of functions \mathcal{F} and set of labels Labels . Then the DMCFE scheme $\text{DMCFE}' = (\text{Setup}', \text{KeyGen}', \text{KeyDerShare}', \text{KeyDerComb}', \text{Enc}', \text{Dec}')$ 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:*

$$\begin{aligned} \text{Adv}_{\text{DMCFE}', \mathcal{A}}^{\text{adt-any-IND}}(\lambda, n) &\leq \text{Adv}_{\text{DMCFE}, \mathcal{B}}^{\text{adt-pos-IND}}(\lambda, n) \\ &\quad + \frac{2q_{H_1} + (2n + 1) \cdot (q_{H_2} q_{\text{QEnc}} + q_{\text{QEnc}}^2)}{2^\lambda}, \end{aligned}$$

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.

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. [3] 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 way⁸, 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}_ρ , $\rho = (\mathbb{Z}_L, n, m, L, L)$. Finally, Section 5.3 recalls the MCFE construction from [3].

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

The [3] 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}_ρ , $\rho = (\mathbb{Z}, 1, m, X, Y)$ satisfying two properties, called *two-step decryption* and *linear encryption* [3]. As indicated in [3], 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 $\langle \mathbf{x}, \mathbf{y} \rangle$ as long as the bounds $\|\mathbf{x}\|_\infty < X$, $\|\mathbf{y}\|_\infty < Y$ 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 [3] 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 [3]). A secret-key FE scheme $\text{FE} = (\text{Setup}, \text{KeyGen}, \text{KeyDer}, \text{Enc}, \text{Dec})$ 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 Setup^* , $\text{Dec}_1, \text{Dec}_2$ and an encoding function \mathcal{E} such that:

1. For all $\lambda \in \mathbb{N}$, $\text{Setup}^*(1^\lambda, 1^n)$ 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 $\mathcal{E} : \mathbb{Z}_L \times \mathbb{Z} \rightarrow \mathbb{G}$.
2. For all $\text{msk} \leftarrow \text{KeyGen}(\text{pp}), \mathbf{x} \in \mathbb{Z}^m, \text{ct} \leftarrow \text{Enc}(\text{pp}, \text{msk}, \mathbf{x}), \mathbf{y} \in \mathbb{Z}^m$, and $\text{sk} \leftarrow \text{KeyDer}(\text{msk}, \mathbf{y})$, we have

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

for some $\text{noise} \in \mathbb{N}$ that depends on ct and sk . Furthermore, it holds that $\Pr[\text{noise} < B] = 1 - \text{negl}(\lambda)$, where the probability is taken over the random coins of KeyGen and KeyDer . Note that there is no restriction on the norm of $\langle \mathbf{x}, \mathbf{y} \rangle$ 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 $\gamma, \gamma' \in \mathbb{Z}_L, \text{noise}, \text{noise}' \in \mathbb{Z}$, we have

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

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

Definition 5.2 (Linear encryption [3]). A secret-key FE scheme $\text{FE} = (\text{Setup}, \text{KeyGen}, \text{KeyDer}, \text{Enc}, \text{Dec})$ 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 $\mathbf{x}, \mathbf{x}' \in \mathbb{Z}^m$, the following are identically distributed:

$$\text{Add}(\text{Enc}(\text{pp}, \text{msk}, \mathbf{x}), \mathbf{x}'), \text{ and } \text{Enc}(\text{pp}, \text{msk}, (\mathbf{x} + \mathbf{x}' \bmod L)) .$$

Recall that the value $L \in \mathbb{N}$ is defined as part of the output of the algorithm Setup^* (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. [3] in Fig. 7, to which we made the following modifications. First, our description does not need a setup procedure Setup^{ot} , which now simply defines (n, m, L) . Second, the steps of the original Setup^{ot} in Abdalla et al. [3] are now defined in the $\text{KeyGen}^{\text{ot}}$ 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 [3] for a proof of correctness. As for its security with respect to adaptive corruptions, we need to modify the proof of Abdalla et al. [3] to account for corruption queries.

Define $\text{pp}_{\text{ot}} = (n, m, L)$ <u>KeyGen^{ot}(pp_{ot}) :</u> $\{\mathbf{u}_i\}_{i \in [n]} \leftarrow (\mathbb{Z}_L^m)^n$ Return $\text{msk} := \{\text{msk}_i\}_{i \in [n]} = \{\mathbf{u}_i\}_{i \in [n]}$ <u>Enc^{ot}(pp_{ot}, msk_i, \mathbf{x}_i) :</u> Parse $\text{msk}_i = \mathbf{u}_i$ Return $\text{ct}_i := \mathbf{u}_i + \mathbf{x}_i \bmod L$	<u>KeyDer^{ot}(pp_{ot}, msk, \mathbf{y}) :</u> Parse $\text{msk} = \{\mathbf{u}_i\}_{i \in [n]}, \mathbf{y} = (\mathbf{y}_1, \dots, \mathbf{y}_n)$ Return $\text{dk}_{\mathbf{y}} := \sum_{i \in [n]} \langle \mathbf{u}_i, \mathbf{y}_i \rangle$ <u>Dec^{ot}(pp_{ot}, dk_y, \mathbf{y}, $\{\text{ct}_i\}_{i \in [n]}$) :</u> Parse $\mathbf{y} = (\mathbf{y}_1, \dots, \mathbf{y}_n)$ Return $\sum_{i \in [n]} \langle \text{ct}_i, \mathbf{y}_i \rangle - \text{dk}_{\mathbf{y}} \bmod L$
---	---

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

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

Proof. Let \mathcal{A} be an adversary against the adt-one-IND security of the MCFE^{ot} scheme with advantage $\text{Adv}_{\text{MCFE}^{\text{ot}}, \mathcal{A}}^{\text{adt-one-IND}}(\lambda)$. Let $\text{sta-one-sel-IND}_{\beta}^{\text{MCFE}^{\text{ot}}}(\lambda, n, \mathcal{B})$ be a variant of the $\text{sta-one-IND}_{\beta}^{\text{MCFE}^{\text{ot}}}(\lambda, n, \mathcal{B})$ experiment in which the selective adversary \mathcal{B} additionally specifies the encryption challenges $\{\mathbf{x}_i^b\}_{i \in [n], b \in \{0,1\}}$ 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:

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

After adversary \mathcal{B} made its guesses $\{\mathbf{x}_i^b\}_{i \in [n], b \in \{0,1\}}$ 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 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 MCFE^{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 MCFE^{ot} satisfies perfect sta-one-sel-IND security (i.e., $\text{Adv}_{\text{MCFE}^{\text{ot}}, \mathcal{B}}^{\text{sta-one-sel-IND}}(\lambda) = 0$), 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 $\text{sta-one-sel-IND}_{\beta}^{\text{MCFE}^{\text{ot}}}(\lambda, n, \mathcal{B})$. This can be seen by using the fact that, in the selective security game, all $\{\mathbf{x}_i^{\beta} \in \mathbb{Z}_L^m\}_{i \in [n]}$ have identical distributions: $\{\mathbf{u}_i \bmod L\}_{i \in [n]}$ and $\{\mathbf{u}_i - \mathbf{x}_i^{\beta} \bmod L\}$, with $\mathbf{u}_i \leftarrow_R \mathbb{Z}_L^m$. This also holds for the corrupted positions $i \in \mathcal{CS}$, because in this case it holds that $\mathbf{x}_i^0 = \mathbf{x}_i^1$.

$\mathcal{H}_\beta(1^\lambda, \mathcal{B})$	$\mathcal{O}_K(i, \mathbf{y})$
$(\mathcal{CS}, \{\mathbf{x}_i^b\}_{i \in [n], b \in \{0,1\}}) \leftarrow \mathcal{B}(1^\lambda, 1^n)$	Parse $\mathbf{y} = (\mathbf{y}_1, \dots, \mathbf{y}_n)$
For $i \in [n]$,	$\mathbf{sk}_\mathbf{y} = \sum_{i \in [n]} \langle \mathbf{u}_i - \mathbf{x}_i^\beta, \mathbf{y}_i \rangle$
$\mathbf{u}_i \leftarrow \mathbb{Z}_L^m; \mathbf{ct}_i \leftarrow \mathbf{u}_i$	Return $\mathbf{sk}_\mathbf{y}$
$\alpha \leftarrow \mathcal{B}^{\mathcal{O}_K(\cdot)}(\{\mathbf{u}_i\}_{i \in \mathcal{CS}}, \{\mathbf{ct}_i\}_{i \in [n]})$	
Output α	

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

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 $\langle \mathbf{x}_i^\beta, \mathbf{y}_i \rangle$, which is independent of β by the definition of the security game. \square

5.3 Inner-Product MCFE over \mathbb{Z}

In Fig. 9, we recall the construction of [3] of a pos-IND-secure scheme MCFE = (Setup, KeyGen, KeyDer, Enc, Dec) from the (one-IND-secure) MCFE scheme $\text{MCFE}^{\text{ot}} = (\text{KeyGen}^{\text{ot}}, \text{KeyDer}^{\text{ot}}, \text{Enc}^{\text{ot}}, \text{Dec}^{\text{ot}})$ described in Section 5.2 and from any any-IND-secure scheme FE = (Setup^{si}, KeyGen^{si}, KeyDer^{si}, Enc^{si}, Dec^{si}) for a single input. As for the one-time scheme in Section 5.2, we also modified the KeyGen procedure in [3] 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 [3] 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 the multi-client scheme MCFE^{ot} is adt-one-IND-secure. Then the multi-client scheme MCFE is adt-pos-IND-secure. Namely, for any PPT adversary \mathcal{A} , there exist PPT adversaries \mathcal{B} and \mathcal{B}' such that*

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

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 $\text{adt-pos-IND}_0^{\text{MCFE}}(\lambda, n, \mathcal{A})$ game and G_2 the $\text{adt-pos-IND}_1^{\text{MCFE}}(\lambda, n, \mathcal{A})$ 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 $\text{Win}_{\mathcal{A}}^{G_i}(\lambda, n)$, which is $\Pr[G_i(\lambda, n, \mathcal{A}) = 1]$. 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 $(\mathbf{x}_i^{0,j}, \mathbf{x}_i^{1,j})$.

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*

$$|\text{Win}_{\mathcal{A}}^{G_0}(\lambda, n) - \text{Win}_{\mathcal{A}}^{G_1}(\lambda, n)| \leq \text{Adv}_{\text{MCFE}^{\text{ot}}, \mathcal{B}}^{\text{adt-one-IND}}(\lambda, n) .$$

Setup($1^\lambda, 1^n$) :

$\text{pp}_{\text{si}} \leftarrow \text{Setup}^{\text{si}}(1^\lambda, 1^n)$
 Set $\text{pp}_{\text{ot}} := (n, m, L)$, with $\rho_{\text{si}} = (\mathbb{Z}, 1, m, 3X, Y)$ and L implicitly defined from pp_{si}
 Return $\text{pp} = (\text{pp}_{\text{si}}, \text{pp}_{\text{ot}})$

KeyGen(pp) :

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

Enc($\text{pp}, \text{sk}_i, \mathbf{x}_i$) :

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

KeyDer($\text{pp}, \text{msk}, \mathbf{y}$) :

Parse $\text{msk} = \{\text{msk}_i^{\text{si}}, \mathbf{u}_i\}_{i \in [n]}$, $\mathbf{y} = (\mathbf{y}_1, \dots, \mathbf{y}_n)$
 For $i \in [n]$, $\text{sk}_{i,\mathbf{y}} \leftarrow \text{KeyDer}^{\text{si}}(\text{pp}_{\text{si}}, \text{msk}_i^{\text{si}}, \mathbf{y}_i)$
 $\text{dk}_{\mathbf{y}} := \text{KeyDer}^{\text{ot}}(\text{pp}_{\text{ot}}, \{\mathbf{u}_i\}_{i \in [n]}, \mathbf{y})$
 Return $\text{sk}_{\mathbf{y}} := (\{\text{sk}_{i,\mathbf{y}}\}_{i \in [n]}, \text{dk}_{\mathbf{y}})$

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

Parse $\text{sk}_{\mathbf{y}} = (\{\text{sk}_{i,\mathbf{y}}\}_{i \in [n]}, \text{dk}_{\mathbf{y}})$
 For $i \in [n]$, $\mathcal{E}(\langle \mathbf{u}_i + \mathbf{x}_i, \mathbf{y}_i \rangle \bmod L, \text{noise}_i) \leftarrow \text{Dec}_1^{\text{si}}(\text{pp}_{\text{si}}, \text{sk}_{i,\mathbf{y}}, \text{ct}_i)$
 Return $\text{Dec}_2^{\text{si}}(\text{pp}_{\text{si}}, \mathcal{E}(\langle \mathbf{u}_1 + \mathbf{x}_1, \mathbf{y}_1 \rangle \bmod L, \text{noise}_1)) \circ \dots$
 $\circ \mathcal{E}(\langle \mathbf{u}_n + \mathbf{x}_n, \mathbf{y}_n \rangle \bmod L, \text{noise}_n) \circ \mathcal{E}(-\text{dk}_{\mathbf{y}}, 0)$

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

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

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

Proof. Compared to G_0 , G_1 replaces the encryptions of $\mathbf{x}_i^{0,j} - \mathbf{x}_i^{0,1} + \mathbf{x}_i^{1,1}$ with the encryptions of $\mathbf{x}_i^{0,j} - \mathbf{x}_i^{0,1} + \mathbf{x}_i^{1,1}$ 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 $\text{adt-one-IND}_\beta^{\text{MCFE}}$ experiment. In the beginning \mathcal{B} generates the parameters $\text{pp} = (\text{pp}_{\text{si}}, \text{pp}_{\text{ot}}) \leftarrow \text{Setup}(1^\lambda, 1^n)$ and the keys $\text{msk}_i^{\text{si}} \leftarrow \text{KeyGen}(\text{pp}_{\text{si}})$ for all the positions $i \in [n]$. Whenever \mathcal{A} asks a query $\text{QKeyD}'(\mathbf{y} = (\mathbf{y}_1, \dots, \mathbf{y}_n))$, \mathcal{B} uses its own key derivation oracle to get $\text{dk}_{\mathbf{y}} = \sum_{i \in [n]} \langle \mathbf{u}_i, \mathbf{y}_i \rangle$ and computes the keys $\text{sk}_{i,\mathbf{y}} \leftarrow \text{KeyDer}^{\text{si}}(\text{pp}_{\text{si}}, \text{msk}_i^{\text{si}}, \mathbf{y}_i)$ 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 $\text{QEnc}'(i, \mathbf{x}_i^{0,1}, \mathbf{x}_i^{1,1})$ by \mathcal{A} gets forwarded to the challenger. \mathcal{B} receives $\text{ct}_{i,\text{ot}} = \mathbf{u}_i + \mathbf{x}_i^{\beta,1}$ as an answer, computes $\text{ct}_i^1 = \text{Enc}^{\text{si}}(\text{pp}_{\text{si}}, \text{msk}_i^{\text{si}}, \mathbf{u}_i + \mathbf{x}_i^{\beta,1})$, and returns it to \mathcal{A} . For all further queries ($j > 1$), \mathcal{B} produces ct_i^j by encrypting $(\mathbf{x}_i^{0,j} - \mathbf{x}_i^{0,1} + \text{ct}_{i,\text{ot}}) \bmod L$.

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

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

$$|\text{Win}_{\mathcal{A}}^{G_0}(\lambda, n) - \text{Win}_{\mathcal{A}}^{G_1}(\lambda, n)| \leq \text{Adv}_{\text{MCFE}^{\text{ot}}, \mathcal{B}}^{\text{adt-one-IND}}(\lambda, n) .$$

□

In the next step we consider game G_2 . In this game, we change the encryption from $\text{Enc}^{\text{si}}(\text{pp}_{\text{si}}, \text{msk}_i^{\text{si}}, \mathbf{x}_i^{0,j} - \mathbf{x}_i^{0,1} + \mathbf{u}_i + \mathbf{x}_i^{1,1})$ to $\text{Enc}^{\text{si}}(\text{pp}_{\text{si}}, \text{msk}_i^{\text{si}}, \mathbf{x}_i^{1,j} - \mathbf{x}_i^{1,1} + \mathbf{u}_i + \mathbf{x}_i^{1,1})$ 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

$$|\text{Win}_{\mathcal{A}}^{G_1}(\lambda, n) - \text{Win}_{\mathcal{A}}^{G_2}(\lambda, n)| = \sum_{k=1}^n |\text{Win}_{\mathcal{A}}^{G_{1,k-1}}(\lambda, n) - \text{Win}_{\mathcal{A}}^{G_{1,k}}(\lambda, n)| ,$$

where G_1 corresponds to game $G_{1,0}$ and whereas G_2 is identical to game $G_{1,n}$.

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*

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

Proof. $G_{1,k}$ replaces the encryption of $\mathbf{x}_i^{0,j} - \mathbf{x}_i^{0,1} + \mathbf{x}_i^{1,1}$ with encryptions of $\mathbf{x}_i^{1,j} - \mathbf{x}_i^{1,1} + \mathbf{x}_i^{1,1}$ in all slots, for $i \leq k$. As already described in the preliminaries, it must hold that $\langle \mathbf{x}_i^{0,j} - \mathbf{x}_i^{0,1}, \mathbf{y}_i \rangle = \langle \mathbf{x}_i^{1,j} - \mathbf{x}_i^{1,1}, \mathbf{y}_i \rangle$ for all queries. Hence $\langle \mathbf{x}_i^{0,j} - \mathbf{x}_i^{0,1} + \mathbf{x}_i^{1,1}, \mathbf{y}_i \rangle = \langle \mathbf{x}_i^{1,j} - \mathbf{x}_i^{1,1} + \mathbf{x}_i^{1,1}, \mathbf{y}_i \rangle$, and since $\|\mathbf{x}_i^{0,j} - \mathbf{x}_i^{0,1} + \mathbf{x}_i^{1,1}\|_\infty <$

$3X$ and $\|\mathbf{x}_i^{1,j} - \mathbf{x}_i^{1,1} + \mathbf{x}_i^{1,1}\|_\infty < 3X$, 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 any-IND $_{\beta}^{\text{FE}}$ 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 mpk_k^{si} , corresponding to the k -th encryption instance. In the next step, \mathcal{B}_k randomly chooses $\mathbf{u}_i \in \mathbb{Z}_L^m$ for all $i \in [n]$ and runs the KeyGen procedure to get msk_i^{si} for all $i \neq k$.

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

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

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

As in the security proof of the MIFE scheme in [3], we remark that by the two-step property Definition 5.2, ct_k^j is identically distributed to $\text{Enc}^{\text{si}}(\text{pp}_{\text{si}}, \text{msk}_k^{\text{si}}, \mathbf{x}_k^{j,\beta} - \mathbf{x}_k^{1,\beta} + \mathbf{x}_k^{1,1} + \mathbf{u}_k \bmod L)$, which is itself equal to $\text{Enc}^{\text{si}}(\text{pp}_{\text{si}}, \text{msk}_k^{\text{si}}, \text{Enc}^{\text{ot}}(\mathbf{x}_k^{j,\beta} - \mathbf{x}_k^{1,\beta} + \mathbf{x}_k^{1,1}))$.

In the case that the adversary \mathcal{A} asks a corruption query $\text{QCor}'(k)$ 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 $\mathbf{x}_k^{1,0} = \mathbf{x}_k^{1,1}$. 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 $\text{QCor}'(i)$ for $i \neq k$, \mathcal{B}_k simply returns $(\text{msk}_i^{\text{si}}, \mathbf{u}_i)$ 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} :

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

□

⁹ As in [3], 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 $\langle \mathbf{x}_i^{0,j} - \mathbf{x}_i^{0,1}, \mathbf{y}_i \rangle = \langle \mathbf{x}_i^{1,j} - \mathbf{x}_i^{1,1}, \mathbf{y}_i \rangle$.

$G_0(1^\lambda, \mathcal{A}), \boxed{G_1(1^\lambda, \mathcal{A})}, \boxed{G_2(1^\lambda, \mathcal{A})} :$ $pp \leftarrow \text{Setup}(1^\lambda, 1^n)$ $\{sk_i\}_{i \in [n]} \leftarrow \text{KeyGen}(pp)$ $\alpha \leftarrow \mathcal{A}^{\text{QEnc}'(\cdot, \cdot, \cdot), \text{QKeyD}'(\cdot), \text{QCor}'(\cdot)}(pp)$ Output: α $\text{QEnc}'(i, x^0, x^1)$ Return $\text{Enc}(pp, sk_i, x_i^{0,j} - x_i^{0,1} + x_i^{0,1})$ <div style="border: 1px solid black; padding: 2px; display: inline-block;">Return $\text{Enc}(pp, sk_i, x_i^{0,j} - x_i^{0,1} + x_i^{1,1})$</div> <div style="border: 1px dashed black; padding: 2px; display: inline-block;">Return $\text{Enc}(pp, sk_i, x_i^{1,j} - x_i^{1,1} + x_i^{1,1})$</div> $\text{QKeyD}'(y)$ $dk_y = \sum_{i \in [n]} \langle u_i, y_i \rangle,$ For $i \in [n]$, $sk_{i,y} \leftarrow \text{KeyDer}^{si}(pp, msk_i^{si}, y_i)$ Return $(\{sk_{i,y}\}_{i \in [n]}, dk_y)$	$G_{1,k}(1^\lambda, \mathcal{A}) :$ $pp \leftarrow \text{Setup}(1^\lambda, 1^n)$ $\{sk_i\}_{i \in [n]} \leftarrow \text{KeyGen}(pp)$ $\alpha \leftarrow \mathcal{A}^{\text{QEnc}'(\cdot, \cdot, \cdot), \text{QKeyD}'(\cdot), \text{QCor}'(\cdot)}(pp)$ Output: α $\text{QEnc}'(i, x^0, x^1)$ If $i \leq k$ return $\text{Enc}(pp, sk_i, x_i^{1,j} - x_i^{1,1} + x_i^{1,1})$ If $i > k$ return $\text{Enc}(pp, sk_i, x_i^{0,j} - x_i^{0,1} + x_i^{1,1})$ $\text{QKeyD}'(y)$ $dk_y = \sum_{i \in [n]} \langle u_i, y_i \rangle,$ For $i \in [n]$, $sk_{i,y} \leftarrow \text{KeyDer}^{si}(pp, msk_i^{si}, y_i)$ Return $(\{sk_{i,y}\}_{i \in [n]}, dk_y)$
---	---

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

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. \square

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

1. Abdalla, M., Benhamouda, F., Kohlweiss, M., Waldner, H.: Decentralizing inner-product functional encryption. Cryptology ePrint Archive, Report 2019/020 (2019), <http://eprint.iacr.org/2019/020>
2. Abdalla, M., Bourse, F., De Caro, A., Pointcheval, D.: Simple functional encryption schemes for inner products. In: Katz, J. (ed.) PKC 2015. LNCS, vol. 9020, pp.

- 733–751. Springer, Heidelberg (Mar / Apr 2015). https://doi.org/10.1007/978-3-662-46447-2_33
3. Abdalla, M., Catalano, D., Fiore, D., Gay, R., Ursu, B.: Multi-input functional encryption for inner products: Function-hiding realizations and constructions without pairings. In: Shacham, H., Boldyreva, A. (eds.) CRYPTO 2018, Part I. LNCS, vol. 10991, pp. 597–627. Springer, Heidelberg (Aug 2018). https://doi.org/10.1007/978-3-319-96884-1_20
 4. Abdalla, M., Gay, R., Raykova, M., Wee, H.: Multi-input inner-product functional encryption from pairings. In: Coron, J., Nielsen, J.B. (eds.) EUROCRYPT 2017, Part I. LNCS, vol. 10210, pp. 601–626. Springer, Heidelberg (Apr / May 2017). https://doi.org/10.1007/978-3-319-56620-7_21
 5. Agrawal, S., Libert, B., Stehlé, D.: Fully secure functional encryption for inner products, from standard assumptions. In: Robshaw, M., Katz, J. (eds.) CRYPTO 2016, Part III. LNCS, vol. 9816, pp. 333–362. Springer, Heidelberg (Aug 2016). https://doi.org/10.1007/978-3-662-53015-3_12
 6. Ananth, P., Jain, A.: Indistinguishability obfuscation from compact functional encryption. In: Gennaro, R., Robshaw, M.J.B. (eds.) CRYPTO 2015, Part I. LNCS, vol. 9215, pp. 308–326. Springer, Heidelberg (Aug 2015). https://doi.org/10.1007/978-3-662-47989-6_15
 7. Badrinarayanan, S., Gupta, D., Jain, A., Sahai, A.: Multi-input functional encryption for unbounded arity functions. In: Iwata, T., Cheon, J.H. (eds.) ASIACRYPT 2015, Part I. LNCS, vol. 9452, pp. 27–51. Springer, Heidelberg (Nov / Dec 2015). https://doi.org/10.1007/978-3-662-48797-6_2
 8. Baltico, C.E.Z., Catalano, D., Fiore, D., Gay, R.: Practical functional encryption for quadratic functions with applications to predicate encryption. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017, Part I. LNCS, vol. 10401, pp. 67–98. Springer, Heidelberg (Aug 2017). https://doi.org/10.1007/978-3-319-63688-7_3
 9. Bellare, M., Desai, A., Jokipii, E., Rogaway, P.: A concrete security treatment of symmetric encryption. In: 38th FOCS. pp. 394–403. IEEE Computer Society Press (Oct 1997). <https://doi.org/10.1109/SFCS.1997.646128>
 10. Benhamouda, F., Joye, M., Libert, B.: A new framework for privacy-preserving aggregation of time-series data. ACM Trans. Inf. Syst. Secur. **18**(3), 10:1–10:21 (2016). <https://doi.org/10.1145/2873069>, <http://doi.acm.org/10.1145/2873069>
 11. Bishop, A., Jain, A., Kowalczyk, L.: Function-hiding inner product encryption. In: Iwata, T., Cheon, J.H. (eds.) ASIACRYPT 2015, Part I. LNCS, vol. 9452, pp. 470–491. Springer, Heidelberg (Nov / Dec 2015). https://doi.org/10.1007/978-3-662-48797-6_20
 12. Boneh, D., Sahai, A., Waters, B.: Functional encryption: Definitions and challenges. In: Ishai, Y. (ed.) TCC 2011. LNCS, vol. 6597, pp. 253–273. Springer, Heidelberg (Mar 2011). https://doi.org/10.1007/978-3-642-19571-6_16
 13. Brakerski, Z., Komargodski, I., Segev, G.: Multi-input functional encryption in the private-key setting: Stronger security from weaker assumptions. Journal of Cryptology **31**(2), 434–520 (Apr 2018). <https://doi.org/10.1007/s00145-017-9261-0>
 14. Chan, T.H.H., Shi, E., Song, D.: Privacy-preserving stream aggregation with fault tolerance. In: Keromytis, A.D. (ed.) FC 2012. LNCS, vol. 7397, pp. 200–214. Springer, Heidelberg (Feb / Mar 2012)
 15. Chotard, J., Dufour Sans, E., Gay, R., Phan, D.H., Pointcheval, D.: Decentralized multi-client functional encryption for inner product. In: Peyrin, T., Galbraith, S. (eds.) ASIACRYPT 2018, Part II. LNCS, vol. 11273, pp. 703–732. Springer, Heidelberg (Dec 2018). https://doi.org/10.1007/978-3-030-03329-3_24

16. Chotard, J., Dufour Sans, E., Gay, R., Phan, D.H., Pointcheval, D.: Multi-client functional encryption with repetition for inner product. Cryptology ePrint Archive, Report 2018/1021 (2018), <http://eprint.iacr.org/2018/1021>
17. Datta, P., Dutta, R., Mukhopadhyay, S.: Functional encryption for inner product with full function privacy. In: Cheng, C.M., Chung, K.M., Persiano, G., Yang, B.Y. (eds.) PKC 2016, Part I. LNCS, vol. 9614, pp. 164–195. Springer, Heidelberg (Mar 2016). https://doi.org/10.1007/978-3-662-49384-7_7
18. Emura, K.: Privacy-preserving aggregation of time-series data with public verifiability from simple assumptions. In: Pieprzyk, J., Suriadi, S. (eds.) Information Security and Privacy - 22nd Australasian Conference, ACISP. Lecture Notes in Computer Science, vol. 10343, pp. 193–213. Springer, Heidelberg, Germany, Auckland, New Zealand (2017). https://doi.org/10.1007/978-3-319-59870-3_11
19. Fan, X., Tang, Q.: Making public key functional encryption function private, distributively. In: Abdalla, M., Dahab, R. (eds.) PKC 2018, Part II. LNCS, vol. 10770, pp. 218–244. Springer, Heidelberg (Mar 2018). https://doi.org/10.1007/978-3-319-76581-5_8
20. Goldwasser, S., Gordon, S.D., Goyal, V., Jain, A., Katz, J., Liu, F.H., Sahai, A., Shi, E., Zhou, H.S.: Multi-input functional encryption. In: Nguyen, P.Q., Oswald, E. (eds.) EUROCRYPT 2014. LNCS, vol. 8441, pp. 578–602. Springer, Heidelberg (May 2014). https://doi.org/10.1007/978-3-642-55220-5_32
21. Joye, M., Libert, B.: A scalable scheme for privacy-preserving aggregation of time-series data. In: Sadeghi, A.R. (ed.) FC 2013. LNCS, vol. 7859, pp. 111–125. Springer, Heidelberg (Apr 2013). https://doi.org/10.1007/978-3-642-39884-1_10
22. Li, Q., Cao, G.: Efficient and privacy-preserving data aggregation in mobile sensing. In: 20th IEEE International Conference on Network Protocols, ICNP. pp. 1–10. IEEE Computer Society, Austin, TX, USA (2012). <https://doi.org/10.1109/ICNP.2012.6459985>
23. O’Neill, A.: Definitional issues in functional encryption. Cryptology ePrint Archive, Report 2010/556 (2010), <http://eprint.iacr.org/2010/556>
24. Sahai, A., Waters, B.R.: Fuzzy identity-based encryption. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 457–473. Springer, Heidelberg (May 2005). https://doi.org/10.1007/11426639_27
25. Shi, E., Chan, T.H.H., Rieffel, E.G., Chow, R., Song, D.: Privacy-preserving aggregation of time-series data. In: NDSS 2011. The Internet Society (Feb 2011)