Decentralized Multi-Client Functional Encryption for Inner Product

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

[1] XLIM, University of Limoges, CNRS
[2] DIENS, École normale supérieure, CNRS,
PSL University, Paris, France
[3] INRIA, Paris, France
{jeremy.chotard,edufoursans,romain.gay,phan,david.pointcheval}@ens.fr

Abstract. We consider a situation where multiple parties, owning data that have to be frequently updated, agree to share weighted sums of these data with some aggregator, but where they do not wish to reveal their individual data, and do not trust each other. We combine techniques from Private Stream Aggregation (PSA) and Functional Encryption (FE), to introduce a primitive we call Decentralized Multi-Client Functional Encryption (DMCFE), for which we give a practical instantiation for Inner Product functionalities. This primitive allows various senders to *non-interactively* generate ciphertexts which support inner-product evaluation, with functional decryption keys that can also be generated *non-interactively*, in a distributed way, among the senders. Interactions are required during the setup phase only. We prove adaptive security of our constructions, while allowing corruptions of the clients, in the random oracle model.

Keywords. Decentralized, Multi-Client, Functional Encryption, Inner Product.

1 Introduction

Functional Encryption (FE) [9,15,18,28] is a new paradigm for encryption which extends the traditional “all-or-nothing” requirement of Public-Key Encryption 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 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. This is the most general form of encryption as it encompasses identity-based encryption, attribute-based encryption, broadcast encryption.

However, whereas the input can be large, like a high-dimensional vector, the basic definition of FE implies that the input data comes from only one party: all the coordinates of the vector are provided by one party, and all are encrypted at the same time. In many practical applications, the data are an aggregation of information that comes from different parties that may not trust each other.

A naive way to distribute the ciphertext generation would be to take an FE scheme and to have a trusted party handling the setup and the key generation phases, while the encryption procedure would be left to many clients to execute by Multi-Party Computation (MPC). This straw man construction has two obvious weaknesses:

1. Generating any ciphertext requires potentially heavy interactions, with everybody simultaneously on line, and the full ciphertext has to be generated at once, with all the components being known at the same time;
2. Some authority (the trusted third party) reserves the power to recover every client's private data.

Multi-Client Functional Encryption [16, 20] addresses the former issue of independent generation of the ciphertext, and we introduce Decentralized Multi-Client Functional Encryption to address the latter, without any central authority nor master secret key.

Multi-Client Functional Encryption. In Multi-Client Functional Encryption (MCFE), as defined in [16, 20], the single input x to the encryption procedure is broken down into an input vector $(x_1, \ldots, x_n)$ where the components are independent. An index i for each client and a (typically time-based) label ℓ are used for every encryption: $(c_1 = \mathsf{Encrypt}(1, x_1, \ell), \ldots, c_n = \mathsf{Encrypt}(n, x_n, \ell))$. Anyone owning a functional decryption key dk_f, for an n-ary function f and multiple ciphertexts *for the same label* ℓ, $c_1 = \mathsf{Encrypt}(1, x_1, \ell), \ldots, c_n = \mathsf{Encrypt}(n, x_n, \ell)$, can compute $f(x_1, \ldots, x_n)$ but nothing else about the individual x_i's. The combination of ciphertexts generated for different labels does not give a valid global ciphertext and the adversary learns nothing from it. MCFE is similar to the naive construction described above with MPC, except that ciphertext generation now simply takes one round, and each ciphertext c_i can also be generated independently for the others.

Decentralized Multi-Client Functional Encryption. Still, MCFE requires a trusted party to generate a master key msk and to distribute the encryption keys ek_i to the clients and the functional decryption keys dk_f to the decryptors. In our scenario, however, the clients do not want to rely on any authority. We would thus be interested in a decentralized version of MCFE, where no authority is involved, but the generation of functional decryption keys remains an efficient process under the control of the clients themselves. We introduce the notion of Decentralized Multi-Client Functional Encryption (DMCFE), in which the authority is removed and the clients work together to generate appropriate functional decryption keys. We stress that the authority is not simply *distributed* to a larger number of parties, but that the resulting protocol is indeed *decentralized*: each client has complete control over their individual data and the functional keys they authorize the generation of.

1.1 A Use Case

Consider a financial firm that wants to compute aggregates of several companies' private data (profits, number of sales) so that it can better understand the dynamics of a sector. The companies may be willing to help the financial firm understand the sector as whole, or may be offered compensation for their help, but they don't trust the financial firm or each other with their individual data. After setting up a DMCFE, each company encrypts its private data with a time-stamp label under its private key. Together, they can give the financial firm a decryption aggregation key that only reveals a sum on the companies' private data weighted by public information (employee count, market value) for a given time-stamp. New keys can retroactively decrypt aggregates on old data.

1.2 Related Work

In their more general form, FE and MCFE schemes have been introduced in [5,6,10,16–19,27,30] but unfortunately, they all rely on non standard cryptographic assumptions (indistinguishability obfuscation, single-input FE for circuits, or multilinear maps). It is more important in practice, and it is an interesting challenge, to build FE for restricted (but concrete) classes of functions, satisfying standard security definitions, under well-understood assumptions.

Inner-Product Functional Encryption. 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 [26], but in a weak security model, named *selective security*. Later on, Agrawal, Libert and Stehlé [4] considered *adaptive security* for IP-FE and proposed constructions whose security is based on DDH, LWE or Paillier's Decisional Composite Residuosity (DCR) [25] assumptions.

Private Stream Aggregation (PSA). This notion, also referred to as Privacy-Preserving Aggregation of Time-Series Data, is an older primitive introduced by Shi *et al.* [29]. It is quite similar to our target DMCFE scheme, however PSA does not consider the possibility of adaptively generating different keys for different inner-product evaluations, 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 DMCFE. Further research on PSA has focused on achieving new properties or better efficiency [8,11,13,21,23,24] but not on enabling new functionalities.

Multi-Input Functional Encryption. Goldwasser *et al.* [16] introduced the notion of Multi-Input Functional Encryption (MIFE) which breaks down a single input x into an input vector $(x_1, \ldots, x_n)$ where the components are independent (as does MCFE), but for which there is no notion of ciphertext index or label: 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 $c_1 = \mathsf{Encrypt}(x_1), \ldots, c_n = \mathsf{Encrypt}(x_n)$, can compute $f(x_1, \ldots, x_n)$ but nothing else about the individual x_i's. Numerous applications of MIFE have been given in detail in [16].

As with MCFE, general purpose MIFE schemes rely on indistinguishability obfuscation or multilinear maps, which we currently do not know how to instantiate under standard cryptographic assumptions. Extending IP-FE to the multi-input setting has proved technically challenging. [3] builds the first Multi-Input IP-FE, that is, each input slot encrypts a vector $\boldsymbol{x}_i \in \mathbb{Z}_p^m$ for some dimension m, each functional decryption key is associated with a vector $\boldsymbol{y}$, and decryption recovers $\langle \boldsymbol{x}, \boldsymbol{y} \rangle$ where $\boldsymbol{x} := (\boldsymbol{x}_i \| \cdots \| \boldsymbol{x}_n)$, $\boldsymbol{y} \in \mathbb{Z}_p^{n \cdot m}$, and n denotes the number of slots, which can be set up arbitrarily. They prove their construction secure under standard assumptions (SXDH, and in fact, k-Lin for any $k \geq 1$) in bilinear groups. Concurrently, [22] build a two-input (i.e. $n = 2$) FE using similar assumptions in bilinear groups. Very recently, [2, 12] gave a *function-hiding* multi-input FE for inner products, where the functional decryption keys do not reveal their underlying functions. [2] also gives a generic transformation from single to multi-input for IP-FE, which gives the first multi-input constructions whose security rely on DDH, LWE, or DCR.

In multi-input FE, every ciphertext for every slot can be combined with any other ciphertext for any other slot, and used with functional decryption keys to decrypt an exponential number of values, as soon as there are more than one ciphertext per slot. This "mix-and-match" feature is crucial for some of the applications of MIFE, such as building Indistinguishability Obfuscation [16]. However, it also means the information leaked about the underlying plaintext is enormous, and in many applications, the security guarantees simply become void, especially when many functional decryption keys are queried. In the case of inner product, as soon as m well-chosen functional decryption keys are queried (i.e. for linearly independent vectors), the plaintexts are completely revealed. In the multi-client setting however, since only ciphertexts with the same label (think of it as a time-stamp, for instance) can be combined for decryption, information leakage of the plaintext is much reduced.

The fact that clients have more control over how much information is leaked about their data, and that we remove the need for a central authority in the case of DMCFE, makes our schemes better suited for real-world use.

1.3 Multi-Client Functional Encryption

We remark that, as for MIFE, private-key MCFE is more relevant than its public-key counterpart (this is explained in [16], or [3] in the context of IP-FE).

Essentially, in a public-key MCFE, an encryption of unknown plaintext x_i (for some label ℓ) can be used together with encryptions of arbitrarily chosen values x'_j for each slot $j \in [n]$ (for the same label ℓ) and a functional decryption key for some function f, to obtain the value $f(x'_1, \cdots, x'_{i_1}, x_i, x'_{i+1}, \cdots, x'_n)$. Since the values x'_j for $j \neq i$ are arbitrarily chosen, this reveals typically too much information on x_i for practical uses. In the case of inner product, that means that, from $\mathsf{Enc}(i, x_i, \ell)$, $\mathsf{dk}_{\boldsymbol{y}}$, and the public key, one can efficiently extract the values $x_i y_i + \sum_{j \neq i} x'_j y_j$ for chosen x'_j, which exactly reveals the partial inner product $x_i y_i$ (see [3] for more details on the limitations of public-key IP-FE in the multi-input setting).

Security is defined with an indistinguishability game, where the adversary has to distinguish between encryptions of chosen plaintexts $(x_i^0)_{i\in[n]}$ and $(x_i^1)_{i\in[n]}$. The inherent leakage of information about the plaintext given by functional decryption keys dk_f is captured by a Finalize procedure in the security game, where the advantage is set to zero if the adversary performed a trivial attack, in the sense that correctness allows the adversary to distinguish encryptions of $(x_i^0)_{i\in[n]}$ from $(x_i^1)_{i\in[n]}$, simply because the underlying functions f of the decryption keys tell apart these plaintexts, i.e. $f(x_1^0, \cdots, x_n^0) \neq f(x_1^1, \cdots, x_n^1)$.

In the public-key setting, 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 $f(x_1^0, \cdot, \ldots, \cdot) = f(x_1^1, \cdot, \ldots, \cdot)$, $f(\cdot, x_2^0, \ldots, \cdot) = f(\cdot, x_2^1, \ldots, \cdot)$, $\ldots$, $f(\cdot, \cdot, \ldots, x_n^0) = f(\cdot, \cdot, \ldots, x_n^1)$. Again, this would essentially exclude any function. A private-key encryption solves this issue, and is still well-suited for practical applications.

In this paper, we will thus consider this private-key setting which naturally fits the MCFE (and DMCFE) model as each component in the plaintext is separately provided by a different 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. More precisely, we propose such an MCFE for Inner-Product functions in Section 4, that is secure even against adaptive corruptions of the senders.

1.4 Decentralized Multi-Client Functional Encryption

While it allows independent generation of the ciphertexts, MCFE (like MIFE) still 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 taken out of the equation.

We thus introduce the notion of Decentralized Multi-Client Functional Encryption (DMCFE), in which the setup phase and the generation of functional decryption keys are decentralized among the same clients as the ones that generate the ciphertexts. We are interested in minimizing interactions during those operations. While one can do it, in a generic way, using MPC, our target is *at least* a non-interactive generation of the functional decryption keys, that we achieve in Section 5, again for Inner-Product functions. The one-time setup phase might remain interactive, but this has to be done once only.

Scheme	MCFE	ABDP15 [1]
SetUp	Pick $(s_i)_{i\in[n]}$ at random	Pick $(s_i)_{i\in[n]}$ at random and set $v_i = g^{s_i}$
Encrypt	Each client i, on input (x_i, s_i, ℓ), return $c_i = g^{x_i} \cdot \mathcal{H}(\ell)^{s_i}$	On input $((x_i)_i, (v_i)_i)$, pick $r \xleftarrow{\$} \mathbb{Z}_p$, return $(c_0 = g^r, (c_i = g^{x_i} \cdot v_i^r)_i)$
DKeyGen	On input $((y_i)_i, (s_i)_i)$, return $\mathsf{dk}_{\boldsymbol{y}} = \sum_i y_i s_i$	On input $((y_i)_i, (s_i)_i)$, return $\mathsf{dk}_{\boldsymbol{y}} = \sum_i y_i s_i$
Decrypt	Discrete logarithm on $g^\gamma = \frac{\prod_i c_i^{y_i}}{\mathcal{H}(\ell)^{\mathsf{dk}_{\boldsymbol{y}}}}$	Discrete logarithm on $g^\gamma = \frac{\prod_i c_i^{y_i}}{c_0^{\mathsf{dk}_{\boldsymbol{y}}}}$

Fig. 1. Comparison of the Inner-Product FE scheme from Abdalla *et al.* [1] and a similar MCFE obtained by introducing a hash function $\mathcal{H}$.

1.5 Technical Overview

We briefly showcase the techniques that allow us to build efficient MCFE and DMCFE schemes. The schemes we introduce later enjoy adaptive security (aka full security), where encryption queries are made adaptively by the adversary against the security game, but for the sake of clarity, we will here give an informal description of a selectively-secure scheme from the DDH assumption, where queries are made beforehand. Namely, the standard security notion for FE is indistinguishability-based, where the adversary has access to a Left-or-Right oracle, that on input (m_0, m_1) either always encrypts m_0 or always encrypts m_1. While for the adaptive security, the adversary can query this oracle adaptively, in the *selective* setting, all queries are made at the beginning, before seeing the public parameters.

We first design a secret-key MCFE scheme building up from the public-key FE scheme introduced by Abdalla *et al.* [1] (itself a selectively-secure scheme) where we replace the global randomness with a hash function (modeled as a random oracle for the security analysis), in order to make the generation of the ciphertexts independent for each client. The comparison is illustrated in Figure 1. Note that for the final decryption to be possible, one needs the function evaluation γ to be small enough, within this discrete logarithm setting. This is one limitation, which is still reasonable for real-world applications that use concrete numbers, that are not of cryptographic size.

If we write $c_0 = g^r$ in the single input case and $c_0 = \mathcal{H}(\ell)$ in the Multi-Client case, we have $c_i = g^{x_i} {c_0}^{s_i}$ for $i \in [n]$ in both cases. In the public-key scheme from [1], s_i was private, and only $v_i = g^{s_i}$ was known to the encryptor. Since we are now dealing with private encryption, the encryptor can use s_i. Correctness then follows from

$$g^{\gamma} = \frac{\prod_i c_i^{y_i}}{{c_0}^{\mathsf{dk}_{\boldsymbol{y}}}} = \frac{\prod_i (g^{x_i}{c_0}^{s_i})^{y_i}}{{c_0}^{\mathsf{dk}_{\boldsymbol{y}}}} = \frac{g^{\sum_i x_i y_i}{c_0}^{\sum_i y_i s_i}}{{c_0}^{\mathsf{dk}_{\boldsymbol{y}}}} = \frac{g^{\sum_i x_i y_i}{c_0}^{\mathsf{dk}_{\boldsymbol{y}}}}{{c_0}^{\mathsf{dk}_{\boldsymbol{y}}}} = g^{\langle \boldsymbol{x}, \boldsymbol{y} \rangle}.$$

We further define this MCFE scheme and prove it selectively secure under the DDH assumption in Appendix B.

We can easily decentralize the above protocol using standard MPC techniques, but as we mentioned, our main goal is to minimize interactions during the DKeyGen protocol. This simple protocol can illustrate our main insight: we need to provide the aggregator with the decryption key $\langle \boldsymbol{s}, \boldsymbol{y} \rangle$. Since the s_i's are owned individually by the clients, we are interested in a protocol that would let them send shares from which the decryptor would recover an agreed upon Inner Product on their individual inputs. This sounds like a job for MCFE.

More precisely, sending $\mathsf{Encrypt}(s_i)$ under some other key t_i would not solve our problem, because we would still need to provide $\langle \boldsymbol{t}, \boldsymbol{y} \rangle$ to enable decryption, so we send $\mathsf{Encrypt}(y_i s_i)$ under t_i. Now we only need to compute one decryption key: the key for the inner product with vector $\mathbf{1} = (1, \ldots, 1)$, namely $\sum_i t_i$.

There is one final caveat. The result of the inner product evaluation requires a final discrete logarithm computation, and we are no longer operating on real-world data, but on random elements from $\mathbb{Z}_p$. Any attempt to recover the discrete logarithm is hopeless, and we are stuck with $g^{\langle \boldsymbol{s}, \boldsymbol{y} \rangle}$. We work around this issue by using pairings, which effectively enable us to decrypt using only $g^{\langle \boldsymbol{s}, \boldsymbol{y} \rangle}$. The standard SXDH assumption on pairing groups states that the DDH assumption holds in both groups, so introducing pairings doesn't compromise the security of our scheme. Our fully-secure DMCFE from pairings, that inherits from this approach, is described in Section 5.

1.6 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 present the first solutions for Inner-Product Functional Encryption in the Multi-Client and Decentralized Multi-Client settings:

1. **Efficiency**: the proposed schemes are highly practical as their efficiency is comparable to that of the DDH-based IP-FE scheme from [4]. A value x_i is encrypted as a unique group element C_i. The setup phase, key generation and decryption all take time linear in the number of participants, and encryption takes time linear in its input.
2. **Security under a standard assumption**: our schemes are all adaptively secure under either the classical DDH assumption or the standard SXDH assumption.
3. **Security against adaptive corruptions**: In addition, we successfully address corruptions of clients, even adaptive ones in the MCFE setting, exploring what Goldwasser *et al.* [16] highlighted as an "interesting direction".

4. **Non interactivity**: The DMCFE scheme we present in Section 5 has a key generation protocol that does not require interactions.

Refer to Figure 2 for a comparison of the different schemes mentioned here. We leave open the problems of considering LWE-based or Paillier-based constructions and of extending this work beyond inner-product functions.

Scheme	Multiple Inner Products	Non Interactive Setup	Non Interactive Encrypt	Non Interactive KeyGen	Decentralized
PSA [29]	✗	✓	✓	N/A	✗
Section 1: Straw man Distributed FE	✓	✓	✗	✓	✗
Section 4: MCFE	✓	✓	✓	✓	✗
Section 5: DMCFE	✓	✗	✓	✓	✓

Fig. 2. Comparison of different cryptographic solutions to the problem of linearly aggregating Private Multi-Client data.

2 Definitions and Security Models

This section is devoted to defining MCFE and DMCFE and the security models that are appropriate for those primitives, in the indistinguishability setting.

2.1 Multi-Client Functional Encryption

An MCFE scheme encrypts vectors of data from several senders and allows the controlled computation of functions on these heterogeneous data. We now define a private-key MCFE as in [16, 20]:

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 user 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 functional decryption key* dk_f*, a label ℓ, and an 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 $\mathsf{Decrypt}(\mathsf{dk}_f, \ell, \boldsymbol{C}_\ell = (C_{\ell,i})_i) = f(\boldsymbol{x} = (x_i)_i)$.

The security model is quite similar to the one defined for FE, but as noted in [16, 20], one has to consider corruptions, since the senders do not trust each other, and they can collude and give their secret keys to the adversary who will play on their behalf.

Definition 2 (IND-Security Game for MCFE). *Let us consider an 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 $(\mathsf{mpk}, \mathsf{msk}, (\mathsf{ek}_i)_i) \leftarrow \mathsf{SetUp}(\lambda)$ and chooses a random bit $b \xleftarrow{\$} \{0,1\}$. It provides mpk to the adversary $\mathcal{A}$;*
- *Encryption queries $\mathsf{QEncrypt}(i, x^0, x^1, \ell)$: $\mathcal{A}$ has unlimited and adaptive access to a Left-or-Right encryption oracle, and receives the ciphertext $C_{\ell,i}$ generated by $\mathsf{Encrypt}(\mathsf{ek}_i, x^b, \ell)$. We note that any further query for the same pair (ℓ, i) will later be ignored;*
- *Functional decryption key queries $\mathsf{QDKeyGen}(f)$: $\mathcal{A}$ has unlimited and adaptive access to the $\mathsf{DKeyGen}(\mathsf{msk}, f)$ algorithm for any input function f of its choice. It is given back the functional decryption key dk_f;*
- *Corruption 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;*
- *Finalize: $\mathcal{A}$ provides its guess b' on the bit b, and this procedure outputs the result β of the security game, according to the analysis given below.*

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 $\mathsf{QCorrupt}$ during the whole game), and $\mathcal{HS}$ the set of honest (non-corrupted) senders. We set the output to $\beta \leftarrow b'$, unless one of the three cases below is true, in which case we set $\beta \xleftarrow{\$} \{0,1\}$:

1. *some $\mathsf{QEncrypt}(i, x_i^0, x_i^1, \ell)$-query has been asked for an index $i \in \mathcal{CS}$ with $x_i^0 \neq x_i^1$;*
2. *for some label ℓ, an encryption-query $\mathsf{QEncrypt}(i, x_i^0, x_i^1, \ell)$ has been asked for some $i \in \mathcal{HS}$, but encryption-queries $\mathsf{QEncrypt}(j, x_j^0, x_j^1, \ell)$ have not all been asked for all $j \in \mathcal{HS}$;*
3. *for some label ℓ and for some function f asked to $\mathsf{QDKeyGen}$, there exists a pair of vectors $(\boldsymbol{x}^0 = (x_i^0)_i, \boldsymbol{x}^1 = (x_i^1)_i)$ such that $f(\boldsymbol{x}^0) \neq f(\boldsymbol{x}^1)$, when*
   - *$x_i^0 = x_i^1$, for all $i \in \mathcal{CS}$;*
   - *$\mathsf{QEncrypt}(i, x_i^0, x_i^1, \ell)$-queries have been asked for all $i \in \mathcal{HS}$.*

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 [7], 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: with the encryption key, the adversary could simply re-encrypt and compare in case of deterministic encryption;
2. intuitively, if some input is missing, no function evaluation can be done by the adversary, so we enforce the adversary to ask $\mathsf{QEncrypt}$-queries for all the non-corrupted keys (since the adversary can generate any ciphertext itself for the corrupted components) as soon as one label is used;
3. for any functional decryption key, all the possible evaluations should not trivially allow the adversary to distinguish the ciphertexts generated through $\mathsf{QEncrypt}$-queries (on honest components).

In all these 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 (as explained in the introduction, the labels are typically time-stamps, only used once).

Remark 3. While the third constraint aims at preventing the adversary from trivially winning by guessing the bit b from the evaluation of a functional decryption, the two first might look artificial, but they are required for our proof to go through with our constructions:

- with a probabilistic encryption scheme, one could hope to remove the first one, but up to now, we only have deterministic constructions, which is quite classical in the private-key setting (such as symmetric encryption);
- depending on the scheme, an encryption on an "inactive" component (a component that has no impact on the value of a function f, for instance the ith ciphertext in the case of $f_{\boldsymbol{y}} : \boldsymbol{x} \rightarrow \langle \boldsymbol{x}, \boldsymbol{y} \rangle$ when $y_i = 0$) might not be needed for a complete evaluation, as is the case in our schemes (see Section 4). Moreover, our keys are homomorphic: from $\mathsf{dk}_{f_{\boldsymbol{y}}}$ and $\mathsf{dk}_{f_{\boldsymbol{y}'}}$, one can easily obtain $\mathsf{dk}_{f_{\boldsymbol{y}+\boldsymbol{y}'}}$. Rather than defining the inactivity of components of functions in the span of those queried, we simply require that ciphertexts be obtained for every component for a given label (either through an explicit query to $\mathsf{QEncrypt}$ or thanks to the encryption key obtained from $\mathsf{QCorrupt}$), which is consistent with the use-case we outlined in Section 1.1. One could also enforce, by construction, all the queries to be asked and otherwise guarantee that no information is leaked about the plaintexts, which is not the case of our schemes.

Weaker Notions. 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 a master secret key msk to generate the functional decryption keys. We would like to avoid such a powerful authority, and make the scheme totally decentralized among the owners of the data (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 functional decryption key generators, for a functional decryptor $\mathcal{FD}$. 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 decryptors, but since they could collude and combine all the functional decryption keys, in the description below, and in the security model, we will consider only one functional decryptor $\mathcal{FD}$. As already noticed, we could simply use the definition of MCFE [16,20], where the setup and the functional decryption key algorithms are replaced by MPC protocols among the clients. But this could lead to a quite interactive process. We thus focus on efficient one-round key generation protocols $\mathsf{DKeyGen}$ that can be split in a first step $\mathsf{DKeyGenShare}$ that generates partial keys and the combining algorithm $\mathsf{DKeyComb}$ that combines partial keys into the functional decryption key.

Definition 4 (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 the setup protocol and four 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 user encryption key ek_i, a value x_i to encrypt, and a label ℓ, and outputs the ciphertext $C_{\ell,i}$;*
- $\mathsf{DKeyGenShare}(\mathsf{sk}_i, \ell_f)$*: Takes as input a user secret key sk_i and a label ℓ_f, and outputs the partial functional decryption key $\mathsf{dk}_{f,i}$ for a function $f : \mathcal{M}^n \to \mathcal{R}$ that is described in ℓ_f;*
- $\mathsf{DKeyComb}((\mathsf{dk}_{f,i})_i, \ell_f)$*: Takes as input the partial functional decryption keys and eventually outputs the functional decryption key dk_f;*
- $\mathsf{Decrypt}(\mathsf{dk}_f, \ell, \boldsymbol{C})$*: Takes as input a functional decryption key dk_f, a label ℓ, and an 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 all the secret and encryption keys, as well as the (partial) functional decryption keys. Similarly, the function f might be included in the (partial) 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 $f : \mathcal{M}^n \to \mathcal{R}$, 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 $\mathsf{dk}_f \leftarrow \mathsf{DKeyComb}((\mathsf{DKeyGenShare}(\mathsf{sk}_i, \ell_f))_i, \ell_f)$, then we have $\mathsf{Decrypt}(\mathsf{dk}_f, \ell, \boldsymbol{C}_\ell = (C_{\ell,i})_i) = f(\boldsymbol{x} = (x_i)_i)$.

The security model is quite similar to the one defined above for MCFE, except that for the DKeyGen protocol, the adversary has access to transcripts of the communications and can make some senders play maliciously. Corrupt-queries additionally reveal the secret keys sk_i.

Definition 5 (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 $(\mathsf{mpk}, (\mathsf{sk}_i)_i, (\mathsf{ek}_i)_i) \leftarrow \mathsf{SetUp}(\lambda)$ and chooses a random bit $b \xleftarrow{\$} \{0,1\}$. It provides mpk to the adversary $\mathcal{A}$;*
- *Encryption queries $\mathsf{QEncrypt}(i, x^0, x^1, \ell)$: $\mathcal{A}$ has unlimited and adaptive access to a Left-or-Right encryption oracle, and receives the ciphertext $C_{\ell,i}$ generated by $\mathsf{Encrypt}(\mathsf{ek}_i, x^b, \ell)$. We note that any further query for the same pair (ℓ, i) will later be ignored;*
- *Functional decryption key queries $\mathsf{QDKeyGen}(i, f)$: $\mathcal{A}$ has unlimited and adaptive access to the (non-corrupted) senders running the $\mathsf{DKeyGenShare}(\mathsf{sk}_i, f)$ algorithm for any input function f of its choice. It is given back the partial functional decryption key $\mathsf{dk}_{f,i}$;*
- *Corruptions queries $\mathsf{QCorrupt}(i)$: $\mathcal{A}$ 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, and this procedure outputs the result β of the security game, according to the analysis given below.*

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 $\mathsf{QCorrupt}$ during the whole game), and $\mathcal{HS}$ the set of honest (non-corrupted) senders. We set the output to $\beta \leftarrow b'$, unless one of the three cases below is true, in which case we set $\beta \xleftarrow{\$} \{0,1\}$:

1. *some $\mathsf{QEncrypt}(i, x_i^0, x_i^1, \ell)$-query has been asked for an index $i \in \mathcal{CS}$ with $x_i^0 \neq x_i^1$;*
2. *for some label ℓ, an encryption-query $\mathsf{QEncrypt}(i, x_i^0, x_i^1, \ell)$ has been asked for some $i \in \mathcal{HS}$, but encryption-queries $\mathsf{QEncrypt}(j, x_j^0, x_j^1, \ell)$ have not all been asked for all $j \in \mathcal{HS}$;*
3. *for some label ℓ and for some function f asked to $\mathsf{QDKeyGen}$ for all $i \in \mathcal{CS}$, there exists a pair of vectors $(\boldsymbol{x}^0 = (x_i^0)_i, \boldsymbol{x}^1 = (x_i^1)_i)$ such that $f(\boldsymbol{x}^0) \neq f(\boldsymbol{x}^1)$, when*
   - *$x_i^0 = x_i^1$, for all $i \in \mathcal{CS}$;*
   - *$\mathsf{QEncrypt}(i, x_i^0, x_i^1, \ell)$-queries have been asked for all $i \in \mathcal{HS}$.*

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

We define sel-IND (selective) and sta-IND (static) security for DMCFE as we did for MCFE.

3 Notations and Assumptions

3.1 Groups

Prime Order Group. We use a prime-order group generator GGen, a probabilistic polynomial time (PPT) algorithm that on input the security parameter 1^λ returns a description $\mathcal{G} = (\mathbb{G}, p, P)$ of an additive cyclic group $\mathbb{G}$ of order p for a 2λ-bit prime p, whose generator is P.

We use implicit representation of group elements as introduced in [14]. For $a \in \mathbb{Z}_p$, define $[a] = aP \in \mathbb{G}$ as the *implicit representation* of a in $\mathbb{G}$. More generally, for a matrix $\mathbf{A} = (a_{ij}) \in \mathbb{Z}_p^{n \times m}$ we define $[\mathbf{A}]$ as the implicit representation of $\mathbf{A}$ in $\mathbb{G}$:

$$[\mathbf{A}] := \begin{pmatrix} a_{11}P & ... & a_{1m}P \\ \\ a_{n1}P & ... & a_{nm}P \end{pmatrix} \in \mathbb{G}^{n \times m}$$

We will always use this implicit notation of elements in $\mathbb{G}$, i.e., we let $[a] \in \mathbb{G}$ be an element in $\mathbb{G}$. Note that from a random $[a] \in \mathbb{G}$ it is generally hard to compute the value a (discrete logarithm problem in $\mathbb{G}$). Obviously, given $[a], [b] \in \mathbb{G}$ and a scalar $x \in \mathbb{Z}_p$, one can efficiently compute $[ax] \in \mathbb{G}$ and $[a + b] = [a] + [b] \in \mathbb{G}$.

Pairing Group. We also use a pairing group generator PGGen, a PPT algorithm that on input 1^λ returns a description $\mathcal{PG} = (\mathbb{G}_1, \mathbb{G}_2, p, P_1, P_2, e)$ of asymmetric pairing groups where $\mathbb{G}_1$, $\mathbb{G}_2$, $\mathbb{G}_T$ are additive cyclic groups of order p for a 2λ-bit prime p, P_1 and P_2 are generators of $\mathbb{G}_1$ and $\mathbb{G}_2$, respectively, and $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is an efficiently computable (non-degenerate) bilinear map. Define $P_T := e(P_1, P_2)$, which is a generator of $\mathbb{G}_T$. We again use implicit representation of group elements. For $s \in \{1, 2, T\}$ and $a \in \mathbb{Z}_p$, define $[a]_s = aP_s \in \mathbb{G}_s$ as the implicit representation of a in G_s. Given $[a]_1$, $[a]_2$, one can efficiently compute $[ab]_T$ using the pairing e. For two matrices $\mathbf{A}$, $\mathbf{B}$ with matching dimensions define $e([\mathbf{A}]_1, [\mathbf{B}]_2) := [\mathbf{AB}]_T \in \mathbb{G}_T$.

Compatibility. Our construction from Section 4 uses a prime-order group, while the one from Section 5 uses pairing groups. Since the latter use the former as a building block, we must use groups that are compatible with each other. Notice that one can generate a prime-order group either with $\mathcal{G} := (\mathbb{G}, p, P) \xleftarrow{\$} \mathsf{GGen}(1^\lambda)$, but also using $\mathcal{PG} := (\mathbb{G}_1, \mathbb{G}_2, p, P_1, P_2, e) \xleftarrow{\$} \mathsf{PGGen}(1^\lambda)$, and setting $\mathbb{G} := \mathbb{G}_1$. This is possible here because we use asymmetric pairings and rely on the SXDH assumption in the pairing group, which is DDH in $\mathbb{G}_1$ and $\mathbb{G}_2$. More details on computational assumptions follow.

3.2 Computational Assumptions

Definition 6 (Decisional Diffie-Hellman Assumption). *The Decisional Diffie-Hellman Assumption states that, in a prime-order group $\mathcal{G} \xleftarrow{\$} \mathsf{GGen}(1^\lambda)$, no PPT adversary can distinguish between the two following distributions with non-negligible advantage:*

$$\{([a],[r],[ar]) \mid a, r \xleftarrow{\$} \mathbb{Z}_p\} \textit{ and } \{([a],[r],[s]) \mid a, r, s \xleftarrow{\$} \mathbb{Z}_p\}.$$

Equivalently, this assumption states it is hard to distinguish, knowing $[a]$, a random element from the span of $[\boldsymbol{a}]$ for $\boldsymbol{a} = \binom{1}{a}$, from a random element in $\mathbb{G}^2$: $[\boldsymbol{a}] \cdot r = [\boldsymbol{a}r] = \binom{[r]}{[ar]} \approx \binom{[r]}{[s]}$.

Definition 7 (Symmetric eXternal Diffie-Hellman Assumption). *The Symmetric eXternal Diffie-Hellman (SXDH) Assumption states that, in a pairing group $\mathcal{PG} \xleftarrow{\$} \mathsf{PGGen}(1^\lambda)$, the DDH assumption holds in both $\mathbb{G}_1$ and $\mathbb{G}_2$.*

4 A Fully-Secure MCFE for Inner Product

After the first construction drafted in the introduction, from the Abdalla *et al.* [1] selectively-secure FE, we propose another construction of MCFE for inner product adapted from the Agrawal *et al.* [4] scheme. We also provide the full security analysis under the DDH assumption, since the security proof of our DMCFE construction will rely on it.

Overview of the Construction. This construction is an extension of the previous one proposed in the introduction: we first extended the scheme from Abdalla *et al.* [1] in the multi-client setting with a hash function. Because of the selective security of the underlying scheme, our first proposal was just selectively secure too. We now adapt the Agrawal *et al.* [4] scheme, in the same manner. This construction and its proof of adaptive security are for the sake of clarity, since the proof of our next DMCFE will be made clearer when reducing to this one.

4.1 Description

We use a prime-order group, and the bracket notation, as defined in Section 3.1.

- $\mathsf{SetUp}(\lambda)$: Takes as input the security parameter, and generates prime-order group $\mathcal{G} := (\mathbb{G}, p, P) \xleftarrow{\$} \mathsf{GGen}(1^\lambda)$, and $\mathcal{H}$ a full-domain hash function onto $\mathbb{G}^2$. It also generates the encryption keys $\boldsymbol{s}_i \xleftarrow{\$} \mathbb{Z}_p^2$, for $i = 1, \ldots, n$. The public parameters mpk consist of $(\mathbb{G}, p, g, \mathcal{H})$, while the encryption keys are $\mathsf{ek}_i = \boldsymbol{s}_i$ for $i = 1, \ldots, n$, and the master secret key is $\mathsf{msk} = ((\mathsf{ek}_i)_i)$, (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 $\mathsf{ek}_i = \boldsymbol{s}_i$ and the label ℓ. It computes $[\boldsymbol{u}_\ell] := \mathcal{H}(\ell) \in \mathbb{G}^2$, and outputs the ciphertext $[c_i] = [\boldsymbol{u}_\ell^\top \boldsymbol{s}_i + x_i] \in \mathbb{G}$;

- $\mathsf{DKeyGen}(\mathsf{msk}, \boldsymbol{y})$: Takes as input $\mathsf{msk} = (\boldsymbol{s}_i)_i$ 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 $\mathsf{dk}_{\boldsymbol{y}} = (\boldsymbol{y}, \sum_i \boldsymbol{s}_i \cdot y_i) \in \mathbb{Z}_p^n \times \mathbb{Z}_p^2$;
- $\mathsf{Decrypt}(\mathsf{dk}_{\boldsymbol{y}}, \ell, ([c_i])_{i \in [n]})$: Takes as input a functional decryption key $\mathsf{dk}_{\boldsymbol{y}} = (\boldsymbol{y}, \boldsymbol{d})$, a label ℓ, and ciphertexts. It computes $[\boldsymbol{u}_\ell] := \mathcal{H}(\ell)$, $[\alpha] = \sum_i [c_i] \cdot y_i - [\boldsymbol{u}_\ell^\top] \cdot \boldsymbol{d}$, and eventually solves the discrete logarithm to extract and return α.

Note that, as for [4], the result α must be polynomially bounded to efficiently compute the discrete logarithm in the last decryption step: let $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{Z}_p^n$, we have:

$$\begin{aligned}[\alpha] &= \sum_i [c_i] \cdot y_i - [\boldsymbol{u}_\ell^\top] \cdot \boldsymbol{d} = \sum_i [\boldsymbol{u}_\ell^\top \boldsymbol{s}_i + x_i] \cdot y_i - [\boldsymbol{u}_\ell^\top] \cdot \sum_i y_i \boldsymbol{s}_i \\ &= \sum_i [\boldsymbol{u}_\ell^\top] \cdot \boldsymbol{s}_i y_i + \sum_i [x_i] \cdot y_i - [\boldsymbol{u}_\ell^\top] \cdot \sum_i y_i \boldsymbol{s}_i = [\sum_i x_i y_i].\end{aligned}$$

4.2 Security Analysis

Theorem 8 (IND-Security). *The above MCFE protocol (see Section 4.1) is IND-secure under the DDH assumption, in the random oracle model. More precisely, we have*

$$\mathsf{Adv}^{\mathit{IND}}(\mathcal{A}) \leq 2Q \cdot \mathsf{Adv}_{\mathbb{G}}^{\mathit{ddh}}(t) + \mathsf{Adv}_{\mathbb{G}}^{\mathit{ddh}}(t + 4Q \times t_{\mathbb{G}}) + \frac{2Q}{p},$$

for any adversary $\mathcal{A}$, running within time t, where Q is the number of (direct and indirect —asked by QEncrypt*-queries—) queries to $\mathcal{H}$ (modeled as a random oracle), and $t_{\mathbb{G}}$ is the time for an exponentiation in $\mathbb{G}$.*

We stress that this Theorem supports both adaptive encryption queries and adaptive corruptions.

Proof Technique. To obtain adaptive security, we use a technique that consists of first proving perfect security in the selective variant of the involved games, then, using a guessing (a.k.a. complexity leveraging) argument, which incurs an exponential security loss, we obtain the same security guarantees in the adaptive games. Since the security in the selective game is perfect (the advantage of any adversary is exactly zero), the exponential security loss is multiplied by a zero term, and the overall adaptive security is preserved. This technique has been used before in [31] in the context of Attribute-Based Encryption, or more recently, in [2,3] in the context of multi-input IP-FE. We defer to [31, Remark 1] and [3, Remark 5] for more details on this proof technique.

Proof. We proceed using hybrid games, described in Fig. 3. Let $\mathcal{A}$ be a PPT adversary. For any game G_{index}, we denote by $\mathsf{Adv}_{\mathsf{index}} := |\Pr[G_{\mathsf{index}}(\mathcal{A}) | b = 1] - \Pr[G_{\mathsf{index}}(\mathcal{A}) | b = 0]|$, where the probability is taken over the random coins of G_{index} and $\mathcal{A}$. Also, by event $G_{\mathsf{index}}(\mathcal{A})$, or just G_{index} when there is no ambiguity, we mean that the Finalize procedure in game G_{index} (defined as in Definition 2) returns $\beta = 1$ from the adversary's answer b' when interacting with $\mathcal{A}$.

```
Games $G_0$, $G_1$, $\boxed{G_2,\ (G_{3.q.1})_{q\in[Q+1]},\ (G_{3.q.2}, G_{3.q.3})_{q\in[Q]}}$

$\mathcal{G} \leftarrow \mathsf{GGen}(1^\lambda)$, for all $i \in [n]$, $\boldsymbol{s}_i \xleftarrow{\$} \mathbb{Z}_p^2$, $\mathsf{ek}_i := \boldsymbol{s}_i$, $\mathsf{msk} := (\boldsymbol{s}_i)_i$, $\mathsf{mpk} := (\mathbb{G}, p, g)$.
$\boxed{a \xleftarrow{\$} \mathbb{Z}_p,\ \boldsymbol{a} := \binom{1}{a},\ \boldsymbol{a}^\perp := \binom{-a}{1}}$
Sample a full-domain hash function $\mathcal{H}$ onto $\mathbb{G}^2$, and a bit $b \xleftarrow{\$} \{0,1\}$.
$b' \leftarrow \mathcal{A}^{\mathsf{QEncrypt}(\cdot,\cdot,\cdot,\cdot),\mathsf{QDKeyGen}(\cdot),\mathsf{QCorrupt}(\cdot),\mathsf{RO}(\cdot)}(\mathsf{mpk})$.
Run Finalize on $b'$.

$\mathsf{RO}(\ell)$:                                   // $G_0$, $\boxed{G_1}$, $\boxed{G_2,\ G_{3.q.1},\ \colorbox{lightgray}{$G_{3.q.2},\ G_{3.q.3}$}}$
$[\boldsymbol{u}_\ell] := \mathcal{H}(\ell)$, $\boxed{[\boldsymbol{u}_\ell] := \mathsf{RF}(\ell)}$, $\boxed{[\boldsymbol{u}_\ell] := [\boldsymbol{a} \cdot r_\ell], \text{ with } r_\ell := \mathsf{RF}'(\ell)}$
$\colorbox{lightgray}{On the $q$'th (fresh) query: $[\boldsymbol{u}_\ell] := \mathsf{RF}'(\ell) \cdot \boldsymbol{a} + \mathsf{RF}''(\ell) \cdot \boldsymbol{a}^\perp$}$
Return $[\boldsymbol{u}_\ell]$.

$\mathsf{QEncrypt}(i, x_i^0, x_i^1, \ell)$:                  // $G_0$, $G_1$, $G_2$, $\boxed{G_{3.q.1},\ G_{3.q.2},\ \colorbox{lightgray}{$G_{3.q.3}$}}$
$[\boldsymbol{u}_\ell] := \mathsf{RO}(\ell)$,
$[c_i] := [\boldsymbol{u}_\ell^\top] \cdot \boldsymbol{s}_i + [x_i^b]$
$\boxed{\text{If } [\boldsymbol{u}_\ell] \text{ is computed on the } j \text{ RO-query, for } j < q\text{: } [c_i] := [\boldsymbol{u}_\ell^\top] \cdot \boldsymbol{s}_i + [x_i^0]}$
$\colorbox{lightgray}{If $[\boldsymbol{u}_\ell]$ is computed on the $q$-th RO-query: $[c_i] := [\boldsymbol{u}_\ell^\top] \cdot \boldsymbol{s}_i + [x_i^0]$}$
Return $[c_i]$

$\mathsf{QDKeyGen}(\boldsymbol{y})$: Return $\sum_i y_i \boldsymbol{s}_i$.        //$G_0$, $G_1$, $G_2$, $G_{3.q.1}, G_{3.q.2}, G_{3.q.3}$

$\mathsf{QCorrupt}(i)$: Return $\boldsymbol{s}_i$.                // $G_0$, $G_1$, $G_2$, $G_{3.q.1}, G_{3.q.2}, G_{3.q.3}$
```

Fig. 3. Games for the proof of Theorem 8. Here, RF, RF', RF'' are random functions onto $\mathbb{G}^2$, $\mathbb{Z}_p$, and $\mathbb{Z}_p^*$, respectively, that are computed on the fly. In each procedure, the components inside a solid (dotted, gray) frame are only present in the games marked by a solid (dotted, gray) frame. The Finalize procedure is defined as in Definition 2.

Game G_0: This is the IND-security game as given in Definition 2. Note that the hash function $\mathcal{H}$ is modeled as a random oracle RO onto $\mathbb{G}^2$. This is essentially used to generate $[\boldsymbol{u}_\ell] = \mathcal{H}(\ell)$.

Game G_1: We simulate the answers to any new RO-query by a truly random pair in $\mathbb{G}^2$, on the fly. The simulation remains perfect, and so $\mathsf{Adv}_0 = \mathsf{Adv}_1$.

Game G_2: We simulate the answers to any new RO-query by a truly random pair in the span of $[\boldsymbol{a}]$ for $\boldsymbol{a} := \binom{1}{a}$, with $a \xleftarrow{\$} \mathbb{Z}_p$. This uses the Multi-DDH assumption, which tightly reduces to the DDH assumption using the random-self reducibility (see Lemma 10, in Appendix A): $\mathsf{Adv}_1 - \mathsf{Adv}_2 \leq \mathsf{Adv}_{\mathbb{G}}^{\mathsf{ddh}}(t + 4Q \times t_{\mathbb{G}})$, where Q is the number of RO-queries and $t_{\mathbb{G}}$ the time for an exponentiation.

Game G_3: We simulate any $\mathsf{QEncrypt}$ query as the encryption of x_i^0 instead of x_i^b and go back for the answers to any new RO query by a truly random pair in $\mathbb{G}^2$.

```
Games $(G^\star_{3.q.2}, G^\star_{3.q.3})_{q\in[Q]}$:
$\big(\mathsf{state}, (z_i \in \mathbb{Z}_p^2 \cup \{\perp\})_{i\in[n]}\big) \leftarrow \mathcal{A}(1^\lambda, 1^n)$
$\mathcal{G} \leftarrow \mathsf{GGen}(1^\lambda)$, for all $i \in [n]$, $\boldsymbol{s}_i \xleftarrow{\$} \mathbb{Z}_p^2$, $\mathsf{ek}_i := \boldsymbol{s}_i$, $\mathsf{msk} := (\boldsymbol{s}_i)_i$, $\mathsf{mpk} := (\mathbb{G}, p, g)$.
$a \xleftarrow{\$} \mathbb{Z}_p$, $\boldsymbol{a} := \binom{1}{a}$, $\boldsymbol{a}^\perp := \binom{-a}{1}$, $b \xleftarrow{\$} \{0,1\}$.
$b' \leftarrow \mathcal{A}^{\mathsf{QEncrypt}(\cdot,\cdot,\cdot,\cdot),\mathsf{QDKeyGen}(\cdot),\mathsf{QCorrupt}(\cdot),\mathsf{RO}(\cdot)}(\mathsf{mpk}, \mathsf{state})$.
Run Finalize on $b'$.

$\mathsf{RO}(\ell)$:                                              // $G^\star_{3.q.2}$, $G^\star_{3.q.3}$
$[\boldsymbol{u}_\ell] := [\boldsymbol{a} \cdot r_\ell]$, with $r_\ell := \mathsf{RF}'(\ell)$
On the $q$'th (fresh) query: $[\boldsymbol{u}_\ell] := [\mathsf{RF}'(\ell) \cdot \boldsymbol{a} + \mathsf{RF}''(\ell) \cdot \boldsymbol{a}^\perp]$
Return $[\boldsymbol{u}_\ell]$.

$\mathsf{QEncrypt}(i, x_i^0, x_i^1, \ell)$:                      // $\boxed{G^\star_{3.q.2}}$, $G^\star_{3.q.3}$
$[\boldsymbol{u}_\ell] := \mathsf{RO}(\ell)$,
$[c_i] := [\boldsymbol{u}_\ell^\top] \cdot \boldsymbol{s}_i + [x_i^b]$
If $[\boldsymbol{u}_\ell]$ is computed on the $j$-th RO-query with $j < q$: $[c_i] := [\boldsymbol{u}_\ell^\top] \cdot \boldsymbol{s}_i + [x_i^0]$.
If $[\boldsymbol{u}_\ell]$ is computed on the $q$-th RO-query, then:
• if $(x_i^0, x_i^1) \neq z_i$, the game ends and returns $\beta \xleftarrow{\$} \{0,1\}$.
• otherwise, $[c_i] := [\boldsymbol{u}_\ell^\top] \cdot \boldsymbol{s}_i \boxed{+[x_i^b]}$ $+[x_i^0]$ , $\mathcal{S} := \mathcal{S} \cup \{i\}$.
Return $[c_i]$.

$\mathsf{QDKeyGen}(\boldsymbol{y})$: Return $\sum_i y_i \boldsymbol{s}_i$.          //$G^\star_{3.q.2}, G^\star_{3.q.3}$
$\mathsf{QCorrupt}(i)$:                                          // $G^\star_{3.q.2}, G^\star_{3.q.3}$
If $z_i = (x_i^0, x_i^1)$ with $x_i^0 \neq x_i^1$, the game ends, and returns $\beta \xleftarrow{\$} \{0,1\}$.
Return $\boldsymbol{s}_i$.
```

Fig. 4. Games $G^\star_{3.q.2}$ and $G^\star_{3.q.3}$, with $q \in [Q]$, for the proof of Theorem 8. Here, RF, RF' are random functions onto $\mathbb{G}^2$, and $\mathbb{Z}_p$, respectively, that are computed on the fly. In each procedure, the components inside a solid (gray) frame are only present in the games marked by a solid (gray) frame.

While it is clear that in this last game the advantage of any adversary is exactly 0 since b does not appear anywhere, the gap between G_2 and G_3 will be proven using a hybrid technique on the RO-queries. We thus index the following games by q, where $q = 1, \ldots, Q$. Note that only distinct RO-queries are counted, since a second similar query is answered as the first one. We detail this proof because the technique is important.

$G_{3.1.1}$**:** This is exactly game G_2. Thus, $\mathsf{Adv}_2 = \mathsf{Adv}_{3.1.1}$.

$G_{3.q.1} \rightsquigarrow G_{3.q.2}$**:** We first change the distribution of the output of the q-th RO-query, from uniformly random in the span of $[\boldsymbol{a}]$ to uniformly random over $\mathbb{G}^2$, using the DDH assumption. Then, we use the basis $(\binom{1}{a}, \binom{-a}{1})$ of $\mathbb{Z}_p^2$, to write a uniformly random vector over $\mathbb{Z}_p^2$ as $u_1 \cdot \boldsymbol{a} + u_2 \cdot \boldsymbol{a}^\perp$, where $u_1, u_2 \xleftarrow{\$} \mathbb{Z}_p$. Finally, we switch to $u_1 \cdot \boldsymbol{a} + u_2 \cdot \boldsymbol{a}^\perp$ where $u_1 \xleftarrow{\$} \mathbb{Z}_p$, and $u_2 \xleftarrow{\$} \mathbb{Z}_p^*$, which only changes the adversary view by a statistical distance of

$1/p$: $\mathsf{Adv}_{3.q.1} - \mathsf{Adv}_{3.q.2} \leq \mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}}(t) + 1/p$. The last step with $u_2 \in \mathbb{Z}_p^*$ will be important to guarantee that $\boldsymbol{u}_\ell^\top \boldsymbol{a}^\perp \neq 0$.

$G_{3.q.2} \rightsquigarrow G_{3.q.3}$**:** We now change the generation of the ciphertext $[c_i] := [\boldsymbol{u}_\ell^\top] \cdot \boldsymbol{s}_i + [x_i^b]$ by $[c_i] := [\boldsymbol{u}_\ell^\top] \cdot \boldsymbol{s}_i + [x_i^0]$, where $[\boldsymbol{u}_\ell]$ corresponds to the q-th RO-query. We then prove this does not change the adversary's view.

Note that if the output of the q-th RO-query is not used by $\mathsf{QEncrypt}$-queries, then the games $G_{3.q.2}$ and $G_{3.q.3}$ are identical. But we can show this is true too when there are RO-queries that are really involved in $\mathsf{QEncrypt}$-queries, and show that $\mathsf{Adv}_{3.q.2} = \mathsf{Adv}_{3.q.3}$ in that case too, in two steps. In Step 1, we show that there exists a PPT adversary $\mathcal{B}^\star$ such that $\mathsf{Adv}_{3.q.t} = (p^2+1)^n \cdot \mathsf{Adv}^\star_{3.q.t}(\mathcal{B}^\star)$, for $t = 2, 3$, where the games $G^\star_{3.q.2}$ and $G^\star_{3.q.3}$ are selective variants of games $G_{3.q.2}$ and $G_{3.q.3}$ respectively (see Fig. 4), where $\mathsf{QCorrupt}$ queries are asked before the initialization phase. In Step 2, we show that for all PPT adversaries $\mathcal{B}^\star$, we have $\mathsf{Adv}^\star_{3.q.2}(\mathcal{B}^\star) = \mathsf{Adv}^\star_{3.q.3}(\mathcal{B}^\star)$. This will conclude the two steps.

Step 1. We build a PPT adversary $\mathcal{B}^\star$ playing against $G^\star_{3.q.t}$ for $t = 2, 3$, such that $\mathsf{Adv}_{3.q.t} = (p^2+1)^n \cdot \mathsf{Adv}^\star_{3.q.t}(\mathcal{B}^\star)$.

Adversary $\mathcal{B}^\star$ first guesses for all $i \in [n]$, $z_i \overset{\$}{\leftarrow} \mathbb{Z}_p^2 \cup \{\perp\}$, which it sends to its selective game $G^\star_{3.q.t}$. That is, each guess z_i is either a pair of values (x_i^0, x_i^1) queried to $\mathsf{QEncrypt}$, or $\perp$, which means no query to $\mathsf{QEncrypt}$. Then, it simulates $\mathcal{A}$'s view using its own oracles. When $\mathcal{B}^\star$ guesses successfully (call E that event), it simulates $\mathcal{A}$'s view exactly as in $G_{3.q.t}$. If the guess was not successful, then $\mathcal{B}^\star$ stops the simulation and outputs a random bit β. Since event E happens with probability $(p^2+1)^{-n}$ and is independent of the view of adversary $\mathcal{A}$: $\mathsf{Adv}^\star_{3.q.t}(\mathcal{B}^\star)$ is equal to

$$\left| \Pr[G^\star_{3.q.t} | b = 0, E] \cdot \Pr[E] + \frac{\Pr[\neg E]}{2} - \Pr[G^\star_{3.q.t} | b = 1, E] \cdot \Pr[E] - \frac{\Pr[\neg E]}{2} \right|$$
$$= \Pr[E] \cdot |\Pr[G^\star_{3.q.t} | b = 0, E] - \Pr[G^\star_{3.q.t} | b = 1, E]| = (p^2+1)^{-n} \cdot \mathsf{Adv}_{3.q.t}.$$

Step 2. We assume the values $(z_i)_{i \in [n]}$ sent by $\mathcal{B}^\star$ are consistent, that is, they don't make the game end and return a random bit, and Finalize on b' does not return a random bit independent of b' (call E' this event).

We show that games $G^\star_{3.q.2}$ and $G^\star_{3.q.3}$ are identically distributed, conditioned on E'. To prove it, we use the fact that the two following distributions are identical, for any choice of γ:

$$(\boldsymbol{s}_i)_{i \in [n], z_i = (x_i^0, x_i^1)} \text{ and } \left(\boldsymbol{s}_i + \boldsymbol{a}^\perp \cdot \gamma(x_i^b - x_i^0)\right)_{i \in [n], z_i = (x_i^0, x_i^1)},$$

where $\boldsymbol{a}^\perp := \binom{-a}{1} \in \mathbb{Z}_p^2$ and $\boldsymbol{s}_i \overset{\$}{\leftarrow} \mathbb{Z}_p^2$, for all $i = 1, \ldots, n$. This is true since the $\boldsymbol{s}_i$ are independent of the z_i (note that this is true because we are in a selective setting, while this would not necessarily be true with adaptive $\mathsf{QEncrypt}$-queries). Thus, we can re-write $\boldsymbol{s}_i$ into $\boldsymbol{s}_i + \boldsymbol{a}^\perp \cdot \gamma(x_i^b - x_i^0)$ without changing the distribution of the game.

We now take a look at where the extra terms $\boldsymbol{a}^\perp \cdot \gamma(x_i^b - x_i^0)$ actually appear in the adversary's view:

- They do not appear in the output of $\mathsf{QCorrupt}$, because we assume event E' holds, which implies that if $z_i \neq \bot$, then i is not queried to $\mathsf{QCorrupt}$ or $x_i^1 = x_i^0$.
- They might appear in $\mathsf{QDKeyGen}(\boldsymbol{y})$ as

$$\mathsf{dk}_{\boldsymbol{y}} = \sum_{i \in [n]} \boldsymbol{s}_i \cdot y_i + \boldsymbol{a}^{\perp} \cdot \gamma \textstyle\sum_{i : z_i = (x_i^0, x_i^1)} y_i (x_i^b - x_i^0) \, .$$

  But the gray term equals $\mathbf{0}$ by the constraints for E' in Definition 2: for all $i \in \mathcal{HS}$, $z_i \neq \bot$; if $i \in \mathcal{CS}$ and $z_i \neq \bot$, $x_i^1 = x_i^0$; and $f(\boldsymbol{x}^0) = f(\boldsymbol{x}^1)$, hence $\sum_{i : z_i = (x_i^0, x_i^1)} y_i (x_i^b - x_i^0) = 0$.
- Eventually, they appear in the output of the $\mathsf{QEncrypt}$-queries which use $[\boldsymbol{u}_\ell]$ computed on the q-th RO-query, since for all others, the vector $[\boldsymbol{u}_\ell]$ lies in the span of $[\boldsymbol{a}]$, and $\boldsymbol{a}^\top \boldsymbol{a}^\perp = 0$. We thus have $[c_i] := [\boldsymbol{u}_\ell^\top] \cdot \boldsymbol{s}_i + (x_i^b - x_i^0)\gamma[\boldsymbol{u}_\ell^\top]\boldsymbol{a}^\perp + [x_i^b]$. Since $\boldsymbol{u}_\ell^\top \boldsymbol{a}^\perp \neq 0$, we can choose $\gamma = -1/\boldsymbol{u}_\ell^\top \boldsymbol{a}^\perp \bmod p$, and then $[c_i] = [\boldsymbol{u}_\ell^\top] \cdot \boldsymbol{s}_i + [x_i^0]$, which is the encryption of x_i^0. We stress that γ is independent of the index i, and so this simultaneously converts all the encryptions of x_i^b into encryptions of x_i^0. Finally, reverting these statistically perfect changes, we obtain that $[c_i]$ is identically distributed to $[\boldsymbol{u}_\ell^\top] \cdot \boldsymbol{s}_i + [x_i^0]$, as in game $G^\star_{3.q.3}$.

Thus, when event E' happens, the games are identically distributed. When $\neg E$ happens, the games both return $\beta \xleftarrow{\$} \{0,1\}$: $\mathsf{Adv}^\star_{3.q.2}(\mathcal{B}^\star) = \mathsf{Adv}^\star_{3.q.3}(\mathcal{B}^\star)$. As a conclusion, we get $\mathsf{Adv}_{3.q.2} = \mathsf{Adv}_{3.q.3}$.

$G_{3.q.3} \rightsquigarrow G_{3.q+1.1}$**:** This transition is the reverse of $G_{3.q.1} \rightsquigarrow G_{3.q.2}$, namely, we use the DDH assumption to switch back the distribution of $[\boldsymbol{u}_\ell]$ computed on the q-th RO-query from uniformly random over $\mathbb{G}^2$ (conditioned on the fact that $\boldsymbol{u}_\ell^\top \boldsymbol{a}^\perp \neq 0$) to uniformly random in the span of $[\boldsymbol{a}]$: $\mathsf{Adv}_{3.q.3} - \mathsf{Adv}_{3.q+1.1} \leq \mathsf{Adv}_{\mathbb{G}}^{\mathsf{ddh}}(t) + 1/p$.

As a conclusion, since $G_{3.Q+1.1} = G_3$, we have $\mathsf{Adv}_2 - \mathsf{Adv}_3 \leq 2Q(\mathsf{Adv}_{\mathbb{G}}^{\mathsf{ddh}}(t) + 1/p)$. In addition, $\mathsf{Adv}_3 = 0$, which concludes the proof.

5 A Statically-Secure DMCFE for Inner Product

Overview of the Scheme. Our construction of MCFE for inner product uses functional decryption keys $\mathsf{dk}_{\boldsymbol{y}} = (\boldsymbol{y}, \langle \boldsymbol{s}, \boldsymbol{y} \rangle) = (\boldsymbol{y}, \boldsymbol{d})$, where $\boldsymbol{d} = \langle \boldsymbol{s}, \boldsymbol{y} \rangle = \sum_i s_i y_i = \langle \boldsymbol{t}, \mathbf{1} \rangle$, 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 from the introduction with our MCFE for inner product, that is self-enabling, to describe a DMCFE for inner product. However, this is not straightforward as 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 $\boldsymbol{d} = \langle \boldsymbol{s}, \boldsymbol{y} \rangle$, which comes up when we use our scheme to encrypt encryption keys.

Nevertheless, following this idea we can overcome the concern above with pairings: One can only recover $[\boldsymbol{d}]$, but using a pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$, one can use our MCFE in both $\mathbb{G}_1$ and $\mathbb{G}_2$. This allows us to compute the functional decryption in $\mathbb{G}_T$, to get $[\langle \boldsymbol{x}, \boldsymbol{y} \rangle]_T$, which is decryptable as $\langle \boldsymbol{x}, \boldsymbol{y} \rangle$ is small enough.

5.1 Construction

Let us describe the new construction, using an asymmetric pairing group, as in Section 3.1.

- $\mathsf{SetUp}(\lambda)$: Generates $\mathcal{PG} := (\mathbb{G}_1, \mathbb{G}_2, p, P_1, P_2, e) \overset{\$}{\leftarrow} \mathsf{PGGen}(1^\lambda)$. Samples two full-domain hash functions $\mathcal{H}_1$ and $\mathcal{H}_2$ onto $\mathbb{G}_1^2$ and $\mathbb{G}_2^2$ respectively. Each sender $\mathcal{S}_i$ generates $\boldsymbol{s}_i \overset{\$}{\leftarrow} \mathbb{Z}_p^2$ for all $i \in [n]$, and interactively generates $\mathbf{T}_i \overset{\$}{\leftarrow} \mathbb{Z}_p^{2\times 2}$ such that $\sum_{i\in[n]} \mathbf{T}_i = \mathbf{0}$. One then sets $\mathsf{mpk} \leftarrow (\mathcal{PG}, \mathcal{H}_1, \mathcal{H}_2)$, and for $i = 1, \ldots, n$, $\mathsf{ek}_i = \boldsymbol{s}_i$, $\mathsf{sk}_i = (\boldsymbol{s}_i, \mathbf{T}_i)$;
- $\mathsf{Encrypt}(\mathsf{ek}_i, x_i, \ell)$: Takes as input the value x_i to encrypt, under the key $\mathsf{ek}_i = \boldsymbol{s}_i$ and the label ℓ. It computes $[\boldsymbol{u}_\ell]_1 := \mathcal{H}_1(\ell) \in \mathbb{G}_1^2$, and outputs the ciphertext $[c_i]_1 = [\boldsymbol{u}_\ell^\top \boldsymbol{s}_i + x_i]_1 \in \mathbb{G}_1$;
- $\mathsf{DKeyGenShare}(\mathsf{sk}_i, \boldsymbol{y})$: on input $\boldsymbol{y} \in \mathbb{Z}_p^n$ that defines the function $f_{\boldsymbol{y}}(\boldsymbol{x}) = \langle \boldsymbol{x}, \boldsymbol{y} \rangle$, and the secret key $\mathsf{sk}_i = (\boldsymbol{s}_i, \mathbf{T}_i)$, it computes $[\boldsymbol{v}_{\boldsymbol{y}}]_2 := \mathcal{H}_2(\boldsymbol{y}) \in \mathbb{G}_2^2$, $[\boldsymbol{d}_i]_2 := [y_i \cdot \boldsymbol{s}_i + \mathbf{T}_i \boldsymbol{v}_{\boldsymbol{y}}]_2$, and returns the partial decryption key as $\mathsf{dk}_{\boldsymbol{y},i} := ([\boldsymbol{d}_i]_2)$.
- $\mathsf{DKeyComb}((\mathsf{dk}_{\boldsymbol{y},i})_{i\in[n]}, \boldsymbol{y})$: the partial decryption keys $(\mathsf{dk}_{\boldsymbol{y},i} = ([\boldsymbol{d}_i]_2))_{i\in[n]}$, lead to $\mathsf{dk}_{\boldsymbol{y}} := (\boldsymbol{y}, [\boldsymbol{d}]_2)$, where $[\boldsymbol{d}]_2 = \sum_{i\in[n]} [\boldsymbol{d}_i]_2$;
- $\mathsf{Decrypt}(\mathsf{dk}_{\boldsymbol{y}}, \ell, ([c_i]_1)_{i\in[n]})$: on input the decryption key $\mathsf{dk}_{\boldsymbol{y}} = [\boldsymbol{d}]_2$, the label ℓ, and ciphertexts $([c_i]_1)_{i\in[n]}$, it computes $[\alpha]_T := \sum_{i\in[n]} e([c_i]_1, [y_i]_2) - e([\boldsymbol{u}_\ell]_1^\top, [\boldsymbol{d}]_2)$, and eventually solve the discrete logarithm in basis $[1]_T$ to extract and return α.

Correctness: Let $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{Z}_p^n$, we have:

$$\begin{aligned}[\boldsymbol{d}]_2 &= \sum_{i\in[n]} [\boldsymbol{d}_i]_2 = \sum_{i\in[n]} [y_i \cdot \boldsymbol{s}_i + \mathbf{T}_i \boldsymbol{v}_{\boldsymbol{y}}]_2 \\ &= [\sum_{i\in[n]} y_i \cdot \boldsymbol{s}_i]_2 + [\boldsymbol{v}_{\boldsymbol{y}}]_2 \cdot \sum_{i\in[n]} \mathbf{T}_i = [\sum_{i\in[n]} y_i \cdot \boldsymbol{s}_i]_2.\end{aligned}$$

Thus:

$$\begin{aligned}[\alpha]_T &:= \sum_{i\in[n]} e([c_i]_1, [y_i]_2) - e([\boldsymbol{u}_\ell]_1^\top, [\boldsymbol{d}]_2) \\ &= \sum_i [(\boldsymbol{u}_\ell^\top \boldsymbol{s}_i + x_i) y_i]_T - [\sum_{i\in[n]} y_i \boldsymbol{u}_\ell^\top \boldsymbol{s}_i]_T = [\sum_i x_i y_i]_T.\end{aligned}$$

5.2 Security Analysis

Theorem 9 (`sta-IND`-Security). *The above DMCFE protocol (see Section 5.1) is `sta-IND` secure under the SXDH assumption, in the random oracle model. Namely, for any PTT adversary $\mathcal{A}$, there exist PPT adversaries $\mathcal{B}_1$ and $\mathcal{B}_2$ such that:*

$$\begin{aligned}\mathsf{Adv}^{\mathtt{IND}}(\mathcal{A}) \leq 2Q_1 \cdot \mathsf{Adv}^{\mathit{ddh}}_{\mathbb{G}_1}(t) + 2Q_2 \cdot \mathsf{Adv}^{\mathit{ddh}}_{\mathbb{G}_2}(t) + \frac{2Q_1 + 2Q_2}{p}\\ + \mathsf{Adv}^{\mathit{ddh}}_{\mathbb{G}_1}(t + 4Q_1 \times t_{\mathbb{G}_1}) + 2 \cdot \mathsf{Adv}^{\mathit{ddh}}_{\mathbb{G}_2}(t + 4Q_2 \times t_{\mathbb{G}_2}),\end{aligned}$$

where Q_1 and Q_2 are the number of (direct and indirect) queries to $\mathcal{H}_1$ and $\mathcal{H}_2$ respectively (modeled as random oracles). The former being asked by $\mathsf{QEncrypt}$*-queries and the latter being asked by* $\mathsf{QDKeyGen}$*-queries.*

We stress that this Theorem supports adaptive encryption queries, but static corruptions only.

Proof. We proceed using hybrid games, described in Fig. 5, with similar notations as in the previous proof.

Game G_0: This is the `sta-IND`-security game as given in Definition 5, but with the set $\mathcal{CS}$ of corrupted senders known from the beginning. Note that the hash functions $\mathcal{H}_1$ and $\mathcal{H}_2$ are modeled as random oracles. The former is used to generate $[\boldsymbol{u}_\ell]_1 := \mathcal{H}_1(\ell) \in \mathbb{G}_1^2$ and the latter $[\boldsymbol{v_y}]_2 := \mathcal{H}_2(\boldsymbol{y}) \in \mathbb{G}_2^2$.

Game G_1: We replace the hash function $\mathcal{H}_2$ by a random oracle RO_2 that generates random pairs from $\mathbb{G}_2^2$ on the fly. In addition, for any $\mathsf{QDKeyGen}$-query on a corrupted index $i \in \mathcal{CS}$, one generates the partial functional decryption key by itself, without explicitly querying $\mathsf{QDKeyGen}$. Hence, we can assume that $\mathcal{A}$ does not query $\mathsf{QCorrupt}$ and $\mathsf{QDKeyGen}$ on the same indices $i \in [n]$. The simulation remains perfect, and so $\mathsf{Adv}_0 = \mathsf{Adv}_1$.

Game G_2: Now, the outputs of RO_2 are uniformly random in the span of $[\boldsymbol{b}]_2$ for $\boldsymbol{b} := \binom{1}{a'}$, with $a' \xleftarrow{\$} \mathbb{Z}_p$. As in the previous proof, we have $\mathsf{Adv}_1 - \mathsf{Adv}_2 \leq \mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}_2}(t + 4Q_2 \times t_{\mathbb{G}_2})$, where Q_2 is the number of RO_2-queries and $t_{\mathbb{G}_2}$ the time for an exponentiation.

Game G_3: We replace all the partial key decryption answers by $\mathsf{dk}_{\boldsymbol{y},i} := [y_i \cdot \boldsymbol{s}_i + \boldsymbol{w}_i \cdot (\boldsymbol{b}^\perp)^\top \boldsymbol{v_y} + \mathbf{T}_i \boldsymbol{v_y}]_2$, for new $\boldsymbol{w}_i \xleftarrow{\$} \mathbb{Z}_p^2$, such that $\sum_i \boldsymbol{w}_i = \boldsymbol{0}$, for each $\boldsymbol{y}$. We show below that $\mathsf{Adv}_2 = \mathsf{Adv}_3$.

Game G_4: We switch back the distribution of all the vectors $[\boldsymbol{v_y}]_2$ output by RO_2, from uniformly random in the span of $[\boldsymbol{b}]_2$, to uniformly random over $\mathbb{G}_2^2$, thus back to $\mathcal{H}_2(\boldsymbol{y})$. This transition is reverse to the two first transitions of this proof: $\mathsf{Adv}_3 - \mathsf{Adv}_4 \leq \mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}_2}(t + 4Q_2 \times t_{\mathbb{G}_2})$.

In order to prove the gap between G_2 and G_3, we do a new hybrid proof:

Game $G_{3.1.1}$: This is exactly game G_2. Thus, $\mathsf{Adv}_2 = \mathsf{Adv}_{3.1.1}$.

```
Games $G_0$, $G_1$, $\boxed{G_2,\ (G_{3.q.1})_{q\in[Q_{\mathsf{dk}}+1]},\ \colorbox{lightgray}{$(G_{3.q.2}, G_{3.q.3})_{q\in[Q_{\mathsf{dk}}]}$}}$, $G_4$

$\mathcal{PG} \leftarrow \mathsf{PGGen}(1^\lambda)$, $\forall i \in [n]$: $\boldsymbol{s}_i \xleftarrow{\$} \mathbb{Z}_p^2$, $\mathbf{T}_i \xleftarrow{\$} \mathbb{Z}_p^{2\times 2}$, such that $\sum_{i\in[n]} \mathbf{T}_i = \mathbf{0}$
$\mathsf{ek}_i := \boldsymbol{s}_i$, $\mathsf{sk}_i := (\boldsymbol{s}_i, \mathbf{T}_i)$, $\mathsf{mpk} := (\mathbb{G}, p, g)$.
$\boxed{a' \xleftarrow{\$} \mathbb{Z}_p,\ \boldsymbol{b} := \binom{1}{a'}}$
Sample full-domain hash functions $\mathcal{H}_1$ onto $\mathbb{G}_1^2$ and $\mathcal{H}_2$ onto $\mathbb{G}_2^2$.
Sample a bit $b \xleftarrow{\$} \{0,1\}$.
$b' \leftarrow \mathcal{A}^{\mathsf{QEncrypt}(\cdot,\cdot,\cdot,\cdot),\mathsf{QDKeyGen}(\cdot,\cdot),\mathsf{QCorrupt}(\cdot),\mathsf{RO}_1(\cdot),\mathsf{RO}_2(\cdot)}(\mathsf{mpk})$.
Run Finalize on $b'$.

$\underline{\mathsf{RO}_1(\ell)}$:    // $G_0$, $G_1$, $G_2$, $G_{3.q.1}$, $G_{3.q.2}$, $G_{3.q.3}$
Return $\mathcal{H}_1(\ell)$.

$\underline{\mathsf{RO}_2(\boldsymbol{y})}$:    // $G_0$, $\dbox{$G_1$}$, $\boxed{G_2,\ G_{3.q.1},\ \colorbox{lightgray}{$G_{3.q.2}, G_{3.q.3}$}}$, $G_4$
$[\boldsymbol{v}_{\boldsymbol{y}}]_2 := \mathcal{H}_2(\boldsymbol{y})$, $\dbox{$[\boldsymbol{v}_{\boldsymbol{y}}]_2 := \mathsf{RF}(\boldsymbol{y})$}$, $\boxed{[\boldsymbol{v}_{\boldsymbol{y}}]_2 := [\boldsymbol{b} \cdot t_{\boldsymbol{y}}]_2, \text{ with } t_{\boldsymbol{y}} := \mathsf{RF}'(\boldsymbol{y})}$
$\colorbox{lightgray}{On the $q$-th $\mathsf{RO}_2$-query: $[\boldsymbol{v}_{\boldsymbol{y}}]_2 := \mathsf{RF}(\boldsymbol{y})$}$
Return $[\boldsymbol{v}_{\boldsymbol{y}}]_2$.

$\underline{\mathsf{QEncrypt}(i, x_i^0, x_i^1, \ell)}$:    // $G_0$, $G_1$, $G_2$, $G_{3.q.1}$, $G_{3.q.2}$, $G_{3.q.3}$, $G_4$
$[\boldsymbol{u}_\ell]_1 := \mathsf{RO}_1(\ell)$,
$[c_i]_1 := [\boldsymbol{u}_\ell^\top]_1 \cdot \boldsymbol{s}_i + [x_i^b]_1$
Return $[c_i]$

$\underline{\mathsf{QDKeyGen}(\boldsymbol{y}, i)}$:    //$G_0$, $G_1$, $G_2$, $\boxed{G_{3.q.1},\ G_{3.q.2},\ \colorbox{lightgray}{$G_{3.q.3}$}}$, $\dbox{$G_4$}$
Compute $[\boldsymbol{v}_{\boldsymbol{y}}]_2 := \mathsf{RO}_2(\boldsymbol{y})$, $\mathsf{dk}_{\boldsymbol{y},i} := [y_i \cdot \boldsymbol{s}_i + \mathbf{T}_i \boldsymbol{v}_{\boldsymbol{y}}]_2$, set $\mathcal{S} := \mathcal{S} \cup \{i\}$.
$\boxed{\begin{array}{l}\text{If } [\boldsymbol{v}_{\boldsymbol{y}}]_2 \text{ is computed on the } j\text{-th } \mathsf{RO}_2\text{-query, for } j < q\text{:}\\ \mathsf{dk}_{\boldsymbol{y},i} := [y_i \cdot \boldsymbol{s}_i + \mathsf{RF}_i(\boldsymbol{y}) + \mathbf{T}_i \boldsymbol{v}_{\boldsymbol{y}}]_2.\end{array}}$
$\colorbox{lightgray}{$\begin{array}{l}\text{If } [\boldsymbol{v}_{\boldsymbol{y}}]_2 \text{ is computed on the } q\text{-th } \mathsf{RO}_2\text{-query:}\\ \mathsf{dk}_{\boldsymbol{y},i} := [y_i \cdot \boldsymbol{s}_i + \mathsf{RF}_i(\boldsymbol{y}) + \mathbf{T}_i \boldsymbol{v}_{\boldsymbol{y}}]_2.\end{array}$}$
$\dbox{$\mathsf{dk}_{\boldsymbol{y},i} := [y_i \cdot \boldsymbol{s}_i + \mathsf{RF}_i(\boldsymbol{y}) + \mathbf{T}_i \boldsymbol{v}_{\boldsymbol{y}}]_2.$}$
Return $\mathsf{dk}_{\boldsymbol{y},i}$.

$\underline{\mathsf{QCorrupt}(i)}$: Return $(\boldsymbol{s}_i, \mathbf{T}_i)$.    // $G_0$, $G_1$, $G_2$, $G_{3.q.1}$, $G_{3.q.2}$, $G_{3.q.3}$, $G_4$
```

Fig. 5. Games for the proof of Theorem 9. Here, RF, RF' are random functions onto $\mathbb{G}_2^2$ and $\mathbb{Z}_p$, respectively, that are computed on the fly. The RF_i are random functions conditioned on the fact that $\sum_{i\in[n]} \mathsf{RF}_i$ is the zero function. In each procedure, the components inside a solid (dotted, gray) frame are only present in the games marked by a solid (dotted, gray) frame. The Finalize procedure is defined as in Definition 5.

$G_{3.q.1} \rightsquigarrow G_{3.q.2}$**:** As in the previous proof, we first change the distribution of the output of the q-th RO_2-query, from uniformly random in the span of $[\boldsymbol{b}]$ to uniformly random over $\mathbb{G}^2$, using the DDH assumption. Then, we use the basis $\big(\big(\begin{smallmatrix}1\\a'\end{smallmatrix}\big), \big(\begin{smallmatrix}-a'\\1\end{smallmatrix}\big)\big)$ of $\mathbb{Z}_p^2$, to write a uniformly random vector over $\mathbb{Z}_p^2$ as $v_1 \cdot \boldsymbol{b} + v_2 \cdot \boldsymbol{b}^{\perp}$, where $v_1, v_2 \xleftarrow{\$} \mathbb{Z}_p$. Finally, we switch to $v_1 \cdot \boldsymbol{b} + v_2 \cdot \boldsymbol{b}^{\perp}$ where $v_1 \xleftarrow{\$} \mathbb{Z}_p$, and $v_2 \xleftarrow{\$} \mathbb{Z}_p^*$, which only changes the adversary view by a statistical distance of $1/p$: $\mathsf{Adv}_{3.q.1} - \mathsf{Adv}_{3.q.2} \leq \mathsf{Adv}_{\mathbb{G}}^{\mathsf{ddh}}(t) + 1/p$. The last step with $v_2 \in \mathbb{Z}_p^*$ will be important to guarantee that $\boldsymbol{v}_{\boldsymbol{y}}^{\top} \boldsymbol{b}^{\perp} \neq 0$.

$G_{3.q.2} \rightsquigarrow G_{3.q.3}$**:** We now change the simulation of $\mathsf{dk}_{\boldsymbol{y},i}$ from $\mathsf{dk}_{\boldsymbol{y},i} = [y_i \cdot \boldsymbol{s}_i + \mathbf{T}_i \boldsymbol{v}_{\boldsymbol{y}}]_2$ to $\mathsf{dk}_{\boldsymbol{y},i} = [y_i \cdot \boldsymbol{s}_i + \mathsf{RF}_i(\boldsymbol{y}) + \mathbf{T}_i \boldsymbol{v}_{\boldsymbol{y}}]_2$, with some RF_i functions onto $\mathbb{Z}_p^2$ such that $\sum_i \mathsf{RF}_i(\boldsymbol{y}) = 0$ for any input $\boldsymbol{y}$. We prove $\mathsf{Adv}_{3.q.2} = \mathsf{Adv}_{3.q.3}$. To this aim, we use the fact that the two following distributions are identical, for any choice of $\boldsymbol{w}_i \xleftarrow{\$} \mathbb{Z}_p^2$, such that $\sum_i \boldsymbol{w}_i = \mathbf{0}$:

$$(\mathbf{T}_i)_{i \in \mathcal{HS}} \text{ and } (\mathbf{T}_i + \boldsymbol{w}_i (\boldsymbol{b}^{\perp})^{\top})_{i \in \mathcal{HS}},$$

where for all $i \in [n]$, $\mathbf{T}_i \xleftarrow{\$} \mathbb{Z}_p^{2 \times 2}$ such that $\sum_i \mathbf{T}_i = \mathbf{0}$, and $\boldsymbol{b}^{\perp} := \big(\begin{smallmatrix}-a'\\1\end{smallmatrix}\big)$. The extra terms $(\boldsymbol{w}_i (\boldsymbol{b}^{\perp})^{\top})_{i \in \mathcal{HS}}$ only appear in the output of the queries to $\mathsf{QDKeyGen}$ which use the vector $[\boldsymbol{v}_{\boldsymbol{y}}]_2$ computed on the q-th RO_2-query (if there are such queries), because for all other queries, $[\boldsymbol{v}_{\boldsymbol{y}}]_2$ lies in the span of $[\boldsymbol{b}]_2$, and $\boldsymbol{b}^{\top} \boldsymbol{b}^{\perp} = 0$. We thus have $\mathsf{dk}_{\boldsymbol{y},i} := [y_i \cdot \boldsymbol{s}_i + \boldsymbol{w}_i \cdot (\boldsymbol{b}^{\perp})^{\top} \boldsymbol{v}_{\boldsymbol{y}} + \mathbf{T}_i \boldsymbol{v}_{\boldsymbol{y}}]_2$. Since $\boldsymbol{v}_{\boldsymbol{y}}$ is such that $\boldsymbol{v}_{\boldsymbol{y}}^{\top} \boldsymbol{b}^{\perp} \neq 0$, $(\boldsymbol{b}^{\perp})^{\top} \boldsymbol{v}_{\boldsymbol{y}} \neq 0$. In that case, the vectors $\boldsymbol{w}_i \cdot (\boldsymbol{b}^{\perp})^{\top} \boldsymbol{v}_{\boldsymbol{y}}$ are uniformly random over $\mathbb{Z}_p^2$ such that $\sum_i \boldsymbol{w}_i \cdot (\boldsymbol{b}^{\perp})^{\top} \boldsymbol{v}_{\boldsymbol{y}} = \mathbf{0}$, which is as in $G_{3.q.3}$, by setting $\mathsf{RF}_i(\boldsymbol{y}) := \boldsymbol{w}_i \cdot (\boldsymbol{b}^{\perp})^{\top} \boldsymbol{v}_{\boldsymbol{y}}$.

$G_{3.q.3} \rightsquigarrow G_{3.q+1.1}$**:** This transition is the reverse of $G_{3.q.1} \rightsquigarrow G_{3.q.2}$, namely, we use the DDH assumption to switch back the distribution of $[\boldsymbol{v}_{\boldsymbol{y}}]_2$ to uniformly random in the span of $[\boldsymbol{b}]_2$: $\mathsf{Adv}_{3.q.3} - \mathsf{Adv}_{3.q+1.1} \leq \mathsf{Adv}_{\mathbb{G}_2}^{\mathsf{ddh}}(t) + 1/p$.

Then one can note that $G_{3.Q_2+1.1} = G_3$, but also that in Game G_4, all the $\mathsf{dk}_{\boldsymbol{y},i}$ output by $\mathsf{QDKeyGen}$ can be computed only knowing $\sum_{i \in [n]} \boldsymbol{s}_i \cdot y_i$, which is exactly the functional decryption key $\mathsf{dk}_{\boldsymbol{y}}$ from our MCFE in Section 4.1. This follows from the fact that the values $\mathsf{RF}_i(\boldsymbol{y})$ perfectly mask the vectors $\boldsymbol{s}_i \cdot y_i$, up to revealing $\sum_{i \in [n]} \boldsymbol{s}_i \cdot y_i$ (as the RF_i must sum up to the zero function). Thus, we can reduce to the `IND`-security of the MCFE from Section 4.1 (or even `sta-IND`-security) by designing an adversary $\mathcal{B}$ against the MCFE from Section 4.1: Adversary $\mathcal{B}$ first samples $\mathbf{T}_i \xleftarrow{\$} \mathbb{Z}_p^{2 \times 2}$ for all $i \in [n]$, such that $\sum_{i \in [n]} \mathbf{T}_i = \mathbf{0}$. It sends $\mathcal{CS}$ given by $\mathcal{A}$ (set of static corruptions), then it receives mpk from the MCFE security game, as well as the secret keys $\boldsymbol{s}_i$ for $i \in \mathcal{CS}$. It forwards mpk as well as $(\boldsymbol{s}_i, \mathbf{T}_i)$ for $i \in \mathcal{CS}$ to $\mathcal{A}$. Then

– $\mathcal{B}$ answers oracle calls to RO_1, RO_2 and $\mathsf{QEncrypt}$ from $\mathcal{A}$ using its own oracles.

– To answer $\mathsf{QDKeyGen}(i, \boldsymbol{y})$: if i is the last non-corrupted index for $\boldsymbol{y}$, $\mathcal{B}$ queries its own $\mathsf{QDKeyGen}$ oracle on $\boldsymbol{y}$, to get $\mathsf{dk}_{\boldsymbol{y}} := \sum_i \boldsymbol{s}_i \cdot y_i \in \mathbb{Z}_p^2$, computes $[\boldsymbol{v}_{\boldsymbol{y}}]_2 := \mathcal{H}_2(\boldsymbol{y})$, and returns $\mathsf{dk}_{\boldsymbol{y},i} := [\mathsf{dk}_{\boldsymbol{y}} + \mathsf{RF}_i(\boldsymbol{y}) + \mathbf{T}_i \boldsymbol{v}_{\boldsymbol{y}}]_2$ to $\mathcal{A}$. Otherwise, it computes $[\boldsymbol{v}_{\boldsymbol{y}}]_2 := \mathcal{H}_2(\boldsymbol{y})$, and returns $\mathsf{dk}_{\boldsymbol{y},i} := [\mathsf{RF}_i(\boldsymbol{y}) + \mathbf{T}_i \boldsymbol{v}_{\boldsymbol{y}}]_2$ to $\mathcal{A}$. The random functions RF_i are computed on the fly, such that their sum is the zero function.

We stress that this last simulation requires to know $\mathcal{CS}$ and $\mathcal{HS}$, hence static corruptions only. From this reduction, one gets

$$\mathsf{Adv}_4 \leq 2Q_1 \cdot \mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}_1}(t) + \mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}_1}(t + 4Q_1 \times t_{\mathbb{G}_1}) + \frac{2Q_1}{p},$$

where Q_1 denotes the number of calls to RO_1, $t_{\mathbb{G}_1}$ denotes the time to compute an exponentiation in $\mathbb{G}_1$. This concludes the proof.

6 Conclusion

Multi-Client 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:

- For inner-product, in the DDH-based setting with ElGamal-like encryption, we have a strong restriction on the plaintexts, since the inner-product has to be small, in order to allow complete decryption of the scalar evaluation. It is an interesting problem to consider whether the LWE-based and DCR-based schemes can address this issue.
- 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 fully secure (adaptive encryptions and adaptive corruptions), for a larger class of functions than that of inner-product functions. The security analyses of our concrete constructions heavily rely on the linear properties of inner-product functions, however, the global methodology of the constructions themselves is not restricted to the inner-product setting. Therefore, new constructions could follow it.

Acknowledgments

This work was supported in part by the European Community's Seventh Framework Programme (FP7/2007-2013 Grant Agreement no. 339563 – CryptoCloud), the European Community's Horizon 2020 Project FENTEC (Grant Agreement no. 780108), and Google PhD fellowship.

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., Catalano, D., Fiore, D., Gay, R., Ursu, B.: Multi-input functional encryption for inner products: Function-hiding realizations and constructions without pairings. In: Shacham, H., Boldyreva, A. (eds.) CRYPTO 2018, Part I. LNCS, vol. 10991, pp. 597–627. Springer (2018)
3. Abdalla, M., Gay, R., Raykova, M., Wee, H.: Multi-input inner-product functional encryption from pairings. In: Coron, J., Nielsen, J.B. (eds.) EUROCRYPT 2017, Part I. LNCS, vol. 10210, pp. 601–626. Springer, Heidelberg (Apr / May 2017)
4. 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)
5. 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)
6. 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)
7. 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)
8. 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)
9. 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)
10. 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)
11. 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)
12. Datta, P., Okamoto, T., Tomida, J.: Full-hiding (unbounded) multi-input inner product functional encryption from the k-linear assumption. In: Abdalla, M., Dahab, R. (eds.) PKC 2018, Part II. LNCS, vol. 10770, pp. 245–277. Springer, Heidelberg (Mar 2018)
13. Emura, K.: Privacy-preserving aggregation of time-series data with public verifiability from simple assumptions. In: Australasian Conference on Information Security and Privacy. pp. 193–213. Springer (2017)
14. Escala, A., Herold, G., Kiltz, E., Ràfols, C., Villar, J.: An algebraic framework for Diffie-Hellman assumptions. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part II. LNCS, vol. 8043, pp. 129–147. Springer, Heidelberg (Aug 2013)
15. 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)

16. 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)
17. 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)
18. 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)
19. 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)
20. 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
21. Joye, M., Libert, B.: A scalable scheme for privacy-preserving aggregation of time-series data. In: Sadeghi, A.R. (ed.) FC 2013. LNCS, vol. 7859, pp. 111–125. Springer, Heidelberg (Apr 2013)
22. Lee, K., Lee, D.H.: Two-input functional encryption for inner products from bilinear maps. IACR Cryptology ePrint Archive 2016, 432 (2016), http://eprint.iacr.org/2016/432
23. Li, Q., Cao, G.: Efficient and privacy-preserving data aggregation in mobile sensing. In: ICNP 2012. pp. 1–10. IEEE Computer Society (2012)
24. 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)
25. 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)
26. 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)
27. 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)
28. 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)
29. 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)
30. 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)
31. Wee, H.: Dual system encryption via predicate encodings. In: Lindell, Y. (ed.) TCC 2014. LNCS, vol. 8349, pp. 616–637. Springer, Heidelberg (Feb 2014)

A Multi DDH Assumption

Theorem 10. *For any distinguisher $\mathcal{A}$ running within time t, the best advantage $\mathcal{A}$ can get in distinguishing*

$$\mathcal{D}_m = \{(X, (Y_j, Z_j = \mathit{CDH}(X, Y_j))_j) \mid X, Y_j \xleftarrow{\$} \mathbb{G}, j = 1, \ldots, m\}$$
$$\mathcal{D}'_m = \{(X, (Y_j, Z_j)_j) \mid X, Y_j, Z_j \xleftarrow{\$} \mathbb{G}, j = 1, \ldots, m\}.$$

is bounded by $\mathsf{Adv}^{\mathsf{ddh}}(t + 4m \times t_{\mathbb{G}})$, where $t_{\mathbb{G}}$ is the time for an exponentiation in $\mathbb{G}$.

Proof. One can first note that the best advantage one can get, within time t, between

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

is bounded by $\mathsf{Adv}^{\mathsf{ddh}}(t)$. This is actually the DDH assumption. One can note that $\mathcal{D}_m$ and $\mathcal{D}'_m$ can be rewritten as

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

Since, from (X, Y, Z), the m tuples require 4 additional exponentiations per index j, one get the expected bound.

B A Selectively-Secure MCFE

B.1 Description

In this section, we formally present the selectively secure MCFE scheme for inner product we described in Section 1. 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 $s_i \xleftarrow{\$} \mathbb{Z}_p$, 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. It computes $[u_\ell] := \mathcal{H}(\ell) \in \mathbb{G}$, and outputs the ciphertext $[c_i] = [u_\ell s_i + x_i] \in \mathbb{G}$;
- $\mathsf{DKeyGen}(\mathsf{msk}, \boldsymbol{y})$: Takes as input $\mathsf{msk} = (s_i)_i$ 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 $\mathsf{dk}_{\boldsymbol{y}} = (\boldsymbol{y}, \sum_i s_i y_i) \in \mathbb{Z}_p^n \times \mathbb{Z}_p$;
- $\mathsf{Decrypt}(\mathsf{dk}_{\boldsymbol{y}}, \ell, ([c_i])_{i \in [n]})$: Takes as input a decryption key $\mathsf{dk}_{\boldsymbol{y}} = (\boldsymbol{y}, d)$, a label ℓ. It computes $[u_\ell] := \mathcal{H}(\ell)$, $[\alpha] = \sum_i y_i \cdot [c_i] - d \cdot [u_\ell]$, and eventually solves the discrete logarithm to extract and return α.

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

$$\begin{aligned}[\alpha] &= \sum_i y_i \cdot [c_i] - d \cdot [u_\ell] = \sum_i y_i \cdot [u_\ell s_i + x_i] - [u_\ell] \cdot \sum_i s_i y_i \\ &= [u_\ell] \cdot \sum_i s_i y_i + [\sum_i x_i y_i] - [u_\ell] \cdot \sum_i s_i y_i = [\sum_i x_i y_i].\end{aligned}$$

B.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 of the pairs of messages for the Left-or-Right encryption oracle ($\mathsf{QEncrypt}$-queries). However, it can adaptively ask for functional decryption keys ($\mathsf{QDKeyGen}$-queries) and encryption keys ($\mathsf{QCorrupt}$-queries). Concretely, the challenger is provided (plaintext,label) pairs: $(x^b_{j,i}, \ell_j)_{b\in\{0,1\},i\in[n],j\in[Q]}$, where Q is the number of query to $\mathsf{QEncrypt}(i, \cdot, \cdot)$, each one for a different label ℓ_j (note that in the security model, we assume each slots are queried the same number of time, on different labels). The challenge ciphertexts $C_{i,j} = \mathsf{Encrypt}(\mathsf{ek}_i, x^b_{j,i}, \ell_j)$, for the random bit b, are returned to the adversary.

Note that the adversary committing to challenge ciphertexts also limits its ability to corrupt users during the game: it must corrupt clients for which it didn't ask a ciphertext and cannot corrupt any client from which it asked a ciphertext for $x^0_{j,i} \neq x^1_{j,i}$.

B.3 Security Analysis

Theorem 11 (sel-IND Security). *The MCFE protocol described above (see Appendix B.1) is sel-IND secure under the DDH assumption, in the random oracle model. More precisely, we have*

$$\mathsf{Adv}^{\mathtt{IND}}(\mathcal{A}) \leq 2Q \cdot \mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}}(t),$$

for any adversary $\mathcal{A}$, running within time t, where Q is the number of encryption queries per slot.

Proof. We proceed using hybrid games, described in Fig. 6, with the same notations as in the previous proofs.

Game G_0: This is the sel-IND security game as given in Definition 2 (see the paragraph about weaker models), with all the encryption queries being sent first: they are stored in $z_{j,i} = (x^0_{j,i}, x^1_{j,i})$, for $j \in [Q]$ and $i \in [n]$, where j is for the j-th $\mathcal{H}$-query that specifies the label ℓ_j and i is for the index of the sender. If the query is not asked, we have $z_{j,i} = \perp$. Note that the hash function $\mathcal{H}$ is modeled as a random oracle RO onto $\mathbb{G}$. This is used to generate $[u_\ell] = \mathcal{H}(\ell)$.

```
Games $G_0$, $G_1$, $(G_{2.q})_{q\in[Q+1]}$
$(\mathsf{state}, (\ell_j, z_{j,i})_{i\in[n],j\in[Q]}) \leftarrow \mathcal{A}(1^\lambda, 1^n)$
where each $z_{j,i} = (x^0_{j,i}, x^1_{j,i}) \in \mathbb{Z}_p^2$, or $z_{j,i} = \bot$, which stands for no query.
$\mathcal{G} \leftarrow \mathsf{GGen}(1^\lambda)$, for all $i \in [n]$, $s_i \xleftarrow{\$} \mathbb{Z}_p$,$\mathsf{ek}_i := s_i$, $\mathsf{msk} := (s_i)_i$, $\mathsf{mpk} := (\mathbb{G}, p, g)$.
$C_{j,i} = \mathsf{QEncrypt}(i, x^0_{j,i}, x^1_{j,i}, \ell_j)$ for $i \in [n], j \in [Q]$ such that $z_{j,i} = (x^0_{j,i}, x^1_{j,i})$.
$b' \leftarrow \mathcal{A}^{\mathsf{QDKeyGen}(\cdot),\mathsf{QCorrupt}(\cdot),\mathsf{RO}(\cdot)}(\mathsf{mpk}, \mathsf{state})$.
Run Finalize on $b'$.

$\mathsf{RO}(\ell)$:                                        // $G_0$, $\boxed{G_1, G_{2.q}}$
$[u_\ell] := \mathcal{H}(\ell)$ $\boxed{, [u_\ell] := \mathsf{RF}(\ell)}$.
Return $[u_\ell]$.

$\mathsf{QEncrypt}(i, x^0_i, x^1_i, \ell)$:                         // $G_0$, $G_1$, $\boxed{G_{2.q}}$
$[u_\ell] := \mathsf{RO}(\ell)$,
$[c_i] := [u_\ell] \cdot s_i + [x^b_i]$
$\boxed{\text{If } \ell = \ell_j \text{ with } j < q\text{: } [c_i] := [u_\ell s_i + x^0_i]}$
Return $[c_i]$.

$\mathsf{QDKeyGen}(\boldsymbol{y})$:                                  // $G_0$, $G_1$, $G_{2.q}$
Return $\sum_i y_i s_i$.

$\mathsf{QCorrupt}(i)$:                                     // $G_0$, $G_1$, $G_{2.q}$
Return $s_i$.
```

Fig. 6. Games G_0, G_1, $(G_{2.})_{q\in[Q+1]}$, for the proof of Theorem 11. Here, RF is a random function onto $\mathbb{G}$, that is computed on the fly. Note that $\mathsf{QEncrypt}$ is only used as a subroutine of the initialization of the game and is not accessible to the adversary. In each procedure, the components inside a solid frame are only present in the games marked by a solid frame.

Game G_1: We simulate the answers to any new RO query by computing a truly random element of $\mathbb{G}$, on the fly. The simulation remains perfect, so $\mathsf{Adv}_0 = \mathsf{Adv}_1$.

Game G_2: We simulate every encryption as the encryption of x^0_i instead of x^b_i.

While it is clear that in this last game the advantage of any adversary is exactly 0 since b does not appear anywhere, the gap between G_1 and G_2 will be proven using an hybrid argument on the RO-queries. We thus index the following games by q, where $q = 1, \ldots, Q$. Note that only distinct RO-queries are counted, since a second similar query is answered as the first one.

$G_{2.1}$: This is exactly game G_1. Thus, $\mathsf{Adv}_1 = \mathsf{Adv}_{2.1}$.

$G_{2.q} \rightsquigarrow G_{2.q+1}$: We change the generation of the ciphertexts from $[c_{q,i}] := [u_{\ell_q} s_i + x^b_{q,i}]$ to $[c_{q,i}] := [u_{\ell_q} s_i + x^0_{q,i}]$. We proceed in three steps:

Step 1. We use the fact that the two following distributions are identical, for any choice of γ:

$$(s_i)_{i\in[n],z_{q,i}=(x^0_{q,i},x^b_{q,i})} \text{ and } \big(s_i+\gamma(x^0_{q,i}-x^b_{q,i})\big)_{i\in[n],z_{q,i}=(x^0_{q,i},x^1_{q,i})},$$

where $s_i \xleftarrow{\$} \mathbb{Z}_p$, for all $i \in [n]$. This is true since the s_i are independent of the $z_{q,i}$ (we are in a selective setting, so the s_i's are generated after the $z_{q,i}$'s have been chosen). Thus, we can re-write s_i into $s_i + \gamma(x^0_{q,i} - x^b_{q,i})$ without changing the distribution of the game.
Note that when Finalize does not output a random bit $\beta \xleftarrow{\$} \{0,1\}$ independent of the guess b', γ does not appear in the outputs of $\mathsf{QCorrupt}(i)$, since it must be that $x^0_i = x^1_i$ or $z_{q,i} = \perp$, and it does not appear in the output of $\mathsf{QDKeyGen}(\boldsymbol{y})$ either, since $\sum_i s_i \cdot y_i + \sum_i \gamma(x^0_{q,i} - x^b_{q,i})y_i$, where the gray term equals zero by Definition 1. The fact that γ does not appear in the outputs of these oracles will be crucial for step 2, which applies DDH on $[\gamma]$.

Step 2. We use the DDH assumption to replace the $[u_{\ell_q}\gamma]$ that appear in the output of the q-th query to $\mathsf{QEncrypt}$ queries with $[r_{\ell_q} + 1]$ with $r_{\ell_q} \xleftarrow{\$} \mathbb{Z}_p$. This is possible since the rest of the adversary view can be generated only from $[\gamma]$ and $[r_{\ell_q} + 1]$. This increases the adversary's advantage by no more than $\mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}}(t)$. We now have:

$$\begin{aligned}[c_{q,i}] :=&[u_{\ell_q}s_i + (x^0_{q,i} - x^b_{q,i})(r_{\ell_q} + 1) + x^b_{q,i}]\\ =&[u_{\ell_q}s_i + r_{\ell_q}(x^0_{q,i} - x^b_{q,i}) + x^0_{q,i} - x^b_{q,i} + x^b_{q,i}]\\ =&[u_{\ell_q}s_i + r_{\ell_q}(x^0_{q,i} - x^b_{q,i}) + x^0_{q,i}].\end{aligned}$$

Step 3. We switch $[r_{\ell_q}]$ in the output of the q-query to $\mathsf{QEncrypt}$ back to $[u_{\ell_q}\gamma]$, using the DDH assumption again. This is possible since the adversary's view is simulatable solely from $[\gamma]$, $[u_{\ell_q}]$, and $[r_{\ell_q}]$. We finally undo the distribution change on the $\boldsymbol{s}_i$, which brings us to $G_{2.q+1}$.

As a conclusion, since $G_{2.Q+1} = G_2$, we have $\mathsf{Adv}_1 - \mathsf{Adv}_2 \leq 2Q \cdot \mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}}(t)$. In addition, $\mathsf{Adv}_2 = 0$, which concludes the proof.