Unbounded Inner-Product Functional Encryption with Succinct Keys

Edouard Dufour-Sans[1,2] and David Pointcheval[1,2]

[1] DIENS, École normale supérieure, CNRS, PSL University, Paris, France
[2] INRIA, Paris, France
{edufoursans,david.pointcheval}@ens.fr

Abstract. In 2015, Abdalla *et al.* introduced Inner-Product Functional Encryption, where both ciphertexts and decryption keys are vectors of fixed size n, and keys enable the computation of an inner product between the two. In practice, however, the size of the data parties are dealing with may vary over time. Having a public key of size n can also be inconvenient when dealing with very large vectors.
We define the Unbounded Inner-Product functionality in the context of Public-Key Functional Encryption, and introduce schemes that realize it under standard assumptions. In an Unbounded Inner-Product Functional Encryption scheme, a public key allows anyone to encrypt *unbounded vectors*, that are essentially mappings from $\mathbb{N}^*$ to $\mathbb{Z}_p$. The owner of the master secret key can generate functional decryption keys for other unbounded vectors. These keys enable one to evaluate the inner product between the unbounded vector underlying the ciphertext and the unbounded vector in the functional decryption key, provided certain conditions on the two vectors are met. We build Unbounded Inner-Product Functional Encryption by introducing pairings, using a technique similar to that of Boneh-Franklin Identity-Based Encryption. A byproduct of this is that our scheme can be made Identity-Based "for free". It is also the first Public-Key Inner-Product Functional Encryption Scheme with a constant-size public key (and master secret key), as well constant-size functional decryption keys: each consisting of just one group element.

Keywords. Unbounded Vectors, Functional Encryption, Inner Product.

1 Introduction

Functional Encryption (FE) [8, 10, 13, 17] is a new paradigm for encryption that does away with the "all-or-nothing" requirement of traditional Public-Key Encryption. FE allows users to learn specific functions of the encrypted data: for any function f from a class $\mathcal{F}$, a functional decryption key dk_f can be computed such that, given any ciphertext c with underlying plaintext x, using dk_f, a user can efficiently compute $f(x)$, but does not get any additional information about x. This is the most general form of encryption as it encompasses identity-based encryption, attribute-based encryption, broadcast encryption.

FE schemes for general functionalities have been introduced [4, 5, 12–14, 16, 19] but have thus far always been based on non-standard assumptions such as indistinguishability obfuscation or multilinear maps.

Inner-Product Functional Encryption. In 2015, Abdalla, Bourse, De Caro, and Pointcheval [1] (ABDP) suggested it might be worthwhile to instead give FE schemes for more restricted functionalities, but with reasonable efficiency and security proofs relying on better understood assumptions. They built FE schemes for the Inner-Product functionality which they proved selectively secure under the Decisional Diffie-Hellman and Learning-with-Errors assumptions. There are now variants with adaptive security [3].

1.1 Motivation

Inner-Product Functional Encryption (IPFE) enables many interesting applications, such as the computation of aggregate statistics or the evaluation of regression models, but, unfortunately, it until now required that the data being processed have a fixed size. The public and secret keys

also scale with this size, which can prove an inconvenience. We would like to construct schemes in which the public key is of constant, small size (ideally, a single group element), but where encrypting large vectors —in fact, arbitrarily large vectors— remains possible.

Let us go back to one of the motivating examples of IPFE: that of a school encrypting all the grades of each student, by discipline, as part of a single ciphertext every quarter. An authority can then distribute keys that enable one to compute a specific student's average grade (weighted by coefficients or by class hours), or the average over a class. It can also give keys that reveal the average grade in Mathematics or in Physics, always without jeopardizing the confidentiality of individual data, beyond what one can learn about the individuals from their aggregate. Now assume that a new student joins the school from one quarter to another. We would like to avoid the school having to query the authority for a new, readjusted public key (or for an extension of the current one). Whether the old keys should still work on the new, larger ciphertexts is to be decided on a case by case basis, and justifies our introducing multiple definitions: our *strict* and *permissive* notions.

One may wonder why the new keys could not be derived by a hash function (in the random oracle model, as we will use) in previous IPFE schemes. This would be for the public key, but with no way to derive the private keys required to generate the functional decryption keys.

1.2 Our Results

We introduce the first Unbounded Inner-Product Functional Encryption schemes (UIPFE). Both schemes share the following features:

1. **Unboundedness**: They enable the encryption of, and the generation of functional decryption keys for, unbounded vectors;
2. **Succinct keys**: In both cases the master secret key is a single secret scalar $s \in \mathbb{Z}_p$, and the public key is a corresponding group element $g_1^s \in \mathbb{G}_1$. Furthermore, the functional decryption keys simply consist of a group element $d \in \mathbb{G}_2$, in addition to the public vector describing the function evaluated by the functional decryption key;
3. **Identity-Based Access Control**: We consider both the computation on encrypted data aspect and the access control aspect of FE by letting users specify an identity in their ciphertext. The master authority gives functional decryption keys that limit evaluations of the unbounded inner product to ciphertexts of a given identity. This only expands the possible applications of our schemes, as the naive behavior can always be achieved by using the constant null identity.

Our main scheme is:

1. **Strict**: It only allows decryption when the domain of the ciphertext matches that of the key. In a sense, it may thus be thought of as operating infinitely many IPFE schemes in parallel.
2. **Selectively secure under a standard assumption**: We prove the security of our first scheme under the classical DBDH assumption, in the random oracle model.

We also introduce a scheme which is:

1. **Permissive**: It allows decryption when the support of the key (see Section 3.3) is included in the domain of the ciphertext.
2. **Selectively secure**: We prove the security of our second scheme in the random oracle model under ℓeDBDH, an interactive assumption we introduce. It resembles the DBDH assumption, except for the fact that the adversary can query linear combinations that depend on the CDH of the elements of one group, on condition that they never fully reveal it.

1.3 Concurrent Work

In concurrent and independent work, [18] also showed how to build Unbounded Inner-Product Functional Encryption from Bilinear Maps. Remarkably, their constructions do not require random oracles, and they prove full security under the SXDH assumption. Their constructions, however, are significantly less suited to practical use, since public keys require 28 group elements, ciphertexts 7 per coordinate and decryption keys 7 per coordinate (note that our decryption keys only require one group element regardless of the size of the function). Moreover, when decrypting, their schemes require a number of pairing evaluations that scales linearly with the sizes of the vectors, while ours compute a single pairing per decryption. Their constructions (*ct-dominant*) are what we call *permissive*, with the additional strong restriction that indices in the ciphertext must be contiguous (their E:con notion which requires the indices of the ciphertext to be *consecutive*). Moreover, they do not explicitly consider access control, while our schemes operate in the Identity-Based framework.

1.4 Related Work: Private-Key Multi-Input Inner-Product Functional Encryption for Unboundedly Many Inputs

Goldwasser *et al.* [11] introduced the notion of Multi-Input Functional Encryption for cases where we want the functions being evaluated on encrypted data to take multiple inputs, with each input corresponding to a different ciphertext. Abdalla *et al.* gave the first construction of Multi Input Functional Encryption for Inner Products [2], and Datta, Okamato and Tomida [9] recently showed how to achieve what they call *Unbounded Private-Key Multi-Input Inner-Product Functional Encryption*. While this is an important result, we must stress that they tackle a problem which significantly differs from ours: they encrypt vectors of constant size, and the Unbounded adjective applies to the number of inputs: they can generate keys which enable the evaluation of an inner product on a number of ciphertexts (inputs) which is not *a priori* bounded, while in our work it is the individual ciphertext (input) which has unbounded length. A perhaps more striking difference is that their scheme is Private-Key, with the encryption procedure requiring the master secret key, while we tackle the Public-Key setting.

1.5 Paper Organization

In Section 2, we define unbounded vectors, inner products between them and a pseudo-norm on them. We also recall the setting of pairing groups and the DBDH assumption. Section 3 defines FE, its security, and the different functionalities we are interested in. We build the first Strict Identity-Based Unbounded IPFE from standard assumptions in Section 4, and prove it selectively secure in the random oracle model under the DBDH assumption. Finally, in Section 5, we give a construction for Permissive Identity-Based Unbounded IPFE which we prove selectively secure in the random oracle model under an interactive variant of DBDH.

2 Notations

2.1 Unbounded Vectors

Both the plaintexts we are encrypting and the functions for which we will be generating keys will be referred to as unbounded vectors or lists. We write them as $\boldsymbol{x} = (x_i)_{i\in\mathcal{D}}$ or $\boldsymbol{y} = (y_i)_{i\in\mathcal{D}'}$, respectively, where both $\mathcal{D}$ and $\mathcal{D}'$ are finite subsets of $\mathbb{N}^*$, and $x_i, y_j \in \mathbb{Z}_p$ for $i \in \mathcal{D}, j \in \mathcal{D}'$. The vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ are thus mappings from $\mathbb{N}^*$ to $\mathbb{Z}_p$, and $\mathcal{D}$ (resp. $\mathcal{D}'$) is the explicit domain of $\boldsymbol{x}$ (resp. $\boldsymbol{y}$). When the context is clear, we will sometimes assimilate the vector space $\{(z_i)_{i\in\mathcal{D}} | z_i \in \mathbb{Z}_p\}$ and the isomorphic space $\mathbb{Z}_p^n$ where $n = |\mathcal{D}|$, the latter being more convenient for discussing changes of bases.

Inner Products. For $\boldsymbol{x} = (x_i)_{i\in\mathcal{D}}$ and $\boldsymbol{y} = (y_i)_{i\in\mathcal{D}'}$ we define the inner product as:

$$\langle \boldsymbol{x}, \boldsymbol{y}\rangle = \sum_{i\in\mathcal{D}\cap\mathcal{D}'} x_i y_i.$$

This comes from the fact that for indices $i \notin \mathcal{D}$, implicitly $x_i = 0$.

2.2 (Pseudo)Norm

Our proofs will require that given $\boldsymbol{x}^b \in \mathbb{Z}_p^n$ for $b \in \{0,1\}$ with $\boldsymbol{x}^0 \neq \boldsymbol{x}^1$ (with the same domain) and $\boldsymbol{y} \in \mathbb{Z}_p^n$, if we pick a basis $(\boldsymbol{z}_1, \ldots, \boldsymbol{z}_{n-1})$ of $(\boldsymbol{x}^0 - \boldsymbol{x}^1)^\perp$ and use ζ to denote the coefficient of $(\boldsymbol{x}^0 - \boldsymbol{x}^1)$ in the decomposition of $\boldsymbol{y}$ in basis $(\boldsymbol{x}^0 - \boldsymbol{x}^1, \boldsymbol{z}_1, \ldots, \boldsymbol{z}_{n-1})$,

$$\langle \boldsymbol{y}, \boldsymbol{x}^0\rangle = \langle \boldsymbol{y}, \boldsymbol{x}^1\rangle \implies \zeta = 0.$$

This is not true in general. From $\langle \boldsymbol{y}, \boldsymbol{x}^0\rangle = \langle \boldsymbol{y}, \boldsymbol{x}^1\rangle$ we can deduce that $\zeta \cdot \langle \boldsymbol{x}^0 - \boldsymbol{x}^1, \boldsymbol{x}^0 - \boldsymbol{x}^1\rangle = 0$, but we can only conclude if $\langle \boldsymbol{x}^0 - \boldsymbol{x}^1, \boldsymbol{x}^0 - \boldsymbol{x}^1\rangle \neq 0 \bmod p$. Previous works achieve this by bounding the individual components of $\boldsymbol{x}^0$ and $\boldsymbol{x}^1$, but this is not sufficient for unbounded vectors since we do not know n *a priori*. Instead, for any $\boldsymbol{x} = (x_i)_{i\in\mathcal{D}}$ we define

$$||\boldsymbol{x}|| = \min_{\{(x'_i)_{i\in\mathcal{D}}\in\mathbb{Z}^{\mathcal{D}} | x'_i \equiv x_i (mod\ p)\ \forall i\in\mathcal{D}\}} \sqrt{\sum_{i\in\mathcal{D}} x_i'^2}$$

where squaring and summation take place in $\mathbb{Z}$. It is easy to verify that for all vectors $\boldsymbol{a}$ and $\boldsymbol{b}$, $||\boldsymbol{a} - \boldsymbol{b}|| \leq ||\boldsymbol{a}|| + ||\boldsymbol{b}||$ and $||\boldsymbol{a}|| = 0 \implies \boldsymbol{a} = \boldsymbol{0}$ in $\mathbb{Z}_p^n$. We will always require that plaintext vectors being encrypted verify $||\boldsymbol{x}|| < \frac{\sqrt{p}}{2}$, so that

$$||\boldsymbol{x}^0 - \boldsymbol{x}^1||^2 \leq (||\boldsymbol{x}^0|| + ||\boldsymbol{x}^1||)^2 < (\frac{\sqrt{p}}{2} + \frac{\sqrt{p}}{2})^2 \leq p$$

and since $\langle \boldsymbol{x}^0 - \boldsymbol{x}^1, \boldsymbol{x}^0 - \boldsymbol{x}^1\rangle = 0 \bmod p \iff ||\boldsymbol{x}^0 - \boldsymbol{x}^1||^2 = 0 \bmod p$ that would imply $||\boldsymbol{x}^0 - \boldsymbol{x}^1||^2 = 0$ and thus $\boldsymbol{x}^0 = \boldsymbol{x}^1$ in $\mathbb{Z}_p^n$, which would contradict our assumption.

2.3 Pairing Group

We use a pairing group generator PGGen, a PPT algorithm that on input 1^λ returns a description $\mathcal{PG} = (\mathbb{G}_1, \mathbb{G}_2, p, P_1, P_2, e)$ of asymmetric pairing groups where $\mathbb{G}_1$, $\mathbb{G}_2$, $\mathbb{G}_T$ are additive cyclic groups of order p for a 2λ-bit prime p, P_1 and P_2 are generators of $\mathbb{G}_1$ and $\mathbb{G}_2$, respectively, and $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is an efficiently computable (non-degenerate) bilinear map. Define $P_T := e(P_1, P_2)$, which is a generator of $\mathbb{G}_T$.

We always use implicit representation of group elements. For $s \in \{1, 2, T\}$ and $a \in \mathbb{Z}_p$, define $[a]_s = aP_s \in \mathbb{G}_s$ as the implicit representation of a in $\mathbb{G}_s$. Note that from a random $[a]_s \in \mathbb{G}_s$ it is generally hard to compute the value a (discrete logarithm problem in $\mathbb{G}_s$). Obviously, given $[a]_s, [b]_s \in \mathbb{G}_s$ and a scalar $x \in \mathbb{Z}_p$, one can efficiently compute $[ax]_s \in \mathbb{G}_s$ and $[a + b]_s = [a]_s + [b]_s \in \mathbb{G}_s$.

More generally, for $s \in \{1, 2, T\}$ and a matrix $\mathbf{A} = (a_{ij}) \in \mathbb{Z}_p^{n\times m}$ we define $[\mathbf{A}]_s$ as the implicit representation of $\mathbf{A}$ in $\mathbb{G}_s$:

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

Given $[a]_1$, $[a]_2$, one can efficiently compute $[ab]_T$ using the pairing e. For two matrices $\mathbf{A}$, $\mathbf{B}$ with matching dimensions define $e([\mathbf{A}]_1, [\mathbf{B}]_2) := [\mathbf{AB}]_T \in \mathbb{G}_T$.

Using these notations, we can recall the seminal Decisional Bilinear Diffie-Hellman Assumption [7], adapted to the asymmetric setting:

Definition 1 (Decisional Bilinear Diffie-Hellman Assumption). *The Decisional Bilinear Diffie-Hellman (DBDH) Assumption in the asymmetric setting states that, in a pairing group* $\mathcal{G} \xleftarrow{\$} \mathsf{PGGen}(1^\lambda)$*, no PPT adversary can distinguish between the two following distributions with non-negligible advantage, where* $a, b, c, r \xleftarrow{\$} \mathbb{Z}_p$*:*

$$\{([a]_1, [b]_1, [a]_2, [c]_2, [abc]_T)\} \text{ and } \{([a]_1, [b]_1, [a]_2, [c]_2, [r]_T)\}.$$

3 Definitions and Security Models

3.1 Functional Encryption

We give the definition of Functional Encryption as originally defined in [8, 15].

Definition 2 (Functional Encryption). *A functional encryption scheme for a functionality* $\mathcal{F} : \mathcal{K} \times \mathcal{X} \to \mathcal{Z}$ *(where we require that the key space* $\mathcal{K}$ *contains the empty key* ϵ*) is a tuple of PPT algorithms* $\mathsf{SetUp}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}$ *defined as follows.*

- $\mathsf{SetUp}(\lambda)$*: takes as input a security parameter* 1^λ *and outputs a master secret key* msk *and a public key* pk*.*
- $\mathsf{KeyGen}(\mathsf{msk}, k)$*: takes as input the master secret key and a key description* $k \in \mathcal{K}$*, and outputs a functional decryption key* dk_k*.*
- $\mathsf{Encrypt}(\mathsf{pk}, x)$*: takes as input the public key* pk *and a message* $x \in \mathcal{X}$*, and outputs a ciphertext* c*.*
- $\mathsf{Decrypt}(\mathsf{dk}_k, c)$*: takes as input a functional decryption key* dk_k *and a ciphertext* c*, and returns an output* $y \in \mathcal{Z} \cup \{\perp\}$*, where* $\perp$ *is a special rejection symbol.*

We implicitly assume that mpk is included in msk and in all the encryption keys ek_i as well as the functional decryption keys dk_k.

Correctness. The correctness property states that, given $(\mathsf{pk}, \mathsf{msk}) \leftarrow \mathsf{SetUp}(\lambda)$, for any key description $k \in \mathcal{K}$ and any message $x \in \mathcal{X}$, if $c \leftarrow \mathsf{Encrypt}(\mathsf{pk}, x)$ and $\mathsf{dk}_k \leftarrow \mathsf{DKeyGen}(\mathsf{msk}, k)$, then $\mathsf{Decrypt}(\mathsf{dk}_k, c) = F(k, x)$.

Security. For any stateful adversary $\mathcal{A}$, and any functional encryption scheme, we define the following advantage.

$$\mathsf{Adv}_{\mathcal{A}}(\lambda) := \Pr\left[\beta' = \beta : \begin{array}{l} (\mathsf{pk}, \mathsf{msk}) \leftarrow \mathsf{SetUp}(1^\lambda) \\ (x_0, x_1) \leftarrow \mathcal{A}^{\mathsf{KeyGen}(\mathsf{msk}, \cdot)}(\mathsf{pk}) \\ \beta \xleftarrow{\$} \{0,1\} \\ c \leftarrow \mathsf{Encrypt}(\mathsf{pk}, x_\beta) \\ \beta' \leftarrow \mathcal{A}^{\mathsf{KeyGen}(\mathsf{msk}, \cdot)}(c) \end{array}\right] - \frac{1}{2},$$

with the restriction that $F(\epsilon, x_0) = F(\epsilon, x_1)$ and that for all key descriptions k queried to $\mathsf{KeyGen}(\mathsf{msk}, \cdot)$, the equation $F(k, x_0) = F(k, x_1)$ must hold. We say the scheme is IND-CPA secure if for all PPT adversaries $\mathcal{A}$, $\mathsf{Adv}_{\mathcal{A}}(\lambda) = \mathrm{negl}(\mathfrak{K})$.

A Weaker Notion. One may define a weaker variant of indistinguishability, called *Selective Security* or `sel-IND` security: the encryption queries are sent before the initialization.

3.2 The Unbounded Inner-Product Functionality

Inner-Product Functional Encryption as defined in [1], and later works, takes messages of fixed length and outputs ciphertexts of the same fixed length. Messages are vectors of n scalars, indexed from 1 to n. We will show how to build Inner-Product Functional Encryption schemes for arbitrary-size vectors.

While bounded message IPFE only considers vectors with contiguous indices we do not require this in our definitions to make them more general.

We give four definitions of Inner-Product Functional Encryption for Unbounded Vectors. The first two differ in their requirement on the domains of the ciphertexts and the keys for encryption to be successful. The last two are Identity-Based variants of the first two.

Definition 3 (Strict Unbounded IPFE).

- $\mathcal{K} = \{\epsilon\} \cup \{(y_i)_{i\in\mathcal{D}'} | \mathcal{D}' \subset \mathbb{N}^* \textit{ finite, } y_i \in \mathbb{Z}_p \ \forall i \in \mathcal{D}'\}$;
- $\mathcal{X} = \{\boldsymbol{x} = (x_i)_{i\in\mathcal{D}} | \mathcal{D} \subset \mathbb{N}^* \textit{ finite, } x_i \in \mathbb{Z}_p \ \forall i \in \mathcal{D} \textit{ and } ||\boldsymbol{x}|| < \frac{\sqrt{p}}{2}\}$;
- $\mathcal{Z} = \mathbb{Z}_p$;
- $F(\epsilon, (x_i)_{i\in\mathcal{D}}) = \mathcal{D}$ *and*

$$F((y_i)_{i\in\mathcal{D}'}, (x_i)_{i\in\mathcal{D}}) = \begin{cases} \langle \boldsymbol{y}, \boldsymbol{x} \rangle & \textit{if } \mathcal{D}' = \mathcal{D}; \\ \perp & \textit{otherwise.} \end{cases}$$

Definition 4 (Permissive Unbounded IPFE).

- $\mathcal{K} = \{\epsilon\} \cup \{(y_i)_{i\in\mathcal{D}'} | \mathcal{D}' \subset \mathbb{N}^* \textit{ finite, } y_i \in \mathbb{Z}_p \ \forall i \in \mathcal{D}'\}$;
- $\mathcal{X} = \{\boldsymbol{x} = (x_i)_{i\in\mathcal{D}} | \mathcal{D} \subset \mathbb{N}^* \textit{ finite, } x_i \in \mathbb{Z}_p \ \forall i \in \mathcal{D}\}$;
- $\mathcal{Z} = \mathbb{Z}_p$;
- $F(\epsilon, (x_i)_{i\in\mathcal{D}}) = \mathcal{D}$ *and*

$$F((y_i)_{i\in\mathcal{D}'}, (x_i)_{i\in\mathcal{D}}) = \begin{cases} \langle \boldsymbol{y}, \boldsymbol{x} \rangle & \textit{if } \mathcal{D}' \subset \mathcal{D}; \\ \perp & \textit{otherwise.} \end{cases}$$

Definition 5 (Strict Identity-Based Unbounded IPFE).

- $\mathcal{K} = \{\epsilon\} \cup \{(\mathsf{id}', (y_i)_{i\in\mathcal{D}'}) | \mathsf{id}' \in \{0,1\}^*, \mathcal{D}' \subset \mathbb{N}^* \textit{ finite, } y_i \in \mathbb{Z}_p \ \forall i \in \mathcal{D}'\}$;
- $\mathcal{X} = \{(\mathsf{id}, \boldsymbol{x} = (x_i)_{i\in\mathcal{D}}) | \mathsf{id} \in \{0,1\}^*, \mathcal{D} \subset \mathbb{N}^* \textit{ finite, } x_i \in \mathbb{Z}_p \ \forall i \in \mathcal{D}\}$;
- $\mathcal{Z} = \mathbb{Z}_p$;
- $F(\epsilon, (\mathsf{id}, (x_i)_{i\in\mathcal{D}})) = (\mathsf{id}, \mathcal{D})$ *and*

$$F((\mathsf{id}', (y_i)_{i\in\mathcal{D}'}), (\mathsf{id}, (x_i)_{i\in\mathcal{D}})) = \begin{cases} \langle \boldsymbol{y}, \boldsymbol{x} \rangle & \textit{if } \mathcal{D}' = \mathcal{D} \textit{ and } \mathsf{id} = \mathsf{id}'; \\ \perp & \textit{otherwise.} \end{cases}$$

Definition 6 (Permissive Identity-Based Unbounded IPFE).

- $\mathcal{K} = \{\epsilon\} \cup \{(\mathsf{id}', (y_i)_{i\in\mathcal{D}'}) | \mathsf{id}' \in \{0,1\}^*, \mathcal{D}' \subset \mathbb{N}^* \textit{ finite, } y_i \in \mathbb{Z}_p \ \forall i \in \mathcal{D}'\}$;
- $\mathcal{X} = \{(\mathsf{id}, \boldsymbol{x} = (x_i)_{i\in\mathcal{D}}) | \mathsf{id} \in \{0,1\}^*, \mathcal{D} \subset \mathbb{N}^* \textit{ finite, } x_i \in \mathbb{Z}_p \ \forall i \in \mathcal{D}\}$;
- $\mathcal{Z} = \mathbb{Z}_p$;
- $F(\epsilon, (\mathsf{id}, (x_i)_{i\in\mathcal{D}})) = (\mathsf{id}, \mathcal{D})$ *and*

$$F((\mathsf{id}', (y_i)_{i\in\mathcal{D}'}), (\mathsf{id}, (x_i)_{i\in\mathcal{D}})) = \begin{cases} \langle \boldsymbol{y}, \boldsymbol{x} \rangle & \textit{if } \mathcal{D}' \subseteq \mathcal{D} \textit{ and } \mathsf{id} = \mathsf{id}'; \\ \perp & \textit{otherwise.} \end{cases}$$

3.3 An Alternative Security Definition

To prove our permissive scheme secure, we will require a slightly different definition of security than the standard one, so we introduce it here. Like ABDP and later works on practical IPFE, key generation in our scheme is homomorphic: $\mathsf{KeyGen}(\mathsf{msk}, \boldsymbol{y}_1)+\mathsf{KeyGen}(\mathsf{msk}, \boldsymbol{y}_2) = \mathsf{KeyGen}(\mathsf{msk}, \boldsymbol{y}_1+\boldsymbol{y}_2)$. Moreover, ciphertexts are not required for inactive slots in the key. For instance, from $\mathsf{KeyGen}(\mathsf{msk}, \boldsymbol{y})$ where $\boldsymbol{y} = (y_j)_{j\in\mathcal{D}}$ and for some $i \in \mathcal{D}$, $y_i = 0$, one can evaluate $\sum_{j\in\mathcal{D}} x_j y_j$ from $(\mathsf{Encrypt}(\mathsf{pk}, (x_j))_{j\in\mathcal{D}, j\neq i}$. The standard security game of Functional Encryption does not take this into account as it only considers the domain of the vector $\boldsymbol{y}$. Let us first define, for any unbounded vector $\boldsymbol{z} = (z_i)_{i\in\mathcal{D}}$, its domain as $\mathsf{Domain}(\boldsymbol{z}) = \mathcal{D}$ and its support as $\mathsf{Support}(\boldsymbol{z}) = \{i \in \mathcal{D} | z_i \neq 0\}$. The support is thus the set of the active slots.

Definition 7 (Homomorphic Key Security).
In Homomorphic Key `IND` (and `sel-IND`) security, we modify the conditions for ignoring the adversary's guess as follows:

If for some $m \in \mathbb{N}^*$ and $\boldsymbol{y}^1, ..., \boldsymbol{y}^m$ queried to $\mathsf{KeyGen}(msk, \cdot)$,
there are $\omega_i \in \mathbb{Z}_p$, for all $i \in [m]$ such that, having defined $\boldsymbol{y} \leftarrow \sum_i \omega_i \boldsymbol{y}^i$,
$\mathsf{Support}(\boldsymbol{y}) \subseteq \mathsf{Domain}(\boldsymbol{x}^0) = \mathsf{Domain}(\boldsymbol{x}^1)$ and $\langle \boldsymbol{y}, \boldsymbol{x}^0 \rangle \neq \langle \boldsymbol{y}, \boldsymbol{x}^1 \rangle$,
then, ignore the adversary's guess.

Indeed, if the adversary can find a linear combination of the keys that make an inactive slot critical on the challenge ciphertext, then it can trivially win the game. This is very specific to the permissive constructions that allow any $\mathcal{D}' \subset \mathcal{D}$.

4 A Strict Identity-Based Unbounded IPFE

4.1 Description of the Scheme

We first present a selectively-secure **strict** identity-based UIPFE:

- $\mathsf{SetUp}(\lambda)$: Pick a pairing group $\mathcal{PG} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, e)$ of prime order p. Pick a full-domain hash function $\mathcal{H}$ into $\mathbb{G}_2$. Pick $s \xleftarrow{\$} \mathbb{Z}_p$ and publish $\mathsf{pk} = [s]_1$. Set $\mathsf{msk} = (s, \mathsf{pk})$.
- $\mathsf{Encrypt}(\mathsf{pk}, \mathsf{id}, \boldsymbol{x})$: Take as input an unbounded vector $\boldsymbol{x} = (x_i)_{i\in\mathcal{D}}$ where $\mathcal{D} \subset \mathbb{N}^*$ is finite, an identity id and the public key pk. Pick $r \xleftarrow{\$} \mathbb{Z}_p$, and output $\boldsymbol{C} = ([r]_1, (c_i)_{i\in\mathcal{D}})$ where $c_i = [x_i]_T + e([s]_1, r[u_{\mathsf{id}||\mathcal{D}||i}]_2)$ and $[u_{\mathsf{id}||\mathcal{D}||i}]_2 := \mathcal{H}(\mathsf{id}||\mathcal{D}||i)$ for all $i \in \mathcal{D}$.
- $\mathsf{KeyGen}(\mathsf{msk}, \mathsf{id}', \boldsymbol{y})$: Take as input an unbounded vector $\boldsymbol{y} = (y_i)_{i\in\mathcal{D}'}$ (where $\mathcal{D}' \subset \mathbb{N}^*$ is finite) representing its associated inner-product function, an identity id' and the master secret key $\mathsf{msk} = (s, \mathsf{pk})$. Output
$$\mathsf{dk}_{\boldsymbol{y}} = (\boldsymbol{y}, -s \sum_{i\in\mathcal{D}'} y_i [u_{\mathsf{id}'||\mathcal{D}'||i}]_2)$$
where $[u_{\mathsf{id}'||\mathcal{D}'||i}]_2 := \mathcal{H}(\mathsf{id}'||\mathcal{D}'||i)$ for all $i \in \mathcal{D}'$.
- $\mathsf{Decrypt}(\mathsf{dk}_y, \boldsymbol{C})$: Take as input a ciphertext $\boldsymbol{C} = (c_0, (c_i)_{i\in\mathcal{D}})$ and a decryption key $\mathsf{dk}_y = ((y_i)_{i\in\mathcal{D}} = \boldsymbol{y}, d)$. Compute
$$[\alpha]_T = e(c_0, d) + \sum_{i\in\mathcal{D}} y_i c_i$$
and recover the discrete logarithm to output α.

We clarify that $\cdot||\cdot$ denotes an efficient injective encoding into the set of binary strings.

Correctness. When $\mathsf{id} = \mathsf{id'}$ we have:

$$\begin{aligned}[\alpha]_T &= e(c_0, d) + \sum_{i\in\mathcal{D}} y_i c_i \\ &= e([r]_1, -s\sum_{i\in\mathcal{D}} y_i[u_{\mathsf{id}||\mathcal{D}||i}]_2)) + \sum_{i\in\mathcal{D}} y_i([x_i]_T + e([s]_1, r[u_{\mathsf{id}||\mathcal{D}||i}]_2)) \\ &= [\sum_{i\in\mathcal{D}} -sry_i u_{\mathsf{id}||\mathcal{D}||i} + y_i x_i + sry_i u_{\mathsf{id}||\mathcal{D}||i}]_T = [\sum_{i\in\mathcal{D}} y_i x_i]_T = [\langle \boldsymbol{y}, \boldsymbol{x}\rangle]_T.\end{aligned}$$

4.2 Security Analysis

Theorem 8 (`sel-IND` Security). *The Strict Identity-Based UIPFE scheme described above is `sel-IND`-secure under the DBDH assumption, in the random oracle model for $\mathcal{H}$.*

Proof. Given an adversary $\mathcal{A}$ that breaks the `sel-IND` security of our scheme, we construct an adversary $\mathcal{B}$ that breaks the DBDH assumption.

$\mathcal{B}$ receives a DBDH tuple $([a]_1, [b]_1, [a]_2, [c]_2, [d]_T)$. $\mathcal{B}$'s goal is to guess whether $d = abc$ or d is uniformly random. $\mathcal{A}$ chooses a pair of challenge vectors $(\boldsymbol{x}^0 = (x_i^0)_{i\in\mathcal{D}^*}, \boldsymbol{x}^1 = (x_i^1)_{i\in\mathcal{D}^*})$ to be encrypted under identity id^* and sends them to $\mathcal{B}$.

From now on, we write $|\mathcal{D}^*| = n$ and assimilate $\{(w_i)_{i\in\mathcal{D}^*} | w_i \in \mathbb{Z}_p \ \forall i \in \mathcal{D}^*\}$ with the vector space $\mathbb{Z}_p^n$, where $m : \mathcal{D}^* \to [n]$ maps the original indices to those in $\mathbb{Z}_p^n$.

Then, we follow the proof technique from [1], with a basis $(\boldsymbol{z}_1, \ldots, \boldsymbol{z}_{n-1})$ of $(\boldsymbol{x}^0 - \boldsymbol{x}^1)^\perp$. $\mathcal{B}$ also picks $n-1$ random scalars $(r_1, \ldots, r_{n-1}) \in \mathbb{Z}_p^{n-1}$. The family $(\boldsymbol{x}^0 - \boldsymbol{x}^1, \boldsymbol{z}_1, \ldots, \boldsymbol{z}_{n-1})$ is a basis of $\mathbb{Z}_p^n$ and we can write the canonical vectors e_i as

$$e_i = \alpha_i \cdot (\boldsymbol{x}^0 - \boldsymbol{x}^1) + \sum_{j\in[n-1]} \lambda_{i,j} \cdot \boldsymbol{z}_j$$

for some $\alpha_i \in \mathbb{Z}_p$, $\lambda_{i,j} \in \mathbb{Z}_p$, for all $i \in [n]$, $j \in [n-1]$. $\mathcal{B}$ can now simulate $\mathcal{A}$'s view:

- **Public Key.** $\mathcal{B}$ simply sets $\mathsf{pk} = [a]_1$ (implictly setting the master secret key msk to be the unknown scalar a) and sends it to $\mathcal{A}$.
- **Random Oracle Calls.** On any fresh input $str = \mathsf{id}||\mathcal{D}||i$, if $\mathsf{id}||\mathcal{D} \neq \mathsf{id}^*||\mathcal{D}^*$ or $i \notin \mathcal{D}^*$, $\mathcal{B}$ returns a random group element in $\mathbb{G}_2$, *the discrete logarithm of which* it stores as h_{str} and reuses upon a later request for the same input. On input $\mathsf{id}^*||\mathcal{D}^*||i$ for some $i \in \mathcal{D}^*$, $\mathcal{B}$ returns

$$\alpha_{m(i)}[c]_2 + \sum_{j\in[n-1]} \lambda_{m(i),j}[r_j]_2$$

  which it doesn't need to store because the above formula is deterministic.
- **Ciphertext.** $\mathcal{B}$ picks $\beta \in \{0, 1\}$ and generates a ciphertext for $\boldsymbol{x}^\beta$ from $[b]_1$, $[a]_2$ and $[d]_T$ as $c_0 = [b]_1$ and:

$$c_i = [x_i^\beta]_T + \alpha_{m(i)}[d]_T + \left(\sum_{j\in[n-1]} \lambda_{m(i),j} r_j\right) e([b]_1, [a]_2)$$

  for all $i \in \mathcal{D}^*$.
- **Decryption Keys.** $\mathcal{A}$ will input $\mathsf{id'}, \boldsymbol{y} = (y_i)_{i\in\mathcal{D}'}$. Make those calls to the random oracle that haven't been made for inputs $\mathsf{id'}||\mathcal{D}'||i$ for $i \in \mathcal{D}'$. If $\mathsf{id'} \neq \mathsf{id}^*$ or $\mathcal{D}' \neq \mathcal{D}^*$ simply return $(\boldsymbol{y}, -\sum_{i\in\mathcal{D}'} h_{\mathsf{id'}||\mathcal{D}'||i} y_i [a]_2)$. Otherwise write $(y_i)_{i\in\mathcal{D}^*} = \zeta \cdot (\boldsymbol{x}^0 - \boldsymbol{x}^1) + \sum_{i\in[n-1]} \nu_i \cdot \boldsymbol{z}_i$ for $\zeta \in \mathbb{Z}_p$, $\nu_i \in \mathbb{Z}_p$, for all $i \in [n-1]$. Then, return

$$\mathsf{dk}_{\boldsymbol{y}} = \left(\boldsymbol{y}, -\left(\sum_{i\in[n-1]} \nu_i \left(\sum_{j\in[n-1]} \lambda_{i,j} r_j\right)\right)[a]_2\right).$$

At the end of the simulation if $\mathcal{A}$ correctly guesses β, $\mathcal{B}$ guesses that $d = abc$ (the tuple is a proper BDH tuple), otherwise it guesses that d is uniformly random. It remains to be verified that $\mathcal{B}$ correctly simulates $\mathcal{A}$'s environment:

- The master public key and the random oracle responses are clearly uniformly random, thus properly distributed, despite the change of basis.
- From Section 2.2 we know that the coefficient ζ of $\boldsymbol{x}^0 - \boldsymbol{x}^1$ in the decomposition of a $\boldsymbol{y}$ for which a key has been queried is zero, otherwise the adversary $\mathcal{A}$ will not pass the final condition and its guess will be ignored. Hence, this contribution disappears from the functional key. The simulation of this key is perfect unless the attack is not legitimate;
- Now, notice that when $\mathcal{B}$ receives a true BDH tuple, it properly returns an encryption of $\boldsymbol{x}^\beta$, but when $[d]_T$ is uniformly random, the bit β is perfectly hidden.

Under the DBDH assumption, $\mathcal{A}$ cannot distinguish between these situations and thus, as in the latter, has no information on β. This concludes the proof. □

5 A Permissive Identity-Based Unbounded IPFE

5.1 Description of the Scheme

We now present a selectively-secure **permissive** identity-based UIPFE:

- $\mathsf{SetUp}(\lambda)$: Pick a pairing group $\mathcal{PG} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, e)$ of prime order p. Pick a full-domain hash function $\mathcal{H}$ into $\mathbb{G}_2$. Pick $s \xleftarrow{\$} \mathbb{Z}_p$ and publish $\mathsf{pk} = [s]_1$. Set $\mathsf{msk} = (s, \mathsf{pk})$.
- $\mathsf{Encrypt}(\mathsf{pk}, \mathsf{id}, \boldsymbol{x})$: Take as input an unbounded vector $\boldsymbol{x} = (x_i)_{i\in\mathcal{D}}$ where $\mathcal{D} \subset \mathbb{N}^*$ is finite, an identity id and the public key pk. Pick $r \xleftarrow{\$} \mathbb{Z}_p$, and output $\boldsymbol{C} = ([r]_1, (c_i)_{i\in\mathcal{D}})$ where $c_i = [x_i]_T + e([s]_1, r[u_{\mathsf{id}||i}]_2)$ and $[u_{\mathsf{id}||i}]_2 := \mathcal{H}(\mathsf{id}||i)$ for all $i \in \mathcal{D}$.
- $\mathsf{KeyGen}(\mathsf{msk}, \mathsf{id'}, \boldsymbol{y})$: Take as input an unbounded vector $\boldsymbol{y} = (y_i)_{i\in\mathcal{D}'}$ (where $\mathcal{D}' \subset \mathbb{N}^*$ is finite) representing its associated inner-product function, an identity $\mathsf{id'}$ and the master secret key $\mathsf{msk} = (s, \mathsf{pk})$. Output
$$\mathsf{dk}_{\boldsymbol{y}} = (\boldsymbol{y}, -s\sum_{i\in\mathcal{D}'} y_i[u_{\mathsf{id'}||i}]_2)$$
where $[u_{\mathsf{id'}||i}]_2 := \mathcal{H}(\mathsf{id'}||i)$ for all $i \in \mathcal{D}'$.
- $\mathsf{Decrypt}(\mathsf{dk}_y, \boldsymbol{C})$: Take as input a ciphertext $\boldsymbol{C} = (c_0, (c_i)_{i\in\mathcal{D}})$ and a decryption key $\mathsf{dk}_y = ((y_i)_{i\in\mathcal{D}'} = \boldsymbol{y}, d)$. Compute
$$[\alpha]_T = e(c_0, d) + \sum_{i\in\mathcal{D}'} y_i c_i$$
and recover the discrete logarithm to output α.

Correctness. When $\mathsf{id} = \mathsf{id'}$ we have:

$$\begin{aligned}
[\alpha]_T &= e(c_0, d) + \sum_{i\in\mathcal{D}} y_i c_i \\
&= e([r]_1, -s\sum_{i\in\mathcal{D}'} y_i[u_{\mathsf{id}||i}]_2)) + \sum_{i\in\mathcal{D}'} y_i([x_i]_T + e([s]_1, r[u_{\mathsf{id}||i}]_2)) \\
&= [\sum_{i\in\mathcal{D}'} -sry_i u_{\mathsf{id}||i} + y_i x_i + sry_i u_{\mathsf{id}||i}]_T = [\sum_{i\in\mathcal{D}'} y_i x_i]_T = [\langle \boldsymbol{y}, \boldsymbol{x}\rangle]_T.
\end{aligned}$$

5.2 New Assumption

Unfortunately, we will not be able to prove the security of this new scheme under a standard assumption. We thus define a new interactive one, that allows the adversary to see some linear combinations:

Definition 9 (Linearly Extended Decisional Bilinear Diffie-Hellman Assumption). *The Linearly Extended Decisional Bilinear Diffie-Hellman (ℓeDBDH) Assumption states that no PPT adversary $\mathcal{A}$ should be able to win the following game against a challenger $\mathcal{C}$ with non-negligible advantage:*

- *Initialize: $\mathcal{C}$ picks $a, b, c, r \xleftarrow{\$} \mathbb{Z}_p$ and $\delta \xleftarrow{\$} \{0,1\}$. If $\delta = 0$, $\mathcal{C}$ sends*
$$([a]_1, [b]_1, [a]_2, [c]_2, [abc]_T)$$
to $\mathcal{A}$, otherwise it sends
$$([a]_1, [b]_1, [a]_2, [c]_2, [r]_T).$$
- *Extension Queries: $\mathcal{A}$ has unlimited access to an oracle that, on input $i \in \mathbb{N}^*$:*
  - *if it stored a value h_i for i, reuses it;*
  - *otherwise, picks $h_i \xleftarrow{\$} \mathbb{Z}_p$, sends it to $\mathcal{A}$ and stores it;*
- *Linear Extension Queries: $\mathcal{A}$ has unlimited access to an oracle that, on input $(y_i)_{i\in\mathcal{D}}$ for some finite $S \subset \mathbb{N}$:*
  1. *For each $i \in \mathcal{D} \setminus \{0\}$:*
     - *if it stored a value h_i for i, reuses it;*
     - *otherwise, picks $h_i \xleftarrow{\$} \mathbb{Z}_p$ and stores it;*
  2. *stores $(y_i)_i$ and sends $[y_0 ac + \sum_{i\in\mathcal{D}, i\neq 0} y_i h_i a]_2$ to $\mathcal{A}$.*
- *Finalize: $\mathcal{A}$ provides its guess δ' on $\mathcal{C}$'s bit δ. $\mathcal{C}$ uses the stored $((y_i^{(k)})_i)_k$ to check that $e_0 \notin \boldsymbol{Span}((\boldsymbol{y}^{(k)})_k)$, and if so it outputs $\beta := \delta'$, otherwise it outputs $\beta \xleftarrow{\$} \{0,1\}$.*

5.3 Security Analysis

Theorem 10 (Homomorphic Key `sel-IND` Security). *The Permissive Identity-Based UIPFE scheme described above is Homomorphic Key `sel-IND`-secure under the ℓeDBDH assumption, in the random oracle model for $\mathcal{H}$.*

Proof. Given an adversary $\mathcal{A}$ that breaks the `sel-IND` security of our scheme, we construct an adversary $\mathcal{B}$ that breaks the ℓeDBDH assumption.

$\mathcal{B}$ receives a DBDH tuple $([a]_1, [b]_1, [a]_2, [c]_2, [d]_T)$ from a ℓeDBDH oracle. $\mathcal{B}$'s goal is to guess whether $d = abc$ or d is uniformly random. $\mathcal{A}$ chooses a pair of challenge vectors $(\boldsymbol{x}^0 = (x_i^0)_{i\in\mathcal{D}^*}, \boldsymbol{x}^1 = (x_i^1)_{i\in\mathcal{D}^*})$ to be encrypted under identity id^* and sends them to $\mathcal{B}$.

From now on we write $|\mathcal{D}^*| = n$ and assimilate $\{(w_i)_{i\in\mathcal{D}^*} | w_i \in \mathbb{Z}_p \ \forall i \in \mathcal{D}^*\}$ with the vector space $\mathbb{Z}_p^n$, and define $m : \mathcal{D}^* \to [n]$ which maps the original indices to those in $\mathbb{Z}_p^n$ and $m^\perp : \mathbb{N} \setminus \mathcal{D}^* \to \mathbb{N}^*$ which maps the other indices into $\mathbb{N}^*$.

$\mathcal{B}$ picks a basis $(\boldsymbol{z}_1, \ldots, \boldsymbol{z}_{n-1})$ of $(\boldsymbol{x}^0 - \boldsymbol{x}^1)^\perp$ as well as $n-1$ random scalars $(r_1, \ldots, r_{n-1}) \in \mathbb{Z}_p^{n-1}$. $(\boldsymbol{x}^0 - \boldsymbol{x}^1, \boldsymbol{z}_1, \ldots, \boldsymbol{z}_{n-1})$ is a basis of $\mathbb{Z}_p^n$ and we can write the canonical vectors e_i as

$$e_i = \alpha_i \cdot (\boldsymbol{x}^0 - \boldsymbol{x}^1) + \sum_{j\in[n-1]} \lambda_{i,j} \cdot \boldsymbol{z}_j$$

for some $\alpha_i \in \mathbb{Z}_p$, $\lambda_{i,j} \in \mathbb{Z}_p$, for all $i \in [n]$, $j \in [n-1]$.$\mathcal{B}$ can now simulate $\mathcal{A}$'s view:

- **Public Key.** $\mathcal{B}$ simply sets $\mathsf{pk} = [a]_1$ (implictly setting the master secret key msk to be the unknown scalar a) and sends it to $\mathcal{A}$.
- **Random Oracle Calls.** On any fresh input $str = \mathsf{id}||i$, if $\mathsf{id} \neq \mathsf{id}^*$, $\mathcal{B}$ returns a random group element in $\mathbb{G}_2$, *the discrete logarithm of which* it stores as h_{str} and reuses upon a later request for the same input. On input $\mathsf{id}^*||i$ for some $i \notin \mathcal{D}^*$, $\mathcal{B}$ makes an Extension Query to the ℓeDBDH oracle with input $m^\perp(i)$ and forwards its ouput to $\mathcal{A}$. On input $\mathsf{id}^*||i$ for some $i \in \mathcal{D}^*$, $\mathcal{B}$ returns
$$\alpha_{m(i)}[c]_2 + \sum_{j\in[n-1]} \lambda_{m(i),j}[r_j]_2$$
which it doesn't need to store because the above formula is deterministic.

– **Ciphertext.** $\mathcal{B}$ picks $\beta \in \{0,1\}$ and generates a ciphertext for $\boldsymbol{x}^\beta$ from $[b]_1$, $[a]_2$ and $[d]_T$ as $c_0 = [b]_1$ and:

$$c_i = [x_i^\beta]_T + \alpha_{m(i)}[d]_T + \left(\sum_{j\in[n-1]} \lambda_{m(i),j} r_j\right) e([b]_1, [a]_2)$$

for all $i \in \mathcal{D}^*$.

– **Decryption Keys.** $\mathcal{A}$ will input $\mathsf{id}', \boldsymbol{y} = (y_i)_{i\in\mathcal{D}'}$. Make those calls to the random oracle that haven't been made for inputs $\mathsf{id}'||i$ for $i \in \mathcal{D}'$. If $\mathsf{id}' \neq \mathsf{id}^*$ or $\mathcal{D}' \neq \mathcal{D}^*$ simply return $(\boldsymbol{y}, -\sum_{i\in\mathcal{D}'} h_{\mathsf{id}'||i} y_i [a]_2)$. Otherwise write $\mathcal{D}_1 = \mathcal{D}^* \setminus \{0\} \cap \mathcal{D}'$ and $\mathcal{D}_2 = \mathcal{D}' \setminus \mathcal{D}_1$. Decompose $\boldsymbol{y}$ as $(y_i)_{i\in\mathcal{D}^*} = \zeta(\boldsymbol{x}^0 - \boldsymbol{x}^1) + \sum_{i\in[n-1]} \nu_i \boldsymbol{z}_i$ for $\zeta \in \mathbb{Z}_p$, $\nu_i \in \mathbb{Z}_p$, for all $i \in [n-1]$. Make a Linear Extension Query to the $\ell e\mathsf{DBDH}$ oracle for input $(y'_i)_{i\in\{0\}\cup m^\perp(\mathcal{D}_2)}$ such that $y'_{m^\perp(i)} = y_i$ for all $i \in \mathcal{D}_2$ and $y'_0 = \zeta$, which returns $D \in \mathbb{G}_2$. Then, return

$$\mathsf{dk}_{\boldsymbol{y}} = \left(\boldsymbol{y}, -D - \left(\sum_{i\in[n-1]} \nu_i \left(\sum_{j\in[n-1]} \lambda_{i,j} r_j\right)\right)[a]_2\right).$$

At the end of the simulation if $\mathcal{A}$ correctly guesses β, $\mathcal{B}$ guesses that $d = abc$ (the tuple is a proper BDH tuple), otherwise it guesses that d is uniformly random. It remains to be verified that $\mathcal{B}$ correctly simulates $\mathcal{A}$'s environment:

– The master public key, functional decryption key and the random oracle responses are clearly uniformly random, thus properly distributed, despite the change of basis;
– From Section 3.3, we know that the span of all queried keys will not contain a key with domain included in $\mathcal{D}^*$ with a non zero component on $\boldsymbol{x}^0 - \boldsymbol{x}^1$, which guarantees that $\mathcal{B}$ does not break the condition that bars trivial victories in the $\ell e\mathsf{DBDH}$ game;
– Now, notice that when $\mathcal{B}$ receives a true BDH tuple, it properly returns an encryption of $\boldsymbol{x}^\beta$, but when $[d]_T$ is uniformly random, the bit β is perfectly hidden.

Under the $\ell e\mathsf{DBDH}$ assumption $\mathcal{A}$ cannot distinguish between these situations and thus, as in the latter, has no information on β. This concludes the proof. □

6 Open Problems

We have introduced constructions that are quite efficient in terms of size, since every key involved consists of a single group element, and thus the computational load is also much lower than in [18]. In addition, the vector ciphertexts do not need their domain to be a unique interval as in [18]. Still, several interesting problems remain open, and we now list promising directions for future research:

– Building Unbounded IPFE for any behavior without pairings, either groups without multilinearity or from other assumptions.
– Building Unbounded Functional Encryption schemes for different functionalities, such as Quadratic Polynomials (which already require pairings in the bounded setting [6]).
– Achieving adaptive security or removing random oracles with minimal overhead.

Acknowledgments

We would like to thank the anonymous reviewers for detailed comments. This work was supported in part by the European Community's Seventh Framework Programme (FP7/2007-2013 Grant Agreement no. 339563 – CryptoCloud) and the European Community's Horizon 2020 Project FENTEC (Grant Agreement no. 780108).

References

1. Abdalla, M., Bourse, F., De Caro, A., Pointcheval, D.: Simple functional encryption schemes for inner products. In: Katz, J. (ed.) PKC 2015. LNCS, vol. 9020, pp. 733–751. Springer, Heidelberg (Mar / Apr 2015) (Pages 1, 6, and 8.)
2. Abdalla, M., Gay, R., Raykova, M., Wee, H.: Multi-input inner-product functional encryption from pairings. In: Coron, J., Nielsen, J.B. (eds.) EUROCRYPT 2017, Part I. LNCS, vol. 10210, pp. 601–626. Springer, Heidelberg (Apr / May 2017) (Page 3.)
3. Agrawal, S., Libert, B., Stehlé, D.: Fully secure functional encryption for inner products, from standard assumptions. In: Robshaw, M., Katz, J. (eds.) CRYPTO 2016, Part III. LNCS, vol. 9816, pp. 333–362. Springer, Heidelberg (Aug 2016) (Page 1.)
4. Ananth, P., Brakerski, Z., Segev, G., Vaikuntanathan, V.: From selective to adaptive security in functional encryption. In: Gennaro, R., Robshaw, M.J.B. (eds.) CRYPTO 2015, Part II. LNCS, vol. 9216, pp. 657–677. Springer, Heidelberg (Aug 2015) (Page 1.)
5. Badrinarayanan, S., Goyal, V., Jain, A., Sahai, A.: Verifiable functional encryption. In: Cheon, J.H., Takagi, T. (eds.) ASIACRYPT 2016, Part II. LNCS, vol. 10032, pp. 557–587. Springer, Heidelberg (Dec 2016) (Page 1.)
6. Baltico, C.E.Z., Catalano, D., Fiore, D., Gay, R.: Practical functional encryption for quadratic functions with applications to predicate encryption. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017, Part I. LNCS, vol. 10401, pp. 67–98. Springer, Heidelberg (Aug 2017) (Page 11.)
7. Boneh, D., Franklin, M.K.: Identity based encryption from the Weil pairing. SIAM Journal on Computing 32(3), 586–615 (2003) (Page 4.)
8. Boneh, D., Sahai, A., Waters, B.: Functional encryption: Definitions and challenges. In: Ishai, Y. (ed.) TCC 2011. LNCS, vol. 6597, pp. 253–273. Springer, Heidelberg (Mar 2011) (Pages 1 and 5.)
9. Datta, P., Okamoto, T., Tomida, J.: Full-hiding (unbounded) multi-input inner product functional encryption from the k-linear assumption. In: Abdalla, M., Dahab, R. (eds.) PKC 2018, Part II. LNCS, vol. 10770, pp. 245–277. Springer, Heidelberg (Mar 2018) (Page 3.)
10. Garg, S., Gentry, C., Halevi, S., Raykova, M., Sahai, A., Waters, B.: Candidate indistinguishability obfuscation and functional encryption for all circuits. In: 54th FOCS. pp. 40–49. IEEE Computer Society Press (Oct 2013) (Page 1.)
11. Goldwasser, S., Gordon, S.D., Goyal, V., Jain, A., Katz, J., Liu, F.H., Sahai, A., Shi, E., Zhou, H.S.: Multi-input functional encryption. In: Nguyen, P.Q., Oswald, E. (eds.) EUROCRYPT 2014. LNCS, vol. 8441, pp. 578–602. Springer, Heidelberg (May 2014) (Page 3.)
12. Goldwasser, S., Kalai, Y.T., Popa, R.A., Vaikuntanathan, V., Zeldovich, N.: How to run turing machines on encrypted data. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part II. LNCS, vol. 8043, pp. 536–553. Springer, Heidelberg (Aug 2013) (Page 1.)
13. Goldwasser, S., Kalai, Y.T., Popa, R.A., Vaikuntanathan, V., Zeldovich, N.: Reusable garbled circuits and succinct functional encryption. In: Boneh, D., Roughgarden, T., Feigenbaum, J. (eds.) 45th ACM STOC. pp. 555–564. ACM Press (Jun 2013) (Page 1.)
14. Gorbunov, S., Vaikuntanathan, V., Wee, H.: Functional encryption with bounded collusions via multi-party computation. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO 2012. LNCS, vol. 7417, pp. 162–179. Springer, Heidelberg (Aug 2012) (Page 1.)
15. O'Neill, A.: Definitional issues in functional encryption. Cryptology ePrint Archive, Report 2010/556 (2010), http://eprint.iacr.org/2010/556 (Page 5.)
16. Sahai, A., Seyalioglu, H.: Worry-free encryption: functional encryption with public keys. In: Al-Shaer, E., Keromytis, A.D., Shmatikov, V. (eds.) ACM CCS 10. pp. 463–472. ACM Press (Oct 2010) (Page 1.)
17. Sahai, A., Waters, B.R.: Fuzzy identity-based encryption. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 457–473. Springer, Heidelberg (May 2005) (Page 1.)
18. Tomida, J., Takashima, K.: Unbounded inner product functional encryption from bilinear maps. In: Peyrin, T., Galbraith, S. (eds.) ASIACRYPT 2018, Part II. LNCS, vol. 11273, pp. 609–639. Springer, Heidelberg (Dec 2018) (Pages 3 and 11.)
19. Waters, B.: A punctured programming approach to adaptively secure functional encryption. In: Gennaro, R., Robshaw, M.J.B. (eds.) CRYPTO 2015, Part II. LNCS, vol. 9216, pp. 678–697. Springer, Heidelberg (Aug 2015) (Page 1.)