Decentralized Multi-Client Functional Encryption for Inner Product

Jérémy Chotard[1,2,3], Edouard Dufour Sans[2,3], Duong Hieu Phan[1], and David Pointcheval[2,3]

[1] XLIM, University of Limoges, CNRS
[2] DIENS, École normale supérieure, CNRS, PSL Research University, Paris, France
[3] INRIA

Abstract. Multi-input functional encryption is a very useful generalization of Functional Encryption, which has been motivated by Goldwasser *et al.* from Eurocrypt '14. All the constructions, however, rely on non-standard assumptions. Very recently, at Eurocrypt '17, Abdalla *et al.* considered a restricted case and proposed an efficient multi-input inner-product functional encryption scheme.

In this paper, regarding the case of inner product, we argue that the multi-client setting (MCFE, for Multi-Client Functional Encryption), which borrows techniques from both Functional Encryption and Private Stream Aggregation, is better suited to real-life applications because of the strong restrictions implied by linear relations. We then propose a practical solution for Multi-Client Inner-Product Functional Encryption (IP-MCFE) which relies on the sole DDH assumption and supports adaptive corruptions.

In MCFE schemes, each data input is encrypted by a different client, and the clients might not trust anybody for the functional decryption keys. It thus seems quite important to remove any authority, while allowing corruptions of the clients by the adversary. We thus propose the notion of Decentralized Multi-Client Functional Encryption (DMCFE) and provide a generic construction from two MCFE schemes with particular properties. More concretely, combining two instantiations of our previous IP-MCFE, we can build an efficient and non-interactive decentralized scheme for inner product. Our construction relies on the SXDH assumption, and supports adaptive corruptions in the random oracle model.

Keywords. Functional Encryption, Inner Product, Private Stream Aggregation, Multi-Client, Decentralized.

1 Introduction

Functional Encryption (FE) [8, 12, 16, 25] is a new paradigm for encryption which extends the traditional "all-or-nothing" requirement in a much more flexible way. FE allows users to learn specific functions of the encrypted data: for any function f from a class $\mathcal{F}$, a secret functional decryption key dk_f can be computed such that, given any ciphertext c with underlying plaintext x, using dk_f, a user can efficiently compute $f(x)$, but does not get any additional information about x.

FE is the most general form of encryption as it encompasses identity-based encryption, attribute-based encryption, broadcast encryption, depending on the function f. However, the basic definition of FE implies that all the input data come from one party. In many practical applications, the data is an aggregation of information that comes from different parties and they may not trust each other. In order to deal with this scenario, two approaches have been introduced: Multi-Input FE (MIFE) [13, 14, 18] and Multi-Client Functional Encryption (MCFE) [13, 18].

Multi-Input Functional Encryption. Goldwasser *et al.* [13, 14, 18] introduced the notion of Multi-Input Functional Encryption (MIFE) which extends a single input x to an input vector $(x_1, \dots, x_n)$ where the components are independent. This allows many users to input their own data: user i can enter x_i and encrypt it as $c_i = \mathsf{Encrypt}(x_i)$. Anyone owning a functional decryption key dk_f, for an n-ary function f and multiple ciphertexts $c_1 = \mathsf{Encrypt}(x_1), \dots, c_n = \mathsf{Encrypt}(x_n)$, can compute $f(x_1, \dots, x_n)$ but nothing else about the individual x_i's. Numerous applications of MIFE have been given in detail in [13].

The security notion for FE and MIFE requires that no one should be able to guess which messages (between two lists) have been encrypted, under the restriction that the outputs of the functions for which the adversary has asked for a functional decryption key are the same for both lists of plaintexts. This excludes functional decryption keys that could trivially tell apart which ciphertexts have been encrypted. No security notion can do better, but even just n ciphertexts for an n-ary function might exclude the adversary from knowing any functional decryption key. This is particularly significant in the case of MIFE, where permutations and mix-and-match combinations can generate many valid vectors of ciphertexts.

Multi-Client Functional Encryption. For Multi-Client Functional Encryption (MCFE), as defined in [13, 18], both an index i for the client and a time-based counter t are used for the encryption: $(c_1 = \mathsf{Encrypt}(1, x_1, t), \dots, c_n = \mathsf{Encrypt}(n, x_n, t))$. Therefore, the combination of different ciphertexts, generated at different time periods, does not give a valid ciphertext and the adversary learns nothing from it. This makes possible to relax the restriction on the functional decryption keys the adversary can ask for in the security game, which becomes much more useful in practice. More generally, we can allow distinct clients that do not trust each other to submit their ciphertexts $c_i = \mathsf{Encrypt}(i, x_i, \ell)$ for any label ℓ, so that any vector under the same label with all the ordered indexes $i = 1, \dots, n$ can be decrypted with a functional decryption key. But since they do not trust anybody, they should not trust any authority either to generate the keys. We would thus be interested in a decentralized version of MCFE, where no authority is involved, but the generation of decryption keys remains an efficient process.

1.1 Related work

In the more general form, FE, MIFE, and MCFE schemes have been proposed [4,5,9,13,15–17,24,27] but unfortunately, they all rely on non standard cryptographic assumptions (indistinguishability obfuscation, single-input FE for circuits, or multilinear maps). However, it is more important in practice, and this is an interesting challenge, to build FE for restricted (but concrete) classes of functions, satisfying standard security definitions, under well-understood assumptions.

Inner Product. In 2015, Abdalla, Bourse, De Caro, and Pointcheval [1] considered the question of building FE for inner-product functions. In their paper, they show that inner-product functional encryption (IP-FE) can be efficiently realized under standard assumptions like the Decisional Diffie-Hellman (DDH) and Learning-with-Errors (LWE) assumptions [23], but for the selective security model only. Later on, Agrawal, Libert and Stehlé [3] considered adaptive security for IP-FE and proposed constructions the security of which is based on DDH, LWE or Paillier's Decisional Composite Residuosity (DCR) [22] assumptions.

The extension from IP-FE to IP-FE in the multi-input or multi-client settings is not simple. No construction from the DDH, LWE or DCR assumptions had been proposed until this year: at Eurocrypt '17, Abdalla *et al.* [2] proposed an efficient Multi-Input Inner-Product Functional Encryption (IP-MIFE) scheme that relies on the k-Lin assumption in prime-order bilinear groups.

Private Stream Aggregation (PSA). This notion, also referred to as Privacy-Preserving Aggregation of Time-Series Data, is an older notion introduced by Shi *et al.* [26]. It is quite similar to a natural decentralization of the MCFE scheme as just discussed above, with the main distinction being that PSA doesn't consider the possibility of generating different keys for different inner products, but only enables the aggregator to compute the *sum* of the clients' data for each time period. PSA also typically involves a Differential Privacy component, which has yet to be studied in the larger setting of MCFE. Further research on PSA has focused on achieving new properties or better efficiency [7,10,11,19–21] but not on enabling new functionalities.

1.2 Limitations of Multi-Input Functional Encryption

When considering the multi-input setting, the standard security notion has to deal with mix-and-match challenge ciphertexts in the security game: the adversary can input the ciphertexts in any order it wants and one must take into account the fact that the adversary can learn a lot from all the possible combinations of the challenge ciphertexts, even with one functional decryption key. Then, the security model might exclude the adversary from knowing any functional decryption key to avoid trivial attacks: the security notion becomes void.

If we assume an index when encrypting the messages: $c_1 = \mathsf{Encrypt}(1, x_1), \dots,$ $c_n = \mathsf{Encrypt}(n, x_n)$, one can only compute $f(x_1, \dots, x_n)$ from $c_1, \dots, c_n$, without the possibility of using x_i at some other position. The security notion becomes more realistic, but even with a strict ordering, one can still ask for another series of ciphertexts $c'_1 = \mathsf{Encrypt}(1, x'_1), \ldots, c'_n = \mathsf{Encrypt}(n, x'_n)$, and then mix-and-match the x_i's and the x'_i's, which leads to 2^n possible vectors: again, this likely excludes the adversary from asking for any functional decryption key. As mentioned above, almost all the constructions of MIFE rely on indistinguishability obfuscation or multilinear maps which we do not know how to instantiate under standard cryptographic assumptions. An exception is the construction of MIFE for inner product from Abdalla *et al.* [2], where the function $F_{\boldsymbol{y}}$ is the inner product $\langle \boldsymbol{x}, \boldsymbol{y} \rangle$ of the message $\boldsymbol{x}$ and the key $\boldsymbol{y}$, both seen as n-vectors of scalars.

Let us elaborate on Inner-Product Functional Encryption (IP-FE). We first look at the $n-$ary inner-product function: for a $\boldsymbol{y} = (y_1, \ldots, y_n)$ which specifies the function $f_{\boldsymbol{y}}$, on the message $\boldsymbol{x} = (x_1, \ldots, x_n)$, we define $f_{\boldsymbol{y}}(\boldsymbol{x}) = \langle \boldsymbol{x}, \boldsymbol{y} \rangle = \sum_i \langle x_i, y_i \rangle$. We observe that the input message $\boldsymbol{x}$ is already a vector and a *natural extension* into the multi-input setting is to allow each user to enter his data x_i, at a specific position. However, in case of MIFE, due to the security requirements, when considering standard notions of multiple challenge-ciphertexts in the case of inner-product functions, from just two different $n-$ary ciphertexts, any combination leads to 2^n valid ciphertexts, and as many linear equations from just one functional decryption key on $\boldsymbol{y}$, unless all of its components are 0. The decryption key for $\boldsymbol{y} = (0, \ldots, 0)$ is the only allowed query. This makes MIFE for Inner Product a completely void primitive.

In order to overcome this issue, Abdalla *et al.* [2] introduced another extension of IP-MIFE, by encrypting vectors of vectors, and then the functions takes vectors as individual inputs:

$$F_{\boldsymbol{y}_1, \ldots, \boldsymbol{y}_n}(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n) = \sum_i \langle \boldsymbol{x}_i, \boldsymbol{y}_i \rangle$$

where each $\boldsymbol{x}_i, \boldsymbol{y}_i$ are also vectors. They argue then that an exponential number of constraints on the whole vectors of vectors $\boldsymbol{y} = (\boldsymbol{y}_1, \ldots, \boldsymbol{y}_n)$ can be succinctly characterized by a quadratic constraint on the components $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_n$ and because the $\boldsymbol{y}_i$ are now vectors (of dimension strictly larger than 1), the space of decryption keys the adversary is allowed to query can still be large. But this does not solve the issue for 1-vectors $\boldsymbol{x}_i$'s, which is the situation we are interested in.

1.3 Multi-Client Functional Encryption

In the model of Multi-Client Functional Encryption, the above combination is avoided. MCFE is similar to MIFE but each ciphertext is associated to both an index and a time period t that we call and denote a label ℓ in this paper, thus a ciphertext vector is of the form $(c_1 = \mathsf{Encrypt}(1, x_1, \ell), \ldots, c_n = \mathsf{Encrypt}(n, x_n, \ell))$ for a label ℓ. Therefore, the combination of different ciphertexts, under different labels, does not give a valid ciphertext and the adversary learns nothing in doing so. In contrast to IP-MIFE, IP-MCFE supports the natural initial extension where the x_i's and y_i's can be scalars and not vectors. It fits more practical applications. MCFE also seems to fit real-life applications as, in practice, it is natural to associate data of the same record with a label: the users know which computation they need to perform on their specific joint data and it does not really make sense to combine different parts from different ciphertexts.

We also remark that, for MCFE, as explained in [13], counter-intuitively, the private-key setting is much more relevant than the public-key setting. Indeed, let us consider the case where $\mathsf{Encrypt}(.)$ is a public function and suppose that the adversary receives a challenge ciphertext $(c_1^* = \mathsf{Encrypt}(1, x_1^b, \ell), \ldots, c_n^* = \mathsf{Encrypt}(n, x_n^b, \ell))$ from its two chosen plaintexts $\boldsymbol{x}^0 = (x_1^0, \ldots, x_n^0)$ and $\boldsymbol{x}^1 = (x_1^1, \ldots, x_n^1)$. Then the adversary can combine any part c_i^* of the challenge ciphertext with ciphertexts $c_i = \mathsf{Encrypt}(i, x_i, \ell)$ it could generate itself, to make an acceptable ciphertext and use a functional decryption key to evaluate the function on the underlying vector. In order to prevent the adversary from a trivial win, one should make the restriction that the adversary is only allowed to ask functional decryption keys dk_f for functions f that satisfy $f(x_1^0, \cdot, \ldots, \cdot) = f(x_1^1, \cdot, \ldots, \cdot)$, $f(\cdot, x_2^0, \ldots, \cdot) = f(\cdot, x_2^0, \ldots, \cdot)$, $\ldots$, $f(\cdot, \cdot, \ldots, x_n^0) = f(\cdot, \cdot, \ldots, x_n^1)$. Again, this would essentially exclude any function. A private encryption solves this issue.

In this paper, we will thus consider this private-key setting which naturally fits the MCFE model as each component in the plaintext is separately provided by a client. In such a case, the corruption of some clients is an important issue, since several of them could collude to learn information about other clients' inputs.

1.4 Decentralized Multi-Client Functional Encryption

MCFE (like MIFE) assumes the existence of a trusted third-party who runs the SetUp algorithm and distributes the functional decryption keys. This third-party, if malicious or corrupted, can easily undermine any client's privacy. We are thus interested in building a scheme in which such a third-party is entirely removed from the equation.

We introduce the notion of Decentralized Multi-Client Functional Encryption (DMCFE), in which the setup phase and the generation of functional decryption keys is decentralized. We are interested in minimizing interaction during those operations. While a natural decentralization of our first MCFE scheme boasts a non-interactive setup phase, it requires interactions every time the clients agree to generate a new key. A scheme in which the setup phase is interactive but no interaction is required when generating new keys would arguably be more interesting.

1.5 Our contributions

Practical constructions of functional encryption for specific classes of functions is of high interest. In this paper, we focus on MCFE and DMCFE for Inner Product.

We propose the first solution for Inner-Product Functional Encryption in the Multi-Client setting that enjoys many interesting properties:

1. Efficiency: the proposed scheme is highly practical as it is as efficient as the DDH-based IP-FE scheme from [1]. A value x_i is encrypted as a unique group

element C_i. We stress that, in the multi-client setting, this is optimal because each input value x_i is independently added in a ciphertext by a client and therefore, for an n-ary input vector, the ciphertext should contain at least $O(n)$ elements.
2. Security under a standard assumption: our scheme is selectively secure under the classical DDH assumption.

In addition, we consider corruptions of users, and even adaptive corruptions, which Goldwasser *et al.* [13] outline as an "intersting direction".

We also formalize the new notion of Decentralized Multi-Client Functional Encryption, and then focus on DMCFE for Inner Product:

– We give a generic transformation for building DMCFE schemes from MCFE schemes that satisfy some reasonable properties.
– Whereas our above MCFE for Inner Product does not satisfy all these assumptions, we still manage to instantiate the first DMCFE for Inner Product from it, at the cost of introducing pairings. We prove that our scheme enjoys selective security and supports adaptive corruptions under the SXDH assumption.

We leave open the problems of considering LWE-based constructions and of extending this work beyond inner-product functions.

2 Definitions

2.1 Multi-Client Functional Encryption

A MCFE scheme encrypts vectors of data from several senders and allows the controlled computation of functions on these heterogeneous data. The information is structured in a table of n-vectors (which can thus be seen as an $m \times n$-matrix), where each component (each column of the matrix) is provided by one of the n distinct and independent senders. Since each component of the vectors should be provided by a specific sender only, a secret encryption key will be given to each sender. We thus define a secret-encryption MCFE as in [13,18]:

Definition 1 (Multi-Client Functional Encryption). *A multi-client functional encryption on $\mathcal{M}$ over a set of n senders is defined by four algorithms:*

– $\mathsf{SetUp}(\lambda)$*: Takes as input the security parameter λ, and outputs the public parameters* mpk*, the master secret key* msk *and the n encryption keys* ek_i*;*
– $\mathsf{Encrypt}(\mathsf{ek}_i, x_i, \ell)$*: Takes as input a personal encryption key* ek_i*, a value x_i to encrypt, and a label ℓ, and outputs the ciphertext $C_{\ell,i}$;*
– $\mathsf{DKeyGen}(\mathsf{msk}, f)$*: Takes as input the master secret key* msk *and a function $f : \mathcal{M}^n \to \mathcal{R}$, and outputs a functional decryption key* dk_f*;*
– $\mathsf{Decrypt}(\mathsf{dk}_f, \ell, \boldsymbol{C})$*: Takes as input a decryption key* dk_f*, a label ℓ, and a n-vector ciphertext $\boldsymbol{C}$, and outputs $f(\boldsymbol{x})$, if $\boldsymbol{C}$ is a valid encryption of $\boldsymbol{x} = (x_i)_i \in \mathcal{M}^n$ for the label ℓ, or $\perp$ otherwise.*

We make the assumption that mpk is included in msk and in all the encryption keys ek_i as well as the functional decryption keys dk_f. The correctness property states that, given $(\mathsf{mpk}, \mathsf{msk}, (\mathsf{ek}_i)_i) \leftarrow \mathsf{SetUp}(\lambda)$, for any label ℓ, any function $f : \mathcal{M}^n \to \mathcal{R}$, and any vector $\boldsymbol{x} = (x_i)_i \in \mathcal{M}^n$, if $C_{\ell,i} \leftarrow \mathsf{Encrypt}(\mathsf{ek}_i, x_i, \ell)$, for $i \in \{1, \ldots, n\}$, and $\mathsf{dk}_f \leftarrow \mathsf{DKeyGen}(\mathsf{msk}, f)$, then $\mathsf{Decrypt}(\mathsf{dk}_f, \ell, \boldsymbol{C}_\ell = (C_{\ell,i})_i) = f(\boldsymbol{x} = (x_i)_i)$ with overwhelming probability.

Definition 2 (IND-Security Game for MCFE). *Let us consider a MCFE scheme over a set of n senders. No adversary $\mathcal{A}$ should be able to win the following security game against a challenger $\mathcal{C}$:*

- *Initialization: the challenger $\mathcal{C}$ runs the setup algorithm* $(\mathsf{mpk}, \mathsf{msk}, (\mathsf{ek}_i)_i) \leftarrow \mathsf{SetUp}(\lambda)$ *to receive the parameters* mpk*, the master secret key* msk*, and the n encryption keys* ek_i*. It also chooses a random bit $b \xleftarrow{\$} \{0,1\}$. It provides* mpk *to the adversary $\mathcal{A}$;*
- *Encryption queries* $\mathsf{QEncrypt}$*: $\mathcal{A}$ has unlimited and adaptive access to a Left-or-Right encryption oracle, which on input (i, x^0, x^1, ℓ) runs* $\mathsf{Encrypt}(\mathsf{ek}_i, x^b, \ell)$*, and outputs the ciphertext $C_{\ell,i}$. We note that a second query for the same pair (ℓ, i) will later be ignored;*
- *Functional decryption key queries* $\mathsf{QDKeyGen}$*: $\mathcal{A}$ has unlimited and adaptive access to the* $\mathsf{DKeyGen}$ *algorithm for any input function f of its choice. It is given back the functional decryption key* dk_f*;*
- *Corruptions queries* $\mathsf{QCorrupt}$*: $\mathcal{A}$ can make an unlimited number of adaptive corruption queries on input index i, to get the encryption key* ek_i *of any sender i of its choice.*
- *Finalize: $\mathcal{A}$ provides its guess b' on the bit b.*

The output β of the game depends on some conditions, where $\mathcal{CS}$ is the set of corrupted senders (the set of indexes i input to $\mathsf{QCorrupt}$*), and $\mathcal{HS}$ the set of honest (non-corrupted) senders:*

1. *if, for some encryption query* $\mathsf{QEncrypt}(i, x_i^0, x_i^1, \ell)$*, for an index $i \in \mathcal{CS}$, $x_i^0 \neq x_i^1$, then $\beta \xleftarrow{\$} \{0,1\}$;*
2. *if, for some label ℓ and pair of vectors $\boldsymbol{x}^0 = (x_i^0)_i$ and $\boldsymbol{x}^1 = (x_i^1)_i$, where (i, x_i^0, x_i^1, ℓ) has been input to* $\mathsf{QEncrypt}$ *for all $i \in \mathcal{HS}$ and $x_i^0 = x_i^1$ for all $i \in \mathcal{CS}$, for some function f input to* $\mathsf{QDKeyGen}$*, $f(\boldsymbol{x}^0) \neq f(\boldsymbol{x}^1)$, then $\beta \xleftarrow{\$} \{0,1\}$;*
3. *otherwise, $\beta \leftarrow b'$.*

We say this MCFE is IND-secure if for any adversary $\mathcal{A}$, $\mathsf{Adv}^{\mathtt{IND}}(\mathcal{A}) = |P[\beta = 1|b = 1] - P[\beta = 1|b = 0]|$ is negligible.

Informally, this is the usual Left-or-Right indistinguishability [6], but where the adversary should not be able to get ciphertexts or functional decryption keys that trivially help distinguish the encrypted vectors:

1. since the encryption might be deterministic, if we allow Left-or-Right encryption queries even for corrupted encryption keys, these queries should be on identical messages;

2. for any functional decryption key, since the adversary can generate any ciphertext for the corrupted components, any such value should not help distinguishing the ciphertexts generated through $\mathsf{QEncrypt}$ (on honest components);

in which cases the guess of the adversary is not considered (a random bit β is output). Otherwise, this is a legitimate attack, and the guess b' of the adversary is output. We stress that we bar the adversary from querying several ciphertexts under the same pair (ℓ, i). In real life, it is of course the responsibility of the senders not to encrypt under the same label twice.

One may define weaker variants of indistinguishability, where some queries can only be sent *before* the initialization phase:

– Selective Security (`sel-IND`): the encryption queries ($\mathsf{QEncrypt}$) are sent before the initialization;
– Static Security (`sta-IND`): the corruption queries ($\mathsf{QCorrupt}$) are sent before the initialization.

2.2 Decentralized Multi-Client Functional Encryption

In MCFE, an authority owns msk to generate the functional decryption keys. We would like to avoid requiring such an authority, and make the scheme totally decentralized among the senders. We thus define DMCFE, for Decentralized Multi-Client Functional Encryption. In this context, there are n senders $(\mathcal{S}_i)_i$, for $i = 1, \ldots, n$, who will play the role of both the encrypting players and the decryption key generators for a functional decrypter $\mathcal{FD}$. Of course, the senders do not trust each other and they want to control the functional decryption keys that will be generated. There may be several functional decrypters, but since they could combine all the functional keys, in the description below, and in the security model, we will consider only one functional decrypter $\mathcal{FD}$.

Definition 3 (Decentralized Multi-Client Functional Encryption). *A decentralized multi-client functional encryption on $\mathcal{M}$ between a set of n senders $(\mathcal{S}_i)_i$, for $i = 1, \ldots, n$, and a functional decrypter $\mathcal{FD}$ is defined by two protocols and two algorithms:*

– $\mathsf{SetUp}(\lambda)$*: This is a protocol between the senders $(\mathcal{S}_i)_i$ that eventually generate their own secret keys sk_i and encryption keys ek_i, as well as the public parameters mpk;*
– $\mathsf{Encrypt}(\mathsf{ek}_i, x_i, \ell)$*: Takes as input a personal encryption key ek_i, a value x_i to encrypt, and a label ℓ, and outputs the ciphertext $C_{\ell,i}$;*
– $\mathsf{DKeyGen}((\mathsf{sk}_i)_i, f)$*: This is a protocol, with a function $f : \mathcal{M}^n \to \mathcal{R}$ as common input to each sender $\mathcal{S}_i$, in addition to their own secret key sk_i, which eventually outputs the functional decryption key dk_f;*
– $\mathsf{Decrypt}(\mathsf{dk}_f, \ell, \boldsymbol{C})$*: Takes as input a decryption key dk_f, a label ℓ, and a n-vector ciphertext $\boldsymbol{C}$, and outputs $f(\boldsymbol{x})$, if $\boldsymbol{C}$ is a valid encryption of $\boldsymbol{x} = (x_i)_i \in \mathcal{M}^n$ for the label ℓ, or $\perp$ otherwise;*

As above, we make the assumption that mpk is included in all the secret and encryption keys, as well as the functional decryption keys. The correctness property states that, given $(\mathsf{mpk}, (\mathsf{sk}_i)_i, (\mathsf{ek}_i)_i) \leftarrow \mathsf{SetUp}(\lambda)$, for any label ℓ, any function $f : \mathcal{M}^n \rightarrow \mathcal{R}$, and any vector $\boldsymbol{x} = (x_i)_i \in \mathcal{M}^n$, if $C_{\ell,i} \leftarrow \mathsf{Encrypt}(\mathsf{ek}_i, x_i, \ell)$, for $i \in \{1, \ldots, n\}$, and $\mathsf{dk}_f \leftarrow \mathsf{DKeyGen}((\mathsf{sk}_i)_i, f)$, then $\mathsf{Decrypt}(\mathsf{dk}_f, \ell, \boldsymbol{C}_\ell = (C_{\ell,i})_i) = f(\boldsymbol{x} = (x_i)_i)$ with overwhelming probability.

The security model is quite similar to the previous one, but corrupt-queries are important since the senders do not trust each other, and they now reveal the secret keys sk_i's:

Definition 4 (IND-Security Game for DMCFE). *Let us consider a DMCFE scheme between a set of n senders. No adversary $\mathcal{A}$ should be able to win the following security game against a challenger $\mathcal{C}$:*

- *Initialization: the challenger $\mathcal{C}$ runs the setup protocol $(\mathsf{mpk}, (\mathsf{sk}_i)_i, (\mathsf{ek}_i)_i) \leftarrow \mathsf{SetUp}(\lambda)$ to receive the parameters mpk, and the n pairs of secret and encryption keys $(\mathsf{sk}_i, \mathsf{ek}_i)_i$. It also chooses a random bit $b \xleftarrow{\$} \{0, 1\}$. It provides mpk to the adversary $\mathcal{A}$;*
- *Encryption queries $\mathsf{QEncrypt}$: $\mathcal{A}$ has unlimited and adaptive access to a Left-or-Right encryption oracle, which on input (i, x^0, x^1, ℓ) runs $\mathsf{Encrypt}(\mathsf{ek}_i, x^b, \ell)$, and outputs the ciphertext $C_{\ell,i}$. We note that a second query for the same pair (ℓ, i) will later be ignored;*
- *Functional decryption key queries $\mathsf{QDKeyGen}$: $\mathcal{A}$ has unlimited and adaptive access to the (non-corrupted) senders running the $\mathsf{DKeyGen}$ protocol for any input function f of its choice. It is given back the functional decryption key dk_f;*
- *Corruptions queries $\mathsf{QCorrupt}$: $\mathcal{A}$ can make an unlimited number of adaptive corruption queries on input index i, to get the secret and encryption keys $(\mathsf{sk}_i, \mathsf{ek}_i)$ of any sender i of its choice.*
- *Finalize: $\mathcal{A}$ provides its guess b' on the bit b.*

The output β of the game depends on some conditions, where $\mathcal{CS}$ is the set of corrupted senders (the set of indexes i input to $\mathsf{QCorrupt}$), and $\mathcal{HS}$ the set of honest (non-corrupted) senders:

1. *if, for some encryption query $\mathsf{QEncrypt}(i, x_i^0, x_i^1, \ell)$, for an index $i \in \mathcal{CS}$, $x_i^0 \neq x_i^1$, then $\beta \xleftarrow{\$} \{0, 1\}$;*
2. *if, for some label ℓ and pair of vectors $\boldsymbol{x}^0 = (x_i^0)_i$ and $\boldsymbol{x}^1 = (x_i^1)_i$, where (i, x_i^0, x_i^1, ℓ) has been input to $\mathsf{QEncrypt}$ for all $i \in \mathcal{HS}$ and $x_i^0 = x_i^1$ for all $i \in \mathcal{CS}$, for some function f obtained from the $\mathsf{QDKeyGen}$ protocol, $f(\boldsymbol{x}^0) \neq f(\boldsymbol{x}^1)$, then $\beta \xleftarrow{\$} \{0, 1\}$;*
3. *otherwise, $\beta \leftarrow b'$.*

We say this DMCFE is IND-secure if for any adversary $\mathcal{A}$, $\mathsf{Adv}^{\mathtt{IND}}(\mathcal{A}) = |P[\beta = 1 | b = 1] - P[\beta = 1 | b = 0]|$ is negligible.

Similarly to MCFE, we can define the weaker `sel-IND` and `sta-IND` security notions for DMCFE.

3 A Centralized Construction for the Inner Product

3.1 Description

In this section, we present a first MCFE scheme for inner product. It is inspired by Abdalla *et al.*'s scheme [1]:

– $\mathsf{SetUp}(\lambda)$: Takes as input the security parameter, and generates a group $\mathbb{G}$ of prime order $p \approx 2^\lambda$, $g \in \mathbb{G}$ a generator, and $\mathcal{H}$ a full-domain hash function onto $\mathbb{G}$. It also generates the encryption keys $s_i \overset{\$}{\leftarrow} \mathbb{Z}_p$, for $i = 1, \ldots, n$, and sets $\boldsymbol{s} = (s_i)_i$. The public parameters mpk consist of $(\mathbb{G}, p, g, \mathcal{H})$, while the master secret key is $\mathsf{msk} = \boldsymbol{s}$ and the encryption keys are $\mathsf{ek}_i = s_i$ for $i = 1, \ldots, n$ (in addition to mpk, which is omitted);
– $\mathsf{Encrypt}(\mathsf{ek}_i, x_i, \ell)$: Takes as input the value x_i to encrypt, under the key $\mathsf{ek}_i = s_i$ and the label ℓ, and outputs the ciphertext $C_{\ell,i} = \mathcal{H}(\ell)^{s_i} \cdot g^{x_i}$;
– $\mathsf{DKeyGen}(\mathsf{msk}, \boldsymbol{y})$: Takes as input $\mathsf{msk} = \boldsymbol{s}$ and an inner-product function defined by $\boldsymbol{y}$ as $f_{\boldsymbol{y}}(\boldsymbol{x}) = \langle \boldsymbol{x}, \boldsymbol{y} \rangle$, and outputs the functional decryption key $\mathsf{dk}_{\boldsymbol{y}} = (\boldsymbol{y}, \langle \boldsymbol{s}, \boldsymbol{y} \rangle)$;
– $\mathsf{Decrypt}(\mathsf{dk}, \ell, \boldsymbol{C})$: Takes as input a decryption key $\mathsf{dk} = (\boldsymbol{y}, dk)$, a label ℓ, and a ciphertext $\boldsymbol{C} = (C_i)_i$, to compute $g^\alpha = (\prod_i C_i^{y_i}) \times \mathcal{H}(\ell)^{-dk}$, and eventually solve the discrete logarithm to extract and return α.

We stress that, as for Abdalla *et al.*'s scheme [1], the result α should not be too large to allow the final discrete logarithm computation.

Correctness : if the scalar dk in the decryption functional key $\mathsf{dk}_{\boldsymbol{y}} = (\boldsymbol{y}, dk)$ is indeed $dk = \langle \boldsymbol{s}, \boldsymbol{y} \rangle$, then

$$(\prod_i C_i^{y_i}) \times \mathcal{H}(\ell)^{-dk} = (\prod_i C_i^{y_i}) \times \mathcal{H}(\ell)^{-\langle \boldsymbol{s},\boldsymbol{y} \rangle} = \prod_i (\mathcal{H}(\ell)^{s_i} \cdot g^{x_i})^{y_i} \times \mathcal{H}(\ell)^{-\langle \boldsymbol{s},\boldsymbol{y} \rangle}$$
$$= \mathcal{H}(\ell)^{\langle \boldsymbol{s},\boldsymbol{y} \rangle} \cdot g^{\langle \boldsymbol{x},\boldsymbol{y} \rangle} \times \mathcal{H}(\ell)^{-\langle \boldsymbol{s},\boldsymbol{y} \rangle} = g^{\langle \boldsymbol{x},\boldsymbol{y} \rangle}.$$

3.2 Selective Security

Like Abdalla *et al.*'s original scheme [1], our protocol can only be proven secure in the weaker security model, where the adversary has to commit in advance to all the pairs of messages for the Left-or-Right encryption oracle (QEncrypt-queries). However, he can adaptively ask for functional decryption keys (QDKeyGen-queries) and encryption keys (QCorrupt-queries). Concretely, the challenger is provided with the two m-row matrices $X^0 = (\boldsymbol{x}_j^0)_j$, $X^1 = (\boldsymbol{x}_j^1)_j$, with a vector of labels $\boldsymbol{\ell} = (\ell_j)_j$, all with $j = 1, \ldots, m$. The challenge ciphertexts $C_{j,i} = \mathsf{Encrypt}(\ell_j, \mathsf{ek}_i, x_{j,i}^b)$, for the random bit b, are returned to the adversary.

Note that the adversary can keep some messages void, but necessarily the global pair, $x_{j,i}^0 = x_{j,i}^1 = \perp$, which means that the adversary does not ask for this ciphertext $C_{j,i}$. Since in the security model we exclude two encryption queries for the same pair (ℓ, i), the vector $\boldsymbol{\ell}$ must have distinct components ($\ell_j \neq \ell_{j'}$, for any $j \neq j'$).

Definition 5 (Selective Security for MCFE). *Let us consider an MCFE over a set of n senders. No adversary $\mathcal{A}$ should be able to win the following security game against a challenger $\mathcal{C}$:*

- *Initialization with $(\boldsymbol{\ell}, X^0 = (\boldsymbol{x}_j^0)_j, X^1 = (\boldsymbol{x}_j^1)_j)$: the challenger $\mathcal{C}$ runs the setup algorithm $(\mathsf{mpk}, \mathsf{msk}, (\mathsf{ek}_i)_i) \leftarrow \mathsf{SetUp}(\lambda)$ to receive the parameters mpk, the master secret key msk, and the n encryption keys ek_i. It also chooses a random bit $b \overset{\$}{\leftarrow} \{0,1\}$ and runs $C_{j,i} \leftarrow \mathsf{Encrypt}(\ell_j, \mathsf{ek}_i, x_{j,i}^b)$, if both $x_{j,i}^0 \neq \perp$ and $x_{j,i}^1 \neq \perp$, otherwise $C_{j,i} \leftarrow \perp$, for $j = 1, \ldots, m$ and $i = 1, \ldots, n$. Eventually, $\mathcal{C}$ provides mpk, and all the ciphertexts $C_{j,i}$ to the adversary $\mathcal{A}$;*
- *Functional decryption key queries $\mathsf{QDKeyGen}$: $\mathcal{A}$ has unlimited and adaptive access to the $\mathsf{DKeyGen}$ algorithm for any function f of its choice. It is given back the functional decryption key dk_f;*
- *Corruptions queries $\mathsf{QCorrupt}(i)$: $\mathcal{A}$ can make an unlimited number of adaptive corruption queries on input index i, to get the encryption key ek_i of any sender i of its choice. The index i is added to the set $\mathcal{CS}$ of corrupted senders;*
- *Finalize: $\mathcal{A}$ provides its guess b' on the bit b.*

The output β of the game depends on the same conditions as above:

- *if, for some $i \in \mathcal{CS}$ and some j, $x_{j,i}^0 \neq \perp$, and $x_{j,i}^0 \neq x_{j,i}^1$ (otherwise $x_{j,i}^0 = x_{j,i}^1 \in \mathcal{M} \cup \{\perp\}$ for any $i \in \mathcal{CS}$), then $\beta \overset{\$}{\leftarrow} \{0,1\}$;*
- *if, for some j, there are vectors $\boldsymbol{x}^0$ and $\boldsymbol{x}^1$ that are equal to $\boldsymbol{x}_j^0$ and $\boldsymbol{x}_j^1$ respectively, except for the indexes $i \in \mathcal{CS}$, where any common value can be set, such that for some function f asked to $\mathsf{QDKeyGen}$, $f(\boldsymbol{x}^0) \neq f(\boldsymbol{x}^1)$, then $\beta \overset{\$}{\leftarrow} \{0,1\}$;*
- *otherwise, $\beta \leftarrow b'$.*

We say this MCFE is `sel-IND`-secure if for any adversary $\mathcal{A}$, $\mathsf{Adv}^{\texttt{sel-IND}}(\mathcal{A}) = |P[\beta = 1|b = 1] - P[\beta = 1|b = 0]|$ is negligible.

3.3 Security Analysis

We will show below that the above scheme is secure under the DDH assumption:

Definition 6 (Decisional Diffie-Hellman Assumption (DDH)). *In a group $\mathbb{G}$ of prime order p, with a generator g, the DDH assumption states that the two following distributions are computationally indistinguishable:*

$$\mathcal{D} = \{(X = g^x, Y = g^y, Z = \mathit{CDH}(X,Y) = g^{xy}) \mid x, y \overset{\$}{\leftarrow} \mathbb{Z}_p\}$$
$$\mathcal{D}' = \{(X = g^x, Y = g^y, Z = g^z) \mid x, y, z \overset{\$}{\leftarrow} \mathbb{Z}_p\}.$$

We denote $\mathsf{Adv}_{\mathbb{G}}^{\mathsf{ddh}}(\mathcal{A})$ the advantage of an adversary $\mathcal{A}$ in distinguishing the two distributions: $\mathsf{Adv}_{\mathbb{G}}^{\mathsf{ddh}}(\mathcal{A}) = \Pr_{\mathcal{D}'}[\mathcal{A}(X,Y,Z) = 1] - \Pr_{\mathcal{D}}[\mathcal{A}(X,Y,Z) = 1]$. We also use the notation $\mathsf{Adv}_{\mathbb{G}}^{\mathsf{ddh}}(t)$ for the advantage of the best distinguisher running within time t.

This assumption implies a Multi-DDH version:

Lemma 7. *Under the* DDH *assumption in any group* $\mathbb{G}$*, the two following distributions are indistinguishable:*

$$\mathcal{D}_0 = \left\{ (A_i, B_j, \mathsf{CDH}(A_i, B_j))_{i,j} \mid A_i, B_j \overset{\$}{\leftarrow} \mathbb{G} \right\}$$
$$\mathcal{D}_1 = \left\{ (A_i, B_j, C_{i,j})_{i,j} \mid A_i, B_j, C_{i,j} \overset{\$}{\leftarrow} \mathbb{G} \right\}$$

More precisely, the advantage of the best distinguisher between these two distributions within time t is $\mathsf{Adv}^{\mathsf{mddh}}_{\mathbb{G}}(n, m, t) \leq n \times \mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}}(t + 4mt_{\mathbb{G}})$, where $t_{\mathbb{G}}$ is the time for an exponentiation in $\mathbb{G}$, n is the maximum number of indexes i, and m is the maximum number of j.

The full proof of this classical lemma can be found in the Appendix A. We now state the security level of the MCFE scheme presented in Section 3.1:

Theorem 8. *The* MCFE *scheme presented in Section 3.1 is* `sel-IND`*-secure under the* DDH *assumption, in the random oracle model. More precisely, for any adversary $\mathcal{A}$ within running time bounded by t, and m selective encryption queries (but any adaptive corruption queries and adaptive functional decryption key queries),*

$$\mathsf{Adv}^{\texttt{sel-IND}}(\mathcal{A}) \leq \mathsf{Adv}^{\mathsf{mddh}}_{\mathbb{G}}(m, n, t) \leq m \times \mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}}(t + 4nt_{\mathbb{G}}),$$

where $t_{\mathbb{G}}$ is the time for an exponentiation in $\mathbb{G}$.

Proof (Sketch of Proof). Since we are dealing with Inner Product functions, for a corrupted key $\mathsf{ek}_i = s_i$, the adversary can compute $y_i \cdot s_i$ by itself for any vector $\boldsymbol{y}$, so we just have to consider restrictions to the sub-space with the components of non-corrupted indexes. The conditions (for not setting β at random) can sum up to:

– for any $i \in \mathcal{CS}$, the i-th column of $X^0 - X^1$ is 0 (we assume that $x^0_{j,i} - x^1_{j,i} = 0$ if they are both equal to $\perp$);
– for any $\boldsymbol{y}$ asked to QDKeyGen, $(X^0 - X^1) \cdot \boldsymbol{y} = 0$.

It is interesting to note that in this scheme, we have $\mathsf{QCorrupt}(i) = \mathsf{ek}_i = \mathsf{DKeyGen}(\mathsf{msk}, \boldsymbol{e_i})$ where $\boldsymbol{e_i} = (\delta_{ij})_j$, so the corruption resiliency is ensured by a similar condition as the adaptivity of functional decryption key queries: $(X^0 - X^1) \cdot \boldsymbol{e_i} = 0$. The remaining issue could be the adaptivity of such corruption queries. But since in the selective security model, the messages $X^0 = (\boldsymbol{x}^0_j)_j$ and $X^1 = (\boldsymbol{x}^1_j)_j$ are chosen by $\mathcal{A}$ before having seen any other parameter, the simulator can generate keys in order to explicitly know all the keys ek_i for the i-th columns of $X^0 - X^1$ equal to 0: no other key can be asked.

The ciphertext is essentially a label-dependent one-time pad: $\boldsymbol{k}_\ell + \boldsymbol{x}^b_\ell$. Because of the restrictions on the functional decryption keys, if the simulator $\mathcal{S}$ adds a random vector $\boldsymbol{w} \in \langle X^0 - X^1 \rangle$ to the ciphertext, this will not change anything from the view of the adversary after functional decryption since $\langle \boldsymbol{x}^b_\ell + \boldsymbol{w}, \boldsymbol{y} \rangle = \langle \boldsymbol{x}^b_\ell, \boldsymbol{y} \rangle$ for any $\boldsymbol{y}$ in the sub-space orthogonal to $\langle X^0 - X^1 \rangle$. Thus, choosing $\boldsymbol{w}_\ell = \boldsymbol{x}^b_\ell - \boldsymbol{x}^{1-b}_\ell$

makes the encryption of $\boldsymbol{x}_\ell^b$ perfectly indistinguishable from that of $\boldsymbol{x}_\ell^{1-b}$. Our proof shows that for any label ℓ, $\mathcal{S}$ can alter the $\langle X^0 - X^1 \rangle$-projection of the mask $\boldsymbol{k}_\ell$ to make such random vectors, independent from each other, appear. We make this possible under the DDH assumption, by making $\boldsymbol{k}_\ell$ a vector of CDH values. We then show that this change does not affect the view of the adversary $\mathcal{A}$.

Since this proof is simpler, but in the same vein as the proof of Theorem 15, it is postponed to the Appendix B.

4 From MCFE to Efficient DMCFE

4.1 The Interactivity of DMCFE Schemes

Any MCFE can generically be decentralized by distributing the SetUp and DKeyGen algorithms into protocols between the $\mathcal{S}_i$'s. However, while it may be reasonable to require many interactions during the setup phase, since it is only executed once, such a decentralization might make the key generation process costly and inconvenient: every time the functional decrypter $\mathcal{FD}$ wants to be able to evaluate a new function, he needs to get all the $\mathcal{S}_i$'s to interactively generate a new functional decryption key.

We call a DMCFE scheme *non-interactive* if its DKeyGen protocol is non-interactive between the $\mathcal{S}_i$'s, but just requires one round between the functional decrypter and the senders (As explained above, we only focus on efficient DKeyGen since the SetUp is run once only. But of course, the less interactive the latter is, the better it is for the global protocol).

4.2 MCFE-Enabled MCFE and Self-Enabling MCFE

We notice that our above IP-MCFE scheme has a very remarkable property: it enables the computation of inner products on each client's input using functional decryption keys which are themselves inner products on each client's secret key with the function. This suggests we might be able to provide the functional decryption key to the functional decrypter using the same scheme. We now formalize this notion in a more general way.

Definition 9 ($\widetilde{\mathcal{E}}$-Enabled MCFE). *Let $\mathcal{E}$ and $\widetilde{\mathcal{E}}$ be two MCFE schemes. We say that $\mathcal{E}$ is $\widetilde{\mathcal{E}}$-enabled if*

- *the master key* msk *of $\mathcal{E}$ is simply the union of the encryption keys* ek_i*;*
- *there are a function T onto $\widetilde{\mathcal{M}}$ and a function $\widetilde{F} : \widetilde{\mathcal{M}}^n \to \widetilde{\mathcal{R}}$ in the set of functions of $\widetilde{\mathcal{E}}$ such that for any function $f : \mathcal{M}^n \to \mathcal{R}$ in the set of functions of $\mathcal{E}$,*
$$\mathsf{dk}_f = \mathcal{E}.\mathsf{DKeyGen}(\mathsf{msk}, f) = \widetilde{F}((T(\mathcal{E}.\mathsf{ek}_i, f))_{i \in 1,\ldots,n}).$$
.

Definition 10 (Self-Enabling MCFE). *Let $\mathcal{E}$ be an MCFE scheme. We say that $\mathcal{E}$ is self-enabling if it is $\mathcal{E}$-enabled.*

4.3 A Generic Construction

Let us consider any pair $(\mathcal{E}, \widetilde{\mathcal{E}})$ of MCFE schemes such that $\mathcal{E}$ is $\widetilde{\mathcal{E}}$-enabled, with the functions T and $\widetilde{F}$ defined as above. We can construct a non-interactive DMCFE scheme for the set of functions of $\mathcal{E}$:

– $\mathsf{SetUp}(\lambda)$: The n senders $(\mathcal{S}_i)_i$ execute $\mathcal{E}.\mathsf{SetUp}(\lambda)$ and $\widetilde{\mathcal{E}}.\mathsf{SetUp}(\lambda)$ via MPC to generate the encryption keys $(\mathcal{E}.\mathsf{ek}_i)_i$ and $(\widetilde{\mathcal{E}}.\mathsf{ek}_i)_i$ and the shares $(\widetilde{\mathsf{msk}_i})_i$ of the master secret $\widetilde{\mathcal{E}}.\mathsf{msk}$. Recall that we don't need to share $\mathcal{E}.\mathsf{msk}$ since this is already $\mathcal{E}.\mathsf{msk} = (\mathcal{E}.\mathsf{ek}_i)_i$. They also execute $\widetilde{\mathcal{E}}.\mathsf{DKeyGen}(\widetilde{\mathcal{E}}.\mathsf{msk}, \widetilde{F})$ via MPC to generate $\widetilde{dk_{\widetilde{F}}}$.
  We set $\mathsf{mpk} = (\mathcal{E}.\mathsf{mpk}, \widetilde{\mathcal{E}}.\mathsf{mpk}, \widetilde{dk_{\widetilde{F}}})$, $\mathsf{sk}_i = (\widetilde{\mathsf{msk}_i}, \widetilde{\mathcal{E}}.\mathsf{ek}_i)$, and $\mathsf{ek}_i = \mathcal{E}.\mathsf{ek}_i$. $\mathcal{FD}$ receives mpk;
– $\mathsf{Encrypt}(\mathsf{ek}_i, x_i, \ell) = \mathcal{E}.\mathsf{Encrypt}(\mathsf{ek}_i, x_i, \ell)$;
– $\mathsf{DKeyGen}((\mathsf{sk}_i)_i, f)$: Each $\mathcal{S}_i$ computes $\widetilde{C_i} = \widetilde{\mathcal{E}}.\mathsf{Encrypt}(\widetilde{\mathcal{E}}.\mathsf{ek}_i, T(\mathsf{ek}_i, f), f)$. The functional decryptor $\mathcal{FD}$ can compute

$$\mathsf{dk}_f = \widetilde{\mathcal{E}}.\mathsf{Decrypt}(\widetilde{dk_{\widetilde{F}}}, f, (\widetilde{C_i})_{i\in\{1,\ldots,n\}});$$

– $\mathsf{Decrypt}(\mathsf{dk}, \ell, \boldsymbol{C}) = \mathcal{E}.\mathsf{Decrypt}(\mathsf{dk}, \ell, \boldsymbol{C})$.

Correctness follows from the correctness of $\mathcal{E}$ and the relation

$$\begin{aligned}\widetilde{\mathcal{E}}.\mathsf{Decrypt}(\widetilde{dk_{\widetilde{F}}}, f, (\widetilde{C_i})_i) &= \widetilde{\mathcal{E}}.\mathsf{Decrypt}(\widetilde{dk_{\widetilde{F}}}, f, (\widetilde{\mathcal{E}}.\mathsf{Encrypt}(\widetilde{\mathcal{E}}.\mathsf{ek}_i, T(\mathsf{ek}_i, f), f))_i) \\ &= \widetilde{F}((T(\mathsf{ek}_i, f))_i) = \mathsf{dk}_f\end{aligned}$$

4.4 Security Analysis

To prove the security of our scheme we first need to define the following property:

Definition 11 (Samplability of $\widetilde{F}$). *Given a function T, a distribution $\mathcal{D}$ of n-vectors, and a function f from a set of functions $\mathcal{F}$, we define:*

$$\mathcal{D}_{T,\mathcal{D}}(f) = \left\{ (T(\mathsf{ek}'_i, f))_{i\in\{1,\ldots,n\}} \middle| (\mathsf{ek}'_i)_{i\in\{1,\ldots,n\}} \xleftarrow{\$} \mathcal{D} \right\}.$$

A function $\widetilde{F}$ is said to be samplable with regards to T and $\mathcal{D}$ if, for all f, for all $X \subset \{1,\ldots,n\}$, for any $x \in \mathrm{Im}(\widetilde{F})$ and set $(\mathsf{ek}_i)_{i\in X}$, there is an efficient sampler $\mathcal{SA}_{\widetilde{F},T,\mathcal{D}}(X, x, (\mathsf{ek}_i)_{i\in X}, f)$ from the conditional distribution

$$\left\{ (T'_i)_i \xleftarrow{\$} \mathcal{D}_{T,\mathcal{D}}(f) \mid x = \widetilde{F}((T'_i)_i) \text{ and } \forall i \in X, T'_i = T(\mathsf{ek}_i, f) \right\}.$$

In practice, $\mathcal{DS}_{T,\mathcal{D}}(f)$ is the distribution of the shares of dk_f that can be recombined by the function $\widetilde{F}$, since the distribution $\mathcal{D}$ is the distribution of the encryption keys generated by the SetUp phase of $\mathcal{E}$. The conditional distribution will simply consider the shares $(T'_i)_i$ that still recombine to dk_f, but for some fixed values: the corrupted ones. Then the set X will be the set $\mathcal{CS}$ of the corrupted senders and $x = \mathsf{dk}_f$. We will be interested in the sampling

$$\mathcal{SA}_{\widetilde{F},T,\mathcal{E}.\mathsf{SetUp}}(\mathcal{CS}, \mathsf{dk}_f, (\mathsf{ek}_i)_{i\in\mathcal{CS}}, f).$$

This appears in the proof because we are trying to prove indistinguishability on the ciphertexts that are sent during $\mathsf{DKeyGen}$. We cannot simply replace those ciphertexts at random since the adversary would then get a random decryption key, which it would easily notice when decrypting: decryption would fail! We need to encrypt plaintexts that are chosen randomly but under the constraint that the adversary still recovers the correct decryption key, and that the decryption keys of the corrupted senders are unchanged.

Security Model. We will limit ourselves to proving *static* security (`sta-IND`): we will show that under certain conditions, our scheme is secure in a game with adaptive queries but static corruptions. This means the attacker must decide which senders to corrupt before the initialization phase. Without this condition, every client is at risk of being corrupted at some point in the future. Using the client's secret key, the adversary could trivially detect that the $(T'_i)_i$'s were chosen at random. Note that we will be able to prove security against adaptive corruptions for a practical instantiation for inner product later on.

Theorem 12. *Let $(\mathcal{E}, \widetilde{\mathcal{E}})$ be a pair of `sta-IND`-secure MCFE schemes such that $\mathcal{E}$ is $\widetilde{\mathcal{E}}$-enabled via functions $\widetilde{F}$ and T such that $\widetilde{F}$ is samplable with regards to T and the set of functions of $\mathcal{E}$. Then the non-interactive DMCFE scheme from Section 4.3 is itself `sta-IND`-secure. More precisely:*

$$\mathsf{Adv}^{\texttt{sta-IND}}_{\mathsf{DMCFE}}(t) \leq \mathsf{Adv}^{\texttt{sta-IND}}_{\mathcal{E}}(t + qt_s) + \mathsf{Adv}^{\texttt{sta-IND}}_{\widetilde{\mathcal{E}}}(t + qt_s)$$

where t_s is the time associated with a sampling using $\mathcal{SA}_{\widetilde{F},T,\mathcal{E}.\mathsf{SetUp}}$ and q is the number of adaptive functional decryption key queries.

Proof. The security proof follows a series a games. We start from the real game. Then, we apply the sampling algorithm to generate random ciphertexts that hide the encryption keys of $\mathcal{E}$ under the indistinguishability of $\widetilde{\mathcal{E}}$. Then, we simply apply the security of $\mathcal{E}$:

Game $\mathbf{G}_0$: The first game is the real game where the simulator $\mathcal{S}$ perfectly simulates the view of the adversary.

- Initialization: $\mathcal{A}$ chooses a set of corrupt senders $\mathcal{CS}$. $\mathcal{S}$ runs the setup protocol $(\mathsf{mpk}, (\mathsf{sk}_i)_i, (\mathsf{ek}_i)_i) \leftarrow \mathsf{SetUp}(\lambda)$ to receive the parameters $\mathsf{mpk} = (\mathcal{E}.\mathsf{mpk}, \widetilde{\mathcal{E}}.\mathsf{mpk}, \widetilde{dk}_{\widetilde{F}})$, and the n pairs of secret and encryption keys $(\mathsf{sk}_i = (\widetilde{\mathsf{msk}}_i, \widetilde{\mathcal{E}}.\mathsf{ek}_i), \mathsf{ek}_i = \mathcal{E}.\mathsf{ek}_i)_i$. It chooses a random bit $b \overset{\$}{\leftarrow} \{0,1\}$. It finally provides mpk and the secret and encryption keys $(\mathsf{sk}_i, \mathsf{ek}_i)$ of sender i, for $i \in \mathcal{CS}$, to the adversary $\mathcal{A}$;
- Encryption queries $\mathsf{QEncrypt}$: $\mathcal{A}$ sends a request (i, x^0, x^1, ℓ) and receives $C_{\ell,i} = \mathsf{Encrypt}(\mathsf{ek}_i, x^b, \ell)$ from $\mathcal{S}$. We note that a second query for the same pair (ℓ, i) will later be ignored;

– Functional decryption key queries $\mathsf{QDKeyGen}$: $\mathcal{A}$ requests a decryption key for a function f of its choice. It is given, for all $i \in \{1, \dots, n\}$, the

$$\widetilde{C_i} = \widetilde{\mathcal{E}}.\mathsf{Encrypt}(\widetilde{\mathcal{E}}.\mathsf{ek}_i, T(\mathsf{ek}_i, f), f)$$

from which it recovers the functional decryption key dk_f;

– Finalize: $\mathcal{A}$ provides its guess b' on the bit b and $\mathcal{S}$ filters the cases were $\mathcal{A}$ requested unauthorized decryption keys or corruptions by setting $\beta \xleftarrow{\$} \{0,1\}$. In the other cases, $\mathcal{S}$ sets $\beta \leftarrow b'$.

We define the advantage of $\mathcal{A}$ at the end of this game as

$$\mathsf{Adv}_{\mathbf{G}_0}(\mathcal{A}) = |\Pr[\beta = 1 | b = 1] - \Pr[\beta = 1 | b = 0]| = \mathsf{Adv}^{\mathtt{IND}}(\mathcal{A}).$$

The conditions for not setting β at random are

– for any $i \in \mathcal{CS}$ (the set of corrupt senders), for any x^0, x^1 such that for some ℓ, $\mathsf{QEncrypt}(i, x^0, x^1, \ell)$ was requested, $x^0 = x^1$;

– for any f asked to $\mathsf{QDKeyGen}$, for any pair of vectors $(X^0, X^1) \in \mathbb{Z}_p^{2n}$ such that for all $i \in \mathcal{CS}$ $X_i^0 = X_i^1$ and for all $i \in \mathcal{HS}$ (the set of non-corrupted senders, and thus honest senders) and for some ℓ, $\mathsf{QEncrypt}(i, X_i^0, X_i^1, \ell)$ was requested, $f(X^0) = f(X^1)$.

Game $\mathbf{G}_1$: In this game $\mathcal{S}$ randomly replaces his response for $i \in \mathcal{HS}$ when answering a request for a functional decryption key.

– Functional decryption key queries: $\mathcal{A}$ requests a decryption key for a function f of its choice. $\mathcal{S}$ computes dk_f, sets

$$(T'_i)_i \xleftarrow{\$} \mathcal{SA}_{\widetilde{F}, T, \mathcal{E}.\mathsf{SetUp}}(\mathcal{CS}, \mathsf{dk}_f, (\mathsf{ek}_i)_{i \in \mathcal{CS}}, f)$$

and sends $\widetilde{C_i}' = \widetilde{\mathcal{E}}.\mathsf{Encrypt}(\widetilde{\mathcal{E}}.\mathsf{ek}_i, T'_i, f)$ for all $i \in \{1, \dots, n\}$ to $\mathcal{A}$, who can then recover dk_f.

Since $\widetilde{\mathcal{E}}$ is $\mathtt{sta\text{-}IND}$-secure, this game is indistinguishable from the previous one, with: $\mathsf{Adv}_{\mathbf{G}_0}(\mathcal{A}) - \mathsf{Adv}_{\mathbf{G}_1}(\mathcal{A}) \leq \mathsf{Adv}_{\widetilde{\mathcal{E}}}^{\mathtt{sta\text{-}IND}}(t + qt_s)$. We indeed toke care not to alter evaluation of the function $\widetilde{F}$ on the plaintexts $(T'_i)_i$ with the sampling algorithm.

Game $\mathbf{G}_2$: In this game $\mathcal{S}$ will use $\mathcal{A}$'s guesses to try to attack a challenger $\mathcal{C}$ that simulates the real game for scheme $\mathcal{E}$.

– Initialization: $\mathcal{A}$ chooses a set of corrupt senders $\mathcal{CS}$, which it sends to $\mathcal{S}$ who forwards it to $\mathcal{C}$. $\mathcal{C}$ runs the setup protocol $(\mathcal{E}.\mathsf{mpk}, (\mathcal{E}.\mathsf{sk}_i)_i, (\mathcal{E}.\mathsf{ek}_i)_i) \leftarrow \mathcal{E}.\mathsf{SetUp}(\lambda)$, chooses a random bit $b \xleftarrow{\$} \{0,1\}$, and provides $\mathcal{E}.\mathsf{mpk}$ and the secret and encryption keys $(\mathcal{E}.\mathsf{sk}_i, \mathcal{E}.\mathsf{ek}_i)$ of sender i, for $i \in \mathcal{CS}$, to $\mathcal{S}$. $\mathcal{S}$ runs $(\widetilde{\mathcal{E}}.\mathsf{mpk}, (\widetilde{\mathcal{E}}.\mathsf{sk}_i)_i, (\widetilde{\mathcal{E}}.\mathsf{ek}_i)_i) \leftarrow \widetilde{\mathcal{E}}.\mathsf{SetUp}(\lambda)$, and combines the

$$(\mathcal{E}.\mathsf{mpk}, (\mathcal{E}.\mathsf{sk}_i)_{i \in \mathcal{CS}}, (\mathcal{E}.\mathsf{ek}_i)_{i \in \mathcal{CS}}, \widetilde{\mathcal{E}}.\mathsf{mpk}, (\widetilde{\mathcal{E}}.\mathsf{sk}_i)_{i \in \mathcal{CS}}, (\widetilde{\mathcal{E}}.\mathsf{ek}_i)_{i \in \mathcal{CS}})$$

to form $(\mathsf{mpk}, (\mathsf{sk}_i)_{i \in \mathcal{CS}}, (\mathsf{ek}_i)_{i \in \mathcal{CS}})$ which it sends to $\mathcal{A}$;

– Encryption queries $\mathsf{QEncrypt}$: $\mathcal{A}$ sends a request (i, x^0, x^1, ℓ), which $\mathcal{S}$ forwards to $\mathcal{C}$. $\mathcal{C}$ responds with $C_{\ell,i} = \mathsf{Encrypt}(\mathsf{ek}_i, x^b, \ell)$ which $\mathcal{S}$ forwards to $\mathcal{A}$. A second query for the same pair (ℓ, i) will later be ignored;

- Functional decryption key queries $\mathsf{QDKeyGen}$: $\mathcal{A}$ requests a decryption key for a function f of its choice. $\mathcal{S}$ forwards the request to $\mathcal{C}$ and receives dk_f. $\mathcal{S}$ then sets
$$(T'_i)_i \xleftarrow{\$} \mathcal{SA}_{\widetilde{F},T,\mathcal{E}.\mathsf{SetUp}}(\mathcal{CS}, \mathsf{dk}_f, (\mathsf{ek}_i)_{i\in\mathcal{CS}}, f)$$
and sends $\widetilde{C_i}' = \widetilde{\mathcal{E}}.\mathsf{Encrypt}(\widetilde{\mathcal{E}}.\mathsf{ek}_i, T'_i, f)$ for all $i \in \{1,\ldots,n\}$ to $\mathcal{A}$, who recovers dk_f;
- Finalize: $\mathcal{A}$ provides its guess b' on the bit b and $\mathcal{S}$ forwards it to $\mathcal{C}$, who filters the cases were $\mathcal{S}$ (and thus $\mathcal{A}$) requested unauthorized decryption keys or corruptions by setting $\beta \xleftarrow{\$} \{0,1\}$. In the other cases, $\mathcal{C}$ sets $\beta \leftarrow b'$.

This reduction shows that
$$\mathsf{Adv}_{\mathbf{G}_1}(\mathcal{A}) = \mathsf{Adv}_{\mathbf{G}_2}(\mathcal{A}) = \mathsf{Adv}_{\mathbf{G}_2}(\mathcal{S}) \leq \mathsf{Adv}_{\mathcal{E}}^{\texttt{sta-IND}}(t+qt_s)$$
and it follows that
$$\mathsf{Adv}_{\mathbf{G}_0} \leq \mathsf{Adv}_{\mathcal{E}}^{\texttt{sta-IND}}(t+qt_s) + \mathsf{Adv}_{\widetilde{\mathcal{E}}}^{\texttt{sta-IND}}(t+qt_s)$$

Corollary 13. *From any* `sta-IND`*-secure* MCFE *scheme* $\mathcal{E}$ *which is self-enabling via functions* $\widetilde{F}$ *and* T *such that* $\widetilde{F}$ *is samplable with regards to* T *and the set of functions of* $\mathcal{E}$*, we can construct a non-interactive* DMCFE *scheme for the set of functions of* $\mathcal{E}$ *that is also* `sta-IND`*-secure.*

5 A DMCFE for Inner Product

5.1 Introduction

Our construction of MCFE for inner product uses functional decryption keys $\mathsf{dk}_{\boldsymbol{y}} = (\boldsymbol{y}, \langle \boldsymbol{s}, \boldsymbol{y}\rangle) = (\boldsymbol{y}, dk_{\boldsymbol{y}})$, where $dk_{\boldsymbol{y}} = \langle \boldsymbol{s}, \boldsymbol{y}\rangle = \sum_i s_i y_i = \langle \boldsymbol{t}, \mathbf{1}\rangle$, with $t_i = s_i y_i$, for $i = 1,\ldots,n$, and $\mathbf{1} = (1,\ldots,1)$. Hence, one can split $\mathsf{msk} = \boldsymbol{s}$ into $\mathsf{msk}_i = s_i$, define $T(\mathsf{msk}_i, \boldsymbol{y}) = t_i = s_i y_i$ and $F(\boldsymbol{t}) = \langle \boldsymbol{t}, \mathbf{1}\rangle$. We could thus wish to use the above generic construction with our MCFE for inner product, that is self-enabling, to describe a DMCFE for inner product.

However, this is not straightforward:

- our MCFE supports adaptive corruptions, while the generic construction can just achieve security for static corruptions. This would be a big loss from our security goal;
- our MCFE only supports selective security, which means that all the messages to be encrypted have to be known from the beginning. Of course, this could just lead to a selectively-secure DMCFE, but the second level of encryption has to encrypt the t_i's, that are derived from the functional decryption key queries. Hence, our MCFE seems to limit to functional decryption key queries known from the beginning too. This would also be a strong limitation;

– our MCFE only allows small results for the function evaluations, since a discrete logarithm has to be computed. While, for real-life applications, it might be reasonable to assume the plaintexts and any evaluations on them are small enough, it is impossible to recover such a large scalar as $dk_{\boldsymbol{y}} = \langle \boldsymbol{s}, \boldsymbol{y} \rangle$, which comes up when we use our scheme to encrypt encryption keys.

Nevertheless, following the idea from the generic construction, we can overcome the concerns above:

– One can only recover $g^{dk_{\boldsymbol{y}}}$, but using pairings, say $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$, one can use our MCFE for both $\mathcal{E}$ and $\widetilde{\mathcal{E}}$. The former running in $\mathbb{G}_1$ while the latter runs in $\mathbb{G}_2$. This allows us to compute the functional decryption in $\mathbb{G}_T$, to get $g_T^{\langle \boldsymbol{x}, \boldsymbol{y} \rangle}$, which is decryptable as $\langle \boldsymbol{x}, \boldsymbol{y} \rangle$ is small enough;
– Since the choice of the two sets of messages X^0 and X^1 specifies the possibly corrupted senders and the set of authorized functional decryption keys, the selective security of the two MCFE's is enough to achieve the selective security of our DMCFE, and we can still support adaptive corruptions.

5.2 Construction

Let us describe the new construction, in a type 3 pairing-friendly structure $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$:

– $\mathsf{SetUp}(\lambda)$: Choose a type 3 pairing-friendly structure, with a bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$, where the groups $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ are of prime order $p \approx 2^\lambda$, and $g \in \mathbb{G}_1$, $\widetilde{g} \in \mathbb{G}_2$ and $g_T = e(g, \widetilde{g}) \in \mathbb{G}_T$ are three generators. One also needs two full-domain hash functions $\mathcal{H}$ and $\widetilde{\mathcal{H}}$ onto $\mathbb{G}_1$ and $\mathbb{G}_2$ respectively. Each $\mathcal{S}_i$ generates two encryption keys $s_i, \widetilde{s_i} \overset{\$}{\leftarrow} \mathbb{Z}_p$, for $i = 1, \ldots, n$. The $(\mathcal{S}_i)_i$ then interactively compute $\widetilde{dk_{\boldsymbol{1}}} = \langle \widetilde{\boldsymbol{s}}, \boldsymbol{1} \rangle = \sum_{i=1}^n \widetilde{s_i}$. One then sets $\mathsf{mpk} \leftarrow (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g, \widetilde{g}, g_T, \mathcal{H}, \widetilde{\mathcal{H}}, \widetilde{dk_{\boldsymbol{1}}})$, and for $i = 1, \ldots, n$, $\mathsf{ek}_i = s_i$, $\mathsf{sk}_i = (s_i, \widetilde{s_i})$;
– $\mathsf{Encrypt}(\mathsf{ek}_i, x_i, \ell)$ outputs the ciphertext $C_{\ell,i} = \mathcal{H}(\ell)^{s_i} \cdot g^{x_i}$;
– $\mathsf{DKeyGen}((\mathsf{sk}_i)_i, \boldsymbol{y})$: from $\boldsymbol{y}$ that defines the function $f_{\boldsymbol{y}}(\boldsymbol{x}) = \langle \boldsymbol{x}, \boldsymbol{y} \rangle$, each sender $\mathcal{S}_i$ with his secret key $\mathsf{sk}_i = (s_i, \widetilde{s_i})$ computes $\widetilde{C_{\boldsymbol{y},i}} = \widetilde{\mathcal{H}}(\boldsymbol{y})^{\widetilde{s_i}} \cdot \widetilde{g}^{s_i \cdot y_i}$. The functional decrypter can compute $dk_{\boldsymbol{y}} \leftarrow \prod_i \widetilde{C_{\boldsymbol{y},i}} \times \widetilde{\mathcal{H}}(\boldsymbol{y})^{-\widetilde{dk_{\boldsymbol{1}}}}$ and build $\mathsf{dk}_{\boldsymbol{y}} = (\boldsymbol{y}, dk_{\boldsymbol{y}})$;
– $\mathsf{Decrypt}(\mathsf{dk}, \ell, \boldsymbol{C})$: Takes as input a decryption key $\mathsf{dk} = (\boldsymbol{y}, dk)$, a label ℓ, and a ciphertext $\boldsymbol{C} = (C_i)_i$, to compute

$$g_T^\alpha = e(\prod_i C_i^{y_i}, \widetilde{g}) / e(\mathcal{H}(\ell), dk),$$

and eventually solve the discrete logarithm in basis g_T to extract and return α.

Correctness : let us first show that the functional key is similar to that of the previous scheme, the only difference being an added basis $\widetilde{g}$:

$$\begin{aligned} dk = \prod_i \widetilde{C_{\boldsymbol{y},i}} \times \widetilde{\mathcal{H}}(\boldsymbol{y})^{-\widetilde{dk_1}} &= \prod_i \widetilde{\mathcal{H}}(\boldsymbol{y})^{\widetilde{s_i}} \cdot \widetilde{g}^{s_i \cdot y_i} \times \widetilde{\mathcal{H}}(\boldsymbol{y})^{-\sum_i \widetilde{s_i}} \\ &= \widetilde{\mathcal{H}}(\boldsymbol{y})^{\sum_i \widetilde{s_i}} \cdot \widetilde{g}^{\sum_i s_i \cdot y_i} \times \widetilde{\mathcal{H}}(\boldsymbol{y})^{-\sum_i \widetilde{s_i}} = \widetilde{g}^{\langle \boldsymbol{s}, \boldsymbol{y} \rangle}. \end{aligned}$$

The functional decryption of a ciphertext of $\boldsymbol{x}$ thus leads to the correct inner product:

$$\begin{aligned} e(\prod_i C_i^{y_i}, \widetilde{g})/e(\mathcal{H}(\ell), dk) &= e(\prod_i (\mathcal{H}(\ell)^{s_i} \cdot g^{x_i})^{y_i}, \widetilde{g})/e(\mathcal{H}(\ell), \widetilde{g}^{\langle \boldsymbol{s}, \boldsymbol{y} \rangle}) \\ &= e(\mathcal{H}(\ell)^{\langle \boldsymbol{s}, \boldsymbol{y} \rangle} \cdot g^{\langle \boldsymbol{x}, \boldsymbol{y} \rangle}, \widetilde{g})/e(\mathcal{H}(\ell), \widetilde{g})^{\langle \boldsymbol{s}, \boldsymbol{y} \rangle} \\ &= e(\mathcal{H}(\ell), \widetilde{g})^{\langle \boldsymbol{s}, \boldsymbol{y} \rangle} \cdot e(g, \widetilde{g})^{\langle \boldsymbol{x}, \boldsymbol{y} \rangle}/e(\mathcal{H}(\ell), \widetilde{g})^{\langle \boldsymbol{s}, \boldsymbol{y} \rangle} = g_T^{\langle \boldsymbol{x}, \boldsymbol{y} \rangle}. \end{aligned}$$

5.3 Security Analysis

We will show below that the above scheme is secure under the SXDH assumption:

Definition 14 (Symmetric eXternal Diffie-Hellman Assumption). *In a type 3 pairing-friendly structure $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$, the Symmetric eXternal Diffie-Hellman (SXDH) Assumption states that both the DDH assumption in $\mathbb{G}_1$ and the DDH assumption in $\mathbb{G}_2$ hold. We then denote by $\mathsf{Adv}^{\mathsf{sxdh}}(t)$ the maximum of $\mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}_1}(t)$ and $\mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}_2}(t)$.*

Theorem 15. *The DMCFE protocol for Inner Product presented in Section 5, with n senders, is `sel-IND`-secure under the SXDH assumption, in the random oracle model. More precisely, for any adversary $\mathcal{A}$ within running time bounded by t, m selective encryption queries, q adaptive functional decryption key queries but any adaptive corruption queries,*

$$\begin{aligned} \mathsf{Adv}^{\texttt{sel-IND}}(\mathcal{A}) &\leq \mathsf{Adv}^{\mathsf{mddh}}_{\mathbb{G}_1}(n, q, t) + \mathsf{Adv}^{\mathsf{mddh}}_{\mathbb{G}_2}(n, m, t) \\ &\leq n \times \mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}_1}(t + 4qt_{\mathbb{G}_1}) + n \times \mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}_2}(t + 4mt_{\mathbb{G}_2}) \\ &\leq 2n \times \mathsf{Adv}^{\mathsf{sxdh}}(t + 4\max\{qt_{\mathbb{G}_1}, mt_{\mathbb{G}_2}\}), \end{aligned}$$

where $t_{\mathbb{G}_1}$ and $t_{\mathbb{G}_2}$ are the times for an exponentiation in $\mathbb{G}_1$ or $\mathbb{G}_2$ respectively.

Again, the game for selective security is similar to the `IND` security game, except for the fact that the labels $\boldsymbol{\ell} = (\ell_i)_i$ and the plaintexts $X^0 = (\boldsymbol{x}_j^0)_j$ and $X^1 = (\boldsymbol{x}_j^1)_j)$ are chosen in advance by the adversary.

Proof (Sketch of Proof). While this scheme does not exactly follow the black-box construction described previously, it uses the same idea of combining two MCFE schemes: the first one in $\mathbb{G}_1$ to encrypt messages and the second one in $\mathbb{G}_2$ to generate functional decryption keys. So, even if we cannot re-use the proof of the original scheme, the main idea is the same.

Unlike in the original scheme, the adversary ends the security game with the data of the two schemes: the usual information of the functional encryption scheme, and $n \times q$ encryptions $\widetilde{C_{\boldsymbol{y}_j,i}}$ of the $s_i \cdot y_{j,i}$ as well as the key $\widetilde{dk}_{\mathbf{1}} = \sum_{i=1}^{n} \widetilde{s}_i$. We can show these informations do not leak anything on $\widetilde{\boldsymbol{s}}$ by modifying it into $\widetilde{\boldsymbol{s}} + \boldsymbol{w}$ with a particular $\boldsymbol{w}$ without $\mathcal{A}$ being able to detect it.

We provide the full proof in Section 6.

6 Proof of Theorem 15

The main issue in DMCFE from MCFE is that the functional decryption key generation protocol leaks more than just the functional decryption key. It additionally encrypts (under the keys of $\widetilde{\mathcal{E}}$) data related to the encryption keys of $\mathcal{E}$. We thus first show how to remove any information about the $\mathcal{E}$-keys in the $\widetilde{\mathcal{E}}$-ciphertexts, and then the proof follows like the previous one to show that $\mathcal{E}$ does not leak any information about the plaintexts.

Game $\mathbf{G}_0$: The first game is the real game where the simulator $\mathcal{S}$ perfectly simulates the view of the adversary honestly generating the secret keys, given the pairing-friendly structure $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$ of prime order p, generators $g \in \mathbb{G}_1$, $\widetilde{g} \in \mathbb{G}_2$, $g_T = e(g, \widetilde{g}) \in \mathbb{G}_T$ with full-domain hash functions $\mathcal{H}$ and $\widetilde{\mathcal{H}}$ onto $\mathbb{G}_1$ and $\mathbb{G}_2$ respectively, modeled as random oracles:

- Hash function queries: since we are in the random oracle model, $\mathcal{S}$ sets two empty lists Λ and $\widetilde{\Lambda}$ of triples. For any query ℓ to $\mathcal{H}$, $\mathcal{S}$ looks for a triple $(\ell, \star, h) \in \Lambda$. If such a triple exists, it outputs h, otherwise it chooses a random $h \overset{\$}{\leftarrow} \mathbb{G}_1$, stores $(\ell, \perp, h)$ in Λ, and outputs h. The process is the same for any query to $\widetilde{\mathcal{H}}$ except that the values $\widetilde{h}$ are chosen in $\mathbb{G}_2$ instead;
- Initialization with $(\boldsymbol{\ell}, X^0, X^1)$, for $X^0, X^1 \in \mathbb{Z}_p^{m \times n}$: $\mathcal{S}$ sets $\mathsf{msk} = (\boldsymbol{s}, \widetilde{\boldsymbol{s}})$ with $\boldsymbol{s}, \widetilde{\boldsymbol{s}} \overset{\$}{\leftarrow} \mathbb{Z}_p^n$, $\mathcal{S}$ extracts the n encryption keys $\mathsf{ek}_i = (s_i, \widetilde{s}_i)$, for $i = 1, \dots, n$, computes $\widetilde{dk}_{\mathbf{1}} = \sum_{i=1}^{n} \widetilde{s}_i$, and sets the public parameter $\mathsf{msk} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, g, \widetilde{g}, g_T = e(g, \widetilde{g}), \mathcal{H}, \widetilde{\mathcal{H}}, \widetilde{dk})$. It also chooses a random bit $b \overset{\$}{\leftarrow} \{0, 1\}$. It also emulates $\mathsf{Encrypt}$ by setting and outputting $C_{j,i} = \mathcal{H}(\ell_j)^{s_i} \cdot g^{x^b_{j,i}}$, for valid inputs, for $j = 1, \dots, m$ and $i = 1, \dots, n$;
- Functional decryption key queries: for any query $\mathsf{QDKeyGen}(\boldsymbol{y})$, for a vector $\boldsymbol{y}$, $\mathcal{S}$ computes and sends $\widetilde{C_{\boldsymbol{y},i}} = \widetilde{\mathcal{H}}(\boldsymbol{y})^{\widetilde{s}_i} \cdot \widetilde{g}^{s_i \cdot y_i}$ for all the honest senders;
- Corruption: for a sender i, $\mathcal{S}$ sends back $\mathsf{ek}_i = (s_i, \widetilde{s}_i)$;
- Finalize: $\mathcal{A}$ provides its guess b' on the bit b and $\mathcal{S}$ filters the cases were $\mathcal{A}$ requested unauthorized decryption keys or corruptions by setting $\beta \overset{\$}{\leftarrow} \{0, 1\}$. In the other cases, $\mathcal{S}$ sets $\beta \leftarrow b'$.

We define the advantage of $\mathcal{A}$ at the end of this game as

$$\mathsf{Adv}_{\mathbf{G}_0}(\mathcal{A}) = |\Pr[\beta = 1 | b = 1] - \Pr[\beta = 1 | b = 0]| = \mathsf{Adv}^{\mathtt{sel\text{-}IND}}(\mathcal{A}).$$

As already noted for the basic scheme, the conditions for not setting β at random are

- for any $i \in \mathcal{CS}$ (the set of corrupted senders), the i-th column of $X^0 - X^1$ is 0 (we assume that $x^0_{j,i} - x^1_{j,i} = 0$ if they are both equal to $\perp$);
- for any $\boldsymbol{y}$ input to QDKeyGen, $(X^0 - X^1) \cdot \boldsymbol{y} = 0$.

However, now, a corruption leaks both s_i and $\widetilde{s_i}$: s_i is still the decryption functional key for the vector $\boldsymbol{e}_i = (\delta_{i,j})_j$, and $\widetilde{s_i}$ is just conditioned by the property $\widetilde{dk_{\mathbf{1}}} = \sum_{i=1}^n \widetilde{s_i}$.

We thus know that the corrupted senders can only be among the set $\mathcal{PCS}$ of indexes i such that the i-th column of $X^0 - X^1$ is 0: $\mathcal{CS} \subseteq \mathcal{PCS}$. We can say that $\mathcal{PCS}$ is the set of potentially corrupted senders, which is known from the beginning, while $\mathcal{CS}$ will only be known at the end of the game, because of the adaptive corruptions.

Game $\mathbf{G}_1$: In this game, we simply modify the simulation of the random oracles.

- Hash function queries: $\mathcal{S}$ sets two empty lists Λ and $\widetilde{\Lambda}$ of triples. For any query ℓ to $\mathcal{H}$, $\mathcal{S}$ looks for a triple $(\ell, \star, h) \in \Lambda$. If such a triple exists, it outputs h, otherwise it chooses a random $r \xleftarrow{\$} \mathbb{Z}_p$, sets $h \leftarrow g^r$, stores (ℓ, r, h) in Λ, and outputs h. The same modifications applies for $\widetilde{\mathcal{H}}$, with a random $\widetilde{r}$ and $\widetilde{h} = \widetilde{g}^{\widetilde{r}}$.

This simulation is perfectly indistinguishable from the previous one:

$$\mathsf{Adv}_{\mathbf{G}_1}(\mathcal{A}) = \mathsf{Adv}_{\mathbf{G}_0}(\mathcal{A}).$$

But one can note that, now, for every ciphertext on valid inputs, for $i = 1, \ldots, n$ and $j = 1, \ldots, m$,

$$C_{j,i} = \mathcal{H}(\ell_j)^{s_i} \cdot g^{x^b_{j,i}} = g^{r_j s_i} \cdot g^{x^b_{j,i}}, \text{ since } \mathcal{H}(\ell_j) = g^{r_j}.$$

Similarly, in $\mathbb{G}_2$:

$$\widetilde{C_{\boldsymbol{y},i}} = \widetilde{\mathcal{H}}(\boldsymbol{y})^{\widetilde{s_i}} \cdot \widetilde{g}^{s_i y_i} = \widetilde{g}^{\widetilde{r_{\boldsymbol{y}}} \widetilde{s_i}} \cdot \widetilde{g}^{s_i y_i}, \text{ since } \widetilde{\mathcal{H}}(y) = \widetilde{g}^{\widetilde{r}_{\boldsymbol{y}}}.$$

Game $\mathbf{G}_2$: In this game, we split the vector space $S = \mathbb{Z}_p^n$ according to the set $\mathcal{PCS}$, with cardinal c, of potentially corrupted senders. More precisely, we consider the space $S = \mathbb{Z}_p^n$ of all the possible secret keys $\widetilde{\boldsymbol{s}}$, and $\widetilde{S}_0$ is the space spanned by the $c+1$ n-vectors $\{\mathbf{1}, (\boldsymbol{e}_i)_{i \in \mathcal{PCS}}\}$ while $\widetilde{S}_1$ is the orthogonal. We know that S is the orthogonal direct sum of $\widetilde{S}_0$ and $\widetilde{S}_1$. The dimension of $\widetilde{S}_0$ is $c+1$, while the dimension of $\widetilde{S}_1$ is $n-c-1$. We now apply a change of basis in the initialization:

- Initialization with $(\boldsymbol{\ell}, X^0, X^1)$, for $X^0, X^1 \in \mathbb{Z}_p^{m \times n}$: $\mathcal{S}$ choose a $(n-c-1)$-basis $(\boldsymbol{\mu}_\eta)_\eta$ of $\widetilde{S}_1$ and $\widetilde{\boldsymbol{s}}$ is randomly chosen as a random linear combination of $(\boldsymbol{\mu}_\eta)_\eta$, $(\boldsymbol{e}_\eta)_\eta$ and $\mathbf{1}$: for random scalar $\gamma_\eta, \varepsilon_\eta, \varepsilon \xleftarrow{\$} \mathbb{Z}_p$

$$\widetilde{\boldsymbol{s}} = \sum \gamma_\eta \boldsymbol{\mu}_\eta + \sum_{\eta \in \mathcal{PCS}} \varepsilon_\eta \boldsymbol{e}_\eta + \varepsilon \mathbf{1},$$

and so each component is defined as

$$\widetilde{s_i} = \sum \gamma_\eta \langle \boldsymbol{\mu}_\eta, \boldsymbol{e}_i \rangle + \sum_{\eta \in \mathcal{PCS}} \varepsilon_\eta \langle \boldsymbol{e}_\eta, \boldsymbol{e}_i \rangle + \varepsilon.$$

We will omit the sets for the sums when there is no ambiguity, for the sake of clarity: for $\widetilde{S}_1$ basis-elements, $\eta = 1, \ldots, n-c-1$, while for $\widetilde{S}_0$ basis-elements, $\eta \in \mathcal{PCS}$: For $i \in \mathcal{PCS}$, $\widetilde{s_i} = \varepsilon_i + \varepsilon \bmod p$, while for $i \notin \mathcal{PCS}$, $\widetilde{s_i} = \sum \gamma_\eta \mu_{\eta,i} + \varepsilon \bmod p$. $\mathcal{S}$ then sets $\mathsf{msk} = (\boldsymbol{s}, \widetilde{\boldsymbol{s}})$ with $\boldsymbol{s} \xleftarrow{\$} \mathbb{Z}_p^n$, extracts the n encryption keys $\mathsf{ek}_i = (s_i, \widetilde{s_i})$, for $i = 1, \ldots, n$, and computes

$$\begin{aligned}\widetilde{dk_{\mathbf{1}}} = \sum_{i=1}^{n} \widetilde{s_i} &= \langle \sum \gamma_\eta \boldsymbol{\mu}_\eta + \sum_{\eta \in \mathcal{PCS}} \varepsilon_\eta \boldsymbol{e}_\eta + \varepsilon \mathbf{1}, \mathbf{1} \rangle \\ &= \sum \gamma_\eta \langle \boldsymbol{\mu}_\eta, \mathbf{1} \rangle + \sum_{\eta \in \mathcal{PCS}} \varepsilon_\eta + n\varepsilon = \sum_{\eta \in \mathcal{PCS}} \varepsilon_\eta + n\varepsilon \bmod p,\end{aligned}$$

since $\boldsymbol{\mu}_\eta \perp \mathbf{1}$ for all η. $\mathcal{S}$ then sets the public parameters $\mathsf{mpk} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, g, \widetilde{g}, g_T = e(g, \widetilde{g}), \mathcal{H}, \widetilde{\mathcal{H}}, \widetilde{dk_{\mathbf{1}}})$. It also chooses a random bit $b \xleftarrow{\$} \{0,1\}$. It also emulates $\mathsf{Encrypt}$ by setting and outputting $C_{j,i} = \mathcal{H}(\ell_j)^{s_i} \cdot g^{x^b_{j,i}}$, for valid inputs, for $j = 1, \ldots, m$ and $i = 1, \ldots, n$; This simulation is perfectly indistinguishable from the previous one:

$$\mathsf{Adv}_{\mathbf{G}_2}(\mathcal{A}) = \mathsf{Adv}_{\mathbf{G}_1}(\mathcal{A}).$$

Game $\mathbf{G}_3$: We still use a $(c+1)$-basis $\mathbf{1} \cup (\boldsymbol{e}_\eta)_\eta$ of $\widetilde{S}_0$ and an $(n-c-1)$-basis $(\boldsymbol{\mu}_\eta)_\eta$ of $\widetilde{S}_1$. But for the master secret key, one just chooses random scalars $\varepsilon, \varepsilon_\eta \xleftarrow{\$} \mathbb{Z}_p$, while the scalars $\gamma_\eta \in \mathbb{Z}_p$ are implicitly defined by random group elements $\widetilde{L}_\eta \xleftarrow{\$} \mathbb{G}_2$, since the scalars are not used anymore:

- Initialization with $(\boldsymbol{\ell}, X^0, X^1)$, for $X^0, X^1 \in \mathbb{Z}_p^{m \times n}$: $\mathcal{S}$ chooses a $(n-c-1)$-base $(\boldsymbol{\mu}_\eta)_\eta$ of $\widetilde{S}_1$ and $\widetilde{\boldsymbol{s}}$ is randomly and implicitly chosen as a random linear combination of $(\boldsymbol{\mu}_\eta)_\eta$, $(\boldsymbol{e}_\eta)_\eta$ and $\mathbf{1}$: for random scalar $\varepsilon_\eta, \varepsilon \xleftarrow{\$} \mathbb{Z}_p$, we define the γ_η as the discrete logarithm of the random group elements $\widetilde{L}_\eta \xleftarrow{\$} \mathbb{G}_2$: $\widetilde{\boldsymbol{s}} = \sum \gamma_\eta \boldsymbol{\mu}_\eta + \sum_{\eta \in \mathcal{PCS}} \varepsilon_\eta \boldsymbol{e}_\eta + \varepsilon \mathbf{1}$. Then $\mathcal{S}$ computes $\widetilde{dk_{\mathbf{1}}} = \sum_{\eta \in \mathcal{PCS}} \varepsilon_\eta + n\varepsilon \bmod p$ with the explicit values only.
- Functional decryption key queries: for any query $\mathsf{QDKeyGen}(\boldsymbol{y})$, for a vector $\boldsymbol{y}$ and for $i = 1, \ldots, n$, $\mathcal{S}$ computes

$$\widetilde{C_{\boldsymbol{y},i}} = \begin{cases} \widetilde{g}^{\widetilde{r}_{\boldsymbol{y}}(\varepsilon_i + \varepsilon)} \times \widetilde{g}^{s_i y_i} & \text{if } i \in \mathcal{PCS} \\ \prod \widetilde{L}_\eta^{\widetilde{r}_{\boldsymbol{y}} \mu_{\eta,i}} \times \widetilde{g}^{\widetilde{r}_{\boldsymbol{y}} \varepsilon} \times \widetilde{g}^{s_i y_i} & \text{if } i \notin \mathcal{PCS} \end{cases}$$

- Corruption: for any sender $i \in \mathcal{PCS}$, $\mathcal{S}$ sends $\mathsf{ek}_i = (s_i, \widetilde{s_i} = \varepsilon_i + \varepsilon)$;
- Finalize: same as in $\mathbf{G}_2$.

Since the explicit values of the scalars $(\gamma_\eta)_\eta$ are not needed, and

$$\begin{aligned}\widetilde{C_{\boldsymbol{y},i}} &= \widetilde{g}^{\widetilde{r}_{\boldsymbol{y}}(\varepsilon_i + \varepsilon)} \times \widetilde{g}^{s_i y_i} = (\widetilde{g}^{\widetilde{r}_{\boldsymbol{y}}})^{\widetilde{s_i}} \times \widetilde{g}^{s_i y_i} = \widetilde{\mathcal{H}}(\boldsymbol{y})^{\widetilde{s_i}} \times \widetilde{g}^{s_i y_i}, \text{ if } i \in \mathcal{PCS} \\ \widetilde{C_{\boldsymbol{y},i}} &= \prod \widetilde{L}_\eta^{\widetilde{r}_{\boldsymbol{y}} \mu_{\eta,i}} \times \widetilde{g}^{\widetilde{r}_{\boldsymbol{y}} \varepsilon} \times \widetilde{g}^{s_i y_i} = \prod \widetilde{g}^{\widetilde{r}_{\boldsymbol{y}} \gamma_\eta \mu_{\eta,i}} \times \widetilde{g}^{\widetilde{r}_{\boldsymbol{y}} \varepsilon} \times \widetilde{g}^{s_i y_i}, \text{ if } i \notin \mathcal{PCS} \\ &= (\widetilde{g}^{\widetilde{r}_{\boldsymbol{y}}})^{\sum \gamma_\eta \mu_{\eta,i} + \varepsilon} \times \widetilde{g}^{s_i y_i} = (\widetilde{g}^{\widetilde{r}_{\boldsymbol{y}}})^{\widetilde{s_i}} \times \widetilde{g}^{s_i y_i} = \widetilde{\mathcal{H}}(\boldsymbol{y})^{\widetilde{s_i}} \times \widetilde{g}^{s_i y_i},\end{aligned}$$

this simulation is perfectly indistinguishable from the previous one:

$$\mathsf{Adv}_{\mathbf{G}_3}(\mathcal{A}) = \mathsf{Adv}_{\mathbf{G}_2}(\mathcal{A}).$$

Game $\mathbf{G}_4$: Now, we are given $\widetilde{L}_\eta \xleftarrow{\$} \mathbb{G}_2$ for $\eta = 1, \dots, n-c-1$, $\widetilde{H}_{\boldsymbol{y}} \xleftarrow{\$} \mathbb{G}_2$ for any $\boldsymbol{y}$ queried to $\widetilde{\mathcal{H}}$, and we denote by $\widetilde{M}_{\eta,\boldsymbol{y}}$ the Diffie-Hellman value of any pair $\widetilde{L}_\eta$ and $\widetilde{H}_{\boldsymbol{y}}$ in basis $\widetilde{g}$. The initialization is kept unchanged, but uses the above $\widetilde{L}_\eta$. This only impacts the hash function queries and the functional decryption key queries:

- Hash function queries: for any new query $\boldsymbol{y}$ to $\widetilde{\mathcal{H}}$, one outputs $\widetilde{H}_{\boldsymbol{y}}$ and adds $(\boldsymbol{y}, \perp, \widetilde{H}_{\boldsymbol{y}})$ to $\widetilde{\Lambda}$;
- Functional decryption key queries: for any query $\mathsf{QDKeyGen}(\boldsymbol{y})$, for a vector $\boldsymbol{y}$ and for $i = 1, \dots, n$, $\mathcal{S}$ computes

$$\widetilde{C_{\boldsymbol{y},i}} = \begin{cases} \widetilde{H}_{\boldsymbol{y}}^{\varepsilon_i+\varepsilon} \times \widetilde{g}^{s_i y_i} & \text{if } i \in \mathcal{PCS} \\ \prod \widetilde{M}_{\eta,\boldsymbol{y}}^{\mu_{\eta,i}} \times \widetilde{H}_{\boldsymbol{y}}^{\varepsilon} \times \widetilde{g}^{s_i y_i} & \text{if } i \notin \mathcal{PCS} \end{cases}$$

Everything else remains the same, and thus this simulation is perfectly indistinguishable from the previous one:

$$\mathsf{Adv}_{\mathbf{G}_4}(\mathcal{A}) = \mathsf{Adv}_{\mathbf{G}_3}(\mathcal{A}).$$

Game $\mathbf{G}_5$: This game is exactly as above, except that $\widetilde{M}_{\eta,\boldsymbol{y}} \xleftarrow{\$} \mathbb{G}_2$ for $\eta = 1, \dots, n-c-1$ and any query $\boldsymbol{y}$. As a consequence, under the MDDH assumption in $\mathbb{G}_2$, $\mathbf{G}_5$ is indistinguishable from the previous one:

$$\mathsf{Adv}_{\mathbf{G}_5}(\mathcal{A}) - \mathsf{Adv}_{\mathbf{G}_4}(\mathcal{A}) \leq \mathsf{Adv}_{\mathbb{G}_2}^{\mathsf{mddh}}(n, q, t),$$

where t is an upper-bound on the execution time of $\mathcal{A}$.

Game $\mathbf{G}_6$: In this game, we once again split the vector space $S = \mathbb{Z}_p^n$ for the key $\boldsymbol{s}$: we consider $S = \mathbb{Z}_p^n$ the space of all the possible secret keys $\boldsymbol{s}$, and S_1 is the space spanned by the m n-vectors in $X^0 - X^1$, while S_0 is the orthogonal. We know that S is the orthogonal direct sum of S_0 and S_1. Let us denote by k the dimension of S_0, then the dimension of S_1 is $n-k$. We do another change of basis in the initialization phase:

- Initialization with $(\boldsymbol{\ell}, X^0, X^1)$, for $X^0, X^1 \in \mathbb{Z}_p^{m \times n}$: $\mathcal{S}$ chooses a k-basis $(\boldsymbol{\sigma}_\eta)_\eta$ of S_0 and an $(n-k)$-basis $(\boldsymbol{\tau}_\eta)_\eta$ of S_1. The vector $\boldsymbol{s}$ is randomly chosen as a random linear combination of $(\boldsymbol{\sigma}_\eta)_\eta$ and $(\boldsymbol{\tau}_\eta)_\eta$: for random scalars $\alpha_\eta, \beta_\eta \xleftarrow{\$} \mathbb{Z}_p$,

$$\boldsymbol{s} = \sum \alpha_\eta \boldsymbol{\sigma}_\eta + \sum \beta_\eta \boldsymbol{\tau}_\eta,$$

and so each component is defined as

$$\begin{aligned} s_i &= \sum \alpha_\eta \langle \boldsymbol{\sigma}_\eta, \boldsymbol{e}_i \rangle + \sum \beta_\eta \langle \boldsymbol{\tau}_\eta, \boldsymbol{e}_i \rangle \\ &= \begin{cases} \sum \alpha_\eta \sigma_{\eta,i} & \text{if } i \in \mathcal{PCS} \\ \sum \alpha_\eta \sigma_{\eta,i} + \sum \beta_\eta \tau_{\eta,i} & \text{if } i \notin \mathcal{PCS} \end{cases} \end{aligned}$$

Indeed, elements in S_1 are combinations of the columns of $X_0 - X_1$ at the (possibly) corrupted positions: these columns are all zeroes. So, for any η, for all $i \in \mathcal{PCS}$, $\tau_{\eta,i} = 0$.

Again, for the sake of clarity, we omit the sets for the sums: for S_1 basis-elements, $\eta = 1, \ldots, n-k$, while for S_0 basis-elements, $\eta = 1, \ldots, k$. Then, $\mathcal{S}$ processes as in $\mathbf{G}_5$, for the explicit and implicit values of $\widetilde{s_i}$, as well as for $\widetilde{dk_{\mathbf{1}}} = \sum_{\eta \in \mathcal{PCS}} \varepsilon_\eta + n\varepsilon \bmod p$. It sets $\mathsf{ek}_i = (s_i, \widetilde{s_i})$ for $i \in \mathcal{PCS}$. It then computes the ciphertexts, for $i = 1, \ldots, n$ and $j = 1, \ldots, m$, as

$$C_{j,i} = \prod g^{r_j \alpha_\eta \sigma_{\eta,i}} \cdot \prod g^{r_j \beta_\eta \tau_{\eta,i}} \cdot g^{x^b_{j,i}}.$$

Again, for all $i \in \mathcal{PCS}$, $C_{j,i} = \prod g^{r_j \alpha_\eta \sigma_{\eta,i}} \cdot g^{x^0_{j,i}} = \prod g^{r_j \alpha_\eta \sigma_{\eta,i}} \cdot g^{x^1_{j,i}}$.

– Functional decryption key queries: for any query $\mathsf{QDKeyGen}(\boldsymbol{y})$, for a vector $\boldsymbol{y}$ and for $i = 1, \ldots, n$, $\mathcal{S}$ computes

$$\widetilde{C_{\boldsymbol{y},i}} = \begin{cases} \widetilde{H}_{\boldsymbol{y}}^{\varepsilon_i + \varepsilon} \times \widetilde{g}^{y_i \sum \alpha_\eta \sigma_{\eta,i}} & \text{if } i \in \mathcal{PCS} \\ \prod \widetilde{M}_{\eta,\boldsymbol{y}}^{\mu_{\eta,i}} \times \widetilde{H}_{\boldsymbol{y}}^{\varepsilon} \times \widetilde{g}^{y_i (\sum \alpha_\eta \sigma_{\eta,i} + \sum \beta_\eta \tau_{\eta,i})} & \text{if } i \notin \mathcal{PCS} \end{cases}$$

– Finalize: same finalization as in $\mathbf{G}_5$.

Since modifications are just formal rewriting, $\mathbf{G}_6$ is perfectly indistinguishable from the previous one:

$$\mathsf{Adv}_{\mathbf{G}_6}(\mathcal{A}) = \mathsf{Adv}_{\mathbf{G}_5}(\mathcal{A}).$$

Game $\mathbf{G}_7$: We still use orthogonal bases for S_0 and S_1 as well as for $\widetilde{S}_0$ and $\widetilde{S}_1$, but while one chooses explicit random scalars $\alpha_\eta \xleftarrow{\$} \mathbb{Z}_p$, one implicitly defines $\beta_\eta \in \mathbb{Z}_p$ as the discrete logarithms of random group elements $B_\eta \xleftarrow{\$} \mathbb{G}_1$.

– Initialization with $(\boldsymbol{\ell}, X^0, X^1)$, for $X^0, X^1 \in \mathbb{Z}_p^{m \times n}$: $\mathcal{S}$ chooses random bases: $(\boldsymbol{\sigma}_\eta)_\eta$ for S_0, $(\boldsymbol{\tau}_\eta)_\eta$ for S_1, $(\boldsymbol{\mu}_\eta)_\eta$ for $\widetilde{S}_1$ while $\widetilde{S}_0$ is spanned by $(\boldsymbol{e}_\eta)_\eta$ and $\mathbf{1}$. For random scalars $\alpha_\eta \xleftarrow{\$} \mathbb{Z}_p$ and random group elements $B_\eta \xleftarrow{\$} \mathbb{G}_1$, one defines β_η as the discrete logarithm of the B_η in basis g to implicitly define $\boldsymbol{s} = \sum \alpha_\eta \boldsymbol{\sigma}_\eta + \sum \beta_\eta \boldsymbol{\tau}_\eta$, or equivalently

$$s_i = \begin{cases} \sum \alpha_\eta \sigma_{\eta,i} & \text{if } i \in \mathcal{PCS} \\ \sum \alpha_\eta \sigma_{\eta,i} + \sum \beta_\eta \tau_{\eta,i} & \text{if } i \notin \mathcal{PCS} \end{cases}$$

$\mathcal{S}$ then computes the ciphertexts as follows, for $i = 1, \ldots, n$ and $j = 1, \ldots, m$:

$$C_{j,i} = \prod g^{r_j \alpha_\eta \sigma_{\eta,i}} \cdot \prod B_\eta^{r_j \tau_{\eta,i}} \cdot g^{x^b_{j,i}}.$$

Again, for all $i \in \mathcal{PCS}$, $C_{j,i} = \prod g^{r_j \alpha_\eta \sigma_{\eta,i}} \cdot g^{x^0_{j,i}} = \prod g^{r_j \alpha_\eta \sigma_{\eta,i}} \cdot g^{x^1_{j,i}}$.

– Functional decryption key queries: for any query $\mathsf{QDKeyGen}(\boldsymbol{y})$, for a vector $\boldsymbol{y}$ and for $i = 1, \ldots, n$, $\mathcal{S}$ computes

$$\widetilde{C_{\boldsymbol{y},i}} = \begin{cases} \widetilde{H}_{\boldsymbol{y}}^{\varepsilon_i + \varepsilon} \times \widetilde{g}^{y_i \sum \alpha_\eta \sigma_{\eta,i}} & \text{if } i \in \mathcal{PCS} \\ \prod \widetilde{M}_{\eta,\boldsymbol{y}}^{\mu_{\eta,i}} \times \widetilde{H}_{\boldsymbol{y}}^{\varepsilon} \times \widetilde{g}^{y_i \sum \alpha_\eta \sigma_{\eta,i}} & \text{if } i \notin \mathcal{PCS} \end{cases}$$

– Corruption: for any sender $i \in \mathcal{PCS}$, $\mathcal{S}$ sends $\mathsf{ek}_i = (s_i = \sum \alpha_\eta \sigma_{\eta,i}, \widetilde{s_i} = \varepsilon_i + \varepsilon)$;

– Finalize: same as in $\mathbf{G}_6$.

Actually, whereas the ciphertext $\widetilde{C_{\boldsymbol{y},i}}$ is unchanged for $i \in \mathcal{PCS}$, it should be different for $i \notin \mathcal{PCS}$. Following the formula from the previous game:

$$\begin{aligned}\widetilde{C_{\boldsymbol{y},i}} &= \prod \widetilde{M}_{\eta,\boldsymbol{y}}^{\mu_{\eta,i}} \times \widetilde{H}_{\boldsymbol{y}}^{\varepsilon} \times \widetilde{g}^{y_i(\sum \alpha_\eta\sigma_{\eta,i}+\sum \beta_\eta\tau_{\eta,i})} \\ &= \prod \widetilde{M}_{\eta,\boldsymbol{y}}^{\mu_{\eta,i}} \times \widetilde{H}_{\boldsymbol{y}}^{\varepsilon} \times \widetilde{g}^{y_i(\sum \alpha_\eta\sigma_{\eta,i})} \prod \widetilde{g}^{\beta_\eta y_i \tau_{\eta,i}} \\ &= \left(\prod \widetilde{M}_{\eta,\boldsymbol{y}}^{\mu_{\eta,i}}\right) \times \left(\prod (\widetilde{g}^{\beta_\eta})^{y_i\tau_{\eta,i}}\right) \times \widetilde{H}_{\boldsymbol{y}}^{\varepsilon} \times \widetilde{g}^{y_i \sum \alpha_\eta\sigma_{\eta,i}}\end{aligned}$$

Let us study $A_{\boldsymbol{y},i} = \prod \widetilde{M}_{\eta,\boldsymbol{y}}^{\mu_{\eta,i}}$. If we write $\widetilde{M}_{\eta,\boldsymbol{y}} = g^{m_{\eta,\boldsymbol{y}}}$, then $A_{\boldsymbol{y},i} = g^{\sum m_{\eta,\boldsymbol{y}}\mu_{\eta,i}}$, where the $m_{\eta,\boldsymbol{y}}$'s are fresh and random scalars (different for any $\boldsymbol{y}$). However, one can remark that $\boldsymbol{\mu}_\eta \in \widetilde{S}_1$ for any η: $\sum_i \mu_{\eta,i} = 0$. Then

$$\prod_i A_{\boldsymbol{y},i} = \prod_i \prod_\eta \widetilde{M}_{\eta,\boldsymbol{y}}^{\mu_{\eta,i}} = \prod_\eta \prod_i \widetilde{M}_{\eta,\boldsymbol{y}}^{\mu_{\eta,i}} = \prod_\eta \widetilde{M}_{\eta,\boldsymbol{y}}^{\sum_i \mu_{\eta,i}} = \prod_\eta \widetilde{M}_{\eta,\boldsymbol{y}}^{0} = 1.$$

Let us consider a random list (A_i) of group elements in $\mathbb{G}_1$ that additionally satisfy $\prod A_i = 1$: if $A_i = g^{a_i}$, we have the system $a_i = \sum m_{\eta,\boldsymbol{y}}\mu_{\eta,i}$, for $i \in \mathcal{PCS}$. Since $(\boldsymbol{\mu}_\eta)_\eta$ is a basis of $\widetilde{S}_1$, there is a unique set of $(m_{\eta,\boldsymbol{y}})_\eta$ that satisfies this system of linear equations. As a consequence, $(A_{\boldsymbol{y},i})_i$ follows a perfectly uniform distribution among the vectors of group elements such that their product is 1.

If we additionally note $(B_{\boldsymbol{y},i} = \prod_\eta \widetilde{g}^{\beta_\eta y_i \tau_{\eta,i}})_i$, similarly, any $\boldsymbol{y}$ being orthogonal to S_1, then $\sum_i y_i\tau_{\eta,i} = 0$:

$$\prod_i B_{\boldsymbol{y},i} = \prod_i \prod_\eta \widetilde{g}^{\beta_\eta y_i \tau_{\eta,i}} = \prod_\eta (\widetilde{g}^{\beta_\eta})^{\sum_i y_i\tau_{\eta,i}} = \prod_\eta (\widetilde{g}^{\beta_\eta})^0 = 1.$$

As a consequence, if one defines $C_{\boldsymbol{y},i} = A_{\boldsymbol{y},i}B_{\boldsymbol{y},i}$, the $C_{\boldsymbol{y},i}$'s follow exactly the same distribution as the $A_{\boldsymbol{y},i}$'s. Therefore, the $\widetilde{C_{\boldsymbol{y},i}}$'s in this game (defined as $C_{\boldsymbol{y},i} \times \widetilde{H}_{\boldsymbol{y}}^{\varepsilon} \times \widetilde{g}^{y_i \sum \alpha_\eta\sigma_{\eta,i}}$) follow exactly the same distribution as in the previous game (defined as $A_{\boldsymbol{y},i} \times \widetilde{H}_{\boldsymbol{y}}^{\varepsilon} \times \widetilde{g}^{y_i \sum \alpha_\eta\sigma_{\eta,i}}$):

$$\mathsf{Adv}_{\mathbf{G}_7}(\mathcal{A}) = \mathsf{Adv}_{\mathbf{G}_6}(\mathcal{A}).$$

In addition, one can note that in this game, the simulation does not use the implicit β_η's.

Game $\mathbf{G}_8$: Now, we are given $B_\eta \overset{\$}{\leftarrow} \mathbb{G}_1$ for $\eta = 1, \ldots, n-k$, as well as $H_j \overset{\$}{\leftarrow} \mathbb{G}_1$, for $j = 1, \ldots, m$, and we denote by $D_{\eta,j}$ the Diffie-Hellman value of any pair B_η and H_j in basis g:

– Hash function queries: for any new query ℓ_j to $\mathcal{H}$, one outputs H_j and adds $(\ell_j, \perp, H_j)$ to Λ);
– Initialization with $(\boldsymbol{\ell}, X^0, X^1)$, for $X^0, X^1 \in \mathbb{Z}_p^{m\times n}$: $\mathcal{S}$ chooses random bases: $(\boldsymbol{\sigma}_\eta)_\eta$ for S_0, $(\boldsymbol{\tau}_\eta)_\eta$ for S_1, $(\boldsymbol{\mu}_\eta)_\eta$ for $\widetilde{S}_1$ while $\widetilde{S}_0$ is spanned by $(\boldsymbol{e}_\eta)_\eta$ and $\mathbf{1}$. For random scalars $\alpha_\eta \overset{\$}{\leftarrow} \mathbb{Z}_p$, one defines $s_i = \sum \alpha_\eta\sigma_{\eta,i}$ for

all $i \in \mathcal{PCS}$. $\mathcal{S}$ then computes the ciphertexts as follows, for $i = 1, \ldots, n$ and $j = 1, \ldots, m$:

$$C_{j,i} = \prod H_j^{\alpha_\eta \sigma_{\eta,i}} \cdot \prod D_{\eta,j}^{\tau_{\eta,i}} \cdot g^{x^b_{j,i}}.$$

– Functional decryption key queries: for any query $\mathsf{QDKeyGen}(\boldsymbol{y})$, for a vector $\boldsymbol{y}$ and for $i = 1, \ldots, n$, $\mathcal{S}$ computes

$$\widetilde{C_{\boldsymbol{y},i}} = \begin{cases} \widetilde{H}_{\boldsymbol{y}}^{\varepsilon_i + \varepsilon} \times \widetilde{g}^{y_i \sum \alpha_\eta \sigma_{\eta,i}} & \text{if } i \in \mathcal{PCS} \\ \prod \widetilde{M}_{\eta,\boldsymbol{y}}^{\mu_{\eta,i}} \times \widetilde{H}_{\boldsymbol{y}}^{\varepsilon} \times \widetilde{g}^{y_i \sum \alpha_\eta \sigma_{\eta,i}} & \text{if } i \notin \mathcal{PCS} \end{cases}$$

– Corruption: for any sender $i \in \mathcal{PCS}$, $\mathcal{S}$ sends $\mathsf{ek}_i = (s_i = \sum \alpha_\eta \sigma_{\eta,i}, \widetilde{s}_i = \varepsilon_i + \varepsilon)$;

– Finalize: same as in $\mathbf{G}_7$.

For all j, if we write $H_j = g^{r_j}$, then $D_{\eta,j} = B_\eta^{r_j}$, and thus the ciphertexts $C_{j,i}$ follow exactly the same distribution as in the previous game:

$$\mathsf{Adv}_{\mathbf{G}_8}(\mathcal{A}) = \mathsf{Adv}_{\mathbf{G}_7}(\mathcal{A}).$$

Game $\mathbf{G}_9$: This game is exactly as above, except that $D_{\eta,j} \xleftarrow{\$} \mathbb{G}_1$, for $\eta = 1, \ldots, n-k$ and $j = 1, \ldots, m$. Using the MDDH assumption in $\mathbb{G}_1$, $\mathbf{G}_9$ is indistinguishable from the previous one:

$$\mathsf{Adv}_{\mathbf{G}_9}(\mathcal{A}) - \mathsf{Adv}_{\mathbf{G}_8}(\mathcal{A}) \leq \mathsf{Adv}^{\mathsf{mddh}}_{\mathbb{G}_1}(n, m, t),$$

where t is an upper-bound on the execution time of $\mathcal{A}$.

Let us now summarize this last game:

– Hash function queries: $\mathcal{S}$ sets two empty lists Λ and $\widetilde{\Lambda}$ of triples. For any query ℓ to $\mathcal{H}$, $\mathcal{S}$ looks for a triple $(\ell, \star, h) \in \Lambda$. If such a triple exists, it outputs h, otherwise it chooses a random $h \xleftarrow{\$} \mathbb{G}_1$, stores $(\ell, \perp, h)$ in Λ, and outputs h. The process is the same for any query to $\widetilde{\mathcal{H}}$ except that the values $\widetilde{h}$ are chosen in $\mathbb{G}_2$ instead;

– Initialization with $(\boldsymbol{\ell}, X^0, X^1)$, for $X^0, X^1 \in \mathbb{Z}_p^{m \times n}$: we denote by $\mathcal{PCS}$ the set of possibly corrupted senders (this set is that of the indexes of zero-columns in $X_0 - X_1$), which is of cardinal c.
  - We denote by S_1 the space spanned by the m n-vectors in $X^0 - X^1$, while S_0 is the orthogonal in $S = \mathbb{Z}_p^n$;
  - We denote by $\widetilde{S}_0$ the space spanned by the $c+1$ n-vectors $\{\mathbf{1}, (\boldsymbol{e}_i)_{i \in \mathcal{PCS}}\}$, while $\widetilde{S}_1$ is the orthogonal in $S = \mathbb{Z}_p^n$;
  - We denote by k the dimension of S_0, then the dimension of S_1 is $n - k$;
  - $\mathcal{S}$ chooses a k-basis $(\boldsymbol{\sigma}_\eta)_\eta$ of S_0 and an $(n-k)$-basis $(\boldsymbol{\tau}_\eta)_\eta$ of S_1, as well as k random scalars $\alpha_\eta \xleftarrow{\$} \mathbb{Z}_p$, for $\eta = 1, \ldots, k$;
  - $\mathcal{S}$ chooses a $(n-c-1)$-basis $(\boldsymbol{\mu}_\eta)_\eta$ of $\widetilde{S}_1$, while $\{\mathbf{1}, (\boldsymbol{e}_i)_{i \in \mathcal{PCS}}\}$ is a $(c+1)$-basis of $\widetilde{S}_0$, as well as c random scalars $\varepsilon_\eta \xleftarrow{\$} \mathbb{Z}_p$, for $\eta = 1, \ldots, c$ and $\varepsilon \leftarrow \mathbb{Z}_p$;

- $\mathcal{S}$ then computes the ciphertexts as follows, for $i = 1, \ldots, n$ and $j = 1, \ldots, m$:
$$C_{j,i} = \prod H_j^{\alpha_\eta \sigma_{\eta,i}} \cdot \prod D_{\eta,j}^{\tau_{\eta,i}} \cdot g^{x^b_{j,i}}.$$
where $H_j = \mathcal{H}(\lambda_j) \xleftarrow{\$} \mathbb{G}_1$, and $D_{\eta,j} \xleftarrow{\$} \mathbb{G}_1$, for $\eta = 1, \ldots, n-k$.

– Functional decryption key queries: for any query $\mathsf{QDKeyGen}(\boldsymbol{y})$, for a vector $\boldsymbol{y}$ and for $i = 1, \ldots, n$, $\mathcal{S}$ computes:
$$\widetilde{C_{\boldsymbol{y},i}} = \begin{cases} \widetilde{H}_{\boldsymbol{y}}^{\varepsilon_i + \varepsilon} \times \widetilde{g}^{y_i \sum \alpha_\eta \sigma_{\eta,i}} & \text{if } i \in \mathcal{PCS} \\ \prod \widetilde{M}_{\eta,\boldsymbol{y}}^{\mu_{\eta,i}} \times \widetilde{H}_{\boldsymbol{y}}^{\varepsilon} \times \widetilde{g}^{y_i \sum \alpha_\eta \sigma_{\eta,i}} & \text{if } i \notin \mathcal{PCS} \end{cases}$$
where $\widetilde{H}_{\boldsymbol{y}} = \widetilde{\mathcal{H}}(\boldsymbol{y}) \xleftarrow{\$} \mathbb{G}_2$, and $\widetilde{M}_{\eta,\boldsymbol{y}} \xleftarrow{\$} \mathbb{G}_2$, for $\eta = 1, \ldots, n-c-1$.

– Corruption: for any sender $i \in \mathcal{PCS}$, $\mathcal{S}$ sends $\mathsf{ek}_i = (s_i = \sum \alpha_\eta \sigma_{\eta,i}, \widetilde{s_i} = \varepsilon_i + \varepsilon)$;

– Finalize: $\mathcal{A}$ provides its guess b' on the bit b and $\mathcal{S}$ filters the cases were $\mathcal{A}$ requests unauthorized dk or corruptions by setting $\beta \xleftarrow{\$} \{0,1\}$. In the other cases, $\mathcal{S}$ sets $\beta \leftarrow b'$.

Finally, the only leakage about b from this last game is in the ciphertexts $\{C_{i,j}\}$:
$$C_{j,i} = \prod_{\eta=1}^{k} H_j^{\alpha_\eta \sigma_{\eta,i}} \cdot \prod_{\eta=1}^{n-k} D_{\eta,j}^{\tau_{\eta,i}} \cdot g^{x^b_{j,i}} = \prod_{\eta=1}^{k} H_j^{\alpha_\eta \sigma_{\eta,i}} \cdot \prod_{\eta=1}^{n-k} D_{\eta,j}^{\tau_{\eta,i}} \cdot g^{x^{1-b}_{j,i}} \cdot g^{x^b_{j,i} - x^{1-b}_{j,i}}.$$

But $\boldsymbol{x}_j^b - \boldsymbol{x}_j^{1-b} \in S_1$, and can thus be written as $\sum_\eta \xi_\eta \boldsymbol{\tau}_\eta$:
$$g^{x^b_{j,i} - x^{1-b}_{j,i}} = g^{\sum_\eta \xi_\eta \tau_{\eta,i}} = \prod_\eta (g^{\xi_\eta})^{\tau_{\eta,i}}.$$

As a consequence,
$$C_{j,i} = \prod_{\eta=1}^{k} H_j^{\alpha_\eta \sigma_{\eta,i}} \cdot \prod_{\eta=1}^{n-k} (D_{\eta,j} \cdot g^{\xi_\eta})^{\tau_{\eta,i}} \cdot g^{x^{1-b}_{j,i}} = \prod_{\eta=1}^{k} H_j^{\alpha_\eta \sigma_{\eta,i}} \cdot \prod_{\eta=1}^{n-k} E_{\eta,j}^{\tau_{\eta,i}} \cdot g^{x^{1-b}_{j,i}}$$

where $E_{\eta,j} = D_{\eta,j} \cdot g^{\xi_\eta}$ for $\eta = 1, \ldots, n-k$ and $j = 1, \ldots, m$. When the $(D_{\eta,j})$'s all follow independent uniform distributions in $\mathbb{G}_1$, the $(E_{\eta,j})$'s all do so as well. As a consequence, the ciphertexts from honest senders do not leak any information about b, and thus the advantage of any adversary (even powerful) in this game is 0: $\mathsf{Adv}_{\mathbf{G}_9}(\mathcal{A}) = 0$.

Eventually, by combining all the gaps, one gets
$$\mathsf{Adv}_{\mathbf{G}_0}(\mathcal{A}) \leq \mathsf{Adv}^{\mathsf{mddh}}_{\mathbb{G}_1}(n, m, t) + \mathsf{Adv}^{\mathsf{mddh}}_{\mathbb{G}_2}(n, q, t).$$

7 Conclusion

Multi-Client/Multi-Input Functional Encryption and Decentralized Cryptosystems are invaluable tools for many emerging applications such as cloud services or big data. These applications often involve many parties who contribute their data to enable the extraction of knowledge, while protecting their individual privacy with minimal trust in the other parties, including any central authority. We make an important step towards combining the desired functionalities and properties by introducing the notion of Decentralized Multi-Client Functional Encryption. It opens some interesting directions:

– Our generic construction of Decentralized Multi-Client Functional Encryption is quite general, and not restricted to the inner-product setting. Therefore, new constructions of Multi-Client Functional Encryption schemes can benefit from our work by immediately yielding efficient decentralized schemes. When considering the inner-product function, existing constructions would almost fit the requirements of the generic construction, except some restrictions on the plaintext. In particular, it is often required the inner-product to be small. We overcome this issue in DDH-based IP-DMCFE by using pairings. It is an interesting problem to consider whether the LWE-based and DCR-based schemes can be adapted to fit our generic construction.
– Getting all the desired properties, namely efficiency, new functionalities and the strongest security level, is a challenging problem. One of the main challenges is to construct an efficient, non-interactive DMCFE which is adaptively secure, for a larger class of functions than that of inner-product functions.

Acknowledgments.

This work was supported in part by the European Community's Seventh Framework Programme (FP7/2007-2013 Grant Agreement no. 339563 – CryptoCloud).

References

1. 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)
2. 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 (May 2017)
3. 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)
4. Ananth, P., Brakerski, Z., Segev, G., Vaikuntanathan, V.: From selective to adaptive security in functional encryption. In: Gennaro, R., Robshaw, M.J.B. (eds.) CRYPTO 2015, Part II. LNCS, vol. 9216, pp. 657–677. Springer, Heidelberg (Aug 2015)

5. Badrinarayanan, S., Goyal, V., Jain, A., Sahai, A.: Verifiable functional encryption. In: Cheon, J.H., Takagi, T. (eds.) ASIACRYPT 2016, Part II. LNCS, vol. 10032, pp. 557–587. Springer, Heidelberg (Dec 2016)
6. 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)
7. 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)
8. 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)
9. Brakerski, Z., Komargodski, I., Segev, G.: Multi-input functional encryption in the private-key setting: Stronger security from weaker assumptions. In: Fischlin, M., Coron, J.S. (eds.) EUROCRYPT 2016, Part II. LNCS, vol. 9666, pp. 852–880. Springer, Heidelberg (May 2016)
10. 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)
11. Emura, K.: Privacy-preserving aggregation of time-series data with public verifiability from simple assumptions. Cryptology ePrint Archive, Report 2017/479 (2017), http://eprint.iacr.org/2017/479
12. Garg, S., Gentry, C., Halevi, S., Raykova, M., Sahai, A., Waters, B.: Candidate indistinguishability obfuscation and functional encryption for all circuits. In: 54th FOCS. pp. 40–49. IEEE Computer Society Press (Oct 2013)
13. 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)
14. Goldwasser, S., Goyal, V., Jain, A., Sahai, A.: Multi-input functional encryption. Cryptology ePrint Archive, Report 2013/727 (2013), http://eprint.iacr.org/2013/727
15. Goldwasser, S., Kalai, Y.T., Popa, R.A., Vaikuntanathan, V., Zeldovich, N.: How to run turing machines on encrypted data. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part II. LNCS, vol. 8043, pp. 536–553. Springer, Heidelberg (Aug 2013)
16. Goldwasser, S., Kalai, Y.T., Popa, R.A., Vaikuntanathan, V., Zeldovich, N.: Reusable garbled circuits and succinct functional encryption. In: Boneh, D., Roughgarden, T., Feigenbaum, J. (eds.) 45th ACM STOC. pp. 555–564. ACM Press (Jun 2013)
17. Gorbunov, S., Vaikuntanathan, V., Wee, H.: Functional encryption with bounded collusions via multi-party computation. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO 2012. LNCS, vol. 7417, pp. 162–179. Springer, Heidelberg (Aug 2012)
18. Gordon, S.D., Katz, J., Liu, F.H., Shi, E., Zhou, H.S.: Multi-input functional encryption. Cryptology ePrint Archive, Report 2013/774 (2013), http://eprint.iacr.org/2013/774
19. 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)
20. Li, Q., Cao, G.: Efficient and privacy-preserving data aggregation in mobile sensing. In: ICNP 2012. pp. 1–10. IEEE Computer Society (2012)

21. Li, Q., Cao, G.: Efficient privacy-preserving stream aggregation in mobile sensing with low aggregation error. In: De Cristofaro, E., Wright, M. (eds.) PETS 2013. LNCS, vol. 7981, pp. 60–81. Springer Berlin Heidelberg, Berlin, Heidelberg (2013)
22. Paillier, P.: Public-key cryptosystems based on composite degree residuosity classes. In: Stern, J. (ed.) EUROCRYPT'99. LNCS, vol. 1592, pp. 223–238. Springer, Heidelberg (May 1999)
23. Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. In: Gabow, H.N., Fagin, R. (eds.) 37th ACM STOC. pp. 84–93. ACM Press (May 2005)
24. Sahai, A., Seyalioglu, H.: Worry-free encryption: functional encryption with public keys. In: Al-Shaer, E., Keromytis, A.D., Shmatikov, V. (eds.) ACM CCS 10. pp. 463–472. ACM Press (Oct 2010)
25. 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)
26. 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)
27. Waters, B.: A punctured programming approach to adaptively secure functional encryption. In: Gennaro, R., Robshaw, M.J.B. (eds.) CRYPTO 2015, Part II. LNCS, vol. 9216, pp. 678–697. Springer, Heidelberg (Aug 2015)

A Proof of Lemma 7

In order to prove this Lemma, one can first note the indistinguishability of

$$\begin{aligned}\mathcal{D} &= \{(X, Y, Z = \mathsf{CDH}(X, Y)) \mid X, Y \xleftarrow{\$} \mathbb{G}\}\\ \mathcal{D}' &= \{(X, Y, Z) \mid X, Y, Z \xleftarrow{\$} \mathbb{G}\}.\end{aligned}$$

under the DDH assumption. Then, we also have the indistinguishability of multiple-distribution, with $j = 1, \ldots, m$:

$$\begin{aligned}\mathcal{D}_m &= \{(X, (Y_j = g^{u_j} Y^{v_j}, Z_j = X^{u_j} \cdot \mathsf{CDH}(X, Y)^{v_j})_j) \mid X, Y \xleftarrow{\$} \mathbb{G}, u_j, v_j \xleftarrow{\$} \mathbb{Z}_p\}\\ \mathcal{D}'_m &= \{(X, (Y_j = g^{u_j} Y^{v_j}, Z_j = X^{u_j} \cdot Z^{v_j})_j) \mid X, Y, Z \xleftarrow{\$} \mathbb{G}, u_j, v_j \xleftarrow{\$} \mathbb{Z}_p\}.\end{aligned}$$

More precisely, the generation of these distributions for the above distribution of (X, Y, Z) requires 4 additional exponentiations per index j. But distinguishing the latter distributions is equivalent to distinguish the former distributions.

$$\mathsf{Adv}^{\mathcal{D}_m, \mathcal{D}'_m}(\mathcal{A}) \leq \mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}}(t + 4m \times t_{\mathbb{G}}),$$

where $t_{\mathbb{G}}$ is the time for an exponentiation in $\mathbb{G}$, and t is an upper-bound on the running time of any distinguisher $\mathcal{A}$. Furthermore, we can rewrite $\mathcal{D}_m$ and $\mathcal{D}'_m$:

$$\begin{aligned}\mathcal{D}_m &= \{(X, (Y_j, Z_j = \mathsf{CDH}(X, Y_j))_j) \mid X, Y_j \xleftarrow{\$} \mathbb{G}\}\\ \mathcal{D}'_m &= \{(X, (Y_j, Z_j)_j) \mid X, Y_j, Z_j \xleftarrow{\$} \mathbb{G}\}.\end{aligned}$$

Let us now consider the distributions

$$\mathcal{D}_0 = \{((X_i)_i, (Y_j)_j, (Z_{i,j} = \mathsf{CDH}(X_i, Y_j))_{i,j}) \mid X_i, Y_j \xleftarrow{\$} \mathbb{G}\}$$
$$\mathcal{D}_n = \{((X_i)_i, (Y_j)_j, (Z_{i,j})_{i,j}) \mid X_i, Y_j, Z_{i,j} \xleftarrow{\$} \mathbb{G}\}.$$

as well as the hybrid distribution

$$\mathcal{D}_k = \left\{ \left((X_i)_i, (Y_j)_j, \begin{pmatrix} \mathsf{CDH}(X_i, Y_j))_{i,j}) & \text{for } i > k \\ Z_{i,j} & \text{for } i \leq k \end{pmatrix}_{i,j} \right) \middle| \, X_i, Y_j, Z_{i,j} \xleftarrow{\$} \mathbb{G} \right\}.$$

This distribution is indeed the above $\mathcal{D}_0$ for $k = 0$ and $\mathcal{D}_n$ for $k = n$.

Let us be given a tuple $(X, (Y_j, Z_j)_j)$ that comes either from $\mathcal{D}_m$ or from $\mathcal{D}'_m$, and we build the tuple

$$\left(\begin{pmatrix} X_i & \text{with } X_i \xleftarrow{\$} \mathbb{G} \\ X & \\ g^{x_i} & \text{with } r_i \xleftarrow{\$} \mathbb{Z}_p \end{pmatrix}_i, (Y_j)_j, \begin{pmatrix} Z_{i,j} & \text{with } Z_{i,j} \xleftarrow{\$} \mathbb{G} \\ Z_j & \\ Y_j^{x_i} & \end{pmatrix}_{i,j} \right) \quad \begin{matrix} \text{for } i < k \\ \text{for } i = k \\ \text{for } i > k \end{matrix}.$$

One can remark that if the initial tuple comes from $\mathcal{D}'_m$, then the new tuple follows distribution $\mathcal{D}_k$, whereas if it comes from $\mathcal{D}_m$, the new tuple follows distribution $\mathcal{D}_{k-1}$. Hence, for any distinguisher $\mathcal{A}$ against the distributions $\mathcal{D}_0$ and $\mathcal{D}_n$, within running time bounded by t, and for any k,

$$\mathsf{Adv}^{\mathcal{D}_{k-1}, \mathcal{D}_k}(\mathcal{A}) \leq \mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}}(t + 4m \times t_{\mathbb{G}}).$$

Using the triangular inequality:

$$\mathsf{Adv}^{\mathcal{D}_0, \mathcal{D}_n}(\mathcal{A}) \leq n \times \mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}}(t + 4m \times t_{\mathbb{G}}).$$

Taking the maximum among all the adversaries withing running time bounded by t:

$$\mathsf{Adv}^{\mathsf{mddh}}_{\mathbb{G}}(n, m, t) \leq n \times \mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}}(t + 4m \times t_{\mathbb{G}}).$$

B Proof of Theorem 8

The proof consists of a series of games, that starts with the real game, in the random oracle model. In $\mathbf{G}_1$, the simulator takes total control of the random oracle. In $\mathbf{G}_2$, the simulator splits the vector space into two orthogonal subspaces for describing the keys: $\langle X^0 - X^1 \rangle \oplus \langle X^0 - X^1 \rangle^\perp$. $\mathbf{G}_3$ and $\mathbf{G}_4$ make appear CDH tuples, and the DDH assumption is applied for $\mathbf{G}_5$. In the last game, the advantage of any adversary is 0.

Game $\mathbf{G}_0$: The first game is the real game where the simulator $\mathcal{S}$ perfectly simulates the view of the adversary honestly generating the secret keys, with a given group $\mathbb{G}$ of prime order p, and a generator g, in the random oracle for the full-domain hash function $\mathcal{H}$:

– Hash function queries: since we are in the random oracle model, $\mathcal{S}$ sets an empty list Λ of triples. For any query ℓ to $\mathcal{H}$, $\mathcal{S}$ looks for a triple $(\ell, \star, h) \in \Lambda$. If such a triple exists, it outputs h, otherwise it chooses a random $h \xleftarrow{\$} \mathbb{G}$, stores $(\ell, \perp, h)$ in Λ, and outputs h;
– Initialization with $(\boldsymbol{\ell}, X^0, X^1)$, with $X^0, X^1 \in \mathbb{Z}_p^{m \times n}$: $\mathcal{S}$ sets $\mathsf{msk} = (\mathbb{G}, p, g, \mathcal{H}, \boldsymbol{s})$ for $\boldsymbol{s} \xleftarrow{\$} \mathbb{Z}_p^n$ to emulate the SetUp algorithm. Then, $\mathcal{S}$ extracts the n encryption keys $\mathsf{ek}_i = s_i$, for $i = 1, \ldots, n$ and chooses a random bit $b \xleftarrow{\$} \{0, 1\}$. It also emulates $\mathsf{Encrypt}$ by setting and outputting $C_{j,i} = \mathcal{H}(\ell_j)^{s_i} \cdot g^{x^b_{j,i}}$, for valid inputs, for $j = 1, \ldots, m$ and $i = 1, \ldots, n$;
– Functional decryption key queries: for any query $\mathsf{QDKeyGen}(\boldsymbol{y})$, for a vector $\boldsymbol{y}$, $\mathcal{S}$ computes $\mathsf{dk}_{\boldsymbol{y}} = (\mathbb{G}, p, g, \mathcal{H}, \boldsymbol{y}, \langle \boldsymbol{s}, \boldsymbol{y} \rangle)$;
– Corruption: for a sender i, $\mathcal{S}$ sends back ek_i;
– Finalize: $\mathcal{A}$ provides its guess b' on the bit b and $\mathcal{S}$ filters the cases were $\mathcal{A}$ requests unauthorized dk or corruptions queries by setting $\beta \xleftarrow{\$} \{0, 1\}$. In the other cases, $\mathcal{S}$ sets $\beta \leftarrow b'$.

We define the advantage of $\mathcal{A}$ at the end of this game as

$$\mathsf{Adv}_{\mathbf{G}_0}(\mathcal{A}) = |\Pr[\beta = 1|b = 1] - \Pr[\beta = 1|b = 0]| = \mathsf{Adv}^{\mathtt{sel\text{-}IND}}(\mathcal{A}).$$

Game $\mathbf{G}_1$: In this game, we just modify the simulation of the random oracle.
– Hash function queries: $\mathcal{S}$ sets an empty list Λ of triples. For any query ℓ to $\mathcal{H}$, $\mathcal{S}$ looks for a triple $(\ell, \star, h) \in \Lambda$. If such a triple exists, it outputs h, otherwise it chooses a random $r \xleftarrow{\$} \mathbb{Z}_p$, sets $h \leftarrow g^r$, stores (ℓ, r, h) in Λ, and outputs h.

This simulation is perfectly indistinguishable from the previous one:

$$\mathsf{Adv}_{\mathbf{G}_1}(\mathcal{A}) = \mathsf{Adv}_{\mathbf{G}_0}(\mathcal{A}).$$

But one can note that, now, for every ciphertext on valid inputs, for $i = 1, \ldots, n$ and $j = 1, \ldots, m$,

$$C_{j,i} = \mathcal{H}(\ell_j)^{s_i} \cdot g^{x^b_{j,i}} = g^{r_j s_i} \cdot g^{x^b_{j,i}}, \text{ since } \mathcal{H}(\ell_j) = g^{r_j}.$$

Game $\mathbf{G}_2$: In this game, we split the vector space according to the two matrices X^0 and X^1. More precisely, we consider $S = \mathbb{Z}_p^n$ the space of all the possible master secret keys $\boldsymbol{s}$, and S_1 is the space spanned by the m n-vectors in $X^0 - X^1$ while S_0 is the orthogonal. We know that S is the orthogonal direct sum of S_0 and S_1. Let us denote by k the dimension of S_0, then the dimension of S_1 is $n - k$. This change of basis principally affects the initialization, but we add some remarks about consequences on the other phases:
– Initialization with $(\boldsymbol{\ell}, X^0, X^1)$, with $X^0, X^1 \in \mathbb{Z}_p^{m \times n}$: $\mathcal{S}$ chooses a k-basis $(\boldsymbol{\sigma}_\eta)_\eta$ of S_0 and an $(n-k)$-basis $(\boldsymbol{\tau}_\eta)_\eta$ of S_1. The master secret key is randomly chosen as a random linear combination of $(\boldsymbol{\sigma}_\eta)_\eta$ and $(\boldsymbol{\tau}_\eta)_\eta$: for random scalars $\alpha_\eta, \beta_\eta \xleftarrow{\$} \mathbb{Z}_p$,

$$\boldsymbol{s} = \sum_{\eta=1}^{k} \alpha_\eta \boldsymbol{\sigma}_\eta + \sum_{\eta=1}^{n-k} \beta_\eta \boldsymbol{\tau}_\eta : \text{ for } i = 1, \ldots, n, s_i = \sum_{\eta=1}^{k} \alpha_\eta \sigma_{\eta,i} + \sum_{\eta=1}^{n-k} \beta_\eta \tau_{\eta,i}.$$

Then, $\mathcal{S}$ processes as in $\mathbf{G}_1$, setting the keys $\mathsf{ek}_i = s_i$ and computing the ciphertexts as above. One can note that, for every ciphertext, for $i = 1, \ldots, n$ and $j = 1, \ldots, m$, we have:

$$\begin{aligned} C_{j,i} = g^{r_j s_i} \cdot g^{x^b_{j,i}} &= g^{r_j(\sum_{\eta=1}^{k} \alpha_\eta \sigma_{\eta,i} + \sum_{\eta=1}^{n-k} \beta_\eta \tau_{\eta,i})} \cdot g^{x^b_{j,i}} \\ &= \prod_{\eta=1}^{k} g^{r_j \alpha_\eta \sigma_{\eta,i}} \cdot \prod_{\eta=1}^{n-k} g^{r_j \beta_\eta \tau_{\eta,i}} \cdot g^{x^b_{j,i}} \\ &= \prod_{\eta=1}^{k} (g^{r_j})^{\alpha_\eta \sigma_{\eta,i}} \cdot \prod_{\eta=1}^{n-k} (g^{r_j \beta_\eta})^{\tau_{\eta,i}} \cdot g^{x^b_{j,i}}. \end{aligned}$$

– Functional decryption key queries: same key query as in $\mathbf{G}_1$. One should note that for any authorized query $\boldsymbol{y}$, $(X^0 - X^1) \cdot \boldsymbol{y} = 0$, and so $\boldsymbol{y} \in S_0$. Thus the functional keys only uses the $\boldsymbol{\sigma}_\eta$ vectors and the α_η coefficients, since any query $\boldsymbol{y} \in S_0 = S_1^\perp$:

$$\begin{aligned} \langle \boldsymbol{s}, \boldsymbol{y} \rangle &= \langle \sum_{\eta=1}^{k} \alpha_\eta \boldsymbol{\sigma}_\eta + \sum_{\eta=1}^{n-k} \beta_\eta \boldsymbol{\tau}_\eta, \boldsymbol{y} \rangle \\ &= \sum_{\eta=1}^{k} \alpha_\eta \langle \boldsymbol{\sigma}_\eta, \boldsymbol{y} \rangle + \sum_{\eta=1}^{n-k} \beta_\eta \langle \boldsymbol{\tau}_\eta, \boldsymbol{y} \rangle = \sum_{\eta=1}^{k} \alpha_\eta \langle \boldsymbol{\sigma}_\eta, \boldsymbol{y} \rangle \end{aligned}$$

– Finalize: same finalization as in $\mathbf{G}_1$. One should note the Finalize counts the game if, for any corrupted sender $i \in \mathcal{CS}$, the i-th column of $X^0 - X^1$ is 0. This condition implies $\tau_{\eta,i} = 0$ for $\eta = 1, \ldots, n-k$, and the corresponding ek_i is:

$$\mathsf{ek}_i = s_i = \sum_{\eta=1}^{k} \alpha_\eta \sigma_{\eta,i}$$

Thus, if the corruption is valid, it does not leak any information about the S_0 components α_η of $\boldsymbol{s}$.

All the rest remains unchanged, so this simulation is perfectly indistinguishable from the previous one:

$$\mathsf{Adv}_{\mathbf{G}_2}(\mathcal{A}) = \mathsf{Adv}_{\mathbf{G}_1}(\mathcal{A}).$$

Game $\mathbf{G}_3$: We still use a k-basis $(\boldsymbol{\sigma}_\eta)_\eta$ of S_0 and an $(n-k)$-basis $(\boldsymbol{\tau}_\eta)_\eta$ of S_1. But for the master secret key, one just chooses random scalars $\alpha_\eta \xleftarrow{\$} \mathbb{Z}_p$, while the scalars $\beta_\eta \in \mathbb{Z}_p$ are implicitly defined by random group elements $B_\eta \xleftarrow{\$} \mathbb{G}$.

– Initialization with $(\boldsymbol{\ell}, X^0, X^1)$, with $X^0, X^1 \in \mathbb{Z}_p^{m \times n}$: $\mathcal{S}$ chooses a k-basis $(\boldsymbol{\sigma}_\eta)_\eta$ of S_0 and an $(n-k)$-basis $(\boldsymbol{\tau}_\eta)_\eta$ of S_1. The master secret key is randomly chosen as a random linear combination of $(\boldsymbol{\sigma}_\eta)_\eta$ and $(\boldsymbol{\tau}_\eta)_\eta$: for random scalars $\alpha_\eta \xleftarrow{\$} \mathbb{Z}_p$ and random group elements $B_\eta \xleftarrow{\$} \mathbb{G}$, we define β_η as the discrete logarithm of the B_η in basis g to implicitly describe:

$$\boldsymbol{s} = \sum_{\eta=1}^{k} \alpha_\eta \boldsymbol{\sigma}_\eta + \sum_{\eta=1}^{n-k} \beta_\eta \boldsymbol{\tau}_\eta : \text{ for } i = 1, \ldots, n, s_i = \sum_{\eta=1}^{k} \alpha_\eta \sigma_{\eta,i} + \sum_{\eta=1}^{n-k} \beta_\eta \tau_{\eta,i}.$$

$\mathcal{S}$ then computes the ciphertexts as follows, for $i = 1, \ldots, n$ and $j = 1, \ldots, m$,:

$$\begin{aligned} C_{j,i} &= \prod_{\eta=1}^{k} (g^{r_j})^{\alpha_\eta \sigma_{\eta,i}} \cdot \prod_{\eta=1}^{n-k} (B_\eta^{r_j})^{\tau_{\eta,i}} \cdot g^{x^b_{j,i}} \\ &= \prod_{\eta=1}^{k} (g^{r_j})^{\alpha_\eta \sigma_{\eta,i}} \cdot \prod_{\eta=1}^{n-k} (g^{r_j \beta_\eta})^{\tau_{\eta,i}} \cdot g^{x^b_{j,i}} \end{aligned}$$

– Functional decryption key queries: for any query $\mathsf{QDKeyGen}(\boldsymbol{y})$, for a vector $\boldsymbol{y}$, $\mathcal{S}$ computes $\mathsf{dk}_{\boldsymbol{y}} = (\mathbb{G}, p, g, \mathcal{H}, \boldsymbol{y}, \langle \boldsymbol{s}, \boldsymbol{y} \rangle = \sum_{\eta=1}^{k} \alpha_\eta \langle \boldsymbol{\sigma}_\eta, \boldsymbol{y} \rangle)$. $\mathcal{S}$ computes $\langle \boldsymbol{s}, \boldsymbol{y} \rangle$ with the explicit part of the key only.
– Corruption: for a sender i, $\mathcal{S}$ sends $\mathsf{ek}_i = \sum_{\eta=1}^{k} \alpha_\eta \sigma_{\eta,i}$
– Finalize: same as in $\mathbf{G}_2$.

Since the explicit values of the scalars (β_η) are not needed, this simulation is perfectly indistinguishable from the previous one:

$$\mathsf{Adv}_{\mathbf{G}_3}(\mathcal{A}) = \mathsf{Adv}_{\mathbf{G}_2}(\mathcal{A}).$$

Game $\mathbf{G}_4$: Now, we are given random group elements $B_\eta \xleftarrow{\$} \mathbb{G}$, for $\eta = 1, \ldots, n-k$, as well as $H_j \xleftarrow{\$} \mathbb{G}$, for $j = 1, \ldots, m$. In addition, we denote by $D_{\eta,j}$ the Diffie-Hellman value of B_η and H_j in basis g, for $\eta = 1, \ldots, n-k$ and $j = 1, \ldots, m$. During the initialization, we program the random oracle, with $\mathcal{H}(\ell_j) \leftarrow H_j$, for $j = 1, \ldots, m$ (and add $(\ell_j, \perp, H_j)$ to Λ), and the ciphertexts use theses hash values:

– Initialization with $(\boldsymbol{\ell}, X^0, X^1)$, with $X^0, X^1 \in \mathbb{Z}_p^{m \times n}$: $\mathcal{S}$ chooses msk and sets the ek_i as in $\mathbf{G}_3$, then computes the ciphertexts as follows, for $i = 1, \ldots, n$ and $j = 1, \ldots, m$,:

$$C_{j,i} = \prod_{\eta=1}^{k} H_j^{\alpha_\eta \sigma_{\eta,i}} \cdot \prod_{\eta=1}^{n-k} D_{\eta,j}^{\tau_{\eta,i}} \cdot g^{x^b_{j,i}} = \prod_{\eta=1}^{k} (g^{r_j})^{\alpha_\eta \sigma_{\eta,i}} \cdot \prod_{\eta=1}^{n-k} (B_\eta^{r_j})^{\tau_{\eta,i}} \cdot g^{x^b_{j,i}},$$

if r_j is the discrete logarithm of H_j in basis g. Everything else remains the same, and so this simulation is perfectly indistinguishable from the previous one:

$$\mathsf{Adv}_{\mathbf{G}_4}(\mathcal{A}) = \mathsf{Adv}_{\mathbf{G}_3}(\mathcal{A}).$$

Game $\mathbf{G}_5$: This game is exactly as above, except that $D_{\eta,j} \xleftarrow{\$} \mathbb{G}$, for $\eta = 1, \ldots, n-k$ and $j = 1, \ldots, m$. As a consequence, under the DDH assumption, $\mathbf{G}_5$ is indistinguishable from the previous one:

$$\mathsf{Adv}_{\mathbf{G}_5}(\mathcal{A}) - \mathsf{Adv}_{\mathbf{G}_4}(\mathcal{A}) \leq n \times \mathsf{Adv}^{\mathsf{ddh}}(t + 4mt_e),$$

where t is an upper-bound on the execution time of $\mathcal{A}$.

Let us now recapitulate the description of this last game:

- Hash function queries: since we are in the random oracle model, $\mathcal{S}$ sets an empty list Λ of triples. For any query ℓ to $\mathcal{H}$, $\mathcal{S}$ looks for a triple $(\ell, \star, h) \in \Lambda$. If such a triple exists, it outputs h, otherwise it chooses a random $h \overset{\$}{\leftarrow} \mathbb{G}$, stores $(\ell, \perp, h)$ in Λ, and outputs h;
- Initialization with $(\boldsymbol{\ell}, X^0, X^1)$, with $X^0, X^1 \in \mathbb{Z}_p^{m \times n}$: We denote by S_1 the space spanned by the m n-vectors in $X^0 - X^1$ while S_0 is the orthogonal. Let us denote by k the dimension of S_0, then the dimension of S_1 is $n-k$. $\mathcal{S}$ chooses a k-basis $(\boldsymbol{\sigma}_\eta)_\eta$ of S_0 and an $(n-k)$-basis $(\boldsymbol{\tau}_\eta)_\eta$ of S_1, as well as k random scalars $\alpha_\eta \overset{\$}{\leftarrow} \mathbb{Z}_p$, for $\eta = 1, \ldots, k$. $\mathcal{S}$ then computes the ciphertexts as follows, for $i = 1, \ldots, n$ and $j = 1, \ldots, m$,:
$$C_{j,i} = \prod_{\eta=1}^{k} H_j^{\alpha_\eta \sigma_{\eta,i}} \cdot \prod_{\eta=1}^{n-k} D_{\eta,j}^{\tau_{\eta,i}} \cdot g^{x^b_{j,i}},$$
where $H_j = \mathcal{H}(\lambda_j) \overset{\$}{\leftarrow} \mathbb{G}$, and $D_{\eta,j} \overset{\$}{\leftarrow} \mathbb{G}$, for $\eta = 1, \ldots, n-k$.
- Functional decryption key queries: for any query $\mathsf{QDKeyGen}(\boldsymbol{y})$, for a vector $\boldsymbol{y}$, $\mathcal{S}$ computes $\mathsf{dk}_{\boldsymbol{y}} = (\mathbb{G}, p, g, \mathcal{H}, \boldsymbol{y}, \sum_{\eta=1}^{k} \alpha_\eta \langle \boldsymbol{\sigma}_\eta, \boldsymbol{y} \rangle)$.
- Corruption: for a sender i, $\mathcal{S}$ sends $\mathsf{ek}_i = \sum_{\eta=1}^{k} \alpha_\eta \sigma_{\eta,i}$
- Finalize: $\mathcal{A}$ provides its guess b' on the bit b and $\mathcal{S}$ filters the cases were $\mathcal{A}$ requests unauthorized dk or corruptions queries by setting $\beta \overset{\$}{\leftarrow} \{0,1\}$. In the other cases, $\mathcal{S}$ sets $\beta \leftarrow b'$.

Finally, the only leakage about b from this last game is in the ciphertexts $\{C_{i,j}\}$:
$$C_{j,i} = \prod_{\eta=1}^{k} H_j^{\alpha_\eta \sigma_{\eta,i}} \cdot \prod_{\eta=1}^{n-k} D_{\eta,j}^{\tau_{\eta,i}} \cdot g^{x^b_{j,i}} = \prod_{\eta=1}^{k} H_j^{\alpha_\eta \sigma_{\eta,i}} \cdot \prod_{\eta=1}^{n-k} D_{\eta,j}^{\tau_{\eta,i}} \cdot g^{x^{1-b}_{j,i}} \cdot g^{x^b_{j,i} - x^{1-b}_{j,i}}.$$

But $\boldsymbol{x}_j^b - \boldsymbol{x}_j^{1-b} \in S_1$, and so can be written as $\sum_\eta \xi_\eta \boldsymbol{\tau}_\eta$:
$$g^{x^b_{j,i} - x^{1-b}_{j,i}} = g^{\sum_\eta \xi_\eta \tau_{\eta,i}} = \prod_\eta (g^{\xi_\eta})^{\tau_{\eta,i}}.$$

As a consequence,
$$C_{j,i} = \prod_{\eta=1}^{k} H_j^{\alpha_\eta \sigma_{\eta,i}} \cdot \prod_{\eta=1}^{n-k} (D_{\eta,j} \cdot g^{\xi_\eta})^{\tau_{\eta,i}} \cdot g^{x^{1-b}_{j,i}} = \prod_{\eta=1}^{k} H_j^{\alpha_\eta \sigma_{\eta,i}} \cdot \prod_{\eta=1}^{n-k} E_{\eta,j}^{\tau_{\eta,i}} \cdot g^{x^{1-b}_{j,i}}$$

where $E_{\eta,j} = D_{\eta,j} \cdot g^{\xi_\eta}$ for $\eta = 1, \ldots, n-k$ and $j = 1, \ldots, m$. When the $(D_{\eta,j})$ all follow independent uniform distributions in $\mathbb{G}$, the $(E_{\eta,j})$ all do so too. As a consequence, the ciphertexts from honest senders do not leak any information about b, and so the advantage of any adversary (even powerful) in this game is 0: $\mathsf{Adv}_{\mathbf{G}_5}(\mathcal{A}) = 0$.

The differences between the advantages sum up to $(n-k) \times \mathsf{Adv}^{\mathsf{ddh}}(T + 4mt_e)$, which can be upper-bounded by $n \times \mathsf{Adv}^{\mathsf{ddh}}(T + 4mt_e)$.