How to leverage hardness of constant-degree expanding polynomials over $\mathbb{R}$ to build $i\mathcal{O}$

Aayush Jain
aayushjain@cs.ucla.edu
Amit Sahai
sahai@cs.ucla.edu

UCLA

Abstract. In this work, we introduce and construct D-restricted Functional Encryption (FE) for any constant $D \geq 3$, based only on the SXDH assumption over bilinear groups. This generalizes the notion of 3-restricted FE recently introduced and constructed by Ananth et al. (ePrint 2018) in the generic bilinear group model.

A $D = (d+2)$-restricted FE scheme is a secret key FE scheme that allows an encryptor to efficiently encrypt a message of the form $M = (\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z})$. Here, $\boldsymbol{x} \in \mathbb{F}_{\mathbf{p}}^{d \times n}$ and $\boldsymbol{y}, \boldsymbol{z} \in \mathbb{F}_{\mathbf{p}}^{n}$. Function keys can be issued for a function $f = \Sigma_{\boldsymbol{I}=(i_1,..,i_d,j,k)}\ c_{\boldsymbol{I}} \cdot \boldsymbol{x}[1,i_1] \cdots \boldsymbol{x}[d,i_d] \cdot \boldsymbol{y}[j] \cdot \boldsymbol{z}[k]$ where the coefficients $c_{\boldsymbol{I}} \in \mathbb{F}_{\mathbf{p}}$. Knowing the function key and the ciphertext, one can learn $f(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z})$, if this value is bounded in absolute value by some polynomial in the security parameter and n. The security requirement is that the ciphertext hides $\boldsymbol{y}$ and $\boldsymbol{z}$, although it is not required to hide $\boldsymbol{x}$. Thus $\boldsymbol{x}$ can be seen as a public attribute.

D-restricted FE allows for useful evaluation of constant-degree polynomials, while only requiring the SXDH assumption over bilinear groups. As such, it is a powerful tool for leveraging hardness that exists in constant-degree expanding families of polynomials over $\mathbb{R}$. In particular, we build upon the work of Ananth et al. to show how to build indistinguishability obfuscation ($i\mathcal{O}$) assuming only SXDH over bilinear groups, LWE, and assumptions relating to weak pseudorandom properties of constant-degree expanding polynomials over $\mathbb{R}$.

1 Introduction

Program obfuscation transforms a computer program P into an equivalent program $O(P)$ such that any secrets present within P are "as hard as possible" to extract from $O(P)$. This property can be formalized by the notion of indistinguishability obfuscation ($i\mathcal{O}$) [11, 37]. Formally, $i\mathcal{O}$ requires that given any two equivalent programs P_1 and P_2 of the same size, a computationally bounded adversary cannot distinguish $O(P_1)$ from $O(P_2)$. $i\mathcal{O}$ has far-reaching application [30, 56], significantly expanding the scope of problems to which cryptography can be applied [56, 42, 29, 23, 32, 40, 15, 36, 39, 20].

The work of [30] gave the first mathematical candidate $i\mathcal{O}$ construction, and since then several additional candidates have been proposed and studied [28, 25, 33, 26, 38, 19, 10, 54, 4, 9, 21, 17, 24, 41, 18, 38, 22, 52, 51, 27, 43, 48, 7, 47].

Constructing $i\mathcal{O}$ without MMaps. Until 2018, all known constructions relied on multilinear maps [25, 26, 28, 33]. Unfortunately, multilinear map constructions are complex and surviving multilinear map security models [31, 13, 49] are themselves complex and difficult to analyze, as they have had to be modified in light of a sequence of attacks on multilinear map candidates [21, 17, 24, 41, 18, 38, 22, 52, 51].

This state of affairs is troubling scientifically, as we would like to be able to reduce the security of $i\mathcal{O}$ to problems that are simple to state, and where the underlying mathematics has a long history of study.

Everything old is new again: low-degree polynomials over the reals. Humanity has studied solving systems of (low-degree) polynomials over the reals for hundreds of years. Is it possible to use *hardness* associated with polynomial systems over the reals cryptographically? Surprisingly, despite hundreds of years of study, remarkably little is known about average-case hardness corresponding to *expanding* polynomial systems, where the number of real variables is n, and the polynomial equations over them is $n^{1+\epsilon}$ for $\epsilon > 0$.

The recent works of [5, 46, 1] introduced a new way constructing $i\mathcal{O}$ without relying on multilinear maps, by looking to hardness that may be present in degree two [5, 46, 1] or degree three [5] expanding polynomial systems over the reals.

The primary goal of our work is to extend the approach proposed by [5] to be able to use hardness associated with suitable expanding polynomial systems of *any constant degree*.

Leveraging low degree pseudorandomness over Z to build $i\mathcal{O}$. The key idea behind the work of [5] is to posit the existence of weak pseudorandom objects that are closely related to polynomials of degree 2 or 3 over the integers. They then introduce the crucial notion of 3-restricted functional encryption, which is a notion of functional encryption that allows for a *restricted* but still useful evaluation of degree-3 polynomials. This notion allows for the *natural* application of expanding families of degree-3 polynomials. (See below for further discussion on restricted-FE and its uses.)

Departing from previous work [7, 44, 47] that required at least trilinear maps to construct any meaningful FE for degree-3 functions, [5] show how to construct 3-restricted FE using only *bilinear maps*. Finally, by combining 3-restricted FE with the weak pseudorandom objects mentioned above, they achieve $i\mathcal{O}$ (also assuming LWE).

The goals of our present work are two-fold:

- To show how to extend the above approach beyond degree 3, to any constant degree D for $D \geq 3$. To do so, the key ingredient we construct is D-restricted FE, again *only* using bilinear maps regardless of the constant D.

– Furthermore, we construct D-restricted FE assuming only the SXDH assumption to hold over the bilinear map groups, instead of the generic bilinear model that was needed in [5].

We now elaborate.

D-restricted FE. A D-restricted FE scheme naturally generalizes the notion of 3-restricted FE scheme from [5]. We will write $D = d+2$ for notational convenience. Such a scheme is a secret key FE scheme that allows an encryptor to encrypt a message of the form $M = (\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z})$. Here, $\boldsymbol{x} \in \mathbb{F}^{d \times n}$ and $\boldsymbol{y}, \boldsymbol{z} \in \mathbb{F}_{\mathbf{p}}^{n}$. Function keys can be issued for a function $f = \Sigma_{\boldsymbol{I}=(i_1,..,i_d,j,k)}\ c_{\boldsymbol{I}} \cdot \boldsymbol{x}[1, i_1] \cdots \boldsymbol{x}[d, i_d] \cdot \boldsymbol{y}[j] \cdot \boldsymbol{z}[k]$ where the coefficients $c_{\boldsymbol{I}} \in \mathbb{F}_{\mathbf{p}}$. Knowing the key and the ciphertext, one can learn $f(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z})$, if this value is bounded in absolute value by some polynomial in the security parameter and n. The security requirement is that ciphertext hides $\boldsymbol{y}$ and $\boldsymbol{z}$, although it is not required to hide $\boldsymbol{x}$. Thus $\boldsymbol{x}$ can be seen as a public attribute. For implications to $i\mathcal{O}$, we require that encryption complexity should grow only linearly in n (upto polynomial factor in the security parameter).

Observe that for a given family of degree-D polynomials Q fixed in a function key, the notion of D-restricted FE allows an encryptor to choose the values of all variables $\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}$ at the time of encryption, and the decryptor will obtain $Q(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z})$. This allows for the most natural use of degree-D polynomials. We stress this point because other, less natural uses, are possible without using D-restricted FE, but these are unsatisfactory: One example would be where along with the polynomial Q the values of all variables $\boldsymbol{x}$ would also be fixed inside the function key. This would reduce the degree-D polynomials Q to quadratic polynomials, and just quadratic FE would then suffice (see, e.g., [46, 1]). However, again, this latter, less natural, approach would not allow $\boldsymbol{x}$ to be chosen freshly with each encryption. With our notion of D-restricted FE, such an unnatural setting – where some variables are fixed but others are freshly chosen with each encryption – can be avoided completely.

Why is it important to go beyond degree 3? At the core of the new works that construct $i\mathcal{O}$ without multilinear maps is the following key question: For some constant D, do there exist "expanding" distributions of polynomials $q_1, \ldots, q_m$ of degree D, where $m = n^{1+\epsilon}$ with polynomially-bounded coefficients, such that if one obtains $\boldsymbol{x} = (x_1, \ldots, x_n) \in \mathbb{Z}^n$ by sampling each x_i from a "nice" distribution with polynomially-bounded support, then is it hard to solve for $\boldsymbol{x}$ given $q_1(\boldsymbol{x}), \ldots, q_m(\boldsymbol{x})$? Remarkably, even though this question has a many-hundred year history within mathematics and nearly every branch of science, surprisingly little is known about *hardness* in this setting! And yet the hardness of such inversion problems is necessary (though not sufficient, see below) for this new line of work on constructing $i\mathcal{O}$.

Recently, [12] gave evidence that such problems may *not* be hard for $D = 2$. The case for $D = 3$ is less studied, and seems related to questions like the hardness of RANDOM 3-SAT. However, it seems that increasing D to larger constants should give us more confidence that hard distributions exist. For example, for

$D = 5$ and larger, this becomes related to the hardness of natural generalizations of the Goldreich PRG [35, 53]. It is also likely that as D grows, hardness "kicks in" for smaller values of n, similar to how the hardness of RANDOM k-SAT for constant $k > 3$ can be observed experimentally for much smaller values of n, than for RANDOM 3-SAT. Thus, our study could impact efficiency, as well.

Since studying the hardness of solving expanding families of polynomial equations over $\mathbb{R}$ is an exciting new line of cryptanalytic research, it is particularly important to study what values of D are cryptographically interesting. Before our work, only $D = 2$ and $D = 3$ were known to lead to $i\mathcal{O}$; our work shows that hardness for any constant degree D is interesting and cryptographically useful.

We stress that ensuring the hardness of solving for $\boldsymbol{x}$ given $q_1(\boldsymbol{x}), \ldots, q_m(\boldsymbol{x})$ is just the first step. Our work also clarifies the actual hardness assumptions that we need to imply $i\mathcal{O}$ as the following two assumptions. Since $D > 2$, let $D = d + 2$ for the rest of the discussion.

Weak LWE with leakage. This assumption says that there exists distributions χ over the integers and Q over families of multilinear degree-D polynomials such that the following two distributions are weakly indistinguishable, meaning that no efficient adversary can correctly identify the distribution from which a sample arose with probability above $\frac{1}{2} + 1/4\lambda$.

Distribution $\mathcal{D}_1$: Fix a prime modulus $\mathbf{p} = O(2^\lambda)$. Run $Q(n, B, \epsilon)$ to obtain polynomials $(q_1, ..., q_{\lfloor n^{1+\epsilon} \rfloor})$. Sample a secret $\boldsymbol{s} \leftarrow \mathbb{Z}_p^\lambda$ and sample $\boldsymbol{a}_{j,i} \leftarrow \mathbb{Z}_p^\lambda$ for $j \in [d], i \in [n]$. Finally, for every $j \in [d], i \in [n]$, sample $e_{j,i}, y_i, z_i \leftarrow \chi$, and write $\boldsymbol{e_j} = (e_{j,1}, \ldots, e_{j,n})$, $\boldsymbol{y} = (y_1, \ldots, y_n)$, $\boldsymbol{z} = (z_1, \ldots, z_n)$. Output:

$$\{\boldsymbol{a}_{j,i}, \langle \boldsymbol{a}_{j,i}, \boldsymbol{s} \rangle + e_{j,i} \bmod p\}_{j \in [d], i \in [n]}$$

along with

$$\{q_k, q_k(\boldsymbol{e_1}, \ldots, \boldsymbol{e_d}, \boldsymbol{y}, \boldsymbol{z})\}_{k \in [n^{1+\epsilon}]}$$

Distribution $\mathcal{D}_2$ is the same as $\mathcal{D}_1$, except that we additionally sample $e'_{j,i} \leftarrow \chi$ for $j \in [d], i \in [n]$. The output is now

$$\{\boldsymbol{a}_{j,i}, \langle \boldsymbol{a}_{j,i}, \boldsymbol{s} \rangle + e'_{j,i} \bmod p\}_{j \in [d], i \in [n]}$$

along with

$$\{q_k, q_k(\boldsymbol{e_1}, \ldots, \boldsymbol{e_d}, \boldsymbol{y}, \boldsymbol{z})\}_{k \in [n^{1+\epsilon}]}$$

We can think of the polynomials $q_k(\boldsymbol{e_1}, \ldots, \boldsymbol{e_d}, \boldsymbol{y}, \boldsymbol{z})$ as "leaking" some information about the LWE errors $e_{j,i}$. The assumption above states that such leakage provides only a limited advantage to the adversary. Critically, the fact that there are $n^2 > n^{1+\epsilon}$ quadratic monomials involving just $\boldsymbol{y}$ and $\boldsymbol{z}$ above, which are not used in the LWE samples at all, is crucial to avoiding linearization attacks over $\mathbb{Z}_p$ in the spirit of Arora-Ge [8]. For more discussion of the security of the above assumption in the context of $D = 3$, see [12].

The second assumption deals only with expanding degree-D polynomials over the reals, and requires that these polynomials are weakly perturbation resilient.

Weak Perturbation-Resilience. The second assumption is that there exists polynomials that for the same parameters above the following two distributions are weakly indistinguishable. By weakly indistinguishability we mean that no efficient adversary can correctly identify the distribution from which a sample arose with probability above $1 - 2/\lambda$. Let $\delta_i \in \mathbb{Z}$ be such that $|\delta_i| < B(\lambda, n)$ for some polynomial B and $i \in [n^{1+\epsilon}]$:

Distribution $\mathcal{D}_1$ consists of the evaluated polynomial samples. That is, we output:

$$\{q_k, q_k(\boldsymbol{e_1}, \ldots, \boldsymbol{e_d}, \boldsymbol{y}, \boldsymbol{z})\}_{k \in [n^{1+\epsilon}]}$$

Distribution $\mathcal{D}_2$ consists of the evaluated polynomial samples with added perturbations δ_i for $i \in [n^{1+\epsilon}]$. That is, we output:

$$\{q_k, q_k(\boldsymbol{e_1}, \ldots, \boldsymbol{e_d}, \boldsymbol{y}, \boldsymbol{z}) + \delta_k\}_{k \in [n^{1+\epsilon}]}$$

These assumptions are sketched here informally; the formal definitions are given in Section 5.

Our Results: Our results can be summarized as follows. First, we construct a $(d+2)$ restricted FE scheme from SXDH assumption.

Theorem 1. *Assuming SXDH over bilinear maps, there is a construction of a $(d+2)$ restricted FE scheme for any constant $d \geq 1$.*

Then, we give candidates of perturbation resilient generators that can be implemented using a $(d+2)$ restricted FE scheme. Finally, using such a perturbation resilient generator and $(d+2)$ restricted FE, we construct $i\mathcal{O}$ via the approach given by [5]. Here is our final theorem.

Theorem 2. *For any constant integer $d \geq 1$, two distinguishing gaps $\mathsf{adv}_1, \mathsf{adv}_2$, if $\mathsf{adv}_1 + \mathsf{adv}_2 \leq 1 - 2/\lambda$ then assuming,*

- *Subexponentially hard LWE.*
- *Subexponentially hard SXDH.*
- *PRGs with*
  - *Stretch of $k^{1+\epsilon}$ (length of input being k bits) for some constant $\epsilon > 0$.*
  - *Block locality $d+2$.*
  - *Security with distinguishing gap bounded by adv_1 against adversaries of sub-exponential size.*
- *$\mathsf{d}\Delta\mathsf{RG}$ with distinguishing gap bounded by adv_2 against adversaries of size 2^λ. Details about the notion of $\mathsf{d}\Delta\mathsf{RG}$ can be found in Sections 5 and 6.*

there exists a secure $i\mathcal{O}$ scheme for $P/poly$.

We now proceed to more detailed, but still informal, technical overview of our techniques.

2 Technical Overview

$(d+2)$-restricted FE. The key technical tool constructed in this work is the notion of $(d+2)-$restricted FE (dFE for short) for any constant integer $d \geq 1$. We recall that a dFE scheme over $\mathbb{F}_{\mathbf{p}}$ is a secret key functional encryption scheme for the functions f of the following form: $f : \mathbb{F}_{\mathbf{p}}^{n \times (d+2)} \to \mathbb{F}_{\mathbf{p}}$. To be precise, f takes as input $(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z})$ where $\boldsymbol{x} \in \mathbb{F}_{\mathbf{p}}^{n \times (d)}$ and $\boldsymbol{y}, \boldsymbol{z} \in \mathbb{F}_{\mathbf{p}}^{n}$. Then it computes $f(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}) = \Sigma_{\boldsymbol{I}=(i_1,..,i_d,j,k)} c_{\boldsymbol{I}} \cdot \boldsymbol{x}[1, i_1] \cdots \boldsymbol{x}[d, i_d] \cdot \boldsymbol{y}[j] \cdot \boldsymbol{z}[k]$ where each coefficient $c_{\boldsymbol{I}} \in \mathbb{F}_{\mathbf{p}}$. We require the decryption to be efficient only if the output is bounded in norm by a polynomial bound $B(\lambda, n)$. Security of a dFE scheme intuitively requires that ciphertext only reveals the d public components $\boldsymbol{x}$ and the output of the decryption.

Before we describe our construction, let us first recall the construction of 3-restricted FE from [5]:

3-restricted FE [5]. Before getting to 3 restricted FE, let's first recap how secret key quadratic functional encryption schemes [45] work at a high level. Let's say that the encryptor wants to encrypt $\boldsymbol{y}, \boldsymbol{z} \in \mathbb{F}_{\mathbf{p}}^{n}$. The master secret key consists of two secret random vectors $\boldsymbol{\beta}, \boldsymbol{\gamma} \in \mathbb{F}_{\mathbf{p}}^{n}$ that are used for enforcement of computations done on $\boldsymbol{y}$ and $\boldsymbol{z}$ respectively. The idea is that the encryptor encodes $\boldsymbol{y}$ and $\boldsymbol{\beta}$ using some randomness r, and similarly encodes $\boldsymbol{z}$ and $\boldsymbol{\gamma}$ together as well. These encodings are created using bilinear maps in one of the two base groups. These encodings are constructed so that the decryptor can compute an encoding of $[g(\boldsymbol{y}, \boldsymbol{z}) - rg(\boldsymbol{\beta}, \boldsymbol{\gamma})]_t$ in the target group for *any* quadratic function g. The function key for the given function f is constructed in such a manner that it allows the decryptor to compute the encoding $[rf(\boldsymbol{\beta}, \boldsymbol{\gamma})]_t$ in the target group. Thus the output $[f(\boldsymbol{y}, \boldsymbol{z})]_t$ can be recovered in the exponent by computing the sum of $[rf(\boldsymbol{\beta}, \boldsymbol{\gamma})]_t$ and $[f(\boldsymbol{y}, \boldsymbol{z}) - rf(\boldsymbol{\beta}, \boldsymbol{\gamma})]_t$ in the exponent. As long as $f(\boldsymbol{y}, \boldsymbol{z})$ is polynomially small, this value can then be recovered efficiently.

Clearly the idea above only works for degree-2 computations, if we use bilinear maps. However, the work of [5] built upon this idea nevertheless to construct a 3-restricted FE scheme. Recall, in a 3-restricted FE one wants to encrypt three vectors $\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z} \in \mathbb{F}_{\mathbf{p}}^{n}$. While $\boldsymbol{y}$ and $\boldsymbol{z}$ are required to be hidden, $\boldsymbol{x}$ is not required to be hidden.

In their scheme, in addition to $\boldsymbol{\beta}, \boldsymbol{\gamma} \in \mathbb{F}_{\mathbf{p}}^{n}$ in case of a quadratic FE, another vector $\boldsymbol{\alpha} \in \mathbb{F}_{\mathbf{p}}^{n}$ is also sampled that is used to enforce the correctness of the $\boldsymbol{x}$ part of the computation. As before, given the ciphertext one can compute $[\boldsymbol{y}[j]\boldsymbol{z}[k] - r\boldsymbol{\beta}[j]\boldsymbol{\gamma}[k]]_t$ for $j, k \in [n]$. But this is clearly not enough, as these encodings do not involve $\boldsymbol{x}$ in any way. Thus, in addition, an encoding of $r(\boldsymbol{x}[i] - \boldsymbol{\alpha}[i])$ is also given in the ciphertext for $i \in [n]$. Inside the function key, there are corresponding encodings of $\boldsymbol{\beta}[j]\boldsymbol{\gamma}[k]$ for $j, k \in [n]$ which the decryptor can pair with encoding of $r(\boldsymbol{x}[i] - \boldsymbol{\alpha}[i])$ to form the encoding $[r(\boldsymbol{x}[i] - \boldsymbol{\alpha}[i])\boldsymbol{\beta}[j]\boldsymbol{\gamma}[k]]_t$ in the target group.

Now observe that,

$$\begin{aligned}&\boldsymbol{x}[i] \cdot \big(\boldsymbol{y}[j]\boldsymbol{z}[k] - r\boldsymbol{\beta}[j]\boldsymbol{\gamma}[k]\big) + r(\boldsymbol{x}[i] - \boldsymbol{\alpha}[i]) \cdot \boldsymbol{\beta}[j]\boldsymbol{\gamma}[k] \\ =&\boldsymbol{x}[i]\boldsymbol{y}[j]\boldsymbol{z}[k] - r\boldsymbol{\alpha}[i]\boldsymbol{\beta}[j]\boldsymbol{\gamma}[k]\end{aligned}$$

Above, since $\boldsymbol{x}[i]$ is public, the decryptor can herself take $(\boldsymbol{y}[j]\boldsymbol{z}[k]-r\boldsymbol{\beta}[j]\boldsymbol{\gamma}[k])$, which she already has, and multiply it with $\boldsymbol{x}[i]$ in the exponent. This allows her to compute encoding of $[\boldsymbol{x}[i]\boldsymbol{y}[j]\boldsymbol{z}[k] - r\boldsymbol{\alpha}[i]\boldsymbol{\beta}[j]\boldsymbol{\gamma}[k]]_t$. Combining these encodings appropriately, she can obtain $[g(\boldsymbol{x},\boldsymbol{y},\boldsymbol{z}) - rg(\boldsymbol{\alpha},\boldsymbol{\beta},\boldsymbol{\gamma})]_t$ for any degree-3 multilinear function g. Given the function key for f and the ciphertext, one can compute $[rf(\boldsymbol{\alpha},\boldsymbol{\beta},\boldsymbol{\gamma})]_t$ which can be used to unmask the output. This is because the ciphertext contains an encoding of r in one of the base groups and the function key contains an encoding of $f(\boldsymbol{\alpha},\boldsymbol{\beta},\boldsymbol{\gamma})$ in the other group and pairing them results in $[rf(\boldsymbol{\alpha},\boldsymbol{\beta},\boldsymbol{\gamma})]_t$.

The work of [5] shows how to analyze the security of the construction above in a generic bilinear group model.

Towards constructing $(d+2)-$restricted FE. Now let's consider how we can extend the approach discussed above for the case of $d = 2$. Suppose now we want to encrypt $\boldsymbol{u},\boldsymbol{x},\boldsymbol{y}$ and $\boldsymbol{z}$. Here $\boldsymbol{y},\boldsymbol{z}$ are supposed to be private while $\boldsymbol{x}$ and $\boldsymbol{u}$ are not required to be hidden. Let's now also have $\boldsymbol{\phi} \in \mathbb{F}_{\mathbf{p}}^n$ to enforce $\boldsymbol{u}$ part of the computation. How can we generalize the idea above to allow for degree-4 computations? One straightforward idea is to release encodings of $r(\boldsymbol{u}[i_1]\boldsymbol{x}[i_2] - \boldsymbol{\phi}[i_1]\boldsymbol{\alpha}[i_2])$ for $i_1, i_2 \in [n]$ in the ciphertext instead of encodings of $r(\boldsymbol{x}[i_2] - \boldsymbol{\alpha}[i_2])$ like before. This would permit the computation of $[f(\boldsymbol{u},\boldsymbol{x},\boldsymbol{y},\boldsymbol{z}) - rf(\boldsymbol{\phi},\boldsymbol{\alpha},\boldsymbol{\beta},\boldsymbol{\gamma})]_t$. However, such an approach would not be efficient enough for our needs: we require the complexity of encryption to be linear in n. However, the approach above would need to provide n^2 encodings corresponding to $r(\boldsymbol{u}[i_1]\boldsymbol{x}[i_2]-\boldsymbol{\phi}[i_1]\boldsymbol{\alpha}[i_2])$ for every $i_1, i_2 \in [n]$.

Our first idea: A "ladder" of enforcement. Let's now take a step back. Notice that our 3-restricted FE scheme already allows one to compute $[\boldsymbol{x}[i_2]\boldsymbol{y}[j]\boldsymbol{z}[k] - r\boldsymbol{\alpha}[i_2]\boldsymbol{\beta}[j]\boldsymbol{\gamma}[k]]_t$ for any $i_2, j, k \in [n]$. We want to leverage this existing capability to bootstrap to degree-4 computations.

Suppose the decryptor is also able to generate the encoding $[r(\boldsymbol{u}[i_1] - \boldsymbol{\phi}[i_1]) \cdot \boldsymbol{\alpha}[i_2]\boldsymbol{\beta}[j]\boldsymbol{\gamma}[k]]_t$ for any $i_1, i_2, j, k \in [n]$. Then, she can generate the encoding $[\boldsymbol{u}[i_1]\boldsymbol{x}[i_2]\boldsymbol{y}[j]\boldsymbol{z}[k] - \boldsymbol{\phi}[i_1]\boldsymbol{\alpha}[i_2]\boldsymbol{\beta}[j]\boldsymbol{\gamma}[k]]_t$ as follows:

$$\begin{aligned} &r(\boldsymbol{u}[i_1] - \boldsymbol{\phi}[i_1])\boldsymbol{\alpha}[i_2]\boldsymbol{\beta}[j]\boldsymbol{\gamma}[k] + \boldsymbol{u}[i_1] \cdot \left(\boldsymbol{x}[i_2]\boldsymbol{y}[j]\boldsymbol{z}[k] - r\boldsymbol{\alpha}[i_2]\boldsymbol{\beta}[j]\boldsymbol{\gamma}[k]\right) \\ =&\boldsymbol{u}[i_1]\boldsymbol{x}[i_2]\boldsymbol{y}[j]\boldsymbol{z}[k] - r\boldsymbol{\phi}[i_1]\boldsymbol{\alpha}[i_2]\boldsymbol{\beta}[j]\boldsymbol{\gamma}[k] \end{aligned}$$

Notice that $\boldsymbol{u}$ is public so the decryptor can herself take $(\boldsymbol{x}[i_2]\boldsymbol{y}[j]\boldsymbol{z}[k] - r\boldsymbol{\alpha}[i_2]\boldsymbol{\beta}[j]\boldsymbol{\gamma}[k])$, which she already has, and multiply it with $\boldsymbol{u}[i_1]$ in the exponent. To allow the computation of $[r(\boldsymbol{u}[i_1] - \boldsymbol{\phi}[i_1])\boldsymbol{\alpha}[i_2]\boldsymbol{\beta}[j]\boldsymbol{\gamma}[k]]_t$ we can provide additionally encodings of $(\boldsymbol{u}[i_1] - r\boldsymbol{\phi}[i_1])$ in the ciphertexts for $i_1 \in [n]$ and corresponding encodings of $\boldsymbol{\alpha}[i_2]\boldsymbol{\beta}[j]\boldsymbol{\gamma}[k]$ for $i_2, j, k \in [n]$ in the function key that can be paired together.

What next? As before, the decryptor can homomorphically compute on these encodings and learn $[f(\boldsymbol{u},\boldsymbol{x},\boldsymbol{y},\boldsymbol{z}) - rf(\boldsymbol{\phi},\boldsymbol{\alpha},\boldsymbol{\beta},\boldsymbol{\gamma})]_t$. Finally, the decryptor can compute $[rf(\boldsymbol{\phi},\boldsymbol{\alpha},\boldsymbol{\beta},\boldsymbol{\gamma})]_t$ by pairing an encoding of r given in the ciphertext and

and encoding of $f(\boldsymbol{\phi}, \boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma})$ given in the function key. Thus, the output can be unmasked in the exponent.

Observe that this solution preserves linear efficiency of the ciphertext. As of now we have not told anything about how security is argued. From computation point of view, this solution indeed turns out to be insightful as this process can now be generalized to form a ladder of enforcement for any constant degree-D computations.

Laddered computations for any constant degree $(d+2)$. First let's set up some notation. Let $\boldsymbol{x} \in \mathbb{F}_{\mathbf{p}}^{d \times n}$ be the public part of the plain-text and $\boldsymbol{y}, \boldsymbol{z} \in \mathbb{F}_{\mathbf{p}}^{n}$. Let $\boldsymbol{\alpha} \in \mathbb{F}_{\mathbf{p}}^{d \times n}$ be the vector of random field elements corresponding to $\boldsymbol{x}$. Similarly, $\boldsymbol{\beta}$ and $\boldsymbol{\gamma}$ in $\mathbb{F}_{\mathbf{p}}^{n}$ be the vector of random elements correspoding to $\boldsymbol{y}$ and $\boldsymbol{z}$ respectively.

The next observation is the following. Suppose the decryptor can generate the following terms by pairing encodings present in the ciphertext and encodings present in the functional key, for every $\boldsymbol{I} = (i_1, .., i_d, j, k) \in [n]^D$.

- $[\boldsymbol{y}[j]\boldsymbol{z}[k] - r\beta_j\gamma_k]_t$ for $j, k \in [n]$.
- $[r(\boldsymbol{x}[d, i_d] - \boldsymbol{\alpha}[d, i_d]) \cdot \boldsymbol{\beta}[j]\boldsymbol{\gamma}[k]]_t$
- $[r(\boldsymbol{x}[d-1, i_{d-1}] - \boldsymbol{\alpha}[d-1, i_{d-1}]) \cdot \boldsymbol{\alpha}[d, i_d]\boldsymbol{\beta}[j]\boldsymbol{\gamma}[k]]_t$
- $\ldots$
- $[r(\boldsymbol{x}[1, i_1] - \boldsymbol{\alpha}[1, i_1]) \cdot \boldsymbol{\alpha}[2, i_2] \cdots \boldsymbol{\alpha}[d, i_d]\boldsymbol{\beta}[j]\boldsymbol{\gamma}[k]]_t$

As before, the decryptor can also obtain an encoding $[rf(\boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma})]_t$ corresponding to the degree-D multilinear function f in the function key.

The main observation to generalize the $D = 4$ case discussed above is then the following. Consider the first two terms: $[\boldsymbol{y}[j]\boldsymbol{z}[k] + r\beta_j\gamma_k]_t$ and $[r(\boldsymbol{x}[d, i_d] - \boldsymbol{\alpha}[d, i_d])\boldsymbol{\beta}[j]\boldsymbol{\gamma}[k]]_t$ and note that:

$$\begin{aligned} &\boldsymbol{x}[d, i_d](\boldsymbol{y}[j]\boldsymbol{z}[k] - r\beta_j\gamma_k) + r(\boldsymbol{x}[d, i_d] - \boldsymbol{\alpha}[d, i_d])\boldsymbol{\beta}[j]\boldsymbol{\gamma}[k] \\ =&\boldsymbol{x}[d, i_d]\boldsymbol{y}[j]\boldsymbol{z}[k] - r\boldsymbol{\alpha}[d, i_d]\boldsymbol{\beta}[j]\boldsymbol{\gamma}[k] \end{aligned}$$

This observation allows the decryptor to compute an encoding

$$\mathsf{Int}_d = [\boldsymbol{x}[d, i_d]\boldsymbol{y}[j]\boldsymbol{z}[k] - r\boldsymbol{\alpha}[d, i_d]\boldsymbol{\beta}[j]\boldsymbol{\gamma}[k]]_t$$

using encodings of the first two types in the list above.

Next observe that using the encoding,

$$[r(\boldsymbol{x}[d-1, i_{d-1}] - \boldsymbol{\alpha}[d-1, i_{d-1}]) \cdot \boldsymbol{\alpha}[d, i_d]\boldsymbol{\beta}[j]\boldsymbol{\gamma}[k]]_t$$

and encoding Int_d one can compute

$$\mathsf{Int}_{d-1} = [\boldsymbol{x}[d-1, i_{d-1}]\boldsymbol{x}[d, i_d]\boldsymbol{y}[j]\boldsymbol{z}[k] - r\boldsymbol{\alpha}[d-1, i_{d-1}]\boldsymbol{\alpha}[d, i_d]\boldsymbol{\beta}[j]\boldsymbol{\gamma}[k]]_t$$

This is because,

$$\begin{aligned}&\boldsymbol{x}[d-1,i_{d-1}]\cdot(\boldsymbol{x}[d,i_d]\boldsymbol{y}[j]\boldsymbol{z}[k]-r\boldsymbol{\alpha}[d,i_d]\boldsymbol{\beta}[j]\boldsymbol{\gamma}[k])\\&+r(\boldsymbol{x}[d-1,i_{d-1}]-\boldsymbol{\alpha}[d-1,i_{d-1}])\cdot\boldsymbol{\alpha}[d,i_d]\boldsymbol{\beta}[j]\boldsymbol{\gamma}[k]\\=&\boldsymbol{x}[d-1,i_{d-1}]\boldsymbol{x}[d,i_d]\boldsymbol{y}[j]\boldsymbol{z}[k]-r\boldsymbol{\alpha}[d-1,i_{d-1}]\boldsymbol{\alpha}[d,i_d]\boldsymbol{\beta}[j]\boldsymbol{\gamma}[k]\end{aligned}$$

Continuing this way up a "ladder" the decryptor can compute

$$\mathsf{Mon}_{\boldsymbol{I}}=[\Pi_{\ell\in[d]}\boldsymbol{x}[\ell,i_\ell]\boldsymbol{y}[j]\boldsymbol{z}[k]-r\Pi_{\ell\in[d]}\boldsymbol{\alpha}[\ell,i_\ell]\boldsymbol{\beta}[j]\boldsymbol{\gamma}[k]]_t$$

Observe that the term $\Pi_{\ell\in[d]}\boldsymbol{x}[\ell,i_\ell]\boldsymbol{y}[j]\boldsymbol{z}[k]-r\Pi_{\ell\in[d]}\boldsymbol{\alpha}[\ell,i_\ell]\boldsymbol{\beta}[j]\boldsymbol{\gamma}[k]$ corresponding to $\mathsf{Mon}_{\boldsymbol{I}}$ can be generated as a linear combination of terms from the list above. Once $\mathsf{Mon}_{\boldsymbol{I}}$ is computed then the decryptor can do the following. Since $f=\Sigma_{\boldsymbol{I}=(i_1,..,i_d,j,k)}c_{\boldsymbol{I}}\boldsymbol{x}[1,i_1]\cdots\boldsymbol{x}[d,i_d]\boldsymbol{y}[j]\boldsymbol{z}[k]$, the decryptor can then compute:

$$\mathsf{Mon}_f=[f(\boldsymbol{x},\boldsymbol{y},\boldsymbol{z})-rf(\boldsymbol{\alpha},\boldsymbol{\beta},\boldsymbol{\gamma})]_t$$

Finally using $[rf(\boldsymbol{\alpha},\boldsymbol{\beta},\boldsymbol{\gamma})]_t$ the decryptor can recover $[f(\boldsymbol{x},\boldsymbol{y},\boldsymbol{z})]_t$

How to base security on SXDH? So far, we have just described a potential computation pattern that allows the decryptor to obtain the function output given a function key and a ciphertext. Any scheme that allows constructing the terms described above in the ladder is guaranteed to satisfy correctness. But how do we argue security?

We rely on a primitive called Canonical Function Hiding Inner Product Encryption (cIPE for short). A cIPE scheme allows the decryptor to compute the inner product of a vector encoded in the ciphertext, with a vector encoded in the function key. Also, intuitively, cIPE guarantees that the vector embedded in the function key is also hidden given the function key. More precisely, given any vectors $\boldsymbol{v},\boldsymbol{v}',\boldsymbol{u},\boldsymbol{u}'$ such that $\langle\boldsymbol{u},\boldsymbol{v}\rangle=\langle\boldsymbol{u}',\boldsymbol{v}'\rangle$, no efficient adversary can distinguish between a ciphertext encoding $\boldsymbol{u}$ and a function key encoding $\boldsymbol{v}$, from a ciphertext encoding $\boldsymbol{u}'$ and a function key encoding $\boldsymbol{v}'$.

Furthermore, syntactically speaking, in a cIPE scheme, we will require the following to be true:

- The encryption algorithm just computes exponentiation and multiplication operations in G_1. The encryption of a vector $(a_1,..,a_4)$ can just be computed knowing $g_1^{a_i}$ for $i\in[4]$ and the master secret key.
- Key generation algorithm just computes exponentiation and multiplication operation in G_2. The function key for a vector $(b_1,..,b_4)$ can just be computed knowing $g_2^{b_i}$ for $i\in[4]$ and the master secret key.
- The decryption process just computes pairing operations and then computes group multiplications over G_t. The output is produced in G_t. The element g_t^a is represented as $[a]_t$ for the rest of the paper.

Such a cIPE scheme was given by [44], where it was instantiated from SXDH over bilinear maps. That work also used cIPE to build quadratic FE from SXDH.

We will also make use of cIPE in our construction of D-restricted FE. Note, however, that unlike in the case of quadratic FE, our construction, and crucially our proof of security, will also need to incorporate the "ladder" enforcement mechanism sketched above. We are able to do so still relying only on the SXDH assumption.

We note that the size of the vectors encrypted using a cIPE scheme cannot grow with n, to achieve linear efficiency. In fact, we just use four-dimensional vectors.

Realizing the Ladder: Warm-up Construction for $d+2=4$. Here is a warm-up construction for the case of $d=2$ (i.e. D=4).
$\underline{\mathsf{Setup}}(1^\lambda, 1^n)$: On input security parameter 1^λ and length 1^n,

- Run cIPE setup as follows. $\mathbf{sk}_0 \leftarrow \mathsf{cIPE.Setup}(1^\lambda, 1^4)$. Thus these keys are used to encrypt vectors in $\mathbb{F}_{\mathbf{p}}^4$.
- Then run cIPE setup algorithm $2 \cdot n$ times. That is, for every $\ell \in [2]$ and $i_\ell \in [n]$, compute $\mathbf{sk}^{(\ell,i_\ell)} \leftarrow \mathsf{cIPE.Setup}(1^\lambda, 1^4)$.
- Sample $\boldsymbol{\alpha} \leftarrow \mathbb{F}_{\mathbf{p}}^{2\times n}$. Also sample $\boldsymbol{\beta}, \boldsymbol{\gamma} \leftarrow \mathbb{F}_{\mathbf{p}}^n$.
- For every set $\boldsymbol{I} = (i_1, i_2, j, k)$ in $[n]^4$ do the following. Let $\boldsymbol{I}' = (i_2, j, k)$ and $\boldsymbol{I}'' = (j, k)$. Compute $\mathsf{Key}_{\boldsymbol{I}'}^{(1,i_1)} =$

$$\mathsf{cIPE.KeyGen}(\mathbf{sk}^{(1,i_1)}, (\boldsymbol{\alpha}[2,i_2]\boldsymbol{\beta}[j]\boldsymbol{\gamma}[k], \boldsymbol{\alpha}[1,i_1]\boldsymbol{\alpha}[2,i_2]\boldsymbol{\beta}[j]\boldsymbol{\gamma}[k], 0, 0))$$

  Similarly, compute $\mathsf{Key}_{\boldsymbol{I}''}^{(2,i_2)} =$

$$\mathsf{cIPE.KeyGen}(\mathbf{sk}^{(2,i_2)}, (\boldsymbol{\beta}[j]\boldsymbol{\gamma}[k], \boldsymbol{\alpha}[2,i_2]\boldsymbol{\beta}[j]\boldsymbol{\gamma}[k], 0, 0))$$

- Output $\mathsf{MSK} = (\{\mathbf{sk}^{(\ell,i_\ell)}, \mathsf{Key}_{\boldsymbol{I}}^{(\ell,i_\ell)}\}_{\ell,i_\ell,\boldsymbol{I}}, \boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma}, \mathbf{sk}_0)$

$\underline{\mathsf{Enc}(\mathsf{MSK}, \boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z})}$: The input message $M = (\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z})$ consists of a public attribute $\boldsymbol{x} \in \mathbb{F}_{\mathbf{p}}^{2\times n}$ and private vectors $\boldsymbol{y}, \boldsymbol{z} \in \mathbb{F}_{\mathbf{p}}^n$. Perform the following operations:

- Parse $\mathsf{MSK} = (\{\mathbf{sk}^{(\ell,i_\ell)}, \mathsf{Key}_{\boldsymbol{I}}^{(\ell,i_\ell)}\}_{\ell,i_\ell,\boldsymbol{I}}, \boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma}, \mathbf{sk}_0)$.
- Sample $r \leftarrow \mathbb{F}_{\mathbf{p}}$.
- Compute $\mathsf{CT}_0 = \mathsf{cIPE.Enc}(\mathbf{sk}_0, (r, 0, 0, 0))$.
- Sample $\mathbf{sk}' \leftarrow \mathsf{cIPE.Setup}(1^\lambda, 1^4)$.
- Compute $\mathsf{CTC}_j \leftarrow \mathsf{cIPE.Enc}(\mathbf{sk}, (\boldsymbol{y}[j], \boldsymbol{\beta}[j], 0, 0))$ for $j \in [n]$
- Compute $\mathsf{CTK}_k \leftarrow \mathsf{cIPE.Enc}(\mathbf{sk}, (\boldsymbol{z}[k], -r\boldsymbol{\gamma}[k], 0, 0))$ for $k \in [n]$.
- For every $\ell \in [2]$, $i_\ell \in [n]$, compute $\mathsf{CT}^{(\ell,i_\ell)} = \mathsf{cIPE.Enc}(\mathbf{sk}^{(\ell,i_\ell)}, (r\boldsymbol{x}[\ell, i_\ell], -r, 0, 0))$.
- Output $\mathsf{CT} = (\boldsymbol{x}, \mathsf{CT}_0, \{\mathsf{CTC}_j, \mathsf{CTK}_k, \mathsf{CT}^{(\ell,i_\ell)}\}_{\ell\in[2],i_\ell\in[n],j\in[n],k\in[n]})$

$\underline{\mathsf{KeyGen}(\mathsf{MSK}, f)}$: On input the master secret key MSK and function f,

- Parse $\mathsf{MSK} = (\{\mathbf{sk}^{(\ell,i_\ell)}, \mathsf{Key}_{\boldsymbol{I}}^{(\ell,i_\ell)}\}_{\ell,i_\ell,\boldsymbol{I}}, \boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma}, \mathbf{sk}_0)$.
- Compute $\theta_f = f(\boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma})$.

- Compute $\mathsf{Key}_{0,f} = \mathsf{cIPE.KeyGen}(\mathbf{sk}_0, (\theta_f, 0, 0, 0))$
- Output $sk_f = (\mathsf{Key}_{0,f}, \{\mathsf{Key}_{\boldsymbol{I}}^{(\ell,i_\ell)}\}_{\ell,i_\ell,\boldsymbol{I}})$.

Observe how the computation proceeds. This scheme allows to generate all terms in the ladder described above as follows:

Consider all terms associated with the vector $\boldsymbol{I} = (i_1, i_2, j, k) \in [n]^4$.

- $[\boldsymbol{y}[j]\boldsymbol{z}[k] - r\beta_j\gamma_k]_t = \mathsf{cIPE.Dec}(\mathsf{CTK}_k, \mathsf{CTC}_j)$
- $[r(\boldsymbol{x}[2,i_2] - \boldsymbol{\alpha}[2,i_2])\boldsymbol{\beta}[j]\boldsymbol{\gamma}[k]]_t = \mathsf{cIPE.Dec}(\mathsf{Key}_{\boldsymbol{I}''}^{(2,i_2)}, \mathsf{CT}^{(2,i_2)})$ where $\boldsymbol{I}'' = (j,k)$.
- $[r(\boldsymbol{x}[1,i_1] - \boldsymbol{\alpha}[1,i_1])\boldsymbol{\alpha}[2,i_2]\boldsymbol{\beta}[j]\boldsymbol{\gamma}[k]]_t = \mathsf{cIPE.Dec}(\mathsf{Key}_{\boldsymbol{I}'}^{(1,i_1)}, \mathsf{CT}^{(1,i_1)})$ where $\boldsymbol{I}'' = (i_2, j, k)$
- $[rf(\boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma})]_t = \mathsf{cIPE.Dec}(\mathsf{Key}_{0,f}, \mathsf{CT}_0)$.

Thus, we can compute $[f(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z})]_t$. We now briefly describe how security is proven.

Security Proof: Key Points. We use SXDH and function hiding property of the cIPE scheme crucially to argue security. The hybrid strategy is the following.

1. First we switch $\boldsymbol{y}$ to $\mathbf{0}$ vector in the challenge ciphertext, changing one component at a time.
2. To maintain correctness of output, we simultaneously introduce an offset in the function key to maintain correctness of decryption.
3. Once $\boldsymbol{y}$ is switched, $\boldsymbol{z}$ can be switched to vector $\mathbf{0}$, due to the function hiding property of the cIPE scheme. This is because the inner products remain the same in both the case as $\boldsymbol{y}$ is always $\mathbf{0}$ and inner product of any vector with all zero vector is 0. Finally, we are in the hybrid where the challenge ciphertext just depends on $\boldsymbol{x}$ and in particular totally independent of $\boldsymbol{y}$ and $\boldsymbol{z}$.

Step (1) is most challenging here, and requires careful pebbling and hard-wiring arguments made using SXDH and function hiding security property of cIPE. We point the reader to a detailed proof provided in Section **??**.

New $\Delta\mathsf{RG}$ candidates: Our construction of D-restricted FE enables us to meaningfully consider $\Delta\mathsf{RG}$ candidates that are implementable by D-restricted FE using degree-D polynomials. This enables a much richer class of potential $\Delta\mathsf{RG}$ candidates than those implementable by 3-restricted FE [5]. In Section 6, we describe a few of the new avenues for constructing $\Delta\mathsf{RG}$ candidates that we open by our construction of D-restricted FE.

Reader's Guide. The rest of the paper is organized as follows. In Section 3 we recall the definition of indistinguishability obfuscation and other prerequisites for the paper. In Section 4 we define formally the notions of $(d+2)$ restricted FE. Thereafter, in Section 5 perturbation resilient generator ($\Delta\mathsf{RG}$ for short) is

defined. Both primitives are central to this paper. In Section 6 we give candidate constructions of $\Delta\mathsf{RG}$ and show how to implement it using a $(d+2)$ restricted FE scheme. In Section 7 we show how to construct $(d+2)$ restricted FE using SXDH. Finally, in Section 8 we stitch all these primitives to show how to build obfuscation.

3 Preliminaries

We denote the security parameter by λ. For a distribution X we denote by $x \leftarrow X$ the process of sampling a value x from the distribution X. Similarly, for a set $\mathcal{X}$ we denote by $x \leftarrow \mathcal{X}$ the process of sampling x from the uniform distribution over $\mathcal{X}$. For an integer $n \in \mathbb{N}$ we denote by $[n]$ the set $\{1, .., n\}$. A function $\mathsf{negl} : \mathbb{N} \to \mathbb{R}$ is negligible if for every constant $c > 0$ there exists an integer N_c such that $\mathsf{negl}(\lambda) < \lambda^{-c}$ for all $\lambda > N_c$.

By $\approx_c$ we denote computational indistinguishability. We say that two ensembles $\mathcal{X} = \{\mathcal{X}_\lambda\}_{\lambda\in\mathbb{N}}$ and $\mathcal{Y} = \{\mathcal{Y}_\lambda\}_{\lambda\in\mathbb{N}}$ are computationally indistinguishable if for every probabilistic polynomial time adversary $\mathcal{A}$ there exists a negligible function negl such that $\left|\Pr_{x\leftarrow\mathcal{X}_\lambda}[\mathcal{A}(1^\lambda, x) = 1] - \Pr_{y\leftarrow\mathcal{Y}_\lambda}[\mathcal{A}(1^\lambda, y) = 1]\right| \leq \mathsf{negl}(\lambda)$ for every sufficiently large $\lambda \in \mathbb{N}$.

For a field element $a \in \mathbb{F}_{\mathbf{p}}$ represented in $[-p/2, p/2]$, we say that $-B < a < B$ for some positive integer B if its representative in $[-p/2, p/2]$ lies in $[-B, B]$.

Definition 1 (Distinguishing Gap). *For any adversary $\mathcal{A}$ and two distributions $\mathcal{X} = \{\mathcal{X}_\lambda\}_{\lambda\in\mathbb{N}}$ and $\mathcal{Y} = \{\mathcal{Y}_\lambda\}_{\lambda\in\mathbb{N}}$, define $\mathcal{A}$'s distinguishing gap in distinguishing these distributions to be* $|\Pr_{x\leftarrow\mathcal{X}_\lambda}[\mathcal{A}(1^\lambda, x) = 1] - \Pr_{y\leftarrow\mathcal{Y}_\lambda}[\mathcal{A}(1^\lambda, y) = 1]|$

By boldfaced letters such as $\boldsymbol{v}$ we will denote multidimensional matrices. Whenever dimension is unspecified we mean them as vectors.

Throughout, we denote by an adversary an interactive machine that takes part in a protocol with the challenger. Thus, we model such an adversary as a tuple of circuits $(C_1, ..., C_t)$ where t is the number of messages exchanged. Each circuit takes as input the state output by the previous circuit, among other messages. The size of adversary is defined as sum of size of each circuit.

3.1 Indistinguishability Obfuscation ($i\mathcal{O}$)

The notion of indistinguishability obfuscation ($i\mathcal{O}$), first conceived by Barak et al. [11], guarantees that the obfuscation of two circuits are computationally indistinguishable as long as they both are equivalent circuits, i.e., the output of both the circuits are the same on every input. Formally,

Definition 2 (Indistinguishability Obfuscator ($i\mathcal{O}$) for Circuits). *A uniform PPT algorithm $i\mathcal{O}$ is called an indistinguishability obfuscator for a circuit family $\{\mathcal{C}_\lambda\}_{\lambda\in\mathbb{N}}$, where $\mathcal{C}_\lambda$ consists of circuits C of the form $C : \{0,1\}^n \to \{0,1\}$ with $n = n(\lambda)$, if the following holds:*

– **Completeness:** *For every $\lambda \in \mathbb{N}$, every $C \in \mathcal{C}_\lambda$, every input $x \in \{0,1\}^n$, we have that*

$$\mathsf{Pr}\left[C'(x) = C(x) \ : \ C' \leftarrow i\mathcal{O}(\lambda, C)\right] = 1$$

– **Indistinguishability:** *For any PPT distinguisher D, there exists a negligible function $\mathsf{negl}(\cdot)$ such that the following holds: for all sufficiently large $\lambda \in \mathbb{N}$, for all pairs of circuits $C_0, C_1 \in \mathcal{C}_\lambda$ such that $C_0(x) = C_1(x)$ for all inputs $x \in \{0,1\}^n$ and $|C_0| = |C_1|$, we have:*

$$\left|\mathsf{Pr}\left[D(\lambda, i\mathcal{O}(\lambda, C_0)) = 1\right] - \mathsf{Pr}[D(\lambda, i\mathcal{O}(\lambda, C_1)) = 1]\right| \leq \mathsf{negl}(\lambda)$$

– **Polynomial Slowdown***: For every $\lambda \in \mathbb{N}$, every $C \in \mathcal{C}_\lambda$, we have that $|i\mathcal{O}(\lambda, C)| = \mathrm{poly}(\lambda, C)$.*

3.2 Bilinear Maps and Assumptions

Let PPGen be a probabilistic polynomial time algorithm that on input 1^λ returns a description $(e, G_1, G_2, G_T, g_1, g_2, \mathbf{p})$ of asymmetric pairing groups where G_1, G_2 and G_T are groups of order $\mathbf{p}$ for a 2λ bit prime $\mathbf{p}$. g_1 and g_2 are generators of G_1 and G_2 respectively. $e : G_1 \times G_2 \to G_T$ is an efficiently computable non-degenerate bilinear map. Define $g_t = e(g_1, g_2)$ as the generator of G_T.

Representation: We use the following representation to describe group elements. For any $b \in \{1, 2, T\}$ define by $[x]_b$ for $x \in \mathbb{F}_\mathbf{p}$ as g_b^x. This notation will be used throughout. We now describe SXDH assumption relative to PPGen.

Definition 3. *(SXDH Assumption relative to PPGen.) We say that $SXDH$ assumption holds relative to PPGen, if $(e, G_1, G_2, G_T, g_1, g_2, \mathbf{p}) \leftarrow \mathsf{PPGen}$, then for any group g_ℓ for $\ell \in \{1, 2, t\}$, it holds that, for any polynomial time adversary $\mathcal{A}$:*

$$\left|\Pr_{r,s,u \leftarrow \mathbb{F}_\mathbf{p}}[\mathcal{A}([r]_\ell, [s]_\ell, [r \cdot s]_\ell) = 1] - \Pr_{r,s,u \leftarrow \mathbb{F}_\mathbf{p}}[\mathcal{A}([r]_\ell, [s]_\ell, [u]_\ell) = 1]\right| \leq \mathsf{negl}(\lambda)$$

Further, if $\mathsf{negl}(\lambda)$ is $O(2^{-\lambda^c})$ for some $c > 0$, then we say that subexponential $SXDH$ holds relative to PPGen.

3.3 Canonical Function Hiding Inner Product FE

We now describe the notion of a canonical function hiding inner product FE proposed by [44]. A canonical function hiding scheme FE scheme consists of the following algorithms:

– $\mathsf{PPSetup}(1^\lambda) \to \mathsf{pp}$. On input the security parameter, $\mathsf{PPSetup}$, outputs parameters pp, which contain description of the groups and the plain text space $\mathbb{Z}_\mathbf{p}$.

- $\mathsf{Setup}(\mathsf{pp}, 1^n) \to \mathbf{sk}$. The setup algorithm takes as input the length of vector 1^n and parameters pp and outputs a secret key $\mathbf{sk}$. We assume that pp is always implicitly given as input to this algorithm and the algorithms below (sometimes we omit this for ease of notation).
- $\mathsf{Enc}(\mathbf{sk}, \boldsymbol{x}) \to \mathsf{CT}$. The encryption algorithm takes as input a vector $x \in \mathbb{Z}_{\mathbf{p}}^n$ and outputs a ciphertext CT.
- $\mathsf{KeyGen}(\mathbf{sk}, \boldsymbol{y}) \to \mathbf{sk}_{\boldsymbol{y}}$. The key generation algorithm on input the master secret key $\mathbf{sk}$ and a function vector $y \in \mathbb{Z}_{\mathbf{p}}^n$ and outputs a function key $\mathbf{sk}_{\boldsymbol{y}}$
- $\mathsf{Dec}(1^B, \mathbf{sk}_{\boldsymbol{y}}, \mathsf{CT}) \to m^*$. The decryption algorithm takes as input a ciphertext CT, a function key $\mathbf{sk}_{\boldsymbol{y}}$ and a bound B and it outputs a value m^*. Further, it is run in two steps. First step Dec_0, computes $[\langle \boldsymbol{x}, \boldsymbol{y} \rangle]_T$ (if the keys and ciphertexts were issued for $\boldsymbol{x}$ and $\boldsymbol{y}$) and then the second step, Dec_1, computes its discrete log, if this value lies in $[-B, B]$

A cIPE scheme satisfies linear efficiency, correctness, function hiding security and a canonical structure requirement. All of these are described in Section D.

4 Key Notion 1: $(d+2)-$restricted FE

In this section we describe the notion of a $(d+2)$-restricted functional encryption scheme (denoted by dFE). Let d denote any positive integer constant. Informally, dFE scheme is a functional encryption scheme that supports homogeneous polynomials of degree $d+2$ having degree 1 in $d+2$ input vectors. d out of those $d+2$ vectors are public. This is a generalisation of the notion of a three restricted FE scheme proposed by [5].

Notation: Throughout, we denote by boldfaced letters (multi-dimensional) matrices, where dimensions are either explicitly or implicitly defined.

Function class of interest: Consider a set of functions $\mathcal{F}_{\mathsf{dFE}} = \mathcal{F}_{\mathsf{dFE},\lambda,\mathbf{p},n} = \{f : \mathbb{F}_{\mathbf{p}}^{n(d+2)} \to \mathbb{F}_{\mathbf{p}}\}$ where $\mathbb{F}_{\mathbf{p}}$ is a finite field of order $\mathbf{p}(\lambda)$. Here n is seen as a function of λ. Each $f \in \mathcal{F}_{\lambda,\mathbf{p},n}$ takes as input $d+2$ vectors $(\boldsymbol{x}[1], ..., \boldsymbol{x}[d], \boldsymbol{y}, \boldsymbol{z})$ of length n over $\mathbb{F}_{\mathbf{p}}$ and computes a polynomial of the form $\Sigma c_{i_1,...,i_d,j,k} \cdot \boldsymbol{x}[1, i_1] \cdot ... \cdot \boldsymbol{x}[d, i_d] \cdot \boldsymbol{y}[j] \cdot \boldsymbol{z}[k]$, where $c_{i_1,..,i_d,j,k}$ are coefficients from $\mathbb{F}_{\mathbf{p}}$ for very $i_1, ..., i_d, j, k \in [n]^{d+2}$.

Syntax. Consider the set of functions $\mathcal{F}_{\mathsf{dFE},\lambda,\mathbf{p},n}$ as described above. A $(d+2)-$restricted functional encryption scheme dFE for the class of functions $\mathcal{F}_{\mathsf{dFE}}$ (described above) consists of the following PPT algorithms:

- **Setup,** $\mathsf{Setup}(1^\lambda, 1^n)$: On input security parameter λ (and the number of inputs $n = poly(\lambda)$), it outputs the master secret key MSK.
- **Encryption,** $\mathsf{Enc}(\mathsf{MSK}, \boldsymbol{x}[1], ..., \boldsymbol{x}[d], \boldsymbol{y}, \boldsymbol{z})$: On input the encryption key MSK and input vectors $\boldsymbol{x} \in \mathbb{F}_{\mathbf{p}}^{d \times n}$, $\boldsymbol{y}$ and $\boldsymbol{z}$ (all in $\mathbb{F}_{\mathbf{p}}^n$) it outputs ciphertext CT. Here $\boldsymbol{x}$ is seen as a public attribute and $\boldsymbol{y}$ and $\boldsymbol{z}$ are thought of as private messages.

- **Key Generation,** $\mathsf{KeyGen}(\mathsf{MSK}, f)$: On input the master secret key MSK and a function $f \in \mathcal{F}_{\mathsf{dFE}}$, it outputs a functional key $sk[f]$.
- **Decryption,** $\mathsf{Dec}(sk[f], 1^B, \mathsf{CT})$: On input functional key $sk[f]$, a bound $B = poly(\lambda)$ and a ciphertext CT, it outputs the result out.

We define correctness property below.

B-Correctness. Consider any function $f \in \mathcal{F}_{\mathsf{dFE}}$ and any plaintext $\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z} \in \mathbb{F}_{\mathbf{p}}$ (dimensions are defined above). Consider the following process:

- $sk[f] \leftarrow \mathsf{KeyGen}(\mathsf{MSK}, f)$.
- $\mathsf{CT} \leftarrow \mathsf{Enc}(\mathsf{MSK}, \boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z})$
- If $f(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}) \in [-B, B]$, set $\theta = f(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z})$, otherwise set $\theta = \perp$.

The following should hold:

$$\mathsf{Pr}\left[\mathsf{Dec}(sk[f], 1^B, \mathsf{CT}) = \theta\right] \geq 1 - \mathsf{negl}(\lambda),$$

for some negligible function negl.

Linear Efficiency: We require that for any message $(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z})$ where $\boldsymbol{x} \in \mathbb{F}_{\mathbf{p}}^{d \times n}$ and $\boldsymbol{y}, \boldsymbol{z} \in \mathbb{F}_{\mathbf{p}}^{n}$ the following happens:

- Let $\mathsf{MSK} \leftarrow \mathsf{Setup}(1^\lambda, 1^n)$.
- Compute $\mathsf{CT} \leftarrow \mathsf{Enc}(\mathsf{MSK}, \boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z})$.

The size of encryption circuit computing CT is less than $n \times (d+2) \log_2 \mathbf{p} \cdot poly(\lambda)$. Here $poly$ is some polynomial independent of n.

4.1 Semi-functional Security

We define the following auxiliary algorithms.

Semi-functional Key Generation, $\mathsf{sfKG}(\mathsf{MSK}, f, \theta)$: On input the master secret key MSK, function f and a value θ, it computes the semi-functional key $sk[f, \theta]$.

Semi-functional Encryption, $\mathsf{sfEnc}(\mathsf{MSK}, \boldsymbol{x}, 1^{|\boldsymbol{y}|}, 1^{|\boldsymbol{z}|})$: On input the master encryption key MSK, a public attribute $\boldsymbol{x}$ and length of messages $\boldsymbol{y}, \boldsymbol{z}$, it computes a semi-functional ciphertext $\mathsf{ct}_{\mathsf{sf}}$.

We define two security properties associated with the above two auxiliary algorithms. We will model the security definitions along the same lines as semi-functional FE.

Definition 4 (Indistinguishability of Semi-functional Ciphertexts). *A $(d+2)$-restricted functional encryption scheme dFE for a class of functions $\mathcal{F}_{\mathsf{dFE}} = \{\mathcal{F}_{\mathsf{dFE},\lambda,\mathbf{p},n}\}_{\lambda \in \mathbb{N}}$ is said to satisfy* **indistinguishability of semi-functional ciphertexts property** *if there exists a constant $c > 0$ such that for sufficiently*

large $\lambda \in \mathbb{N}$ and any adversary $\mathcal{A}$ of size 2^{λ^c}, the probability that $\mathcal{A}$ succeeds in the following experiment is $2^{-\lambda^c}$.

$\mathsf{Expt}(1^\lambda, \mathbf{b})$*:*

1. *$\mathcal{A}$ specifies the following:*
   - *Challenge message $M^* = (\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z})$. Here $\boldsymbol{y}, \boldsymbol{z}$ is in $\mathbb{F}_{\mathbf{p}}^n$ and $\boldsymbol{x}$ is in $\mathbb{F}_{\mathbf{p}}^{d \times n}$.*
   - *It can also specify additional messages $\{M_k = (\boldsymbol{x}_k, \boldsymbol{y}_k, \boldsymbol{z}_k)\}_{k \in [q]}$ Here $\boldsymbol{y}_k, \boldsymbol{z}_k$ is in $\mathbb{F}_{\mathbf{p}}^n$ and $\boldsymbol{x}_k$ is in $\mathbb{F}_{\mathbf{p}}^{d \times n}$. Here q is a polynomial in n, λ.*
   - *It also specifies functions $f_1, \ldots, f_\eta$ and hardwired values $\theta_1, \ldots, \theta_\eta$ where η is a polynomial in n, λ.*
2. *The challenger checks if $\theta_k = f_k(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z})$ for every $k \in [\eta]$. If this check fails, the challenger aborts the experiment.*
3. *The challenger computes the following*
   - *Compute $sk[f_k, \theta_k] \leftarrow \mathsf{sfKG}(\mathsf{MSK}, f_k, \theta_k)$, for every $k \in [\eta]$.*
   - *If $\mathbf{b} = 0$, compute $\mathsf{CT}^* \leftarrow \mathsf{sfEnc}(\mathsf{MSK}, \boldsymbol{x}, 1^{|\boldsymbol{y}|}, 1^{|\boldsymbol{z}|})$. Else, compute $\mathsf{CT}^* \leftarrow \mathsf{Enc}(\mathsf{MSK}, \boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z})$.*
   - *$\mathsf{CT}_i \leftarrow \mathsf{Enc}(\mathsf{MSK}, M_i)$, for every $i \in [q]$.*
4. *The challenger sends $\left(\{\mathsf{CT}_i\}_{i \in [q]}, \mathsf{CT}^*, \{sk[f_k, \theta_k]\}_{k \in [\eta]}\right)$ to $\mathcal{A}$.*
5. *The adversary outputs a bit b'.*

We say that the adversary $\mathcal{A}$ succeeds in $\mathsf{Expt}(1^\lambda, \mathbf{b})$ with probability ε if it outputs $b' = \mathbf{b}$ with probability $\frac{1}{2} + \varepsilon$.

We now define indistinguishability of semi-functional keys property.

Definition 5 (Indistinguishability of Semi-functional Keys). *A $(d+2)$-restricted FE scheme dFE for a class of functions $\mathcal{F}_{\mathsf{dFE}} = \{\mathcal{F}_{\mathsf{dFE},\lambda,\mathbf{p},n}\}_{\lambda \in \mathbb{N}}$ is said to satisfy* **indistinguishability of semi-functional keys property** *if there exists a constant $c > 0$ such that for all sufficiently large λ, any PPT adversary $\mathcal{A}$ of size 2^{λ^c}, the probability that $\mathcal{A}$ succeeds in the following experiment is $2^{-\lambda^c}$.*

$\mathsf{Expt}(1^\lambda, \mathbf{b})$*:*

1. *$\mathcal{A}$ specifies the following:*
   - *It can specify messages $M_j = \{(\boldsymbol{x}_i, \boldsymbol{y}_i, \boldsymbol{z}_i)\}_{j \in [q]}$ for some polynomial q. Here $\boldsymbol{y}_i, \boldsymbol{z}_i$ is in $\mathbb{F}_{\mathbf{p}}^n$ and $\boldsymbol{x}_i$ is in $\mathbb{F}_{\mathbf{p}}^{d \times n}$.*
   - *It specifies functions $f_1, \ldots, f_\eta \in \mathcal{F}_{\mathsf{dFE}}$ and hardwired values $\theta_1, \ldots, \theta_\eta \in \mathbb{F}_p$. Here η is some polynomial in λ, n.*
2. *Challenger computes the following :*
   - *If $\mathbf{b} = 0$, compute $sk[f_i]^* \leftarrow \mathsf{KeyGen}(\mathsf{MSK}, f_i)$ for all $i \in [\eta]$. Otherwise, compute $sk[f_i]^* \leftarrow \mathsf{sfKG}(\mathsf{MSK}, f_i, \theta_i)$ for all $i \in [\eta]$.*
   - *$\mathsf{CT}_i \leftarrow \mathsf{Enc}(\mathsf{MSK}, M_j)$, for every $j \in [q]$.*
3. *Challenger then sends $\left(\{\mathsf{CT}_i\}_{i \in [q]}, \{sk[f_i]^*\}_{i \in [\eta]}\right)$ to $\mathcal{A}$.*
4. *$\mathcal{A}$ outputs b'.*

The success probability of $\mathcal{A}$ is defined to be ε if $\mathcal{A}$ outputs $b' = \mathbf{b}$ with probability $\frac{1}{2} + \varepsilon$.

If a $(d+2)$-restricted FE scheme satisfies both the above definitions, then it is said to satisfy semi-functional security.

Definition 6 (Semi-functional Security). *Consider a $(d+2)$-restricted FE scheme* dFE *for a class of functions $\mathcal{F}$. We say that* dFE *satisfies* **semi-functional security** *if it satisfies indistinguishability of semi-functional ciphertexts property (Definition 4) and indistinguishability of semi-functional keys property (Definition 5).*

Remark: Two remarks are in order:

1. First, we define sub-exponential security here as that notion is useful for our construction of $i\mathcal{O}$. The definition can be adapted to polynomial security naturally.
2. Semi-functional security implies indistinguishability based notion naturally. This is pointed out in [7].

5 Key Notion 2: Perturbation Resilient Generator

Now we describe the notion of a Perturbation Resilient Generator ($\Delta\mathsf{RG}$ for short), proposed by [5]. A $\Delta\mathsf{RG}$ consists of the following algorithms:

- $\mathsf{Setup}(1^\lambda, 1^n, B) \to (\mathsf{pp}, \mathsf{Seed})$. The setup algorithm takes as input a security parameter λ, the length parameter 1^n and a polynomial $B = B(\lambda)$ and outputs a seed $\mathsf{Seed} \in \{0,1\}^*$ and public parameters pp.
- $\mathsf{Eval}(\mathsf{pp}, \mathsf{Seed}) \to (h_1, ..., h_\ell)$, evaluation algorithm output a vector $(h_1, ..., h_\ell) \in \mathbb{Z}^\ell$. Here ℓ is the stretch of $\Delta\mathsf{RG}$.

We have following properties of a $\Delta\mathsf{RG}$ scheme.

Efficiency: We require for $\mathsf{Setup}(1^\lambda, 1^n, B) \to (\mathsf{pp}, \mathsf{Seed})$ and $\mathsf{Eval}(\mathsf{pp}, \mathsf{Seed}) \to (h_1, ..., h_\ell)$,

- $|\mathsf{Seed}| = n \cdot poly(\lambda)$ for some polynomial $poly$ independent of n. The size of Seed is linear in n.
- For all $i \in [\ell]$, $|h_i| < poly(\lambda, n)$. The norm of each output component h_i in $\mathbb{Z}$ is bounded by some polynomial in λ and n.

Perturbation Resilience: We require that for large enough security parameter λ, for every polynomial B, there exists a large enough polynomial $n_B(\lambda)$ such that for any $n > n_B$, there exists an efficient sampler $\mathcal{H}$ such that for $\mathsf{Setup}(1^\lambda, 1^n, B) \to (\mathsf{pp}, \mathsf{Seed})$ and $\mathsf{Eval}(\mathsf{pp}, \mathsf{Seed}) \to (h_1, ..., h_\ell)$, we have that for any distinguisher D of size 2^λ

$$|\Pr[D(x \xleftarrow{\$} \mathcal{D}_1) = 1] - \Pr[D(x \xleftarrow{\$} \mathcal{D}_2) = 1]| < 1 - 2/\lambda$$

Here $\mathcal{D}_1$ and $\mathcal{D}_2$ are defined below:

- Distribution $\mathcal{D}_1$: Compute $\mathsf{Setup}(1^\lambda, 1^n, B) \to (\mathsf{pp}, \mathsf{Seed})$ and $\mathsf{Eval}(\mathsf{pp}, \mathsf{Seed}) \to (h_1, ..., h_\ell)$. Output $(\mathsf{pp}, h_1, ..., h_\ell)$.
- Distribution $\mathcal{D}_2$: Compute $\mathsf{Setup}(1^\lambda, 1^n, B) \to (\mathsf{pp}, \mathsf{Seed})$ and $\mathcal{H}(\mathsf{pp}, \mathsf{Seed}) \to (h_1, .., h_\ell)$. Output $(\mathsf{pp}, h_1 + a_1, ..., h_\ell + a_\ell)$.

Remark: Note that one could view ΔRG as a candidate sampler $\mathcal{H}$ itself.

Now we describe the notion of Perturbation Resilient Generator implementable by a $(d+2)$-restricted FE scheme (dΔRG for short.)

5.1 ΔRG implementable by $(d+2)-$restricted FE

A ΔRG scheme implementable by $(d+2)$-Restricted FE (dΔRG for short) is a perturbation resilient generator with additional properties. We describe syntax again for a complete specification.

- $\mathsf{Setup}(1^\lambda, 1^n, B) \to (\mathsf{pp}, \mathsf{Seed})$. The setup algorithm takes as input a security parameter λ, the length parameter 1^n and a polynomial $B = B(\lambda)$ and outputs a seed Seed and public parameters pp. Here, $\mathsf{Seed} = (\mathsf{Seed.pub}(1), \mathsf{Seed.pub}(2), ..., \mathsf{Seed.pub}(d), \mathsf{Seed.priv}(1), \mathsf{Seed.priv}(2))$ is a vector on $\mathbb{F}_{\mathbf{p}}$ for a modulus $\mathbf{p}$, which is also the modulus used in $(d+2)$-restricted FE scheme. There are $d+2$ components of this vector, where d of the $d+2$ componens are public and two components are private, each in $\mathbb{F}_{\mathbf{p}}^{npoly(\lambda)}$. Also each part can be partitioned into subcomponents as follows. $\mathsf{Seed.pub}(j) = (\mathsf{Seed.pub}(j,1), ..., \mathsf{Seed.pub}(j,n))$ for $j \in [d]$, $\mathsf{Seed.priv}(j) = (\mathsf{Seed.priv}(j,1), ..., \mathsf{Seed.priv}(j,n))$ for $j \in [2]$. Here, each sub component is in $\mathbb{F}_{\mathbf{p}}^{poly(\lambda)}$ for some fixed polynomial *poly* independent of n. Also, $\mathsf{pp} = (\mathsf{Seed.pub}(1), \ldots, \mathsf{Seed.pub}(d), q_1, .., q_\ell)$ where each q_i is a degree $d+2$ multilinear polynomial described below. We require syntactically there exists two algorithms $\mathsf{SetupSeed}$ and $\mathsf{SetupPoly}$ such that Setup can be decomposed follows:
  1. $\mathsf{SetupSeed}(1^\lambda, 1^n, B) \to \mathsf{Seed}$. The $\mathsf{SetupSeed}$ algorithm outputs the seed.
  2. $\mathsf{SetupPoly}(1^\lambda, 1^n, B) \to q_1, ..., q_\ell$. The $\mathsf{SetupPoly}$ algorithm outputs $q_1, .., q_\ell$.
- $\mathsf{Eval}(\mathsf{pp}, \mathsf{Seed}) \to (h_1, ..., h_\ell)$, evaluation algorithm output a vector $(h_1, ..., h_\ell) \in \mathbb{Z}^\ell$. Here for $i \in [\ell]$, $h_i = q_i(\mathsf{Seed})$ and ℓ is the stretch of dΔRG. Here q_i is a homoegenous multilinear degree $d+2$ polynomial where each monomial has degree 1 in $\{\mathsf{pub}(j)\}_{j\in[d+2]}$ and $\{\mathsf{priv}(j)\}_{j\in[2]}$ components of the seed.

The security and efficiency requirements are same as before.

Remark: To construct $i\mathcal{O}$ we need the stretch of dΔRG to be equal to $\ell = n^{1+\epsilon}$ for some constant $\epsilon > 0$.

6 dΔRG Candidates

We now describe our candidate for dΔRG implementable by a $(d+2)-$ restricted FE scheme. All these candidates use a large enough prime modulus $\mathbf{p} = O(2^\lambda)$, which is the same as the modulus used by $(d+2)-$restricted FE. Then, let χ be a distribution used to sample input elements over $\mathbb{Z}$. Let Q denote a polynomial sampler. Next we give candidate in terms of χ and Q but give concrete instantiations later.

6.1 dΔRG Candidate

– $\mathsf{Setup}(1^\lambda, 1^n, B) \to (\mathsf{pp}, \mathsf{Seed})$. Sample a secret $\boldsymbol{s} \leftarrow \mathbb{F}_{\mathbf{p}}^{1\times \mathsf{n}_{\Delta\mathsf{RG}}}$ for $\mathsf{n}_{\Delta\mathsf{RG}} = poly(\lambda)$ such that $\mathsf{LWE}_{\mathsf{n}_{\Delta\mathsf{RG}}, n\cdot d, \mathbf{p}, \chi}$ holds. Here χ is a bounded distribution with bound $poly(\lambda)$ (see Section C.1 for definitions). Let $\mathcal{Q}$ denote an efficiently samplable distribution of homogeneous degree $(d+2)$ polynomials (instantiated later). Then proceed with $\mathsf{SetupSeed}$ as follows:

1. Sample $\boldsymbol{a}_{i,j} \leftarrow \mathbb{F}_{\mathbf{p}}^{1\times \mathsf{n}_{\Delta\mathsf{RG}}}$ for $i \in [d], j \in [n]$.
2. Sample $e_{i,j} \leftarrow \chi$ for $i \in [d], j \in [n]$.
3. Compute $r_{i,j} = \langle \boldsymbol{a}_{i,j}, \boldsymbol{s}\rangle + e_{i,j} \mod \mathbf{p}$ in $\mathbb{F}_{\mathbf{p}}$ for $i \in [d], j \in [n]$.
4. Define $\boldsymbol{w}_{i,j} = (\boldsymbol{a}_{i,j}, r_{i,j})$ for $i \in [d], j \in [d]$.
5. Set $\mathsf{Seed.pub}(j,i) = \boldsymbol{w}_{j,i}$ for $j \in [d], i \in [n]$.
6. Sample $y_i, z_i \leftarrow \chi$ for $i \in [n]$.
7. Set $\boldsymbol{t} = (-\boldsymbol{s}, 1)$. Note that $\langle \boldsymbol{w}_{j,i}, \boldsymbol{t}\rangle = e_{j,i}$ for $j \in [d], i \in [n]$.
8. Set $\boldsymbol{y}'_i = y_i \otimes^d \boldsymbol{t}$. (tensor $\boldsymbol{t}$, d times)
9. Set $\mathsf{Seed.priv}(1,i) = \boldsymbol{y}'_i$ for $i \in [n]$.
10. Set $\mathsf{Seed.priv}(2,i) = z_i$ for $i \in [n]$.

Now we describe $\mathsf{SetupPoly}$. Fix $\eta = n^{1+\epsilon}$.

1. Write $\boldsymbol{e}_j = (e_{j,1}, \ldots, e_{j,n})$ for $j \in [d]$, $\boldsymbol{y} = (y_1, \ldots, y_n)$ and $\boldsymbol{z} = (z_1, \ldots, z_n)$.
2. Sample polynomials q'_ℓ for $\ell \in [\eta]$ as follows.
3. $q'_\ell = \Sigma_{\boldsymbol{I}=(i_1,..,i_d,j,k)} c_{\boldsymbol{I}} e_{1,i_1} \cdots e_{d,i_d} y_j z_k$ where coefficients $c_{\boldsymbol{I}}$ are bounded by $poly(\lambda)$. These polynomials are sampled according to $\mathcal{Q}$
4. Define q_i be a multilinear homogeneous degree $d+2$ polynomial takes as input $\mathsf{Seed} = (\{\boldsymbol{w}_{j,i}\}_{j\in[d],i\in[n]}, \boldsymbol{y}'_1, \ldots, \boldsymbol{y}'_n, \boldsymbol{z})$. Then it computes each monomial $c_{\boldsymbol{I}} e_{1,i_1} \cdots e_{d,i_d} y_j z_k$ as follows and then adds all the results (thus computes $q'_i(\boldsymbol{e}_1, \ldots, \boldsymbol{e}_d, \boldsymbol{y}, \boldsymbol{z})$):
   - Compute $c_{\boldsymbol{I}} \langle \boldsymbol{w}_{1,i_1}, \boldsymbol{t}\rangle \cdots \langle \boldsymbol{w}_{d,i_d}, \boldsymbol{t}\rangle y_j z_k$. This step requires $\boldsymbol{y}'_i = y_i \otimes^d \boldsymbol{t}$ to perform this computation.
5. Output $q_1, ..., q_\eta$.

– $\mathsf{Eval}(\mathsf{pp}, \mathsf{Seed}) \to (h_1, ..., h_\eta)$, evaluation algorithm output a vector $(h_1, ..., h_\eta) \in \mathbb{Z}^\eta$. Here for $i \in [\eta]$, $h_i = q_i(\mathsf{Seed})$ and η is the stretch of $\mathsf{d}\Delta\mathsf{RG}$. Here q_i is a degree $d+2$ homogenenous multilinear polynomial (defined above) which is degree 1 in d public and 2 private components of the seed.

Efficiency:

1. Note that Seed contains $n \cdot d$ LWE samples $\boldsymbol{w}_{i,j}$ for $i \in [d], j \in [n]$ of dimension $\mathsf{n}_{\Delta\mathsf{RG}}$. Along with the samples, it contains elements $\boldsymbol{y}'_i = y_i \otimes^d \boldsymbol{t}$ for $i \in [n]$ and elements z_i for $i \in [n]$. Note that the size of these elements are bounded by $poly(\lambda)$ and is independent of n.
2. The values $h_i = q_i(\mathsf{Seed}) = \Sigma_{\boldsymbol{I}=(i_1,..,i_d,j,k)} c_{\boldsymbol{I}} e_{1,i_1} \cdots e_{d,i_d} y_j z_k$. Since χ is a bounded distribution, bounded by $poly(\lambda, n)$, and coefficients $c_{\boldsymbol{I}}$ are also polynomially bounded, each $|h_i| < poly(\lambda, n)$ for $i \in [m]$.

6.2 Instantiations

We now give various instantiations of Q. Let χ be the discrete gaussian distribution with 0 mean and standard deviation n. The following candidate is proposed by [12] based on the investigation of the hardness of families of expanding polynomials over the reals.

Instantiation 1: 3SAT Based Candidate. Let $t = B^2\lambda$. Sample each polynomial q'_i for $i \in [\eta]$ as follows. $q'_i(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_t, \boldsymbol{y}_1, \ldots, \boldsymbol{y}_t, \boldsymbol{z}_1, \ldots, \boldsymbol{z}_t) = \Sigma_{j\in[t]} q'_{i,j}(\boldsymbol{x_j}, \boldsymbol{y_j}, \boldsymbol{z_j})$. Here $\boldsymbol{x}_j \in \chi^{d\times n}$ and $\boldsymbol{y}_j, \boldsymbol{z}_j \in \chi^n$ for $j \in [t]$. In other words, q'_i is a sum of t polynomials $q'_{i,j}$ over t disjoint set of variables. Let $d = 1$ for this candidate.

Now we describe how to sample $q'_{i,j}$ for $j \in [\eta]$.

1. Sample randomly inputs $\boldsymbol{x}^*, \boldsymbol{y}^*, \boldsymbol{z}^* \in \{0,1\}^n$.
2. To sample $q'_{i,j}$ do the following. Sample three indices randomly and independently $i_1, i_2, i_3 \leftarrow [n]$. Sample three signs $b_{1,i,j}, b_{2,i,j}, b_{3,i,j} \in \{0,1\}$ uniformly such that $b_{1,i,j} \oplus b_{2,i,j} \oplus b_{3,i,j} \oplus \boldsymbol{x}^*[i_1] \oplus \boldsymbol{y}^*[i_2] \oplus \boldsymbol{z}^*[i_3] = 1$.
3. Set $q'_{i,j}(\boldsymbol{x_j}, \boldsymbol{y_j}, \boldsymbol{z_j}) = 1 - (b_{1,i,j} \cdot \boldsymbol{x_j}[i_1] + (1 - b_{1,i,j}) \cdot (1 - \boldsymbol{x_j}[i_1])) \cdot (b_{2,i,j} \cdot \boldsymbol{y_j}[i_2] + (1 - b_{2,i,j}) \cdot (1 - \boldsymbol{y_j}[i_2])) \cdot ((b_{3,i,j} \cdot \boldsymbol{z_j}[i_3] + (1 - b_{3,i,j}) \cdot (1 - \boldsymbol{z_j}[i_3]))$

Remark:

1. Note that any clause of the form $a_1 \vee a_2 \vee a_3$ can be written as $1 - (1 - a_1)(1 - a_2)(1 - a_3)$ over integers where a_1, a_2, a_3 are literals in first case and take values in $\{0, 1\}$, and thus any random satisfiable 3SAT formula can be converted to polynomials in this manner.
2. Similarly, the above construction can be generalised to degree $(d + 2)$-SAT style construction by considering $(d + 2)-$SAT for any constant positive integer d and translating them to polynomials.

Instantiation 2: Goldreich's One-way Function Based Candidate. Goldreich's one-way function [35] consists of a predicate P involving $d + 2$ variables and computes a boolean function that can be expressed a degree $d + 2$ polynomial over the integers. Our candidate $q'_{i,j}(\boldsymbol{x_j}, \boldsymbol{y_j}, \boldsymbol{z_j})$ consists of the following step.

1. Sample $d + 2$ indices $i_1, ..., i_{d+2} \in [n]$.
2. Output $q'_{i,j} = P(\boldsymbol{x_j}[1, i_1], \ldots, \boldsymbol{x_j}[d, i_d], \boldsymbol{y_j}[i_{d+1}], \boldsymbol{z_j}[i_{d+2}])$.

For $d = 3$, [53] provided with the following candidate.

$P(a_1, .., a_5) = a_1 \oplus a_2 \oplus a_3 \oplus a_4 a_5$ where each $a_i \in \{0, 1\}$. Note that this can be naturally converted to a polynomial as follows. Rewrite $a \oplus b = (1-a)b + (1-b)a$ and this immediately gives rise to a polynomial over the integers.

6.3 Simplifying Assumptions

In this section, we remark that $\mathsf{d}\Delta\mathsf{RG}$ assumption can be simplified from being an exponential family of assumptions to two simpler assumptions as follows. We provide two sub-assumptions, which together imply $\mathsf{d}\Delta\mathsf{RG}$ assumptions.

LWE with degree $d+2$ leakage. There exists a polynomial sampler Q and a constant $\epsilon > 0$, such that for every large enough $\lambda \in \mathbb{N}$, and every polynomial bound $B = B(\lambda)$ there exist large enough polynomial $n_B = \lambda^c$ such that for every positive integer $n > n_B$, there exists a $poly(n)-$bounded discrete gaussian distribution χ such that the following two distrbutions are close (we define closeness later). We define the following two distributions:

Distribution $\mathcal{D}_1$:

- Fix a prime modulus $p = O(2^\lambda)$.
- (**Sample Polynomials.**) Run $Q(n, B, \epsilon) = (q_1, ..., q_{\lfloor n^{1+\epsilon} \rfloor})$.
- (**Sample Secret.**) Sample a secret $\boldsymbol{s} \leftarrow \mathbb{Z}_p^\lambda$
- Sample $\boldsymbol{a}_{j,i} \leftarrow \mathbb{Z}_p^\lambda$ for $j \in [d], i \in [n]$.
- (**Sample LWE Errors.**) For every $j \in [d], i \in [n]$, sample $e_{j,i}, y_i, z_i \leftarrow \chi$. Write $\boldsymbol{e}_j = (e_{j,1}, \ldots, e_{j,n})$ for $j \in [d]$, $\boldsymbol{y} = (y_1, \ldots, y_n)$ and $\boldsymbol{z} = (z_1, \ldots, z_n)$.
- Output $\{\boldsymbol{a}_{j,i}, \langle \boldsymbol{a}_{j,i}, \boldsymbol{s} \rangle + e_{j,i} \mod p\}_{j \in [d], i \in [n]}$ and $\{q_k, q_k(\boldsymbol{e}_1, \ldots, \boldsymbol{e}_d, \boldsymbol{y}, \boldsymbol{z})\}_{k \in [\lfloor n^{1+\epsilon} \rfloor]}$

Distribution $\mathcal{D}_2$:

- Fix a prime modulus $p = O(2^\lambda)$.
- (**Sample Polynomials.**) Run $Q(n, B, \epsilon) = (q_1, ..., q_{\lfloor n^{1+\epsilon} \rfloor})$.
- (**Sample Secret.**) Sample a secret $\boldsymbol{s} \leftarrow \mathbb{Z}_p^\lambda$
- Sample $\boldsymbol{a}_{j,i} \leftarrow \mathbb{Z}_p^\lambda$ for $j \in [d], i \in [n]$.
- (**Sample independent LWE Errors.**) For every $j \in [d], i \in [n]$, sample $e_{j,i}, e'_{j,i}, y_i, z_i \leftarrow \chi$. [1] Write $\boldsymbol{e}'_j = (e'_{j,1}, \ldots, e'_{j,n})$, $\boldsymbol{e}_j = (e_{j,1}, \ldots, e_{j,n})$ for $j \in [d]$, $\boldsymbol{y} = (y_1, \ldots, y_n)$ and $\boldsymbol{z} = (z_1, \ldots, z_n)$.
- Output $\{\boldsymbol{a}_{j,i}, \langle \boldsymbol{a}_{j,i}, \boldsymbol{s} \rangle + e'_{j,i} \mod p\}_{j \in [d], i \in [n]}$ and $\{q_k, q_k(\boldsymbol{e}_1, \ldots, \boldsymbol{e}_d, \boldsymbol{y}, \boldsymbol{z})\}_{k \in [\lfloor n^{1+\epsilon} \rfloor]}$

The assumption states that there exists a constant $\epsilon_{adv} > 0$ such that for any adversary $\mathcal{A}$ of size $2^{\lambda^{\epsilon_{adv}}}$, the following holds:

$$|\Pr[\mathcal{A}(\mathcal{D}_1) = 1] - \Pr[\mathcal{A}(\mathcal{D}_2) = 1]| < 1/2\lambda$$

Remark. This assumption says that to a bounded adversary, the advantage of distinguishing the tuple consisting of polynomials samples, along with correlated LWE samples with tuple consisting of polynomials samples, along with uncorrelated LWE samples is bounded by $1/2\lambda$. Second assumption says that the tuple of polynomial samples looks close to independent discrete gaussian variables with a large enough variance and 0 mean. Below we define the notion of a $(B, \delta)-$smooth distribution.

Definition 7. *$(B, \delta)-$Smooth distribution $\mathcal{N}$ is an efficiently samplable distribution over $\mathbb{Z}$ with the property that $\Delta(\mathcal{N}, \mathcal{N} + b) \leq \delta$ for any $b \in [-B, B]$.*

[1] Thus, we can observe that χ should be a distribution such that LWE assumption holds with respect to χ and parameters specified above

Weak Pseudo-Independence Generator Assumption [3, 46]. For the parameters defined above, the assumption states that there exists a constant $\epsilon_{adv} > 0$ such that for any adversary $\mathcal{A}$ of size $2^{\lambda^{\epsilon_{adv}}}$, the following holds:

$$|\Pr[\mathcal{A}(\mathcal{D}_1) = 1] - \Pr[\mathcal{A}(\mathcal{D}_2) = 1]| < 1 - 3/\lambda$$

where distributions are defined below.
Distribution $\mathcal{D}_1$:

- Fix a prime modulus $p = O(2^\lambda)$.
- (**Sample Polynomials.**) Run $Q(n, B, \epsilon) = (q_1, ..., q_{\lfloor n^{1+\epsilon} \rfloor})$.
- For every $j \in [d], i \in [n]$, sample $e_{j,i}, y_i, z_i \leftarrow \chi$. Write $\boldsymbol{e}_j = (e_{j,1}, \ldots, e_{j,n})$ for $j \in [d]$, $\boldsymbol{y} = (y_1, \ldots, \boldsymbol{y}_n)$ and $\boldsymbol{z} = (z_1, \ldots, z_n)$.
- Output $\{q_k, q_k(\boldsymbol{e}_1, \ldots, \boldsymbol{e}_d, \boldsymbol{y}, \boldsymbol{z})\}_{k \in [\lfloor n^{1+\epsilon} \rfloor]}$

Distribution $\mathcal{D}_2$:

- Fix a prime modulus $p = O(2^\lambda)$.
- (**Sample Polynomials.**) Run $Q(n, B, \epsilon) = (q_1, ..., q_{\lfloor n^{1+\epsilon} \rfloor})$.
- Output $\{q_k, h_k \leftarrow \mathcal{N}\}_{k \in [\lfloor n^{1+\epsilon} \rfloor]}$

Here $\mathcal{N}$ is a $(B, \frac{1}{n^2\lambda})-$smooth distribution.
Thus we have,

Claim. Assuming,

1. LWE with degree $d + 2$ leakage.
2. Weak Pseudo-Independence Generator Assumption

There exists a $\mathsf{d}\Delta\mathsf{RG}$ scheme.

Proof. (Sketch.) This is immediate and the proof goes in three hybrids. First, we use LWE with degree $d+2$ leakage assumption with $1/2\lambda$ security loss. In the next hybrid, we sample from $\mathcal{N}$ given to us by Weak Pseudo-Independence Generator Assumption. With that, we have another $1-3/\lambda$ loss in the security. Finally, we move to a hybrid where all perturbations are 0. This leads to a security loss of $n^{1+\epsilon} \times \frac{1}{n^2\lambda} < \frac{1}{n^{1-\epsilon}\lambda}$ due to statistical distance. Adding these security losses, we prove the claim. Thus $\mathcal{H}$ just uses $\mathcal{N}$ to sample each component independently.

7 Constructing $(d + 2)$ restricted FE from bilinear maps

In this section we describe our construction for a $d + 2-$restricted FE scheme.

We now describe our construction as follows:

7.1 Construction

Ingredients: Our main ingredient is a secret-key function hiding canonical function-hiding inner product functional encryption clPE (refer Section D for a definition).

Notation: We denote by $\mathbb{F}_{\mathbf{p}}$ the field on which the computation is done in slotted encodings.

1. By boldfaced letters, we denote (multi-dimensional) matrices, where dimensions are specified. Messages are of the form $(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z})$. Here, $\boldsymbol{x} \in \mathbb{F}_{\mathbf{p}}^{d \times n}$. $\boldsymbol{y}, \boldsymbol{z} \in \mathbb{F}_{\mathbf{p}}^{n}$.
2. **Function class of interest:** We consider a set of functions $\mathcal{F}_{\mathsf{dFE}} = \mathcal{F}_{\mathsf{dFE},\lambda,\mathbf{p},n} = \{f : \mathbb{F}_{\mathbf{p}}^{n(d+2)} \to \mathbb{F}_{\mathbf{p}}\}$ where $\mathbb{F}_{\mathbf{p}}$ is a finite field of order $\mathbf{p}(\lambda)$. Here n is seen as a function of λ. Each $f \in \mathcal{F}_{\lambda,\mathbf{p},n}$ takes as input $d+2$ vectors $(\boldsymbol{x}[1], \ldots, \boldsymbol{x}[d], \boldsymbol{y}, \boldsymbol{z})$ over $\mathbb{F}_{\mathbf{p}}$ and computes a polynomial of the form $\Sigma c_{i_1,\ldots,i_d,j,k} \cdot \boldsymbol{x}[1, i_1] \cdots \boldsymbol{x}[d, i_d] \cdot \boldsymbol{y}[j] \cdot \boldsymbol{z}[k]$, where $c_{i_1,\ldots,i_d,j,k}$ are coefficients from $\mathbb{F}_{\mathbf{p}}$.

Notation. For a secret key generated for the cIPE encryption algorithm, by using primed variables such as sk' we denote the secret key that is not generated during the setup of the dFE scheme but during its encryption algorithm. We describe the construction below.

$\underline{\mathsf{Setup}(1^\lambda, 1^n)}$: On input security parameter 1^λ and length 1^n,

- Sample $\mathsf{pp} \leftarrow \mathsf{cIPE.PPSetup}(1^\lambda)$. Let us assume $\mathsf{pp} = (e, G_1, G_2, G_T, g_1, g_2, \mathbb{Z}_{\mathbf{p}})$.
- Run cIPE setup as follows. $\mathbf{sk}_0 \leftarrow \mathsf{cIPE.Setup}(\mathsf{pp}, 1^4)$. Thus these keys are used to encrypt vectors in $\mathbb{F}_{\mathbf{p}}^4$.
- Then run cIPE setup algorithm $n \cdot d$ times. That is, for every $\ell \in [d]$ and $i_\ell \in [n]$, compute $\mathbf{sk}^{(\ell,i_\ell)} \leftarrow \mathsf{cIPE.Setup}(\mathsf{pp}, 1^4)$.
- Sample $\boldsymbol{\alpha} \leftarrow \mathbb{F}_{\mathbf{p}}^{d \times n}$. Also sample $\boldsymbol{\beta}, \boldsymbol{\gamma} \leftarrow \mathbb{F}_{\mathbf{p}}^{n}$.
- For $\ell \in [d]$, $i_\ell \in [n]$ and every set $\boldsymbol{I} = (i_{\ell+1}, \ldots, i_d, j, k) \in [n]^{d-\ell+2}$ compute $\mathsf{Key}_{\boldsymbol{I}}^{(\ell,i_\ell)} =$

  $$\mathsf{cIPE.KeyGen}(\mathbf{sk}^{(\ell,i_\ell)}, (\boldsymbol{\alpha}[\ell+1, i_{\ell+1}] \cdots \boldsymbol{\alpha}[d, i_d]\boldsymbol{\beta}[j]\boldsymbol{\gamma}[k], \boldsymbol{\alpha}[\ell, i_\ell] \cdots \boldsymbol{\alpha}[d, i_d]\boldsymbol{\beta}[j]\boldsymbol{\gamma}[k], 0, 0))$$

- Output $\mathsf{MSK} = (\{\mathbf{sk}^{(\ell,i_\ell)}, \mathsf{Key}_{\boldsymbol{I}}^{(\ell,i_\ell)}\}_{\ell,i_\ell,\boldsymbol{I}}, \boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma}, \mathbf{sk}_0)$

$\underline{\mathsf{KeyGen}(\mathsf{MSK}, f)}$: On input the master secret key MSK and function f,

- Parse $\mathsf{MSK} = (\{\mathbf{sk}^{(\ell,i_\ell)}, \mathsf{Key}_{\boldsymbol{I}}^{(\ell,i_\ell)}\}_{\ell,i_\ell,\boldsymbol{I}}, \boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma}, \mathbf{sk}_0)$.
- Compute $\theta_f = f(\boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma})$.
- Compute $\mathsf{Key}_{0,f} = \mathsf{cIPE.KeyGen}(\mathbf{sk}_0, (\theta_f, 0, 0, 0))$
- Output $sk_f = (\mathsf{Key}_{0,f}, \{\mathsf{Key}_{\boldsymbol{I}}^{(\ell,i_\ell)}\}_{\ell,i_\ell,\boldsymbol{I}})$

$\underline{\mathsf{Enc}(\mathsf{MSK}, \boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z})}$: The input message $M = (\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z})$ consists of a public attribute $\boldsymbol{x} \in \mathbb{F}_{\mathbf{p}}^{d \times n}$ and private vectors $\boldsymbol{y}, \boldsymbol{z} \in \mathbb{F}_{\mathbf{p}}^{n}$. Perform the following operations:

- Parse $\mathsf{MSK} = (\{\mathbf{sk}^{(\ell,i_\ell)}, \mathsf{Key}_{\boldsymbol{I}}^{(\ell,i_\ell)}\}_{\ell,i_\ell,\boldsymbol{I}}, \boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma}, \mathbf{sk}_0)$.
- Sample $r \leftarrow \mathbb{F}_{\mathbf{p}}$.
- Compute $\mathsf{CT}_0 = \mathsf{cIPE.Enc}(\mathbf{sk}_0, (r, 0, 0, 0))$.
- Sample $\mathbf{sk}' \leftarrow \mathsf{cIPE.Setup}(\mathsf{pp}, 1^4)$.
- Compute $\mathsf{CTC}_j \leftarrow \mathsf{cIPE.Enc}(\mathbf{sk}, (\boldsymbol{y}[j], \boldsymbol{\beta}[j], 0, 0))$ for $j \in [n]$

- Compute $\mathsf{CTK}_k \leftarrow \mathsf{cIPE.Enc}(\mathbf{sk}, (\boldsymbol{z}[k], -r\boldsymbol{\gamma}[k], 0, 0))$ for $k \in [n]$.
- For every $\ell \in [d]$, $i_\ell \in [n]$, compute $\mathsf{CT}^{(\ell,i_\ell)} = \mathsf{cIPE.Enc}(\mathbf{sk}^{(\ell,i_\ell)}, (r\boldsymbol{x}[\ell, i_\ell], -r, 0, 0))$.
- Output $\mathsf{CT} = (\boldsymbol{x}, \mathsf{CT}_0, \{\mathsf{CTC}_j, \mathsf{CTK}_k, \mathsf{CT}^{(\ell,i_\ell)}\}_{\ell\in[d],i_\ell\in[n],j\in[n],k\in[n]})$

$\underline{\mathsf{Dec}(1^B, sk_f, \mathsf{CT})}$:

- Parse $\mathsf{CT} = (\boldsymbol{x}, \mathsf{CT}_0, \{\mathsf{CTC}_j, \mathsf{CTK}_k, \mathsf{CT}^{(\ell,i_\ell)}\}_{\ell\in[d],i_\ell\in[n],j\in[n],k\in[n]})$.
- Parse $sk_f = \{\mathsf{Key}_{0,f}, \mathsf{Key}_{\boldsymbol{I}}^{(\ell,i_\ell)}\}_{\ell,i_\ell,\boldsymbol{I}}$.
- For every $\ell \in [d]$ and $\boldsymbol{I} = (i_\ell, \ldots, i_d, j, k) \in [n]^{d-\ell+3}$ do the following. Let $\boldsymbol{I}'$ be such that $\boldsymbol{I} = i_\ell || \boldsymbol{I}'$. In other words, $\boldsymbol{I}'$ has all but first element of $\boldsymbol{I}$. Compute $\mathsf{Mon}_{\boldsymbol{I}'}^{(\ell,i_\ell)} = \mathsf{cIPE.Dec}(\mathsf{Key}_{\boldsymbol{I}'}^{(\ell,i_\ell)}, \mathsf{CT}^{(\ell,i_\ell)}) = [r(\boldsymbol{x}[\ell, i_\ell] - \boldsymbol{\alpha}[\ell, i_\ell])\boldsymbol{\alpha}[\ell - 1, i_{\ell-1}] \cdots \boldsymbol{\alpha}[d, i_d]\boldsymbol{\beta}[j]\boldsymbol{\gamma}[k]]_T$.
- Compute $\mathsf{Mon}_0 = \mathsf{cIPE.Dec}(\mathsf{Key}_{0,f}, \mathsf{CT}_0) = [rf(\boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma})]_T$.
- Compute $\mathsf{Mon}^{(j,k)} = \mathsf{cIPE.Dec}(\mathsf{CTK}_k, \mathsf{CTC}_j) = [\boldsymbol{y}[j]\boldsymbol{z}[k] - r\boldsymbol{\beta}[j]\boldsymbol{\gamma}[k]]_T$.
- Let $f = \Sigma_{\boldsymbol{I}=(i_1,\ldots,i_d,j,k)} c_{\boldsymbol{I}} \boldsymbol{x}[1, i_1] \cdots \boldsymbol{x}[d, i_d]\boldsymbol{y}[j]\boldsymbol{z}[k]$. Now fix $\boldsymbol{I} = (i_1, \ldots, i_d, j, k)$. For the monomial corresponding to $\boldsymbol{I}$ compute $\mathsf{Int}_{\boldsymbol{I}} = [\boldsymbol{x}[1, i_1] \cdots \boldsymbol{x}[d, i_d]\boldsymbol{y}[j]\boldsymbol{z}[k] - r\boldsymbol{\alpha}[1, i_1] \cdots \boldsymbol{\alpha}[d, i_d]\boldsymbol{\beta}[j]\boldsymbol{\gamma}[k]]_T$ as follows.
  1. For $v \in [d]$, denote $\boldsymbol{I}_v = (i_v, \ldots, i_d, j, k)$ and $\boldsymbol{I}'_v = (i_{v+1}, \ldots, i_d, j, k)$.
  2. Compute $\mathsf{Int}_{\boldsymbol{I}} = \Pi_{v\in[d]} \mathsf{Mon}^{(v,i_v)} {}_{\boldsymbol{I}'_v}{}^{\rho_{\boldsymbol{I}_v}}$. We describe $\rho_{\boldsymbol{I}_v}$ shortly.
  3. We want these coefficients $\rho_{\boldsymbol{I}_v}$ such that $\mathsf{Int}_{\boldsymbol{I}} = [\Sigma_{v\in[d]} \rho_v(\boldsymbol{x}[v, i_v]\boldsymbol{\alpha}[v + 1, i_{v+1}] \cdots \boldsymbol{\alpha}[d, i_d]\boldsymbol{\beta}[j]\boldsymbol{\gamma}[k] - r\boldsymbol{\alpha}[v, i_v] \cdots \boldsymbol{\alpha}[d, i_d]\boldsymbol{\beta}[j]\boldsymbol{\gamma}[k])]_T$.
  4. This defines $\rho_{\boldsymbol{I}_1} = 1$ and $\rho_{\boldsymbol{I}_v} = \boldsymbol{x}[1, i_1], \ldots, \boldsymbol{x}[v - 1, i_{v-1}]$ for $v \in [d]$.
  5. This can be verified for $d = 2$ as follows.

$$\begin{aligned}&\boldsymbol{x}[1, i_1]\boldsymbol{x}[2, i_2](\boldsymbol{y}[j]\boldsymbol{z}[k] - r\boldsymbol{\beta}[j]\boldsymbol{\gamma}[k]) + \boldsymbol{x}[1, i_1]r(\boldsymbol{x}[2, i_2]\\ &-\boldsymbol{\alpha}[i_2])\boldsymbol{\beta}[j]\boldsymbol{\gamma}[k] + r(\boldsymbol{x}[1, i_1] - \boldsymbol{\alpha}[1, i_1])\boldsymbol{\alpha}[i_2]\boldsymbol{\beta}[j]\boldsymbol{\gamma}[k]\\ &=\boldsymbol{x}[1, i_1]\boldsymbol{x}[2, i_2]\boldsymbol{y}[j]\boldsymbol{z}[k] - r\boldsymbol{\alpha}[1, i_1]\boldsymbol{\alpha}[2, i_2]\boldsymbol{\beta}[j]\boldsymbol{\beta}[k]\end{aligned}$$

  In this way, the process holds for any d.
- Finally compute $\mathsf{Int}_1 = \Pi_{\boldsymbol{I}=(i_1,..,i_d)} \mathsf{Int}_{\boldsymbol{I}}^{c_I} = [f(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}) - rf(\boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma})]_T$.
- Compute $\mathsf{Int}_1 \cdot \mathsf{Mon}_0 = [f(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z})]_T$. Using brute force, check if $|f(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z})| < B$. If that is the case, output $f(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z})$ otherwise output $\perp$.

We now discuss correctness and linear efficiency:

Correctness: Correctness is implicit from the description of the decryption algorithm.

Linear Efficiency: Note that ciphertext is of the following form:

$\mathsf{CT} = (\boldsymbol{x}, \mathsf{CT}_0, \{\mathsf{CTC}_j, \mathsf{CTK}_k, \mathsf{CT}^{(\ell,i_\ell)}\}_{\ell\in[d],i_\ell\in[n],j\in[n],k\in[n]})$. Thus there are $n \times (d + 1) + 1$ cIPE ciphertexts and n cIPE function keys for vectors of length 4. Hence, the claim holds due to the efficiency of cIPE.

7.2 Security Proof

Here is our theorem.

Theorem 3. *Assuming SXDH holds relative to* PPGen*, the construction described in Section 7 satisfies semi-functional security.*

First we describe the semi-functional encryption and key generation algorithm.
$\underline{\mathsf{sfKG}(\mathsf{MSK}, f, \Delta)}$: On input the master secret key MSK, function f and a value Δ, perform the following steps. The change from regular key generation algorithm is marked with boldfaced [**Change**].

- Parse $\mathsf{MSK} = (\{\mathbf{sk}^{(\ell,i_\ell)}, \mathsf{Key}_{\boldsymbol{I}}^{(\ell,i_\ell)}\}_{\ell,i_\ell,\boldsymbol{I}}, \boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma}, \mathbf{sk}_0)$.
- Compute $\theta_f = f(\boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma})$.
- [**Change**] Compute $\mathsf{Key}_{0,f} = \mathsf{cIPE.KeyGen}(\mathbf{sk}_0, (\theta_f, \Delta, 0, 0))$
- Output $sk_f = (\mathsf{Key}_{0,f}, \{\mathbf{sk}^{(\ell,i_\ell)}, \mathsf{Key}_{\boldsymbol{I}}^{(\ell,i_\ell)}\}_{\ell,i_\ell,\boldsymbol{I}})$

Note that the decryption of a functional ciphertext using a semi-functional key is identical as Δ is in in a slot that does not interfere with the computation.

We now describe semi-functional encryption algorithm:
$\underline{\mathsf{sfEnc}(\mathsf{MSK}, \boldsymbol{x}, 1^n)}$: On input the public attribute $\boldsymbol{x} \in \mathbb{F}_{\mathbf{p}}^{d \times n}$ and MSK. Perform the following operations. The change from regular encryption algorithm is marked with boldfaced [**Change**].

- Parse $\mathsf{MSK} = (\{\mathbf{sk}^{(\ell,i_\ell)}, \mathsf{Key}_{\boldsymbol{I}}^{(\ell,i_\ell)}\}_{\ell,i_\ell,\boldsymbol{I}}, \boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma}, \mathbf{sk}_0)$.
- Sample $r \leftarrow \mathbb{F}_{\mathbf{p}}$.
- [**Change**] Compute $\mathsf{CT}_0 = \mathsf{cIPE.Enc}(\mathbf{sk}_0, (r, 1, 0, 0))$.
- Sample $\mathbf{sk}' \leftarrow \mathsf{cIPE.Setup}(\mathsf{pp}, 1^4)$. Note a 1 placed in the second slot.
- [**Change**] Compute $\mathsf{CTC}_j \leftarrow \mathsf{cIPE.Enc}(\mathbf{sk}', (0, \boldsymbol{\beta}[j], 0, 0))$ for $j \in [n]$. Note that 0 is encrypted instead of $\boldsymbol{y}[j]$.
- [**Change**] Compute $\mathsf{CTK}_k \leftarrow \mathsf{cIPE.Enc}(\mathbf{sk}', (0, -r\boldsymbol{\gamma}[k], 0, 0))$ for $k \in [n]$. Note that 0 is encrypted instead of $\boldsymbol{z}[k]$.
- For every $\ell \in [d]$, $i_\ell \in [n]$, compute $\mathsf{CT}^{(\ell,i_\ell)} = \mathsf{cIPE.Enc}(\mathbf{sk}^{(\ell,i_\ell)}, (r\boldsymbol{x}[\ell, i_\ell], -r, 0, 0))$.
- Output $\mathsf{CT} = (\boldsymbol{x}, \mathsf{CT}_0, \{\mathsf{CTC}_j, \mathsf{CTK}_k, \mathsf{CT}^{(\ell,i_\ell)}\}_{\ell\in[d],i_\ell\in[n],j\in[n],k\in[n]})$

Note that the decryption of semi-functional ciphertext with a semi-functional key with value Δ will return $f(\boldsymbol{x}, \mathbf{0}, \mathbf{0}) + \Delta = \Delta$ as the output of the decryption.

Indistinguishability of Semi-Functional Key: This is straight-forward to show

Lemma 1. *Assuming* cIPE *is a canonical function hiding inner product FE scheme, the scheme described in Section 7, satisfies indistinguishability of semi-functional keys property.*

Proof. The only difference between the distribution of keys and ciphertexts corresponding to challenge bit 0 and challenge bit 1 in the security game of

semi-functional key security is the following. If challenge bit is 0, the function keys are functionally generated while if challenge bit is 1 the function keys are semi-functionally generated. A function key for a function f is of the form $sk_f = (\mathsf{Key}_{0,f}, \{\mathbf{sk}^{(\ell,i_\ell)}, \mathsf{Key}_{\boldsymbol{I}}^{(\ell,i_\ell)}\}_{\ell,i_\ell,\boldsymbol{I}})$. The only difference lies in the component $\mathsf{Key}_{0,f}$. If challenge bit is 0, it is generated as $\mathsf{cIPE.KeyGen}(\mathbf{sk}_0, (\theta_f, 0, 0, 0))$ otherwise as $\mathsf{cIPE.KeyGen}(\mathbf{sk}_0, (\theta_f, \Delta, 0, 0))$. Note in both the cases ciphertexts generated by sk^0 always has 0 in the second slot, so the inner products in both the cases remain the same. The proof now holds because of the function hiding security of cIPE.

Theorem 4. *Assuming SXDH holds relative to* PPGen*, the scheme described in Section 7, satisfies indistinguishability of semi-functional ciphertexts property.*

Proof. We now list hybrids and prove indistinguishability between them. In the first hybrid the challenge ciphertext is generated honestly and the keys are semi-functionally generated. In the last hybrid, the challenge ciphertext is generated semi-functionally and the keys are semi-functionally generated.

$\underline{\mathsf{Setup}(1^\lambda, 1^n)}$: On input security parameter λ,

$\mathbf{Hybrid}_0$:

1. The challenger does setup for the scheme as follows:
   - Sample $\mathsf{pp} \leftarrow \mathsf{cIPE.PPSetup}(1^\lambda)$. Let us assume $\mathsf{pp} = (e, G_1, G_2, G_T, g_1, g_2, \mathbb{Z}_{\mathbf{p}})$.
   - Run cIPE setup as follows. $\mathbf{sk}_0 \leftarrow \mathsf{cIPE.Setup}(\mathsf{pp}, 1^4)$. Thus these keys are used to encrypt vectors in $\mathbb{F}_{\mathbf{p}}^4$.
   - Then run cIPE setup algorithm $n \cdot d$ times. That is, for every $\ell \in [d]$ and $i_\ell \in [n]$, compute $\mathbf{sk}^{(\ell,i_\ell)} \leftarrow \mathsf{cIPE.Setup}(\mathsf{pp}, 1^4)$.
   - Sample $\boldsymbol{\alpha} \leftarrow \mathbb{F}_{\mathbf{p}}^{d\times n}$. Also sample $\boldsymbol{\beta}, \boldsymbol{\gamma} \leftarrow \mathbb{F}_{\mathbf{p}}^n$.
   - For $\ell \in [d]$, $i_\ell \in [n]$ and every set $\boldsymbol{I} = (i_{\ell+1}, ..., i_d, j, k) \in [n]^{d-\ell+2}$ compute
     $\mathsf{Key}_{\boldsymbol{I}}^{(\ell,i_\ell)} = \mathsf{cIPE.KeyGen}(\mathbf{sk}^{(\ell,i_\ell)}, (\boldsymbol{\alpha}[\ell+1, i_{\ell+1}] \cdots \boldsymbol{\alpha}[d, i_d]\boldsymbol{\beta}[j]\boldsymbol{\gamma}[k],$
     $\boldsymbol{\alpha}[\ell, i_\ell] \cdots \boldsymbol{\alpha}[d, i_d]\boldsymbol{\beta}[j]\boldsymbol{\gamma}[k], 0, 0))$
   - Output $\mathsf{MSK} = (\{\mathbf{sk}^{(\ell,i_\ell)}, \mathsf{Key}_{\boldsymbol{I}}^{(\ell,i_\ell)}\}_{\ell,i_\ell,\boldsymbol{I}}, \boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma}, \mathbf{sk}_0)$
2. Then adversary releases challenge messages $m_i = (\boldsymbol{x}_i, \boldsymbol{y}_i, \boldsymbol{z}_i)$ for $i \in [L]$. It also gives out a challenge message $m = (\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z})$ in $\mathbb{Z}_{\mathbf{p}}^{d\times n}$. We now describe how the challenge ciphertext CT is generated. Other ciphertexts $\mathsf{CT}[i]$ for $i \in [L]$ are generated similarly.
   - Parse $\mathsf{MSK} = (\{\mathbf{sk}^{(\ell,i_\ell)}, \mathsf{Key}_{\boldsymbol{I}}^{(\ell,i_\ell)}\}_{\ell,i_\ell,\boldsymbol{I}}, \boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma}, \mathbf{sk}_0)$.
   - Sample $r \leftarrow \mathbb{F}_{\mathbf{p}}$.
   - Compute $\mathsf{CT}_0 = \mathsf{cIPE.Enc}(\mathbf{sk}_0, (r, 0, 0, 0))$.
   - Sample $\mathbf{sk}' \leftarrow \mathsf{cIPE.Setup}(\mathsf{pp}, 1^4)$. Note a 1 placed in the second slot.
   - Compute $\mathsf{CTC}_j \leftarrow \mathsf{cIPE.Enc}(\mathbf{sk}', (\boldsymbol{y}[j], \boldsymbol{\beta}[j], 0, 0))$ for $j \in [n]$.
   - Compute $\mathsf{CTK}_k \leftarrow \mathsf{cIPE.Enc}(\mathbf{sk}', (\boldsymbol{z}[k], -r\boldsymbol{\gamma}[k], 0, 0))$ for $k \in [n]$.
   - For every $\ell \in [d]$, $i_\ell \in [n]$, compute $\mathsf{CT}^{(\ell,i_\ell)} = \mathsf{cIPE.Enc}(\mathbf{sk}^{(\ell,i_\ell)}, (r\boldsymbol{x}[\ell, i_\ell], -r, 0, 0))$.
   - Output $\mathsf{CT} = (\boldsymbol{x}, \mathsf{CT}_0, \{\mathsf{CTC}_j, \mathsf{CTK}_k, \mathsf{CT}^{(\ell,i_\ell)}\}_{\ell\in[d],i_\ell\in[n],j\in[n],k\in[n]})$

3. Adversary also (selectively) asks for keys functions f_i for $i \in [L]$. Let us say $\Delta_i = f_i(m)$ for $i \in [L]$. Then adversary is given semi-functional keys generated as follows. Below we denote the procedure for a single function f.
   - Parse $\mathsf{MSK} = (\{\mathbf{sk}^{(\ell,i_\ell)}, \mathsf{Key}_{\boldsymbol{I}}^{(\ell,i_\ell)}\}_{\ell,i_\ell,\boldsymbol{I}}, \boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma}, \mathbf{sk}_0)$.
   - Compute $\theta_f = f(\boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma})$.
   - Compute $\mathsf{Key}_{0,f} = \mathsf{cIPE.KeyGen}(\mathbf{sk}_0, (\theta_f, \Delta, 0, 0))$
   - Output $sk_f = (\mathsf{Key}_{0,f}, \{\mathbf{sk}^{(\ell,i_\ell)}, \mathsf{Key}_{\boldsymbol{I}}^{(\ell,i_\ell)}\}_{\ell,i_\ell,\boldsymbol{I}})$
4. Adversary then outputs a bit $b' \in \{0,1\}$

$\mathbf{Hybrid}_1$:

1. The challenger does setup for the scheme as follows:
   - Sample $\mathsf{pp} \leftarrow \mathsf{cIPE.PPSetup}(1^\lambda)$. Let us assume $\mathsf{pp} = (e, G_1, G_2, G_T, g_1, g_2, \mathbb{Z}_{\mathbf{p}})$.
   - Run cIPE setup as follows. $\mathbf{sk}_0 \leftarrow \mathsf{cIPE.Setup}(\mathsf{pp}, 1^4)$. Thus these keys are used to encrypt vectors in $\mathbb{F}_{\mathbf{p}}^4$.
   - Then run cIPE setup algorithm $n \cdot d$ times. That is, for every $\ell \in [d]$ and $i_\ell \in [n]$, compute $\mathbf{sk}^{(\ell,i_\ell)} \leftarrow \mathsf{cIPE.Setup}(\mathsf{pp}, 1^4)$.
   - Sample $\boldsymbol{\alpha} \leftarrow \mathbb{F}_{\mathbf{p}}^{d\times n}$. Also sample $\boldsymbol{\beta}, \boldsymbol{\gamma} \leftarrow \mathbb{F}_{\mathbf{p}}^n$.
   - For $\ell \in [d]$, $i_\ell \in [n]$ and every set $\boldsymbol{I} = (i_{\ell+1}, ..., i_d, j, k) \in [n]^{d-\ell+2}$ compute
     $\mathsf{Key}_{\boldsymbol{I}}^{(\ell,i_\ell)} = \mathsf{cIPE.KeyGen}(\mathbf{sk}^{(\ell,i_\ell)}, (\boldsymbol{\alpha}[\ell+1, i_{\ell+1}] \cdots \boldsymbol{\alpha}[d, i_d]\boldsymbol{\beta}[j]\boldsymbol{\gamma}[k], \boldsymbol{\alpha}[\ell, i_\ell] \cdots \boldsymbol{\alpha}[d, i_d]\boldsymbol{\beta}[j]\boldsymbol{\gamma}[k], 0, 0))$
   - Output $\mathsf{MSK} = (\{\mathbf{sk}^{(\ell,i_\ell)}, \mathsf{Key}_{\boldsymbol{I}}^{(\ell,i_\ell)}\}_{\ell,i_\ell,\boldsymbol{I}}, \boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma}, \mathbf{sk}_0)$
2. Then adversary releases challenge messages $m_i = (\boldsymbol{x}_i, \boldsymbol{y}_i, \boldsymbol{z}_i)$ for $i \in [L]$. It also gives out a challenge message $m = (\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z})$ in $\mathbb{Z}_{\mathbf{p}}^{d\times n}$. We now describe how the challenge ciphertext CT is generated. Other ciphertexts $\mathsf{CT}[i]$ for $i \in [L]$ are generated similarly.
   - Parse $\mathsf{MSK} = (\{\mathbf{sk}^{(\ell,i_\ell)}, \mathsf{Key}_{\boldsymbol{I}}^{(\ell,i_\ell)}\}_{\ell,i_\ell,\boldsymbol{I}}, \boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma}, \mathbf{sk}_0)$.
   - Sample $r \leftarrow \mathbb{F}_{\mathbf{p}}$.
   - [**Change**] Compute $\mathsf{CT}_0 = \mathsf{cIPE.Enc}(\mathbf{sk}_0, (0, 0, 1, 0))$.
   - Sample $\mathbf{sk}' \leftarrow \mathsf{cIPE.Setup}(\mathsf{pp}, 1^4)$. Note a 1 placed in the second slot.
   - Compute $\mathsf{CTC}_j \leftarrow \mathsf{cIPE.Enc}(\mathbf{sk}', (\boldsymbol{y}[j], \boldsymbol{\beta}[j], 0, 0))$ for $j \in [n]$.
   - Compute $\mathsf{CTK}_k \leftarrow \mathsf{cIPE.Enc}(\mathbf{sk}', (\boldsymbol{z}[k], -r\boldsymbol{\gamma}[k], 0, 0))$ for $k \in [n]$.
   - For every $\ell \in [d]$, $i_\ell \in [n]$, compute $\mathsf{CT}^{(\ell,i_\ell)} = \mathsf{cIPE.Enc}(\mathbf{sk}^{(\ell,i_\ell)}, (r\boldsymbol{x}[\ell, i_\ell], -r, 0, 0))$.
   - Output $\mathsf{CT} = (\boldsymbol{x}, \mathsf{CT}_0, \{\mathsf{CTC}_j, \mathsf{CTK}_k, \mathsf{CT}^{(\ell,i_\ell)}\}_{\ell\in[d],i_\ell\in[n],j\in[n],k\in[n]})$
3. Adversary also (selectively) asks for keys functions f_i for $i \in [L]$. Let us say $\Delta_i = f_i(m)$ for $i \in [L]$. Then adversary is given semi-functional keys generated as follows. Below we denote the procedure for a single function f and corresponding value Δ.
   - Parse $\mathsf{MSK} = (\{\mathbf{sk}^{(\ell,i_\ell)}, \mathsf{Key}_{\boldsymbol{I}}^{(\ell,i_\ell)}\}_{\ell,i_\ell,\boldsymbol{I}}, \boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma}, \mathbf{sk}_0)$.
   - Compute $\theta_f = f(\boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma})$.
   - [**Change**] Compute $\mathsf{Key}_{0,f} = \mathsf{cIPE.KeyGen}(\mathbf{sk}_0, (\theta_f, \Delta, r\theta_f, 0))$
   - Output $sk_f = (\mathsf{Key}_{0,f}, \{\mathbf{sk}^{(\ell,i_\ell)}, \mathsf{Key}_{\boldsymbol{I}}^{(\ell,i_\ell)}\}_{\ell,i_\ell,\boldsymbol{I}})$
4. Adversary then outputs a bit $b' \in \{0, 1\}$

Lemma 2. *Due to security of* cIPE*, for any polymomial time distinguisher* $\mathcal{A}$*,* $|\Pr[\mathcal{A}(\mathbf{Hybrid}_0) = 1] - \Pr[\mathcal{A}(\mathbf{Hybrid}_1)]| < 2^{-\lambda^\epsilon}$ *for some constant* $\epsilon > 0$

Proof. The only difference between $\mathbf{Hybrid}_0$ and $\mathbf{Hybrid}_1$ is the way how challenge ciphertext component CT_0 and function key components $\mathsf{Key}_{0,f}$ for all queried functions f are generated. In $\mathbf{Hybrid}_0$,

- CT_0 is generated as an encryption of $(r, 0, 0, 0)$

– $\mathsf{Key}_{0,f}$ is generated as a key for the vector $(\theta_f, \Delta, 0, 0)$

In $\mathbf{Hybrid}_1$,

– CT_0 is generated as an encryption of $(0, 0, 1, 0)$
– $\mathsf{Key}_{0,f}$ is generated as a key for the vector $(\theta_f, \Delta, r\theta_f, 0)$

Thus observe that the decryption of all $\mathsf{CT}[\mathsf{ind}]_0$ for $\mathsf{ind} \in [L]$ and CT_0 with keys $\mathsf{Key}_{0,f}$ stay the same and thus, by the security of cIPE encryption scheme, the proof follows.

Next hybrid is the following: $\mathbf{Hybrid}_2$:

1. This hybrid is the same as the previous hybrid except that the challenge ciphertext, CT, and keys $\mathsf{Key}_{\boldsymbol{I}}^{(\ell,i_\ell)}$ for every $\ell \in [d], i_\ell \in [n]$ and $\boldsymbol{I} = (i_{\ell+1}, ..., i_d, j, k) \in [n]^{d+2-\ell}$ are generated differently.
   - Sample $\mathsf{pp} \leftarrow \mathsf{cIPE.PPSetup}(1^\lambda)$. Let us assume $\mathsf{pp} = (e, G_1, G_2, G_T, g_1, g_2, \mathbb{Z}_{\mathbf{p}})$.
   - Run cIPE setup as follows. $\mathbf{sk}_0 \leftarrow \mathsf{cIPE.Setup}(\mathsf{pp}, 1^4)$. Thus these keys are used to encrypt vectors in $\mathbb{F}_{\mathbf{p}}^4$.
   - Then run cIPE setup algorithm $n \cdot d$ times. That is, for every $\ell \in [d]$ and $i_\ell \in [n]$, compute $\mathbf{sk}^{(\ell,i_\ell)} \leftarrow \mathsf{cIPE.Setup}(\mathsf{pp}, 1^4)$.
   - Sample $\boldsymbol{\alpha} \leftarrow \mathbb{F}_{\mathbf{p}}^{d \times n}$. Also sample $\boldsymbol{\beta}, \boldsymbol{\gamma} \leftarrow \mathbb{F}_{\mathbf{p}}^n$.
   - [**Change**] Sample $r \leftarrow \mathbb{F}_{\mathbf{p}}$ (randomness used for the ciphertext).
   - [**Change**] For $\ell \in [d]$, $i_\ell \in [n]$ and every set $\boldsymbol{I} = (i_{\ell+1}, ..., i_d, j, k) \in [n]^{d-\ell+2}$ compute $\mathsf{Key}_{\boldsymbol{I}}^{(\ell,i_\ell)} = \mathsf{cIPE.KeyGen}(\mathbf{sk}^{(\ell,i_\ell)}, (\boldsymbol{\alpha}[\ell+1, i_{\ell+1}] \cdots \boldsymbol{\alpha}[d, i_d]\boldsymbol{\beta}[j]\boldsymbol{\gamma}[k], \boldsymbol{\alpha}[\ell, i_\ell] \cdots \boldsymbol{\alpha}[d, i_d]\boldsymbol{\beta}[j]\boldsymbol{\gamma}[k], (\boldsymbol{x}[\ell, i_\ell] - \boldsymbol{\alpha}[\ell, i_\ell])r\boldsymbol{\alpha}[\ell+1, i_{\ell+1}] \cdots \boldsymbol{\alpha}[d, i_d]\boldsymbol{\beta}[j]\boldsymbol{\gamma}[k], 0))$
   - Output $\mathsf{MSK} = (\{\mathbf{sk}^{(\ell,i_\ell)}, \mathsf{Key}_{\boldsymbol{I}}^{(\ell,i_\ell)}\}_{\ell,i_\ell,\boldsymbol{I}}, \boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma}, \mathbf{sk}_0)$
2. Then adversary releases challenge messages $m_i = (\boldsymbol{x}_i, \boldsymbol{y}_i, \boldsymbol{z}_i)$ for $i \in [L]$. It also gives out a challenge message $m = (\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z})$ in $\mathbb{Z}_{\mathbf{p}}^{d \times n}$. We now describe how the challenge ciphertext CT is generated. Other ciphertexts $\mathsf{CT}[i]$ for $i \in [L]$ are generated similarly as in previous hybrids.
   - Parse $\mathsf{MSK} = (\{\mathbf{sk}^{(\ell,i_\ell)}, \mathsf{Key}_{\boldsymbol{I}}^{(\ell,i_\ell)}\}_{\ell,i_\ell,\boldsymbol{I}}, \boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma}, \mathbf{sk}_0)$.
   - Compute $\mathsf{CT}_0 = \mathsf{cIPE.Enc}(\mathbf{sk}_0, (0, 0, 1, 0))$.
   - Sample $\mathbf{sk}' \leftarrow \mathsf{cIPE.Setup}(\mathsf{pp}, 1^4)$. Note a 1 placed in the second slot.
   - Compute $\mathsf{CTC}_j \leftarrow \mathsf{cIPE.Enc}(\mathbf{sk}', (\boldsymbol{y}[j], \boldsymbol{\beta}[j], 0, 0))$ for $j \in [n]$.
   - Compute $\mathsf{CTK}_k \leftarrow \mathsf{cIPE.Enc}(\mathbf{sk}', (\boldsymbol{z}[k], -r\boldsymbol{\gamma}[k], 0, 0))$ for $k \in [n]$.
   - [**Change**] For every $\ell \in [d]$, $i_\ell \in [n]$, compute $\mathsf{CT}^{(\ell,i_\ell)} = \mathsf{cIPE.Enc}(\mathbf{sk}^{(\ell,i_\ell)}, (0, 0, 1, 0))$.
   - Output $\mathsf{CT} = (\boldsymbol{x}, \mathsf{CT}_0, \{\mathsf{CTC}_j, \mathsf{CTK}_k, \mathsf{CT}^{(\ell,i_\ell)}\}_{\ell \in [d], i_\ell \in [n], j \in [n], k \in [n]})$
3. Adversary also (selectively) asks for keys functions f_i for $i \in [L]$. Let us say $\Delta_i = f_i(m)$ for $i \in [L]$. Then adversary is given semi-functional keys generated as follows. Below we denote the procedure for a single function f and corresponding value Δ.
   - Parse $\mathsf{MSK} = (\{\mathbf{sk}^{(\ell,i_\ell)}, \mathsf{Key}_{\boldsymbol{I}}^{(\ell,i_\ell)}\}_{\ell,i_\ell,\boldsymbol{I}}, \boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma}, \mathbf{sk}_0)$.
   - Compute $\theta_f = f(\boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma})$.
   - Compute $\mathsf{Key}_{0,f} = \mathsf{cIPE.KeyGen}(\mathbf{sk}_0, (\theta_f, \Delta, r\theta_f, 0))$
   - Output $sk_f = (\mathsf{Key}_{0,f}, \{\mathbf{sk}^{(\ell,i_\ell)}, \mathsf{Key}_{\boldsymbol{I}}^{(\ell,i_\ell)}\}_{\ell,i_\ell,\boldsymbol{I}})$
4. Adversary then outputs a bit $b' \in \{0, 1\}$

Lemma 3. *Due to security of* cIPE*, for any polymomial time distinguisher* $\mathcal{A}$*,* $|\Pr[\mathcal{A}(\mathbf{Hybrid}_1) = 1] - \Pr[\mathcal{A}(\mathbf{Hybrid}_2)]| < 2^{-\lambda^\epsilon}$ *for some constant* $\epsilon > 0$

Proof. The only difference between $\mathbf{Hybrid}_1$ and $\mathbf{Hybrid}_2$ is the way how function key component $\mathsf{Key}_{\boldsymbol{I}}^{(\ell,i_\ell)}$ and challenge ciphertext component $\mathsf{CT}^{(\ell,i_\ell)}$ are generated for $\ell \in [d]$, $i_\ell \in [n]$ and $\boldsymbol{I} = (i_{\ell+1}, ..., i_d, j, k) \in [n]^{d+2-\ell}$.

In $\mathbf{Hybrid}_1$,

- $\mathsf{CT}^{(\ell,i_\ell)}$ is generated as an encryption of $(r\boldsymbol{x}[\ell,i_\ell],-r,0,0)$
- $\mathsf{Key}_{\boldsymbol{I}}^{(\ell,i_\ell)}$ is generated as a key for the vector $(\boldsymbol{\alpha}[\ell+1,i_{\ell+1}]\cdots\boldsymbol{\alpha}[d,i_d]\boldsymbol{\beta}[j]\boldsymbol{\gamma}[k]$ $,\boldsymbol{\alpha}[\ell+1,i_{\ell+1}]\cdots\boldsymbol{\alpha}[d,i_d]\boldsymbol{\beta}[j]\boldsymbol{\gamma}[k],0,0)$

In $\mathbf{Hybrid}_2$,

- $\mathsf{CT}^{(\ell,i_\ell)}$ is generated as an encryption of $(0,0,1,0)$
- $\mathsf{Key}_{\boldsymbol{I}}^{(\ell,i_\ell)}$ is generated as a key for the vector $(\boldsymbol{\alpha}[\ell+1,i_{\ell+1}]\cdots\boldsymbol{\alpha}[d,i_d]\boldsymbol{\beta}[j]\boldsymbol{\gamma}[k]$ $,\boldsymbol{\alpha}[\ell+1,i_{\ell+1}]\cdots\boldsymbol{\alpha}[d,i_d]\boldsymbol{\beta}[j]\boldsymbol{\gamma}[k],r(\boldsymbol{x}[\ell,i_\ell]-\boldsymbol{\alpha}[\boldsymbol{\ell},\boldsymbol{i_\ell}])\boldsymbol{\alpha}[\ell+1,i_{\ell+1}]\cdots\boldsymbol{\alpha}[d,i_d]\boldsymbol{\beta}[j]\boldsymbol{\gamma}[k]$ $,0)$

Thus observe that the decryption of all $\mathsf{CT}[\mathsf{ind}]^{(\ell,i_\ell)}$ and $\mathsf{CT}^{(\ell,i_\ell)}$ with keys $\mathsf{Key}_{\boldsymbol{I}}^{(\ell,i_\ell)}$ for $\mathsf{ind}\in[L]$ stay the same and thus, by function hiding security of clPE encryption scheme, the proof follows.

Now we describe a sequence of hybrids and prove indistinguishability between them. Denote $\mathbf{Hybrid}_{3,0} = \mathbf{Hybrid}_2$ and then define $\mathbf{Hybrid}_{3,t\in[n]}$:

1. This hybrid is the same as the previous hybrid except that the challenge ciphertext, CT, and key components $\mathsf{Key}_{0,f}$ are generated differently.
   - Sample $\mathsf{pp} \leftarrow \mathsf{cIPE.PPSetup}(1^\lambda)$. Let us assume $\mathsf{pp} = (e, G_1, G_2, G_T, g_1, g_2, \mathbb{Z}_{\mathbf{p}})$.
   - Run cIPE setup as follows. $\mathbf{sk}_0 \leftarrow \mathsf{cIPE.Setup}(\mathsf{pp}, 1^4)$. Thus these keys are used to encrypt vectors in $\mathbb{F}^4_{\mathbf{p}}$.
   - Then run cIPE setup algorithm $n \cdot d$ times. That is, for every $\ell \in [d]$ and $i_\ell \in [n]$, compute $\mathbf{sk}^{(\ell,i_\ell)} \leftarrow \mathsf{cIPE.Setup}(\mathsf{pp}, 1^4)$.
   - Sample $\boldsymbol{\alpha} \leftarrow \mathbb{F}^{d\times n}_{\mathbf{p}}$. Also sample $\boldsymbol{\beta}, \boldsymbol{\gamma} \leftarrow \mathbb{F}^n_{\mathbf{p}}$.
   - Sample $r \leftarrow \mathbb{F}_{\mathbf{p}}$ (randomness used for the ciphertext).
   - For $\ell \in [d]$, $i_\ell \in [n]$ and every set $\boldsymbol{I} = (i_{\ell+1}, ..., i_d, j, k) \in [n]^{d-\ell+2}$ compute $\mathsf{Key}^{(\ell,i_\ell)}_{\boldsymbol{I}} = \mathsf{cIPE.KeyGen}(\mathbf{sk}^{(\ell,i_\ell)}, (\boldsymbol{\alpha}[\ell+1, i_{\ell+1}] \cdots \boldsymbol{\alpha}[d, i_d]\boldsymbol{\beta}[j]\boldsymbol{\gamma}[k], \boldsymbol{\alpha}[\ell, i_\ell] \cdots \boldsymbol{\alpha}[d, i_d]\boldsymbol{\beta}[j]\boldsymbol{\gamma}[k], (\boldsymbol{x}[\ell, i_\ell] - \boldsymbol{\alpha}[\ell, i_\ell])r\boldsymbol{\alpha}[\ell+1, i_{\ell+1}] \cdots \boldsymbol{\alpha}[d, i_d]\boldsymbol{\beta}[j]\boldsymbol{\gamma}[k], 0))$
   - Output $\mathsf{MSK} = (\{\mathbf{sk}^{(\ell,i_\ell)}, \mathsf{Key}^{(\ell,i_\ell)}_{\boldsymbol{I}}\}_{\ell,i_\ell,\boldsymbol{I}}, \boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma}, \mathbf{sk}_0)$
2. Then adversary releases challenge messages $m_i = (\boldsymbol{x}_i, \boldsymbol{y}_i, \boldsymbol{z}_i)$ for $i \in [L]$. It also gives out a challenge message $m = (\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z})$ in $\mathbb{Z}^{d\times n}_{\mathbf{p}}$. We now describe how the challenge ciphertext CT is generated. Other ciphertexts $\mathsf{CT}[i]$ for $i \in [L]$ are generated similarly as in previous hybrids.
   - Parse $\mathsf{MSK} = (\{\mathbf{sk}^{(\ell,i_\ell)}, \mathsf{Key}^{(\ell,i_\ell)}_{\boldsymbol{I}}\}_{\ell,i_\ell,\boldsymbol{I}}, \boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma}, \mathbf{sk}_0)$.
   - Compute $\mathsf{CT}_0 = \mathsf{cIPE.Enc}(\mathbf{sk}_0, (0, 0, 1, 0))$.
   - Sample $\mathbf{sk}' \leftarrow \mathsf{cIPE.Setup}(\mathsf{pp}, 1^4)$. Note a 1 placed in the second slot.
   - [**Change**] Compute $\mathsf{CTC}_j \leftarrow \mathsf{cIPE.Enc}(\mathbf{sk}', (0, \boldsymbol{\beta}[j], 0, 0))$ for $j \in [t]$ and $\mathsf{CTC}_j \leftarrow \mathsf{cIPE.Enc}(\mathbf{sk}', (\boldsymbol{y}[j], \boldsymbol{\beta}[j], 0, 0))$ otherwise.
   - Compute $\mathsf{CTK}_k \leftarrow \mathsf{cIPE.Enc}(\mathbf{sk}', (0, -r\boldsymbol{\gamma}[k], 0, 0))$ for $k \in [n]$.
   - For every $\ell \in [d]$, $i_\ell \in [n]$, compute $\mathsf{CT}^{(\ell,i_\ell)} = \mathsf{cIPE.Enc}(\mathbf{sk}^{(\ell,i_\ell)}, (0, 0, 1, 0))$.
   - Output $\mathsf{CT} = (\boldsymbol{x}, \mathsf{CT}_0, \{\mathsf{CTC}_j, \mathsf{CTK}_k, \mathsf{CT}^{(\ell,i_\ell)}\}_{\ell\in[d],i_\ell\in[n],j\in[n],k\in[n]})$
3. Adversary also (selectively) asks for keys functions f_i for $i \in [L]$. Let us say $\Delta_i = f_i(m)$ for $i \in [L]$. Then adversary is given semi-functional keys generated as follows. Below we denote the procedure for a single function f and corresponding value Δ.
   - Parse $\mathsf{MSK} = (\{\mathbf{sk}^{(\ell,i_\ell)}, \mathsf{Key}^{(\ell,i_\ell)}_{\boldsymbol{I}}\}_{\ell,i_\ell,\boldsymbol{I}}, \boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma}, \mathbf{sk}_0)$.
   - Compute $\theta_f = f(\boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma})$.
   - [**Change**] Compute $\mathsf{Key}_{0,f} = \mathsf{cIPE.KeyGen}(\mathbf{sk}_0, (\theta_f, \Delta, r\theta_f + \Delta', 0))$ where $\Delta' = f(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}) - f(\boldsymbol{x}, \boldsymbol{yt}, \boldsymbol{z})$ where $\boldsymbol{yt} = (0, .., 0, y_{t+1}, ..., y_n)$.
   - Output $sk_f = (\mathsf{Key}_{0,f}, \{\mathbf{sk}^{(\ell,i_\ell)}, \mathsf{Key}^{(\ell,i_\ell)}_{\boldsymbol{I}}\}_{\ell,i_\ell,\boldsymbol{I}})$
4. Adversary then outputs a bit $b' \in \{0, 1\}$

Since indistinguishability of these hybrids require further sub-hybrids, we describe it later in Section 7.3.

Hybrid$_4$:

1. This hybrid is the same as the previous hybrid except that the challenge ciphertext, CT, and key components $\mathsf{Key}_{0,f}$ are generated differently.
   - Sample $\mathsf{pp} \leftarrow \mathsf{cIPE.PPSetup}(1^\lambda)$. Let us assume $\mathsf{pp} = (e, G_1, G_2, G_T, g_1, g_2, \mathbb{Z}_{\mathbf{p}})$.
   - Run cIPE setup as follows. $\mathbf{sk}_0 \leftarrow \mathsf{cIPE.Setup}(\mathsf{pp}, 1^4)$. Thus these keys are used to encrypt vectors in $\mathbb{F}_{\mathbf{p}}^4$.
   - Then run cIPE setup algorithm $n \cdot d$ times. That is, for every $\ell \in [d]$ and $i_\ell \in [n]$, compute $\mathbf{sk}^{(\ell,i_\ell)} \leftarrow \mathsf{cIPE.Setup}(\mathsf{pp}, 1^4)$.
   - Sample $\boldsymbol{\alpha} \leftarrow \mathbb{F}_{\mathbf{p}}^{d\times n}$. Also sample $\boldsymbol{\beta}, \boldsymbol{\gamma} \leftarrow \mathbb{F}_{\mathbf{p}}^n$.
   - Sample $r \leftarrow \mathbb{F}_{\mathbf{p}}$ (randomness used for the ciphertext).
   - For $\ell \in [d]$, $i_\ell \in [n]$ and every set $\boldsymbol{I} = (i_{\ell+1}, ..., i_d, j, k) \in [n]^{d-\ell+2}$ compute $\mathsf{Key}_{\boldsymbol{I}}^{(\ell,i_\ell)} = \mathsf{cIPE.KeyGen}(\mathbf{sk}^{(\ell,i_\ell)}, (\boldsymbol{\alpha}[\ell+1, i_{\ell+1}] \cdots \boldsymbol{\alpha}[d, i_d]\boldsymbol{\beta}[j]\boldsymbol{\gamma}[k], \boldsymbol{\alpha}[\ell, i_\ell] \cdots \boldsymbol{\alpha}[d, i_d]\boldsymbol{\beta}[j]\boldsymbol{\gamma}[k], (\boldsymbol{x}[\ell, i_\ell] - \boldsymbol{\alpha}[\ell, i_\ell])r\boldsymbol{\alpha}[\ell+1, i_{\ell+1}] \cdots \boldsymbol{\alpha}[d, i_d]\boldsymbol{\beta}[j]\boldsymbol{\gamma}[k], 0))$
   - Output $\mathsf{MSK} = (\{\mathbf{sk}^{(\ell,i_\ell)}, \mathsf{Key}_{\boldsymbol{I}}^{(\ell,i_\ell)}\}_{\ell,i_\ell,\boldsymbol{I}}, \boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma}, \mathbf{sk}_0)$
2. Then adversary releases challenge messages $m_i = (\boldsymbol{x}_i, \boldsymbol{y}_i, \boldsymbol{z}_i)$ for $i \in [L]$. It also gives out a challenge message $m = (\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z})$ in $\mathbb{Z}_{\mathbf{p}}^{d\times n}$. We now describe how the challenge ciphertext CT is generated. Other ciphertexts $\mathsf{CT}[i]$ for $i \in [L]$ are generated similarly as in previous hybrids.
   - Parse $\mathsf{MSK} = (\{\mathbf{sk}^{(\ell,i_\ell)}, \mathsf{Key}_{\boldsymbol{I}}^{(\ell,i_\ell)}\}_{\ell,i_\ell,\boldsymbol{I}}, \boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma}, \mathbf{sk}_0)$.
   - Compute $\mathsf{CT}_0 = \mathsf{cIPE.Enc}(\mathbf{sk}_0, (0, 0, 1, 0))$.
   - Sample $\mathbf{sk}' \leftarrow \mathsf{cIPE.Setup}(\mathsf{pp}, 1^4)$. Note a 1 placed in the second slot.
   - Compute $\mathsf{CTC}_j \leftarrow \mathsf{cIPE.Enc}(\mathbf{sk}', (0, \boldsymbol{\beta}[j], 0, 0))$ for $j \in [n]$.
   - [**Change**] Compute $\mathsf{CTK}_k \leftarrow \mathsf{cIPE.Enc}(\mathbf{sk}', (0, -r\boldsymbol{\gamma}[k], 0, 0))$ for $k \in [n]$.
   - For every $\ell \in [d]$, $i_\ell \in [n]$, compute $\mathsf{CT}^{(\ell,i_\ell)} = \mathsf{cIPE.Enc}(\mathbf{sk}^{(\ell,i_\ell)}, (0, 0, 1, 0))$.
   - Output $\mathsf{CT} = (\boldsymbol{x}, \mathsf{CT}_0, \{\mathsf{CTC}_j, \mathsf{CTK}_k, \mathsf{CT}^{(\ell,i_\ell)}\}_{\ell\in[d],i_\ell\in[n],j\in[n],k\in[n]})$
3. Adversary also (selectively) asks for keys functions f_i for $i \in [L]$. Let us say $\Delta_i = f_i(m)$ for $i \in [L]$. Then adversary is given semi-functional keys generated as follows. Below we denote the procedure for a single function f and corresponding value Δ.
   - Parse $\mathsf{MSK} = (\{\mathbf{sk}^{(\ell,i_\ell)}, \mathsf{Key}_{\boldsymbol{I}}^{(\ell,i_\ell)}\}_{\ell,i_\ell,\boldsymbol{I}}, \boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma}, \mathbf{sk}_0)$.
   - Compute $\theta_f = f(\boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma})$.
   - [**Change**] Compute $\mathsf{Key}_{0,f} = \mathsf{cIPE.KeyGen}(\mathbf{sk}_0, (\theta_f, \Delta, r\theta_f + \Delta', 0))$ where $\Delta = \Delta' = f(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z})$
   - Output $sk_f = (\mathsf{Key}_{0,f}, \{\mathbf{sk}^{(\ell,i_\ell)}, \mathsf{Key}_{\boldsymbol{I}}^{(\ell,i_\ell)}\}_{\ell,i_\ell,\boldsymbol{I}})$
4. Adversary then outputs a bit $b' \in \{0, 1\}$

Lemma 4. *Due to security of* cIPE*, for any polymomial time distinguisher* $\mathcal{A}$*,* $|\Pr[\mathcal{A}(\mathbf{Hybrid}_{3,n}) = 1] - \Pr[\mathcal{A}(\mathbf{Hybrid}_4)]| < 2^{-\lambda^\epsilon}$ *for some constant* $\epsilon > 0$

Proof. The only difference between $\mathbf{Hybrid}_{3,n}$ and $\mathbf{Hybrid}_4$ is the way how challenge ciphertext components CTK_k for $k \in [n]$ are generated.

In $\mathbf{Hybrid}_{3,n}$,

- CTK_k is generated as a cIPE key for the vector $(\boldsymbol{z}[k], -r\boldsymbol{\gamma}[k], 0, 0)$.

In $\mathbf{Hybrid}_4$,

– CTK_k is generated as a cIPE key for the vector $(0, -r\boldsymbol{\gamma}[k], 0, 0)$.

Thus observe that the decryption of all CTC_j with keys CTK_k for $j, k \in [n]$ is the same in both hybrids and thus, by the security of cIPE encryption scheme, the proof follows.

Hybrid$_5$:

1. This hybrid is the same as the previous hybrid except that the challenge ciphertext, CT and the cIPE keys $\mathsf{Key}_{\boldsymbol{I}}^{(\ell,i_\ell)}$ are generated differently.
   - Sample $\mathsf{pp} \leftarrow \mathsf{cIPE.PPSetup}(1^\lambda)$. Let us assume $\mathsf{pp} = (e, G_1, G_2, G_T, g_1, g_2, \mathbb{Z}_{\mathbf{p}})$.
   - Run cIPE setup as follows. $\mathbf{sk}_0 \leftarrow \mathsf{cIPE.Setup}(\mathsf{pp}, 1^4)$. Thus these keys are used to encrypt vectors in $\mathbb{F}_{\mathbf{p}}^4$.
   - Then run cIPE setup algorithm $n \cdot d$ times. That is, for every $\ell \in [d]$ and $i_\ell \in [n]$, compute $\mathbf{sk}^{(\ell,i_\ell)} \leftarrow \mathsf{cIPE.Setup}(\mathsf{pp}, 1^4)$.
   - Sample $\boldsymbol{\alpha} \leftarrow \mathbb{F}_{\mathbf{p}}^{d \times n}$. Also sample $\boldsymbol{\beta}, \boldsymbol{\gamma} \leftarrow \mathbb{F}_{\mathbf{p}}^n$.
   - Sample $r \leftarrow \mathbb{F}_{\mathbf{p}}$ (randomness used for the ciphertext).
   - [**Change**] For $\ell \in [d]$, $i_\ell \in [n]$ and every set $\boldsymbol{I} = (i_{\ell+1}, ..., i_d, j, k) \in [n]^{d-\ell+2}$ compute $\mathsf{Key}_{\boldsymbol{I}}^{(\ell,i_\ell)} = \mathsf{cIPE.KeyGen}(\mathbf{sk}^{(\ell,i_\ell)}, (\boldsymbol{\alpha}[\ell+1, i_{\ell+1}] \cdots \boldsymbol{\alpha}[d, i_d]\boldsymbol{\beta}[j]\boldsymbol{\gamma}[k], \boldsymbol{\alpha}[\ell, i_\ell] \cdots \boldsymbol{\alpha}[d, i_d]\boldsymbol{\beta}[j]\boldsymbol{\gamma}[k], 0, 0))$
   - Output $\mathsf{MSK} = (\{\mathbf{sk}^{(\ell,i_\ell)}, \mathsf{Key}_{\boldsymbol{I}}^{(\ell,i_\ell)}\}_{\ell,i_\ell,\boldsymbol{I}}, \boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma}, \mathbf{sk}_0)$
2. Then adversary releases challenge messages $m_i = (\boldsymbol{x}_i, \boldsymbol{y}_i, \boldsymbol{z}_i)$ for $i \in [L]$. It also gives out a challenge message $m = (\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z})$ in $\mathbb{Z}_{\mathbf{p}}^{d \times n}$. We now describe how the challenge ciphertext CT is generated. Other ciphertexts $\mathsf{CT}[i]$ for $i \in [L]$ are generated similarly as in previous hybrids.
   - Parse $\mathsf{MSK} = (\{\mathbf{sk}^{(\ell,i_\ell)}, \mathsf{Key}_{\boldsymbol{I}}^{(\ell,i_\ell)}\}_{\ell,i_\ell,\boldsymbol{I}}, \boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma}, \mathbf{sk}_0)$.
   - Compute $\mathsf{CT}_0 = \mathsf{cIPE.Enc}(\mathbf{sk}_0, (0, 0, 1, 0))$.
   - Sample $\mathbf{sk}' \leftarrow \mathsf{cIPE.Setup}(\mathsf{pp}, 1^4)$. Note a 1 placed in the second slot.
   - Compute $\mathsf{CTC}_j \leftarrow \mathsf{cIPE.Enc}(\mathbf{sk}', (0, \boldsymbol{\beta}[j], 0, 0))$ for $j \in [n]$.
   - Compute $\mathsf{CTK}_k \leftarrow \mathsf{cIPE.Enc}(\mathbf{sk}', (0, -r\boldsymbol{\gamma}[k], 0, 0))$ for $k \in [n]$.
   - [**Change**] For every $\ell \in [d]$, $i_\ell \in [n]$, compute $\mathsf{CT}^{(\ell,i_\ell)} = \mathsf{cIPE.Enc}(\mathbf{sk}^{(\ell,i_\ell)}, (r\boldsymbol{x}[\ell, i_\ell], -r, 0, 0))$.
   - Output $\mathsf{CT} = (\boldsymbol{x}, \mathsf{CT}_0, \{\mathsf{CTC}_j, \mathsf{CTK}_k, \mathsf{CT}^{(\ell,i_\ell)}\}_{\ell \in [d], i_\ell \in [n], j \in [n], k \in [n]})$
3. Adversary also (selectively) asks for keys functions f_i for $i \in [L]$. Let us say $\Delta_i = f_i(m)$ for $i \in [L]$. Then adversary is given semi-functional keys generated as follows. Below we denote the procedure for a single function f and corresponding value Δ.
   - Parse $\mathsf{MSK} = (\{\mathbf{sk}^{(\ell,i_\ell)}, \mathsf{Key}_{\boldsymbol{I}}^{(\ell,i_\ell)}\}_{\ell,i_\ell,\boldsymbol{I}}, \boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma}, \mathbf{sk}_0)$.
   - Compute $\theta_f = f(\boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma})$.
   - Compute $\mathsf{Key}_{0,f} = \mathsf{cIPE.KeyGen}(\mathbf{sk}_0, (\theta_f, \Delta, r\theta_f + \Delta', 0))$ where $\Delta = \Delta' = f(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z})$
   - Output $sk_f = (\mathsf{Key}_{0,f}, \{\mathbf{sk}^{(\ell,i_\ell)}, \mathsf{Key}_{\boldsymbol{I}}^{(\ell,i_\ell)}\}_{\ell,i_\ell,\boldsymbol{I}})$
4. Adversary then outputs a bit $b' \in \{0, 1\}$

Lemma 5. *Due to security of* cIPE*, for any polymomial time distinguisher* $\mathcal{A}$*,* $|\Pr[\mathcal{A}(\mathbf{Hybrid}_4) = 1] - \Pr[\mathcal{A}(\mathbf{Hybrid}_5)]| < 2^{-\lambda^\epsilon}$ *for some constant* $\epsilon > 0$

Proof. The only difference between $\mathbf{Hybrid}_4$ and $\mathbf{Hybrid}_5$ is the way how challenge ciphertext components $\mathsf{CT}^{\ell,i_\ell}$ for $\ell \in [d], i_\ell \in [n]$ and function key components $\mathsf{Key}_{\boldsymbol{I}}^{(\ell,i_\ell)}$ are generated.

In $\mathbf{Hybrid}_4$,

- $\mathsf{CT}^{(\ell,i_\ell)}$ is generated for the vector $(0,0,1,0)$.
- $\mathsf{Key}_{\boldsymbol{I}}^{(\ell,i_\ell)}$ is generated for vector $(\boldsymbol{\alpha}[\ell+1,i_{\ell+1}]\cdots\boldsymbol{\alpha}[d,i_d]\boldsymbol{\beta}[j]\boldsymbol{\gamma}[k], \boldsymbol{\alpha}[\ell+1,i_{\ell+1}]\cdots\boldsymbol{\alpha}[d,i_d]\boldsymbol{\beta}[j]\boldsymbol{\gamma}[k], r(\boldsymbol{x}[\ell,i_\ell]-\boldsymbol{\alpha}[\ell,i_\ell])\boldsymbol{\alpha}[\ell+1,i_{\ell+1}]\cdots\boldsymbol{\alpha}[d,i_d]\boldsymbol{\beta}[j]\boldsymbol{\gamma}[k], 0)$.

In $\mathbf{Hybrid}_5$,

- $\mathsf{CT}^{(\ell,i_\ell)}$ is generated for the vector $(r\boldsymbol{x}[\ell,i_\ell], -r, 0, 0)$.
- $\mathsf{Key}_{\boldsymbol{I}}^{(\ell,i_\ell)}$ is generated for vector $(\boldsymbol{\alpha}[\ell+1,i_{\ell+1}]\cdots\boldsymbol{\alpha}[d,i_d]\boldsymbol{\beta}[j]\boldsymbol{\gamma}[k], \boldsymbol{\alpha}[\ell+1,i_{\ell+1}]\cdots\boldsymbol{\alpha}[d,i_d]\boldsymbol{\beta}[j]\boldsymbol{\gamma}[k], 0, 0)$.

Thus observe that the decryption of all $\mathsf{CT}^{(\ell,i_\ell)}$ (as well as $\mathsf{CT}[\mathsf{ind}]^{(\ell,i_\ell)}$) with keys $\mathsf{Key}_{\boldsymbol{I}}^{(\ell,i_\ell)}$ for $\mathsf{ind} \in [L]$ is the same in both hybrids and thus, by the security of clPE encryption scheme, the proof follows.

Hybrid$_6$:

1. This hybrid is the same as the previous hybrid except that the challenge ciphertext component, CT_0 and the cIPE keys $\mathsf{Key}_{0,f}$ are generated differently.
   - Sample $\mathsf{pp} \leftarrow \mathsf{cIPE.PPSetup}(1^\lambda)$. Let us assume $\mathsf{pp} = (e, G_1, G_2, G_T, g_1, g_2, \mathbb{Z}_{\mathbf{p}})$.
   - Run cIPE setup as follows. $\mathbf{sk}_0 \leftarrow \mathsf{cIPE.Setup}(\mathsf{pp}, 1^4)$. Thus these keys are used to encrypt vectors in $\mathbb{F}_{\mathbf{p}}^4$.
   - Then run cIPE setup algorithm $n \cdot d$ times. That is, for every $\ell \in [d]$ and $i_\ell \in [n]$, compute $\mathbf{sk}^{(\ell,i_\ell)} \leftarrow \mathsf{cIPE.Setup}(\mathsf{pp}, 1^4)$.
   - Sample $\boldsymbol{\alpha} \leftarrow \mathbb{F}_{\mathbf{p}}^{d \times n}$. Also sample $\boldsymbol{\beta}, \boldsymbol{\gamma} \leftarrow \mathbb{F}_{\mathbf{p}}^n$.
   - Sample $r \leftarrow \mathbb{F}_{\mathbf{p}}$ (randomness used for the ciphertext).
   - For $\ell \in [d]$, $i_\ell \in [n]$ and every set $\boldsymbol{I} = (i_{\ell+1}, ..., i_d, j, k) \in [n]^{d-\ell+2}$ compute
     $\mathsf{Key}_{\boldsymbol{I}}^{(\ell,i_\ell)} = \mathsf{cIPE.KeyGen}(\mathbf{sk}^{(\ell,i_\ell)}, (\boldsymbol{\alpha}[\ell+1, i_{\ell+1}] \cdots \boldsymbol{\alpha}[d, i_d]\boldsymbol{\beta}[j]\boldsymbol{\gamma}[k],$
     $\boldsymbol{\alpha}[\ell, i_\ell] \cdots \boldsymbol{\alpha}[d, i_d]\boldsymbol{\beta}[j]\boldsymbol{\gamma}[k], 0, 0))$
   - Output $\mathsf{MSK} = (\{\mathbf{sk}^{(\ell,i_\ell)}, \mathsf{Key}_{\boldsymbol{I}}^{(\ell,i_\ell)}\}_{\ell,i_\ell,\boldsymbol{I}}, \boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma}, \mathbf{sk}_0)$
2. Then adversary releases challenge messages $m_i = (\boldsymbol{x}_i, \boldsymbol{y}_i, \boldsymbol{z}_i)$ for $i \in [L]$. It also gives out a challenge message $m = (\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z})$ in $\mathbb{Z}_{\mathbf{p}}^{d \times n}$. We now describe how the challenge ciphertext CT is generated. Other ciphertexts $\mathsf{CT}[i]$ for $i \in [L]$ are generated similarly as in previous hybrids.
   - Parse $\mathsf{MSK} = (\{\mathbf{sk}^{(\ell,i_\ell)}, \mathsf{Key}_{\boldsymbol{I}}^{(\ell,i_\ell)}\}_{\ell,i_\ell,\boldsymbol{I}}, \boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma}, \mathbf{sk}_0)$.
   - [**Change**] Compute $\mathsf{CT}_0 = \mathsf{cIPE.Enc}(\mathbf{sk}_0, (r, 1, 0, 0))$.
   - Sample $\mathbf{sk}' \leftarrow \mathsf{cIPE.Setup}(\mathsf{pp}, 1^4)$. Note a 1 placed in the second slot.
   - Compute $\mathsf{CTC}_j \leftarrow \mathsf{cIPE.Enc}(\mathbf{sk}', (0, \boldsymbol{\beta}[j], 0, 0))$ for $j \in [n]$.
   - Compute $\mathsf{CTK}_k \leftarrow \mathsf{cIPE.Enc}(\mathbf{sk}', (0, -r\boldsymbol{\gamma}[k], 0, 0))$ for $k \in [n]$.
   - For every $\ell \in [d]$, $i_\ell \in [n]$, compute $\mathsf{CT}^{(\ell,i_\ell)} = \mathsf{cIPE.Enc}(\mathbf{sk}^{(\ell,i_\ell)}, (r\boldsymbol{x}[\ell, i_\ell],$ $-r, 0, 0))$.
   - Output $\mathsf{CT} = (\boldsymbol{x}, \mathsf{CT}_0, \{\mathsf{CTC}_j, \mathsf{CTK}_k, \mathsf{CT}^{(\ell,i_\ell)}\}_{\ell \in [d], i_\ell \in [n], j \in [n], k \in [n]})$
3. Adversary also (selectively) asks for keys functions f_i for $i \in [L]$. Let us say $\Delta_i = f_i(m)$ for $i \in [L]$. Then adversary is given semi-functional keys generated as follows. Below we denote the procedure for a single function f and corresponding value Δ.
   - Parse $\mathsf{MSK} = (\{\mathbf{sk}^{(\ell,i_\ell)}, \mathsf{Key}_{\boldsymbol{I}}^{(\ell,i_\ell)}\}_{\ell,i_\ell,\boldsymbol{I}}, \boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma}, \mathbf{sk}_0)$.
   - Compute $\theta_f = f(\boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma})$.
   - [**Change**] Compute $\mathsf{Key}_{0,f} = \mathsf{cIPE.KeyGen}(\mathbf{sk}_0, (\theta_f, \Delta, 0, 0))$ where $\Delta = \Delta' = f(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z})$
   - Output $sk_f = (\mathsf{Key}_{0,f}, \{\mathbf{sk}^{(\ell,i_\ell)}, \mathsf{Key}_{\boldsymbol{I}}^{(\ell,i_\ell)}\}_{\ell,i_\ell,\boldsymbol{I}})$
4. Adversary then outputs a bit $b' \in \{0, 1\}$

Lemma 6. *Due to security of* cIPE*, for any polymomial time distinguisher* $\mathcal{A}$*,* $|\Pr[\mathcal{A}(\mathbf{Hybrid}_5) = 1] - \Pr[\mathcal{A}(\mathbf{Hybrid}_6)]| < 2^{-\lambda^\epsilon}$ *for some constant* $\epsilon > 0$

Proof. The only difference between $\mathbf{Hybrid}_5$ and $\mathbf{Hybrid}_6$ is the way how challenge ciphertext components CT_0 and function key components $\mathsf{Key}_{0,f}$ for $i \in [n]$ and functions f are generated.

In $\mathbf{Hybrid}_5$,

- CT_0 is generated for the vector $(0, 0, 1, 0)$.
- $\mathsf{Key}_{0,f}$ is generated for $(\theta_f, \Delta, r\theta_f + \Delta, 0)$.

In $\mathbf{Hybrid}_6$,

- CT_0 is generated for the vector $(r, 1, 0, 0)$.
- $\mathsf{Key}_{0,f}$ is generated for $(\theta_f, \Delta, 0, 0)$.

Thus observe that the decryption of all $\mathsf{CT}[\mathsf{ind}]_0$ (as well as CT_0) with keys $\mathsf{Key}_{0,f}$ for functions f and $\mathsf{ind} \in [L]$ is the same in both hybrids and thus, by the security of cIPE encryption scheme, the proof follows.

The last hybrid exactly corresponds to the security game when both the challenge ciphertexts as well as the function keys are generated using the semi-functional algorithms and thus the claim holds.

7.3 Indistinguishability of $\mathbf{Hybrid}_{3,t}$ and $\mathbf{Hybrid}_{3,t+1}$

Let us now recall $\mathbf{Hybrid}_{3,t}$. Note that $\mathbf{Hybrid}_2 = \mathbf{Hybrid}_{3,0}$.

1. This hybrid is the same as the previous hybrid except that the challenge ciphertext, CT, and key components $\mathsf{Key}_{0,f}$ are generated differently.
   - Sample $\mathsf{pp} \leftarrow \mathsf{cIPE.PPSetup}(1^\lambda)$. Let us assume $\mathsf{pp} = (e, G_1, G_2, G_T, g_1, g_2, \mathbb{Z}_{\mathbf{p}})$.
   - Run cIPE setup as follows. $\mathbf{sk}_0 \leftarrow \mathsf{cIPE.Setup}(\mathsf{pp}, 1^4)$. Thus these keys are used to encrypt vectors in $\mathbb{F}_{\mathbf{p}}^4$.
   - Then run cIPE setup algorithm $n \cdot d$ times. That is, for every $\ell \in [d]$ and $i_\ell \in [n]$, compute $\mathbf{sk}^{(\ell, i_\ell)} \leftarrow \mathsf{cIPE.Setup}(\mathsf{pp}, 1^4)$.
   - Sample $\boldsymbol{\alpha} \leftarrow \mathbb{F}_{\mathbf{p}}^{d \times n}$. Also sample $\boldsymbol{\beta}, \boldsymbol{\gamma} \leftarrow \mathbb{F}_{\mathbf{p}}^n$.
   - Sample $r \leftarrow \mathbb{F}_{\mathbf{p}}$ (randomness used for the ciphertext).
   - For $\ell \in [d]$, $i_\ell \in [n]$ and every set $\boldsymbol{I} = (i_{\ell+1}, ..., i_d, j, k) \in [n]^{d-\ell+2}$ compute $\mathsf{Key}_{\boldsymbol{I}}^{(\ell, i_\ell)} =$

$$\mathsf{cIPE.KeyGen}(\mathbf{sk}^{(\ell, i_\ell)}, (\boldsymbol{\alpha}[\ell+1, i_{\ell+1}] \cdots \boldsymbol{\alpha}[d, i_d]\boldsymbol{\beta}[j]\boldsymbol{\gamma}[k],$$

$$\boldsymbol{\alpha}[\ell, i_\ell] \cdots \boldsymbol{\alpha}[d, i_d]\boldsymbol{\beta}[j]\boldsymbol{\gamma}[k],$$

$$(\boldsymbol{x}[\ell, i_\ell] - \boldsymbol{\alpha}[\ell, i_\ell])r\boldsymbol{\alpha}[\ell+1, i_{\ell+1}] \cdots \boldsymbol{\alpha}[d, i_d]\boldsymbol{\beta}[j]\boldsymbol{\gamma}[k], 0))$$

   - Output $\mathsf{MSK} = (\{\mathbf{sk}^{(\ell, i_\ell)}, \mathsf{Key}_{\boldsymbol{I}}^{(\ell, i_\ell)}\}_{\ell, i_\ell, \boldsymbol{I}}, \boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma}, \mathbf{sk}_0)$
2. Then adversary releases challenge messages $m_i = (\boldsymbol{x}_i, \boldsymbol{y}_i, \boldsymbol{z}_i)$ for $i \in [L]$. It also gives out a challenge message $m = (\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z})$ in $\mathbb{Z}_{\mathbf{p}}^{d \times n}$. We now describe how the challenge ciphertext CT is generated. Other ciphertexts $\mathsf{CT}[i]$ for $i \in [L]$ are generated similarly as in previous hybrids.
   - Parse $\mathsf{MSK} = (\{\mathbf{sk}^{(\ell, i_\ell)}, \mathsf{Key}_{\boldsymbol{I}}^{(\ell, i_\ell)}\}_{\ell, i_\ell, \boldsymbol{I}}, \boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma}, \mathbf{sk}_0)$.
   - Compute $\mathsf{CT}_0 = \mathsf{cIPE.Enc}(\mathbf{sk}_0, (0, 0, 1, 0))$.
   - Sample $\mathbf{sk}' \leftarrow \mathsf{cIPE.Setup}(\mathsf{pp}, 1^4)$. Note a 1 placed in the second slot.
   - [**Change**] Compute $\mathsf{CTC}_j \leftarrow \mathsf{cIPE.Enc}(\mathbf{sk}', (0, \boldsymbol{\beta}[j], 0, 0))$ for $j \in [t]$ and $\mathsf{CTC}_j \leftarrow \mathsf{cIPE.Enc}(\mathbf{sk}', (\boldsymbol{y}[j], \boldsymbol{\beta}[j], 0, 0))$ otherwise.

- Compute $\mathsf{CTK}_k \leftarrow \mathsf{cIPE.Enc}(\mathbf{sk}', (0, -r\boldsymbol{\gamma}[k], 0, 0))$ for $k \in [n]$.
- For every $\ell \in [d], i_\ell \in [n]$, compute $\mathsf{CT}^{(\ell,i_\ell)} = \mathsf{cIPE.Enc}(\mathbf{sk}^{(\ell,i_\ell)}, (0, 0, 1, 0))$.
- Output $\mathsf{CT} = (\boldsymbol{x}, \mathsf{CT}_0, \{\mathsf{CTC}_j, \mathsf{CTK}_k, \mathsf{CT}^{(\ell,i_\ell)}\}_{\ell\in[d],i_\ell\in[n],j\in[n],k\in[n]})$

3. Adversary also (selectively) asks for keys functions f_i for $i \in [L]$. Let us say $\Delta_i = f_i(m)$ for $i \in [L]$. Then adversary is given semi-functional keys generated as follows. Below we denote the procedure for a single function f and corresponding value Δ.
   - Parse $\mathsf{MSK} = (\{\mathbf{sk}^{(\ell,i_\ell)}, \mathsf{Key}_{\boldsymbol{I}}^{(\ell,i_\ell)}\}_{\ell,i_\ell,\boldsymbol{I}}, \boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma}, \mathbf{sk}_0)$.
   - Compute $\theta_f = f(\boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma})$.
   - [**Change**] Compute $\mathsf{Key}_{0,f} = \mathsf{cIPE.KeyGen}(\mathbf{sk}_0, (\theta_f, \Delta, r\theta_f + \Delta', 0))$ where $\Delta' = f(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}) - f(\boldsymbol{x}, \boldsymbol{yt}, \boldsymbol{z})$ where $\boldsymbol{yt} = (0, .., 0, y_{t+1}, ..., y_n)$.
   - Output $sk_f = (\mathsf{Key}_{0,f}, \{\mathbf{sk}^{(\ell,i_\ell)}, \mathsf{Key}_{\boldsymbol{I}}^{(\ell,i_\ell)}\}_{\ell,i_\ell,\boldsymbol{I}})$
4. Adversary then outputs a bit $b' \in \{0, 1\}$

Lemma 7. *Assuming subexponenital SXDH, for any p.p.t. distinguisher* $\mathcal{A}$, $|\Pr[\mathcal{A}(\mathbf{Hybrid}_{3,t})] - \Pr[\mathcal{A}(\mathbf{Hybrid}_{3,t+1})]| < 2^{-\lambda^c}$ *for some constant* $c > 0$ *and* $t \in [0, .., n-1]$.

Proof Structure: We prove indistinguishability between $\mathbf{Hybrid}_{3,0}$ and $\mathbf{Hybrid}_{3,1}$ and then claim that arguments generalise for indistinguishability of the general case. Namely, the following is the sequence of hybrids (we describe them formally later):

1. $\mathbf{Hybrid}_{3,0,1}$: This hybrid is the same as $\mathbf{Hybrid}_{3,0}$ except that challenge ciphertexts and other ciphertexts are generated differently. Namely CTC_1 and CTK_k for $k \in [n]$ are generated differently. CTC_1 is generated as an encryption of vector $(0, 0, 1, 0)$. CTK_k is generated as a cIPE key for the vector $(\boldsymbol{z}[k], -r\boldsymbol{\gamma}[k], \boldsymbol{y}[1]\boldsymbol{z}[k] - r\boldsymbol{\beta}[1]\boldsymbol{\gamma}[k], 0)$. The same changes are performed in other queried ciphertexts. Note that this can be done due to the security of cIPE. Due to canonical structure of cIPE, now it ensures that r, $\boldsymbol{\beta}[1]$ and $\boldsymbol{\gamma}$ are encoded in group 2. Note that $\boldsymbol{\alpha}$ is also encoded in group 2.
2. $\mathbf{Hybrid}_{3,0,2}$: This hybrid is same as the previous hybrid except that now we replace $r\boldsymbol{\alpha}[\ell, i_\ell] \cdots \boldsymbol{\alpha}[d, i_d]\boldsymbol{\beta}[1]\boldsymbol{\gamma}[k]$ for every $\ell \in [d]$ and $i_\ell, ..., i_d \in [n]$ and $k \in [n]$ with a truly random $w_{i_\ell,...,i_d,1,k} \leftarrow \mathbb{F}_{\mathbf{p}}$. Note that this can be done using SXDH in steps. First we set $r\boldsymbol{\beta}[1]\boldsymbol{\gamma}[k]$ as $w_{1,k}$ and then $r\boldsymbol{\alpha}[d, i_d]\boldsymbol{\beta}[1]\boldsymbol{\gamma}[k]$ as $\boldsymbol{\alpha}[d, i_d]w_{1,k}$ and then finally replace it by $w_{i_d,1,k}$. This process generalizes and we can replace all such monomials with independent random elements.
3. $\mathbf{Hybrid}_{3,0,3}$ This hybrid is the same as the previous one now we replace $\boldsymbol{y}[1] = 0$ and simultaneously introduce an offset. This is specified shortly.
4. $\mathbf{Hybrid}_{3,0,4}$ This hybrid is the same as the previous one except now we switch back $w_{i_\ell,..,i_d,1,k}$ with $r\boldsymbol{\alpha}[\ell, i_\ell] \cdots \boldsymbol{\alpha}[d, i_d]\boldsymbol{\beta}[1]\boldsymbol{\gamma}[k]$. This hybrid is same as $\mathbf{Hybrid}_{3,1}$

$\mathbf{Hybrid}_{3,0,1}$

1. This hybrid is the same as the previous hybrid except that the challenge ciphertext component, CTC_1, and CTK_k for every $k \in [n]$ (and the same components of other ciphertexts $\mathsf{CT}[\mathsf{ind}]$ for $\mathsf{ind} \in [L]$) are generated differently.
   - Sample $\mathsf{pp} \leftarrow \mathsf{cIPE.PPSetup}(1^\lambda)$. Let us assume $\mathsf{pp} = (e, G_1, G_2, G_T, g_1, g_2, \mathbb{Z}_{\mathbf{p}})$.
   - Run cIPE setup as follows. $\mathbf{sk}_0 \leftarrow \mathsf{cIPE.Setup}(\mathsf{pp}, 1^4)$. Thus these keys are used to encrypt vectors in $\mathbb{F}_{\mathbf{p}}^4$.
   - Then run cIPE setup algorithm $n \cdot d$ times. That is, for every $\ell \in [d]$ and $i_\ell \in [n]$, compute $\mathbf{sk}^{(\ell,i_\ell)} \leftarrow \mathsf{cIPE.Setup}(\mathsf{pp}, 1^4)$.
   - Sample $\boldsymbol{\alpha} \leftarrow \mathbb{F}_{\mathbf{p}}^{d \times n}$. Also sample $\boldsymbol{\beta}, \boldsymbol{\gamma} \leftarrow \mathbb{F}_{\mathbf{p}}^n$.
   - Sample $r \leftarrow \mathbb{F}_{\mathbf{p}}$ (randomness used for the ciphertext).
   - For $\ell \in [d]$, $i_\ell \in [n]$ and every set $\boldsymbol{I} = (i_{\ell+1}, ..., i_d, j, k) \in [n]^{d-\ell+2}$ compute $\mathsf{Key}_{\boldsymbol{I}}^{(\ell,i_\ell)} =$

     $$\mathsf{cIPE.KeyGen}(\mathbf{sk}^{(\ell,i_\ell)}, (\boldsymbol{\alpha}[\ell+1, i_{\ell+1}] \cdots \boldsymbol{\alpha}[d, i_d]\boldsymbol{\beta}[j]\boldsymbol{\gamma}[k], \boldsymbol{\alpha}[\ell, i_\ell] \cdots \boldsymbol{\alpha}[d, i_d]\boldsymbol{\beta}[j]\boldsymbol{\gamma}[k],$$

     $$(\boldsymbol{x}[\ell, i_\ell] - \boldsymbol{\alpha}[\ell, i_\ell])r\boldsymbol{\alpha}[\ell+1, i_{\ell+1}] \cdots \boldsymbol{\alpha}[d, i_d]\boldsymbol{\beta}[j]\boldsymbol{\gamma}[k], 0))$$

   - Output $\mathsf{MSK} = (\{\mathbf{sk}^{(\ell,i_\ell)}, \mathsf{Key}_{\boldsymbol{I}}^{(\ell,i_\ell)}\}_{\ell,i_\ell,\boldsymbol{I}}, \boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma}, \mathbf{sk}_0)$
2. Then adversary releases challenge messages $m_i = (\boldsymbol{x}_i, \boldsymbol{y}_i, \boldsymbol{z}_i)$ for $i \in [L]$. It also gives out a challenge message $m = (\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z})$ in $\mathbb{Z}_{\mathbf{p}}^{d \times n}$. We now describe how the challenge ciphertext CT is generated. Other ciphertexts $\mathsf{CT}[i]$ for $i \in [L]$ are generated similarly as in previous hybrids.
   - Parse $\mathsf{MSK} = (\{\mathbf{sk}^{(\ell,i_\ell)}, \mathsf{Key}_{\boldsymbol{I}}^{(\ell,i_\ell)}\}_{\ell,i_\ell,\boldsymbol{I}}, \boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma}, \mathbf{sk}_0)$.
   - Compute $\mathsf{CT}_0 = \mathsf{cIPE.Enc}(\mathbf{sk}_0, (0, 0, 1, 0))$.
   - Sample $\mathbf{sk}' \leftarrow \mathsf{cIPE.Setup}(\mathsf{pp}, 1^4)$. Note a 1 placed in the second slot.
   - [**Change**] Compute $\mathsf{CTC}_1 = \mathsf{cIPE.Enc}(\mathbf{sk}', (0, 0, 1, 0))$ and compute $\mathsf{CTC}_j \leftarrow \mathsf{cIPE.Enc}(\mathbf{sk}', (\boldsymbol{y}[j], \boldsymbol{\beta}[j], 0, 0))$ for $j \in [2, .., n]$.
   - [**Change**] Compute $\mathsf{CTK}_k \leftarrow \mathsf{cIPE.Enc}(\mathbf{sk}', (\boldsymbol{z}[k], -r\boldsymbol{\gamma}[k], \boldsymbol{y}[1]\boldsymbol{z}[k] - r\boldsymbol{\beta}[1]\boldsymbol{\gamma}[k], 0))$ for $k \in [n]$.
   - For every $\ell \in [d]$, $i_\ell \in [n]$, compute $\mathsf{CT}^{(\ell,i_\ell)} = \mathsf{cIPE.Enc}(\mathbf{sk}^{(\ell,i_\ell)}, (0, 0, 1, 0))$.
   - Output $\mathsf{CT} = (\boldsymbol{x}, \mathsf{CT}_0, \{\mathsf{CTC}_j, \mathsf{CTK}_k, \mathsf{CT}^{(\ell,i_\ell)}\}_{\ell \in [d], i_\ell \in [n], j \in [n], k \in [n]})$

   Other ciphertexts $\mathsf{CT}[\mathsf{ind}]$ are generated as follows.
   - Parse $\mathsf{MSK} = (\{\mathbf{sk}^{(\ell,i_\ell)}, \mathsf{Key}_{\boldsymbol{I}}^{(\ell,i_\ell)}\}_{\ell,i_\ell,\boldsymbol{I}}, \boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma}, \mathbf{sk}_0)$.
   - Sample $r[\mathsf{ind}]$ from $\mathbb{F}_{\mathbf{p}}$
   - Compute $\mathsf{CT}[\mathsf{ind}]_0 = \mathsf{cIPE.Enc}(\mathbf{sk}_0, (r[\mathsf{ind}], 0, 0, 0))$.
   - Sample $\mathbf{sk}'[\mathsf{ind}] \leftarrow \mathsf{cIPE.Setup}(\mathsf{pp}, 1^4)$. Note a 1 placed in the second slot.
   - [**Change**] Compute $\mathsf{CTC}[\mathsf{ind}]_1 = \mathsf{cIPE.Enc}(\mathbf{sk}'[\mathsf{ind}], (0, 0, 1, 0))$ and compute $\mathsf{CTC}[\mathsf{ind}]_j \leftarrow \mathsf{cIPE.Enc}(\mathbf{sk}'[\mathsf{ind}], (\boldsymbol{y}[\mathsf{ind}][j], \boldsymbol{\beta}[j], 0, 0))$ for $j \in [2, .., n]$.
   - [**Change**] Compute $\mathsf{CTK}[\mathsf{ind}]_k \leftarrow \mathsf{cIPE.Enc}(\mathbf{sk}'[\mathsf{ind}], (\boldsymbol{z}[\mathsf{ind}][k], -r[\mathsf{ind}]\boldsymbol{\gamma}[k], \boldsymbol{y}[\mathsf{ind}][1]\boldsymbol{z}[\mathsf{ind}][k] - r[\mathsf{ind}]\boldsymbol{\beta}[1]\boldsymbol{\gamma}[k], 0))$ for $k \in [n]$.
   - For every $\ell \in [d]$, $i_\ell \in [n]$, compute $\mathsf{CT}^{(\ell,i_\ell)}[\mathsf{ind}] = \mathsf{cIPE.Enc}(\mathbf{sk}^{(\ell,i_\ell)}, (r[\mathsf{ind}]\ \boldsymbol{x}[\mathsf{ind}][\ell, i_\ell], -r[\mathsf{ind}], 0, 0))$.
   - Output $\mathsf{CT}[\mathsf{ind}] = (\boldsymbol{x}[\mathsf{ind}], \mathsf{CT}[\mathsf{ind}]_0, \mathsf{CTC}[\mathsf{ind}]_j, \mathsf{CTK}[\mathsf{ind}]_k, \mathsf{CT}^{(\ell,i_\ell)}[\mathsf{ind}])$ for $\ell \in [d], i_\ell \in [n], j \in [n], k \in [n]$

3. Adversary also (selectively) asks for keys functions f_i for $i \in [L]$. Let us say $\Delta_i = f_i(m)$ for $i \in [L]$. Then adversary is given semi-functional keys generated as follows. Below we denote the procedure for a single function f and corresponding value Δ.
   - Parse $\mathsf{MSK} = (\{\mathbf{sk}^{(\ell,i_\ell)}, \mathsf{Key}_{\boldsymbol{I}}^{(\ell,i_\ell)}\}_{\ell,i_\ell,\boldsymbol{I}}, \boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma}, \mathbf{sk}_0)$.
   - Compute $\theta_f = f(\boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma})$.
   - Compute $\mathsf{Key}_{0,f} = \mathsf{cIPE.KeyGen}(\mathbf{sk}_0, (\theta_f, \Delta, r\theta_f, 0))$
   - Output $sk_f = (\mathsf{Key}_{0,f}, \{\mathbf{sk}^{(\ell,i_\ell)}, \mathsf{Key}_{\boldsymbol{I}}^{(\ell,i_\ell)}\}_{\ell,i_\ell,\boldsymbol{I}})$
4. Adversary then outputs a bit $b' \in \{0,1\}$

Lemma 8. *Due to security of* cIPE*, for any polymomial time distinguisher* $\mathcal{A}$*,* $|\Pr[\mathcal{A}(\mathbf{Hybrid}_{3,0}) = 1] - \Pr[\mathcal{A}(\mathbf{Hybrid}_{3,0,1})]| < 2^{-\lambda^\epsilon}$ *for some constant* $\epsilon > 0$

Proof. The only difference between $\mathbf{Hybrid}_{3,0}$ and $\mathbf{Hybrid}_{3,0,1}$ is the way how challenge ciphertext components CTC_1 and CTK_k for $k \in [n]$ are generated as well as $\mathsf{CTC}[\mathsf{ind}]_1$ and $\mathsf{CTK}[\mathsf{ind}]_k$ for $k \in [n]$ are generated.

In $\mathbf{Hybrid}_{3,0}$,

- CTC_1 is generated for the vector $(\boldsymbol{y}[1], \boldsymbol{\beta}[1], 0, 0)$. $\mathsf{CTC}[\mathsf{ind}]_1$ is generated for the vector $(\boldsymbol{y}[\mathsf{ind}][1], \boldsymbol{\beta}[1], 0, 0)$. Note that each index ind as well as the challenge ciphertext uses different cIPE master secret key.
- CTK_k is generated for $(\boldsymbol{z}[k], -r\boldsymbol{\gamma}[k], 0, 0)$. $\mathsf{CTK}[\mathsf{ind}]_k$ is generated for $(\boldsymbol{z}[\mathsf{ind}][k], -r[\mathsf{ind}]\boldsymbol{\gamma}[k], 0, 0)$.

In $\mathbf{Hybrid}_{3,0,1}$,

- CTC_1 is generated for the vector $(0, 0, 1, 0)$. $\mathsf{CTC}[\mathsf{ind}]_1$ is generated for the vector $(0, 0, 1, 0)$. Note that each index ind as well as the challenge ciphertext uses different cIPE master secret key.
- CTK_k is generated for $(\boldsymbol{z}[k], -r\boldsymbol{\gamma}[k], \boldsymbol{y}[1]\boldsymbol{z}[k] - r\boldsymbol{\beta}[1]\boldsymbol{\gamma}[k], 0)$. $\mathsf{CTK}[\mathsf{ind}]_k$ is generated for $(\boldsymbol{z}[\mathsf{ind}][k], -r[\mathsf{ind}]\boldsymbol{\gamma}[k], \boldsymbol{y}[\mathsf{ind}][1]\boldsymbol{z}[\mathsf{ind}][k] - r[\mathsf{ind}]\boldsymbol{\beta}[1]\boldsymbol{\gamma}[k], 0)$.

Thus observe that the decryption of all $\mathsf{CTC}[\mathsf{ind}]_j$ (as well as CTC_j) with keys CTK_k for $\mathsf{ind} \in [L]$ and $j, k \in [n]$ is the same in both hybrids and thus, by the security of cIPE encryption scheme, the proof follows.

$\mathbf{Hybrid}_{3,0,2}$

1. This hybrid is the same as the previous hybrid except that now we apply SXDH wherever r and $\boldsymbol{\beta}[1]$ is involved in the product.
   - Sample $\mathsf{pp} \leftarrow \mathsf{cIPE.PPSetup}(1^\lambda)$. Let us assume $\mathsf{pp} = (e, G_1, G_2, G_T, g_1, g_2, \mathbb{Z}_{\mathbf{p}})$.
   - Run cIPE setup as follows. $\mathbf{sk}_0 \leftarrow \mathsf{cIPE.Setup}(\mathsf{pp}, 1^4)$. Thus these keys are used to encrypt vectors in $\mathbb{F}_{\mathbf{p}}^4$.
   - Then run cIPE setup algorithm $n \cdot d$ times. That is, for every $\ell \in [d]$ and $i_\ell \in [n]$, compute $\mathbf{sk}^{(\ell,i_\ell)} \leftarrow \mathsf{cIPE.Setup}(\mathsf{pp}, 1^4)$.
   - Sample $\boldsymbol{\alpha} \leftarrow \mathbb{F}_{\mathbf{p}}^{d \times n}$. Also sample $\boldsymbol{\beta}, \boldsymbol{\gamma} \leftarrow \mathbb{F}_{\mathbf{p}}^n$.
   - Sample $r \leftarrow \mathbb{F}_{\mathbf{p}}$ (randomness used for the ciphertext).

- [**Change**] For every $\ell \in [d]$ and $\boldsymbol{I} = (i_\ell, ..., i_d, j, k)$, set $w_{\boldsymbol{I}} = r\boldsymbol{\alpha}[\ell, i_\ell] \cdots \boldsymbol{\alpha}[\ell, i_d]\boldsymbol{\beta}[j]\boldsymbol{\gamma}[k]$ for $j \neq 1$, and randomly sampled element otherwise.
- [**Change**] For $\ell \in [d]$, $i_\ell \in [n]$ and every set $\boldsymbol{I} = (i_\ell, ..., i_d, j, k) \in [n]^{d-\ell+3}$ and $\boldsymbol{I}' = (i_{\ell+1}, ..., i_d, j, k)$ compute $\mathsf{Key}_{\boldsymbol{I}}^{(\ell,i_\ell)} =$

$$\mathsf{cIPE.KeyGen}(\mathbf{sk}^{(\ell,i_\ell)}, (\boldsymbol{\alpha}[\ell+1, i_{\ell+1}] \cdots \boldsymbol{\alpha}[d, i_d]\boldsymbol{\beta}[j]\boldsymbol{\gamma}[k]$$

$$, \boldsymbol{\alpha}[\ell, i_\ell] \cdots \boldsymbol{\alpha}[d, i_d]\boldsymbol{\beta}[j]\boldsymbol{\gamma}[k], (\boldsymbol{x}[\ell, i_\ell]w_{\boldsymbol{I}'} - w_{\boldsymbol{I}}, 0))$$

- Output $\mathsf{MSK} = (\{\mathbf{sk}^{(\ell,i_\ell)}, \mathsf{Key}_{\boldsymbol{I}}^{(\ell,i_\ell)}\}_{\ell,i_\ell,\boldsymbol{I}}, \boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma}, \mathbf{sk}_0)$

2. Then adversary releases challenge messages $m_i = (\boldsymbol{x}_i, \boldsymbol{y}_i, \boldsymbol{z}_i)$ for $i \in [L]$. It also gives out a challenge message $m = (\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z})$ in $\mathbb{Z}_{\mathbf{p}}^{d \times n}$. We now describe how the challenge ciphertext CT is generated. Other ciphertexts $\mathsf{CT}[i]$ for $i \in [L]$ are generated similarly as in previous hybrids.
   - Parse $\mathsf{MSK} = (\{\mathbf{sk}^{(\ell,i_\ell)}, \mathsf{Key}_{\boldsymbol{I}}^{(\ell,i_\ell)}\}_{\ell,i_\ell,\boldsymbol{I}}, \boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma}, \mathbf{sk}_0)$.
   - Compute $\mathsf{CT}_0 = \mathsf{cIPE.Enc}(\mathbf{sk}_0, (0, 0, 1, 0))$.
   - Sample $\mathbf{sk}' \leftarrow \mathsf{cIPE.Setup}(\mathsf{pp}, 1^4)$. Note a 1 placed in the second slot.
   - Compute $\mathsf{CTC}_1 = \mathsf{cIPE.Enc}(\mathbf{sk}', (0, 0, 1, 0))$ and compute $\mathsf{CTC}_j \leftarrow \mathsf{cIPE.Enc}(\mathbf{sk}', (\boldsymbol{y}[j], \boldsymbol{\beta}[j], 0, 0))$ for $j \in [2, .., n]$.
   - [**Change**] Compute $\mathsf{CTK}_k \leftarrow \mathsf{cIPE.Enc}(\mathbf{sk}', (\boldsymbol{z}[k], -r\boldsymbol{\gamma}[k], \boldsymbol{y}[1]\boldsymbol{z}[k] - w_{1,k}, 0))$ for $k \in [n]$.
   - For every $\ell \in [d]$, $i_\ell \in [n]$, compute $\mathsf{CT}^{(\ell,i_\ell)} = \mathsf{cIPE.Enc}(\mathbf{sk}^{(\ell,i_\ell)}, (0, 0, 1, 0))$.
   - Output $\mathsf{CT} = (\boldsymbol{x}, \mathsf{CT}_0, \{\mathsf{CTC}_j, \mathsf{CTK}_k, \mathsf{CT}^{(\ell,i_\ell)}\}_{\ell \in [d], i_\ell \in [n], j \in [n], k \in [n]})$

   Other ciphertexts $\mathsf{CT}[\mathsf{ind}]$ are generated as follows.
   - Parse $\mathsf{MSK} = (\{\mathbf{sk}^{(\ell,i_\ell)}, \mathsf{Key}_{\boldsymbol{I}}^{(\ell,i_\ell)}\}_{\ell,i_\ell,\boldsymbol{I}}, \boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma}, \mathbf{sk}_0)$.
   - Sample $r[\mathsf{ind}]$ from $\mathbb{F}_{\mathbf{p}}$
   - Compute $\mathsf{CT}[\mathsf{ind}]_0 = \mathsf{cIPE.Enc}(\mathbf{sk}_0, (r[\mathsf{ind}], 0, 0, 0))$.
   - Sample $\mathbf{sk}'[\mathsf{ind}] \leftarrow \mathsf{cIPE.Setup}(\mathsf{pp}, 1^4)$. Note a 1 placed in the second slot.
   - Compute $\mathsf{CTC}[\mathsf{ind}]_1 = \mathsf{cIPE.Enc}(\mathbf{sk}'[\mathsf{ind}], (0, 0, 1, 0))$ and compute $\mathsf{CTC}[\mathsf{ind}]_j \leftarrow \mathsf{cIPE.Enc}(\mathbf{sk}'[\mathsf{ind}], (\boldsymbol{y}[\mathsf{ind}][j], \boldsymbol{\beta}[j], 0, 0))$ for $j \in [2, .., n]$.
   - Compute $\mathsf{CTK}[\mathsf{ind}]_k \leftarrow \mathsf{cIPE.Enc}(\mathbf{sk}'[\mathsf{ind}], (\boldsymbol{z}[\mathsf{ind}][k], -r[\mathsf{ind}]\boldsymbol{\gamma}[k], \boldsymbol{y}[\mathsf{ind}][1]\boldsymbol{z}[\mathsf{ind}][k] - r[\mathsf{ind}]\boldsymbol{\beta}[1]\boldsymbol{\gamma}[k], 0))$ for $k \in [n]$.
   - For every $\ell \in [d]$, $i_\ell \in [n]$, compute $\mathsf{CT}^{(\ell,i_\ell)}[\mathsf{ind}] = \mathsf{cIPE.Enc}(\mathbf{sk}^{(\ell,i_\ell)}, (r[\mathsf{ind}]\boldsymbol{x}[\mathsf{ind}][\ell, i_\ell], -r[\mathsf{ind}], 0, 0))$.
   - Output $\mathsf{CT}[\mathsf{ind}] = (\boldsymbol{x}[\mathsf{ind}], \mathsf{CT}[\mathsf{ind}]_0, \mathsf{CTC}[\mathsf{ind}]_j, \mathsf{CTK}[\mathsf{ind}]_k, \mathsf{CT}^{(\ell,i_\ell)}[\mathsf{ind}])$ for all $\ell \in [d], i_\ell \in [n], j \in [n], k \in [n]$

3. Adversary also (selectively) asks for keys functions f_i for $i \in [L]$. Let us say $\Delta_i = f_i(m)$ for $i \in [L]$. Then adversary is given semi-functional keys generated as follows. Below we denote the procedure for a single function f and corresponding value Δ.
   - Parse $\mathsf{MSK} = (\{\mathbf{sk}^{(\ell,i_\ell)}, \mathsf{Key}_{\boldsymbol{I}}^{(\ell,i_\ell)}\}_{\ell,i_\ell,\boldsymbol{I}}, \boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma}, \mathbf{sk}_0)$.

- [**Change**] Compute $\phi_f = \Sigma_{\boldsymbol{I}} c_{\boldsymbol{I}} w_{\boldsymbol{I}}$. Note that any monomial of the form $r\boldsymbol{\alpha}[1, i_1] \cdots \boldsymbol{\alpha}[d, i_d]\boldsymbol{\beta}[j]\boldsymbol{\gamma}$ is replaced with random $w_{i_1,..,i_\ell,j,k}$ if $j = 1$ and unchanged otherwise. This ϕ_f is used below.
- [**Change**] Compute $\mathsf{Key}_{0,f} = \mathsf{cIPE.KeyGen}(\mathbf{sk}_0, (\theta_f, \Delta, \phi_f, 0))$.
- Output $sk_f = (\mathsf{Key}_{0,f}, \{\mathbf{sk}^{(\ell,i_\ell)}, \mathsf{Key}_{\boldsymbol{I}}^{(\ell,i_\ell)}\}_{\ell,i_\ell,\boldsymbol{I}})$

4. Adversary then outputs a bit $b' \in \{0, 1\}$

Lemma 9. *If SXDH holds relative to* PPGen*, then, for any polymomial time distinguisher* $\mathcal{A}$*,* $|\Pr[\mathcal{A}(\mathbf{Hybrid}_{3,0,1}) = 1] - \Pr[\mathcal{A}(\mathbf{Hybrid}_{3,0,2})]| < 2^{-\lambda^\epsilon}$ *for some constant* $\epsilon > 0$

Proof. The only difference between $\mathbf{Hybrid}_{3,0,1}$ and $\mathbf{Hybrid}_{3,0,2}$ is the way how challenge ciphertext components CTK_k, function key components $\mathsf{Key}_{0,f}$ and $\mathsf{CT}^{(\ell,i_\ell)}$ are generated. In all the components exponents $\boldsymbol{\alpha}$, $\boldsymbol{\gamma}$, r and $\boldsymbol{\beta}[1]$ only appear in group 2 (cIPE key elements). Thus we can apply SXDH in steps as follows. Starting from $\mathbf{Hybrid}_{3,0,1}$:

1. Replace all occurrences of $r\boldsymbol{\beta}[1]$ with w_1.
2. Then replace all occurrences of $(r\boldsymbol{\beta}[1]) \cdot \boldsymbol{\gamma}[k]$ with $w_{1,k}$.
3. Inductively for all $\boldsymbol{I} = (i_\ell, ..., i_d, 1, k)$ replace $r\boldsymbol{\alpha}[\ell, i_\ell \cdots \boldsymbol{\alpha}[d, i_d]\boldsymbol{\beta}[1]\boldsymbol{\gamma}[k]$ with a random $w_{\boldsymbol{I}}$. Note this can be done because corresponding elements $\boldsymbol{\alpha}$, r, $\boldsymbol{\beta}[1]$ and $\boldsymbol{\gamma}$ occur only in group 2.

This distribution exactly corresponds to $\mathbf{Hybrid}_{3,0,2}$.

$\mathbf{Hybrid}_{3,0,3}$

1. This hybrid is the same as the previous hybrid except that the challenge ciphertext component CTK_k for every $k \in [n]$ and $\mathsf{Key}_{0,f}$ for all queried f are generated differently.
   - Sample $\mathsf{pp} \leftarrow \mathsf{cIPE.PPSetup}(1^\lambda)$. Let us assume $\mathsf{pp} = (e, G_1, G_2, G_T, g_1, g_2, \mathbb{Z}_{\mathbf{p}})$.
   - Run cIPE setup as follows. $\mathbf{sk}_0 \leftarrow \mathsf{cIPE.Setup}(\mathsf{pp}, 1^4)$. Thus these keys are used to encrypt vectors in $\mathbb{F}_{\mathbf{p}}^4$.
   - Then run cIPE setup algorithm $n \cdot d$ times. That is, for every $\ell \in [d]$ and $i_\ell \in [n]$, compute $\mathbf{sk}^{(\ell,i_\ell)} \leftarrow \mathsf{cIPE.Setup}(\mathsf{pp}, 1^4)$.
   - Sample $\boldsymbol{\alpha} \leftarrow \mathbb{F}_{\mathbf{p}}^{d \times n}$. Also sample $\boldsymbol{\beta}, \boldsymbol{\gamma} \leftarrow \mathbb{F}_{\mathbf{p}}^n$.
   - Sample $r \leftarrow \mathbb{F}_{\mathbf{p}}$ (randomness used for the ciphertext).
   - For every $\ell \in [d]$ and $\boldsymbol{I} = (i_\ell, ..., i_d, j, k)$ set $w_{\boldsymbol{I}} = r\boldsymbol{\alpha}[\ell, i_\ell] \cdots \boldsymbol{\alpha}[\ell, i_d]\boldsymbol{\beta}[j]\boldsymbol{\gamma}[k]$ for $j \neq 1$, and randomly sampled element otherwise.
   - For $\ell \in [d]$, $i_\ell \in [n]$ and every set $\boldsymbol{I} = (i_\ell, ..., i_d, j, k) \in [n]^{d-\ell+3}$ and $\boldsymbol{I}' = (i_{\ell+1}, ..., i_d, j, k)$ compute $\mathsf{Key}_{\boldsymbol{I}}^{(\ell,i_\ell)} =$

$$\mathsf{cIPE.KeyGen}(\mathbf{sk}^{(\ell,i_\ell)}, (\boldsymbol{\alpha}[\ell+1, i_{\ell+1}] \cdots \boldsymbol{\alpha}[d, i_d]\boldsymbol{\beta}[j]\boldsymbol{\gamma}[k]$$

$$, \boldsymbol{\alpha}[\ell, i_\ell] \cdots \boldsymbol{\alpha}[d, i_d]\boldsymbol{\beta}[j]\boldsymbol{\gamma}[k], (\boldsymbol{x}[\ell, i_\ell] w_{\boldsymbol{I}'} - w_{\boldsymbol{I}}, 0))$$

- Output $\mathsf{MSK} = (\{\mathbf{sk}^{(\ell,i_\ell)}, \mathsf{Key}_{\boldsymbol{I}}^{(\ell,i_\ell)}\}_{\ell,i_\ell,\boldsymbol{I}}, \boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma}, \mathbf{sk}_0)$

2. Then adversary releases challenge messages $m_i = (\boldsymbol{x}_i, \boldsymbol{y}_i, \boldsymbol{z}_i)$ for $i \in [L]$. It also gives out a challenge message $m = (\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z})$ in $\mathbb{Z}_{\mathbf{p}}^{d\times n}$. We now describe how the challenge ciphertext CT is generated. Other ciphertexts $\mathsf{CT}[i]$ for $i \in [L]$ are generated similarly as in previous hybrids.
   - Parse $\mathsf{MSK} = (\{\mathbf{sk}^{(\ell,i_\ell)}, \mathsf{Key}_{\boldsymbol{I}}^{(\ell,i_\ell)}\}_{\ell,i_\ell,\boldsymbol{I}}, \boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma}, \mathbf{sk}_0)$.
   - Compute $\mathsf{CT}_0 = \mathsf{cIPE.Enc}(\mathbf{sk}_0, (0,0,1,0))$.
   - Sample $\mathbf{sk}' \leftarrow \mathsf{cIPE.Setup}(\mathsf{pp}, 1^4)$. Note a 1 placed in the second slot.
   - Compute $\mathsf{CTC}_1 = \mathsf{cIPE.Enc}(\mathbf{sk}', (0,0,1,0))$ and compute $\mathsf{CTC}_j \leftarrow \mathsf{cIPE.Enc}(\mathbf{sk}', (\boldsymbol{y}[j], \boldsymbol{\beta}[j], 0, 0))$ for $j \in [2,..,n]$.
   - [**Change**] Compute $\mathsf{CTK}_k \leftarrow \mathsf{cIPE.Enc}(\mathbf{sk}', (\boldsymbol{z}[k], -r\boldsymbol{\gamma}[k], 0 - w_{1,k}, 0))$ for $k \in [n]$.
   - For every $\ell \in [d]$, $i_\ell \in [n]$, compute $\mathsf{CT}^{(\ell,i_\ell)} = \mathsf{cIPE.Enc}(\mathbf{sk}^{(\ell,i_\ell)}, (0,0,1,0))$.
   - Output $\mathsf{CT} = (\boldsymbol{x}, \mathsf{CT}_0, \{\mathsf{CTC}_j, \mathsf{CTK}_k, \mathsf{CT}^{(\ell,i_\ell)}\}_{\ell\in[d],i_\ell\in[n],j\in[n],k\in[n]})$

   Other ciphertexts $\mathsf{CT}[\mathsf{ind}]$ are generated as follows.
   - Parse $\mathsf{MSK} = (\{\mathbf{sk}^{(\ell,i_\ell)}, \mathsf{Key}_{\boldsymbol{I}}^{(\ell,i_\ell)}\}_{\ell,i_\ell,\boldsymbol{I}}, \boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma}, \mathbf{sk}_0)$.
   - Sample $r[\mathsf{ind}]$ from $\mathbb{F}_{\mathbf{p}}$
   - Compute $\mathsf{CT}[\mathsf{ind}]_0 = \mathsf{cIPE.Enc}(\mathbf{sk}_0, (r[\mathsf{ind}], 0, 0, 0))$.
   - Sample $\mathbf{sk}'[\mathsf{ind}] \leftarrow \mathsf{cIPE.Setup}(\mathsf{pp}, 1^4)$. Note a 1 placed in the second slot.
   - Compute $\mathsf{CTC}[\mathsf{ind}]_1 = \mathsf{cIPE.Enc}(\mathbf{sk}'[\mathsf{ind}], (0,0,1,0))$ and compute $\mathsf{CTC}[\mathsf{ind}]_j \leftarrow \mathsf{cIPE.Enc}(\mathbf{sk}'[\mathsf{ind}], (\boldsymbol{y}[\mathsf{ind}][j], \boldsymbol{\beta}[j], 0, 0))$ for $j \in [2,..,n]$.
   - Compute $\mathsf{CTK}[\mathsf{ind}]_k \leftarrow \mathsf{cIPE.Enc}(\mathbf{sk}'[\mathsf{ind}], (\boldsymbol{z}[\mathsf{ind}][k], -r[\mathsf{ind}]\boldsymbol{\gamma}[k], \boldsymbol{y}[\mathsf{ind}][1]\boldsymbol{z}[\mathsf{ind}][k] - r[\mathsf{ind}]\boldsymbol{\beta}[1]\boldsymbol{\gamma}[k], 0))$ for $k \in [n]$.
   - For every $\ell \in [d]$, $i_\ell \in [n]$, compute $\mathsf{CT}^{(\ell,i_\ell)}[\mathsf{ind}] = \mathsf{cIPE.Enc}(\mathbf{sk}^{(\ell,i_\ell)}, (r[\mathsf{ind}]\boldsymbol{x}[\mathsf{ind}][\ell, i_\ell], -r[\mathsf{ind}], 0, 0))$.
   - Output $\mathsf{CT}[\mathsf{ind}] = (\boldsymbol{x}[\mathsf{ind}], \mathsf{CT}[\mathsf{ind}]_0, \mathsf{CTC}[\mathsf{ind}]_j, \mathsf{CTK}[\mathsf{ind}]_k, \mathsf{CT}^{(\ell,i_\ell)}[\mathsf{ind}])$ for all $\ell \in [d], i_\ell \in [n], j \in [n], k \in [n]$.

3. Adversary also (selectively) asks for keys functions f_i for $i \in [L]$. Let us say $\Delta_i = f_i(m)$ for $i \in [L]$. Then adversary is given semi-functional keys generated as follows. Below we denote the procedure for a single function f and corresponding value Δ.
   - Parse $\mathsf{MSK} = (\{\mathbf{sk}^{(\ell,i_\ell)}, \mathsf{Key}_{\boldsymbol{I}}^{(\ell,i_\ell)}\}_{\ell,i_\ell,\boldsymbol{I}}, \boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma}, \mathbf{sk}_0)$.
   - [**Change**] Compute $\phi_f = \Sigma_{\boldsymbol{I}} c_{\boldsymbol{I}} w_{\boldsymbol{I}} + f(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}) - f(\boldsymbol{x}, \boldsymbol{y1}, \boldsymbol{z})$. Here $\boldsymbol{y1} = (0, y_2, ..., y_n)$. Note that any monomial $r\boldsymbol{\alpha}[1, i_1] \cdots \boldsymbol{\alpha}[d, i_d]\boldsymbol{\beta}[j]\boldsymbol{\gamma}$ is replaced with random $w_{i_1,..,i_\ell,j,k}$ if $j = 1$ and unchanged otherwise. This ϕ_f is used below.
   - [**Change**] Compute $\mathsf{Key}_{0,f} = \mathsf{cIPE.KeyGen}(\mathbf{sk}_0, (\theta_f, \Delta, \phi_f, 0))$.
   - Output $sk_f = (\mathsf{Key}_{0,f}, \{\mathbf{sk}^{(\ell,i_\ell)}, \mathsf{Key}_{\boldsymbol{I}}^{(\ell,i_\ell)}\}_{\ell,i_\ell,\boldsymbol{I}})$

4. Adversary then outputs a bit $b' \in \{0, 1\}$

Lemma 10. *For any distinguisher $\mathcal{A}$, $|\Pr[\mathcal{A}(\mathbf{Hybrid}_{3,0,2}) = 1] - \Pr[\mathcal{A}(\mathbf{Hybrid}_{3,0,3})]| = 0$ (statistical indistinguishability)*

Proof. Let us now look at the hardwirings made in CTK_k for $k \in [n]$ (corresponding to the third component). They are described below in both the hybrids.

Distribution Hybrid$_{3,0,2}$:

- CTK_k for $k \in [n]$ has $\boldsymbol{y}[1]\boldsymbol{z}[k] + w_{1,k}$.
- For every $\ell \in [d]$, vector $\boldsymbol{I} = (i_\ell, ..., i_d, j, k)$ and vector $\boldsymbol{I}' = (i_{\ell+1}, ..., i_d, j, k)$ with $j = 1$, $\mathsf{Key}_{\boldsymbol{I}'}^{(\ell,i_\ell)}$ has $\boldsymbol{x}[\ell, i_\ell]w_{\boldsymbol{I}'} - w_{\boldsymbol{I}}$ hardwired.
- $\mathsf{Key}_{0,f}$ has $\Sigma_{\boldsymbol{I}=(i_1,...,i_d,j,k)} c_I w_{\boldsymbol{I}}$ hardwired where $f = \Sigma_{\boldsymbol{I}=(i_1,..,i_d,j,k)} c_{\boldsymbol{I}} \boldsymbol{x}[1, i_1] \cdots \boldsymbol{x}[d, i_d]\boldsymbol{y}[j]\boldsymbol{z}[k]$.

Distribution Hybrid$_{3,0,3}$:

- CTK_k for $k \in [n]$ has $w_{1,k}$ hardwired.
- For every $\ell \in [d]$, vector $\boldsymbol{I} = (i_\ell, ..., i_d, j, k)$ and vector $\boldsymbol{I}' = (i_{\ell+1}, ..., i_d, j, k)$ with $j = 1$, $\mathsf{Key}_{\boldsymbol{I}'}^{(\ell,i_\ell)}$ has $\boldsymbol{x}[\ell, i_\ell]w_{\boldsymbol{I}'} - w_{\boldsymbol{I}}$ hardwired.
- $\mathsf{Key}_{0,f}$ has $\Sigma_{\boldsymbol{I}=(i_1,...,i_d,j,k)} c_I w_{\boldsymbol{I}} + \Sigma_{\boldsymbol{I}=(i_1,...,i_d,1,k)} c_{\boldsymbol{I}} \boldsymbol{x}[1, i_1] \cdots \boldsymbol{x}[d, i_d]\boldsymbol{y}[1]\boldsymbol{z}[k]$ hardwired where $f = \Sigma_{\boldsymbol{I}=(i_1,..,i_d,j,k)} c_{\boldsymbol{I}} \boldsymbol{x}[1, i_1] \cdots \boldsymbol{x}[d, i_d]\boldsymbol{y}[j]\boldsymbol{z}[k]$.

We claim these distributions are identical. We start from **Distribution Hybrid**$_{3,0,2}$ and perform a change in variables to land to **Distribution Hybrid**$_{3,0,3}$. Here is the next distribution.

Distribution 1

- CTK_k for $k \in [n]$ has $\boldsymbol{y}[1]\boldsymbol{z}[k] + w_{1,k}$ hardwired. Set $w'_{1,k} = \boldsymbol{y}[1]\boldsymbol{z}[k] + w_{1,k}$.
- For every $\ell \in [d]$, vector $\boldsymbol{I} = (i_\ell, ..., i_d, j, k)$ and vector $\boldsymbol{I}' = (i_{\ell+1}, ..., i_d, j, k)$ with $j = 1$, $\mathsf{Key}_{\boldsymbol{I}'}^{(\ell,i_\ell)}$ has $\boldsymbol{x}[\ell, i_\ell]w_{\boldsymbol{I}'} - w_{\boldsymbol{I}}$ hardwired. Set $w'_{\boldsymbol{I}} = \boldsymbol{x}[\ell, i_\ell]w_{\boldsymbol{I}'} - w_{\boldsymbol{I}}$.
- $\mathsf{Key}_{0,f}$ has $\psi_f = \Sigma_{\boldsymbol{I}=(i_1,...,i_d,j,k)} c_I w_{\boldsymbol{I}}$ hardwired where $f = \Sigma_{\boldsymbol{I}=(i_1,..,i_d,j,k)} c_{\boldsymbol{I}} \boldsymbol{x}[1, i_1] \cdots \boldsymbol{x}[d, i_d]\boldsymbol{y}[j]\boldsymbol{z}[k]$. We can decompose $\psi_f = \psi_{f,1} + \psi_{f,\neq 1}$. Here $\psi_{f,\neq 1} = \Sigma_{\boldsymbol{I}=(i_1,...,i_d,j\neq 1,k)} c_{\boldsymbol{I}} r\boldsymbol{\alpha}[1, i_1] \cdots \boldsymbol{\alpha}[d, i_d]\boldsymbol{\beta}[j]\boldsymbol{\beta}[k]$ and $\psi_{f,1} = \Sigma_{\boldsymbol{I}=(i_\ell,...,i_d,1,k)} c_{\boldsymbol{I}} w_{\boldsymbol{I}}$
- We can re-write $\psi_{f,1} = \Sigma_{\boldsymbol{I}=(i_\ell,...,i_d,1,k)} \rho'_{\boldsymbol{I}} w'_{\boldsymbol{I}} + f(\boldsymbol{x}, \boldsymbol{y} - \boldsymbol{y1}, \boldsymbol{z})$. This follows from correctness of decryption. Note that these coefficients $\rho'_{\boldsymbol{I}}$ are derived from the coefficients $\rho_{\boldsymbol{I}_v}$ for $v \in [d]$ (which intern depends only on $\boldsymbol{x}$) and the description of polynomial f as pointed out in the description of decryption algorithm. Here $\boldsymbol{y1} = (0, y_2, ..., y_n)$

Now we describe our distribution 2.

Distribution 2

- Set $w'_{1,k}$ randomly in CTK_k for $k \in [n]$.
- For every $\ell \in [d]$ vector $\boldsymbol{I} = (i_\ell, ..., i_d, j, k)$ and vector $\boldsymbol{I}' = (i_{\ell+1}, ..., i_d, j, k)$ with $j = 1$, $\mathsf{Key}_{\boldsymbol{I}'}^{(\ell,i_\ell)}$ has $w'_{\boldsymbol{I}}$ hardwired. Set $w'_{\boldsymbol{I}}$ randomly.
- $\mathsf{Key}_{0,f}$ has $\psi_f = \Sigma_{\boldsymbol{I}=(i_1,...,i_d,j,k)} c_I w_{\boldsymbol{I}}$ hardwired where $f = \Sigma_{\boldsymbol{I}=(i_1,..,i_d,j,k)} c_{\boldsymbol{I}} \boldsymbol{x}[1, i_1] \cdots \boldsymbol{x}[d, i_d]\boldsymbol{y}[j]\boldsymbol{z}[k]$. We can decompose $\psi_f = \psi_{f,1} + \psi_{f,\neq 1}$. Here $\psi_{f,\neq 1} = \Sigma_{\boldsymbol{I}=(i_1,...,i_d,j\neq 1,k)} c_{\boldsymbol{I}} r\boldsymbol{\alpha}[1, i_1] \cdots \boldsymbol{\alpha}[d, i_d]\boldsymbol{\beta}[j]\boldsymbol{\beta}[k]$ and $\psi_{f,1} = \Sigma_{\boldsymbol{I}=(i_\ell,...,i_d,1,k)} c_{\boldsymbol{I}} w_{\boldsymbol{I}}$
- We can re-write $\psi_{f,1} = \Sigma_{\boldsymbol{I}=(i_\ell,...,i_d,1,k)} \rho'_{\boldsymbol{I}} w'_{\boldsymbol{I}} + f(\boldsymbol{x}, \boldsymbol{y} - \boldsymbol{y1}, \boldsymbol{z})$. This follows from correctness of decryption. Note that these coefficients $\rho'_{\boldsymbol{I}}$ are derived from the coefficients $\rho_{\boldsymbol{I}_v}$ for $v \in [d]$ (which intern depends only on $\boldsymbol{x}$) and the description of polynomial f as pointed out in the description of decryption algorithm. Here $\boldsymbol{y1} = (0, y_2, ..., y_n)$

Note that **Distribution** 1 is identical to **Distribution** 2 as

for every $\boldsymbol{I} = (i_\ell, ..., i_d, 1, k)$ and $\boldsymbol{I}' = (i_{\ell+1}, .., i_d, 1, k)$ $w'_{\boldsymbol{I}} = \boldsymbol{x}[\ell, i_\ell] w_{\boldsymbol{I}'} - w_{\boldsymbol{I}}$ and $w'_{1,k} = \boldsymbol{y}[1]\boldsymbol{z}[k] - w_{1,k}$. Since $w_{\boldsymbol{I}}$ is chosen randomly and independently from $w_{\boldsymbol{I}'}$, the claim follows. Once we have this, we can show that **Distribution 2** is identical to **Distribution** 3 (just by re-writing differently).

Distribution 3

- CTK_k for $k \in [n]$ has $w_{1,k}$ chosen randomly. Set $w'_{1,k} = w_{1,k}$.
- For every $\ell \in [d]$ vector $\boldsymbol{I} = (i_\ell, ..., i_d, j, k)$ and vector $\boldsymbol{I}' = (i_{\ell+1}, ..., i_d, j, k)$ with $j = 1$, $\mathsf{Key}_{\boldsymbol{I}'}^{(\ell, i_\ell)}$ has $\boldsymbol{x}[\ell, i_\ell] w_{\boldsymbol{I}'} - w_{\boldsymbol{I}}$ hardwired. Set $w'_{\boldsymbol{I}} = \boldsymbol{x}[\ell, i_\ell] w_{\boldsymbol{I}'} - w_{\boldsymbol{I}}$.
- $\mathsf{Key}_{0,f}$ has $\psi_f = \Sigma_{\boldsymbol{I}=(i_1,...,i_d,j,k)} c_I w_{\boldsymbol{I}}$ hardwired where $f = \Sigma_{\boldsymbol{I}=(i_1,..,i_d,j,k)} c_{\boldsymbol{I}}$ $\boldsymbol{x}[1, i_1] \cdots \boldsymbol{x}[d, i_d] \boldsymbol{y}[j] \boldsymbol{z}[k]$. Note that $\psi_f = \psi_{f,1} + \psi_{f,\neq 1}$. Here $\psi_{f,\neq 1} = \Sigma_{\boldsymbol{I}=(i_1,...,i_d,j\neq 1,k)} c_{\boldsymbol{I}} r \boldsymbol{\alpha}[1, i_1] \cdots \boldsymbol{\alpha}[d, i_d] \boldsymbol{\beta}[j] \boldsymbol{\beta}[k]$ and $\psi_{f,1} = +f(\boldsymbol{x}, \boldsymbol{y} - \boldsymbol{y1}, \boldsymbol{z}) + \Sigma_{\boldsymbol{I}=(i_\ell,...,i_d,1,k)} \rho'_{\boldsymbol{I}} w'_{\boldsymbol{I}}$. Note that these coefficients $\rho'_{\boldsymbol{I}}$ are derived from the coefficients $\rho_{\boldsymbol{I}_v}$ for $v \in [d]$ (which intern depends only on $\boldsymbol{x}$) and the polynomial f as pointed out in the description of decryption algorithm. Here $\boldsymbol{y1} = (0, y_2, \ldots, y_n)$

Note that this **Distribution** 3 corresponds to the **Distribution Hybrid**$_{3,0,3}$ and we are done.

Hybrid$_{3,0,4}$

1. This hybrid is the same as the previous hybrid except that we replace $w_{\boldsymbol{I}}$ for $\boldsymbol{I} = (i_\ell, ..., i_d, 1, k)$ with $r\boldsymbol{\alpha}[\ell, i_\ell] \cdots \boldsymbol{\alpha}[d, i_d] \boldsymbol{\beta}[1] \boldsymbol{\gamma}[k]$
   - Sample $\mathsf{pp} \leftarrow \mathsf{cIPE.PPSetup}(1^\lambda)$. Let us assume $\mathsf{pp} = (e, G_1, G_2, G_T, g_1, g_2, \mathbb{Z}_{\mathbf{p}})$.
   - Run cIPE setup as follows. $\mathbf{sk}_0 \leftarrow \mathsf{cIPE.Setup}(\mathsf{pp}, 1^4)$. Thus these keys are used to encrypt vectors in $\mathbb{F}_{\mathbf{p}}^4$.
   - Then run cIPE setup algorithm $n \cdot d$ times. That is, for every $\ell \in [d]$ and $i_\ell \in [n]$, compute $\mathbf{sk}^{(\ell, i_\ell)} \leftarrow \mathsf{cIPE.Setup}(\mathsf{pp}, 1^4)$.
   - Sample $\boldsymbol{\alpha} \leftarrow \mathbb{F}_{\mathbf{p}}^{d \times n}$. Also sample $\boldsymbol{\beta}, \boldsymbol{\gamma} \leftarrow \mathbb{F}_{\mathbf{p}}^n$.
   - Sample $r \leftarrow \mathbb{F}_{\mathbf{p}}$ (randomness used for the ciphertext).
   - [**Change**] For every $\ell \in [d]$ and $\boldsymbol{I} = (i_\ell, ..., i_d, j, k)$ set $w_{\boldsymbol{I}} = r\boldsymbol{\alpha}[\ell, i_\ell] \cdots \boldsymbol{\alpha}[\ell, i_d] \boldsymbol{\beta}[j] \boldsymbol{\gamma}[k]$
   - For $\ell \in [d]$, $i_\ell \in [n]$ and every set $\boldsymbol{I} = (i_{\ell+1}, ..., i_d, j, k) \in [n]^{d-\ell+2}$ and $\boldsymbol{I}' = (i_{\ell+1}, ..., i_d, j, k)$
     compute $\mathsf{Key}_{\boldsymbol{I}}^{(\ell, i_\ell)} =$

$$\mathsf{cIPE.KeyGen}(\mathbf{sk}^{(\ell, i_\ell)}, (\boldsymbol{\alpha}[\ell + 1, i_{\ell+1}] \cdots \boldsymbol{\alpha}[d, i_d] \boldsymbol{\beta}[j] \boldsymbol{\gamma}[k],$$
$$\boldsymbol{\alpha}[\ell, i_\ell] \cdots \boldsymbol{\alpha}[d, i_d] \boldsymbol{\beta}[j] \boldsymbol{\gamma}[k], (\boldsymbol{x}[\ell, i_\ell] w_{\boldsymbol{I}'} - w_{\boldsymbol{I}}, 0))$$

   - Output $\mathsf{MSK} = (\{\mathbf{sk}^{(\ell, i_\ell)}, \mathsf{Key}_{\boldsymbol{I}}^{(\ell, i_\ell)}\}_{\ell, i_\ell, \boldsymbol{I}}, \boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma}, \mathbf{sk}_0)$
2. Then adversary releases challenge messages $m_i = (\boldsymbol{x}_i, \boldsymbol{y}_i, \boldsymbol{z}_i)$ for $i \in [L]$. It also gives out a challenge message $m = (\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z})$ in $\mathbb{Z}_{\mathbf{p}}^{d \times n}$. We now describe how the challenge ciphertext CT is generated. Other ciphertexts $\mathsf{CT}[i]$ for $i \in [L]$ are generated similarly as in previous hybrids.
   - Parse $\mathsf{MSK} = (\{\mathbf{sk}^{(\ell, i_\ell)}, \mathsf{Key}_{\boldsymbol{I}}^{(\ell, i_\ell)}\}_{\ell, i_\ell, \boldsymbol{I}}, \boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma}, \mathbf{sk}_0)$.

- Compute $\mathsf{CT}_0 = \mathsf{cIPE.Enc}(\mathbf{sk}_0, (0,0,1,0))$.
- Sample $\mathbf{sk}' \leftarrow \mathsf{cIPE.Setup}(\mathsf{pp}, 1^4)$. Note a 1 placed in the second slot.
- Compute $\mathsf{CTC}_1 = \mathsf{cIPE.Enc}(\mathbf{sk}', (0,0,1,0))$ and compute $\mathsf{CTC}_j \leftarrow \mathsf{cIPE.Enc}(\mathbf{sk}', (\boldsymbol{y}[j], \boldsymbol{\beta}[j], 0, 0))$ for $j \in [2,..,n]$.
- Compute $\mathsf{CTK}_k \leftarrow \mathsf{cIPE.Enc}(\mathbf{sk}', (\boldsymbol{z}[k], -r\boldsymbol{\gamma}[k], 0 - w_{1,k}, 0))$ for $k \in [n]$.
- For every $\ell \in [d]$, $i_\ell \in [n]$, compute $\mathsf{CT}^{(\ell,i_\ell)} = \mathsf{cIPE.Enc}(\mathbf{sk}^{(\ell,i_\ell)}, (0,0,1,0))$.
- Output $\mathsf{CT} = (\boldsymbol{x}, \mathsf{CT}_0, \{\mathsf{CTC}_j, \mathsf{CTK}_k, \mathsf{CT}^{(\ell,i_\ell)}\}_{\ell\in[d],i_\ell\in[n],j\in[n],k\in[n]})$

Other ciphertexts $\mathsf{CT}[\mathsf{ind}]$ are generated as follows.

- Parse $\mathsf{MSK} = (\{\mathbf{sk}^{(\ell,i_\ell)}, \mathsf{Key}_{\boldsymbol{I}}^{(\ell,i_\ell)}\}_{\ell,i_\ell,\boldsymbol{I}}, \boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma}, \mathbf{sk}_0)$.
- Sample $r[\mathsf{ind}]$ from $\mathbb{F}_{\mathbf{p}}$
- Compute $\mathsf{CT}[\mathsf{ind}]_0 = \mathsf{cIPE.Enc}(\mathbf{sk}_0, (r[\mathsf{ind}], 0, 0, 0))$.
- Sample $\mathbf{sk}'[\mathsf{ind}] \leftarrow \mathsf{cIPE.Setup}(\mathsf{pp}, 1^4)$. Note a 1 placed in the second slot.
- Compute $\mathsf{CTC}[\mathsf{ind}]_1 = \mathsf{cIPE.Enc}(\mathbf{sk}'[\mathsf{ind}], (0,0,1,0))$.
- Compute $\mathsf{CTC}[\mathsf{ind}]_j \leftarrow \mathsf{cIPE.Enc}(\mathbf{sk}'[\mathsf{ind}], (\boldsymbol{y}[\mathsf{ind}][j], \boldsymbol{\beta}[j], 0, 0))$ for $j \in [2,..,n]$.
- Compute $\mathsf{CTK}[\mathsf{ind}]_k \leftarrow \mathsf{cIPE.Enc}(\mathbf{sk}'[\mathsf{ind}], (\boldsymbol{z}[\mathsf{ind}][k], -r[\mathsf{ind}]\boldsymbol{\gamma}[k], \boldsymbol{y}[\mathsf{ind}][1]$ $\boldsymbol{z}[\mathsf{ind}][k] - r[\mathsf{ind}]\boldsymbol{\beta}[1]\boldsymbol{\gamma}[k], 0))$ for $k \in [n]$.
- For every $\ell \in [d]$, $i_\ell \in [n]$, compute $\mathsf{CT}^{(\ell,i_\ell)}[\mathsf{ind}] = \mathsf{cIPE.Enc}(\mathbf{sk}^{(\ell,i_\ell)}, (r[\mathsf{ind}]\boldsymbol{x}[\mathsf{ind}][\ell, i_\ell], -r[\mathsf{ind}], 0, 0))$.
- Output $\mathsf{CT}[\mathsf{ind}] = (\boldsymbol{x}[\mathsf{ind}], \mathsf{CT}[\mathsf{ind}]_0, \mathsf{CTC}[\mathsf{ind}]_j, \mathsf{CTK}[\mathsf{ind}]_k, \mathsf{CT}^{(\ell,i_\ell)}[\mathsf{ind}])$ for all $\ell \in [d], i_\ell \in [n], j \in [n], k \in [n]$.

3. Adversary also (selectively) asks for keys functions f_i for $i \in [L]$. Let us say $\Delta_i = f_i(m)$ for $i \in [L]$. Then adversary is given semi-functional keys generated as follows. Below we denote the procedure for a single function f and corresponding value Δ.
   - Parse $\mathsf{MSK} = (\{\mathbf{sk}^{(\ell,i_\ell)}, \mathsf{Key}_{\boldsymbol{I}}^{(\ell,i_\ell)}\}_{\ell,i_\ell,\boldsymbol{I}}, \boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma}, \mathbf{sk}_0)$.
   - [**Change**] Compute $\phi_f = \Sigma_{\boldsymbol{I}} c_{\boldsymbol{I}} w_{\boldsymbol{I}} + f(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}) - f(\boldsymbol{x}, \boldsymbol{y1}, \boldsymbol{z})$. Here $\boldsymbol{y1} = (0, y_2, ..., y_n)$. Note that $\phi_f = r\theta_f + f(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}) - f(\boldsymbol{x}, \boldsymbol{y1}, \boldsymbol{z})$ This ϕ_f is used below.
   - Compute $\mathsf{Key}_{0,f} = \mathsf{cIPE.KeyGen}(\mathbf{sk}_0, (\theta_f, \Delta, \phi_f, 0))$.
   - Output $sk_f = (\mathsf{Key}_{0,f}, \{\mathbf{sk}^{(\ell,i_\ell)}, \mathsf{Key}_{\boldsymbol{I}}^{(\ell,i_\ell)}\}_{\ell,i_\ell,\boldsymbol{I}})$
4. Adversary then outputs a bit $b' \in \{0,1\}$

Lemma 11. *If SXDH holds relative to* PPGen*, then, for any polymomial time distinguisher* $\mathcal{A}$*,* $|\Pr[\mathcal{A}(\mathbf{Hybrid}_{3,0,3}) = 1] - \Pr[\mathcal{A}(\mathbf{Hybrid}_{3,0,4})]| < 2^{-\lambda^\epsilon}$ *for some constant* $\epsilon > 0$

Proof. The proof of this similar to lemma 9.

Lemma 12. *Due to security of* cIPE*, for any polymomial time distinguisher* $\mathcal{A}$*,* $|\Pr[\mathcal{A}(\mathbf{Hybrid}_{3,1}) = 1] - \Pr[\mathcal{A}(\mathbf{Hybrid}_{3,0,4})]| < 2^{-\lambda^\epsilon}$ *for some constant* $\epsilon > 0$

Proof. The proof of this lemma is identical to the proof of lemma 8.

8 Construction of $i\mathcal{O}$

Having constructed $(d+2)-$restricted FE for $d \geq 1$ from $SXDH$, we are ready to build $i\mathcal{O}$. First step is to build a semi-functional FE scheme for degree $d+2$ polynomials. In [5], they showed that:

Theorem 5. *Assuming tempered degree three encoding scheme and three-restricted FE scheme exists, there exists a semi-functional FE scheme for cubic polynomials.*

A natural generalisation to degree $(d+2)$ goes via the notion of degree $(d+2)$ tempered encodings. We define semi-functional FE for degree $(d+2)$ polynomials in Section A and degree $(d+2)$ tempered encodings in Section B. In Section B, we construct degree $(d+2)$ tempered encodings from a perturbation resilient generator implementable by a $(d+2)$ restricted FE scheme. Thus we have the following theorem:

Theorem 6. *For any constant integer $d \geq 1$, Assuming:*

1. *Subexponentially hard LWE*
2. *Perturbation resilient generators implementable by $(d+2)$ degree restricted FE for any constant $d \geq 1$ with a stretch of $k^{1+\epsilon}$ (k being the input size and $\epsilon > 0$ is a constant).*

There exists a degree $(d+2)$ tempered encoding scheme.

Once we have tempered degree $(d+2)$ encoding scheme, as shown in [5][2], the following holds:

Theorem 7. *For any constant integer $d \geq 1$, Assuming:*

1. *Tempered degree $(d+2)-$encoding scheme for any constant $d \geq 1$.*
2. *$(d+2)-$degree restricted FE for any constant $d \geq 1$.*

There exists a degree $(d+2)$ semi-functional FE scheme for degree $(d+2)$ polynomials.

Let us now define a circuit class $\mathcal{C}_{n,s}$

Definition 8. *Let λ be a security parameter. By $\mathcal{C}_{n,s}$ we denote the class of circuits $C : \{0,1\}^n \to \{0,1\}^*$ with size bounded by $s(n,\lambda)$ and depth λ. Here both s and n are polynomials in λ. In particular, this class contains NC^1 circuits of size $s(n,\lambda)$.*

Then, as observed by [5, 47], such a degree $(d+2)-$semi-functional FE scheme can be used to construct semi-functional FE for circuits with weak security (refer [5] for a definition).

[2] Although, in [5], the claims were proven for $d = 1$, the framework and the proof does follow for any constant d.

Theorem 8. *For any constant integer $d \geq 1$, Assuming:*

1. *Semi-functional FE scheme for degree $d+2$ polynomials for any constant $d \geq 1$.*
2. *Block-local PRGs with block locality $d+2$ with stretch $k^{1+\epsilon}$ for some constant $\epsilon > 0$ (k being the input size).*

There exist semi-functional FE scheme for circuits (with distinguishing gap $1 - 1/\lambda$) for class $\mathcal{C}_{n,s}$ with $s = n^{1+\epsilon'}$ for some constant $\epsilon' > 0$.

Finally, [5] showed how to perform security amplification for such a semi-functional FE scheme for circuits to construct sublinear secret key FE for circuits. This amplified scheme can be bootstrapped to $i\mathcal{O}$ using works of [6, 16]. Here are the final theorem that we achieve:

Theorem 9. *For any constant $d \geq 1$, Assuming*

- *Subexponentially secure LWE.*
- *Subexponentially secure $(d+2)-$restricted FE scheme* [3].
- *PRGs with*
  - *Stretch of $k^{1+\epsilon}$ (length of input being k bits) for some constant $\epsilon > 0$.*
  - *Block locality $d+2$.*
  - *Security with* negl *distinguishing gap against adversaries of sub-exponential size*[4].
- $\mathsf{d}\Delta\mathsf{RG}$ *with a stretch of $k^{1+\epsilon'}$ for some constant $\epsilon' > 0$.*

there exists a subexponentially secure sublinear secret key FE scheme for $\mathcal{C}_{n,s}$ for $s = n^{1+\varepsilon}$ for some constant $\varepsilon > 0$.

Once, we have subexponentially secure secret key FE for $\mathcal{C}_{n,s}$, then we invoke the following theorem from [7, 47]. This theorem is based on the work of [14], which showed that sublinear secret key FE implies sublinear public key FE (assuming LWE), and the work of [6, 16] which showed that any subexponentially secure sublinear public key FE implies $i\mathcal{O}$.

Theorem 10 ([47, 7]). *Assuming*

- *Subexponentially secure LWE.*
- *Subexponentially secure sublinear secret key FE for $\mathcal{C}_{n,s}$ for $s = n^{1+\varepsilon}$ for some constant $\varepsilon > 0$.*

there an indistinguishability obfuscation scheme for P/poly.

Thus, we have the following result.

Theorem 11. *For any constant integer $d \geq 1$, Assuming*

[3] This can be implemented from SXDH for any constant positive integer d

[4] As pointed out in [5], we can allow a trade-off between the required level of security of $\mathsf{d}\Delta\mathsf{RG}$ and a $d+2$-block local PRG.

- *Subexponentially secure LWE.*
- *Subexponentially secure* $(d+2)-$*restricted FE scheme* [5].
- *PRGs with*
  - *Stretch of* $k^{1+\epsilon}$ *(length of input being* k *bits) for some constant* $\epsilon > 0$.
  - *Block locality* $d+2$.
  - *Security with* negl *distinguishing gap against adversaries of sub-exponential size*[6].
- $\mathsf{d}\Delta\mathsf{RG}$ *with a stretch of* $k^{1+\epsilon'}$ *for some constant* $\epsilon' > 0$.

there exists an $i\mathcal{O}$ *scheme for* $P/poly$.

Since, $(d+2)-$restricted FE can be implemented using SXDH, we get the following theorem.

Theorem 12. *For any constant integer* $d \geq 1$*, Assuming*

- *Subexponentially hard LWE.*
- *Subexponentially hard SXDH*
- *PRGs with*
  - *Stretch of* $k^{1+\epsilon}$ *(length of input being* k *bits) for some constant* $\epsilon > 0$.
  - *Block locality* $d+2$.
  - *Security with* negl *distinguishing gap against adversaries of sub-exponential size.*
- $\mathsf{d}\Delta\mathsf{RG}$ *with a stretch of* $k^{1+\epsilon'}$ *for some constant* $\epsilon' > 0$[7].

there exists an $i\mathcal{O}$ *scheme for* $P/poly$.

Here is the version with the tradeoff.

Theorem 13. *For any constant integer* $d \geq 1$*, two distinguishing gaps* $\mathsf{adv}_1, \mathsf{adv}_2$*, if* $\mathsf{adv}_1 + \mathsf{adv}_2 \leq 1 - 2/\lambda$ *then assuming,*

- *Subexponentially hard LWE.*
- *Subexponentially hard SXDH.*
- *PRGs with*
  - *Stretch of* $k^{1+\epsilon}$ *(length of input being* k *bits) for some constant* $\epsilon > 0$.
  - *Block locality* $d+2$.
  - *Security with distinguishing gap bounded by* adv_1 *against adversaries of sub-exponential size.*
- $\mathsf{d}\Delta\mathsf{RG}$ *with distinguishing gap bounded by* adv_2 *against adversaries of size* 2^λ [8].

there exists a secure $i\mathcal{O}$ *scheme for* $P/poly$.

[5] This can be implemented from SXDH for d

[6] As pointed out in [5], we can allow a trade-off between the required level of security of $\mathsf{d}\Delta\mathsf{RG}$ and a $d+2$-block local PRG.

[7] Instantiations can be found in Section 6.2

[8] Instantiations can be found in Section 6.2

9 Acknowledgement

We would like to thank Prabhanjan Ananth for preliminary discussions on the concept of a $d + 2$ restricted FE scheme. We would also like to thank Pravesh Kothari, Sam Hopkins and Boaz Barak for many useful discussions about our $\mathsf{d}\Delta\mathsf{RG}$ Candidates.

Aayush Jain and Amit Sahai are supported in part from a DARPA/ARL SAFEWARE award, NSF Frontier Award 1413955, and NSF grant 1619348, BSF grant 2012378, a Xerox Faculty Research Award, a Google Faculty Research Award, an equipment grant from Intel, and an Okawa Foundation Research Grant. This material is based upon work supported by the Defense Advanced Research Projects Agency through the ARL under Contract W911NF-15-C- 0205.

Aayush Jain is also supported by a Google PhD Fellowship in Privacy and Security.

The views expressed are those of the authors and do not reflect the official policy or position of the Department of Defense, the National Science Foundation, Google, or the U.S. Government.

References

1. Agrawal, S.: New methods for indistinguishability obfuscation: Bootstrapping and instantiation. IACR Cryptology ePrint Archive **2018**, 633 (2018)
2. Alperin-Sheriff, J., Peikert, C.: Faster bootstrapping with polynomial error. In: CRYPTO. pp. 297–314 (2014)
3. Ananth, P., Brakerski, Z., Khuarana, D., Sahai, A.: New approach against the locality barrier in obfuscation: Pseudo-independent generators. Unpublished Work (2017)
4. Ananth, P., Gupta, D., Ishai, Y., Sahai, A.: Optimizing obfuscation: Avoiding Barrington's theorem. In: ACM CCS. pp. 646–658 (2014)
5. Ananth, P., Jain, A., Khurana, D., Sahai, A.: Indistinguishability obfuscation without multilinear maps: io from lwe, bilinear maps, and weak pseudorandomness. IACR Cryptology ePrint Archive **2018**, 615 (2018)
6. Ananth, P., Jain, A.: Indistinguishability obfuscation from compact functional encryption. In: Advances in Cryptology–CRYPTO 2015, pp. 308–326. Springer (2015)
7. Ananth, P., Sahai, A.: Projective arithmetic functional encryption and indistinguishability obfuscation from degree-5 multilinear maps. EUROCRYPT (2017)
8. Arora, S., Ge, R.: New algorithms for learning in presence of errors. In: Automata, Languages and Programming - 38th International Colloquium, ICALP 2011, Zurich, Switzerland, July 4-8, 2011, Proceedings, Part I. pp. 403–415 (2011)
9. Badrinarayanan, S., Miles, E., Sahai, A., Zhandry, M.: Post-zeroizing obfuscation: New mathematical tools, and the case of evasive circuits. In: Advances in Cryptology - EUROCRYPT. pp. 764–791 (2016)
10. Barak, B., Garg, S., Kalai, Y.T., Paneth, O., Sahai, A.: Protecting obfuscation against algebraic attacks. In: CRYPTO. pp. 221–238 (2014)
11. Barak, B., Goldreich, O., Impagliazzo, R., Rudich, S., Sahai, A., Vadhan, S.P., Yang, K.: On the (im)possibility of obfuscating programs. In: Advances in Cryptology - CRYPTO 2001, 21st Annual International Cryptology Conference, Santa Barbara, California, USA, August 19-23, 2001, Proceedings. pp. 1–18 (2001)

12. Barak, B., Hopkins, S., Jain, A., Kothari, P., Sahai, A.: Sum-of-squares meets program obfuscation, revisited. Unpublished Work (2018)
13. Bartusek, J., Guan, J., Ma, F., Zhandry, M.: Preventing zeroizing attacks on GGH15. IACR Cryptology ePrint Archive **2018**, 511 (2018)
14. Bitansky, N., Nishimaki, R., Passelgue, A., Wichs, D.: From cryptomania to obfustopia through secret-key functional encryption. Cryptology ePrint Archive, Report 2016/558 (2016), http://eprint.iacr.org/2016/558
15. Bitansky, N., Paneth, O., Rosen, A.: On the cryptographic hardness of finding a nash equilibrium. In: FOCS (2015)
16. Bitansky, N., Vaikuntanathan, V.: Indistinguishability obfuscation from functional encryption. In: FOCS. IEEE (2015)
17. Boneh, D., Wu, D.J., Zimmerman, J.: Immunizing multilinear maps against zeroizing attacks. IACR Cryptology ePrint Archive **2014**, 930 (2014), http://eprint.iacr.org/2014/930
18. Brakerski, Z., Gentry, C., Halevi, S., Lepoint, T., Sahai, A., Tibouchi, M.: Cryptanalysis of the quadratic zero-testing of GGH. Cryptology ePrint Archive, Report 2015/845 (2015), http://eprint.iacr.org/
19. Brakerski, Z., Rothblum, G.N.: Virtual black-box obfuscation for all circuits via generic graded encoding. In: TCC. pp. 1–25 (2014)
20. Brzuska, C., Farshim, P., Mittelbach, A.: Indistinguishability obfuscation and uces: The case of computationally unpredictable sources. In: CRYPTO. pp. 188–205 (2014)
21. Cheon, J.H., Han, K., Lee, C., Ryu, H., Stehlé, D.: Cryptanalysis of the multilinear map over the integers. In: EUROCRYPT (2015)
22. Cheon, J.H., Lee, C., Ryu, H.: Cryptanalysis of the new clt multilinear maps. Cryptology ePrint Archive, Report 2015/934 (2015), http://eprint.iacr.org/
23. Cohen, A., Holmgren, J., Nishimaki, R., Vaikuntanathan, V., Wichs, D.: Watermarking cryptographic capabilities. In: STOC (2016)
24. Coron, J., Gentry, C., Halevi, S., Lepoint, T., Maji, H.K., Miles, E., Raykova, M., Sahai, A., Tibouchi, M.: Zeroizing without low-level zeroes: New MMAP attacks and their limitations. In: CRYPTO (2015)
25. Coron, J., Lepoint, T., Tibouchi, M.: Practical multilinear maps over the integers. In: Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part I. pp. 476–493 (2013)
26. Coron, J.S., Lepoint, T., Tibouchi, M.: New multilinear maps over the integers. In: CRYPTO (2015)
27. Döttling, N., Garg, S., Gupta, D., Miao, P., Mukherjee, P.: Obfuscation from low noise multilinear maps. IACR Cryptology ePrint Archive **2016**, 599 (2016)
28. Garg, S., Gentry, C., Halevi, S.: Candidate multilinear maps from ideal lattices. In: Advances in Cryptology - EUROCRYPT 2013, 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings (2013)
29. Garg, S., Gentry, C., Halevi, S., Raykova, M.: Two-round secure MPC from indistinguishability obfuscation. In: Theory of Cryptography - 11th Theory of Cryptography Conference, TCC 2014, San Diego, CA, USA, February 24-26, 2014. Proceedings. pp. 74–94 (2014)
30. Garg, S., Gentry, C., Halevi, S., Raykova, M., Sahai, A., Waters, B.: Candidate indistinguishability obfuscation and functional encryption for all circuits. In: FOCS (2013)

31. Garg, S., Miles, E., Mukherjee, P., Sahai, A., Srinivasan, A., Zhandry, M.: Secure obfuscation in a weak multilinear map model. In: Theory of Cryptography - 14th International Conference, TCC 2016-B, Beijing, China, October 31 - November 3, 2016, Proceedings, Part II. pp. 241–268 (2016)
32. Garg, S., Pandey, O., Srinivasan, A.: Revisiting the cryptographic hardness of finding a nash equilibrium. In: CRYPTO (2016)
33. Gentry, C., Gorbunov, S., Halevi, S.: Graph-induced multilinear maps from lattices. In: TCC. pp. 498–527 (2015)
34. Gentry, C., Sahai, A., Waters, B.: Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In: CRYPTO. pp. 75–92 (2013)
35. Goldreich, O.: Candidate one-way functions based on expander graphs. IACR Cryptology ePrint Archive **2000**, 63 (2000), http://eprint.iacr.org/2000/063
36. Goldwasser, S., Gordon, S.D., Goyal, V., Jain, A., Katz, J., Liu, F., Sahai, A., Shi, E., Zhou, H.: Multi-input functional encryption. In: EUROCRYPT (2014)
37. Goldwasser, S., Rothblum, G.N.: On best-possible obfuscation. In: TCC. pp. 194–213 (2007)
38. Halevi, S.: Graded encoding, variations on a scheme. IACR Cryptology ePrint Archive **2015**, 866 (2015)
39. Hofheinz, D., Jager, T., Khurana, D., Sahai, A., Waters, B., Zhandry, M.: How to generate and use universal samplers. In: ASIACRYPT. pp. 715–744 (2016)
40. Hohenberger, S., Sahai, A., Waters, B.: Replacing a random oracle: Full domain hash from indistinguishability obfuscation. In: EUROCRYPT (2014)
41. Hu, Y., Jia, H.: Cryptanalysis of GGH map. IACR Cryptology ePrint Archive **2015**, 301 (2015)
42. Koppula, V., Lewko, A.B., Waters, B.: Indistinguishability obfuscation for turing machines with unbounded memory. In: STOC (2015)
43. Lin, H.: Indistinguishability obfuscation from constant-degree graded encoding schemes. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 28–57. Springer (2016)
44. Lin, H.: Indistinguishability obfuscation from SXDH on 5-linear maps and locality-5 prgs. In: Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part I. pp. 599–629 (2017)
45. Lin, H.: Indistinguishability obfuscation from sxdh on 5-linear maps and locality-5 prgs. In: CRYPTO. pp. 599–629. Springer (2017)
46. Lin, H., Matt, C.: Pseudo flawed-smudging generators and their application to indistinguishability obfuscation. IACR Cryptology ePrint Archive **2018**, 646 (2018)
47. Lin, H., Tessaro, S.: Indistinguishability obfuscation from bilinear maps and block-wise local prgs. Cryptology ePrint Archive, Report 2017/250 (2017), http://eprint.iacr.org/2017/250
48. Lin, H., Vaikuntanathan, V.: Indistinguishability obfuscation from ddh-like assumptions on constant-degree graded encodings. In: FOCS. pp. 11–20. IEEE (2016)
49. Ma, F., Zhandry, M.: New multilinear maps from CLT13 with provable security against zeroizing attacks. IACR Cryptology ePrint Archive **2017**, 946 (2017)
50. Micciancio, D., Peikert, C.: Trapdoors for lattices: Simpler, tighter, faster, smaller. In: EUROCRYPT. pp. 700–718 (2012)
51. Miles, E., Sahai, A., Zhandry, M.: Annihilation attacks for multilinear maps: Cryptanalysis of indistinguishability obfuscation over GGH13. In: Advances in Cryptology - CRYPTO (2016)

52. Minaud, B., Fouque, P.A.: Cryptanalysis of the new multilinear map over the integers. Cryptology ePrint Archive, Report 2015/941 (2015), http://eprint.iacr.org/
53. Mossel, E., Shpilka, A., Trevisan, L.: On e-biased generators in NC0. In: FOCS. pp. 136–145 (2003)
54. Pass, R., Seth, K., Telang, S.: Indistinguishability obfuscation from semantically-secure multilinear encodings. In: Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part I. pp. 500–517 (2014)
55. Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. In: STOC. pp. 84–93 (2005)
56. Sahai, A., Waters, B.: How to use indistinguishability obfuscation: deniable encryption, and more. In: Shmoys, D.B. (ed.) Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014. pp. 475–484. ACM (2014). https://doi.org/10.1145/2591796.2591825, http://doi.acm.org/10.1145/2591796.2591825

A (Stateful) Semi-Functional Functional Encryption for degree $d + 2$ Polynomials

In this section, we define the notion of Semi-Functional Functional Encryption (referred to as FE_d) for degree $(d + 2)$ polynomials for any constant positive integer d. It is a direct generalisation of semi-functional FE for cubic polynomials in [5].

Function class of interest for FE_d*:* We consider functional encryption scheme for degree $d + 2$ homogenous polynomials over variables over integers $\mathbb{Z}$. Formally, consider a set of functions $\mathcal{F}_{\mathsf{FE}_d,\lambda,n} = \{f : [-\rho, \rho]^n \to \mathbb{Z}\}$ where ρ is some constant. Here n is interpreted as a function of λ. Each $f \in \mathcal{F}_{\mathsf{FE}_d,\lambda,n}$ takes as input $\boldsymbol{x} = (x_1, .., x_n) \in [-\rho, \rho]^n$ and computes a polynomial of the form $\Sigma_{\boldsymbol{I}=(i_1,..,i_{d+2})\in[n]^{d+2}} c_{\boldsymbol{I}} \Pi_{\ell\in[d+2]} \boldsymbol{x}[i_\ell]$ over $\mathbb{Z}$ (where some variables can repeat). The sum of absolute values of the coefficients $\Sigma_{\boldsymbol{I}}|c_{\boldsymbol{I}}| < w(\lambda)$ for some polynomial w independent of n. Such polynomials contain the set of degree $(d+2)$ randomizing polynomials constructed by [47] and thus suffice for $i\mathcal{O}$. Constructing functional encryption for homogenous polynomials suffice to construct functional encryption for all degree $d+2$ polynomials with bounded norm. This is because we can always write any polynomial as a homogeneous polynomial in the same variables and an artificially introduced variable set to 1.

Syntax. Consider the set of functions $\mathcal{F}_{\mathsf{FE}_d} = \mathcal{F}_{\mathsf{FE}_d,\lambda,n}$ as described above. A semi-functional functional encryption scheme FE_d for the class of functions $\mathcal{F}_{\mathsf{FE}_d}$ (described above) consists of the following PPT algorithms:

- **Setup,** $\mathsf{Setup}(1^\lambda, 1^n)$: On input security parameter λ and the length of the message 1^n, it outputs the master secret key MSK.
- **Encryption,** $\mathsf{Enc}(\mathsf{MSK}, \boldsymbol{x})$: On input the encryption key MSK and a vector of integers $\boldsymbol{x} = (x_1, .., x_n) \in [-\rho, \rho]^n$, it outputs ciphertext CT.

- **Key Generation,** $\mathsf{KeyGen}(\mathsf{MSK}, i, f)$: On input the master secret key MSK and an index $i \in [\eta]$ denoting the index of the function in $[\eta]$, function $f \in \mathcal{F}_{\mathsf{FE}_d}$, it outputs a functional key sk_f. Here, η denotes the number of key queries possible. Note that this algorithm is allowed to be stateful.
- **Decryption,** $\mathsf{Dec}(sk_f, \mathsf{CT})$: On input functional key sk_f and a ciphertext CT, it outputs the result *out*.

We define correctness property below.

Correctness. Consider any function $f \in \mathcal{F}_{\mathsf{FE}_d}$, any index $i \in [\eta]$ and any plaintext integer vector $\boldsymbol{x} \in [-\rho, \rho]^n$. Consider the following process:

- $\mathsf{MSK} \leftarrow \mathsf{Setup}(1^\lambda, 1^n)$
- $sk_f \leftarrow \mathsf{KeyGen}(\mathsf{MSK}, i, f)$.
- $\mathsf{CT} \leftarrow \mathsf{Enc}(\mathsf{MSK}, \boldsymbol{x})$

Let $\theta = 1$ if $f(\boldsymbol{x}) \neq 0$, $\theta = 0$ otherwise. The following should hold:

$$\mathsf{Pr}\left[\mathsf{Dec}(sk_f, \mathsf{CT}) = \theta\right] \geq 1 - \mathsf{negl}(\lambda),$$

for some negligible function negl.

Remark 1. We consider a form of semi-functional functional encryption where the decryption algorithm only allows the decryptor to learn if the functional value $f(\boldsymbol{x})$ is 0 or not.

Linear Efficiency: We require that for any message $\boldsymbol{x} \in [-\rho, \rho]^n$ the following holds:

- Let $\mathsf{MSK} \leftarrow \mathsf{Setup}(1^\lambda, 1^n)$.
- Compute $\mathsf{CT} \leftarrow \mathsf{Enc}(\mathsf{MSK}, \boldsymbol{x})$.

The size of the circuit computing CT is less than $npoly(\lambda, \log n)$. Here *poly* is some fixed polynomial independent of n.

A.1 Semi-functional Security

We define the following auxiliary algorithms.

Semi-functional Key Generation, $\mathsf{sfKG}(\mathsf{MSK}, i, f, \theta)$: On input the master secret key MSK, function f, an index i and a value θ, it computes the semi-functional key $sk_{f,\theta}$.

Semi-functional Encryption, $\mathsf{sfEnc}(\mathsf{MSK}, 1^n)$: On input the master encryption key MSK, and the length 1^n, it computes a semi-functional ciphertext $\mathsf{ct}_{\mathsf{sf}}$.

We define two security properties associated with the above auxiliary algorithms. We now define indistinguishability of semi-functional keys property.

Throughout the definition we denote by $\mathcal{S}_\eta$ a set of tuples of dimension η over $\mathcal{F}_{\mathsf{FE}_d}$. Thus $\mathcal{S}_\eta \subseteq \mathcal{F}^\eta_{\mathsf{FE}_d}$.

Definition 9 ($\mathcal{S}_\eta$-Bounded Indistinguishability of Semi-functional Keys). *A Semi-Functional FE scheme for degree $d+2$ polynomials FE_d for a class of functions $\mathcal{F}_{\mathsf{FE}_d} = \{\mathcal{F}_{\mathsf{FE}_3,\lambda,n}\}_{\lambda\in\mathbb{N}}$ is said to satisfy $\mathcal{S}_\eta-$**bounded indistinguishability of semi-functional keys property** if there exists a constant $c > 0$ such that for any sufficiently large $\lambda \in \mathbb{N}$ and any adversary $\mathcal{A}$ of size 2^{λ^c}, the probability that $\mathcal{A}$ succeeds in the following experiment is $2^{-\lambda^c}$.*
$\mathsf{Expt}(1^\lambda, 1^n, \mathbf{b})$:

1. *$\mathcal{A}$ specifies the following:*
   - *It can specify messages $M_j = \{\boldsymbol{x}_i\}_{j\in[q]}$. Here each vector is in $[-\rho,\rho]^n$*
   - *It specifies function queries as follows:*
     - *It specifies $(f_1,\ldots,f_\eta) \in \mathcal{S}_\eta \subseteq \mathcal{F}^\eta_{\mathsf{FE}_d}$.*
     - *It specifies values $\theta_1,\ldots,\theta_\eta$.*
2. *The challenger computes the following:*
   - *$\mathsf{MSK} \leftarrow \mathsf{Setup}(1^\lambda, 1^n)$*
   - *$\mathsf{CT}_i \leftarrow \mathsf{Enc}(\mathsf{MSK}, M_j)$, for every $j \in [q]$.*
   - *If $\mathbf{b} = 0$, compute $sk^*_{f_i} \leftarrow \mathsf{KeyGen}(\mathsf{MSK}, i, f_i)$. Otherwise, compute $sk^*_{f_i} \leftarrow \mathsf{sfKG}(\mathsf{MSK}, i, f_i, \theta_i)$ for all $i \in [\eta]$.*
3. *Challenger sends $\{\mathsf{CT}_i\}_{i\in q}$ and $\{sk^*_{f_i}\}_{i\in[\eta]}$ to $\mathcal{A}$:*
4. *$\mathcal{A}$ outputs b'.*

The success probability of $\mathcal{A}$ is defined to be ε if $\mathcal{A}$ outputs $b' = \mathbf{b}$ with probability $\frac{1}{2} + \varepsilon$.

Definition 10 ($\mathcal{S}_\eta$-Bounded Indistinguishability of Semi-functional Ciphertexts). *For a semi-functional FE scheme FE_d for a class of functions $\mathcal{F}_{\mathsf{FE}_d} = \{\mathcal{F}_{\mathsf{FE}_d,\lambda,n}\}_{\lambda\in\mathbb{N}}$, the $\mathcal{S}_\eta-$**bounded indistinguishability of semi-functional ciphertexts property** is associated with two experiments. The experiments are parameterised with $\mathsf{aux} = (1^\lambda, 1^n, \Gamma, M_i = \{(\boldsymbol{x}_i)\}_{i\in\Gamma}, M^* = (\boldsymbol{x}), f_1, .., f_\eta)$.*
$\underline{\mathsf{Expt}_{\mathsf{aux}}(1^\lambda, 1^n, \mathbf{b})}$:

1. *The challenger sets $\theta_i = f_i(\boldsymbol{x})$ for $i \in [\eta]$. The challenger computes the following:*
2. *Compute $\mathsf{MSK} \leftarrow \mathsf{Setup}(1^\lambda, 1^n)$.*
3. *Compute $sk_{f_k,\theta_k} \leftarrow \mathsf{sfKG}(\mathsf{MSK}, k, f_k, \theta_k)$, for every $k \in [\eta]$.*
4. *$\mathsf{CT}_i \leftarrow \mathsf{Enc}(\mathsf{MSK}, M_i)$, for every $i \in \Gamma$.*
5. *If $\mathbf{b} = 0$, compute $\mathsf{CT}^* \leftarrow \mathsf{Enc}(\mathsf{MSK}, M^*)$.*
6. *If $\mathbf{b} = 1$ compute $\mathsf{CT}^* \leftarrow \mathsf{sfEnc}(\mathsf{MSK}, 1^n)$.*
7. *Output the following:*
   (a) CT_i for $i \in \Gamma$ and CT^*.
   (b) sk_{f_k,θ_k} for $k \in [\eta]$
   (c) M^* and $\{M_i\}_{i\in\Gamma}$
   (d) $f_1,\ldots,f_\eta$

*A semi-functional FE scheme FE_d associated with plaintext space $R = [-\delta,\delta]$ is said to satisfy $\eta-$**indistinguishability of semi–functional ciphertexts property** if the following happens: $\exists c > 0$ such that, $\forall\lambda > \lambda_0$, polynomial $n =$*

$n(\lambda)$, polynomial Γ, for any messages $\{M_i\}_{i\in\Gamma} \in R^n$, $M^* \in R^n$, $(f_1, .., f_\eta) \in \mathcal{S}_\eta$ and any adversary $\mathcal{A}$ of size 2^{λ^c},

$$|\mathsf{Pr}[\mathcal{A}(\mathsf{Expt}_{\mathsf{aux}}(1^\lambda, 1^n, 0) = 1] - \mathsf{Pr}[\mathcal{A}(\mathsf{Expt}_{\mathsf{aux}}(1^\lambda, 1^n, 1)) = 1]| \leq 1 - 2/\lambda + \mathsf{negl}(\lambda)$$

where $\mathsf{aux} = (1^\lambda, 1^n, \Gamma, M_i = \{(\boldsymbol{x}_i)\}_{i\in\Gamma}, M^* = (\boldsymbol{x}), f_1, .., f_\eta)$

If a FE_d scheme satisfies both the above definitions, then it is said to satisfy semi-functional security.

Definition 11 ($\mathcal{S}_\eta-$Bounded Semi-functional Security). *Consider a semi-functional FE scheme for degree $d+2$ polynomials FE_d for a class of functions $\mathcal{F}_{\mathsf{FE}_d}$. We say that FE_d satisfies $\mathcal{S}_\eta-$***bounded semi-functional security** *if it satisfies $\mathcal{S}_\eta-$bounded indistinguishability of semi-functional ciphertexts property (Definition 10) and $\mathcal{S}_\eta-$bounded indistinguishability of semi-functional keys property (Definition 9).*

B Tempered degree $(d+2)$ Encoding

In this section, we describe the notion of a tempered degree $(d+2)$ encoding scheme (TDE for short). The encodings in this scheme are associated with a ring $\mathbb{Z}_{\mathbf{p}}$, for an integer $\mathbf{p} \in \mathbb{Z}^{\geq 0}$ that is fixed by the setup algorithm. The plaintext elements are sampled from the set $R \in \cap[-\delta, \delta]$ for some constant δ. TDE consists of the following polynomial time algorithms:

- **Setup,** $\mathsf{Setup}(1^\lambda, 1^n)$: On input security parameter λ, the number of inputs n, this algorithm outputs public parameters params.
- **Setup-Encode,** $\mathsf{SetupEnc}(\mathsf{params})$: On input params, this algorithm outputs secret encoding parameters sp.
- **Setup-Decode,** $\mathsf{SetupDec}(\mathsf{params})$: On input params, this algorithm outputs (public) decoding parameters $(q_1, ..., q_\eta)$ where $\eta = n^{1+\epsilon}$ described in the security definition.
- **Encode,** $\mathsf{Encode}(\mathsf{sp}, a, \mathsf{ind}, S)$: On input the secret parameter sp, a plaintext element $a \in R$, a level $S = i$ with $i \in \{1, ..., d+2\}$ and an index $\mathsf{ind} \in [n]$, it outputs an encoding $[\mathbf{a}]_{S,\mathsf{ind}}$ with respect to the level S and an index ind. Without loss of generality, this algorithm is deterministic as all the randomness can be chosen during $\mathsf{SetupEnc}$. This encoding satisfies two properties:
  - The encoding $[\mathbf{a}]_{S,\mathsf{ind}} = ([\mathbf{a}]_{S,\mathsf{ind}}.\mathsf{pub}(1), ..., [\mathbf{a}]_{S,\mathsf{ind}}.\mathsf{pub}(d), [\mathbf{a}]_{S,\mathsf{ind}}.\mathsf{priv}(1), [\mathbf{a}]_{S,\mathsf{ind}}.\mathsf{priv}(2))$ consists of d public components $[\mathbf{a}]_{S,\mathsf{ind}}.\mathsf{pub}(i)$ for $i \in [d]$ and two private components $[\mathbf{a}]_{S,\mathsf{ind}}.\mathsf{priv}(1)$ and $[\mathbf{a}]_{S,\mathsf{ind}}.\mathsf{priv}(2)$.
  - $[\mathbf{a}]_{S,\mathsf{ind}}.\mathsf{pub}(i)$ for $i \in [d]$, $[\mathbf{a}]_{S,\mathsf{ind}}.\mathsf{priv}(1)$ and $[\mathbf{a}]_{S,\mathsf{ind}}.\mathsf{priv}(2)$ are vectors over $\mathbb{Z}_{\mathbf{p}}$.
- **Decode,** $\mathsf{Decode}(q, f, \{[\boldsymbol{x}[\mathbf{1}, \mathbf{i}]]_{1,i}\}_{i\in[n]}, ..., \{[\boldsymbol{x}[\mathbf{d+2}, \mathbf{i}]]_{d+2,i}\}_{i\in[n]})$: The decode algorithm takes as input a decoding parameter q, a polynomial $f = \Sigma_{\boldsymbol{I}=(i_1,...,i_{d+2})\in[n]^d} \gamma_{\boldsymbol{I}} \boldsymbol{x}[1, i_1] \cdots \boldsymbol{x}[d+2, i_{d+2}]$ with coefficients $|\gamma_{\boldsymbol{I}}| \leq \delta$. It also takes encodings $\{[\boldsymbol{x}[\mathbf{j}, \mathbf{i}]]_{j,i}\}_{i\in[n], j\in[d+2]}$. It outputs $\mathsf{leak} \in \mathbb{Z}_{\mathbf{p}}$.

Efficiency Properties: Consider the following experiment associated with any $n, \lambda \in \mathbb{N}$, any index $\mathsf{ind} \in [n]$, any level $\ell \in [d+2]$ and any plaintext $x \in [-\delta, \delta]$:

1. $\mathsf{Setup}(1^\lambda, 1^n) \to \mathsf{params}$
2. $\mathsf{SetupEnc}(\mathsf{params}) \to \mathsf{sp}$
3. $\mathsf{Encode}(\mathsf{sp}, x, \mathsf{ind}, \ell) \to [\mathbf{x}]_{\ell,\mathsf{ind}}$

Then we require the circuit size computing $[\mathbf{x}]_{\ell,\mathsf{ind}}$ is less than $poly(\lambda, \log n)$ for some fixed polynomial $poly$.

$(\mathbf{X}, d+2)$*-Multilinear polynomials.* We define the notion of $(\mathbf{X}, d+2)$ multilinear polynomials below.

Definition 12 ($(\mathbf{X}, d+2)$-Multilinear). *Let $\mathbf{X} = \boldsymbol{x}[i, j]$ for $i \in [d+2], j \in [n]$ be $d+2$ sets of variables. A polynomial $p \in \mathbb{Z}_{\mathbf{p}}[\boldsymbol{x}[1,1], \ldots, \boldsymbol{x}[1,n] \ldots, \boldsymbol{x}[d+2,1], \ldots, \boldsymbol{x}[d+2,n]]$ is $(\mathbf{X}, d+2)$-multilinear if every term in the expansion of p is of the form $\tau_{\boldsymbol{I}} \cdot \boldsymbol{x}[1, \boldsymbol{I}[1]] \cdots \boldsymbol{x}[d+2, \boldsymbol{I}[d+2]]$, for some $\boldsymbol{I} \in [n]^{d+2}, \tau_{\boldsymbol{I}} \in \mathbb{Z}_{\mathbf{p}}$.*

degree $d+2$ Evaluation and Correctness: Consider the following experiment associated with any $n, \lambda \in \mathbb{N}$, any index $\mathsf{ind} \in [n]$, any index $\mathsf{ind}_Q \in [\eta]$, any level $\ell \in [d+2]$, any polynomial $f = \Sigma_{\boldsymbol{I}} \gamma_{\boldsymbol{I}} \boldsymbol{x}[1, \boldsymbol{I}[1]] \cdots \boldsymbol{x}[d+2, \boldsymbol{I}[d+2]]$ with $\gamma_{\boldsymbol{I}} \in [-\delta, \delta]$ and any plaintexts $\boldsymbol{x} \in [-\delta, \delta]^{(d+2) \times n}$ for $i \in [n]$:

1. $\mathsf{Setup}(1^\lambda, 1^n) \to \mathsf{params}$
2. $\mathsf{SetupEnc}(\mathsf{params}) \to \mathsf{sp}$
3. $\mathsf{SetupDec}(\mathsf{params}) \to (q_1, \ldots, q_\eta)$
4. $\mathsf{Encode}(\mathsf{sp}, \boldsymbol{x}[j,i], j, i) \to [\boldsymbol{x}[\mathbf{j}, \mathbf{i}]]_{j,i}$ for $i \in [n], j \in [d+2]$
5. Let $q = q_{\mathsf{ind}_Q}$
6. $\mathsf{Decode}(q, f, \{[\boldsymbol{x}[\mathbf{j}, \mathbf{i}]]_{j,i}\}_{j \in [d+2], i \in [n]}) \to \mathsf{leak}$

degree $d+2$ Evaluation: We now describe degree $d+2$ evaluation property. This property states that the $\mathsf{Decode}(q, f, \{[\boldsymbol{x}[\mathbf{j}, \mathbf{i}]]_{j,i}\}_{j \in [d+2], i \in [n]})$ algorithm evaluates an efficiently computable homogeneous degree $d+2$ polynomial $\phi_{q,f}$ which depends on params, f, q, and which is a $(\mathbf{Y}, d+2)$-multilinear polynomial over $\mathbb{Z}_{\mathbf{p}}$ with:

- $\mathbf{Y}[j] = (\{[\boldsymbol{x}[\mathbf{k}, \mathbf{i}]]_{j,i}.\mathsf{pub}(j)\}_{i \in [n], k \in [d+2]})$ for $j \in [d]$.
- $\mathbf{Y}[d+j] = (\{[\boldsymbol{x}[\mathbf{k}, \mathbf{i}]]_{j,i}.\mathsf{priv}(j)\}_{i \in [n], k \in [d+2]})$ for $j \in [2]$.

Correctness: We require that with overwhelming probability over the randomness of the algorithms:

- If $f(\boldsymbol{x}) = 0$, $|\mathsf{leak}| < \mathsf{TDEbound}(\lambda, n)$ for some polynomial $\mathsf{TDEbound}$.
- Otherwise, $|\mathsf{leak}| > \mathsf{TDEbound}(\lambda, n)$.

B.1 Tempered Security

We present the definition of Tempered Security. Let $\mathcal{F}$ be a family of homogenous $(\mathbf{Y}, d+2)$-multilinear δ-bounded polynomials, for some sets of vectors $\mathbf{Y}$ (where $\boldsymbol{Y}$ is of size $(d+2)\times n$). We define $\mathcal{S}_\eta$ to be a subset of η-sized product $\mathcal{F}\times\cdots\times\mathcal{F}$ (also, written as $\mathcal{F}^\eta$).

We first describe the experiments associated with tempered security property. The experiment is associated with a deterministic polynomial time algorithm Sim. It is also parameterised by $\mathsf{aux} = (1^\lambda, 1^n, \boldsymbol{x}, f_1, \ldots, f_\eta)$. Each vector $\boldsymbol{x}$ is in $R^{(d+2)\times n}$ and $f_1, \ldots, f_\eta \in \mathcal{S}_\eta$.

$\underline{\mathsf{Expt}_{\mathsf{aux}}(1^\lambda, 1^n, 0)}$:

1. Challenger performs $\mathsf{Setup}(1^\lambda, 1^n) \to \mathsf{params}$
2. The challenger samples $(q_1, \ldots, q_\eta) \leftarrow \mathsf{SetupDec}(\mathsf{params})$.
3. Challenger performs $\mathsf{SetupEnc}(\mathsf{params}) \to \mathsf{sp}$.
4. Now compute encodings as follows.
   - Compute the encodings, $[\boldsymbol{x}[\mathbf{k}, \mathbf{i}]]_{k,i} \leftarrow \mathsf{Encode}(\mathsf{sp}, \boldsymbol{x}[k, i], k, i)$ for every $k \in [d+2], i \in [n]$.
5. Compute $\mathsf{leak}_j \leftarrow \mathsf{Decode}(q_j, f_j, \{[\boldsymbol{x}[\mathbf{k}, \mathbf{i}]]_{k,i}\}_{k\in[d+2], i\in[n]})$ for $j \in [\eta]$.
6. Output the following:
   (a) Public components of the encodings, $\{[\boldsymbol{x}[\mathbf{k}, \mathbf{i}]]_{k,i}.\mathsf{pub}(j)\}_{k\in[d+2], i\in[n], j\in[d]}$.
   (b) Decoding parameters q_j for $j \in [\eta]$
   (c) Output of decodings, $\{\mathsf{leak}_j\}_{j\in[\eta]}$.

$\underline{\mathsf{Expt}_{\mathsf{aux}}(1^\lambda, 1^n, 1)}$:

1. Challenger performs $\mathsf{Setup}(1^\lambda, 1^n) \to \mathsf{params}$
2. The challenger samples $(q_1, \ldots, q_\eta) \leftarrow \mathsf{SetupDec}(\mathsf{params})$.
3. Challenger performs $\mathsf{SetupEnc}(\mathsf{params}) \to \mathsf{sp}$.
4. Now compute encodings as follows.
   - Compute the encodings, $[\boldsymbol{x}[\mathbf{k}, \mathbf{i}]]_{k,i} \leftarrow \mathsf{Encode}(\mathsf{sp}, 0, k, i)$ for every $k \in [d+2], i \in [n]$.
5. Compute the following for all $j \in [\eta]$:

$$\widehat{\mathsf{leak}_j} \leftarrow \mathsf{Sim}\left(q_j, f_j, \{[\boldsymbol{x}[\mathbf{k}, \mathbf{i}]]_{k,i}\}_{k\in[d+2], i\in[n]}, f_j(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z})\right)$$

   to obtain the simulated outputs.
6. Output the following:
   (a) Public components of the encodings, $\{[\boldsymbol{x}[\mathbf{k}, \mathbf{i}]]_{k,i}.\mathsf{pub}(j)\}_{k\in[d+2], i\in[n], j\in[d]}$.
   (b) Decoding parameters q_j for $j \in [\eta]$
   (c) Output of decodings, $\{\widehat{\mathsf{leak}_j}\}_{j\in[\eta]}$.

Definition 13 (Tempered Security). *A tempered degree $(d+2)$ encoding scheme $\mathsf{TDE} = (\mathsf{Setup}, \mathsf{SetupEnc}, \mathsf{SetupDec}, \mathsf{Encode}, \mathsf{Decode})$ associated with plaintext space $R = [-\delta, \delta]$ is said to satisfy* **Tempered security** *for polynomials (with coefficients over $[-\delta, \delta]$) if there exists an algorithm Sim so that following happens:*

$\exists c > 0$, such that for all large enough security parameter $\lambda \in \mathbb{N}$, and polynomial $n = n(\lambda)$ and any $\boldsymbol{x} \in R^{(d+2)\times n}$, $(f_1, .., f_\eta) \in \mathcal{S}_\eta$ and adversary $\mathcal{A}$ of size 2^{λ^c},

$$|\mathsf{Pr}[\mathcal{A}(\mathsf{Expt}_{\mathsf{aux}}(1^\lambda, 1^n, 0) = 1] - \mathsf{Pr}[\mathcal{A}(\mathsf{Expt}_{\mathsf{aux}}(1^\lambda, 1^n, 1)) = 1]| \leq 1 - 2/\lambda + \mathsf{negl}(\lambda)$$

where $\mathsf{aux} = (1^\lambda, 1^n, \boldsymbol{x}, f_1, \ldots, f_\eta)$ and $\mathsf{negl}(\lambda)$ is some negligible function.

Few remarks are in order:

Remark 2. One can imagine $\mathcal{S}_\eta$ to be an arbitrary subset of $\mathcal{F} \times \cdots \times \mathcal{F}$. However, to pursue our approach, we will set $\mathcal{S}_\eta$ as the $\eta-$sized product of degree $d+2$ polynomials in $n(\lambda)$ variables with the sum of absolute value of coefficients being bounded by some polynomial (in λ) independent of n. As described later, it turns out that this set contains the set of randomizing polynomials constructed by [47], and suffices to get $i\mathcal{O}$.

Remark 3. (On number of query polynomials) In the definition above, an implicit restriction on the number of polynomials (i.e., η polynomials). Indeed, in the instantiation, we only support $\eta = n^{1+\varepsilon}$ for some $0 < \epsilon < 0.5$. This choice of parameters will suffice for our construction of $i\mathcal{O}$. This ε will be set later.

C TDE Construction

C.1 LWE Preliminaries

A full-rank m-dimensional integer lattice $\Lambda \subset \mathbb{Z}^m$ is a discrete additive subgroup whose linear span is $\mathbb{R}^m$. The basis of Λ is a linearly independent set of vectors whose linear combinations are exactly Λ. Every integer lattice is generated as the $\mathbb{Z}$-linear combination of linearly independent vectors $\mathbf{B} = \{\boldsymbol{b}_1, \ldots, \boldsymbol{b}_m\} \subset \mathbb{Z}^m$. For a matrix $\mathbf{A} \in \mathbb{Z}_{\mathbf{p}}^{\mathsf{dim} \times m}$, we define the "$\mathbf{p}$-ary" integer lattices:

$$\Lambda_{\mathbf{p}}^{\perp} = \{\boldsymbol{e} \in \mathbb{Z}^m | \mathbf{A}\boldsymbol{e} = \mathbf{0} \bmod \mathbf{p}\}, \qquad \Lambda_{\mathbf{p}}^{\mathbf{u}} = \{\boldsymbol{e} \in \mathbb{Z}^m | \mathbf{A}\boldsymbol{e} = \boldsymbol{u} \bmod q\}$$

It is obvious that $\Lambda_{\mathbf{p}}^{\boldsymbol{u}}$ is a coset of $\Lambda_{\mathbf{p}}^{\perp}$.

Let Λ be a discrete subset of $\mathbb{Z}^m$. For any vector $\boldsymbol{c} \in \mathbb{R}^m$, and any positive parameter $\sigma \in \mathbb{R}$, let $\rho_{\sigma,\boldsymbol{c}}(\boldsymbol{x}) = \exp(-\pi||\boldsymbol{x} - \boldsymbol{c}||^2/\sigma^2)$ be the Gaussian function on $\mathbb{R}^m$ with center $\boldsymbol{c}$ and parameter σ. Next, we let $\rho_{\sigma,\boldsymbol{c}}(\Lambda) = \sum_{\boldsymbol{x}\in\Lambda} \rho_{\sigma,\boldsymbol{c}}(\boldsymbol{x})$ be the discrete integral of $\rho_{\sigma,\boldsymbol{x}}$ over Λ, and let $\mathcal{D}_{\Lambda,\sigma,\boldsymbol{c}}(\boldsymbol{y}) := \frac{\rho_{\sigma,\boldsymbol{c}}(\boldsymbol{y})}{\rho_{\sigma,\boldsymbol{c}}(\Lambda)}$. We abbreviate this as $\mathcal{D}_{\Lambda,\sigma}$ when $\boldsymbol{c} = \mathbf{0}$. We note that $\mathcal{D}_{Z^m,\sigma}$ is $\sqrt{m}\sigma$-bounded.

Let S^m denote the set of vectors in $\mathbb{R}^m$ whose length is 1. The norm of a matrix $\mathbf{R} \in \mathbb{R}^{m\times m}$ is defined to be $\mathsf{sup}_{\boldsymbol{x}\in S^m}||\mathbf{R}\boldsymbol{x}||$. The LWE problem was introduced by Regev [55], who showed that solving it *on average* is as hard as (quantumly) solving several standard lattice problems *in the worst case*.

Definition 14 (LWE). *For an integer* $\mathbf{p} = \mathbf{p}(\mathsf{dim}) \geq 2$*, and an error distribution* $\chi = \chi(\mathsf{dim})$ *over* $\mathbb{Z}_{\mathbf{p}}$*, the* Learning With Errors problem $\mathsf{LWE}_{di,m,\mathbf{p},\chi}$ *is to distinguish between the following pairs of distributions (e.g. as given by a sampling oracle* $\mathcal{O} \in \{\mathcal{O}_{\boldsymbol{s}}, \mathcal{O}_{\$}\}$*):*

$$\{\mathbf{A}, \boldsymbol{s}^{\mathbf{T}}\mathbf{A} + \boldsymbol{x}^{\mathbf{T}}\} \text{ and } \{\mathbf{A}, \boldsymbol{u}\}$$

where $\mathbf{A} \leftarrow \mathbb{Z}_q^{\mathsf{dim} \times m}$*,* $\boldsymbol{s} \leftarrow \mathbb{Z}_{\mathbf{p}}^{\mathsf{dim}}$*,* $\boldsymbol{u} \leftarrow \mathbb{Z}_{\mathbf{p}}^{m}$*, and* $\boldsymbol{x} \leftarrow \chi^m$*.*

Gadget matrix. The gadget matrix described below is proposed in [50, 2].

Definition 15. *Let* $m = \mathsf{dim} \cdot \lceil \log \mathbf{p} \rceil$*, and define the gadget matrix* $\mathbf{G} = \boldsymbol{g}_2 \otimes \mathbf{I}_{\mathsf{dim}} \in \mathbb{Z}_{\mathbf{p}}^{\mathsf{dim} \times m}$*, where the vector* $\boldsymbol{g}_2 = (1, 2, 4, \ldots, 2^{\lfloor \log \mathbf{p} \rfloor}) \in \mathbb{Z}_{\mathbf{p}}^{\lceil \log \mathbf{p} \rceil}$*. We will also refer to this gadget matrix as "powers-of-two" matrix. We define the inverse function* $\mathbf{G}^{-1} : \mathbb{Z}_{\mathbf{p}}^{\mathsf{dim} \times m} \to \{0,1\}^{m \times m}$ *which expands each entry* $a \in \mathbb{Z}_{\mathbf{p}}$ *of the input matrix into a column of size* $\lceil \log \mathbf{p} \rceil$ *consisting of the bits of binary representations. We have the property that for any matrix* $\mathbf{A} \in \mathbb{Z}_{\mathbf{p}}^{\mathsf{dim} \times m}$*, it holds that* $\mathbf{G} \cdot \mathbf{G}^{-1}(\mathbf{A}) = \mathbf{A}$*.*

C.2 Our TDE construction:

Below we present our TDE construction. This construction is inspired from the homomorphic encryption construction of [34].
Setting the parameters: Set the following parameters:

- $\mathsf{dim} = \lambda^{c_1}$ for some constant $c_1 > 0$. This is the dimension of the secret.
- Let $\mathbf{p} = O(2^{\lambda^{c_2}})$ be a prime and $m = \mathsf{dim}\lceil \log \mathbf{p} \rceil$ is some polynomial, such that $\mathsf{LWE}_{\mathsf{dim},m,\mathbf{p}\chi}$ holds for a distribution χ bounded by a polynomial $B_3(\lambda)$.
- Let $\eta = \ell = n^{1+\epsilon}$ be the stretch of $\mathsf{d}\Delta\mathsf{RG}$ for some constant $\epsilon > 0$.
- $\mathcal{S}_\eta$: We set $\mathcal{S}_\eta$ to be $(\mathcal{F}_{n,B_4})^\eta$. Here, $\mathcal{F}_{n,B_4}$ is the set of homoegenous multilinear degree $d+2$ polynomials with sum of absolute value of coefficients in $[0, B_4]$ for some polynomial $B_4(\lambda)$. This choice turns out to be sufficient to construct $i\mathcal{O}$. Looking ahead, these polynomials will come from the set of degree three randomizing polynomials [47], which satisfy this property. These polynomials can be implemented by our dFE scheme.
- B: The bound B is set to be $m^{d+2}B_3^{d+2}B_4$. This is computed as the maximum norm on the encodings for any function $f \in \mathcal{F}_{n,B_4}$ before smudging with $\mathsf{d}\Delta\mathsf{RG}$ samples.
- $\mathsf{TDEbound} : \mathsf{TDEbound}$ is the maximum norm of the decoded value for any function $f \in \mathcal{F}_{n,B_4}$ which evaluates to 0. It can be upper bounded by $B + poly(\lambda, n)$ for some polynomial $poly(\lambda, n)$. Here, $poly(\lambda, n)$ is the bound on the output of $\mathsf{d}\Delta\mathsf{RG}$, as described in the definition.

We describe a non-commutative product lemma that will be useful to describe our construction. In particular, the function F_{ncp} described in the below lemma will be used in the decode algorithm.

Lemma 13 (Non-commutative Product Lemma). *Suppose we have a vector* $\boldsymbol{a} \in \mathbb{Z}_q^{1\times \mathsf{dim}}$*, matrices* $\mathbf{U} \in \mathbb{Z}_q^{\mathsf{dim}\times m}, \mathbf{V} \in \mathbb{Z}_q^{m\times m}$*. There is a function* $\mathsf{F}_{ncp} : \mathbb{Z}_q^{nm^2\times 1} \times \mathbb{Z}_q^{\mathsf{dim}\times m} \to \mathbb{Z}_q^{1\times m}$ *that given* $\boldsymbol{a}\otimes\mathbf{V}$ *and* $\mathbf{U}$*, computes* $\boldsymbol{a}\mathbf{U}\mathbf{V}$*. That is,* $\mathsf{F}_{ncp}(\boldsymbol{a}\otimes\mathbf{V}, \mathbf{U})$ *outputs* $\boldsymbol{a}\mathbf{U}\mathbf{V}$*. Moreover,* $\mathsf{F}_{ncp}(\boldsymbol{a}\otimes\mathbf{V}, \mathbf{U}) = (\mathsf{q}_1(\boldsymbol{a}\otimes\mathbf{V}, \mathbf{U}), \ldots, \mathsf{q}_m(\boldsymbol{a}\otimes\mathbf{V}, \mathbf{U}))$*, where* q_i *is a quadratic polynomial with every term being a product of an element in* $\boldsymbol{a}\otimes\mathbf{V}$ *and an element in* $\mathbf{U}$*.*

Proof. Let $\boldsymbol{a} = [a_1 \cdots a_{\mathsf{dim}}]$. The $(i,j)^{th}$ element in $\mathbf{U}$ is denoted by $u_{i,j}$, for every $i \in [\mathsf{dim}], j \in [m]$. The $(i,j)^{th}$ element in $\mathbf{V}$ is denoted by $v_{i,j}$.

Observe that the i^{th} element, for every $i \in [m]$, in $\boldsymbol{a}\mathbf{U}$ is denoted by $\sum_{j=1}^{m} a_j u_{ij}$. The i^{th} element in $\boldsymbol{a}\mathbf{U}\mathbf{V}$, for $i \in [m]$, is denoted by $\sum_{k=1}^{m}(\sum_{j=1}^{\mathsf{dim}} a_j u_{kj})\cdot v_{ik}$. The expression $\sum_{k=1}^{m}(\sum_{j=1}^{n} a_j u_{kj})\cdot v_{ik}$ can be rewritten as, $\sum_{k=1}^{m}\sum_{j=1}^{n}(a_j v_{ik})\cdot u_{kj}$. Recall that $\boldsymbol{a}\otimes\mathbf{V}$ is a vector consisting of $a_j v_{ik}$, for every $i \in [\mathsf{dim}], j \in [m], k \in [m]$. Thus, $\sum_{k=1}^{m}\sum_{j=1}^{n}(a_j v_{ik})\cdot u_{kj}$ is a quadratic polynomial, denoted by q_i, with every term being a product of an element in $\boldsymbol{a}\mathbf{V}$ and an element in $\mathbf{U}$. Thus, $\mathsf{q}_i(\boldsymbol{a}\mathbf{V}, \mathbf{U})$ computes the i^{th} element in $\boldsymbol{a}\mathbf{U}\mathbf{V}$, for $i \in [m]$. This completes the proof.

Construction. We describe the scheme TDE below.

- **Setup,** $\mathsf{Setup}(1^\lambda, 1^n)$: On input security parameter λ, 1^n, it sets $\mathsf{params} = (1^\lambda, 1^n, \mathbf{p}, B)$. Function class $\mathcal{S}_\eta$ and parameters B are instantiated later.
- **SetupEncode,** $\mathsf{SetupEnc}(\mathsf{params})$: On input $\mathsf{params} = (1^\lambda, 1^n, \mathbf{p}, B)$, run the following steps:
  1. Sample $\boldsymbol{t} \xleftarrow{\$} \mathbb{F}_{\mathbf{p}}^{\mathsf{dim}\times 1}$ and $\mathbf{C} \xleftarrow{\$} \mathbb{F}_{\mathbf{p}}^{\mathsf{dim}\times m}$.
  2. Set $\boldsymbol{b} = \mathbf{C}^{\mathbf{T}}\boldsymbol{t} + \mathbf{e}^{\mathbf{T}}$, where $\mathbf{e} \leftarrow \chi^m$ with $||e||_\infty \le B_3$.
  3. Set $\mathbf{A} = [\mathbf{C}^{\mathbf{T}}||\boldsymbol{b}]^{\mathbf{T}}$ in $\mathbb{F}_{\mathbf{p}}^{(\mathsf{dim}+1)\times m}$.
  4. Also set $\boldsymbol{s} = (\boldsymbol{t}^{\mathbf{T}}, -1)$ in $\mathbb{F}_{\mathbf{p}}^{1\times(\mathsf{dim}+1)}$
  5. Sample $\mathsf{Seed} \leftarrow \mathsf{d}\Delta\mathsf{RG.SetupSeed}(1^\lambda, 1^n, B)$.
     Without loss of generality assume that $\mathsf{Seed} = (\mathsf{Seed.pub}(1), \ldots, \mathsf{Seed.pub}(d), \mathsf{Seed.priv}(1), \mathsf{Seed.priv}(2))$. Here $\mathsf{Seed.pub}(j) = (\mathsf{Seed.pub}(j,1), \ldots, \mathsf{Seed.pub}(j,n))$ and $\mathsf{Seed.priv}(i) = (\mathsf{Seed.priv}(i,1), \ldots, \mathsf{Seed.priv}(i,n))$ for $i \in [1,2], j \in [d]$ are vectors in $\mathbb{F}_{\mathbf{p}}^n$.
  6. Output $\mathsf{sp} = (\boldsymbol{s}, \mathbf{A}, \mathsf{Seed})$
- **Encode,** $\mathsf{Encode}(\mathsf{sp}, x, \mathsf{ind}, \ell)$: On input $\mathsf{sp} = (\boldsymbol{s}, \mathbf{A}, \mathsf{Seed})$, plaintext $x \in [-\rho, \rho]$, index $\mathsf{ind} \in [n]$ and level $\ell \in [d+2]$, proceed according to the three cases:
  Sample uniformly $\mathbf{R}_{\ell,\mathsf{ind}} \xleftarrow{\$} \{0,1\}^{m\times m}$. Let $\mathbf{G} \in \mathbb{F}_{\mathbf{p}}^{(\mathsf{dim}+1)\times m}$ denote the gadget matrix and let its inverse function be $\mathbf{G}^{-1}(\cdot)$, as given in Definition 15. $\boldsymbol{s}\mathbf{G}\mathbf{e}_\kappa = \lfloor \frac{\mathbf{p}}{2^\kappa} \rfloor$, where $\mathbf{e}_\kappa$ is an indicator vector of dimension m with the κ^{th} position containing 1 and the rest of the elements are zero. κ is chosen so that 2^κ is the smallest power greater than the maximum value of the computation i.e. $n^{d+2}B_3^{d+2}B_4$. Compute $([\mathbf{x}]_{\ell,\mathsf{ind}}.\mathsf{pub}(1), \ldots, [\mathbf{x}]_{\ell,\mathsf{ind}}.\mathsf{pub}(d), , [\mathbf{x}]_{\ell,\mathsf{ind}}.\mathsf{priv}(1), [\mathbf{x}]_{\ell,\mathsf{ind}}.\mathsf{priv}(2))$ as follows.
  Case $\ell = 1$**:**

- Compute $M_{\ell,\mathsf{ind}} = \mathbf{AR}_{\ell,\mathsf{ind}} + x\mathbf{G}$.
- $[\mathbf{x}]_{\ell,\mathsf{ind}}.\mathsf{pub}(1) = (M_{\ell,\mathsf{ind}}, \mathsf{Seed}.\mathsf{pub}(\ell, \mathsf{ind}))$.
- $[\mathbf{x}]_{\ell,\mathsf{ind}}.\mathsf{pub}(j) = (1, \mathsf{Seed}.\mathsf{pub}(j, \mathsf{ind}))$ for $j \in [2, d]$
- $[\mathbf{x}]_{\ell,\mathsf{ind}}.\mathsf{priv}(j) = (1, \mathsf{Seed}.\mathsf{priv}(j, \mathsf{ind}))$ for $j \in [2]$.

Case $\ell \in [2, d]$**:**

- Compute $M_{\ell,\mathsf{ind}} = \mathbf{AR}_{\ell,\mathsf{ind}} + x\mathbf{G}$.
- $[\mathbf{x}]_{\ell,\mathsf{ind}}.\mathsf{pub}(\ell) = (1, (G)^{-1}(M_{\ell,\mathsf{ind}}))$.
- $[\mathbf{x}]_{\ell,\mathsf{ind}}.\mathsf{pub}(j) = 1$ for $j \in [d] \setminus \{\ell\}$
- $[\mathbf{x}]_{\ell,\mathsf{ind}}.\mathsf{priv}(j) = 1$ for $j \in [2]$.

Case $\ell = d + 1$**:**

- Compute $M_{\ell,\mathsf{ind}} = \mathbf{AR}_{\ell,\mathsf{ind}} + x\mathbf{G}$.
- $[\mathbf{x}]_{\ell,\mathsf{ind}}.\mathsf{pub}(j) = (1)$ for $j \in [d]$.
- $[\mathbf{x}]_{\ell,\mathsf{ind}}.\mathsf{priv}(1) = \boldsymbol{s} \otimes \mathbf{G}^{-1}(M_{\ell,\mathsf{ind}})$.
- $[\mathbf{x}]_{\ell,\mathsf{ind}}.\mathsf{priv}(2) = 1$.

Case $\ell = d + 2$**:**

- Compute $M_{\ell,\mathsf{ind}} = \mathbf{AR}_{\ell,\mathsf{ind}} + x\mathbf{G}$.
- $[\mathbf{x}]_{\ell,\mathsf{ind}}.\mathsf{pub}(j) = (1)$ for $j \in [d]$.
- $[\mathbf{x}]_{\ell,\mathsf{ind}}.\mathsf{priv}(1) = 1$.
- $[\mathbf{x}]_{\ell,\mathsf{ind}}.\mathsf{priv}(2) = \mathbf{G}^{-1}(M_{\ell,\mathsf{ind}})$.

We also assume that all these public and private parts of the encodings are padded appropriately with string consisting of zeroes such that their lengths are same. This length is equal to $\ell_{enc} = (\dim \cdot m + m^2 + poly(\lambda)) \log \mathbf{p}$.

Output $([\mathbf{x}]_{\ell,\mathsf{ind}}.\mathsf{pub}(1), \ldots, [\mathbf{x}]_{\ell,\mathsf{ind}}.\mathsf{pub}(d), [\mathbf{x}]_{\ell,\mathsf{ind}}.\mathsf{priv}(1), [\mathbf{x}]_{\ell,\mathsf{ind}}.\mathsf{priv}(2))$.

– **Setup-Decode,** $\mathsf{SetupDec}(\mathsf{params})$: On input $\mathsf{params} = (1^\lambda, 1^n, \mathbf{p}, B)$, generate
$\mathsf{d}\Delta\mathsf{RG}.\mathsf{SetupPoly}(1^\lambda, 1^n, B) \to q_1, \ldots, q_\eta$. Each q_i is a polynomial that takes as input Seed and outputs an integer.

– **Decode,** $\mathsf{Decode}(q, f, \{[\boldsymbol{x}[\mathbf{j}, \mathbf{i}]]_{j,i}\}_{i\in[n], j\in[d+2]})$: Let $f \in \mathcal{S}_\eta = \Sigma_{\boldsymbol{I}=(i_1,..,i_d,j,k)} \gamma_{\boldsymbol{I}} \boldsymbol{x}[1, i_1] \cdots \boldsymbol{x}[d+2, k]$. Decode algorithm works in two step. First it computes an element $u_f \in \mathbb{Z}$ and then it computes $v_q \in \mathbb{Z}$. It outputs $u_f + v_q$. First we describe how u_f is computed. This element is computed monomial by monomial, for every monomial in f. That is compute, $u_{f,\boldsymbol{I}}$, where $\boldsymbol{I} = (i_1, .., i_d, j, k) \in [n]^{d+2}$ for the monomial $\boldsymbol{x}[1, i_1] \cdots \boldsymbol{x}[d+2, k]$ and then output $u_f = \Sigma_{\boldsymbol{I}} \gamma_{\boldsymbol{I}} u_{f,\boldsymbol{I}}$. Now we describe how to compute $u_{f,\boldsymbol{I}}$.

- First observe that for every encoding $[\boldsymbol{x}[\mathbf{j}, \mathbf{i}]]_{j,i}$ for $j \in [d+2], i \in [n]$, if $j \neq 1, d+1, d+2$, there is a ciphertext $\mathbf{M}_{j,i}$ encrypting $\boldsymbol{x}[j, i]$ occurs in the public component j (in its bit-decomposed form) . If $j = d+1$, $\boldsymbol{s} \otimes \mathbf{G}^{-1}(\mathbf{M}_{d+1,i})$ occurs in the private component 1. If $j = d+2$, $\mathbf{M}_{d+2,i}$ in private component 2. If $j = 1$, the bit-decomposed version $\mathbf{M}_{1,i}$ occurs in public component 1.
- Compute $u_{f,\boldsymbol{I}} = \boldsymbol{s}\mathbf{M}_{1,\boldsymbol{I}[1]} \cdot \mathbf{G}^{-1}(\mathbf{M}_{2,\boldsymbol{I}[2]}) \cdots \mathbf{G}^{-1}(\mathbf{M}_{d+2,\boldsymbol{I}[d+2]}) \cdot \mathbf{e}_\kappa$. This step needs to compute a function guaranteed by Lemma 13.
- Compute $v_q = q(\mathsf{Seed})$. Note that both steps can be implemented by homogeneous multilinear degree $d+2$ polynomial which has degree 1 in $\mathsf{pub}(j)$ components for $j \in [d+2]$ and $\mathsf{priv}(j)$ for $j \in [2]$. Thus both these steps are implementable by $(d+2)-$restricted FE.

Output $u_f + v_q$.

We now prove the following properties.

Correctness: Correctness is immediate, following is a sketch. $u_f = e_f + f(\boldsymbol{x})\lfloor \mathbf{p}/2^\kappa \rfloor$ where e_f is some bounded FHE error. If $f(\boldsymbol{x}) = 0$, then u_f is small, otherwise it is larger than $\lfloor \mathbf{p}/2^\kappa \rfloor$. Note that v is always small as it is $\mathsf{d}\Delta\mathsf{RG}$ computation.

degree $d+2$ Evaluation Property. The degree $d+2$ evaluation property can be observed from the description of Decode.

Security. We prove security below.

Theorem 14. *The above scheme satisfies tempered security assuming that $\mathsf{d}\Delta\mathsf{RG}$ is a secure perturbation resilient generator implementable by a $(d+2)$ restricted FE scheme and learning with errors.*

Proof. We first describe the simulator associated with the above scheme.

$\underline{\mathsf{Sim}(q_j, f_j, \{[\boldsymbol{x}[\mathbf{j},\mathbf{i}]]_{j,i}\}_{j\in[d+2],i\in[n]}, f_j(\boldsymbol{x}))}$: On input polynomial q_j, function f_j associated with index $j \in [\eta]$, encodings $\{[\boldsymbol{x}[\mathbf{j},\mathbf{i}]]_{j,i}\}_{j\in[d+2],i\in[n]}$ and output $f_j(\boldsymbol{x})$,

- Using $[\boldsymbol{x}[\mathbf{1},\mathbf{i}]]_{1,i}$ for $i \in [n]$ recover Seed. Let $\mathcal{H}$ denote the associated efficient sampler with the $\mathsf{d}\Delta\mathsf{RG}$. This exists as $\mathsf{d}\Delta\mathsf{RG}$ is a secure perturbation resilient generator. Compute $\mathcal{H}(\mathsf{pp}, \mathsf{Seed}) \to (h_1, ..., h_\eta)$.
- Then output $f_j(\boldsymbol{x})\lfloor \mathbf{p}/2^\kappa \rfloor + h_j$.

We describe the hybrids below. Let $\mathsf{aux} = (1^\lambda, 1^n, \boldsymbol{x}, f_1, ..., f_\eta)$. Vector $\boldsymbol{x}$ is in $R^{(d+2)\times n}$.

Hybrid$_1$: This corresponds to the real experiment. In particular, the output of this hybrid is:

1. Challenger performs $\mathsf{Setup}(1^\lambda, 1^n) \to \mathsf{params}$
2. The challenger samples $(q_1, ..., q_\eta) \leftarrow \mathsf{SetupDec}(\mathsf{params})$.
3. Challenger performs $\mathsf{SetupEnc}(\mathsf{params}) \to \mathsf{sp}$.
4. Now compute encodings as follows.
   - Compute the encodings, $[\boldsymbol{x}[\mathbf{j},\mathbf{i}]]_{j,i} \leftarrow \mathsf{Encode}(\mathsf{sp}, \boldsymbol{x}[j,i], i, j)$ for every $i \in [n], j \in [d+2]$.
5. Compute $\mathsf{leak}_k \leftarrow \mathsf{Decode}(q_k, f_k, \{[\boldsymbol{x}[\mathbf{j},\mathbf{i}]]_{j,i}\}_{j\in[d+2],i\in[n]})$ for $k \in [\eta]$.
6. Output the following:
   (a) Public components of the encodings, $\{[\boldsymbol{x}[\mathbf{j},\mathbf{i}]]_{j,i}.\mathsf{pub}(k)\}_{i\in[n],j\in[d+2],k\in[d]}$.
   (b) Decoding parameters $\{q_j\}$ for $j \in [\eta]$.
   (c) Output of decodings, $\{\mathsf{leak}_j\}_{j\in[\eta]}$.

Hybrid$_2$: In this hybrid, the leakage output by decode is instead generated by the simulator.

1. Challenger performs $\mathsf{Setup}(1^\lambda, 1^n) \rightarrow \mathsf{params}$
2. The challenger samples $(q_1, ..., q_\eta) \leftarrow \mathsf{SetupDec}(\mathsf{params})$.
3. Challenger performs $\mathsf{SetupEnc}(\mathsf{params}) \rightarrow \mathsf{sp}$.
4. Now compute encodings as follows.
   - Compute the encodings, $[\boldsymbol{x}[\mathbf{j}, \mathbf{i}]]_{j,i} \leftarrow \mathsf{Encode}(\mathsf{sp}, \boldsymbol{x}[j, i], i, j)$ for every $i \in [n], j \in [d+2]$.
5. Compute $\{\widehat{\mathsf{leak}_j}\}_{j\in[\eta]} \leftarrow \mathsf{Sim}\left(q_j, f_j, \{[\boldsymbol{x}[\mathbf{j}, \mathbf{i}]]_{j,i}\}_{j\in[d+2],i\in[n]}, f_j(\boldsymbol{x})\right)$.
6. Output the following:
   (a) Public components of the encodings, $\{[\boldsymbol{x}[\mathbf{j}, \mathbf{i}]]_{j,i}.\mathsf{pub}(k)\}_{i\in[n],j\in[d+2],k\in[d]}$.
   (b) Decoding parameters $\{q_j\}$ for $j \in [\eta]$.
   (c) Output of decodings, $\{\widehat{\mathsf{leak}_j}\}_{j\in[\eta]}$.

Claim. Suppose that the $\mathsf{d}\Delta\mathsf{RG}$ assumption is true then for any adversary $\mathcal{A}$ of size at most 2^λ, $|\Pr[\mathcal{A}(\mathbf{Hybrid}_1) = 1] - \Pr[\mathcal{A}(\mathbf{Hybrid}_2)]| \leq 1 - 2/\lambda + \mathsf{negl}(\lambda)$

Proof. The only difference between $\mathbf{Hybrid}_1$ and $\mathbf{Hybrid}_2$ is in how the η number of leakages are generated. In $\mathbf{Hybrid}_1$, the j^{th} leakage is of the form $q_j(\cdot) + a_j$. Note that $a_j = e_{j,fhe} + f_j(\mathbf{x}) \cdot \left\lfloor \frac{\mathbf{p}}{2^\kappa} \right\rfloor$, where $e_{j,fhe}$ is some value in the range $[-B, B]$. In $\mathbf{Hybrid}_2$, the j^{th} leakage is of the form $\widehat{\mathsf{leak}_j} = h_j + f_j(\mathbf{x}) \cdot \left\lfloor \frac{\mathbf{p}}{2^\kappa} \right\rfloor$.

Suppose the output distributions of $\mathbf{Hybrid}_1$ and $\mathbf{Hybrid}_2$ are computationally distinguishable with probability greater than $1 - 2/\lambda + \mathsf{negl}(\lambda)$, we can design an attacker that breaks the $\mathsf{d}\Delta\mathsf{RG}$ assumption as follows. This attacker first generates $(e_{1,fhe}, \ldots, e_{\eta,fhe})$: this is performed by first generating the TDE encodings and then computing $(e_{1,fhe}, \ldots, e_{\eta,fhe})$ as a function of these encodings. The attacker submits this tuple to the challenger of the $3\Delta\mathsf{RG}$. The challenger returns the polynomials $(q_1, \ldots, q_\eta)$ and $(\mathsf{leak}_1, \ldots, \mathsf{leak}_\eta)$. The attacker then submits the degree $(d+2)-$tempered encodings along with $(q_1, \ldots, q_\eta)$ and $(\mathsf{leak}_1 + f_1(\mathbf{x}) \cdot \left\lfloor \frac{\mathbf{p}}{2^\kappa} \right\rfloor, \ldots, \mathsf{leak}_\eta + f_\eta(\mathbf{x}) \cdot \left\lfloor \frac{\mathbf{p}}{2^\kappa} \right\rfloor)$ to the distinguisher (who distinguishes $\mathbf{Hybrid}_1$ and $\mathbf{Hybrid}_2$). The output of the attacker is the same as the output of the distinguisher. Thus, if the distinguisher distinguishes with probability ε then the attacker breaks $\mathsf{d}\Delta\mathsf{RG}$ with probability ε.

$\mathbf{Hybrid}_3$: In this hybrid, generate the encodings as encodings of zeroes. In particular, execute the following operations.

1. Challenger performs $\mathsf{Setup}(1^\lambda, 1^n) \rightarrow \mathsf{params}$
2. The challenger samples $(q_1, ..., q_\eta) \leftarrow \mathsf{SetupDec}(\mathsf{params})$.
3. Challenger performs $\mathsf{SetupEnc}(\mathsf{params}) \rightarrow \mathsf{sp}$.
4. Now compute encodings as follows.
   - Compute the encodings, $[\boldsymbol{x}[\mathbf{j}, \mathbf{i}]]_{j,i} \leftarrow \mathsf{Encode}(\mathsf{sp}, 0, i, j)$ for every $i \in [n], j \in [d+2]$.
5. Compute $\{\widehat{\mathsf{leak}_j}\}_{j\in[\eta]} \leftarrow \mathsf{Sim}\left(q_j, f_j, \{[\boldsymbol{x}[\mathbf{j}, \mathbf{i}]]_{j,i}\}_{j\in[d+2],i\in[n]}, f_j(\boldsymbol{x})\right)$.
6. Output the following:
   (a) Public components of the encodings, $\{[\boldsymbol{x}[\mathbf{j}, \mathbf{i}]]_{j,i}.\mathsf{pub}(k)\}_{i\in[n],j\in[d+2],k\in[d]}$.
   (b) Decoding parameters $\{q_j\}$ for $j \in [\eta]$.

(c) Output of decodings, $\{\widehat{\mathsf{leak}_j}\}_{j\in[\eta]}$.

Claim. Suppose the learning with errors assumption is true, then for any adversary $\mathcal{A}$ of size 2^λ, it holds that $|\Pr[\mathcal{A}(\mathbf{Hybrid}_2)=1]-\Pr[\mathcal{A}(\mathbf{Hybrid}_3)=1]| \leq 2^{-\lambda}$.

Proof. We show the indistinguishability of $\mathbf{Hybrid}_2$ and $\mathbf{Hybrid}_3$ by considering the following sub-hybrids.

$\mathbf{Hybrid}_{2.1}$: The only change between hybrids $\mathbf{Hybrid}_2$ and $\mathbf{Hybrid}_{2.1}$ are in the generation of $\boldsymbol{b}$. In this hybrid, generate $\boldsymbol{b} \xleftarrow{\$} \mathbb{F}_{\mathbf{p}}^m$.

The indistinguishability of hybrids $\mathbf{Hybrid}_2$ and $\mathbf{Hybrid}_{2.1}$ follow from the learning with errors assumption.

$\mathbf{Hybrid}_{2.2}$: The only change between $\mathbf{Hybrid}_{2.1}$ and $\mathbf{Hybrid}_{2.2}$ is in the generation of the public parts of the encodings. Specifically, for every $i \in [n], \ell \in [d+2]$, $\mathbf{M}_{\ell,i} = \mathbf{U}_{\ell,i} + \boldsymbol{x}[j,i]\mathbf{G}$ is generated using a random matrix $\mathbf{U}_{j,i}$.
The statistical indistinguishability of $\mathbf{Hybrid}_{2.1}$ and $\mathbf{Hybrid}_{2.2}$ follows from the extended leftover hash lemma.

$\mathbf{Hybrid}_{2.3}$: The only change between $\mathbf{Hybrid}_{2.2}$ and $\mathbf{Hybrid}_{2.3}$ is in the generation of the public parts of the encodings. Specifically, for every $i \in [n], \ell \in [d+2]$, generate $\mathbf{M}_{\ell,i} = \mathbf{U}_{i,\ell} + 0 \cdot \mathbf{G}$. Generate Seed as before. The output distributions of $\mathbf{Hybrid}_{2.2}$ and $\mathbf{Hybrid}_{2.3}$ are identical.

$\mathbf{Hybrid}_{2.4}$: The only change between $\mathbf{Hybrid}_{2.2}$ and $\mathbf{Hybrid}_{2.3}$ is in the generation of the public parts of the encodings. Specifically, for every $i \in [n], \ell \in [d+2]$, generate $\mathbf{M}_{\ell,i} = \mathbf{A}\mathbf{R}_{\ell,i} + 0\mathbf{G}$.

The statistical indistinguishability of the output distributions of $\mathbf{Hybrid}_{2.3}$ and $\mathbf{Hybrid}_{2.4}$ follows from the extended leftover hash lemma.

Finally, learning with errors assumption implies that the output distributions of $\mathbf{Hybrid}_{2.4}$ and $\mathbf{Hybrid}_3$ are computationally indistinguishable. This concludes the proof.

D Canonical Function Hiding Inner Product FE

We now describe the notion of a canonical function hiding inner product FE proposed by [44]. A canonical function hiding scheme FE scheme consists of the following algorithms:

- $\mathsf{PPSetup}(1^\lambda) \to \mathsf{pp}$. On input the security parameter, $\mathsf{PPSetup}$, outputs parameters pp, which contain description of the groups and the plain text space $\mathbb{Z}_{\mathbf{p}}$.

- $\mathsf{Setup}(\mathsf{pp}, 1^n) \to \mathbf{sk}$. The setup algorithm takes as input the length of vector 1^n and parameters pp and outputs a secret key $\mathbf{sk}$. We assume that pp is always implicitly given as input to this algorithm and the algorithms below (sometimes we omit this for ease of notation).
- $\mathsf{Enc}(\mathbf{sk}, \boldsymbol{x}) \to \mathsf{CT}$. The encryption algorithm takes as input a vector $x \in \mathbb{Z}_{\mathbf{p}}^n$ and outputs a ciphertext CT.
- $\mathsf{KeyGen}(\mathbf{sk}, \boldsymbol{y}) \to \mathbf{sk}_{\boldsymbol{y}}$. The key generation algorithm on input the master secret key $\mathbf{sk}$ and a function vector $y \in \mathbb{Z}_{\mathbf{p}}^n$ and outputs a function key $\mathbf{sk}_{\boldsymbol{y}}$
- $\mathsf{Dec}(1^B, \mathbf{sk}_{\boldsymbol{y}}, \mathsf{CT}) \to m^*$. The decryption algorithm takes as input a ciphertext CT, a function key $\mathbf{sk}_{\boldsymbol{y}}$ and a bound B and it outputs a value m^*. Further, it is run in two steps. First step Dec_0, computes $[\langle \boldsymbol{x}, \boldsymbol{y} \rangle]_T$ (if the keys and ciphertexts were issued for $\boldsymbol{x}$ and $\boldsymbol{y}$) and then the second step, Dec_1, computes its discrete log, if this value lies in $[-B, B]$

We now list the requirements:

B-Correctness: Consider the following process:

1. $\mathsf{PPSetup}(1^\lambda) \to \mathsf{pp}$
2. $\mathsf{Setup}(\mathsf{pp}, 1^n) \to \mathbf{sk}$. Fix any $x, y \in \mathbb{Z}_{\mathbf{p}}^n$
3. $\mathsf{KeyGen}(\mathbf{sk}, \boldsymbol{y}) \to \mathbf{sk}_{\boldsymbol{y}}$.
4. $\mathsf{Enc}(\mathbf{sk}, \boldsymbol{x}) \to \mathsf{CT}$
5. $\mathsf{Dec}(1^B, \mathbf{sk}_{\boldsymbol{y}}, \mathsf{CT}) = \theta$

We require with overwhelming probability the following holds: $\theta = \langle \boldsymbol{x}, \boldsymbol{y} \rangle$ if $\langle \boldsymbol{x}, \boldsymbol{y} \rangle \in [-B, B]$ and $\perp$ otherwise

Linear Efficiency: We require that for any message $(\boldsymbol{x}, \boldsymbol{y}) \in \mathbb{F}_{\mathbf{p}}^n$ the following happens:

- Let $\mathbf{sk} \leftarrow \mathsf{Setup}(1^\lambda, 1^n)$.
- Compute $\mathsf{CT} \leftarrow \mathsf{Enc}(\mathbf{sk}, \boldsymbol{x})$.
- Compute $\mathbf{sk}_{\boldsymbol{y}} \leftarrow \mathsf{KeyGen}(\mathbf{sk}, \boldsymbol{y})$

The size of the circuit computing CT and $\mathbf{sk}_{\boldsymbol{y}}$ is less than $n \log_2 \mathbf{p} \cdot poly'(\lambda) < npoly(\lambda)$. Here $poly$ is some polynomial independent of n.

Canonical Structure: We require the scheme consists of a canonical structure described as follows:

1. $\mathsf{PPSetup}$ runs PPGen (the algorithm used to sample bilinear map parameters) and outputs a bilinear map $(e, G_1, G_2, G_T, g_1, g_2)$ and a plaintext space $\mathbb{Z}_{\mathbf{p}}$ which is the order of G_1, G_2 and G_T.
2. Encryption algorithm encodes the message vector on group G_1.
3. Key generation algorithm encodes the function vector on group G_2.
4. Encryption, setup and key generation algorithm do not use pairing operation at all.

5. The decryption algorithm just computes homomorphically a degree 2 polynomial (namely inner product), on the encodings in the secret key and the secret key (by using pairing e) and then computes discrete log (by doing brute force) on the resulting element in the target group.

We note this structure is satisfied by the construction proposed in [44].

Function hiding security: We say that a secret key IPE scheme cIPE is $\mu-$function hiding if for any stateful p.p.t. adversary $\mathcal{A}$ and sufficently large $\lambda \in \mathbb{N}$ and $n = \lambda^c$ for any constant c the following occurs:

$|\Pr[\mathcal{A}_{\alpha \leftarrow \mathcal{D}_0}(\alpha)] = 1 - \Pr[\mathcal{A}_{\alpha \leftarrow \mathcal{D}_1}(\alpha)]| < \mu(\lambda)$ Where the distributions $\mathcal{D}_0$ and $\mathcal{D}_1$ are generated as follows:

Distribution $\mathcal{D}_b$

1. Run $\mathsf{PPSetup}(1^\lambda) \to \mathsf{pp}$.
2. Run $\mathsf{Setup}(\mathsf{pp}, 1^n) \to (\mathsf{pp}, \mathbf{sk})$.
3. Adversary $\mathcal{A}$ on input pp outputs $(\boldsymbol{x}_i^0, \boldsymbol{x}_i^1)$ and $(\boldsymbol{y}_i^0, \boldsymbol{y}_i^1)$ for $i \in [L]$ for some $L = poly(\lambda)$. Here each vector is in $\mathbb{Z}_{\mathbf{p}}^n$. It is required that $\langle \boldsymbol{x}_i^0, \boldsymbol{y}_j^0 \rangle = \langle \boldsymbol{x}_i^1, \boldsymbol{y}_j^1 \rangle$ for $i, j \in [L]$.
4. Compute $\mathsf{CT}_i = \mathsf{Enc}(\mathbf{sk}, \boldsymbol{x}_i^b)$ for $i \in [L]$ and $\mathbf{sk}_i = \mathsf{KeyGen}(\mathbf{sk}, \boldsymbol{y}_i^b)$ for $i \in [L]$.
5. Output $\{\mathsf{CT}_i, \mathbf{sk}_i\}_{i \in [L]}$.

Theorem 15 (Imported Theorem [44]). *Assuming subexponential SXDH holds relative to* PPGen*, there exists a subexponential canonical function hiding inner product functional encryption scheme.*