Two-Client Inner-Product Functional Encryption, with an Application to Money-Laundering Detection

Paola de Perthuis[1,2,3] and David Pointcheval[2,3]

[1] Cosmian
[2] DIENS, École normale supérieure, CNRS, PSL University, Paris, France
[3] INRIA, Paris, France

Abstract. In this paper, we extend Inner-Product Functional Encryption (IPFE), where there is just a vector in the key and a vector in the single sender's ciphertext, to two-client ciphertexts. More precisely, in our two-client functional encryption scheme, there are two Data Providers who can independently encrypt vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ for a data consumer who can, from a functional decryption key associated to a vector $\boldsymbol{\alpha}$, compute $\sum \alpha_i x_i y_i = \boldsymbol{x} \cdot \mathsf{Diag}(\boldsymbol{\alpha}) \cdot \boldsymbol{y}^\top$. Ciphertexts are linear in the dimension of the vectors, whereas the functional decryption keys are of constant size. We study two interesting particular cases:

- 2-party Inner-Product Functional Encryption, with $\boldsymbol{\alpha} = (1, \ldots, 1)$. There is a unique functional decryption key, which enables the computation of $\boldsymbol{x} \cdot \boldsymbol{y}^\top$ by a third party, where $\boldsymbol{x}$ and $\boldsymbol{y}$ are provided by two independent clients;
- Inner-Product Functional Encryption with a Selector, with $\boldsymbol{x} = \boldsymbol{x}_0 \| \boldsymbol{x}_1$ and $\boldsymbol{y} = \bar{b}^n \| b^n \in \{1^n \| 0^n, 0^n \| 1^n\}$, for some bit b, on the public coefficients $\boldsymbol{\alpha} = \boldsymbol{\alpha}_0 \| \boldsymbol{\alpha}_1$, in the functional decryption key, so that one gets $\boldsymbol{x}_b \cdot \boldsymbol{\alpha}_b^\top$, where $\boldsymbol{x}$ and b are provided by two independent clients.

This result is based on the fundamental Product-Preserving Lemma, which is of independent interest. It exploits Dual Pairing Vector Spaces (DPVS), with security proofs under the SXDH assumption. We provide two practical applications to medical diagnosis for the latter IPFE with Selector, and to money-laundering detection for the former 2-party IPFE, both with strong privacy properties, with adaptative security and the use of labels granting a Multi-Client Functional Encryption (MCFE) security for the scheme, thus enabling its use in practical situations.

1 Introduction

Let us consider the following practical use-case: two distinct data providers (DP_1 and DP_2) with private and sensitive pieces of information want to combine them to allow a data consumer (DC) to get a result from a calculation on these. For instance, DP_1 could be a hospital with a patient's biological monitoring data on which a diagnosis depends, and it could also be affected by a private piece of information relative to the patient's personal historic, owned by DP_2. We want to grant an efficient result calculation by DC based on associated data from both data providers and public coefficients, which after the initialization phase would allow both data providers to send their encrypted data to the data consumer indefinitely without further interaction. This can be formally written as DP_1 owning a vector $\boldsymbol{x} \in \mathbb{Z}_q^n$, and DP_2 owning a vector $\boldsymbol{y} \in \mathbb{Z}_q^n$, and having DC wanting to get $\boldsymbol{x} \cdot D(\boldsymbol{\alpha}) \cdot \boldsymbol{y}^\top$, where $D(\boldsymbol{\alpha})$ is the diagonal matrix with $\boldsymbol{\alpha} \in \mathbb{Z}_q^n$ on the diagonal, *i.e.* $\sum_i \alpha_i x_i y_i$.

This is a particular case of quadratic evaluation, as recently studied in [AGT21], where the quadratic function is diagonal. But the multi-client aspect with labels has many interesting applications. For example, in the above medical use-case, we can even have a more specific setting, where DC wants to get either $\boldsymbol{x}_0 \cdot \boldsymbol{\alpha}_0^\top$ or $\boldsymbol{x}_1 \cdot \boldsymbol{\alpha}_1^\top$ according to a bit b encoded in $\boldsymbol{y}$, which can inform about particular medical predispositions:

$$\boldsymbol{x} = (\boldsymbol{x}_0, \boldsymbol{x}_1) \in \mathbb{Z}_q^{2n} \qquad \boldsymbol{y} = (\bar{b} \cdot \mathbf{1}, b \cdot \mathbf{1}) \in \mathbb{Z}_q^{2n} \qquad \boldsymbol{\alpha} = (\boldsymbol{\alpha}_0, \boldsymbol{\alpha}_1) \in \mathbb{Z}_q^{2n}$$

so that in the end, the obtained result is $\boldsymbol{x}_b \cdot \boldsymbol{\alpha}_b^\top = \langle \boldsymbol{\alpha}_b, \boldsymbol{x}_b \rangle$. This is a particular case of inner-product, with a selector b. And thanks to the labels associated to each vectors $\boldsymbol{x}$ and $\boldsymbol{y}$, and hence to $\boldsymbol{x}_0$, $\boldsymbol{x}_1$, and b, computations are restricted between values encrypted under the same labels.

Another application could be money-laundering detection: following [SNY+16], we can consider a graph of transactions between bank accounts during short windows of time: nodes are the account numbers and the oriented edges are the amounts transferred between accounts; $w_{u,v}$ is the amount of the transactions from u to v in that time-period. When there is no edge, or no transaction, $w_{u,v}$ is set to 0. If one maps all the bank accounts to $[\![1; N]\!]$, one can write, for all vertices v, the vectors of the incoming transactions $\boldsymbol{w}_{v,\mathsf{in}} = (w_{1,v}, \ldots, w_{N,v})$ and of the outgoing transactions $\boldsymbol{w}_{v,\mathsf{out}} = (w_{v,1}, \ldots, w_{v,N})$. The index of similarity between two accounts u and v is calculated as:

$$\sigma(u,v) = \sigma_{\mathsf{in}}(u,v) \times \sigma_{\mathsf{out}}(u,v)$$

where, using the norm L_2:

$$\sigma_{\mathsf{in}}(u,v) = \frac{\sum_x w_{x,u} w_{x,v}}{\sqrt{\sum_x w_{x,u}^2}\sqrt{\sum_x w_{x,v}^2}} = \langle \frac{\boldsymbol{w}_{u,\mathsf{in}}}{||\boldsymbol{w}_{u,\mathsf{in}}||}, \frac{\boldsymbol{w}_{v,\mathsf{in}}}{||\boldsymbol{w}_{v,\mathsf{in}}||} \rangle$$

$$\sigma_{\mathsf{out}}(u,v) = \frac{\sum_x w_{u,x} w_{v,x}}{\sqrt{\sum_x w_{u,x}^2}\sqrt{\sum_x w_{v,x}^2}} = \langle \frac{\boldsymbol{w}_{u,\mathsf{out}}}{||\boldsymbol{w}_{u,\mathsf{out}}||}, \frac{\boldsymbol{w}_{v,\mathsf{out}}}{||\boldsymbol{w}_{v,\mathsf{out}}||} \rangle$$

If we denote $\underline{\boldsymbol{w}} = \boldsymbol{w}/||\boldsymbol{w}||$ the normalized vector, this is

$$\sigma_{\mathsf{in}}(u,v) = \langle \underline{\boldsymbol{w}}_{u,\mathsf{in}}, \underline{\boldsymbol{w}}_{v,\mathsf{in}} \rangle \qquad \sigma_{\mathsf{out}}(u,v) = \langle \underline{\boldsymbol{w}}_{u,\mathsf{out}}, \underline{\boldsymbol{w}}_{v,\mathsf{out}} \rangle$$

As explained in [SNY+16], we can compare all the pairs of accounts to detect similar behaviors that are potentially involved in money-laundering activities. One then applies a clustering method to reduce the large size of the data set, a human investigation will confirm guilty accounts. We will thus build clusters of accounts with a similarity index above a threshold θ for a given time period.

$$\sigma(u,v) = \langle \underline{\boldsymbol{w}}_{u,\mathsf{in}}, \underline{\boldsymbol{w}}_{v,\mathsf{in}} \rangle \times \langle \underline{\boldsymbol{w}}_{u,\mathsf{out}}, \underline{\boldsymbol{w}}_{v,\mathsf{out}} \rangle$$

is the product of two inner-products on vectors generated by two independent banks: these inner-products are a particular case of our general description with $\boldsymbol{\alpha} = (1, \ldots, 1)$.

More detail, with specific optimisations about these two applications, is provided later. We first present our generic protocol to allow the DC to compute $\boldsymbol{x} \cdot D(\boldsymbol{\alpha}) \cdot \boldsymbol{y}^\top = \sum_i \alpha_i x_i y_i$. We stress that in this paper, we consider row vectors.

1.1 Functional Encryption

Functional Encryption (FE), first introduced in [O'N10,BSW11], enables the construction of protocols in which the encrypted secret information x can be partially decrypted by a party with a decryption key sk_f, with f a function, that will allow them to learn $f(x)$ and nothing else about x. The multi-client functional encryption (MCFE) generalization [CDG+18] allows the plaintext $x = (x_1, \ldots, x_n)$ to be owned and encrypted by multiple independent clients. But all the clients agree on a common label so that only those inputs can be combined into the same vector on which the functional decryption applies.

In recent years, efficient constructions for particular cases of functional encryption were studied, and first for linear functionalities: given a vector $\boldsymbol{x}$ encrypted by a party and given the functional key $\mathsf{sk}_{\boldsymbol{y}}$, for a vector $\boldsymbol{y}$, one can compute the inner-product $\langle \boldsymbol{x}, \boldsymbol{y} \rangle$ [ABDP15,ALS16], with extensions to the multi-client setting, when $\boldsymbol{x} = (x_1, \ldots, x_n)$, and each component is encrypted by a different client [CDG+18,CDSG+20]. Quadratic functions have also been dealt, but in the multi-input setting [AGT21], which differs with the multi-client setting by the absence of label. And so, all the contributions of any encryptor can be combined together into the ciphertext of $\boldsymbol{x}$ so that, given the functional key $\mathsf{sk}_{\underline{\mathbf{A}}}$, for a matrix $\underline{\mathbf{A}}$, one can compute the quadratic relation $\boldsymbol{x}\underline{\mathbf{A}}\boldsymbol{x}^\top$.

We are interested in the two-client quadratic functional encryption, where client 1 encrypts $\boldsymbol{x}$ and client 2 encrypts $\boldsymbol{y}$, under a common label so that, given the functional key $\mathsf{sk}_{\underline{\mathbf{A}}}$, for a matrix $\underline{\mathbf{A}}$, one can compute the quadratic relation $\boldsymbol{x}\underline{\mathbf{A}}\boldsymbol{y}^\top$. Our construction is limited to diagonal matrices $\underline{\mathbf{A}} = D(\boldsymbol{\alpha})$, but we will show that this also presents interesting practical applications.

1.2 MIFE and MCFE

In [DOT18], a Multi-Input Functional Encryption (MIFE) for Inner Product has already been presented, and [TT18] proposes an Inner-Product FE, both using a similar approach to ours with Dual Pairing Vector Spaces (DPVS). However, the latter paper provides a proof specific to the scheme (without multiple users), whereas we present below a generic Product-Preserving Lemma, that can be applied in several contexts. And the former paper only addresses MIFE.

While in both MIFE and MCFE, there are multiple users that can independently encrypt the multiple inputs, we here stress the main difference: MCFE allows labels, whereas MIFE does not. Hence, MCFE with a constant label reduces to MIFE. But from the security point of view, MCFE allows each client to encrypt multiple inputs with different labels. Only inputs under the same labels will be combined into a vector on which the functional decryption key will apply. In MIFE, if a client encrypts two inputs, they both can be combined with all the other clients' inputs leading to multiple vectors on which the functional decryption key can apply. This leaks so much information that all the individual inputs can be revealed. Such a leakage is not a weakness of a specific MIFE scheme, but of the functionality: this must be considered in the security model, which results in a very weak level of security. With MCFE, the security model excludes any leakage between inputs encrypted under different labels, so the concrete scheme must guarantee that, which is much more difficult to achieve.

This is the main goal of this paper: whereas [AGT21] proposed an MIFE scheme that allows to obtain the result of $\boldsymbol{x}\underline{\mathbf{A}}\boldsymbol{x}^T$ from encryptions of the coordinates of vectors $\boldsymbol{x}$ by different users without labels, and a decryption key specific to the matrix $\underline{\mathbf{A}}$, we offer a two-client FE scheme that provides the result of $\boldsymbol{x}\underline{\mathbf{A}}\boldsymbol{y}^T$ from encryptions of vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ from two independent clients, and a decryption key specific to the matrix $\underline{\mathbf{A}}$. Even if we only manage to deal with diagonal matrices, this finds applications.

1.3 Contributions

In the above money-laundering detection application, [GGJ+20] suggested using MPC with oblivious comparisons to calculate the similarity coefficients between accounts held by different banks. This means a highly interactive and synchronous protocol whereas our goal is to have an asynchronous setting where a neutral law enforcement third party receives the private data and computes the similarity coefficients.

Our main contribution is thus a two-client functional encryption scheme where client 1 owns and encrypts a vector $\boldsymbol{x} \in \mathbb{Z}_q^n$, client 2 owns and encrypts a vector $\boldsymbol{y} \in \mathbb{Z}_q^n$, while a third party knowing a functional decryption key $\mathsf{dk}_{\boldsymbol{\alpha}}$ can compute $\boldsymbol{x} \cdot D(\boldsymbol{\alpha}) \cdot \boldsymbol{y}^\top$, where $D(\boldsymbol{\alpha})$ is the diagonal matrix with public vector $\boldsymbol{\alpha} \in \mathbb{Z}_q^n$ on the diagonal: $\sum_i \alpha_i x_i y_i$.

We stress again that we are in the multi-client setting, and not just the multi-input, with independent $\boldsymbol{x}$ and $\boldsymbol{y}$ from the two clients, but associated to a common label to allow the application of the functional decryption key, and to give access to the value $\sum_i \alpha_i x_i y_i$. No information leaks if different labels are used.

To this aim, we use dual pairing vector spaces (DPVS) [OT10,OT12a] so that from ciphertexts in each of the two DPVS associated bases, the desired scalar product can be calculated, adding the public coefficients as exponents and using the associated decryption key.

More precisely, we prove and use the Product-Preserving Lemma, that essentially states that no adversary can see the difference, for each component, between the ciphertexts on x_i and y_i and the ciphertexts on 1 and $x_i y_i$, as the product is preserved. This Product-Preserving Lemma is of independent interest for other security proofs where one wants to exploit a product invariant. We provide several examples below. We now stress that our computation $\boldsymbol{x} \cdot D(\boldsymbol{\alpha}) \cdot \boldsymbol{y}^\top$, from two independent inputs $\boldsymbol{x}$ and $\boldsymbol{y}$ can be seen as a particular case from [AGT21], as they allow to compute $\boldsymbol{z}\underline{\mathbf{A}}\boldsymbol{z}^T$ which could be used with

$$\boldsymbol{z} = \boldsymbol{x} || \boldsymbol{y} \qquad\qquad \underline{\mathbf{A}} = \begin{pmatrix} \mathbf{0} & \mathbf{0} \\ D(\boldsymbol{\alpha}) & \mathbf{0} \end{pmatrix}$$

But first, this would result in an MIFE scheme, and not 2-client FE scheme as [AGT21] does not handle labels, and furthermore our functional decryption key is constant-size while theirs is linear in the size of the matrix. Eventually, their security analysis only deals with selective adversaries, who have to commit on the challenge messages before asking functional decryption keys, and ours grants adaptative security.

2 Dual Pairing Vector Spaces

Dual Pairing Vector Spaces (DPVS) have been proposed for efficient constructions with adaptive security [OT10,OT12a]. We will use them here for their orthogonality properties, and the hard subspace membership problem under the SXDH assumption.

2.1 Pairing Vector Spaces

Let us be given any cyclic group $(\mathbb{G} = \langle G \rangle, +)$ of prime order q, denoted additively. We can define the n-dimension $\mathbb{Z}_q$-vector space

$$\mathbb{G}^n = \{X = \boldsymbol{x} \cdot G \stackrel{\mathsf{def}}{=} (X_1 = x_1 \cdot G, \dots, X_n = x_n \cdot G) \,|\, \boldsymbol{x} \in \mathbb{Z}_q^n\},$$

with the following laws:

$$\boldsymbol{x} \cdot G + \boldsymbol{y} \cdot G \stackrel{\mathsf{def}}{=} (\boldsymbol{x} + \boldsymbol{y}) \cdot G \qquad\qquad a \cdot (\boldsymbol{x} \cdot G) \stackrel{\mathsf{def}}{=} (a \cdot \boldsymbol{x}) \cdot G$$

where $\boldsymbol{x} + \boldsymbol{y}$ and $a \cdot \boldsymbol{x}$ are the usual internal and external laws of the vector space $\mathbb{Z}_q^n$. For the sake of clarity, vectors will be row-vectors.

In a pairing-friendly setting $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_t, e, G_1, G_2, q)$, with a bilinear map e from $\mathbb{G}_1 \times \mathbb{G}_2$ into $\mathbb{G}_t$, where G_1 (resp. G_2) is a generator of $\mathbb{G}_1$ (resp. $\mathbb{G}_2$), we can have an additional law between elements in $\mathbb{G}_1^n$ and $\mathbb{G}_2^n$:

$$\begin{aligned} (\boldsymbol{x} \cdot G_1) \times (\boldsymbol{y} \cdot G_2) = X \times Y &= \prod_i e(X_i, Y_i) = \prod_i e(x_i \cdot G_1, y_i \cdot G_2) \\ &= \prod_i g_t^{x_i \cdot y_i} = g_t^{\boldsymbol{x} \cdot \boldsymbol{y}^\top} = g_t^{\langle \boldsymbol{x}, \boldsymbol{y} \rangle} \end{aligned}$$

where $g_t = e(G_1, G_2)$ and $\langle \boldsymbol{x}, \boldsymbol{y} \rangle$ is the inner product between vectors $\boldsymbol{x}$ and $\boldsymbol{y}$.

2.2 Dual Pairing Vector Spaces

We define $\mathcal{E} = (\boldsymbol{e}_i)_i$ the canonical basis of the n-dimensional vector space $\mathbb{Z}_q^n$, where $\boldsymbol{e}_i = (e_{i,1}, \dots, e_{i,n})$, and $e_{i,j} = \delta_{i,j}$: $\delta_{i,j} = 1$ if $i = j$ and $\delta_{i,j} = 0$ otherwise, for $i, j \in \{1, \dots, n\}$. We can also define $\mathbb{E} = (\mathbf{E}_i)_i$ the canonical basis of $\mathbb{G}^n$, where $\mathbf{E}_i = \boldsymbol{e}_i \cdot G = (\delta_{i,j} \cdot G)_j$. The above notation $\boldsymbol{x} \cdot G$ will thus be denoted $(\boldsymbol{x})_{\mathbb{E}} = \sum_i x_i \cdot \mathbf{E}_i = \boldsymbol{x} \cdot \mathbb{E}$. More generally, given any basis

$\mathcal{B} = (\boldsymbol{b}_i)_i$ of $\mathbb{Z}_q^n$, we can define the basis $\mathbb{B} = (\mathbf{B}_i)_i$ of $\mathbb{G}^n$, where $\mathbf{B}_i = \boldsymbol{b}_i \cdot G$. Choosing a random basis $\mathbb{B}$ of $\mathbb{G}^n$ is equivalent to a random choice of an invertible matrix $B \xleftarrow{\$} \mathrm{GL}_n(\mathbb{Z}_q)$, such that $\mathbb{B} = B \cdot \mathbb{E}$. In this case, we can now have $(\boldsymbol{x})_\mathbb{B} = \boldsymbol{x} \cdot \mathbb{B} = \boldsymbol{x} \cdot B \cdot \mathbb{E} = (\boldsymbol{x} \cdot B)_\mathbb{E}$.

In case of pairing-friendly setting, for a dimension n, we will denote $\mathbb{E} = (\mathbf{E}_i)_i$ and $\mathbb{E}^* = (\mathbf{E}^*_i)_i$ the canonical bases of $\mathbb{G}_1^n$ and $\mathbb{G}_2^n$, respectively: $\mathbb{E} \times \mathbb{E}^* = (g_t^{\langle e_i, e_j \rangle})_{i,j} = (g_t^{\delta_{i,j}})_{i,j} = \underline{\mathbf{Id}}_n$. The same way, if we denote $\mathbb{B} = (\mathbf{B}_i)_i = B \cdot \mathbb{E}$ the basis of $\mathbb{G}_1^n$ associated to a matrix B, and $\mathbb{B}^* = (\mathbf{B}^*_i)_i = B^* \cdot \mathbb{E}^*$ the basis of $\mathbb{G}_2^n$ associated to the matrix $B^* = (B^{-1})^\top$, as $B \cdot B^{*\top} = I_n$, $\mathbb{B} \times \mathbb{B}^* = \underline{\mathbf{Id}}_n$. $\mathbb{B}$ and $\mathbb{B}^*$ are called *Dual Orthogonal Bases*. When $(\mathbb{B}, \mathbb{B}^*)$ are associated to a random matrix $B \xleftarrow{\$} \mathrm{GL}_n(\mathbb{Z}_q)$, they are called *Random Dual Orthogonal Bases*. A pairing-friendly setting $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_t, e, G_1, G_2, q)$, with such dual orthogonal bases, is called a *Dual Pairing Vector Space*.

For avoiding ambiguous notations, we used bold letters $\boldsymbol{x}$ for vectors in $\mathbb{Z}_q$, underlined letters $\underline{\mathbf{A}}$ for matrices in $\mathbb{Z}_q$, and capital bold letters $\mathbf{E}$ for vectors in $\mathbb{G}$. More precisely, in general, $\mathbf{B}$ is a vector in $\mathbb{G}_1$ and $\mathbf{B}^*$ is a vector in $\mathbb{G}_2$. Then, $\mathbb{B}$ and $\mathbb{B}^*$ are for bases (sets of independent vectors) in $\mathbb{G}_1$ and $\mathbb{G}_2$ respectively.

2.3 Useful Transformations

As detailed in [DGP21], we can describe several transformations that are indistinguishable, either in a perfect way or in a computational way under the Decisional Diffie-Hellman Problem:

Definition 1 (Decisional Diffie-Hellman Assumption). *The DDH assumption in $\mathbb{G}$, of prime order q with generator G, states that no algorithm can efficiently distinguish the two distributions*

$$\mathcal{D}_0 = \{(a \cdot G, b \cdot G, ab \cdot G), a, b \xleftarrow{\$} \mathbb{Z}_q\} \qquad \mathcal{D}_1 = \{(a \cdot G, b \cdot G, c \cdot G), a, b, c \xleftarrow{\$} \mathbb{Z}_q\}$$

And we will denote by $\mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}}(t)$ the best advantage an algorithm can get in distinguishing the two distributions within time bounded by t.

As we will use pairing-friendly setting, we might alternate with modifications in $\mathbb{G}_1$ and $\mathbb{G}_2$. We will thus make the more general assumption:

Definition 2 (Symmetric eXternal Diffie-Hellman Assumption). *The SXDH assumption in $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_t, e, G_1, G_2, q)$ makes the DDH assumptions in both $\mathbb{G}_1$ and $\mathbb{G}_2$.*

Then, we define $\mathsf{Adv}^{\mathsf{sxdh}}(t) = \max\{\mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}_1}(t), \mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}_2}(t)\}$.

Subspace Indistinguishability. Let us consider a triple $(a \cdot G_1, b \cdot G_1, c \cdot G_1)$, for $c = ab + \tau \bmod q$, that is either a Diffie-Hellman tuple (*i.e.*, $\tau = 0 \bmod q$) or a random tuple (*i.e.*, $\tau \xleftarrow{\$} \mathbb{Z}_q$). For any random dual orthogonal bases $\mathbb{U}$ and $\mathbb{U}^*$ of dimension 2, associated to the matrices U and $U^* = (U^{-1})^\top$, respectively, we can set

$$B = \begin{pmatrix} 1 & a \\ 0 & 1 \end{pmatrix} \qquad B^* = \begin{pmatrix} 1 & 0 \\ -a & 1 \end{pmatrix} \qquad \mathbb{B} = B \cdot \mathbb{U} \qquad \mathbb{B}^* = B^* \cdot \mathbb{U}^*$$

Note that we can compute $\mathbb{B} = (\mathbf{B}_i)_i$, as we know $a \cdot G_1$ and all the scalars in U:

$$\mathbf{B}_1 = \mathbf{U}_1 + a \cdot \mathbf{U}_2 \qquad \mathbf{B}_2 = \mathbf{U}_2$$

This is the same for $\mathbb{B}^*$, excepted for the vector $\mathbf{B}^*_2$ as $a \cdot G_2$ is missing:

$$\mathbf{B}^*_1 = \mathbf{U}^*_1 \qquad \mathbf{B}^*_2 = -a \cdot \mathbf{U}^*_1 + \mathbf{U}^*_2$$

One can thus publish $\{\mathbf{B}_1, \mathbf{B}_2\}$ and $\{\mathbf{B}^*_1\}$, but not $\mathbf{B}^*_2$.

As already remarked, $(\boldsymbol{x})_{\mathbb{B}} = (\boldsymbol{x} \cdot B)_{\mathbb{U}}$, so $(\boldsymbol{x})_{\mathbb{U}} = (\boldsymbol{x} \cdot B^{-1})_{\mathbb{B}}$. Note that $B^{-1} = B^{*\top}$. In particular

$$(b, c)_{\mathbb{U}} = (b, c - ab)_{\mathbb{B}} = (b, \tau)_{\mathbb{B}}$$

where τ can be either 0 or random: the vector is thus either in the subspace spanned by $\mathbf{B}_1$ or in the full space spanned by $\mathbf{B}_1$ and $\mathbf{B}_2$.

Whereas we cannot compute $\mathbf{B}_2^*$, this does not exclude this second component in the vectors: $(\boldsymbol{y})_{\mathbb{U}^*} = (\boldsymbol{y} \cdot B^{*-1})_{\mathbb{B}^*} = (\boldsymbol{y} \cdot B^{\top})_{\mathbb{B}^*}$. So, in particular $(y_1, y_2)_{\mathbb{U}^*} = (y_1 + ay_2, y_2)_{\mathbb{B}^*}$. And when $y_2 = 0$, $(y_1, 0)_{\mathbb{U}^*} = (y_1, 0)_{\mathbb{B}^*}$.

Theorem 3 (Subspace Indistinguishability). *Under the DDH assumption in $\mathbb{G}_1$, for random dual orthogonal bases $\mathbb{B}$ and $\mathbb{B}^*$, once having seen $\mathbb{B}$ and $\mathbb{B}^* \backslash \{\mathbf{B}_2^*\}$, and any vector $(y_1, 0)_{\mathbb{B}^*}$, for any $y_1 \in \mathbb{Z}_q$, one cannot distinguish the vectors $(x_1, 0)_{\mathbb{B}}$ and $(x_1, x_2)_{\mathbb{B}}$, for unknown random $x_1, x_2 \xleftarrow{\$} \mathbb{Z}_q$. More precisely, the best advantage one can get is bounded by $2 \times \mathsf{Adv}_{\mathbb{G}_1}^{\mathsf{ddh}}(t)$.*

Balancing Indistinguishability. With a transformation of the bases, it is possible to alter two components in a vector when there is twice the same value in the other vector, using

$$B = \begin{pmatrix} 1 & -a & a \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \qquad B' = \begin{pmatrix} 1 & 0 & 0 \\ a & 1 & 0 \\ -a & 0 & 1 \end{pmatrix}$$

Lemma 4 (Balancing Indistinguishability). *Under the DDH assumption in $\mathbb{G}_1$, for random dual orthogonal bases $\mathbb{B}$ and $\mathbb{B}^*$, once having seen $\mathbb{B}$ and $\mathbb{B}^* \backslash \{\mathbf{B}_2^*, \mathbf{B}_3^*\}$ and any vector $(y_1, y_2, y_2)_{\mathbb{B}^*}$, for any $y_1, y_2 \in \mathbb{Z}_q$, one cannot distinguish the vectors $(x_1, 0, 0)_{\mathbb{B}}$ and $(x_1, -x_2, x_2)_{\mathbb{B}}$, for unknown random $x_1, x_2 \xleftarrow{\$} \mathbb{Z}_q$. More precisely, the best advantage one can get is bounded by $\mathsf{Adv}_{\mathbb{G}_1}^{\mathsf{ddh}}(t)$.*

Swapping Indistinguishability. Applying twice the above Balancing Indistinguishability Lemma 4, with a chosen shift x_2 on $\mathbf{B}_2$ first, with indistinguishability between $(x_1, x_2, 0)_{\mathbb{B}}$ and $(x_1, x_2 - \theta, \theta)_{\mathbb{B}}$, and thus $(x_1, \theta', \theta)_{\mathbb{B}}$ with $\theta + \theta' = x_2$, and then a second chosen shift x_2 on $\mathbf{B}_3$, with indistinguishability between $(x_1, 0, x_2)_{\mathbb{B}}$ and $(x_1, -\theta, x_2 + \theta)_{\mathbb{B}}$, and thus again $(x_1, \theta', \theta)_{\mathbb{B}}$ with $\theta + \theta' = x_2$:

Theorem 5 (Swapping Indistinguishability). *Under the DDH assumption in $\mathbb{G}_1$, for random dual orthogonal bases $\mathbb{B}$ and $\mathbb{B}^*$, once having seen $\mathbb{B}$ and $\mathbb{B}^* \backslash \{\mathbf{B}_2^*, \mathbf{B}_3^*\}$ and any vector $(y_1, y_2, y_2)_{\mathbb{B}^*}$, for any $y_1, y_2 \in \mathbb{Z}_q$, one cannot distinguish the vectors $(x_1, x_2, 0)_{\mathbb{B}}$ and $(x_1, 0, x_2)_{\mathbb{B}}$, for an unknown random $x_1 \xleftarrow{\$} \mathbb{Z}_q$, but chosen $x_2 \in \mathbb{Z}_q$. More precisely, the best advantage one can get is bounded by $2 \times \mathsf{Adv}_{\mathbb{G}_1}^{\mathsf{ddh}}(t)$.*

Indexing Indistinguishability. We will make use of the major transformation introduced in [OT12b], with orthogonal indices that allow alterations of many components at once, in an indistinguishable way, if labels λ and λ' differ using

$$B = \frac{1}{\lambda' - \lambda} \begin{pmatrix} \lambda' & -\lambda & a\lambda' \\ -1 & 1 & -a \\ 0 & 0 & \lambda' - \lambda \end{pmatrix} \qquad B' = \begin{pmatrix} 1 & 1 & 0 \\ \lambda & \lambda' & 0 \\ -a & 0 & 1 \end{pmatrix}$$

Theorem 6 (Indexing Indistinguishability). *Under the DDH assumption in $\mathbb{G}_1$, for fixed distinct labels $\lambda, \lambda' \in \mathbb{Z}_q$, random dual orthogonal bases $\mathbb{B}$ and $\mathbb{B}^*$, once having seen $\mathbb{B}$ and $\mathbb{B}^* \backslash \{\mathbf{B}_3^*\}$ and any vector $(\pi \cdot (\lambda', -1), y)_{\mathbb{B}^*}$, for any $y \in \mathbb{Z}_q$ but an unknown random $\pi \xleftarrow{\$} \mathbb{Z}_q$, one cannot distinguish the vectors $(\mu \cdot (1, \lambda), 0)_{\mathbb{B}}$ and $(\mu \cdot (1, \lambda), x)_{\mathbb{B}}$, for any $x \in \mathbb{Z}_q$ but an unknown random $\mu \xleftarrow{\$} \mathbb{Z}_q$. More precisely, the best advantage one can get is bounded by $2 \times \mathsf{Adv}_{\mathbb{G}_1}^{\mathsf{ddh}}(t)$.*

Replication Indistinguishability. Eventually, we will have to replicate some components, in order to prepare a balance or swap transformation on the other vector. In this case, this is perfectly indistinguishable using

$$B = \begin{pmatrix} 1 & -1 \\ 0 & 1 \end{pmatrix} \qquad\qquad B' = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}$$

Lemma 7 (Replication Indistinguishability). *For random dual orthogonal bases $\mathbb{B}$ and $\mathbb{B}^*$, once having seen $\mathbb{B}\backslash\{\mathbf{B}_1\}$, $\mathbb{B}^*\backslash\{\mathbf{B}_2^*\}$, one cannot distinguish the vectors $(x_1, x_2)_{\mathbb{B}}, (y_1, y_2)_{\mathbb{B}^*}$ and $(x_1, x_1 + x_2)_{\mathbb{B}}, (y_1 - y_2, y_2)_{\mathbb{B}^*}$, for any $x_1, x_2, y_1, y_2 \in \mathbb{Z}_q$.*

We stress that for the above theorem, $\mathbf{B}_1$ and $\mathbf{B}_2^*$ must not be shown to the adversary, and then with just $\mathbf{B}_2 = \mathbf{U}_2$ and $\mathbf{B}_1^* = \mathbf{U}_1^*$, the adversary cannot know whether one is using $(\mathbb{U}, \mathbb{U}^*)$ or $(\mathbb{B}, \mathbb{B}^*)$.

3 Product-Preserving Lemma

We start with a lemma that will be the basis of our results and applications, using several times the above indistinguishability theorems. This lemma is our main contribution. It is of independent interest, with various applications.

Lemma 8 (Product-Preserving Lemma). *For two random dual orthogonal bases $\mathbb{B}$ and $\mathbb{B}^*$, of length 5, unknown to the adversary $\mathcal{A}$, given a list of indexed vectors, for any (x_i, y_i, a_i, b_i), fixed indices λ_i among Q possible values, but random unknown π_i, μ_i, for $i = [\![1; N]\!]$:*

$$\mathbf{C}_i = (\pi_i(1, \lambda_i), x_i, a_i, 0)_{\mathbb{B}} \qquad\qquad \mathbf{D}_i = (\mu_i(\lambda_i, -1), y_i, b_i, 0)_{\mathbb{B}^*}$$

$\mathcal{A}$ cannot distinguish between the two cases, for fixed $\lambda^* \neq \lambda_i$, and any x^*, y^*, but unknown random π^*, μ^*:

$$\begin{array}{lll} (0) & \mathbf{C}^* = (\pi^*(1, \lambda^*), x^*, 0, 0)_{\mathbb{B}} & \mathbf{D}^* = (\mu^*(\lambda^*, -1), y^*, 0, 0)_{\mathbb{B}^*} \\ (1) & \mathbf{C}^* = (\pi^*(1, \lambda^*), 0, 1, 0)_{\mathbb{B}} & \mathbf{D}^* = (\mu^*(\lambda^*, -1), 0, x^* y^*, 0)_{\mathbb{B}^*} \end{array}$$

with an advantage greater than $4 \cdot (NQ + 1) \cdot (\mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}_1}(t) + \mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}_2}(t))$, and thus greater than $8 \cdot (NQ + 1) \cdot \mathsf{Adv}^{\mathsf{sxdh}}(t)$.

In Figure 1, we present the sequence of games of the security proof of the above lemma, where the π_i's and the μ_i's are used when unknown randomness is required to apply swapping indistinguishability. Grey cells highlight the modified values, and the various transformations of bases are precised in the left column. We stress that the indices $(\lambda_i)_i$ and λ^* have to be known in advance, to be able to enumerate them deterministically, and apply the indexing indistinguishability several taimes. We thus bound their number by Q (which will be the number of hash queries in our applications). The full proof can be found in appendix A.

4 2-Client Inner-Product Functional Encryption

In this section, we first recall the general protocol we want to instantiate, and then we provide and prove a concrete construction.

$\mathbf{G}_0$	$\forall i$	$\mathbf{C}_i = ($	$\pi_i(1,\lambda_i)$	$\mid$	x_i	a_i	$\mid$	0	$)$
		$\mathbf{D}_i = ($	$\mu_i(\lambda_i,-1)$	$\mid$	y_i	b_i	$\mid$	0	$)$
		$\mathbf{C}^* = ($	$\pi^*(1,\lambda^*)$	$\mid$	x^*	0	$\mid$	0	$)$
		$\mathbf{D}^* = ($	$\mu^*(\lambda^*,-1)$	$\mid$	y^*	0	$\mid$	0	$)$
$\mathbf{G}_1$	$\forall i$	$\mathbf{C}_i = ($	$\pi_i(1,\lambda_i)$	$\mid$	x_i	a_i	$\mid$	0	$)$
Replication		$\mathbf{D}_i = ($	$\mu_i(\lambda_i,-1)$	$\mid$	y_i	b_i	$\mid$	y_i	$)$
		$\mathbf{C}^* = ($	$\pi^*(1,\lambda^*)$	$\mid$	x^*	0	$\mid$	0	$)$
		$\mathbf{D}^* = ($	$\mu^*(\lambda^*,-1)$	$\mid$	y^*	0	$\mid$	y^*	$)$
$\mathbf{G}_2$	$\forall i$	$\mathbf{C}_i = ($	$\pi_i(1,\lambda_i)$	$\mid$	x_i	a_i	$\mid$	0	$)$
Swapping		$\mathbf{D}_i = ($	$\mu_i(\lambda_i,-1)$	$\mid$	y_i	b_i	$\mid$	y_i	$)$
		$\mathbf{C}^* = ($	$\pi^*(1,\lambda^*)$	$\mid$	0	0	$\mid$	x^*	$)$
		$\mathbf{D}^* = ($	$\mu^*(\lambda^*,-1)$	$\mid$	y^*	0	$\mid$	y^*	$)$
$\mathbf{G}_3$	$\forall i$	$\mathbf{C}_i = ($	$\pi_i(1,\lambda_i)$	$\mid$	x_i	a_i	$\mid$	0	$)$
Indexing		$\mathbf{D}_i = ($	$\mu_i(\lambda_i,-1)$	$\mid$	y_i	b_i	$\mid$	0	$)$
		$\mathbf{C}^* = ($	$\pi^*(1,\lambda^*)$	$\mid$	0	0	$\mid$	x^*	$)$
		$\mathbf{D}^* = ($	$\mu^*(\lambda^*,-1)$	$\mid$	y^*	0	$\mid$	y^*	$)$
$\mathbf{G}_4$	$\forall i$	$\mathbf{C}_i = ($	$\pi_i(1,\lambda_i)$	$\mid$	x_i	a_i	$\mid$	x_i	$)$
Replication		$\mathbf{D}_i = ($	$\mu_i(\lambda_i,-1)$	$\mid$	y_i	b_i	$\mid$	0	$)$
		$\mathbf{C}^* = ($	$\pi^*(1,\lambda^*)$	$\mid$	0	0	$\mid$	x^*	$)$
		$\mathbf{D}^* = ($	$\mu^*(\lambda^*,-1)$	$\mid$	0	0	$\mid$	y^*	$)$
$\mathbf{G}_5$	$\forall i$	$\mathbf{C}_i = ($	$\pi_i(1,\lambda_i)$	$\mid$	x_i	a_i	$\mid$	0	$)$
Indexing		$\mathbf{D}_i = ($	$\mu_i(\lambda_i,-1)$	$\mid$	y_i	b_i	$\mid$	0	$)$
		$\mathbf{C}^* = ($	$\pi^*(1,\lambda^*)$	$\mid$	0	0	$\mid$	x^*	$)$
		$\mathbf{D}^* = ($	$\mu^*(\lambda^*,-1)$	$\mid$	0	0	$\mid$	y^*	$)$
$\mathbf{G}_6$	$\forall i$	$\mathbf{C}_i = ($	$\pi_i(1,\lambda_i)$	$\mid$	x_i	a_i	$\mid$	0	$)$
Quotient		$\mathbf{D}_i = ($	$\mu_i(\lambda_i,-1)$	$\mid$	y_i	b_i	$\mid$	0	$)$
for $x^* \neq 0$		$\mathbf{C}^* = ($	$\pi^*(1,\lambda^*)$	$\mid$	0	0	$\mid$	1	$)$
		$\mathbf{D}^* = ($	$\mu^*(\lambda^*,-1)$	$\mid$	0	0	$\mid$	x^*y^*	$)$
$\mathbf{G}_7$	$\forall i$	$\mathbf{C}_i = ($	$\pi_i(1,\lambda_i)$	$\mid$	x_i	a_i	$\mid$	0	$)$
Subspace		$\mathbf{D}_i = ($	$\mu_i(\lambda_i,-1)$	$\mid$	y_i	b_i	$\mid$	0	$)$
for $x^* = 0$		$\mathbf{C}^* = ($	$\pi^*(1,\lambda^*)$	$\mid$	0	0	$\mid$	1	$)$
		$\mathbf{D}^* = ($	$\mu^*(\lambda^*,-1)$	$\mid$	0	0	$\mid$	0	$)$
$\mathbf{G}_8$	$\forall i$	$\mathbf{C}_i = ($	$\pi_i(1,\lambda_i)$	$\mid$	x_i	a_i	$\mid$	0	$)$
Replication		$\mathbf{D}_i = ($	$\mu_i(\lambda_i,-1)$	$\mid$	y_i	b_i	$\mid$	$-b_i$	$)$
		$\mathbf{C}^* = ($	$\pi^*(1,\lambda^*)$	$\mid$	0	1	$\mid$	1	$)$
		$\mathbf{D}^* = ($	$\mu^*(\lambda^*,-1)$	$\mid$	0	0	$\mid$	x^*y^*	$)$
$\mathbf{G}_9$	$\forall i$	$\mathbf{C}_i = ($	$\pi_i(1,\lambda_i)$	$\mid$	x_i	a_i	$\mid$	a_i	$)$
Indexing		$\mathbf{D}_i = ($	$\mu_i(\lambda_i,-1)$	$\mid$	y_i	b_i	$\mid$	0	$)$
		$\mathbf{C}^* = ($	$\pi^*(1,\lambda^*)$	$\mid$	0	1	$\mid$	1	$)$
		$\mathbf{D}^* = ($	$\mu^*(\lambda^*,-1)$	$\mid$	0	0	$\mid$	x^*y^*	$)$
$\mathbf{G}_{10}$	$\forall i$	$\mathbf{C}_i = ($	$\pi_i(1,\lambda_i)$	$\mid$	x_i	a_i	$\mid$	a_i	$)$
Swapping		$\mathbf{D}_i = ($	$\mu_i(\lambda_i,-1)$	$\mid$	y_i	b_i	$\mid$	0	$)$
		$\mathbf{C}^* = ($	$\pi^*(1,\lambda^*)$	$\mid$	0	1	$\mid$	1	$)$
		$\mathbf{D}^* = ($	$\mu^*(\lambda^*,-1)$	$\mid$	0	x^*y^*	$\mid$	0	$)$
$\mathbf{G}_{11}$	$\forall i$	$\mathbf{C}_i = ($	$\pi_i(1,\lambda_i)$	$\mid$	x_i	a_i	$\mid$	0	$)$
Replication		$\mathbf{D}_i = ($	$\mu_i(\lambda_i,-1)$	$\mid$	y_i	b_i	$\mid$	0	$)$
		$\mathbf{C}^* = ($	$\pi^*(1,\lambda^*)$	$\mid$	0	1	$\mid$	0	$)$
		$\mathbf{D}^* = ($	$\mu^*(\lambda^*,-1)$	$\mid$	0	x^*y^*	$\mid$	0	$)$

Fig. 1. Sequence of Games for the Product-Preserving Lemma

4.1 General Protocol Description

Two data providers DP_1 and DP_2 send ciphertexts based on their data $\boldsymbol{x}$ and $\boldsymbol{y} \in \mathbb{Z}_q^n$ respectively to a data consumer DC. DC has a functional decryption key $\mathsf{dk}_{\underline{\mathbf{A}}}$ for $\underline{\mathbf{A}} = \mathsf{Diag}((\alpha_i)_{i \in [\![1;n]\!]}) \in \mathbb{Z}_q^{n \times n}$ enabling them to learn $\boldsymbol{x}\underline{\mathbf{A}}\boldsymbol{y}^T = \sum_i \alpha_i x_i y_i$ and nothing else about $\boldsymbol{x}$ nor $\boldsymbol{y}$. This is defined using four algorithms:

$\mathsf{Setup}(1^\kappa)$**.** This algorithm generates the encryption keys ek_1 and ek_2 for DP_1 and DP_2 respectively, and the secret key sk;

$\mathsf{KeyGen}(\mathsf{sk}, \underline{\mathbf{A}})$**.** For $\underline{\mathbf{A}} = \mathsf{Diag}((\alpha_i)_{i \in [\![1;n]\!]}) \in \mathbb{Z}_q^{n \times n}$, using the secret key sk, one generates the functional decryption key $\mathsf{dk}_{\underline{\mathbf{A}}}$;

$\mathsf{Encrypt}(\lambda, \mathsf{ek}, \boldsymbol{z})$**.** Given the label λ and $(\mathsf{ek}, \boldsymbol{z})$, either equal to $(\mathsf{ek}_1, \boldsymbol{x})$ or $(\mathsf{ek}_2, \boldsymbol{y})$, the corresponding data provider returns the ciphertext $\mathbf{E}$;

$\mathsf{Decrypt}(\mathsf{dk}_{\boldsymbol{\alpha}}, \mathbf{C}, \mathbf{D})$**.** Where $\mathbf{C}$ is a ciphertext obtained from DP_1 and $\mathbf{D}$ from DP_2 under labels λ_1 and λ_2 respectively, returns $\boldsymbol{x}\underline{\mathbf{A}}\boldsymbol{y}^T = \sum_i \alpha_i x_i y_i$ if $\lambda_1 = \lambda_2$.

4.2 Our 2-Client Inner-Product Functional Encryption

We provide a construction for the general protocol described in section 4.1 hereafter, using ciphertexts made of vectors of elements in two groups $\mathbb{G}_1$ and $\mathbb{G}_2$ used for pairings, with the first data provider DP_1 using elements from $\mathbb{G}_1$ and the second, DP_2, from $\mathbb{G}_2$. We take advantage of dual pairing vector spaces, ciphertext group exponents being components of an element of $\mathrm{GL}_{12}(\mathbb{Z}_q)$ decomposed into one of its bases $\mathbb{B}$ for DP_1, or in its dual base $\mathbb{B}^*$ for DP_2. As such, when a pairing is performed between ciphertexts from the two data providers, the exponent of the new target group element is the scalar product of the vectors of exponents in both data providers' ciphertexts, and this will procure the expected result, with additional enforcement of identical labels in both ciphertexts, and randomness and additional components to ensure privacy.

Combining the above product-preserving lemma together with IPFE scheme from [ALS16], we can obtain a secure construction, even against adaptive adversaries. However, since we will have two IPFE in $\mathbb{G}_1$ and $\mathbb{G}_2$, we will need secret keys $\underline{\mathbf{S}} = (\boldsymbol{s}_i)_i$ and $\underline{\mathbf{T}} = (\boldsymbol{t}_i)_i$ of dimension 3. To this aim we will use 4 times the above Product-Preserving Lemma 8, on the values $(\boldsymbol{x}_\ell, \boldsymbol{y}_\ell) = (x_{\ell,i}, y_{\ell,i})_i$, for all the labels ℓ, and on the three pairs of randomness $(\sigma_{\ell,j} s_{i,j}, \tau_{\ell,j} t_{i,j})$, for $j = 1, 2, 3$, for all ℓ, i, to lead to unique vectors $\boldsymbol{z}_\ell = (z_{\ell,i} = x_{\ell,i} y_{\ell,i})_i$, the unique secret key $\underline{\mathbf{R}} = (\boldsymbol{r}_i = (r_{i,j} = s_{i,j} t_{i,j})_j)_i$, as well as global randomnesses $\boldsymbol{\rho}_\ell = (\rho_{\ell,j} = \sigma_{\ell,j} \tau_{\ell,j})_j$.

We use a Dual Pairing Vector Space of dimension 12, in a pairing-friendly setting $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_t, e, G_1, G_2, q)$:

$\mathsf{Setup}(1^\kappa)$**.** From a pairing-friendly setting $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_t, e, G_1, G_2, q)$ of appropriate size with respect to the security parameter κ, one chooses random dual orthogonal matrices $B = (\boldsymbol{b}_i)_i \xleftarrow{\$} \mathrm{GL}_{12}(\mathbb{Z}_q)$ and $B^* = (\boldsymbol{b}_i^*)_i \leftarrow (B^{-1})^\top$, which define the bases $(\mathbb{B}, \mathbb{B}^*)$. One also generates secret vectors $\underline{\mathbf{S}} = (\boldsymbol{s}_i)_i, \underline{\mathbf{T}} = (\boldsymbol{t}_i)_i \xleftarrow{\$} (\mathbb{Z}_q^3)^n$. The encryption key ek_1 of DP_1 is set to $(B, \underline{\mathbf{S}})$, while the encryption key ek_2 of DP_2 is set to $(B^*, \underline{\mathbf{T}})$. The secret key is $\mathsf{sk} = (\underline{\mathbf{S}}, \underline{\mathbf{T}})$. The public parameters are just $\mathsf{PK} \leftarrow (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_t, e, G_1, G_2, q)$, together with a collision-resistant hash function $\mathcal{H}$ into $\mathbb{Z}_q$;

$\mathsf{KeyGen}(\mathsf{sk}, \boldsymbol{\alpha})$**.** For a vector $\boldsymbol{\alpha} \in \mathbb{Z}_q^n$, one generates the functional decryption key $\mathsf{dk}_{\boldsymbol{\alpha}} \leftarrow \boldsymbol{d} = \sum \alpha_i (s_{i,1} t_{i,1}, s_{i,2} t_{i,2}, s_{i,3} t_{i,3}) \in \mathbb{Z}_q^3$;

$\mathsf{Encrypt}(\lambda, \mathsf{ek}, \boldsymbol{z})$**,** for a label λ, where ek is either ek_1 or ek_2, and $\boldsymbol{z}$ is either $\boldsymbol{x}$ or $\boldsymbol{y}$, one derives $\lambda_i = \mathcal{H}(\lambda, i) \in \mathbb{Z}_q$, for $i \in [\![1;n]\!]$:

- To encrypt $\boldsymbol{x}$ with $\mathsf{ek}_1 = (B, \underline{\mathbf{S}})$, one generates $\boldsymbol{\sigma} \overset{\$}{\leftarrow} (\mathbb{Z}_q^*)^3$, random $\pi_i \overset{\$}{\leftarrow} \mathbb{Z}_q$, and sets $\mathbf{C} \leftarrow (\lambda, \boldsymbol{C}_0, (\boldsymbol{C}_i)_i)$, where $C_{0,j} = \sigma_j \cdot G_1$, for $j = 1, 2, 3$, and for $i = 1 \ldots, n$,

$$\begin{aligned}\boldsymbol{C}_i &= (\pi_i(\boldsymbol{b}_1 + \lambda_i \cdot \boldsymbol{b}_2) + x_i \cdot \boldsymbol{b}_3 + \sum_{j=1}^{3} \sigma_j s_{i,j} \cdot \boldsymbol{b}_{4+2j}) \cdot G_1 \\ &= (\pi_i \cdot (1, \lambda_i), x_i, 0, 0, \sigma_1 s_{i,1}, 0, \sigma_2 s_{i,2}, 0, \sigma_3 s_{i,3}, 0, 0)_{\mathbb{B}} \\ &= (\pi_i \cdot (1, \lambda_i), x_i, 0, 0, (\sigma_j s_{i,j}, 0)_j, 0)_{\mathbb{B}}\end{aligned}$$

- To encrypt $\boldsymbol{y}$ with $\mathsf{ek}_2 = (B^*, \underline{\mathbf{T}})$, one generates $\boldsymbol{\tau} \overset{\$}{\leftarrow} (\mathbb{Z}_q^*)^3$, random $\mu_i \overset{\$}{\leftarrow} \mathbb{Z}_q$, and sets $\mathbf{D} \leftarrow (\lambda, \boldsymbol{D}_0, (\boldsymbol{D}_i)_i)$, where $D_{0,j} = \tau_j \cdot G_2$, for $j = 1, 2, 3$, and for $i = 1 \ldots, n$,

$$\begin{aligned}\boldsymbol{D}_i &= (\mu_i(\lambda_i \boldsymbol{b}_1^* - \boldsymbol{b}_2^*) + y_i \cdot \boldsymbol{b}_3^* + \sum_{j=1}^{3} \tau_j t_{i,j} \cdot \boldsymbol{b}_{4+2j}^*) \cdot G_2 \\ &= (\mu_i \cdot (\lambda_i, -1), y_i, 0, 0, \tau_1 t_{i,1}, 0, \tau_2 t_{i,2}, 0, \tau_3 t_{i,3}, 0, 0)_{\mathbb{B}^*} \\ &= (\mu_i \cdot (\lambda_i, -1), y_i, 0, 0, (\tau_j t_{i,j}, 0)_j, 0)_{\mathbb{B}^*}\end{aligned}$$

$\mathsf{Decrypt}(\mathsf{dk}_{\boldsymbol{\alpha}}, \mathbf{C}, \mathbf{D})$**.** Due to the orthogonality of the bases $\mathbb{B}$ and $\mathbb{B}^*$, with $g_t = e(G_1, G_2)$, one has

$$\boldsymbol{C}_i \times \boldsymbol{D}_i = [\pi_i \mu_i (\lambda_i' - \lambda_i) + x_i y_i + \sum_{j=1}^{3} \sigma_j \tau_j s_{i,j} t_{i,j}]_{\mathbb{G}_t}$$

If the labels are the same, then $\boldsymbol{C}_i \times \boldsymbol{D}_i = g_t^{x_i y_i} \cdot \prod_j (g_t^{s_{i,j} t_{i,j}})^{\sigma_j \tau_j}$. Then,

$$\begin{aligned}\prod_i (\boldsymbol{C}_i \times \boldsymbol{D}_i)^{\alpha_i} &= \prod_i g_t^{\alpha_i x_i y_i} \cdot \prod_j (g_t^{\alpha_i s_{i,j} t_{i,j}})^{\sigma_j \tau_j} \\ &= g_t^{\sum_i \alpha_i x_i y_i} \cdot \prod_j (g_t^{d_j})^{\sigma_j \tau_j} \\ &= g_t^{\sum_i \alpha_i x_i y_i} \cdot \prod_j e(C_{0,j}, D_{0,j})^{d_j}\end{aligned}$$

One can thus get $g_t^{\sum_i \alpha_i x_i y_i}$ and extract the discrete logarithm $\sum_i \alpha_i x_i y_i$.

4.3 Security Model

About the security, it is clear neither DP_1 nor DP_2 are allowed to see each other's inputs: given $\boldsymbol{C}_i = (\pi_i \cdot (\boldsymbol{b}_1 + \lambda_i \cdot \boldsymbol{b}_2) + x_i \cdot \boldsymbol{b}_3 + \sigma_1 s_{i,1} \cdot \boldsymbol{b}_6 + \sigma_2 s_{i,2} \cdot \boldsymbol{b}_8 + \sigma_3 s_{i,3} \cdot \boldsymbol{b}_{10}) \cdot G_1$, DP_2 can compute $\boldsymbol{C}_i \times (\boldsymbol{b}_3 \cdot G_1) = g_t^{x_i}$. This thus completely leaks x_i.

However, we can expect the Data Consumer not to learn more than $\sum_i \alpha_i x_i y_i$ on a pair of inputs. And moreover, multiple pairs of inputs cannot be mixed. We thus define a security game, where the adversary can get multiple functional decryption keys $\mathsf{dk}_{\boldsymbol{\alpha}}$ for any vector $\boldsymbol{\alpha}$ of their choice, multiple ciphertexts on $(\lambda, \boldsymbol{x}^b, \boldsymbol{y}^b)$ for $(\lambda, \boldsymbol{x}^0, \boldsymbol{x}^1)$ and $(\lambda, \boldsymbol{y}^0, \boldsymbol{y}^1)$ of the adversary's choice, in any order of their choice, but for a random bit b. The adversary wins the game if they have correctly guessed b, without cheating: for all the $\boldsymbol{\alpha}$ for which it got the functional decryption keys, and all the labels λ for which it asked $(\lambda, \boldsymbol{x}^0, \boldsymbol{x}^1)$ and $(\lambda, \boldsymbol{y}^0, \boldsymbol{y}^1)$, $\sum \alpha_i x_i^0 y_i^0 = \sum \alpha_i x_i^1 y_i^1$. Otherwise, this is easy to guess b.

The goal of the adversary is to guess b, in this classical indistinguishability game, that we model with a left-or-right game, with many queries:

Definition 9 (Indistinguishability). *The* IND *security game is defined as follows:*

- $\mathsf{Initialize}$: *The challenger runs the* Setup *algorithm and gives the public parameters* PK *to the adversary. They also draw a random coin* $b \overset{\$}{\leftarrow} \{0,1\}$*;*
- $\mathsf{OKeyGen}(\boldsymbol{\alpha})$: *this is a* KeyGen*-query for any vector* $\boldsymbol{\alpha}$*. It outputs the functional decryption key* $\mathsf{dk}_{\boldsymbol{\alpha}} \leftarrow \mathsf{KeyGen}(\mathsf{sk}, \boldsymbol{\alpha})$*;*
- $\mathsf{LoREncrypt}_{\mathfrak{b}}(\lambda, \boldsymbol{z}^0, \boldsymbol{z}^1)$: *this is a Left-or-Right query, on a new label* λ *for* $\mathfrak{b} \in \{1,2\}$*, and a pair of vectors* $(\boldsymbol{z}^0, \boldsymbol{z}^1)$*. It outputs* $\mathbf{C}^*_{\mathfrak{b}} \leftarrow \mathsf{Encrypt}(\lambda, \mathsf{ek}_{\mathfrak{b}}, \boldsymbol{z}^b)$*;*
- $\mathsf{Finalize}(b')$: *The adversary outputs a guess* b' *for* b*. If for all the* $\boldsymbol{\alpha}$ *asked to the* $\mathsf{OKeyGen}$*-oracle, and all the pairs* $(\boldsymbol{x}^0_\ell, \boldsymbol{x}^1_\ell)$ *and* $(\boldsymbol{y}^0_\ell, \boldsymbol{y}^1_\ell)$ *asked to the* $\mathsf{LoREncrypt}$*-oracle* $\sum \alpha_i x^0_{\ell,i} y^0_{\ell,i} = \sum \alpha_i x^1_{\ell,i} y^1_{\ell,i}$*, where we denote* $(\boldsymbol{x}^0_\ell, \boldsymbol{x}^1_\ell)$ *the pair* $(\boldsymbol{z}^0, \boldsymbol{z}^1)$ *asked for the* ℓ*-th label* λ_ℓ *with* $\mathfrak{b} = 1$ *and* $(\boldsymbol{y}^0_\ell, \boldsymbol{y}^1_\ell)$ *for* $\mathfrak{b} = 2$*. Then one outputs* b' *(for legitimate attacks), otherwise one outputs a random bit (for non-legitimate attacks).*

The advantage of an adversary $\mathcal{A}$ *in this indistinguishability game is defined as* $\mathsf{Adv}^{\mathsf{ind}}(\mathcal{A}) = \Pr[1|b=1] - \Pr[1|b=0] = 2\Pr[b' = b|\mathit{legitimate}] - 1$*.*

We stress that in the above security game, the adversary can ask for $\mathsf{OKeyGen}$ and $\mathsf{LoREncrypt}$ oracles as many times as they want, and in any order. When we specify a "new label", this means that a label can appear only once for each value of $\mathfrak{b}$. If the scheme allows to learn some informations between two vectors encrypted under different labels, this breaks this security level. This would be the case with an MIFE scheme, unable to handle labels.

Definition 10 (Ordered Indistinguishability). *The* ord-IND *security game is exactly the same as the* IND *game except that for each label* λ*,* $\mathsf{LoREncrypt}_2$ *must always be asked before* $\mathsf{LoREncrypt}_1$*.*

We will need the ord-IND security notion in our second construction to drastically reduce the number of ciphertexts sent by DP_2 in the particular case where this data provider sends information to select one or the other half of DP_1's vector. In this case, for a given label, DP_2 must send their ciphertext before DP_1 sends ciphertexts with the same label, because if these were received in the opposite order, our security games would not allow us to apply the product-preserving lemma storing the product in DP_1's ciphertext vector. This is a constraint cominig from our proof: we cannot think of any practical attack if ciphertexts were sent in the opposite order, but we would then not be able to prove the security. This security level is definitely weaker in theory, as it introduces some constraint for the adversary. However, in practice, one can simply make sure DP_2 first sends all its ciphertexts, and then DP_1 does. This is a similar organizational constraint as not generating two ciphertexts under the same label for each data provider. If these constraints are not satisfied, all the security guarantees are lost.

4.4 Security Analysis

Theorem 11. *Our above construction from Section 4.2 satisfies* IND *indistinguishability against adaptive adversaries in time* t *with:*

$$\mathsf{Adv}^{\mathsf{ind}}(\mathcal{A}) \leq (32QN(Nn+1) + 3N + 8) \cdot \mathsf{Adv}^{\mathsf{sxdh}}(t) + 3N/q$$

where N *is the number of different labels the adversary got encryptions for, and* Q *the number of labels queried to the hash function.*

In Figure 2, we present the sequence of games, starting from the real game, where the λ_ℓ are all the distinct labels, and $\lambda_{\ell,i} = \mathcal{H}(\lambda_\ell, i)$ for $i \in [\![1;n]\!]$, for ℓ increased from 1 to N, the total number of distinct labels. We denote $\mathcal{L}_1$ the set of indices ℓ where $\mathsf{LoREncrypt}_1$ is asked first, and $\mathcal{L}_2$ the set of indices where $\mathsf{LoREncrypt}_2$ is asked first. Their sizes are respectively N_1 and N_2. However, we have to get the possible $\lambda_{\ell,i}$ known in advance, in order to enumerate them in the proofs, even if not actually used. We thus program the random oracle for all the Q λ-queries

$\mathbf{G}_0$	$\forall \ell \in \mathcal{L}_1,$	$\boldsymbol{C}_{\ell,i} = (\cdots \mid$	$x^b_{\ell,i}$	0	0	$\mid (\sigma_{\ell,j} s_{i,j}$	0	$)_j \mid 0 \;)$
Random b		$\boldsymbol{D}_{\ell,i} = (\cdots \mid$	$y^b_{\ell,i}$	0	0	$\mid (\tau_{\ell,j} t_{i,j}$	0	$)_j \mid 0 \;)$
	$\forall \ell \in \mathcal{L}_2,$	$\boldsymbol{C}_{\ell,i} = (\cdots \mid$	$x^b_{\ell,i}$	0	0	$\mid (\sigma_{\ell,j} s_{i,j}$	0	$)_j \mid 0 \;)$
		$\boldsymbol{D}_{\ell,i} = (\cdots \mid$	$y^b_{\ell,i}$	0	0	$\mid (\tau_{\ell,j} t_{i,j}$	0	$)_j \mid 0 \;)$
$\mathbf{G}_1$	$\forall \ell \in \mathcal{L}_1,$	$\boldsymbol{C}_{\ell,i} = (\cdots \mid$	0	1	0	$\mid (\sigma_{\ell,j} s_{i,j}$	0	$)_j \mid 0 \;)$
Product-Preserving		$\boldsymbol{D}_{\ell,i} = (\cdots \mid$	0	$x^b_{\ell,i} y^b_{\ell,i}$	0	$\mid (\tau_{\ell,j} t_{i,j}$	0	$)_j \mid 0 \;)$
$\mathbf{G}_2$	$\forall \ell \in \mathcal{L}_1,$	$\boldsymbol{C}_{\ell,i} = (\cdots \mid$	0	1	0	$\mid (\; 0$	1	$)_j \mid 0 \;)$
Product-Preserving		$\boldsymbol{D}_{\ell,i} = (\cdots \mid$	0	$x^b_{\ell,i} y^b_{\ell,i}$	0	$\mid (\; 0$	$\sigma_{\ell,j}\tau_{\ell,j} s_{i,j} t_{i,j}$	$)_j \mid 0 \;)$
$\mathbf{G}_3$	$\forall \ell \in \mathcal{L}_1,$	$\boldsymbol{C}_{\ell,i} = (\cdots \mid$	0	1	0	$\mid (\; 0$	1	$)_j \mid 0 \;)$
Formal Renaming		$\boldsymbol{D}_{\ell,i} = (\cdots \mid$	0	$z^b_{\ell,i}$	0	$\mid (\; 0$	$\rho_{\ell,j} s_{i,j} t_{i,j}$	$)_j \mid 0 \;)$
$\mathbf{G}_4$	$\forall \ell \in \mathcal{L}_1,$	$\boldsymbol{C}_{\ell,i} = (\cdots \mid$	0	1	0	$\mid (\; 0$	1	$)_j \mid 0 \;)$
Random in Span		$\boldsymbol{D}_{\ell,i} = (\cdots \mid$	0	$z^b_{\ell,i}$	0	$\mid (\; 0$	$\rho_\ell v_{1,j} s_{i,j} t_{i,j}$	$)_j \mid 0 \;)$
$\mathbf{G}_5$	$\forall \ell \in \mathcal{L}_1,$	$\boldsymbol{C}_{\ell,i} = (\cdots \mid$	0	1	0	$\mid (\; 0$	1	$)_j \mid 0 \;)$
Swapping		$\boldsymbol{D}_{\ell,i} = (\cdots \mid$	0	$c_{\ell,i}$	0	$\mid (\; 0$	0	$)_j \mid 0 \;)$
$\mathbf{G}_6$	$\forall \ell \in \mathcal{L}_1,$	$\boldsymbol{C}_{\ell,i} = (\cdots \mid$	0	1	0	$\mid (\; 0$	0	$)_j \mid 0 \;)$
Replication		$\boldsymbol{D}_{\ell,i} = (\cdots \mid$	0	$c_{\ell,i}$	0	$\mid (\; 0$	0	$)_j \mid 0 \;)$
	$\forall \ell \in \mathcal{L}_2,$	$\boldsymbol{C}_{\ell,i} = (\cdots \mid$	$x^b_{\ell,i}$	0	0	$\mid (\sigma_{\ell,j} s_{i,j}$	0	$)_j \mid 0 \;)$
		$\boldsymbol{D}_{\ell,i} = (\cdots \mid$	$y^b_{\ell,i}$	0	0	$\mid (\tau_{\ell,j} t_{i,j}$	0	$)_j \mid 0 \;)$
$\mathbf{G}_7$	$\forall \ell \in \mathcal{L}_1,$	$\boldsymbol{C}_{\ell,i} = (\cdots \mid$	0	1	0	$\mid (\; 0$	0	$)_j \mid 0 \;)$
Idem for $\mathcal{L}_2$		$\boldsymbol{D}_{\ell,i} = (\cdots \mid$	0	$c_{\ell,i}$	0	$\mid (\; 0$	0	$)_j \mid 0 \;)$
	$\forall \ell \in \mathcal{L}_2,$	$\boldsymbol{C}_{\ell,i} = (\cdots \mid$	0	0	$c_{\ell,i}$	$\mid (\; 0$	0	$)_j \mid 0 \;)$
		$\boldsymbol{D}_{\ell,i} = (\cdots \mid$	0	0	1	$\mid (\; 0$	0	$)_j \mid 0 \;)$
$\mathbf{G}_8$	$\forall \ell \in \mathcal{L}_1,$	$\boldsymbol{C}_{\ell,i} = (\cdots \mid$	0	1	0	$\mid (\; 0$	0	$)_j \mid 0 \;)$
$b = 0$		$\boldsymbol{D}_{\ell,i} = (\cdots \mid$	0	$c'_{\ell,i}$	0	$\mid (\; 0$	0	$)_j \mid 0 \;)$
	$\forall \ell \in \mathcal{L}_2,$	$\boldsymbol{C}_{\ell,i} = (\cdots \mid$	0	0	$c'_{\ell,i}$	$\mid (\; 0$	0	$)_j \mid 0 \;)$
		$\boldsymbol{D}_{\ell,i} = (\cdots \mid$	0	0	1	$\mid (\; 0$	0	$)_j \mid 0 \;)$

The 2 first components of the $\boldsymbol{C}_{\ell,i}$ and $\boldsymbol{D}_{\ell,i}$ are always $\pi_{\ell,i} \cdot (1, \lambda_{\ell,i})$ and $\mu_{\ell,i} \cdot (\lambda_{\ell,i}, -1)$.
For $\ell \in \mathcal{L}_k : c_{\ell,i} = z^b_{\ell,i} + \rho_\ell \sum_j v_{k,j} s_{i,j} t_{i,j}$, and $c'_{\ell,i} = z^0_{\ell,i} + \rho_\ell \sum_j v_{k,j} s_{i,j} t_{i,j}$.
$\boldsymbol{C}_{\ell,0} = \boldsymbol{\sigma}_\ell \cdot G_1$, $\boldsymbol{D}_{\ell,0} = \boldsymbol{\tau}_\ell \cdot G_2$, and $\boldsymbol{\rho}_\ell = (\sigma_{\ell,j}\tau_{\ell,j})_j$, where

- From $\mathbf{G}_0$ to $\mathbf{G}_3$, $\boldsymbol{\sigma}_\ell, \boldsymbol{\tau}_\ell \overset{\$}{\leftarrow} \mathbb{Z}_q^3$;
- From $\mathbf{G}_4$, for $\ell \in \mathcal{L}_1$, $\boldsymbol{\sigma}_\ell \overset{\$}{\leftarrow} (\mathbb{Z}_q^*)^3$, $\boldsymbol{\rho}_\ell \leftarrow \rho_\ell \cdot \boldsymbol{v}_1$, for $\rho_\ell \overset{\$}{\leftarrow} \mathbb{Z}_q^*$, $\boldsymbol{\tau}_\ell \leftarrow \rho_\ell \cdot (v_{1,j}/\sigma_{\ell,j})_j$;
- From $\mathbf{G}_7$, for $\ell \in \mathcal{L}_2$, $\boldsymbol{\tau}_\ell \overset{\$}{\leftarrow} (\mathbb{Z}_q^*)^3$, $\boldsymbol{\rho}_\ell \leftarrow \rho_\ell \cdot \boldsymbol{v}_2$, for $\rho_\ell \overset{\$}{\leftarrow} \mathbb{Z}_q^*$, $\boldsymbol{\sigma}_\ell \leftarrow \sigma_\ell \cdot (v_{2,j}/\tau_{\ell,j})_j$.

Fig. 2. Sequence of Games for the Indistinguishability Security

asked to $\mathcal{H}$, anticipating Qn possible indices. K denotes the number of $\mathsf{OKeyGen}$-queries. lls highlight the modified values, and the various transformations of bases are precised in the left column. The full proof is provided in appendix B. Note that under the collision-resistance of $\mathcal{H}$, all the $\lambda_{\ell,i}$ are also distinct.

5 Inner-Product Functional Encryption with Selector

The above 2-client IPFE works for all possible inputs from both data providers. However, one may be interested in a very specific sub-case where the second data provider's vector is actually a selector with half of its coefficients equal to one, the other half being zero, i.e., DP_1's input is $\boldsymbol{x} = \boldsymbol{x}_0||\boldsymbol{x}_1$, DP_2's input $\boldsymbol{y} = (1-b)^n||b^n$, with $b \in \{0;1\}$, and the functional coefficients are $\boldsymbol{\alpha} = \boldsymbol{\alpha}_0||\boldsymbol{\alpha}_1$, so that the decryption key $\mathsf{dk}_{\boldsymbol{\alpha}}$ helps to decrypt $\boldsymbol{x}_b \cdot \boldsymbol{\alpha}_b^\top$. For such a particular case, the above scheme would lead to a long ciphertext for $\boldsymbol{y}$, whereas it just contains a bit b. We thus propose a more compact scheme, with a dedicated security result.

5.1 Description

Let us first see how the above general scheme can be improved in the specific case of IPFE with a selector:

$\mathsf{Setup}(1^\kappa)$**.** From a pairing-friendly setting $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_t, e, G_1, G_2, q)$ of appropriate size with respect to the security parameter κ, the function chooses random dual orthogonal matrices $B = (\boldsymbol{b}_i)_i \overset{\$}{\leftarrow} \mathrm{GL}_9(\mathbb{Z}_q)$ and $B^* = (\boldsymbol{b}_i^*)_i \leftarrow (B^{-1})^\top$, which define the bases $(\mathbb{B}, \mathbb{B}^*)$. One generates secret vectors $\underline{\mathbf{S}} = (\boldsymbol{s}_i, \boldsymbol{s}'_i)_{i=1}^n \overset{\$}{\leftarrow} ((\mathbb{Z}_q^2)^2)^n$ and $\underline{\mathbf{T}} = (\boldsymbol{t}, \boldsymbol{t}') \overset{\$}{\leftarrow} (\mathbb{Z}_q^2)^2$. The encryption key ek_1 of DP_1 is set to $(B, \underline{\mathbf{S}})$, the encryption key ek_2 of DP_2 is set to $(B^*, \underline{\mathbf{T}})$, and the secret key is set to $\mathsf{sk} = (\underline{\mathbf{S}}, \underline{\mathbf{T}})$. The public parameters are just $\mathsf{PK} \leftarrow (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_t, e, G_1, G_2, q)$, together with two collision-resistant hash functions $\mathcal{H}, \mathcal{H}'$ into $\mathbb{Z}_q$;

$\mathsf{KeyGen}(\mathsf{sk}, \boldsymbol{\alpha})$**.** For a vector $\boldsymbol{\alpha} = (\boldsymbol{\alpha}_0, \boldsymbol{\alpha}_1) \in (\mathbb{Z}_q^n)^2$, DP_1 and DP_2 generate the functional decryption key

$$\mathsf{dk}_{\boldsymbol{\alpha}} \leftarrow \boldsymbol{d} = \sum_{i=1}^n \alpha_{0,i}(s_{i,1}t_1, s_{i,2}t_2) + \alpha_{1,i}(s'_{i,1}t'_1, s'_{i,2}t'_2) \in \mathbb{Z}_q^2$$

$\mathsf{Encrypt}(\ell, \mathsf{ek}, \boldsymbol{z})$**,** for a label ℓ, where ek is either ek_1 or ek_2, and $\boldsymbol{z}$ is either $\boldsymbol{x} = (\boldsymbol{x}_0, \boldsymbol{x}_1) \in (\mathbb{Z}_q^n)^2$ or b, one derives $\lambda_1 = \mathcal{H}(\ell), \lambda_2 = \mathcal{H}'(\ell) \in \mathbb{Z}_q$:

- To encrypt $\boldsymbol{x} = (\boldsymbol{x}_0, \boldsymbol{x}_1) \in (\mathbb{Z}_q^n)^2$ with $\mathsf{ek}_1 = (B, \underline{\mathbf{S}})$, one generates $\boldsymbol{\sigma} \overset{\$}{\leftarrow} (\mathbb{Z}_q^*)^2$, random $\pi_i, \pi'_i \overset{\$}{\leftarrow} \mathbb{Z}_q$, and returns $\mathbf{C} \leftarrow (\ell, \boldsymbol{C}_0, (\boldsymbol{C}_i, \boldsymbol{C}'_i)_i)$, where $C_{0,j} = \sigma_j \cdot G_1$, for $j = 1,2$, and for $i \in [\![1;n]\!]$:

$$\begin{aligned}\boldsymbol{C}_i &= (\pi_i(\boldsymbol{b}_1 + \lambda_1 \cdot \boldsymbol{b}_2) + x_{0,i} \cdot \boldsymbol{b}_3 + \sigma_1 s_{i,1} \cdot \boldsymbol{b}_5 + \sigma_2 s_{i,2} \cdot \boldsymbol{b}_7) \cdot G_1 \\ &= (\pi_i \cdot (1, \lambda_1), x_{0,i}, 0, \sigma_1 s_{i,1}, 0, \sigma_2 s_{i,2}, 0, 0)_{\mathbb{B}} \\ &= (\pi_i \cdot (1, \lambda_1), x_{0,i}, 0, (\sigma_j s_{i,j}, 0)_j, 0)_{\mathbb{B}} \\ \boldsymbol{C}'_i &= (\pi'_i \cdot (1, \lambda_2), x_{1,i}, 0, (\sigma_j s'_{i,j}, 0)_j, 0)_{\mathbb{B}}\end{aligned}$$

- To encrypt $b \in \{0,1\}$ with $\mathsf{ek}_2 = (B^*, \underline{\mathbf{T}})$, one generates $\boldsymbol{\tau} \overset{\$}{\leftarrow} \mathbb{Z}_q^2$, random $\mu_1, \mu_2 \overset{\$}{\leftarrow} \mathbb{Z}_q$, and returns $\mathbf{D} \leftarrow (\ell, \boldsymbol{D}_0, \boldsymbol{D}_1, \boldsymbol{D}_2)$, where $D_{0,j} = \tau_j \cdot G_2$, for $j = 1,2$, and:

$$\begin{aligned}\boldsymbol{D}_1 &= (\mu_1(\lambda_1 \boldsymbol{b}_1^* - \boldsymbol{b}_2^*) + (1-b) \cdot \boldsymbol{b}_3^* + \tau_1 t_1 \cdot \boldsymbol{b}_5^* + \tau_2 t_2 \cdot \boldsymbol{b}_7^*) \cdot G_2 \\ &= (\mu_1 \cdot (\lambda_1, -1), 1-b, 0, \tau_1 t_1, 0, \tau_2 t_2, 0, 0)_{\mathbb{B}^*} \\ &= (\mu_1 \cdot (\lambda_1, -1), 1-b, 0, (\tau_j t_j, 0)_j, 0)_{\mathbb{B}^*} \\ \boldsymbol{D}_2 &= (\mu_2 \cdot (\lambda_2, -1), b, 0, (\tau_j t'_j, 0)_j, 0)_{\mathbb{B}^*}\end{aligned}$$

$\mathsf{Decrypt}(\mathsf{dk}_{\boldsymbol{\alpha}}, \mathbf{C}, \mathbf{D})$. Due to the orthogonality of the bases $\mathbb{B}$ and $\mathbb{B}^*$, with $g_t = e(G_1, G_2)$, for $i \in [\![1; n]\!]$ one has

$$\begin{aligned}\boldsymbol{C}_i \times \boldsymbol{D}_1 &= [\pi_i \mu_1(\lambda_1^* - \lambda_1) + (1-b)x_{0,i} + \sigma_1\tau_1 s_{i,1}t_1 + \sigma_2\tau_2 s_{i,2}t_2]_{\mathbb{G}_t} \\ \boldsymbol{C}'_i \times \boldsymbol{D}_2 &= [\pi'_i \mu_2(\lambda_2^* - \lambda_2) + bx_{1,i} + \sigma_1\tau_1 s'_{i,1}t'_1 + \sigma_2\tau_2 s'_{i,2}t'_2]_{\mathbb{G}_t}\end{aligned}$$

If the labels are the same, $\boldsymbol{C}_i \times \boldsymbol{D}_1 = g_t^{(1-b)x_{0,i}} \cdot \prod_j (g_t^{s_{i,j}t_j})^{\sigma_j\tau_j}$ and $\boldsymbol{C}'_i \times \boldsymbol{D}_2 = g_t^{bx_{1,i}} \cdot \prod_j (g_t^{s'_{i,j}t'_j})^{\sigma_j\tau_j}$. Then,

$$\begin{aligned}&\prod_{i=1}^{n} (\boldsymbol{C}_i \times \boldsymbol{D}_1)^{\alpha_{0,i}} \times (\boldsymbol{C}'_i \times \boldsymbol{D}_2)^{\alpha_{1,i}} \\ &= \prod_{i=1}^{n} g_t^{(1-b)\alpha_{0,i}x_{0,i}} \cdot \prod_j (g_t^{\alpha_{0,i}s_{i,j}t_j})^{\sigma_j\tau_j} \cdot g_t^{b\alpha_{1,i}x_{1,i}} \cdot \prod_j (g_t^{\alpha_{1,i}s'_{i,j}t'_j})^{\sigma_j\tau_j} \\ &= g_t^{\boldsymbol{\alpha}_b \cdot \boldsymbol{x}_b^\top} \cdot \prod_j (g_t^{d_j})^{\sigma_j\tau_j} = g_t^{\boldsymbol{\alpha}_b \cdot \boldsymbol{x}_b^\top} \cdot \prod_j e(C_{0,j}, D_{0,j})^{d_j}\end{aligned}$$

One can thus get $g_t^{\boldsymbol{\alpha}_b \cdot \boldsymbol{x}_b^\top}$ and extract the discrete logarithm $\boldsymbol{\alpha}_b \cdot \boldsymbol{x}_b^\top$.

This scheme drastically reduces second data provider's ciphertext sizes, from being linear in n to a small constant size.

5.2 Security Analysis

Theorem 12. *Our above construction from Section 5.1 satisfies* ord-IND *indistinguishability against adaptive adversaries in time t with:*

$$\mathsf{Adv}^{\mathsf{ind}}(\mathcal{A}) \leq (2N \cdot (12Qn(N+1) + 25) + 4) \cdot \mathsf{Adv}^{\mathsf{sxdh}}(t) + 3N/q$$

where N is the number of different labels the adversary got encryptions for, and Q the number of labels queried to the hash function.

We stress that in the ord-IND indistinguishability we assume $\mathsf{LoREncrypt}_2$ is always asked first. The full proof is provided in appendix C.

6 Privacy-Preserving Money-Laundering Detection

For money-laundering detection, based on the work of [SNY⁺16] that presented a method for this detection on cleartexts with no privacy, which we now provide, we consider the graph where the vertices are all the bank accounts, and oriented edges represent transactions between them, with a weight corresponding to the amount of money sent. These can be considered at regular time periods. We can apply our above two-client inner-product functional encryption scheme to calculate the similarity of two nodes u and v belonging to distinct banks, that are then the two data providers DP_1 and DP_2 respectively, over a time-period. One wants to compute:

$$\begin{aligned}&\sigma_{\mathsf{in}}(u,v) = \langle \underline{\boldsymbol{w}}_{u,\mathsf{in}}, \underline{\boldsymbol{w}}_{v,\mathsf{in}} \rangle \qquad\qquad \sigma_{\mathsf{out}}(u,v) = \langle \underline{\boldsymbol{w}}_{u,\mathsf{out}}, \underline{\boldsymbol{w}}_{v,\mathsf{out}} \rangle \\ &\sigma(u,v) = \sigma_{\mathsf{in}}(u,v) \times \sigma_{\mathsf{out}}(u,v)\end{aligned}$$

where the vectors $\underline{\boldsymbol{w}}_{u,\mathsf{in}}$ and $\underline{\boldsymbol{w}}_{u,\mathsf{out}}$ are normalized N-vectors that can be generated by DP_1, and $\underline{\boldsymbol{w}}_{v,\mathsf{in}}$ and $\underline{\boldsymbol{w}}_{v,\mathsf{out}}$ can be generated by DP_2. Indeed, using the notations introduced in Section 1, $\underline{\boldsymbol{w}}_{u,\mathsf{in}}$ is the normalized vector of dimension N, of all the incoming transactions, where N is the total number of possible account numbers, and similarly for the other vectors. One may only focus on the computation of $\langle \underline{\boldsymbol{w}}_{u,\mathsf{in}}, \underline{\boldsymbol{w}}_{v,\mathsf{in}} \rangle$, as the same can be done on $\langle \underline{\boldsymbol{w}}_{u,\mathsf{out}}, \underline{\boldsymbol{w}}_{v,\mathsf{out}} \rangle$.

While we want all the inner products between all the nodes, our privacy-preserving technique only allows to combine one vector with a unique other vector, both encrypted under the same label. For each pair of nodes (u, v), one thus defines a label $\lambda_{u,v}$, then DP_1 and DP_2 encrypt $\tilde{\underline{\boldsymbol{w}}}_{u,\mathsf{in}}$ and $\tilde{\underline{\boldsymbol{w}}}_{v,\mathsf{in}}$ respectively under $\lambda_{u,v}$, and the DC can compute the inner-product with the functional decryption key for the Identity matrix. This might look costly, because of the quadratic number of ciphertexts. On the other hand, providing N^2 inner products between N vectors would leak too much information on the inputs, whatever the protocol. One can thus exploit these multiple encryptions of the same vectors to add independent small noise to non-zero components, in the same vein as differential privacy. The level of noise to be introduced is out of the scope of this analysis, but this illustrates the main advantage of labels, and thus of the use of MCFE instead of MIFE, that would not allow the additional protection.

6.1 Communication Complexity

Let us evaluate the communication complexity for the banks and the regulation organisation, when DP_1 manages N_1 accounts and DP_2 manages N_2, among a global number of accounts N, as incoming/outgoing transactions can be with other banks. Our security model indeed requires making ciphertexts for a given label (u, v) to allow combination for the inner-product evaluation. As such, the vector of weights for all incoming and outgoing transactions for each node will have to be encoded differently for each similarity calculation with a new node, meaning DP_1 would have to send $2N_1N_2$ ciphertexts for each vertex u (because both its in and out weight vectors have N components) of their N_1 big set to DC for a comparison with DP_2's nodes. Symmetrically, DP_2 would have to send $2N_1N_2$ N-long vectors of ciphertexts. In the end, for the comparisons between DP_1 and DP_2, DC receives $4NN_1N_2$ ciphertexts.

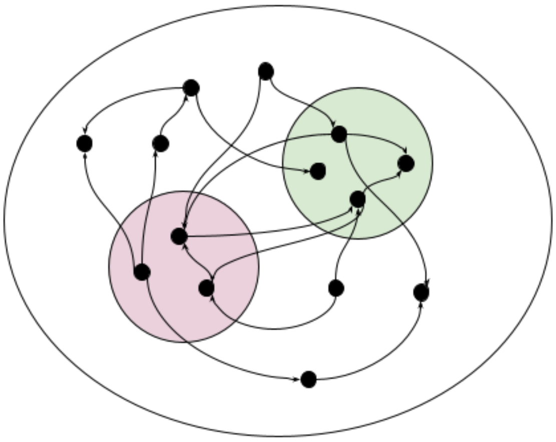

Fig. 3. Illustration of banks DP_1 and DP_2 owning some nodes of the graph of all accounts

We hereafter discuss optimizations to reduce these communication costs when there are many bank accounts and weight vectors are sparse. Another method to reduce these communications can be for each bank to preselect accounts, for instance with a threshold on the amount of transactions transiting through them. Other methods such as the complementary formula growing with a node's amount of transactions and similarity between the amount in its incoming and outgoing transactions presented in [SNY+16] can also help for this preselection of vertices of interest. The banks should then decide for themselves to which extent it is acceptable to let such a selection of accounts leak to other banks collaborating in the comparison.

Our non-optimized version of the protocol is best for small set sizes or graphs with a big connectivity, as is shown in figure 5, in which we reported [SNY$^+$16]'s parameter sets for N and n, the number of accounts in the system and their connectivity respectively, and took $N_1 = N_2 = N_i$, the individual bank sizes, to be a fifth of N. The graph hashing method results in a smaller communication size for their last two parameter sets.

6.2 Reduced Graph with Hashing

Intuition Making similarity calculations in a large graph is by definition an application where the number of comparisons has a quadratic complexity in the number of tested nodes. In money laundering detection, where the graphs considered can be huge, this can become humongous, even though bank infrastructures are considerable. Either a first filtering of nodes of interest can be made, to get reasonable communications, or else we also suggest the following optimization when there are many nodes with a small connectivity (mean number of neighbors a vertex has). The main idea here is to work on much smaller graphs than the original one, even though that means having to consider several ones. To do this we hash the nodes into a smaller space, adding some of them together in common nodes, and perform the similarity calculations on the added weight vectors of nodes landing on the same hash values. We then select nodes where the similarity is in the suspicious range $[\theta; 1]$ for at least τ of the hashed graphs, relying on the weight vectors' sparseness. In the following section we detail the probability calculations we made from the supposed sparseness of weight vectors, to deduce the appropriate size of the set nodes should be hashed into, and the corresponding number of hash functions necessary to make the detection accurate.

Building on the idea exposed in presentation [GGJ$^+$20], similar to Bloom filter techniques, we take r hash functions $H_1, \ldots, H_r$ from $[\![1; N]\!]$ to $[\![1; 2^m]\!]$, and for each function H_s we build the corresponding reduced vertices set $V_s = \{H_s(v); v \in V\}$ from the original N-long set of vertices V, and each vertex v in this new set gets, for the incoming transactions, the sum of the normalized weight vectors of incoming transactions of all the V vertices that H_s projected onto v, written $\boldsymbol{w}_{s,v,\mathsf{in}}$. The same goes for the out weight vectors $\boldsymbol{w}_{s,v,\mathsf{out}}$.

In this version, one makes r computations, one for each hash function reduction, to evaluate to vertices' similarity. In each computation, vectors are still of size N, and if a data provider (a bank) DP_1 wants to compare all their accounts with another bank DP_2's, they then need to send DC $2r2^{2m}N$ ciphertexts, and DP_2 $2r2^{2m}N$, so DC should receive $r2^{2(m+1)}N$ ciphertexts (of constant size, since this is for each component). This thus has to be compared with $4NN_1N_2$ to select the best method.

Parameter calculations We build the following probability model on the bank account data used, in order to set appropriate parameters for the graph hashing technique. We model weight vectors' sparseness with Bernoulli variables, and make sure that for a given inner-product in the original graph, at least σ of the inner-products in the hashed graphs will be with added vectors hashed to the same node that have disjoint supports, so that they do not influence the resulting inner-products. We will select pairs of nodes for which at least $\tau \geq \sigma$ of the inner-products on hashed graphs gave a result in the detection range $[\theta; 1]$ from [SNY$^+$16]. For σ, we take at least half of the number of hash functions to avoid false positives.

Let $(\Omega, \mathcal{A}, P)$ be a probability space, and, to model weight vectors'sparseness,

$$\begin{aligned} X_{v,i} :& \Omega \to \{0; 1\}, \\ & E \mapsto 1 - \delta_{\underline{\boldsymbol{w}}_{v,i},0}, \end{aligned}$$

be the random variable taking value 1 if the i-th component of $\underline{\boldsymbol{w}}_v$ from the original graph is non-zero, else 0. $X_{v,i}$ follows a Bernoulli distribution $B\left(\frac{n}{N}\right)$, where n can be seen as the connectivity of the graph (*i.e.* the number of neighbors for each node) for in or out transactions.

Our optimization will take advantage of having $n \ll N$ in our use-case. It will also exploit the random choice of the hash functions that combine totally independent nodes from the original graph. Defining the sum $X_v = \sum_{k=1}^{N} X_{v,k}$, following the binomial distribution $\mathcal{B}\left(N, \frac{n}{N}\right)$ (as the $X_{v,k}$ are supposed to be independent). Let us set:

$$Z_{v,\mathsf{in},i} : \Omega \to [0;1], E \mapsto \underline{\boldsymbol{w}}_{v,i}$$

(resp. $Z_{v,\mathsf{out},i}$) as the random variable which for $v \in V$ takes the value of $\underline{\boldsymbol{w}}_{v,\mathsf{in}}$ (resp. $\underline{\boldsymbol{w}}_{v,\mathsf{out}}$)'s i-th component, following a law Z. This variable also models weight vectors'sparseness, but instead of taking 0 or 1 values to get the weight vectors' supports, it more precisely takes the transaction amounts inside the weight vectors.

Let:

$$\begin{aligned} Y_{s,v} :&\Omega \to [\![0; N-1]\!], \\ &E \mapsto \sum_{\substack{u \in V \\ u \neq v}} \delta_{H_s(u), H_s(v)} \end{aligned}$$

be the random variable which gives, for a vertex $v \in V$, the number of other vertices from V which H_s maps to the same node as v. $Y_{s,v} \xleftarrow{\$} \mathcal{B}\left(N-1, \frac{1}{2^m}\right)$, and is assumed independent of other $Y_{s',v'}$, and from the X and Z above random variables. Let:

$$X_{s,\bar{v},\mathsf{in},i} : \Omega \to \{0;1\}, E \mapsto 1 - \delta_{\sum_{\substack{H_s(x)=H_s(v) \\ x \neq v}} \underline{\boldsymbol{w}}_{x,\mathsf{in},i}, 0}$$

(or respectively $X_{s,\bar{v},\mathsf{in},i}$) be the boolean random variable which tells whether the i-th component of the sum of the weight vectors added to v's weight vector in hashed graph s, $\sum_{\substack{H_s(x)=H_s(v) \\ x \neq v}} \underline{\boldsymbol{w}}_x$, is null or not. For them not to perturb the inner-product calculation, we will want them to be null on components with transactions on our two original vectors of interest $\underline{\boldsymbol{w}}_a$ and $\underline{\boldsymbol{w}}_b$, and on the other components, we will want them to be null on one or the other of the destination vectors. We write $X_{s,\bar{v},\mathsf{in}} = \sum_{i=1}^{N} X_{s,\bar{v},\mathsf{in},i}$ (and with out respectively).

For $a, b \in V$, and $s \in [\![1; r]\!]$ such that $H_s(a) = u \neq v = H_s(b)$,

$$\begin{aligned} \langle \underline{\boldsymbol{w}}_{a,\mathsf{in}} + \sum_{\substack{H_s(x)=u \\ x \neq a}} \underline{\boldsymbol{w}}_{x,\mathsf{in}}; \underline{\boldsymbol{w}}_{b,\mathsf{in}} + \sum_{\substack{H_s(x)=v \\ x \neq b}} \underline{\boldsymbol{w}}_{x,\mathsf{in}} \rangle = \langle \underline{\boldsymbol{w}}_{a,\mathsf{in}}; \underline{\boldsymbol{w}}_{b,\mathsf{in}} \rangle + \langle \underline{\boldsymbol{w}}_{a,\mathsf{in}}; \sum_{\substack{H_s(x)=v \\ x \neq b}} \underline{\boldsymbol{w}}_{x,\mathsf{in}} \rangle + \langle \underline{\boldsymbol{w}}_{b,\mathsf{in}}; \sum_{\substack{H_s(x)=u \\ x \neq a}} \underline{\boldsymbol{w}}_{x,\mathsf{in}} \rangle \\ + \langle \sum_{\substack{H_s(x)=u \\ x \neq a}} \underline{\boldsymbol{w}}_{x,\mathsf{in}}; \sum_{\substack{H_s(x)=v \\ x \neq b}} \underline{\boldsymbol{w}}_{x,\mathsf{in}} \rangle. \end{aligned}$$

Let $Z_{s,v,\mathsf{in}} : \Omega \to \mathbb{Z}_q^N, E \mapsto \sum_{H_s(x)=H_s(v), x \neq v} r_x \cdot \underline{\boldsymbol{w}}_{x,\mathsf{in}}$ (respectively $Z_{s,v,\mathsf{out}} : \Omega \to \mathbb{Z}_q^N$, $E \mapsto \sum_{H_s(x)=H_s(v), x \neq v} r_x^{-1} \cdot \underline{\boldsymbol{w}}_{x,\mathsf{out}}$), for all $v \in V$, be the random variable, which, for $a \in V$, gives the vector of normalized weights from other nodes in the original graph added with v's normalized weight vector.

It is a sum of $Y_{s,v}$ random variables of the original weight vectors following the distribution Z. As such, the probability that a component of $Z_{s,v,\mathsf{in}}$ (or $Z_{s,v,\mathsf{out}}$) is equal to 0 is

$$p_z = P(Z_{s,v,\mathsf{in},i} = 0) = \left(1 - \frac{n}{N}\right)^{\frac{N}{2^m} - 1}.$$

The expected value of $X_{s,\bar{v},\mathsf{in}}$ is bounded by $\frac{nN}{2^m}$, and is equal to $N\left(1 - \left(1 - \frac{n}{N}\right)^{\frac{N}{2^m}-1}\right)$, assuming its components are independent in our model. The same goes for out vectors of course. Also, the probability that $Z_{s,a,\mathsf{in}}$ has a separate support from the one of $Z_{s,b,\mathsf{in}}$, meaning

$\langle\sum_{\substack{H_s(x)=u\\x\neq a}} \underline{\boldsymbol{w}}_{x,\mathsf{in}};\sum_{\substack{H_s(x)=v\\x\neq b}} \underline{\boldsymbol{w}}_{x,\mathsf{in}}\rangle$ would be null, is:

$$\begin{aligned}P(\forall i\in[\![1;N]\!], z_{s,a,\mathsf{in},i}z_{s,b,\mathsf{in},i}=0|u\neq v) &= (2p_z - P((z_{s,a,\mathsf{in},i}=0)\cap(z_{s,b,\mathsf{in},i}=0)|u\neq v))^N\\ &= \left(\left(1-\frac{n}{N}\right)^{\frac{N}{2^m}-1}\left(2-\left(1-\frac{n}{N}\right)^{\frac{N}{2^m}-1}\right)\right)^N\end{aligned}$$

Similarly, The probability that $Z_{s,a,\mathsf{in}}$ and $Z_{s,b,\mathsf{in}}$ have disjoint supports, but also $Z_{s,a,\mathsf{in}}$ and $\underline{\boldsymbol{w}}_{b,\mathsf{in}}$, and $Z_{s,b,\mathsf{in}}$ and $\underline{\boldsymbol{w}}_{a,\mathsf{in}}$, resulting in null $\langle\underline{\boldsymbol{w}}_{a,\mathsf{in}};\sum_{\substack{H_s(x)=v\\x\neq b}} \underline{\boldsymbol{w}}_{x,\mathsf{in}}\rangle$ and $\langle\underline{\boldsymbol{w}}_{b,\mathsf{in}};\sum_{\substack{H_s(x)=u\\x\neq a}} \underline{\boldsymbol{w}}_{x,\mathsf{in}}\rangle$, which we will call event A_{in}, is, writing $q=1-\frac{n}{N}$, $q'=1-\frac{n'}{N}$, supposing vectors $\underline{\boldsymbol{w}}_{a,\mathsf{in}}$ and $\underline{\boldsymbol{w}}_{b,\mathsf{in}}$ each have n' non-zero components, with uniformly distributed indices:

$$\begin{aligned}P(A_{\mathsf{in}}|u\neq v) &= P(\forall i\in[\![1;N]\!],(z_{s,a,\mathsf{in},i}z_{s,b,\mathsf{in},i}=0)\cap(\underline{w}_{b,\mathsf{in},i}z_{s,a,\mathsf{in},i}=0)\cap(\underline{w}_{a,\mathsf{in},i}z_{s,b,\mathsf{in},i}=0)|u\neq v)\\ &= \left(q^{\frac{N}{2^m}-1}\left(2-q^{\frac{N}{2^m}-1}\right)\left(q'+q^{\frac{N}{2^m}-1}-q'q^{\frac{N}{2^m}-1}\right)^2\right)^N\end{aligned}$$

again with our independence assumptions.

In the case where $H_s(a)=H_s(b)$, which happens with probability 2^{-m}, then $Z_{s,a,\mathsf{in}}$ and $Z_{s,b,\mathsf{in}}$ will be very similar and far from being independent and in fact, as $\underline{\boldsymbol{w}}_a$ is summed in $Z_{s,b,\mathsf{in}}$, and $\underline{\boldsymbol{w}}_b$ in $Z_{s,a,\mathsf{in}}$, and these are vectors with norm 1, A can never happen. So:

$$P(A_{\mathsf{in}})=(1-2^{-m})\left(q^{\frac{N}{2^m}-1}\left(2-q^{\frac{N}{2^m}-1}\right)\left(q'+q^{\frac{N}{2^m}-1}-q'q^{\frac{N}{2^m}-1}\right)^2\right)^N$$

This gives the expected proportion of the r hash functions for which all the vectors hashed to the same value as our two vectors of interest in the original graph are cancelled out in the inner product for a given hash function. Analogously, $P(A_{\mathsf{out}})=P(A_{\mathsf{in}})=p_A$. We will be interested in having A_{in} and A_{out} happen for the same hash functions, and approximating in and out vectors' weights as independent, and calling B the event where both A_{in} and A_{out} happen for a given hash function, $p_B=P(B)=P(A_{\mathsf{in}})P(A_{\mathsf{out}})=p_A^2$.

Relating this with the birthday paradox, with the more restrictive expectation that none of the $\frac{nN}{2^m}$ draws for all the non-null coordinates of nodes hashed to the same bucket (and approximating one vector's non-null components will not land in the same place), whereas in our case we do not care about some vector components being added together if they get cancelled out with a null component in the inner-product, we give the following lower bound on p_A:

$$p_A\geq e^{-\frac{n(nN-2^m)}{2^{m+1}}}.$$

So if we want to have $p_B=p_A^2\geq\frac{1}{2}$, we can thus take: $m\geq\log_2(N)+2\log_2(n)-\log_2(n+2\ln(2))$, and we will take m just above that limit. With $n=0.14$ and $N=10000$, we can thus take $m=8$. We choose values for m using the birthday paradox approximation and then calculate the exact resulting probability p_A and p_B, with results shown in figure 5. Considering the binomial random variable with probability p_B, and r (the number of hash functions) repetitions, the probability of having at least $\sigma\geq 2$ successes (the outcome B) out of the r hash functions (the expected number of successes being rp_B), which we will write $1-F(\sigma-1,r,p_B)$, where F is the cumulative distribution function of the binomial law, is greater than the following Chernoff bound, for $\sigma\leq rp_B+1$:

$$1-F(\sigma-1,r,p_B)\geq 1-\left(\frac{rp_B}{\sigma-1}\right)^{\sigma-1}e^{\sigma-1-rp_B}\qquad(1)$$

Let c be our lower bound on the confidence interval of having at least σ successes B for one of the r hash functions. We choose the minimal r such that the right member of inequality 1 is greater than c, taking $\sigma = \lceil r/2 \rceil$, meaning we want to be sure to have relevant result on at half of the hash functions, as we will take the majority output and want to avoid having false positives. We show the results in figure 5 (whose Python code is provided in appendix E) for $\sigma = \lceil r/2 \rceil$, using N and n parameters from [SNY+16]. $\#C_0$ is the number of ciphertexts to be sent without optimizations and $\#C_H$ with these optimizations.

n	N	m	p_A	p_B	N_i	r	c	$\#C_0$	$\#C_H$
0.36	10^3	7	0.98	0.96	200	18	0.96	1.6E+08	1.2E+09
0.11	10^4	7	0.98	0.97	2000	18	0.96	1.6E+11	1.2E+10
0.14	10^4	8	0.99	0.98	2000	16	0.95	1.6E+11	4.2E+10

Fig. 4. parameters and communications comparison with or without the hash functions optimization. n is the average number of other accounts one account has transactions with, N the total number of existing bank accounts, N_i the number of accounts each of the two banks controls, m is the bit size of the space each hash function maps into, p_A gives, for one graph hashing and two nodes in the original graph, the probability that their in (or out) scalar product in the hashed graph is equal to its value in the original graph, and p_B gives the probability that this both happens for their in *and* out inner-products. r is the number of hash functions required so that we have a resulting c confidence interval of having this happen for at least half of the hash functions, for a given node in the original graph, and thus of detecting similar nodes. $\#C_0$ is the number of ciphertexts in the communication without the hashing strategy, and $\#C_H$ is this communication with the hashing technique. We took input n and N parameter sets from in [SNY+16] with $N_i = N/5$.

If event B happens, then:

$$\langle \underline{\boldsymbol{w}}_{a,\mathsf{in}} + Z_{s,a,\mathsf{in}}; \underline{\boldsymbol{w}}_{b,\mathsf{in}} + Z_{s,b,\mathsf{in}} \rangle \times \langle \underline{\boldsymbol{w}}_{a,\mathsf{out}} + Z_{s,a,\mathsf{out}}; \underline{\boldsymbol{w}}_{b,\mathsf{out}} + Z_{s,b,\mathsf{out}} \rangle = \langle \underline{\boldsymbol{w}}_{a,\mathsf{in}}; \underline{\boldsymbol{w}}_{b,\mathsf{in}} \rangle \times \langle \underline{\boldsymbol{w}}_{a,\mathsf{out}}; \underline{\boldsymbol{w}}_{b,\mathsf{out}} \rangle$$

is in $[0; 1]$, and we will deem a and b suspicious if it is in $[\theta, 1]$, where θ was taken equal to 0.2 in [SNY+16].

To make sure our above theoretical method yielded good results in practice, we compared its results to those from the experiments of [SNY+16], with their parameter sets and our suggested corresponding r and m from figure 4. Because of the hypotheses made in the theoretical approach, especially the supposedly independent binomial law models, we then adapted the number of hash functions r empirically and recorded the results in figure 5. We also empirically adapted the threshold τ on the number of hash functions we need a positive result from to sort a node as positive, and show the results in the figure, using our script in appendix E. Because of the empirical variations on r, we recorded the resulting communication sizes in figure 6, to see when the graph hashing technique was interesting compared with the standard procedure.

Parameter set	1	2	3
N	1000	10000	10000
clean transactions	300	500	200
money laundering trans.	10	100	200
n	0.36	0.11	0.14
r	15	25	20
τ	14	24	19
m	7	7	8
Sensitivity without hashing	0.79	0.88	0.847
Sensitivity with hashing	0.79	0.91	0.9
Fall-out without hashing	0.06	0.002	0.0006
Fall-out with hashing	0.06	0.28	0.25
False negative rate without h.	0.21	0.12	0.13
False negative rate with h.	0.21	0.09	0.1

Fig. 5. We used a detection threshold $\theta = 0.2$ as in [SNY+16], and more generally their three parameter sets. We chose τ empirically, above the limit of $r/2$; it can be adapted depending on the desired sensitivity versus fall-out tradeoff. We also adapted r empirically. Sensitivity gives the true positive rate, fall-out the false positive rate.

n	N	m	N_i	r	$\#C_0$	$\#C_H$
0.36	10^3	7	200	15	1.6E+08	9.8E+08
0.11	10^4	7	2000	25	1.6E+11	1.6E+10
0.14	10^4	8	2000	20	1.6E+11	5.2E+10

Fig. 6. parameters and communications comparison with or without the hash functions optimization from an empirical number of hash functions r.

Our expermiments confirm that using the hashing technique is interesting on [SNY$^+$16]'s last two parameter sets, when n is small and N and N_i are big. We checked that trying to take a parameter m smaller or bigger than the one given by the birthday paradox approximation does not result in any communication complexity improvement with the resulting probability variations. We anyhow show that our parameters from figure 4 detect at least as many of the guilty money laundering accounts as the non-hash version does, as showed in figure 5, though there are more false positives.

6.3 Optimization on the number of ciphertexts for null edges would reveal information

One could remark that in the weight vectors for this use-case, many components are set to zero as each bank account will supposedly only communicate with a small portion of all existing bank accounts. One could suggest using similar techniques as in the selector instantiation of our scheme to use only one ciphertext for all these null components. However, as the other data provider in the computation would have to encode all corresponding ciphertexts with the same label, this would leak information about which accounts each one of them is communicating with, which would not work out in this use-case. If public information can cut out groups of bank accounts, the two data providers might as well agree on it together from the start to reduce vector sizes.

References

ABDP15. Michel Abdalla, Florian Bourse, Angelo De Caro, and David Pointcheval. Simple functional encryption schemes for inner products. In Jonathan Katz, editor, *PKC 2015*, volume 9020 of *LNCS*, pages 733–751. Springer, Heidelberg, March / April 2015.

AGT21. Shweta Agrawal, Rishab Goyal, and Junichi Tomida. Multi-input quadratic functional encryption from pairings. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part IV*, volume 12828 of *LNCS*, pages 208–238, Virtual Event, August 2021. Springer, Heidelberg.

ALS16. Shweta Agrawal, Benoît Libert, and Damien Stehlé. Fully secure functional encryption for inner products, from standard assumptions. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part III*, volume 9816 of *LNCS*, pages 333–362. Springer, Heidelberg, August 2016.

BSW11. Dan Boneh, Amit Sahai, and Brent Waters. Functional encryption: Definitions and challenges. In Yuval Ishai, editor, *TCC 2011*, volume 6597 of *LNCS*, pages 253–273. Springer, Heidelberg, March 2011.

CDG$^+$18. Jérémy Chotard, Edouard Dufour Sans, Romain Gay, Duong Hieu Phan, and David Pointcheval. Decentralized multi-client functional encryption for inner product. In Thomas Peyrin and Steven Galbraith, editors, *ASIACRYPT 2018, Part II*, volume 11273 of *LNCS*, pages 703–732. Springer, Heidelberg, December 2018.

CDSG$^+$20. Jérémy Chotard, Edouard Dufour-Sans, Romain Gay, Duong Hieu Phan, and David Pointcheval. Dynamic decentralized functional encryption. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part I*, volume 12170 of *LNCS*, pages 747–775. Springer, Heidelberg, August 2020.

DGP21. Cécile Delerablée, Lénaïck Gouriou, and David Pointcheval. Key-policy ABE with delegation of rights. Cryptology ePrint Archive, Report 2021/867, 2021. https://eprint.iacr.org/2021/867.

DOT18. Pratish Datta, Tatsuaki Okamoto, and Junichi Tomida. Full-hiding (unbounded) multi-input inner product functional encryption from the k-Linear assumption. In Michel Abdalla and Ricardo Dahab, editors, *PKC 2018, Part II*, volume 10770 of *LNCS*, pages 245–277. Springer, Heidelberg, March 2018.

GGJ+20. Nicolas Gama, Mariya Georgieva, Dimitar Jetchev, Kevin McCarthy, Jakob Odersky, Alexander Petric, and Abson S. Tang. Detecting money laundering activities via secure multi-partycomputation for structural similarities in flow networks. Real World Cryptography, 2020. https://www.youtube.com/watch?v=4hryY6cMPaM&t=2558s.

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

OT10. Tatsuaki Okamoto and Katsuyuki Takashima. Fully secure functional encryption with general relations from the decisional linear assumption. Cryptology ePrint Archive, Report 2010/563, 2010. https://eprint.iacr.org/2010/563.

OT12a. Tatsuaki Okamoto and Katsuyuki Takashima. Fully secure unbounded inner-product and attribute-based encryption. Cryptology ePrint Archive, Report 2012/671, 2012. https://eprint.iacr.org/2012/671.

OT12b. Tatsuaki Okamoto and Katsuyuki Takashima. Fully secure unbounded inner-product and attribute-based encryption. In Xiaoyun Wang and Kazue Sako, editors, *ASIACRYPT 2012*, volume 7658 of *LNCS*, pages 349–366. Springer, Heidelberg, December 2012.

SNY+16. Reza Soltani, Uyen Trang Nguyen, Yang Yang, Mohammad Reza Faghani, Alaa Yagoub, and Aijun An. A new algorithm for money laundering detection based on structural similarity. In *7th IEEE Annual Ubiquitous Computing, Electronics & Mobile Communication Conference, UEMCON 2016, New York City, NY, USA, October 20-22, 2016*, pages 1–7. IEEE, 2016.

TT18. Junichi Tomida and Katsuyuki Takashima. Unbounded inner product functional encryption from bilinear maps. In Thomas Peyrin and Steven Galbraith, editors, *ASIACRYPT 2018, Part II*, volume 11273 of *LNCS*, pages 609–639. Springer, Heidelberg, December 2018.

A Proof of the Product-Preserving Lemma 8

Proof. We detail the full proof of Lemma 8, following the sequence of games presented on Figure 1.

Game $\mathbf{G}_0$: We start from the original distribution, for random $\pi_i, \mu_i \stackrel{\$}{\leftarrow} \mathbb{Z}_q$, but any x_i, y_i, a_i, b_i, for $i \in [\![1; N]\!]$, and fixed λ_i among Q possible values:

$$\mathbf{C}_i = (\pi_i \cdot (1, \lambda_i), x_i, a_i, 0)_{\mathbb{B}} \qquad \mathbf{D}_i = (\mu_i \cdot (\lambda_i, -1), y_i, b_i, 0)_{\mathbb{B}^*}$$

and

$$\mathbf{C}^* = (\pi^*(1, \lambda^*), x^*, 0, 0)_{\mathbb{B}} \qquad \mathbf{D}^* = (\mu^*(\lambda^*, -1), y^*, 0, 0)_{\mathbb{B}^*}$$

for random π^*, μ^* but any λ^*, x^*, y^*. And we denote S_0 the event that the adversary eventually outputs 1, when distinguishing the distributions of $\mathbf{C}^*$. One can note that in this game, we are with the input distribution (0), where the adversary can choose $(\lambda_i, x_i, y_i, a_i, b_i)$ for all i, and λ^*, x^*, y^*, but λ^* is distinct from any other λ_i, and all the labels being among Q possible values.

Game $\mathbf{G}_1$: As the 5-th components in $\mathbf{C}_i$ and $\mathbf{C}^*$ vectors are all 0, one can duplicate the 3-rd components of $\mathbf{D}_i$ and $\mathbf{D}^*$ vectors at the 5-th position, applying the Replication Lemma 7. This is perfectly indistinguishable: $S_0 = S_1$.

Game $\mathbf{G}_2$: Since the 3-rd and 5-th components of $\mathbf{D}_i$ and $\mathbf{D}^*$ vectors are all identical, one can swap x^* from the 3-rd to the 5-th positions in $\mathbf{C}^*$, applying the Swapping Lemma 5 with the first two components for the required randomness: $S_1 - S_2 \leq 2 \cdot \mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}_1}(t)$.

Game $\mathbf{G}_3$: For any i, the 5-th component of $\mathbf{C}_i$ is 0. Applying the Indexing Lemma 6, on $(\mathbb{B}^*, \mathbb{B})$, between λ^* and every possible label $\lambda_i \neq \lambda^*$, iteratively, one replaces each y_i in the 5-th component of $\mathbf{D}_i$ by 0. More precisely, at the beginning of each the game (each value of i), one anticipates the Q possible values in which λ^* and the λ_i are drawn, one makes a guess on λ^*, and enumerates all the $Q-1$ possible values for λ_i. When the guess on λ^*, one aborts. When the guess is correct (which happens with probability $1/Q$): $S_2/Q - S_3/Q \leq 2N \cdot \mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}_2}(t)$, as there are N indexing operations to apply for all the useful λ_i. Other values have no impact. Hence, $S_2 - S_3 \leq 2NQ \cdot \mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}_2}(t)$

Game $\mathbf{G}_4$: One now removes y^* from the 3-rd component of $\mathbf{D}^*$ with the Replication Lemma 7: replicating negatively the 5-th components at the 3-rd positions of $\mathbf{D}$ vectors, one subtracts y^* in $\mathbf{D}^*$, and 0 in other vectors, but adds the 3-rd component to the 5-th in the $\mathbf{C}_i$ vectors. The two games are perfectly indistinguishable: $S_3 = S_4$.

Game $\mathbf{G}_5$: For any i, the 5-th component of $\mathbf{D}_i$ is 0. Applying the Indexing Lemma 6, on $(\mathbb{B}, \mathbb{B}^*)$, between λ^* and every possible label $\lambda_i \neq \lambda^*$, iteratively, one replaces each x_i in the 5-th components of $\mathbf{C}_i$ by 0. More precisely, at the beginning of each the game (each value of i), one anticipates the Q possible values in which λ^* and the λ_i are drawn, one makes a guess on λ^*, and enumerates all the $Q-1$ possible values for λ_i. When the guess on λ^*, one aborts. When the guess is correct (which happens with probability $1/Q$): $S_4/Q - S_5/Q \leq 2N \cdot \mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}_1}(t)$, as there are N indexing operations to apply for all the useful λ_i. Other values have no impact. Hence, $S_4 - S_5 \leq 2NQ \cdot \mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}_1}(t)$.

Game $\mathbf{G}_6$: One replaces the pair (x^*, y^*) by $(1, x^*y^*)$ in the 5-th coordinates of $\mathbf{C}^*$ and $\mathbf{D}^*$ (if $x^* \neq 0$). To this aim, one makes a random guess $X \stackrel{\$}{\leftarrow} \mathbb{Z}_q$ on x^*, and defines the matrices if $X \neq 0$:

$$B = \big(X\big)_5 \qquad B' = \big(1/X\big)_5 \qquad \mathbb{B} = B \cdot \mathbb{U} \qquad \mathbb{B}^* = B' \cdot \mathbb{U}^*$$

or no change otherwise. If during the execution, $x^* \neq X$, one aborts, otherwise the two games are perfectly indistinguishable in case of correct guess: $S_5/q = S_6/q$, hence $S_5 = S_6$.

Game $\mathbf{G}_7$: In case $x^* = 0$, then, the Subspace Indistinguishability Theorem 3 enables the replacement of the y^* coordinate in $\mathbf{D}^*$ with 0 since the corresponding coordinate in $\mathbf{C}^*$ is null and the $\mathbf{D}^*$'s has random components at the beginning of the vector, and then using the same theorem again we replace the zero in the 5-th coordinate of $\mathbf{C}^*$ with a 1, using the now null fifth component of $\mathbf{D}^*$, and the randomness in one of the first two components of $\mathbf{C}^*$. In this case, we have: $S_6 - S_7 \leq 2\mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}_1}(t) + 2\mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}_2}(t)$.

Game $\mathbf{G}_8$: One now duplicates the 1 from the 5-th component of $\mathbf{C}^*$ to the 4-th using the Replication Lemma 7. This also negatively adds the 4-th components at the 5-th positions of $\mathbf{D}_i$ vectors. The two games are perfectly indistinguishable: $S_7 = S_8$.

Game $\mathbf{G}_9$: One applies twice the Indexing Lemma 6, between λ^* and every possible label $\lambda_i \neq \lambda^*$, iteratively, to first on $(\mathbb{B}^*, \mathbb{B})$ to replace each $-b_i$ in the 5-th components of $\mathbf{D}_i$ by 0, while all the 5-th components of the $\mathbf{C}_i$ are 0, and then on $(\mathbb{B}, \mathbb{B}^*)$ to replace the 0 in the 5-th components of $\mathbf{C}_i$ by a_i, while all the 5-th components of the $\mathbf{D}_i$ are 0. One gets $S_8 - S_9 \leq 2QN \cdot (\mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}_1}(t) + \mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}_2}(t))$.

Game $\mathbf{G}_{10}$: One now swaps the 5-th component x^*y^* and the 4-th component 0 of $\mathbf{D}^*$. This easily works as all the $\mathbf{C}_i$ and $\mathbf{C}^*$ vectors have identical 4-th and 5-th components, using the Swapping Lemma 5, with the first two components for randomness: $S_9 - S_{10} \leq 2 \cdot \mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}_2}(t)$.

Game $\mathbf{G}_{11}$: One then withdraws the 4-th components from the 5-th components in $\mathbf{C}^*$ and the $\mathbf{C}_i$'s, while the null 5-th components of $\mathbf{D}_i$'s and $\mathbf{D}^*$ are added to the 4-th components, using the Replication Lemma 7. This game is perfectly indistinguishable from the previous one: $S_{10} - S_{11} = 0$.

The global probability difference is bounded by $0 + 2 \cdot \mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}_1}(t) + 2NQ \cdot \mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}_2}(t) + 0 + 2NQ \cdot \mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}_1}(t) + 2(\mathsf{Adv}_{\mathbb{G}_1}(t) + \mathsf{Adv}_{\mathbb{G}_2}(t)) + 0 + 2NQ \cdot (\mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}_1}(t) + \mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}_2}(t)) + 2 \cdot \mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}_2}(t) + 0 = 4 \cdot (NQ+1) \cdot (\mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}_1}(t) + \mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}_2}(t)) \leq 8 \cdot (NQ+1) \cdot \mathsf{Adv}^{\mathsf{sxdh}}(t)$.

B Proof of Theorem 11 (Indistinguishability of the 2-Client IPFE)

Proof. The security proof follows the sequence of games presented on Figure 2.

Game $\mathbf{G}_0$: This is the real game with $\boldsymbol{x}_\ell$ and $\boldsymbol{y}_\ell$ encrypted in $\mathsf{Encrypt}$-queries: one starts with a pairing-friendly setting $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_t, e, G_1, G_2, q)$, with random dual orthogonal matrices B and B^* of dimension 12 that define $(\mathbb{B}, \mathbb{B}^*)$, and two secret matrices $\underline{\mathbf{S}}, \underline{\mathbf{T}} \stackrel{\$}{\leftarrow} (\mathbb{Z}_q^3)^n$. One sets the encryption keys $\mathsf{ek}_1 \leftarrow (B, \underline{\mathbf{S}})$ and $\mathsf{ek}_2 \leftarrow (B^*, \underline{\mathbf{T}})$, and the secret key $\mathsf{sk} \leftarrow (\underline{\mathbf{S}}, \underline{\mathbf{T}})$. Then, the public parameters $\mathsf{PK} \leftarrow (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_t, e, G_1, G_2, q)$ are provided to the adversary.

- For the query $\mathsf{OKeyGen}(\boldsymbol{\alpha}_k)$, with $\boldsymbol{\alpha}_k \in \mathbb{Z}_q^n$, one outputs

$$\mathsf{dk}_{\boldsymbol{\alpha}_k} \leftarrow \boldsymbol{d}_k = \sum \alpha_{k,i}(s_{i,1}t_{i,1}, s_{i,2}t_{i,2}, s_{i,3}t_{i,3})$$

- For a query $\mathsf{LoREncrypt}_1(\lambda_\ell, \boldsymbol{x}^0_\ell, \boldsymbol{x}^1_\ell)$, one first derives $\lambda_{\ell,i} = \mathcal{H}(\lambda_\ell, i) \in \mathbb{Z}_q$, for $i \in [\![1; n]\!]$, and then outputs $\mathbf{C}_\ell \leftarrow (\lambda_\ell, \boldsymbol{C}_{\ell,0}, (\boldsymbol{C}_{\ell,i})_i)$ where $C_{\ell,0,j} = \boldsymbol{\sigma}_\ell \cdot G_1$, for a random $\boldsymbol{\sigma}_\ell \stackrel{\$}{\leftarrow} (\mathbb{Z}_q^*)^3$, and, for random $\pi_{\ell,i} \stackrel{\$}{\leftarrow} \mathbb{Z}_q$,

$$\forall i \in [\![1; n]\!], \qquad \boldsymbol{C}_{\ell,i} = (\pi_{\ell,i} \cdot (1, \lambda_{\ell,i}), x^b_{\ell,i}, 0, 0, (\sigma_{\ell,j} s_{i,j}, 0)_j, 0)_{\mathbb{B}}$$

- For a query $\mathsf{LoREncrypt}_2(\lambda_\ell, \boldsymbol{y}^0_\ell, \boldsymbol{y}^1_\ell)$, one first derives $\lambda_{\ell,i} = \mathcal{H}(\lambda_\ell, i) \in \mathbb{Z}_q$, for $i \in [\![1; n]\!]$, and then outputs $\mathbf{D}_\ell \leftarrow (\lambda_\ell, \boldsymbol{D}_{\ell,0}, (\boldsymbol{D}_{\ell,i})_i)$ where $D_{\ell,0,j} = \boldsymbol{\tau}_\ell \cdot G_2$, for a random $\boldsymbol{\tau}_\ell \stackrel{\$}{\leftarrow} (\mathbb{Z}_q^*)^3$, and, for random $\mu_{\ell,i} \stackrel{\$}{\leftarrow} \mathbb{Z}_q$,

$$\forall i \in [\![1; n]\!], \qquad \boldsymbol{D}_{\ell,i} = (\mu_{\ell,i} \cdot (\lambda_{\ell,i}, -1), y^b_{\ell,i}, 0, 0, (\tau_{\ell,j} t_{i,j}, 0)_j, 0)_{\mathbb{B}^*}$$

Then we consider $S_0 = \Pr[\beta = b]$, where $\beta \overset{\$}{\leftarrow} \{0,1\}$ if for some $\boldsymbol{\alpha}_k$ asked to the $\mathsf{OKeyGen}$-oracle and some $(\boldsymbol{x}_\ell^0, \boldsymbol{x}_\ell^1)$, $(\boldsymbol{y}_\ell^0, \boldsymbol{y}_\ell^1)$ asked to the $\mathsf{LoREncrypt}$ oracle one has the inequality $\sum \alpha_{k,i} x_{\ell,i}^0 y_{\ell,i}^0 \neq \sum \alpha_{k,i} x_{\ell,i}^1 y_{\ell,i}^1$, which is a non-legitimate attack, or $\beta \leftarrow b'$, as this is a legitimate attack. Our goal is to show that S_0 is close to $1/2$. In the following games, we alter the simulation of all the $\mathsf{LoREncrypt}$-oracles. We denote N the number of distinct λ_ℓ, but anticipate by programming at most Qn values of $\lambda_{\ell,i}$ (among a select set), and K the number of $\mathsf{OKeyGen}$-queries. Additionally, we denote $\mathcal{L}_1$ the set of indices ℓ where $\mathsf{LoREncrypt}_1$ is asked first, and $\mathcal{L}_2$ the set of indices where $\mathsf{LoREncrypt}_2$ is asked first. Their sizes are respectively N_1 and N_2.

Game $\mathbf{G}_1$: We first deal with all the indices $\ell \in \mathcal{L}_1$ such that $\mathsf{LoREncrypt}_1(\lambda_\ell, \boldsymbol{x}_\ell^0, \boldsymbol{x}_\ell^1)$ is asked first. We apply the Product-Preserving Lemma 8 on the vectors $(x_{\ell,i}, 0)$ and $(y_{\ell,i}, 0)$ to replace them into $(0,1)$ and the preserved product $(0, x_{\ell,i} y_{\ell,i})$. As we have to do it N_1 times, each with a guess on the correct label: $S_0 - S_1 \leq 8QN_1(Nn+1) \cdot \mathsf{Adv}^{\mathsf{sxdh}}(t)$.

Game $\mathbf{G}_2$: We apply again the Product-Preserving Lemma 8 on the vectors $(\sigma_{\ell,j} s_{i,j}, 0)$ and $(\tau_{\ell,j} t_{i,j}, 0)$ to replace them into $(0,1)$ and the preserved product $(0, \sigma_{\ell,j}\tau_{\ell,j} s_{i,j} t_{i,j})$, successively for $j = 1,2,3$: $S_1 - S_2 \leq 24QN_1(Nn+1) \cdot \mathsf{Adv}^{\mathsf{sxdh}}(t)$.

Game $\mathbf{G}_3$: We now formally replace $z_{\ell,i} \leftarrow x_{\ell,i} y_{\ell,i}$, and $\tau_{\ell,j} \leftarrow \rho_{\ell,j}/\sigma_{\ell,j}$, for $j = 1,2,3$. This makes no difference: $S_2 = S_3$.

Game $\mathbf{G}_4$: We replace the random choice $\boldsymbol{\rho}_\ell \overset{\$}{\leftarrow} \mathbb{Z}_q^3$ by $\boldsymbol{\rho}_\ell$ in the span of $\boldsymbol{v}_1 = (v_{1,1}, v_{1,2}, v_{1,3})$, for $v_{1,1} = 1$ but random $v_{1,2}, v_{1,3} \overset{\$}{\leftarrow} \mathbb{Z}_q$: $\boldsymbol{\rho}_\ell = \rho_\ell \cdot \boldsymbol{v}_2$ for $\rho_\ell \overset{\$}{\leftarrow} \mathbb{Z}_q$. The simulation of $\boldsymbol{D}_{\ell,0} = (\rho_{\ell,1}/\sigma_{\ell,1} \cdot G_2, \rho_{\ell,2}/\sigma_{\ell,2} \cdot G_2, \rho_{\ell,3}/\sigma_{\ell,3} \cdot G_2)$ does not need to know $\boldsymbol{\rho}_\ell$, but just $\boldsymbol{\rho}_\ell \cdot G_2$, as the vector $\boldsymbol{\sigma}_\ell$ is known. The simulation of the $\boldsymbol{D}_{\ell,i}$ do not either, but just $\boldsymbol{\rho}_\ell \cdot G_2$ too. Under the DDH assumption in $\mathbb{G}_2$, this game is indistinguishable from the previous one, after N_1 successive changes. See Lemma 13. Hence, $S_4 - S_3 \leq N_1 \cdot \mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}_2}(t) + N_1/q$.

Game $\mathbf{G}_5$: Since the 4-th, the 7-th and the 9-th components of $\boldsymbol{C}_{\ell,i}$ vectors are all identical to 1, one can successively swap elements from the 7-th to the 4-th positions, and from the 9-th to the 4-th positions in $\boldsymbol{D}_{\ell,i}$ to get

$$c_{\ell,i} = z_{\ell,i}^b + \rho_\ell \sum_j v_{1,j} s_{i,j} t_{i,j}$$

Applying the Swapping Lemma 5, one gets $S_4 - S_5 \leq 4 \cdot \mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}_2}(t)$.

Game $\mathbf{G}_6$: Since the 7-th and the 9-th components of the $\boldsymbol{D}_{\ell,i}$ vectors are null, we use the Replication Lemma 7 twice to withdraw the 4-th component of the $\boldsymbol{C}_{\ell,i}$'s, 1, to their 7-th and then 9-th components, all the while adding zeros to the 4-th component in the $\boldsymbol{D}_{\ell,i}$'s. This is perfectly indistinguishable from the previous game: $S_5 - S_6 = 0$.

Game $\mathbf{G}_7$: We now do the same as the previous sequence, but for the indices ℓ such that $\mathsf{LoREncrypt}_2(\lambda_\ell, \boldsymbol{x}_\ell^0, \boldsymbol{x}_\ell^1)$ is asked first, where we replace the random choice $\boldsymbol{\rho}_\ell \overset{\$}{\leftarrow} \mathbb{Z}_q^3$ by $\boldsymbol{\rho}_\ell$ in the span of $\boldsymbol{v}_2 = (v_{2,1}, v_{2,2}, v_{2,3})$, for $v_{2,1} = 1$ but random $v_{2,2}, v_{2,3} \overset{\$}{\leftarrow} \mathbb{Z}_q$: $\boldsymbol{\rho}_\ell = \rho_\ell \cdot \boldsymbol{v}_2$ for $\rho_\ell \overset{\$}{\leftarrow} \mathbb{Z}_q$. We can additionally formally replace $r_{i,j} \leftarrow s_{i,j} t_{i,j}$:

$$\forall \ell \in \mathcal{L}_1 \qquad c_{\ell,i} = z_{\ell,i}^b + \rho_\ell \sum_j v_{1,j} r_{i,j} = z_{\ell,i}^b + \rho_\ell \cdot \boldsymbol{r}_i \boldsymbol{v}_1^\top$$

$$\forall \ell \in \mathcal{L}_2 \qquad c_{\ell,i} = z_{\ell,i}^b + \rho_\ell \sum_j v_{2,j} r_{i,j} = z_{\ell,i}^b + \rho_\ell \cdot \boldsymbol{r}_i \boldsymbol{v}_2^\top$$

We thus have the same distance as between $\mathbf{G}_0$ and $\mathbf{G}_6$, but with at most N_2 labels and opposite groups: $S_6 - S_7 \leq 32QN_2(Nn+1) \cdot \mathsf{Adv}^{\mathsf{sxdh}}(t) + (4 + N_2) \cdot \mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}_1}(t) + N_2/q$.
In this game, in the setting $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_t, e, G_1, G_2, q)$, one starts with random dual orthogonal matrices B and B^* of dimension 12 that define $(\mathbb{B}, \mathbb{B}^*)$, two random vectors $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$ with $v_{1,1} = v_{2,1} = 1$, and a secret matrix $\underline{\mathbf{R}} \overset{\$}{\leftarrow} (\mathbb{Z}_q^3)^n$. The public parameters $\mathsf{PK} \leftarrow (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_t, e, G_1, G_2, q)$ are provided to the adversary.

- For the query $\mathsf{OKeyGen}(\boldsymbol{\alpha}_k)$, with $\boldsymbol{\alpha}_k \in \mathbb{Z}_q^n$, one outputs $\mathsf{dk}_{\boldsymbol{\alpha}_k} \leftarrow \boldsymbol{d}_k = (d_{k,j} = \sum \alpha_{k,i} r_{i,j})$;
- For $\ell \in \mathcal{L}_1$: a query $\mathsf{LoREncrypt}_1(\lambda_\ell, \boldsymbol{x}_\ell^0, \boldsymbol{x}_\ell^1)$ is first asked for the label λ_ℓ. One asks for $\mathsf{OEncrypt}_1(\lambda_\ell)$: one first derives $\lambda_{\ell,i} = \mathcal{H}(\lambda_\ell, i) \in \mathbb{Z}_q$, for $i \in [\![1;n]\!]$, and outputs $\mathbf{C}_\ell \leftarrow (\lambda_\ell, \boldsymbol{C}_{\ell,0}, (\boldsymbol{C}_{\ell,i})_i)$, where $\boldsymbol{C}_{\ell,0} = \boldsymbol{\sigma}_\ell \cdot G_1$, for random $\boldsymbol{\sigma}_\ell \overset{\$}{\leftarrow} (\mathbb{Z}_q^*)^3$, and for $i = 1, \ldots, n$, with random $\pi_{\ell,i} \overset{\$}{\leftarrow} \mathbb{Z}_q$,
$$\boldsymbol{C}_{\ell,i} = (\pi_{\ell,i} \cdot (1, \lambda_{\ell,i}), 0, 1, 0, (0,0)_j, 0)_{\mathbb{B}}$$
When $\mathsf{LoREncrypt}_2(\lambda_\ell, \boldsymbol{y}_\ell^0, \boldsymbol{y}_\ell^1)$ is queried, one asks for $\mathsf{OEncrypt}_2(\lambda_\ell, \boldsymbol{z}_\ell)$, where $\boldsymbol{z}_\ell = (x_{\ell,i} y_{\ell,i})_i$ and outputs $\boldsymbol{D}_\ell \leftarrow (\lambda_\ell, \boldsymbol{D}_{\ell,0}, (\boldsymbol{D}_{\ell,i})_i)$, where
$$\boldsymbol{D}_{\ell,0} = \rho_\ell \cdot (v_{1,1}/\sigma_{\ell,1}, v_{1,2}/\sigma_{\ell,2}, v_{1,3}/\sigma_{\ell,3}) \cdot G_2$$
for random $\rho_\ell \overset{\$}{\leftarrow} \mathbb{Z}_q$, and for $i = 1, \ldots, n$, with random $\pi_{\ell,i} \overset{\$}{\leftarrow} \mathbb{Z}_q$,
$$\boldsymbol{D}_{\ell,i} = (\pi_{\ell,i} \cdot (1, \lambda_{\ell,i}), 0, z_{\ell,i}^b + \rho_\ell \cdot \boldsymbol{r}_i \boldsymbol{v}_1^\top, 0, (0,0)_j, 0)_{\mathbb{B}^*}$$
- For $\ell \in \mathcal{L}_2$: the simulation is symmetrical.

For a legitimate attack, we must have $\boldsymbol{z}_\ell^0 \boldsymbol{\alpha}_k^\top = \sum \alpha_{k,i} z_{\ell,i}^0 = \sum \alpha_{k,i} z_{\ell,i}^1 = \boldsymbol{z}_\ell^1 \cdot \boldsymbol{\alpha}_k^\top$ for all the functional key queries $\boldsymbol{\alpha}_k$: all $\boldsymbol{z}_\ell^0 - \boldsymbol{z}_\ell^1$ are orthogonal to all $\boldsymbol{\alpha}_k$.

Game $\mathbf{G}_8$: We eventually replace $\boldsymbol{z}_\ell^b$ by $\boldsymbol{z}_\ell^0$ in all the encryption:
$$\forall \ell \in \mathcal{L}_1 \qquad c'_{\ell,i} = z_{\ell,i}^0 + \rho_\ell \cdot \boldsymbol{r}_i \boldsymbol{v}_1^\top$$
$$\forall \ell \in \mathcal{L}_2 \qquad c'_{\ell,i} = z_{\ell,i}^0 + \rho_\ell \cdot \boldsymbol{r}_i \boldsymbol{v}_2^\top$$
Since b does not appear anywhere, $S_8 = 0$. But we also have to prove there is not much difference between $\mathbf{G}_8$ and $\mathbf{G}_7$: Using Lemma 14, one gets $S_8 - S_7 \leq 2N \cdot \mathsf{Adv}^{\mathsf{sxdh}}(t) + 2N/q$.

We can thus conclude that
$$\begin{aligned} S_0 - S_8 \leq\ & 32QN_1(Nn+1) \cdot \mathsf{Adv}^{\mathsf{sxdh}}(t) + (4 + N_1) \cdot \mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}_2}(t) + N_1/q \\ & + 32QN_2(Nn+1) \cdot \mathsf{Adv}^{\mathsf{sxdh}}(t) + (4 + N_2) \cdot \mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}_1}(t) + N_2/q \\ & + 2N \cdot \mathsf{Adv}^{\mathsf{sxdh}}(t) + 2N/q \\ \leq\ & (32QN(Nn+1) + 3N + 8) \cdot \mathsf{Adv}^{\mathsf{sxdh}}(t) + 3N/q \end{aligned}$$

Lemma 13. *From the view of $\boldsymbol{\rho}_\ell \cdot G_2 \in \mathbb{G}_2^3$, for $\ell = 1, \ldots, N$, one cannot make the difference between $\boldsymbol{\rho}_\ell = \rho_\ell \cdot \boldsymbol{v}$, with $\rho_\ell \overset{\$}{\leftarrow} \mathbb{Z}_q$ for all ℓ, for some random vector $\boldsymbol{v} \in \mathbb{Z}_q^3$ with $v_1 = 1$, and $\boldsymbol{\rho}_\ell \overset{\$}{\leftarrow} \mathbb{Z}_q^3$ for all ℓ, with advantage greater than $N \cdot \mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}_2}(t) + N/q$.*

Proof. For a given ℓ, one can define the hybrid game $\mathbf{G}_\ell$: for some random vector $\boldsymbol{v} \in \mathbb{Z}_q^3$ with $v_1 = 1$

- For all indices $j \in [\![1;\ell]\!]$, one chooses a random $\rho_j \overset{\$}{\leftarrow} \mathbb{Z}_q^*$, and sets $\boldsymbol{\rho}_j \leftarrow \rho_j \cdot \boldsymbol{v}$;
- For all indices $j \in [\![\ell+1;N]\!]$, one chooses $\boldsymbol{\rho}_j \overset{\$}{\leftarrow} \mathbb{Z}_q^3$.

This is clear that in $\mathbf{G}_0$, all the vectors are random, while in $\mathbf{G}_N$, all the vectors are co-linear with $\boldsymbol{v}$.

Let us consider the following game from some ℓ: from a Decisional Diffie-Hellman instance in $\mathbb{G}_2$, $U = u \cdot G_2$, $V = v \cdot G_2$, and $W = w \cdot G_2$, where $u, v \overset{\$}{\leftarrow} \mathbb{Z}_q$ and $w = uv + \delta$ with either $\delta \overset{\$}{\leftarrow} \mathbb{Z}_q$ or $\delta = 0$: one chooses random $a, b \overset{\$}{\leftarrow} \mathbb{Z}_q$ and set $A \leftarrow U$, $B \leftarrow a \cdot G_2 + b \cdot U$, hence $(G_2, A, B) = \boldsymbol{v} \cdot G_2$ with
$$v_1 = 1 \qquad v_2 = u \qquad v_3 = a + ub$$

For all indices $j < \ell$, one sets, for a random $\rho_j \overset{\$}{\leftarrow} \mathbb{Z}_q^*$,

$$\boldsymbol{\rho}_j \cdot G_2 = \rho_j \cdot (G_2, A, B) = \rho_j \cdot \boldsymbol{v} \cdot G_2$$

For all indices $j > \ell$, one chooses a random $\boldsymbol{\rho}_j \overset{\$}{\leftarrow} \mathbb{Z}_q^3$. For the ℓ-th query, one chooses a random $x \overset{\$}{\leftarrow} \mathbb{Z}_q$ and sets

$$\begin{aligned}
\rho_{\ell,1} G_2 &\leftarrow x G_2 + V = (x+v) G_2 = \rho G_2 = \rho_\ell v_1 G_2 \\
\rho_{\ell,2} G_2 &\leftarrow x U + C = (xu + w) G_2 = (\rho u - \delta) G_2 = (\rho_\ell v_2 - \delta) G_2 \\
\rho_{\ell,3} G_2 &\leftarrow a x G_2 + b x U + a V + b W = (ax + bxu + av + bw) G_2 \\
&\qquad = (\rho(a + bu) - \delta b) G_2 = (\rho_\ell v_3 - \delta b) G_2
\end{aligned}$$

where $\rho_\ell = x + v$. If $\delta = 0$, $\boldsymbol{\rho}_\ell = \rho_\ell \cdot \boldsymbol{v}$, for a random ρ_ℓ as $x \overset{\$}{\leftarrow} \mathbb{Z}_q$. Then we are in $\mathbf{G}_\ell$, unless $x = -v$. When $\delta \overset{\$}{\leftarrow} \mathbb{Z}_q$, as $b \overset{\$}{\leftarrow} \mathbb{Z}_q$, $\boldsymbol{\rho}_\ell$ is random in $\mathbb{Z}_q^3$. So we are in $\mathbf{G}_{\ell-1}$.

Lemma 14. *$S_7 - S_8 \leq 2N \cdot \mathsf{Adv}^{\mathsf{sxdh}}(t) + 2N/q$.*

Proof. We proceed iteratively on each ℓ, as in [CDG+18], but in both $\mathbb{G}_1$ and $\mathbb{G}_2$, which requires a randomness with dimension 3:

Game $\mathbf{G}_{7.\ell.0}$ This is the previous game, and we deal with the ℓ-th label that is in $\mathcal{L}_\mathfrak{b}$, for the appropriate $\mathfrak{b}$, that is learnt from the first query of the adversary.

Game $\mathbf{G}_{7.\ell.1}$ We use back $\boldsymbol{\rho}_\ell \overset{\$}{\leftarrow} \mathbb{Z}_q^3$, instead of the span of $\boldsymbol{v}_\mathfrak{b}$: $c'_{\ell,i} = z^b_{\ell,i} + \rho_\ell \cdot \boldsymbol{r}_i \boldsymbol{v}_\mathfrak{b}^\top$ is replaced by $c^1_{\ell,i} = z^b_{\ell,i} + \boldsymbol{r}_i \boldsymbol{\rho}_\ell^\top$, with $S_{7.\ell.0} - S_{7.\ell.1} \leq \mathsf{Adv}^{\mathsf{sxdh}}(t)$, using the same analysis as in the proof of Lemma 13.

Game $\mathbf{G}_{7.\ell.2}$ We stop if eventually $\boldsymbol{\rho}_\ell$ is in the plane spanned by $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$: $S_{7.\ell.1} - S_{7.\ell.2} \leq 1/q$. Otherwise, denoting $\boldsymbol{v}_3$ a unitary vector orthogonal to both $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$, we can note $\boldsymbol{\rho}_\ell = c_1 \boldsymbol{v}_1 + c_2 \boldsymbol{v}_2 + c_3 \boldsymbol{v}_3$, with $c_3 \neq 0 \bmod q$.

Game $\mathbf{G}_{7.\ell.3}$ We replace $\boldsymbol{z}^b_\ell$ by $\boldsymbol{z}^0_\ell$ in the encryption: $c^1_{\ell,i} = z^b_{\ell,i} + \boldsymbol{r}_i \boldsymbol{\rho}_\ell^\top$ is replaced by $c^2_{\ell,i} = z^0_{\ell,i} + \boldsymbol{r}_i \boldsymbol{\rho}_\ell^\top$. As in [CDG+18], we can use the complexity leveraging, by guessing $(\boldsymbol{z}^0_\ell, \boldsymbol{z}^1_\ell)$ to show this makes no difference, by building an adversary that guesses $(\boldsymbol{z}^0_\ell, \boldsymbol{z}^1_\ell)$. We then note S^* the success of this new adversary that makes both a correct guess on $(\boldsymbol{z}^0_\ell, \boldsymbol{z}^1_\ell)$ and a correct guess for b with a legitimate attack.

- Step 1: one guesses $(\boldsymbol{z}^0_\ell, \boldsymbol{z}^1_\ell)$, which can either be in $(\mathbb{Z}_q)^{2n}$ or empty if the second encryption query is not asked. There are thus $q^{2n} + 1$ possibilities. If the guess is correct, one does not change anything, if this is incorrect, one stops and outputs a random bit. One can note that the advantage in this game is $S^*_1 = S_{7.\ell.2}/(q^{2n} + 1)$, as the good guess is exactly $1/(q^{2n} + 1)$, even in case of correct guess of b in a legitimate attack;
- Step 2: we can replace $\boldsymbol{r}_i$ by $\boldsymbol{u}_i = \boldsymbol{r}_i + \gamma(z^b_{\ell,i} - z^0_{\ell,i}) \cdot \boldsymbol{v}_3$, for random $\gamma \overset{\$}{\leftarrow} \mathbb{Z}_q$, as $\underline{\mathbf{R}} = (\boldsymbol{r}_i)_i \overset{\$}{\leftarrow} (\mathbb{Z}_q^3)^n$ is random and independent of $(\boldsymbol{z}^0_\ell, \boldsymbol{z}^1_\ell)$. We have to check this does not impact the view of the adversary:
  - the keys, for all k, are generated as

$$\begin{aligned}
\boldsymbol{d}_k &= \sum_i \alpha_{k,i} \cdot \boldsymbol{u}_i = \sum_i \alpha_{k,i} \cdot \left(\boldsymbol{r}_i + \gamma(z^b_{\ell,i} - z^0_{\ell,i}) \cdot \boldsymbol{v}_3 \right) \\
&= \sum_i \alpha_{k,i} \cdot \boldsymbol{r}_i + \gamma \cdot \left(\sum_i \alpha_{k,i} \cdot (z^b_{\ell,i} - z^0_{\ell,i}) \right) \cdot \boldsymbol{v}_3 = \sum_i \alpha_{k,i} \cdot \boldsymbol{r}_i
\end{aligned}$$

    as for a legitimate attack, $\sum_i \alpha_{k,i}(z^b_{\ell,i} - z^0_{\ell,i}) = 0$.

- the ciphertexts, for all $j \neq \ell$ are all generated with $\boldsymbol{\rho}_j = \rho_j \cdot \boldsymbol{v}_\delta$, for some $\delta \in \{1,2\}$. So, with $b_j = 0$, for $j < \ell$, and $b_j = b$ for $j > \ell$:

$$\begin{aligned} c_{j,i} &= z_{j,i}^{b_j} + \rho_j \cdot \boldsymbol{u}_i \cdot \boldsymbol{v}_\delta^\top = z_{j,i}^{b_j} + \rho_j \cdot \Big(\boldsymbol{r}_i + \gamma(z_{\ell,i}^b - z_{\ell,i}^0) \cdot \boldsymbol{v}_3\Big) \cdot \boldsymbol{v}_\delta^\top \\ &= z_{j,i}^{b_j} + \rho_j \cdot \boldsymbol{r}_i \cdot \boldsymbol{v}_\delta^\top \end{aligned}$$

- excepted $c_{\ell,i}$:

$$\begin{aligned} c_{\ell,i} &= z_{\ell,i}^b + \boldsymbol{u}_i \cdot \boldsymbol{\rho}_\ell^\top \\ &= z_{\ell,i}^b + \boldsymbol{u}_i \cdot \boldsymbol{\rho}_\ell^\top + \gamma(z_{\ell,i}^b - z_{\ell,i}^0) \cdot \boldsymbol{v}_3 \cdot (c_1 \boldsymbol{v}_1^\top + c_2 \boldsymbol{v}_2^\top + c_3 \boldsymbol{v}_3^\top) \\ &= z_{\ell,i}^b + \boldsymbol{u}_i \cdot \boldsymbol{\rho}_\ell^\top + c_3 \gamma(z_{\ell,i}^b - z_{\ell,i}^0) \cdot \boldsymbol{v}_3 \cdot \boldsymbol{v}_3^\top \\ &= z_{\ell,i}^b + c_3 \gamma(z_{\ell,i}^b - z_{\ell,i}^0) + \boldsymbol{u}_i \cdot \boldsymbol{\rho}_\ell^\top = z_{\ell,i}^0 + \boldsymbol{u}_i \cdot \boldsymbol{\rho}_\ell^\top \end{aligned}$$

if $\gamma = -1/c_3$, as $\boldsymbol{v}_3$ is a unitary vector. Then $S_2^* = S_1^*$.

And this is clear that $S_2^* = S_{7.\ell.3}/(q^{2n}+1)$. Hence, $S_{7.\ell.3} = S_{7.\ell.2}$.

Game $\mathbf{G}_{7.\ell.4}$ After having ignored linear vectors, we force $\boldsymbol{\rho}_\ell$ to be in the span of $\boldsymbol{v}_{\mathfrak{b}}$: $S_{7.\ell.3} - S_{7.\ell.4} \leq \mathsf{Adv}^{\mathsf{sxdh}}(t) + 1/q$, as above.

C Proof of Theorem 12 (Indistinguishability of the IPFE with Selector)

As we will have several vectors with the same labels, we first need to propose a variant of the Product-Preserving Lemma 8, whose proof is presented in appendix D:

Lemma 15 (Alternative Product-Preserving Lemma). *For two orthogonal bases $\mathbb{B}$ and $\mathbb{B}^*$, of length 5, unknown to the adversary $\mathcal{A}$, given a list of indexed vectors, for any $(x_{i,1}, \ldots, x_{i,n}, y_i, a_{i,1}, \ldots, a_{i,n}, b_i)$, fixed indices λ_i among Q possible values , but random unknown $\pi_{i,1}, \ldots, \pi_{i,n}, \mu_i$, for $i = [\![1;N]\!]$, $j \in [\![1;n]\!]$:*

$$\mathbf{C}_{i,j} = (\pi_{i,j}(1, \lambda_i), x_{i,j}, a_{i,j}, 0)_{\mathbb{B}} \qquad \mathbf{D}_i = (\mu_i(\lambda_i, -1), y_i, b_i, 0)_{\mathbb{B}^*}$$

$\mathcal{A}$ cannot distinguish between the two cases, for fixed $\lambda^* \neq \lambda_i$, and any $x_1^*, \ldots, x_n^*$ and y^*, but unknown random $\pi_1^*, \ldots, \pi_n^*, \mu^*$:

$$\begin{aligned} &(0) \qquad \mathbf{C}_j^* = (\pi_j^*(1, \lambda^*), x_j^*, 0, 0)_{\mathbb{B}} \qquad \mathbf{D}^* = (\mu^*(\lambda^*, -1), y^*, 0, 0)_{\mathbb{B}^*} \\ &(1) \qquad \mathbf{C}_j^* = (\pi_j^*(1, \lambda^*), 0, x_j^* y^*, 0)_{\mathbb{B}} \qquad \mathbf{D}^* = (\mu^*(\lambda^*, -1), 0, 1, 0)_{\mathbb{B}^*} \end{aligned}$$

with an advantage greater than $2 \cdot (Qn(N+1)+3) \cdot \mathsf{Adv}_{\mathbb{G}_1}^{\mathsf{ddh}}(t) + 2 \cdot (Qn(N+1)+1) \cdot \mathsf{Adv}_{\mathbb{G}_2}^{\mathsf{ddh}}(t))$, and thus greater than $4 \cdot (Qn(N+1)+2) \cdot \mathsf{Adv}^{\mathsf{sxdh}}(t)$.

In Figure 7, we present the sequence of games, starting from the real game, where the λ_ℓ for $\ell \in [\![1;N]\!]$ are all the distinct labels, and $\lambda_{\ell,1} = \mathcal{H}(\lambda_\ell), \lambda_{\ell,2} = \mathcal{H}'(\lambda_\ell)$, with N the total number of distinct labels.

Proof. In this proof we suppose the data consumer DC receives ciphertexts from DP_2 before or at the same time as from DP_1, to be able to get the product preserving property. The security proof follows the sequence of games presented on Figure 7. In the first game, we rename the vectors in order to simplify the notations in the proof.

Game $\mathbf{G}_0$: This is the real game with $\boldsymbol{x}_\ell$ and $\boldsymbol{y}_\ell$ encrypted in $\mathsf{Encrypt}$-queries: one starts with a pairing-friendly setting $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_t, e, G_1, G_2, q)$, with random dual orthogonal matrices B and B^* of dimension 9 that define $(\mathbb{B}, \mathbb{B}^*)$, and secret vectors $\underline{\mathbf{S}} = (\boldsymbol{s}_i, \boldsymbol{s}'_i)_{i=1}^n = (s_{i,1}, s_{i,2})_{i=1}^{2n} \xleftarrow{\$} ((\mathbb{Z}_q^2)^2)^n$ and $\underline{\mathbf{T}} = (\boldsymbol{t}, \boldsymbol{t}') = (t_{k,1}, t_{k,2})_{k=1}^2 \xleftarrow{\$} (\mathbb{Z}_q^2)^2$. One sets $\mathsf{ek}_1 \leftarrow (B, \underline{\mathbf{S}})$, $\mathsf{ek}_2 \leftarrow (B^*, \underline{\mathbf{T}})$, and $\mathsf{sk} \leftarrow (\underline{\mathbf{S}}, \underline{\mathbf{T}})$. Then, the public parameters $\mathsf{PK} \leftarrow (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_t, e, G_1, G_2, q)$ are provided to the adversary.

$$
\begin{array}{lllcclccl}
\mathbf{G}_0 & \forall \ell \in [\![1;N]\!], & \boldsymbol{C}_{\ell,i} = (\ \cdots & \mid x^b_{\ell,i} & 0 & \mid(\sigma_{\ell,j}s_{i,j} & 0 &)_j\mid & 0\ ) \\
 & & \boldsymbol{D}_{\ell,k} = (\ \cdots & \mid y^b_{\ell,k} & 0 & \mid(\tau_{\ell,j}t_{k,j} & 0 &)_j\mid & 0\ ) \\
\mathbf{G}_1 & \forall \ell, & \boldsymbol{C}_{\ell,i} = (\ \cdots & \mid\ 0 & x^b_{\ell,i}y^b_{\ell,\lceil i/(2n)\rceil} & \mid(\sigma_{\ell,j}s_{i,j} & 0 &)_j\mid & 0\ ) \\
\text{Product-Preserving} & & \boldsymbol{D}_{\ell,k} = (\ \cdots & \mid\ 0 & 1 & \mid(\tau_{\ell,j}t_{k,j} & 0 &)_j\mid & 0\ ) \\
\mathbf{G}_2 & \forall \ell, & \boldsymbol{C}_{\ell,i} = (\ \cdots & \mid\ 0 & x^b_{\ell,i}y^b_{\ell,\lceil i/(2n)\rceil} & \mid(\ 0 & \sigma_{\ell,j}\tau_{\ell,j}s_{i,j}t_{\lceil i/(2n)\rceil,j} &)_j\mid & 0\ ) \\
\text{Product-Preserving} & & \boldsymbol{D}_{\ell,k} = (\ \cdots & \mid\ 0 & 1 & \mid(\ 0 & 1 &)_j\mid & 0\ ) \\
\mathbf{G}_3 & \forall \ell, & \boldsymbol{C}_{\ell,i} = (\ \cdots & \mid\ 0 & z^b_{\ell,i} & \mid(\ 0 & \rho_{\ell,j}s_{i,j}t_{\lceil i/(2n)\rceil,j} &)_j\mid & 0\ ) \\
\text{Formal Renaming} & & \boldsymbol{D}_{\ell,k} = (\ \cdots & \mid\ 0 & 1 & \mid(\ 0 & 1 &)_j\mid & 0\ ) \\
\mathbf{G}_4 & \forall \ell, & \boldsymbol{C}_{\ell,i} = (\ \cdots & \mid\ 0 & z^b_{\ell,i} & \mid(\ 0 & \rho_\ell v_j s_{i,j}t_{\lceil i/(2n)\rceil,j} &)_j\mid & 0\ ) \\
\text{Random in Span} & & \boldsymbol{D}_{\ell,k} = (\ \cdots & \mid\ 0 & 1 & \mid(\ 0 & 1 &)_j\mid & 0\ ) \\
\mathbf{G}_5 & \forall \ell, & \boldsymbol{C}_{\ell,i} = (\ \cdots & \mid\ 0 & c_{\ell,i} & \mid(\ 0 & 0 &)_j\mid & 0\ ) \\
\text{Swapping} & & \boldsymbol{D}_{\ell,k} = (\ \cdots & \mid\ 0 & 1 & \mid(\ 0 & 1 &)_j\mid & 0\ ) \\
\mathbf{G}_6 & \forall \ell, & \boldsymbol{C}_{\ell,i} = (\ \cdots & \mid\ 0 & c_{\ell,i} & \mid(\ 0 & 0 &)_j\mid & 0\ ) \\
\text{Replication} & & \boldsymbol{D}_{\ell,k} = (\ \cdots & \mid\ 0 & 1 & \mid(\ 0 & 0 &)_j\mid & 0\ ) \\
\mathbf{G}_7 & \forall \ell, & \boldsymbol{C}_{\ell,i} = (\ \cdots & \mid\ 0 & c'_{\ell,i} & \mid(\ 0 & 0 &)_j\mid & 0\ ) \\
b = 0 & & \boldsymbol{D}_{\ell,k} = (\ \cdots & \mid\ 0 & 1 & \mid(\ 0 & 0 &)_j\mid & 0\ )
\end{array}
$$

The 2 first components of the $\boldsymbol{C}_{\ell,i}$ and $\boldsymbol{D}_{\ell,\lceil i/(2n)\rceil}$ are always $\pi_{\ell,i} \cdot (1, \lambda_{\ell,\lceil i/(2n)\rceil})$ and $\mu_{\ell,\lceil i/(2n)\rceil} \cdot (\lambda_{\ell,\lceil i/(2n)\rceil}, -1)$, for each label $\ell \in [\![1;N]\!]$, and for $i = 1, \ldots, 2n$.
$c_{\ell,i} = z^b_{\ell,i} + \rho_\ell \sum_j v_{k,j}s_{i,j}t_{\lceil i/(2n)\rceil,j}$, and $c'_{\ell,i} = z^0_{\ell,i} + \rho_\ell \sum_j v_{k,j}s_{i,j}t_{\lceil i/(2n)\rceil,j}$.
$\boldsymbol{C}_{\ell,0} = \boldsymbol{\sigma}_\ell \cdot G_1$, $\boldsymbol{D}_{\ell,0} = \boldsymbol{\tau}_\ell \cdot G_2$, where

- From $\mathbf{G}_0$ to $\mathbf{G}_3$, $\boldsymbol{\sigma}_\ell \overset{\$}{\leftarrow} (\mathbb{Z}_q^*)^2$, $\boldsymbol{\tau}_\ell \overset{\$}{\leftarrow} \mathbb{Z}_q^2$, and $\boldsymbol{\rho}_\ell = (\sigma_{\ell,j}\tau_{\ell,j})_j$;
- From $\mathbf{G}_4$, $\boldsymbol{\sigma}_\ell \overset{\$}{\leftarrow} (\mathbb{Z}_q^*)^2$, $\boldsymbol{\rho}_\ell \leftarrow \rho_\ell \cdot \boldsymbol{v}$, for $\rho_\ell \overset{\$}{\leftarrow} \mathbb{Z}_q^*$, and $\boldsymbol{\tau}_\ell \leftarrow \rho_\ell \cdot (v_j/\sigma_{\ell,j})_j$.

Fig. 7. Sequence of Games for the Indistinguishability Security of the IPFE with Selector

- For the query $\mathsf{OKeyGen}(\boldsymbol{\alpha}_k = (\boldsymbol{\alpha}_{k,0}, \boldsymbol{\alpha}_{k,1}))$, we rewrite $\boldsymbol{\alpha}_k = (\alpha_{k,i})_i \in \mathbb{Z}_q^{2n}$, one outputs

$$\mathsf{dk}_{\boldsymbol{\alpha}_k} \leftarrow \boldsymbol{d}_k = \sum \alpha_{k,i}(s_{i,1}t_{\lceil i/(2n)\rceil,1}, s_{i,2}t_{\lceil i/(2n)\rceil,2})$$

- For a query $\mathsf{LoREncrypt}_1(\lambda_\ell, \boldsymbol{x}_\ell^0, \boldsymbol{x}_\ell^1)$, one first derives $\lambda_{\ell,i}$, for $i \in [\![1;2]\!]$, and then outputs $\mathbf{C}_\ell \leftarrow (\lambda_\ell, \boldsymbol{C}_{\ell,0}, (\boldsymbol{C}_{\ell,i})_i)$ where $C_{\ell,0,j} = \boldsymbol{\sigma}_\ell \cdot G_1$, for a random $\boldsymbol{\sigma}_\ell \overset{\$}{\leftarrow} (\mathbb{Z}_q^*)^2$, and, for random $\pi_{\ell,i} \overset{\$}{\leftarrow} \mathbb{Z}_q$,

$$\forall i \in [\![1;2n]\!], \qquad \boldsymbol{C}_{\ell,i} = (\pi_{\ell,i} \cdot (1, \lambda_{\ell,\lceil i/(2n)\rceil}), x_{\ell,i}^b, 0, (\sigma_{\ell,j}s_{i,j}, 0)_j, 0)_{\mathbb{B}}$$

- For a query $\mathsf{LoREncrypt}_2(\lambda_\ell, \boldsymbol{y}_\ell^0, \boldsymbol{y}_\ell^1)$, one first derives $\lambda_{\ell,i}$, for $i \in [\![1;2]\!]$, and then outputs $\mathbf{D}_\ell \leftarrow (\lambda_\ell, \boldsymbol{D}_{\ell,0}, (\boldsymbol{D}_{\ell,i})_i)$ where $D_{\ell,0,j} = \boldsymbol{\tau}_\ell \cdot G_2$, for a random $\boldsymbol{\tau}_\ell \overset{\$}{\leftarrow} \mathbb{Z}_q^2$, and, for random $\mu_{\ell,i} \overset{\$}{\leftarrow} \mathbb{Z}_q$,

$$\forall i \in [\![1;2]\!], \qquad \boldsymbol{D}_{\ell,i} = (\mu_{\ell,i} \cdot (\lambda_{\ell,i}, -1), y_{\ell,i}^b, 0, (\tau_{\ell,j}t_{i,j}, 0)_j, 0)_{\mathbb{B}^*}$$

Then we consider $S_0 = \Pr[\beta = b]$, where $\beta \overset{\$}{\leftarrow} \{0,1\}$ if for some $\boldsymbol{\alpha}_k$ asked to the $\mathsf{OKeyGen}$-oracle and some $(\boldsymbol{x}_\ell^0, \boldsymbol{x}_\ell^1)$, $(\boldsymbol{y}_\ell^0, \boldsymbol{y}_\ell^1)$ asked to the $\mathsf{LoREncrypt}$ oracle one has the inequality $\sum \alpha_{k,i}x_{\ell,i}^0 y_{\ell,i}^0 \neq \sum \alpha_{k,i}x_{\ell,i}^1 y_{\ell,i}^1$, which is a non-legitimate attack, or $\beta \leftarrow b'$, as this is a legitimate attack. Our goal is to show that S_0 is close to 1/2. In the following games, we alter the simulation of all the $\mathsf{LoREncrypt}$-oracles. We denote N the number of distinct λ_ℓ, but anticipate by programming at most $2Q$ values of $\lambda_{\ell,i}$, and K the number of $\mathsf{OKeyGen}$-queries.

Game $\mathbf{G}_1$: We consider $\mathsf{LoREncrypt}_2(\lambda_\ell, \boldsymbol{y}_\ell^0, \boldsymbol{y}_\ell^1)$ is always asked first. We apply the Alternative Product-Preserving Lemma 15 on the 3-rd and 4-th components $(x_{\ell,i}, 0)$ and $(y_{\ell,\lceil i/(2n)\rceil}, 0)$ for each ℓ and $\lceil i/(2n)\rceil$ to replace them into the preserved product $(0, x_{\ell,i}y_{\ell,\lceil i/(2n)\rceil})$ and $(0,1)$. We thus have $S_0 - S_1 \leq 8N \cdot (Qn(N+1)+2) \cdot \mathsf{Adv}^{\mathsf{sxdh}}(t)$.

Game $\mathbf{G}_2$: We apply again the Alternative Product-Preserving Lemma 15 on $(\sigma_{\ell,j}s_{i,j}, 0)$ and $(\tau_{\ell,j}t_{\lceil i/(2n)\rceil,j}, 0)$ to replace them into $(0,1)$ and the preserved product $(0, \sigma_{\ell,j}\tau_{\ell,j}s_{i,j}t_{\lceil i/(2n)\rceil,j})$, successively for $j = 1,2$ and each label: $S_1 - S_2 \leq 16N \cdot (Qn(N+1)+2) \cdot \mathsf{Adv}^{\mathsf{sxdh}}(t)$.

Game $\mathbf{G}_3$: We now formally replace $z_{\ell,i} \leftarrow x_{\ell,i}y_{\ell,\lceil i/(2n)\rceil}$, and $\tau_{\ell,j} \leftarrow \rho_{\ell,j}/\sigma_{\ell,j}$ for $j = 1,2$. This makes no difference: $S_2 = S_3$.

Game $\mathbf{G}_4$: We replace the random choice $\boldsymbol{\rho}_\ell \overset{\$}{\leftarrow} \mathbb{Z}_q^2$ by $\boldsymbol{\rho}_\ell$ in the span of $\boldsymbol{v} = (v_1, v_2)$, for $v_1 = 1$ but a random $v_2 \overset{\$}{\leftarrow} \mathbb{Z}_q$: $\boldsymbol{\rho}_\ell = \rho_\ell \cdot \boldsymbol{v}$ for $\rho_\ell \overset{\$}{\leftarrow} \mathbb{Z}_q$. The simulation of $\boldsymbol{C}_{\ell,0} = (\rho_{\ell,1}/\sigma_{\ell,1} \cdot G_1, \rho_{\ell,2}/\sigma_{\ell,2} \cdot G_1)$ does not require knowledge of $\boldsymbol{\rho}_\ell$, but just $\boldsymbol{\rho}_\ell \cdot G_1$, as the vector $\boldsymbol{\sigma}_\ell$ is known. The simulation of the $\boldsymbol{C}_{\ell,i}$ does not either, but just $\boldsymbol{\rho}_\ell \cdot G_1$ too.

This game is perfectly indistinguishable from the previous one if $\boldsymbol{\rho}_\ell$ is not colinear with $\boldsymbol{v}$ as $v_2 \overset{\$}{\leftarrow} \mathbb{Z}_q$ is random, and applying this transformation for the N indices, we get: $S_4 - S_3 \leq N/q$.

Game $\mathbf{G}_5$: Since the 6-th and the 8-th components of $\boldsymbol{D}_{\ell,k}$ vectors are all identical to 1, one can successively swap elements from the 6-th to the 4-th positions, and from the 8-th to the 4-th positions in $\boldsymbol{C}_{\ell,i}$ to get

$$c_{\ell,i} = z_{\ell,i}^b + \rho_\ell \sum_j v_{1,j}s_{i,j}t_{\lceil i/(2n)\rceil,j}$$

Applying the Swapping Lemma 5, one gets $S_4 - S_5 \leq 4 \cdot \mathsf{Adv}_{\mathbb{G}_1}^{\mathsf{ddh}}(t)$.

Game $\mathbf{G}_6$: Since the 6-th and the 8-th components of the $\boldsymbol{C}_{\ell,i}$ vectors are null, we use the Replication Lemma 7 twice to withdraw the 4-th component of the $\boldsymbol{D}_{\ell,k}$'s, 1, to their 6-th and then 8-th components, all the while adding zeros to the 4-th component in the $\boldsymbol{C}_{\ell,i}$'s. This is perfectly indistinguishable from the previous game: $S_5 - S_6 = 0$.

In this game, in the setting $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_t, e, G_1, G_2, q)$, one starts with random dual orthogonal matrices B and B^* of dimension 9 that define $(\mathbb{B}, \mathbb{B}^*)$, a random vector $\boldsymbol{v}$ with $v_1 = 1$, and a secret matrix $\underline{\mathbf{R}} \overset{\$}{\leftarrow} (\mathbb{Z}_q^2)^n$. We formally replace $r_{i,j} \leftarrow s_{i,j} t_{\lceil i/(2n) \rceil, j}$. The public parameters $\mathsf{PK} \leftarrow (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_t, e, G_1, G_2, q)$ are provided to the adversary.

- For the query $\mathsf{OKeyGen}(\boldsymbol{\alpha}_k)$, with $\boldsymbol{\alpha}_k \in \mathbb{Z}_q^n$, one outputs $\mathsf{dk}_{\boldsymbol{\alpha}_k} \leftarrow \boldsymbol{d}_k = (d_{k,j} = \sum \alpha_{k,i} r_{i,j})$;
- For $\ell \in [\![1; N]\!]$: a query $\mathsf{LoREncrypt}_2(\lambda_\ell, \boldsymbol{y}_\ell^0, \boldsymbol{y}_\ell^1)$ is first asked for the label λ_ℓ. One asks for $\mathsf{OEncrypt}_2(\lambda_\ell)$: one derives $\lambda_{\ell,i}$, for $i \in [\![1; 2]\!]$, and outputs $\mathbf{D}_\ell \leftarrow (\lambda_\ell, \boldsymbol{D}_{\ell,0}, (\boldsymbol{D}_{\ell,i})_i)$, where $\boldsymbol{D}_{\ell,0} = \boldsymbol{\sigma}_\ell \cdot G_2$, for random $\boldsymbol{\sigma}_\ell \overset{\$}{\leftarrow} \mathbb{Z}_q^2$, and for $i = 1, 2$, with random $\mu_{\ell,i} \overset{\$}{\leftarrow} \mathbb{Z}_q$,
$$\boldsymbol{D}_{\ell,i} = (\mu_{\ell,i} \cdot (1, \lambda_{\ell,i}), 0, 1, (0,0)_j, 0)_{\mathbb{B}^*}$$
When $\mathsf{LoREncrypt}_1(\lambda_\ell, \boldsymbol{x}_\ell^0, \boldsymbol{x}_\ell^1)$ is queried, one asks for $\mathsf{OEncrypt}_1(\lambda_\ell, \boldsymbol{z}_\ell)$, where $\boldsymbol{z}_\ell = (x_{\ell,i} y_{\ell, \lceil i/(2n) \rceil})_i$ and outputs $\boldsymbol{C}_\ell \leftarrow (\lambda_\ell, \boldsymbol{C}_{\ell,0}, (\boldsymbol{C}_{\ell,i})_i)$, where
$$\boldsymbol{C}_{\ell,0} = \rho_\ell \cdot (v_{1,1}/\sigma_{\ell,1}, v_{1,2}/\sigma_{\ell,2}) \cdot G_1$$
for random $\rho_\ell \overset{\$}{\leftarrow} \mathbb{Z}_q$, and for $i = 1, \ldots, 2n$, with random $\pi_{\ell,i} \overset{\$}{\leftarrow} \mathbb{Z}_q$,
$$\boldsymbol{C}_{\ell,i} = (\pi_{\ell,i} \cdot (1, \lambda_{\ell, \lceil i/(2n) \rceil}), 0, z_{\ell,i}^b + \rho_\ell \cdot \boldsymbol{r}_i \boldsymbol{v}^\top, (0,0)_j, 0)_{\mathbb{B}}$$

For a legitimate attack, we must have $\boldsymbol{z}_\ell^0 \boldsymbol{\alpha}_k^\top = \sum \alpha_{k,i} z_{\ell,i}^0 = \sum \alpha_{k,i} z_{\ell,i}^1 = \boldsymbol{z}_\ell^1 \cdot \boldsymbol{\alpha}_k^\top$ for all the functional key queries $\boldsymbol{\alpha}_k$: all $\boldsymbol{z}_\ell^0 - \boldsymbol{z}_\ell^1$ are orthogonal to all $\boldsymbol{\alpha}_k$.

Game $\mathbf{G}_7$: We eventually replace $\boldsymbol{z}_\ell^b$ by $\boldsymbol{z}_\ell^0$ in all the encryptions:
$$\forall \ell \in [\![1; N]\!] \qquad\qquad c'_{\ell,i} = z_{\ell,i}^0 + \rho_\ell \cdot \boldsymbol{r}_i \boldsymbol{v}^\top$$
Since b does not appear anywhere, $S_7 = 0$. But we also have to prove there is not much difference between $\mathbf{G}_7$ and $\mathbf{G}_6$: Using Lemma 16, one gets $S_7 - S_6 \leq 2N \cdot \mathsf{Adv}^{\mathsf{sxdh}}(t) + 2N/q$.

We can thus conclude that
$$S_0 - S_7 \leq 24N \cdot (Qn(N+1) + 2) \cdot \mathsf{Adv}^{\mathsf{sxdh}}(t) + 2N \cdot \mathsf{Adv}^{\mathsf{sxdh}}(t) + 4 \cdot \mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}_1}(t) + 3N/q$$
$$(2N \cdot (12Qn(N+1) + 25) + 4) \cdot \mathsf{Adv}^{\mathsf{sxdh}}(t) + 3N/q$$

Lemma 16. *$S_6 - S_7 \leq 2N \cdot \mathsf{Adv}^{\mathsf{sxdh}}(t) + 2N/q$.*

Proof. We proceed iteratively on each ℓ, as in [CDG+18]:

Game $\mathbf{G}_{6.\ell.0}$ This is the previous game, and we deal with the ℓ-th label.

Game $\mathbf{G}_{6.\ell.1}$ We use back $\boldsymbol{\rho}_\ell \overset{\$}{\leftarrow} \mathbb{Z}_q^2$, instead of the span of $\boldsymbol{v}$: $c'_{\ell,i} = z_{\ell,i}^b + \rho_\ell \cdot \boldsymbol{r}_i \boldsymbol{v}^\top$ is replaced by $c_{\ell,i}^1 = z_{\ell,i}^b + \boldsymbol{r}_i \boldsymbol{\rho}_\ell^\top$, with $S_{6.\ell.0} - S_{6.\ell.1} = 0$, using the same analysis as in game 4.

Game $\mathbf{G}_{6.\ell.2}$ We stop if eventually $\boldsymbol{\rho}_\ell$ is colinear to $\boldsymbol{v}$: $S_{6.\ell.1} - S_{6.\ell.2} \leq 1/q$. Otherwise, denoting $\boldsymbol{v}'$ a unitary vector orthogonal to $\boldsymbol{v}$, we can write $\boldsymbol{\rho}_\ell = c_1 \boldsymbol{v} + c_2 \boldsymbol{v}'$, with $c_2 \neq 0 \bmod q$.

Game $\mathbf{G}_{6.\ell.3}$ We replace $\boldsymbol{z}_\ell^b$ by $\boldsymbol{z}_\ell^0$ in the encryption: $c_{\ell,i}^1 = z_{\ell,i}^b + \boldsymbol{r}_i \boldsymbol{\rho}_\ell^\top$ is replaced by $c_{\ell,i}^2 = z_{\ell,i}^0 + \boldsymbol{r}_i \boldsymbol{\rho}_\ell^\top$. As in [CDG+18], we can use the complexity leveraging, by guessing $(\boldsymbol{z}_\ell^0, \boldsymbol{z}_\ell^1)$ to show this makes no difference, by building an adversary that guesses $(\boldsymbol{z}_\ell^0, \boldsymbol{z}_\ell^1)$. We then note S^* the success of this new adversary that makes both a correct guess on $(\boldsymbol{z}_\ell^0, \boldsymbol{z}_\ell^1)$ and a correct guess for b with a legitimate attack.

- Step 1: one guesses $(\boldsymbol{z}_\ell^0, \boldsymbol{z}_\ell^1)$, which can either be in $(\mathbb{Z}_q)^{2n}$ or empty if the second encryption query is not asked. There are thus $q^{2n} + 1$ possibilities. If the guess is correct, one does not change anything, if this is incorrect, one stops and outputs a random bit. One can note that the advantage in this game is $S_1^* = S_{6.\ell.2}/(q^{2n} + 1)$, as the good guess is exactly $1/(q^{2n} + 1)$, even in case of correct guess of b in a legitimate attack;

- Step 2: we can replace $\boldsymbol{r}_i$ by $\boldsymbol{u}_i = \boldsymbol{r}_i + \gamma(z^b_{\ell,i} - z^0_{\ell,i}) \cdot \boldsymbol{v}_3$, for a random $\gamma \xleftarrow{\$} \mathbb{Z}_q$, as $\underline{\mathbf{R}} = (\boldsymbol{r}_i)_i \xleftarrow{\$} (\mathbb{Z}_q^2)^n$ is random and independent of $(\boldsymbol{z}^0_\ell, \boldsymbol{z}^1_\ell)$. We have to check this does not impact the view of the adversary:
  - the keys, for all k, are generated as

$$\begin{aligned}\boldsymbol{d}_k &= \sum_i \alpha_{k,i} \cdot \boldsymbol{u}_i = \sum_i \alpha_{k,i} \cdot \Big(\boldsymbol{r}_i + \gamma(z^b_{\ell,i} - z^0_{\ell,i}) \cdot \boldsymbol{v}'\Big)\\ &= \sum_i \alpha_{k,i} \cdot \boldsymbol{r}_i + \gamma \cdot \left(\sum_i \alpha_{k,i} \cdot (z^b_{\ell,i} - z^0_{\ell,i})\right) \cdot \boldsymbol{v}' = \sum_i \alpha_{k,i} \cdot \boldsymbol{r}_i\end{aligned}$$

  as for a legitimate attack, $\sum_i \alpha_{k,i}(z^b_{\ell,i} - z^0_{\ell,i}) = 0$.
  - the ciphertexts, for all $j \neq \ell$ are all generated with $\boldsymbol{\rho}_j = \rho_j \cdot \boldsymbol{v}$. So, with $b_j = 0$, for $j < \ell$, and $b_j = b$ for $j > \ell$:

$$\begin{aligned}c_{j,i} &= z^{b_j}_{j,i} + \rho_j \cdot \boldsymbol{u}_i \cdot \boldsymbol{v}^\top = z^{b_j}_{j,i} + \rho_j \cdot \Big(\boldsymbol{r}_i + \gamma(z^b_{\ell,i} - z^0_{\ell,i}) \cdot \boldsymbol{v}'\Big) \cdot \boldsymbol{v}^\top\\ &= z^{b_j}_{j,i} + \rho_j \cdot \boldsymbol{r}_i \cdot \boldsymbol{v}^\top\end{aligned}$$

  - excepted $c_{\ell,i}$:

$$\begin{aligned}c_{\ell,i} &= z^b_{\ell,i} + \boldsymbol{u}_i \cdot \boldsymbol{\rho}_\ell^\top\\ &= z^b_{\ell,i} + \boldsymbol{u}_i \cdot \boldsymbol{\rho}_\ell^\top + \gamma(z^b_{\ell,i} - z^0_{\ell,i}) \cdot \boldsymbol{v}' \cdot (c_1 \boldsymbol{v}^\top + c_2 \boldsymbol{v}'^\top)\\ &= z^b_{\ell,i} + \boldsymbol{u}_i \cdot \boldsymbol{\rho}_\ell^\top + c_2\gamma(z^b_{\ell,i} - z^0_{\ell,i}) \cdot \boldsymbol{v}' \cdot \boldsymbol{v}'^\top\\ &= z^b_{\ell,i} + c_2\gamma(z^b_{\ell,i} - z^0_{\ell,i}) + \boldsymbol{u}_i \cdot \boldsymbol{\rho}_\ell^\top = z^0_{\ell,i} + \boldsymbol{u}_i \cdot \boldsymbol{\rho}_\ell^\top\end{aligned}$$

  if $\gamma = -1/c_2$, as $\boldsymbol{v}'$ is a unitary vector. Then $S^*_2 = S^*_1$.

  And it is straightforward that $S^*_2 = S_{6.\ell.3}/(q^{2n} + 1)$. Hence, $S_{6.\ell.3} = S_{6.\ell.2}$.

Game $\mathbf{G}_{6.\ell.4}$ After having ignored linear vectors, we force $\boldsymbol{\rho}_\ell$ to be in the span of $\boldsymbol{v}$: $S_{6.\ell.3} - S_{6.\ell.4} \leq 1/q$, as above.

D Proof of Lemma 15

It follows the steps in the proof of general Product-Preserving Lemma, though here, for each label λ, there are n $\mathbf{C}_j$ ciphertexts and one $\mathbf{D}$ ciphertext, hence the steps presented in Figure 8.

Proof. **Game $\mathbf{G}_0$:** We start from the original distribution, for random $\pi_{i,j}, \mu_i \xleftarrow{\$} \mathbb{Z}_q$, but any $\lambda_i, x_{i,j}, y_i, a_{i,j}, b_i$, for $i \in [\![1; N]\!]$ and $j \in [\![1; n]\!]$:

$$\mathbf{C}_{i,j} = (\pi_{i,j} \cdot (1, \lambda_i), x_{i,j}, a_{i,j}, 0)_{\mathbb{B}} \qquad \mathbf{D}_i = (\mu_i \cdot (\lambda_i, -1), y_i, b_i, 0)_{\mathbb{B}^*}$$

and

$$\mathbf{C}^*_j = (\pi^*_j(1, \lambda^*), x^*_j, 0, 0)_{\mathbb{B}} \qquad \mathbf{D}^* = (\mu^*(\lambda^*, -1), y^*, 0, 0)_{\mathbb{B}^*}$$

for random π^*_j, μ^* but any λ^*, x^*_j, y^*. And we denote S_0 the event that the adversary eventually outputs 1, when distinguishing the distributions of $\mathbf{C}^*_j, \mathbf{D}^*$. One can note that in this game, we are with the input distribution (0), where the adversary can choose $(\lambda_i, x_{i,j}, y_i, a_{i,j}, b_i)$ for all i, j, and λ^*, x^*_j, y^*, but λ^* is distinct from each λ_i, and all the labels being among Q possible values.

Game $\mathbf{G}_1$: As the 5-th components in the $\mathbf{C}_{i,j}$, $\mathbf{C}^*$ vectors are all 0, one can duplicate the 3-rd components of $\mathbf{D}_i$, $\mathbf{D}^*$ vectors at the 5-th position, applying the Replication Lemma 7. This is perfectly indistinguishable from the previous game: $S_0 = S_1$.

$\mathbf{G}_0$		$\mathbf{C}_{i,j} = ($	$\pi_{i,j}(1, \lambda_i)$	$\mid$	$x_{i,j}$	$a_{i,j}$	$\mid$	0	$)$
$\forall i \in [\![1; N]\!]$,		$\mathbf{D}_i = ($	$\mu_i(\lambda_i, -1)$	$\mid$	y_i	b_i	$\mid$	0	$)$
$j \in [\![1; n]\!]$:		$\mathbf{C}^*_j = ($	$\pi^*_j(1, \lambda^*)$	$\mid$	x^*_j	0	$\mid$	0	$)$
		$\mathbf{D}^* = ($	$\mu^*(\lambda^*, -1)$	$\mid$	y^*	0	$\mid$	0	$)$
$\mathbf{G}_1$	$\forall i, j$	$\mathbf{C}_{i,j} = ($	$\pi_{i,j}(1, \lambda_i)$	$\mid$	$x_{i,j}$	$a_{i,j}$	$\mid$	0	$)$
Replication		$\mathbf{D}_i = ($	$\mu_i(\lambda_i, -1)$	$\mid$	y_i	b_i	$\mid$	y_i	$)$
		$\mathbf{C}^*_j = ($	$\pi^*_j(1, \lambda^*)$	$\mid$	x^*_j	0	$\mid$	0	$)$
		$\mathbf{D}^* = ($	$\mu^*(\lambda^*, -1)$	$\mid$	y^*	0	$\mid$	y^*	$)$
$\mathbf{G}_2$	$\forall i, j$	$\mathbf{C}_{i,j} = ($	$\pi_{i,j}(1, \lambda_i)$	$\mid$	$x_{i,j}$	$a_{i,j}$	$\mid$	0	$)$
Swapping		$\mathbf{D}_i = ($	$\mu_i(\lambda_i, -1)$	$\mid$	y_i	b_i	$\mid$	y_i	$)$
		$\mathbf{C}^*_j = ($	$\pi^*_j(1, \lambda^*)$	$\mid$	0	0	$\mid$	x^*_j	$)$
		$\mathbf{D}^* = ($	$\mu^*(\lambda^*, -1)$	$\mid$	y^*	0	$\mid$	y^*	$)$
$\mathbf{G}_3$	$\forall i, j$	$\mathbf{C}_{i,j} = ($	$\pi_{i,j}(1, \lambda_i)$	$\mid$	$x_{i,j}$	$a_{i,j}$	$\mid$	0	$)$
Indexing		$\mathbf{D}_i = ($	$\mu_i(\lambda_i, -1)$	$\mid$	y_i	b_i	$\mid$	0	$)$
		$\mathbf{C}^*_j = ($	$\pi^*_j(1, \lambda^*)$	$\mid$	0	0	$\mid$	x^*_j	$)$
		$\mathbf{D}^* = ($	$\mu^*(\lambda^*, -1)$	$\mid$	y^*	0	$\mid$	y^*	$)$
$\mathbf{G}_4$	$\forall i, j$	$\mathbf{C}_{i,j} = ($	$\pi_{i,j}(1, \lambda_i)$	$\mid$	$x_{i,j}$	$a_{i,j}$	$\mid$	$x_{i,j}$	$)$
Replication		$\mathbf{D}_i = ($	$\mu_i(\lambda_i, -1)$	$\mid$	y_i	b_i	$\mid$	0	$)$
		$\mathbf{C}^*_j = ($	$\pi^*_j(1, \lambda^*)$	$\mid$	0	0	$\mid$	x^*_j	$)$
		$\mathbf{D}^* = ($	$\mu^*(\lambda^*, -1)$	$\mid$	0	0	$\mid$	y^*	$)$
$\mathbf{G}_5$	$\forall i, j$	$\mathbf{C}_{i,j} = ($	$\pi_{i,j}(1, \lambda_i)$	$\mid$	$x_{i,j}$	$a_{i,j}$	$\mid$	0	$)$
Indexing		$\mathbf{D}_i = ($	$\mu_i(\lambda_i, -1)$	$\mid$	y_i	b_i	$\mid$	0	$)$
		$\mathbf{C}^*_j = ($	$\pi^*_j(1, \lambda^*)$	$\mid$	0	0	$\mid$	x^*_j	$)$
		$\mathbf{D}^* = ($	$\mu^*(\lambda^*, -1)$	$\mid$	0	0	$\mid$	y^*	$)$
$\mathbf{G}_6$	$\forall i, j$	$\mathbf{C}_{i,j} = ($	$\pi_{i,j}(1, \lambda_i)$	$\mid$	$x_{i,j}$	$a_{i,j}$	$\mid$	0	$)$
Quotient		$\mathbf{D}_i = ($	$\mu_i(\lambda_i, -1)$	$\mid$	y_i	b_i	$\mid$	0	$)$
for $y^* \neq 0$		$\mathbf{C}^*_j = ($	$\pi^*_j(1, \lambda^*)$	$\mid$	0	0	$\mid$	$x^*_j y^*$	$)$
		$\mathbf{D}^* = ($	$\mu^*(\lambda^*, -1)$	$\mid$	0	0	$\mid$	1	$)$
$\mathbf{G}_7$	$\forall i, j$	$\mathbf{C}_{i,j} = ($	$\pi_{i,j}(1, \lambda_i)$	$\mid$	$x_{i,j}$	$a_{i,j}$	$\mid$	0	$)$
Subspace		$\mathbf{D}_i = ($	$\mu_i(\lambda_i, -1)$	$\mid$	y_i	b_i	$\mid$	0	$)$
for $y^* = 0$		$\mathbf{C}^*_j = ($	$\pi^*_j(1, \lambda^*)$	$\mid$	0	0	$\mid$	$x^*_j y^*$	$)$
		$\mathbf{D}^* = ($	$\mu^*(\lambda^*, -1)$	$\mid$	0	0	$\mid$	1	$)$
$\mathbf{G}_8$	$\forall i, j$	$\mathbf{C}_{i,j} = ($	$\pi_{i,j}(1, \lambda_i)$	$\mid$	$x_{i,j}$	$a_{i,j}$	$\mid$	$-a_{i,j}$	$)$
Replication		$\mathbf{D}_i = ($	$\mu_i(\lambda_i, -1)$	$\mid$	y_i	b_i	$\mid$	0	$)$
		$\mathbf{C}^*_j = ($	$\pi^*_j(1, \lambda^*)$	$\mid$	0	0	$\mid$	$x^*_j y^*$	$)$
		$\mathbf{D}^* = ($	$\mu^*(\lambda^*, -1)$	$\mid$	0	1	$\mid$	1	$)$
$\mathbf{G}_9$	$\forall i, j$	$\mathbf{C}_{i,j} = ($	$\pi_{i,j}(1, \lambda_i)$	$\mid$	$x_{i,j}$	$a_{i,j}$	$\mid$	0	$)$
Indexing		$\mathbf{D}_i = ($	$\mu_i(\lambda_i, -1)$	$\mid$	y_i	b_i	$\mid$	b_i	$)$
		$\mathbf{C}^* = ($	$\pi^*(1, \lambda^*)$	$\mid$	0	0	$\mid$	$x^*_j y^*$	$)$
		$\mathbf{D}^* = ($	$\mu^*(\lambda^*, -1)$	$\mid$	0	1	$\mid$	1	$)$
$\mathbf{G}_{10}$	$\forall i, j$	$\mathbf{C}_{i,j} = ($	$\pi_{i,j}(1, \lambda_i)$	$\mid$	$x_{i,j}$	$a_{i,j}$	$\mid$	0	$)$
Swapping		$\mathbf{D}_i = ($	$\mu_i(\lambda_i, -1)$	$\mid$	y_i	b_i	$\mid$	b_i	$)$
		$\mathbf{C}^*_j = ($	$\pi^*_j(1, \lambda^*)$	$\mid$	0	$x^*_j y^*$	$\mid$	0	$)$
		$\mathbf{D}^* = ($	$\mu^*(\lambda^*, -1)$	$\mid$	0	1	$\mid$	1	$)$
$\mathbf{G}_{11}$	$\forall i, j$	$\mathbf{C}_{i,j} = ($	$\pi_{i,j}(1, \lambda_i)$	$\mid$	$x_{i,j}$	$a_{i,j}$	$\mid$	0	$)$
Replication		$\mathbf{D}_i = ($	$\mu_i(\lambda_i, -1)$	$\mid$	y_i	b_i	$\mid$	0	$)$
		$\mathbf{C}^*_j = ($	$\pi^*_j(1, \lambda^*)$	$\mid$	0	$x^*_j y^*$	$\mid$	0	$)$
		$\mathbf{D}^* = ($	$\mu^*(\lambda^*, -1)$	$\mid$	0	1	$\mid$	0	$)$

Fig. 8. Sequence of Games for the Alternative Product-Preserving Lemma

Game $\mathbf{G}_2$: Since the 3-rd and 5-th components of $\mathbf{D}_i$, $\mathbf{D}^*$ vectors are all identical, one can swap x_j^*'s from the 3-rd to the 5-th positions in $\mathbf{C}_j^*$'s, applying the Swapping Lemma 5 with the first two components for the required randomness: $S_1 - S_2 \leq 2 \cdot \mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}_1}(t)$.

Game $\mathbf{G}_3$: For any i, the 5-th component of the $\mathbf{C}_{i,j}$'s is 0. Applying the Indexing Lemma 6, on $(\mathbb{B}^*, \mathbb{B})$, between λ^* and the $\lambda_i \neq \lambda^*$, iteratively, one replaces each y_i in the 5-th component of the $\mathbf{D}_i$'s by 0, and gets $S_2 - S_3 \leq 2NQ \cdot \mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}_2}(t)$.

Game $\mathbf{G}_4$: One now removes y^* from the 3-rd component of $\mathbf{D}^*$ using the Replication Lemma 7: replicating negatively the 5-th components at the 3-rd positions of $\mathbf{D}$'s vectors, one subtracts y^* in $\mathbf{D}^*$, and 0 in other vectors, but adds the 3-rd component to the 5-th in the $\mathbf{C}$'s vectors. The two games are perfectly indistinguishable: $S_3 = S_4$.

Game $\mathbf{G}_5$: For any j, the 5-th component of $\mathbf{D}_j$ is 0. Applying the Indexing Lemma 6, on $(\mathbb{B}, \mathbb{B}^*)$, between λ^* and the $\lambda_j \neq \lambda^*$, iteratively, one replaces each $x_{i,j}$ in the 5-th components of the $\mathbf{C}_{i,j}$'s by 0, and gets $S_4 - S_5 \leq 2nNQ \cdot \mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}_1}(t)$.

Game $\mathbf{G}_6$: One replaces each pair (x_j^*, y^*) with $(x_j^* y^*, 1)$ in the 5-th coordinates of $\mathbf{C}_j^*$ and $\mathbf{D}^*$. To this aim, one makes a random guess $Y \leftarrow \mathbb{Z}_q$ on y^*, and defines the matrices, if $Y \neq 0$:

$$B = \left(1/Y\right)_5 \qquad B' = \left(Y\right)_5 \qquad \mathbb{B} = B \cdot \mathbb{U} \qquad \mathbb{B}^* = B' \cdot \mathbb{U}^*$$

or no change otherwise. If during the execution, $y^* \neq Y$, one aborts, otherwise the two games are perfectly indistinguishable in case of correct guess: $S_5/q = S_6/q$, hence $S_5 = S_6$.

Game $\mathbf{G}_7$: In case $y^* = 0$, then one applies the subspace indistinguishability theorem 3 twice, to shift the x_j^* 's to zero, using the null fifth coordinate of $\mathbf{D}^*$ and the randomness in the first coordinates of the $\mathbf{C}_j^*$'s, and then to shift the fifth coordinate of $\mathbf{D}^*$ to 1 using one $\mathbf{C}_j^*$'s now null fifth coordinate and the randomness in the first coordinates of $\mathbf{D}^*$. In this case: $S_6 - S_7 \leq 2\mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}_1}(t) + 2\mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}_2}(t)$.

Game $\mathbf{G}_8$: One now duplicates the 1 from the 5-th component of $\mathbf{D}^*$ to the 4-th using the Replication Lemma 7 (and 0 is added to the 4-th components of the $\mathbf{D}_i$'s). This also negatively adds the 4-th components at the 5-th positions of $\mathbf{C}_{i,j}$ and $\mathbf{C}_j^*$ vectors. The two games are perfectly indistinguishable: $S_7 = S_8$.

Game $\mathbf{G}_9$: One applies twice the Indexing Lemma 6, between λ^* and the $\lambda_j \neq \lambda^*$, iteratively, to first replaces each $-a_{i,j}$ in the 5-th components of the $\mathbf{C}_{i,j}$'s by 0, and then the 0 in the 5-th components of the $\mathbf{D}_i$'s by b_i. One gets $S_8 - S_9 \leq 2QnN\mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}_1}(t) + 2Qn\mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}_2}(t)$.

Game $\mathbf{G}_{10}$: One now swaps the 5-th components $x_j^* y^*$ and the 4-th components 0 of the $\mathbf{C}_j^*$'s. This easily works as all the $\mathbf{D}_i$ and $\mathbf{D}^*$ vectors have identical 4-th and 5-th components, using the Swapping Lemma 5, with the first two components for randomness: $S_9 - S_{10} \leq 2 \cdot \mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}_1}(t)$.

Game $\mathbf{G}_{11}$: One then withdraws the 4-th components from the 5-th components in the $\mathbf{D}_j$'s and the $\mathbf{D}^*$, while the null 5-th components of $\mathbf{C}_{i,j}$'s and $\mathbf{C}_j^*$'s are added to the 4-th components, using the Replication Lemma 7. This game is perfectly indistinguishable from the previous one: $S_{10} - S_{11} = 0$.

The global probability difference is bounded by $0 + 2 \cdot \mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}_1}(t) + 2NnQ \cdot \mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}_2}(t) + 0 + 2nQ\mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}_1}(t) + 2 \cdot (\mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}_1}(t) + \mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}_2}(t)) + 0 + 2NnQ \cdot \mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}_1}(t) + 2nQ\mathsf{Adv}_{\mathbb{G}_1} + 2 \cdot \mathsf{Adv}_{\mathbb{G}_1} + 0 = 2Qn \cdot (N+1) \cdot \mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}_1}(t) + 2Qn \cdot (N+1) \cdot \mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}_2}(t) + 6 \cdot \mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}_1}(t) + 2 \cdot \mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}_2}(t) \leq 4 \cdot (Qn(N + 1) + 2) \cdot \mathsf{Adv}^{\mathsf{sxdh}}(t)$.

E Scripts for the Money-Laundering Detection Application

E.1 Parameters script for the Money-Laundering Application

```
from math import *


def get_m(ln, lN, thres_proba):
    return ceil(2 * ln + lN - log2(2**ln - 2 * log(thres_proba)))


def get_p(ln, lN, m, nprime):
    q = 1 - 2 ** (ln - lN)
    q2 = 1 - nprime / 2**lN
    eh = 2 ** (lN - m)
    return (1 - 2 ** (-m)) * (q ** (eh-1) * (2 - q**(eh-1)) * (q2 + q**(eh-1) - q2 * q**(eh-1))**2) ** (2**lN)


def get_approx_p(ln, lN, m):
    q = 1 - 2**(ln-lN)
    eh = 2 ** (lN-m)
    return (q**eh * (2 - q ** eh))**(2**lN)

# relative entropy


def D(a, p):
    return a * log(a / p) + (1 - a) * log((1 - a) / (1 - p))


def Hoeffding(r, p):
    return 1 - exp(-2 * r * p ** 2)


def Chernoff(r, p):
    tau = min(floor(r*p)+1, ceil(r/2))
    if tau < 2:
        if tau == 1:
            # ”Returning the Hoeffding bound instead of the Chernoff bound.”
            return 1 - exp(-2 * r * p**2)
        print("Can’t apply the Chernoff bound if tau is null! ")
        return 0
    return 1 - (r * p / (tau-1))**(tau-1) * exp(tau - 1 - r * p)


def get_r(p, confidence_min):
    r = 3
    while (Chernoff(r, p) < confidence_min):
        r += 1
        if r == 300:
            print("Can’t find an appropriate r! ")
            break
    return r


def calculate_values(confidence_min, lNi, lN, ln, thres_proba, nprime):
    for k in range(len(lN)):
        m = get_m(ln[k], lN[k], thres_proba)
        # # tests if we want to try other m values than from the birthday paradox approximation:
        # if k == 1:
        # m = 22
        # print(”p_B approximation: ”,
        # round(exp(-2**ln[k]*(2**(ln[k]+lN[k])-2**m)/2**(m+1)), 2))
        pA = get_p(ln[k], lN[k], m, nprime)
        approxA = get_approx_p(ln[k], lN[k], m)
        pB = pA ** 2
        approxpB = approxpA ** 2
        r = get_r(pB, confidence_min)
        # we also try empirical r values:
        if k == 3:
            r = 15
        elif k == 4:
            r = 25
        elif k == 5:
            r = 20
        # tau = min(r*pB+1, ceil(r/2))
        new_cfdence = Chernoff(r, pB)
        C0 = 4 * 2 ** (lN[k] + 2 * lNi[k])
        CH = r * 4 * 2 ** (lN[k] + 2 * m)

        print(
            ("\\rowcolor{green!10!white}" if CH <=
             C0 else "\\rowcolor{orange!20!white}"),
            ("$2^{" +
             str(ln[k]) +
             "}$ & $2^{" +
             str(lN[k]) +
             "}$" if k < 0 else str(round(2**ln[k], 2)) + " & $10^{" + str(round(log10(2**lN[k]))) + "}$ "),
            "& ",
            m,
            " & ",
            "{:.2}".format(pA),
            # ” & ”,
            # ”{:.2}”.format(approxpA),
            " & ",
            "{:.2}".format(pB),
            # ” & ”,
            # ”{:.2}”.format(approxpB),
            (" & $2^{" +
             str(lNi[k]) +
             "}$ " if k < 0 else " & " + str(round(2**lNi[k])) + " "), # ” & $10ˆ{” + str(round(log10(2**lNi[k]))) +
                 ↪  ”}$ ”),
            "& ",
            r,
            " & ",
            "{:.2}".format(new_cfdence),
            " & ",
            "{:.1E}".format(C0),
            " & ",
            "{:.1E}".format(CH),
            " \\\\"
        )


confidence_min = 0.95
thres_birthday_paradox_proba = 0.5
nprime = 1
lNi = [ # 25, 25, 19, 16, 10,
    log2(200), log2(2000), log2(
        2000), log2(200), log2(2000), log2(2000)]
lN = [ # 30, 30, 20, 20, 15,
    log2(1000), log2(10000), log2(
        10000), log2(1000), log2(10000), log2(10000)]
ln = [ # 4, 5, 4, 4, 4,
    log2(0.36), log2(0.11), log2(
        0.14), log2(0.36), log2(0.11), log2(0.14)]

calculate_values(confidence_min, lNi, lN, ln,
                 thres_birthday_paradox_proba, nprime)
```

E.2 Experiments script for the Money-Laundering Application

```


from math import ceil
from random import randint, seed
import numpy as np
from numpy import linalg as LA
from joblib import Parallel, delayed

# parameters.

# Total number of nodes in the system:
N = 10000
# Number of nodes owned by bank 1
N1 = 100
# Number of nodes owned by bank 2
```

```
N2 = N1

# Total number of transactions patterns
# because each standard pattern will lead to two nodes with a transaction and each money laundering pattern to two
    ↪ transactions between three nodes.
nb_standard_transaction_ptns = 200
nb_money_laundering_trans_ptns = 200

# for a given pair of nodes with a transaction pattern, the number of time a transaction happen between them
std_trans_cnt_min = 1
std_trans_cnt_max = 3
monlaun_trans_cnt_min = 6
monlaun_trans_cnt_max = 12
# money laundering number of neighbors for each ML pattern
monlaun_interm_min = 1
monlaun_interm_max = 5

# nodes average connectivity i. e. mean number of other nodes they are having transactions with, for incoming xor
    ↪ outgoing edges.
n = (nb_standard_transaction_ptns + nb_money_laundering_trans_ptns *
    (monlaun_interm_max+monlaun_interm_min)/2 * 2) / N
print("n: ", n)

# transaction amounts
mon_laun_amount_input = 10000
mon_laun_amount_output = 9900
std_trans_min = 1
std_trans_max = 5000

# hashing parameters
# number of hash functions:
nb_hash = 20
print("r: ", nb_hash)
# threshold for selection of suspicious node in hash repetitions:
tau = 19
print("tau: ", tau)
# bit size of the space hashed into:
m = 8

# Similarity threshold for detection:
theta = 0.2
hash_theta = theta
print("hash_theta: ", hash_theta)

# returns an upper triangular matrix with similarity indexes.


def calculate_similarity_matrix(in_normed_graph_matrix, out_normed_graph_matrix):
    # number of nodes:
    M = np.size(in_normed_graph_matrix, 1)
    # number of nodes in original graph:
    N = np.size(in_normed_graph_matrix, 0)
    # columns:
    if N != np.size(out_normed_graph_matrix, 1) or M != np.size(out_normed_graph_matrix, 0):
        "Wrong dimensions!"
    sim_matrix = np.zeros([M, M])
    for i in range(M-1):
        for j in range(i+1, M):
            sim_matrix[i, j] = np.inner(out_normed_graph_matrix[i, :], out_normed_graph_matrix[j, :]) * \
                np.inner(in_normed_graph_matrix[:, i],
                         in_normed_graph_matrix[:, j])
    # for i in range(M-1):
    # sim_matrix[i, i+1:] = Parallel(n_jobs=10)(delayed(get_sim_mat_coeff)(i, j, out_normed_graph_matrix,
    # in_normed_graph_matrix) for j in range(i+1, M))
    return sim_matrix


def get_sim_mat_coeff(i, j, out_normed_graph_matrix, in_normed_graph_matrix):
    return np.inner(out_normed_graph_matrix[i, :], out_normed_graph_matrix[j, :]) * \
        np.inner(in_normed_graph_matrix[:, i],
                 in_normed_graph_matrix[:, j])

# returns the list of suspects from the similarity matrix:
```

```
def list_suspects(sim_matrix, thres):
    return np.unique(np.where(np.logical_and(1 >= sim_matrix, sim_matrix >= thres)))


def get_result_rates(true_list, suspects_list, N):
    ML_size = np.size(true_list)
    TP = np.size(np.intersect1d(true_list, suspects_list))
    FP = np.size(suspects_list) - TP
    FN = ML_size - TP
    TN = N - TP - FP - FN
    print("TP: ", TP, " FP: ", FP, " FN: ", FN, " TN: ", TN)
    print("TP rate (sensitivity and recall): ", TP/ML_size, " FP rate (fall-out): ",
          FP/(FP+TN), " FN rate: ", FN/(FN+TP), " TN rate: ", TN/(TN+FP))


# creation of the original graph.
# represented as a matrix in which coefficient [i,j] give the amount of transactions from i to j.
original_graph = np.zeros([N, N])
money_launderers = []
monlaun_intermediates = []
# create standard transactions
for k in range(nb_standard_transaction_ptns):
    trans_cnt = randint(std_trans_cnt_min, std_trans_cnt_max)
    sender = randint(0, N-1)
    receiver = sender
    while receiver == sender:
        receiver = randint(0, N-1)
    for l in range(trans_cnt):
        original_graph[sender,
                       receiver] += randint(std_trans_min, std_trans_max)
# create money laundering transactions
for k in range(nb_money_laundering_trans_ptns):
    trans_cnt = randint(monlaun_trans_cnt_min, monlaun_trans_cnt_max)
    nb_intermediates = randint(monlaun_interm_min, monlaun_interm_max)
    sender = randint(0, N-1)
    if sender not in money_launderers:
        money_launderers.append(sender)
    interms = [sender for i in range(nb_intermediates)]
    receiver = sender
    for i in range(nb_intermediates):
        interm = sender
        while interm in interms or interm == sender:
            interm = randint(0, N-1)
        interms[i] = interm
        if interm not in money_launderers:
            money_launderers.append(interm)
        if interm not in monlaun_intermediates:
            monlaun_intermediates.append(interm)
    while receiver == sender or receiver in interms:
        receiver = randint(0, N-1)
    if receiver not in money_launderers:
        money_launderers.append(receiver)
    for l in range(trans_cnt):
        for i in range(nb_intermediates):
            original_graph[sender,
                           interms[i]] += mon_laun_amount_input/nb_intermediates
            original_graph[interms[i],
                           receiver] += mon_laun_amount_output/nb_intermediates
# norm the graph:
out_normed_graph = np.array([(original_graph[i, :]/LA.norm(original_graph[i, :], 2) if LA.norm(original_graph[i, :], 2) !=
    ↪ 0 else original_graph[i, :])
                              for i in range(np.size(original_graph, 0))])
in_normed_graph = np.array([(original_graph[:, j]/LA.norm(original_graph[:, j], 2) if LA.norm(original_graph[:, j], 2) != 0
    ↪ else original_graph[:, j])
                             for j in range(np.size(original_graph, 1))])
print("The data has been generated. ")
money_launderers = np.unique(money_launderers)
monlaun_intermediates = np.unique(monlaun_intermediates)
print("Money laundering nodes: ", money_launderers)
print("Money laundering intermediates: ", monlaun_intermediates)
```

```
# do the hashing technique
hashed_in = np.zeros([nb_hash, N, 2**m])
hashed_out = np.zeros([nb_hash, 2**m, N])
h_sim_mat = np.zeros([nb_hash, 2**m, 2**m])
for h in range(nb_hash):
    for n in range(N):
        seed((h, n))
        hashed_to = randint(0, 2**m−1)
        # print(hashed_to)
        hashed_in[h][:, hashed_to] += in_normed_graph[:, n]
        hashed_out[h][hashed_to, :] += out_normed_graph[n, :]
    h_sim_mat[h] = calculate_similarity_matrix(hashed_in[h], hashed_out[h])
    # print(h_sim_mat[h])
hash_suspects = []
for n in range(N):
    cnt = 0
    for h in range(nb_hash):
        seed((h, n))
        hashed_to = randint(0, 2**m−1)
        if (np.size(np.where(np.logical_and(hash_theta <= h_sim_mat[h][hashed_to, :], h_sim_mat[h][hashed_to, :] <= 1)))
             ↪  >= 1 or np.size(np.where(np.logical_and(hash_theta <= h_sim_mat[h][:, hashed_to], h_sim_mat[h][:,
             ↪  hashed_to] <= 1))) >= 1):
            cnt += 1
    if cnt >= tau:
        hash_suspects.append(n)
hash_suspects = np.unique(hash_suspects)
print("Suspects were selected with the hashing technique. ")
print("hash_suspects: ", hash_suspects)
print(get_result_rates(monlaun_intermediates, hash_suspects, N))

# print(original_graph)
# print(in_normed_graph)
# print(out_normed_graph)
sim_mat = calculate_similarity_matrix(in_normed_graph, out_normed_graph)
print("The similarity matrix has calculated. ")
soltani_suspects = list_suspects(sim_mat, theta)
print("Suspects from the standard method have been selected. ")
print("Soltani suspects: ", soltani_suspects)
print(get_result_rates(monlaun_intermediates, soltani_suspects, N))
```