Simplifying Constructions and Assumptions for $i\mathcal{O}$

Aayush Jain[*] Huijia Lin[†] Amit Sahai[‡]

December 24, 2019

Abstract

The existence of secure indistinguishability obfuscators ($i\mathcal{O}$) has far-reaching implications, significantly expanding the scope of problems amenable to cryptographic study. A recent line of work [Ananth, Jain, and Sahai, 2018; Aggrawal, 2018; Lin and Matt, 2018; Jain, Lin, Matt, and Sahai, 2019] has developed a new theory for building $i\mathcal{O}$ from simpler building blocks, and represents the state of the art in constructing $i\mathcal{O}$ from succinct and instance-independent assumptions. This line of work has culminated in a construction of $i\mathcal{O}$ from four assumptions, consisting of two standard assumptions, namely sub-exponentially secure LWE and SXDH over bilinear groups, and two other pseudorandomness assumptions: The first assumes weak pseudorandomness properties of generators computable by constant-degree polynomials over the integers, as well as an LWE leakage assumption, introduced by [Jain, Lin, Matt, and Sahai, 2019]. The second assumes the existence of Boolean PRGs with constant block locality [Goldreich 2000, Lin and Tessaro 2017]. In this work, we make the following contributions:

- We completely remove the need to assume a constant-block local PRG. This yields a construction of $i\mathcal{O}$ based on three assumptions of LWE, SXDH and a constant degree perturbation resilient generator [Jain, Lin, Matt, and Sahai, 2019]
- Our construction is arguably simpler and more direct than previous constructions. We construct the notion of special homomorphic encoding (SHE) for all $\mathsf{P/Poly}$ from LWE, by adapting techniques from Predicate Encryption [Gorbunov, Vaikunthanathan and Wee, 2015]. Prior to our work, SHE was only known for the class NC^1, from Ring LWE [Agrawal and Rosen, 2017]. Our new SHE allows our construction of $i\mathcal{O}$ to avoid an intermediate step of bootstrapping via randomized encodings. Indeed, we construct a functional encryption scheme whose ciphertext grows sublinearly only in the output length of the circuits as opposed to its size. This is first such scheme that does not rely on multilinear maps.
- Finally, we investigate a main technical concept facilitating the line of work on $i\mathcal{O}$; namely the notion of *partially hiding functional encryption* introduced by [Ananth, Jain, and Sahai 2018]. The partially hiding functional encryption used in these $i\mathcal{O}$ constructions allows an encryptor to encrypt vectors of the form $\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z} \in \mathbb{Z}_p^n$ and allows any decrptor with a key for function f to learn $\langle f(\boldsymbol{x}), \boldsymbol{y} \otimes \boldsymbol{z} \rangle$. The encryption is allowed to reveal $\boldsymbol{x}$ while keeping $\boldsymbol{y}, \boldsymbol{z}$ hidden. Furthermore, the size of the cipher-text should grow linearly in n.

  We significantly improve the starte of the art for partially hiding functional encryption: Assuming SXDH over bilinear maps, we construct a partially hiding FE scheme where the function f is allowed to be any polynomial sized arithmetic branching program. Prior to our work, the best partially hiding FE only supported the class of constant degree polynomials over $\mathbb{Z}_p$ [Jain, Lin, Matt, and Sahai 2019].

[*]UCLA, aayushjainiitd@gmail.com

[†]UW, rachel@cs.washington.edu

[‡]UCLA, sahai@cs.ucla.edu

1 Introduction

Program obfuscation intuitively seeks to transform a program into an "unintelligible form" while maintaining functionality. In particular, starting with the works of [25, 48], the notion of indistinguishability obfuscation ($i\mathcal{O}$) [16, 29] has turned out to have far-reaching applications, significantly expanding the scope of problems to which cryptography can be applied (see, e.g., [23, 28, 32, 48, 35, 18, 24, 26, 31]). As such, securely constructing $i\mathcal{O}$ is one of the most pressing open problems in cryptography.

A recent line of work [5, 2, 33] has developed a new theory for building $i\mathcal{O}$ from simpler building blocks, and represents the state of the art in constructing $i\mathcal{O}$ from succinct, efficiently falsifiable and instance-independent assumptions. This line of work has culminated [33] in a construction of $i\mathcal{O}$ from four assumptions, consisting of the standard assumptions of sub-exponentially secure LWE and SXDH over bilinear groups, and the existence of two different types of pseudorandomness generators: The first is an arithmetic generator that provides only weak pseudorandomness properties but has low degree, and the second is a Boolean PRG with constant block locality.

This paper aims to further develop this line of work by improving key technical tools, with the aim of minimizing the kinds of new assumptions that are needed to achieve $i\mathcal{O}$. The main consequences of our work are the following:

- We completely eliminate the need for assuming the existence of any kind of locality-limited PRGs.
- We give a construction of a functional encryption scheme (FE) that is arguably simpler and more direct in hindsight. We directly construct functional encryption FE for $\mathsf{P/Poly}$ without going via expensive bootstrapping steps employed in previous works. Such an approach yields much more efficient ciphertexts.
- Finally, we build tools that in our opinion would promote basing $i\mathcal{O}$ from qualitatively better assumptions.

We explain each of these aspects below: This first point is important conceptually because locality – the idea that each output bit of the PRG should only depend on some limited subset of the input bits – goes against our basic intuitions about how pseudorandom generators "should" work. In contrast, the types of arithmetic generators that we use have no locality limitations. We now elaborate further on the newer kinds of pseudorandomness generators proposed in the work of [33], and how our work improves the state of the art.

Smudging (Arithmetic) Noise Generators. We consider generators G that expand a seed $\mathsf{sd} \in \mathbb{Z}_p^n$ (where the modulus p is exponentially large in n) into a polynomially longer output $\boldsymbol{r} \in Z_p^{n^{1.01}}$ such that *i)* $\boldsymbol{r}$ has *small polynomial magnitude* $|\boldsymbol{r}|_\infty = \mathsf{poly}(n)$, and *ii)* is able to *partially* smudge/hide even smaller noises[1]. Such weak randomness generators are formalized in three ways as Perturbation Resilient Generators by [6], Correlated Noise Generators[2] by [2] and Pseudo Flawed-smudging Generator by [39]. For them to be useful in $i\mathcal{O}$ constructions, G needs to have close to minimal degree – "degree 2.5". The seed sd of G is divided into two parts $(\mathsf{sd}_1, \mathsf{sd}_2)$, with sd_1 made public

[1]The fact that the output $\boldsymbol{r}$ is polynomially bounded means that it cannot completely smudge even binary noises.

[2]In the notion of correlated noise generators CNG the outputs completely hide the noises. However, this requirement is impossible to achieve unless the outputs are super-polynomially large. As a result, CNGs are not applicable to our approach.

and sd_2 kept secret. G may have constant degree in the public seed sd_1, but can only have degree 2 in the private seed sd_2 – this is what we mean by "degree 2.5". Current candidates proposed in [33] are based on the hardness properties of constant-degree polynomials over the reals, as well as, an LWE-related assumption (See Section 3.3 for more detail). We stress that these generators impose no constraints on locality; every output component can depend on any number of input components.
Block-Local PRGs are *Boolean* pseudo-random generators mapping n bits to $n^{1.01}$ bits that have the structure property that every output bit depends only on a constant number of blocks of seed bits [27, 40]. There has been a line of works [27, 44, 46, 14, 40, 15] studying the security of candidate (block-)local PRGs. It has been shown that PRGs with block locality 2 do not exist [15, 42]. On the other hand, when the block-locality is a sufficiently large constant, Applebaum and Kachlon [13] based on [11, 12, 9, 10] recently give a local construction whose pseudo-randomness is based on the one-wayness of random local functions.

Comparing with the standard assumptions of LWE and SXDH, perturbation-resilient generators and block-local PRGs are less studied and understood. In particular, the notion of perturbation-resilient generators was only proposed quite recently. Furthermore, the two types of generators are completely different, with different security guarantees, structural properties, and candidates, and are not known to imply each other. Ideally, we would like to eliminate the use of both generators to get the holy grail of basing $i\mathcal{O}$ on standard assumptions.

Our Contribution. In this work, we eliminate the need for local PRGs (while keeping perturbation-resilient generators). We achieve this by developing improved constructions of two key tools – namely *Partially Hiding Functional Encryption and Special Homomorphic Encoding* – used in recent $i\mathcal{O}$ schemes [5, 33, 2]. Our new constructions enable us to build directly Functional Encryption (FE) for all (bounded-depth) polynomial-sized circuits, instead of constant-degree polynomials as in previous works [37, 38, 41, 8, 40, 6, 39, 33]. As a result, we no longer need to bootstrap from FE for constant-degree polynomials to FE for $\mathsf{P/Poly}$ which uses local PRGs. In addition, we also obtain a more efficient FE scheme for $\mathsf{P/Poly}$. Such an approach also leads to much shorter ciphertexts, where the length of the ciphertexts grow sublinearly in just the length of the output of the circuits as opposed to their size. This is the first such scheme that does not rely on mulitilinear maps. Below, we describe our results more formally.
Recall that functional encryption is an advanced form of encryption that allows one to generate a partial decryption key fsk_f associated with a function f mapping length n inputs to length m outputs, such that, when decrypting a ciphertext $\mathsf{fct}(\boldsymbol{x})$ encrypting $\boldsymbol{x}$ using this key, only the output $\boldsymbol{y} = f(\boldsymbol{x})$ is revealed. Our functional encryption scheme for $\mathsf{P/Poly}$ enjoys strong asymptotic efficiency properties, and is based on the following assumptions.

Theorem 1. *Assume*

1. *Subexponentially secure LWE,*
2. *SXDH over bilinear groups, and*
3. *Existence of Perturbation-Resilient Generators.*

There exists a functional encryption scheme for all polynomially-sized circuits, whose ciphertext size is $\mathsf{poly}(\lambda, n, d)\ell_{out}^{1-\epsilon}$*, polynomial in the security parameter* λ*, input length* n*, and depth* d *of* *computation, and sublinear in the output length ℓ_{out} of the computation. As a corollary, there is a sublinearly compact functional encryption scheme for all circuits of depth λ.*

By combining this theorem with the FE-to-$i\mathcal{O}$ transformation of [7, 20, 17, 34], we immediately obtain $i\mathcal{O}$. In particular, we show:

Theorem 2. *Assume*

1. *subexponentially secure LWE,*
2. *subexponentially secure SXDH over bilinear groups, and*
3. *the existence of subexponentially secure Perturbation-Resilient Generators.*

There exists an $i\mathcal{O}$ for $\mathsf{P/Poly}$.

Along the way, we study the notions of Partially Hiding Functional Encryption and Special Homomorphic Encoding. Partially Hiding Functional Encryption (PHFE), similar to the notion of partially hiding predicate encryption in [30], was introduced by [6] and further developed by [39, 33]. PHFE is a restricted form of functional encryption where a part of the message to be encrypted is not hidden by the encryption. Somewhat more formally, we consider messages of the form $\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z} \in \mathbb{Z}_p^{n\times 3}$, where $\boldsymbol{x}$ is not hidden by the PHFE encryption algorithm. Furthermore, in our work, we consider functions f of the form $f(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}) = \sum_{i,j} f_{i,j}(\boldsymbol{x}) \cdot y_i \cdot z_j$ where each $f_{i,j}$ is a polynomial sized arithmetic branching program. A PHFE scheme allows the holder of a function key for such a function f to only learn $\langle f(\boldsymbol{x}), \boldsymbol{y} \otimes \boldsymbol{z}\rangle$ when given an encryption of $(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z})$. We show how to construct such an PHFE scheme assuming only SXDH over bilinear groups.

Theorem 3. *Assume the SXDH assumption over bilinear groups. There is a* PHFE *scheme for any polynomial sized arithmetic branching programs on the public input and degree 2 on the private input.*

This significantly generalizes the previous best PHFE construction under standard assumptions due to [33], where each $f_{i,j}$ was restricted to be a constant-degree polynomial. This is the key technical tool that enables instantiating $i\mathcal{O}$ from a wide variety of smudging noise generators. Our results allow to instantiate $i\mathcal{O}$ from any smudging noise generator such as a $\Delta\mathsf{RG}$ that admits an NC^1 computation in the public part and degree-2 computation in the private part of the seed. Another related work is by Wee [49], where the author constructed a partially hiding *predicate* encryption scheme for functions that compute an arithmetic branching program on public attributes, followed by an inner product predicate on private attributes. This construction can be viewed as a generalization of attribute based encryption and inner product predicate encryption, whereas our construction generalizes attribute based encryption and functional encryption for degree 2 polynomials. Another difference is that the two constructions use very different techniques.

Special Homomorphic Encoding (SHE) introduced by [4] and further developed by [2, 39] is similar to homomorphic encryption, and has the following interface: The setup algorithm samples public parameters pp; to encode a vector $\boldsymbol{x}$, the encoding algorithm uses the public parameters pp as well as a randomly sampled one-time secret hsk, producing encoding $\mathsf{hct}(\boldsymbol{x})$; to homomorphically evaluate a function f, the evaluator operates on $\mathsf{hct}(\boldsymbol{x})$ and pp independently to obtain hct_f and pp_f. The most important part is that decryption has the following special form:

Work	**Assumptions**	$\lvert\mathsf{CT}\rvert$
[5]	LWE+SXDH+ 3-block local PRGs+ degree 3 $\Delta\mathsf{RG}$/ PFG	$s^{1-\epsilon} \cdot \mathsf{poly}(\lambda)$
[2]	Candidate Noisy Linear FE[3]+ Ring LWE	$s^{1-\epsilon} \cdot \mathsf{poly}(\lambda)$
[2] (using [6]) [4]	LWE+SXDH+2-block local PRGs + degree-2 CNG	$s^{1-\epsilon} \cdot \mathsf{poly}(\lambda)$
[33]	LWE+SXDH+ $O(1)-$block local PRGs+ $O(1)-$degree $\Delta\mathsf{RG}$	$s^{1-\epsilon} \cdot \mathsf{poly}(\lambda)$
This work	LWE+SXDH+ $O(1)-$degree $\Delta\mathsf{RG}$	$\ell_{out}^{1-\epsilon} \cdot \mathsf{poly}(\lambda)$

Table 1: Summary of assumptions and efficiency features in related works to construct sublinear Functional Encryption. Above $\Delta\mathsf{RG}$, PFG and CNG are smudging arithmetic noise generators proposed in the works of [6, 39] and [2] respectively. Above in the size comparison, $1 > \epsilon > 0$ is an arbitrary constant. For efficiency properties we consider size of ciphertexts for FE schemes for all circuits of depth λ, size s and output length ℓ_{out}.

$$\mathsf{hct}_f - \langle \mathsf{pp}_f, \mathsf{hsk} \rangle = f(\boldsymbol{x})\lfloor p/2 \rceil + e_f, \tag{1}$$

where $|e_f| \le \mathsf{poly}(\lambda)$. Note that the computation $\langle \mathsf{pp}_f, \mathsf{hsk} \rangle$ on the secret hsk is linear and depends only on pp_f. Therefore, $\mathsf{hsk}_f = \langle \mathsf{pp}_f, \mathsf{hsk} \rangle$ resembles a functional ecryption key, in the sense that it is generated independent of $\mathsf{hct}(\boldsymbol{x})$ and reveals only the perturbed output $f(\boldsymbol{x})\lfloor p/2 \rceil + e_f$. The only drawback is that the noise e_f is sensitive and cannot be revealed. This feature is leveraged in previous FE constructions.

In this work, we construct SHE supporting homomorphic evaluation of all polynomial-sized computations from just LWE. Previous constructions based on LWE can only handle constant degree computations [4, 39], and the one based on RLWE handles NC^1 computation [4].

Theorem 4. *Assume the LWE assumption. There is a* SHE *scheme for* $\mathsf{P/Poly}$ *whose encoding size is* $\mathsf{poly}(\lambda, n, d)$*, polynomial in the security parameter* λ*, message length* n*, and depth* d *of the computation.*

In Table 1, we compare assumptions and efficiency features of various constructions in this line of work.

2 Technical Overview

We first give overviews of our new constructions of PHFE and SHE, and then explain their roles in the $i\mathcal{O}$ construction.

2.1 Our (NC^1, deg-2) PHFE

We construct 1-key PHFE with fully compact ciphertext of size linear in the input length n, for functions $F(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z})$ of the following form, from the SXDH assumption on asymmetric bilinear maps.

[3]We call this a candidate because its security is assumed as-is and not proven based on any succinct or instance independent assumption.

[4]This result requires [2] to rely on the FE hardness amplification theorem of [6].

F maps three vectors $\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z} \in \mathbb{Z}_p^n$ to a (potentially longer) output vector in $\mathbb{Z}_p^m$ (our construction can handle any (polynomial) unbounded m), where each output element is computed by a function $f = F_k$ for $k \in [m]$ as follows:

$$f(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}) = \langle f^0, f^1(\boldsymbol{x}) f^2(\boldsymbol{x}) \dots f^d(\boldsymbol{x}) f^{d+1}(\boldsymbol{y} \otimes \boldsymbol{z}) \rangle \ , \tag{2}$$

where f^0, all $f^i(\boldsymbol{x})$, and $f^{d+1}(\boldsymbol{y} \otimes \boldsymbol{z})$ are matrices of dimension $N \times N$ where $N = \mathsf{poly}(n)$, all f^i functions are linear, and all computation is over $\mathbb{Z}_p$. We observe in Section 6 that such functions can indeed express computation such as $L(g(\boldsymbol{x}), \boldsymbol{y} \otimes \boldsymbol{z})$ with a g computable in Boolean NC^1 and a bilinear L.

Previous constructions of PHFE for constant degree public computation [6, 39, 33] follow a common paradigm: To hide $\boldsymbol{y}||\boldsymbol{z}$, these schemes represent the computation as a sum of monomials $f(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}) = \sum_I c_I \boldsymbol{x}_{I_1} \dots \boldsymbol{x}_{I_d} \boldsymbol{y}_{I_{d+1}} \boldsymbol{z}_{I_{d+2}}$, where $\boldsymbol{c} = \{c_I\}_I$ is the the list of public coefficients, and decryption computes the list of all monomials, hidden by a pseudorandom one-time-pad $\boldsymbol{pd}$, in the exponent of the target group, denoted as $[\Pi] = [\otimes_{i=[d]} \boldsymbol{x} \otimes \boldsymbol{y} \otimes \boldsymbol{z} + \boldsymbol{pd}]$. To reveal the output, decryption also produces $[\pi] = [< \boldsymbol{c}, \boldsymbol{pd} >]$, which ensures that the output $[f(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z})]$ can be derived as follows (and then extracted via brute force discrete logarithm).

$$[f(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z})] = [\langle \boldsymbol{c},\ \otimes_{i=[d]} \boldsymbol{x} \otimes \boldsymbol{y} \otimes \boldsymbol{z} \rangle] = \langle \boldsymbol{c}, [\Pi] \rangle - [\pi] \tag{3}$$

The use of pseudorandom one-time-pad $[\Pi]$ ensures that only $[f(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z})]$ is revealed. Unfortunately, this approach cannot extend beyond constant-degree computations as the number of monomials grows exponentially with the degree d.

To handle polynomial degree computations, we must follow the steps of the computation and hide the intermediate states produced by each step. For the functions of interests, we view $\mathsf{st}^0 = f^{d+1}(\boldsymbol{y} \otimes \boldsymbol{z})$ as the initial state and the i'th state is

$$\forall\ i \in [d],\ \mathsf{st}^i = \left(\prod_{j=d-i+1}^{d} f^j(\boldsymbol{x}) \right) f^{d+1}(\boldsymbol{y} \otimes \boldsymbol{z}) \tag{4}$$

The transition from state i to $i+1$ simply involves multiplication by the matrix $f^{d-i}(\boldsymbol{x})$, $\mathsf{st}^{i+1} = f^{d-i}(\boldsymbol{x})\mathsf{st}^i$. To hide $\boldsymbol{y}, \boldsymbol{z}$, we want decryption to produce every state $[\mathsf{st}^i + \boldsymbol{pd}^i]$, hidden by a pseudorandom one-time pad $\boldsymbol{pd}^i$. From the last $[\mathsf{st}^d + \boldsymbol{pd}^d]$ the output and only the output can be obtained by also revealing $[\langle f^0, \boldsymbol{pd}^d \rangle]$. Next, we show how to enable computing $[\mathsf{st}^i + \boldsymbol{pd}^i]$ inductively, and develop our PHFE along the way.

In the base case of $i = 0$, we want to compute $[f^{d+1}(\boldsymbol{y} \otimes \boldsymbol{z}) + \boldsymbol{pd}^0]$. Note first that since $\boldsymbol{pd}^0$ has the same $N \times N$ dimension as f^{d+1}, it cannot be hardcoded in the ciphertext of PHFE, as otherwise the ciphertext would not be compact. We will set $\boldsymbol{pd}^0 = \rho \mathbf{r}^0$, where ρ is a scalar and sampled at encryption time, and $\mathbf{r}^0$ is of $N \times N$ dimension and sampled at key generation time. When computed in the exponent $[\boldsymbol{pd}^0]$ is pseudorandom, by the SXDH assumption, even when the same $\mathbf{r}^0$ is reused across multiple ciphertexts with different ρ's. Similarly, we set for every $i \in [d]$, $\boldsymbol{pd}^i = \rho \cdot \mathbf{r}^i$.

We observe that $[f^{d+1}(\boldsymbol{y} \otimes \boldsymbol{z}) + \rho \cdot \mathbf{r}^0]$ can be computed using techniques developed in the deg-2 FE construction of [38], using function hiding inner product encryption (IPE). IPE is FE for computing the inner product function: It enables generating a set of secret keys $\{\mathsf{IPEsk}(\boldsymbol{v}_i)\}$

and a set of ciphertexts $\{\mathsf{IPEct}(\boldsymbol{u}_j)\}$, from which the cross inner products $\{\langle \boldsymbol{v}_i, \boldsymbol{u}_j\rangle\}$ are revealed. The function hiding security is stronger than standard indistinguishability based FE security in the sense that it hides not only the vectors encrypted in the ciphertexts, but also vectors encoded in the secret keys. For convenience of notation, below we denote by $\mathsf{IPEsk}^t(\boldsymbol{v})$ and $\mathsf{IPEct}^t(\boldsymbol{u})$ to mean IPE secret keys and ciphertexts generated using the same master secret key IPEmsk^t.

Using function hiding IPE, we enable computing $[f^{d+1}(\boldsymbol{y} \otimes \boldsymbol{z}) + \rho\mathbf{r}^0]$ as follows. Include in the PHFE ciphertext n IPE ciphertexts and secret keys encoding respectively vectors $(y_j||\rho u_j||\mathbf{0})$ and $(z_k||w_k||\mathbf{0})$ for all $i, j \in [n]$, where every u_j, w_k are random scalars sampled at Setup time and kept secret in the master secret key. (The tailing zeros in the vectors do not affect the correctness but are important for the security proof as they provide space for hardcoding intermediate computation results; we omit details in this overview.) Decrypting these IPE ciphertexts and secret keyes reveals $[\boldsymbol{y} \otimes \boldsymbol{z} + \rho \boldsymbol{u} \otimes \boldsymbol{w}]$. From here, to enable computing $[f^{d+1}(\boldsymbol{y} \otimes \boldsymbol{z}) + \rho\mathbf{r}^0]$, we just need to additionally enable computing $[\rho\mathbf{r}^0 - \rho f^{d+1}(\boldsymbol{u} \otimes \boldsymbol{w})]$. The latter can be done by including in the PHFE ciphertext a single IPE ciphertext $\mathsf{IPEct}^2(\rho||0)$ and in the PHFE secret key a set of $N = \mathsf{poly}(n)$ IPE secret keys $\{\mathsf{IPEsk}^2((r^0_l - f^{d+1}_l(\boldsymbol{u} \otimes \boldsymbol{w}))||0)\}_l$, the l'th for computing the l'th element $[\rho r^0_l - \rho f^{d+1}_l(\boldsymbol{u} \times \boldsymbol{w})]$. We summarize the idea below.

$$\begin{aligned} \text{In } \mathsf{pfmsk} \;&:\; \boldsymbol{u}, \boldsymbol{w} \leftarrow \mathbb{Z}_p^n,\ \mathsf{IPEmsk}^2 \\ \text{In } \mathsf{pfsk}(f) \;&:\; \forall l \in [N \times N]\ ,\ \mathsf{IPEsk}^2((r^0_l - f^{d+1}_l(\boldsymbol{u} \otimes \boldsymbol{w}))||0) \\ \text{In } \mathsf{pfct}(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}) \;&:\; \forall j, k \in [n]\ ,\ \mathsf{IPEct}^1(y_j||\rho u_j||\mathbf{0}),\ \ \mathsf{IPEsk}^1(z_k||w_k||\mathbf{0}), \mathsf{IPEct}^2(\rho||0) \end{aligned}$$

In $\mathsf{pfDec}(\mathsf{pfsk}, \mathsf{pfct})$:

$$[\boldsymbol{y} \otimes \boldsymbol{z} + \rho \boldsymbol{u} \otimes \boldsymbol{w}] = \left\{\mathsf{IPEDec}\left(\mathsf{IPEsk}^1(z_k||w_k||\mathbf{0}),\ \mathsf{IPEct}^1(y_j||\rho u_j||\mathbf{0})\right)\right\}_{j,k}$$

$$[\rho\mathbf{r}^0 - \rho f^{d+1}(\boldsymbol{u} \otimes \boldsymbol{w})] = \left\{\mathsf{IPEDec}(\mathsf{IPEsk}^2((r^0_l - f^{d+1}_l(\boldsymbol{u} \otimes \boldsymbol{w}))||0),\ \mathsf{IPEct}^2(\rho||0))\right\}_{l \in [N \times N]}$$

$$f^{d+1}\left([\boldsymbol{y} \otimes \boldsymbol{z} + \rho \boldsymbol{u} \otimes \boldsymbol{w}]\right) + [\rho\mathbf{r}^0 - \rho f^{d+1}(\boldsymbol{u} \otimes \boldsymbol{w})] = [\rho\mathbf{r}^0 + f^{d+1}(\boldsymbol{y} \otimes \boldsymbol{z})]$$

Note that in the last line, we can compute f^{d+1} over elements in the exponent thanks to the linearality of f^{d+1} and the target group. Using proof techniques similar to those used in in [38, 33], we can show that only $[f^{d+1}(\boldsymbol{y} \otimes \boldsymbol{z}) + \rho\mathbf{r}^0]$ is revealed.

In the induction case from i to $i-1$ suppose the PHFE scheme enables computing $[\mathsf{st}^i + \rho\mathbf{r}^i]$, and we want to compute $[\mathsf{st}^{i+1} + \rho\mathbf{r}^{i+1}]$. Since transitioning from st^i to st^{i+1} merely involves multiplication by $f^{d-i}(\boldsymbol{x})$, knowing the public $\boldsymbol{x}$ one can already compute from the padded i'th state $[f^{d-i}(\boldsymbol{x})\mathsf{st}^i + f^{d-i}(\boldsymbol{x})\rho\mathbf{r}^i] = [\mathsf{st}^{i+1} + f^{d-i}(\boldsymbol{x})\rho\mathbf{r}^i]$. Hence, towards computing the padded $i+1$'th state, we just need to additionally enable computing $[\Pi] = [\rho\mathbf{r}^{i+1} - f^{d-i}(\boldsymbol{x})\rho\mathbf{r}^i]$. Due to the linearity of f^{d-i}, every element $l \in [N \times N]$ in the matrix Π is linear in $\rho||\rho\boldsymbol{x}$, and hence can be rewritten as an inner product $\langle \boldsymbol{v}^{d-i}_l,\ (\rho||\rho\boldsymbol{x})\rangle$ between $\rho||\rho\boldsymbol{x}$ and a vector $\boldsymbol{v}^{d-i}_l = \boldsymbol{v}^{d-i}_l(\mathbf{r}^{i+1}, \mathbf{r}^i, f^{d-i})$ determined by the vectors $\mathbf{r}^{i+1}$, $\mathbf{r}^i$, and the function f^{d-i}. Observe that $\boldsymbol{v}^{d-i}_l$ is known at the key generation time. Thus, to enable computing $[\Pi]$, we include in the PHFE ciphertext a single IPE ciphertext $\mathsf{IPEct}^3(\rho||\rho\boldsymbol{x}||0)$, and in the PHFE secret key a set of $N \times N$ IPE secret keys $\mathsf{IPEsk}^3(\boldsymbol{v}^{d-i}_l||0)$, whose decryption produces $[\Pi]$.

$$\begin{aligned}
\text{In } \mathsf{pfmsk} &: & & \mathsf{IPEmsk}^3 \\
\text{In } \mathsf{pfsk}(f) &: & & \forall l \in [N \times N]\ ,\ \mathsf{IPEsk}^3((\boldsymbol{v}_l^{d-i})||0) \\
\text{In } \mathsf{pfct}(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}) &: \quad \boldsymbol{x} & & \mathsf{IPEct}^3(\rho||\rho\boldsymbol{x})
\end{aligned}$$

We emphasize that though each induction step j adds new IPE secret keys $\{\mathsf{IPEsk}^3((\boldsymbol{v}_l^{d-j})||0)\}_l$ to the PHFE secret key, whose size thus scales linearly with the degree d of the public computation, the ciphertext contains only a single $\mathsf{IPEct}^3(\rho||\rho\boldsymbol{x})$ for all induction steps. Therefore, adding the content in the ciphertext for the base case, the overall ciphertext size is linear in the input vector length n. This concludes the overview of our PHFE construction; see Section 6 for the formal construction and security proof.

2.2 Our Special Homomorphic Encoding for P/Poly

We construct a SHE for P/Poly with the following simple interface: A setup algorithm samples a public parameter $\mathsf{pp} \leftarrow \mathsf{SHESetup}(1^\lambda)$ which is shared for all encodings; an encoding algorithm encodes an input $\mathsf{hct}(\boldsymbol{x}) \leftarrow \mathsf{SHEEnc}(\mathsf{pp}, \mathsf{hsk}, \boldsymbol{x})$ using a secret key hsk. Importantly, the homomorphic evaluation algorithm $\mathsf{SHEEval}$ has two subroutines, and admits a decryption equation as in (5) and a polynomially-bounded decryption error (6).

$$\begin{aligned}
\mathsf{SHEEval}(\mathsf{pp}, f, \mathsf{hct}(\boldsymbol{x})) &: \quad \mathsf{hct}_f = \mathsf{SHEEvalCT}(\mathsf{pp}, f, \mathsf{hct}(\boldsymbol{x})) \quad \mathsf{pp}_f = \mathsf{SHEEvalPK}(\mathsf{pp}, f) \\
\mathsf{HDec}(\mathsf{hct}_f, \mathsf{pp}_f, \mathsf{hsk}) &: \quad \mathsf{hct}_f - \langle \mathsf{pp}_f, \mathsf{hsk} \rangle = f(\boldsymbol{x})\lfloor p/2 \rceil + e_f \pmod p \qquad (5) \\
& \quad\ |e_f| = \mathsf{poly}(\lambda) \qquad (6)
\end{aligned}$$

The most important feature of SHE is that the computation $\mathsf{hsk}_f = \langle \mathsf{pp}_f, \mathsf{hsk} \rangle$ on the secret key hsk is linear and depends only on pp_f. The produced hsk_f is almost like a functional decryption key, in the sense that it only reveals the perturbed output $f(\boldsymbol{x})\lfloor p/2 \rceil + e_f$. However, the noise e_f is sensitive and must be hidden.

Previous works [4, 39] constructed SHE by modifying the homomorphic encryption schemes of [22, 21] in order to satisfy the special decryption equation. These constructions are recursive and quite complex, and the overhead due to recursion prevents them from supporting computations beyond NC^1. In this work, instead of starting from existing homomorphic encryption, we start with predicate encryption. Our idea is given that the decryption equation of SHE has a structure that resembles functional encryption, perhaps known predicate encryption schemes (which are functional encryption schemes with one-sided security) can give us hints on how to construct SHE.

Indeed, inspired by the predicate encryption scheme of [30], we immediately obtain an encoding scheme for P/Poly satisfying the special decryption equation (5), which however has exponentially large decryption noises. We briefly describe the encoding below:

- The setup algorithm $\mathsf{SHESetup}(1^\lambda)$ samples a collection of random LWE matrices $\boldsymbol{A}_i \leftarrow \mathbb{Z}_p^{n \times m}$, which are the public parameters $\mathsf{pp} = \{\boldsymbol{A}_i\}$.
- An encoding $\mathsf{hct}(\boldsymbol{x})$ contains LWE samples of form $\boldsymbol{c}_i = \boldsymbol{s}^T \boldsymbol{A}_i + \hat{\boldsymbol{x}}_i \boldsymbol{G} + \boldsymbol{e}_i$, where $\boldsymbol{s} \leftarrow \chi^n$ is a (one-time) LWE secret drawn from the noise distribution χ, $\boldsymbol{G} \in \mathbb{Z}_p^{n \times m}$ is the gadget matrix, and $\boldsymbol{e}_i \leftarrow \chi^m$ is LWE noises. In addition, $\hat{\boldsymbol{x}}_i$ is the i'th bit of a homomorphic encryption ciphertext of $\boldsymbol{x}$.

- The predicate encryption scheme of [30] provides a way to homomorphically evaluate f on $\{\boldsymbol{c}_i, \boldsymbol{A}_i\}$ to obtain $\boldsymbol{c}_f$, and seperately on $\{\boldsymbol{A}_i\}$ to obtain $\boldsymbol{A}_f$, such that, the first coordinate $\boldsymbol{c}_{f,1}$ of $\boldsymbol{c}_f$ and the first column $\boldsymbol{A}_{f,1}$ of $\boldsymbol{A}_f$ satisfy the special decryption equation.

$$\boldsymbol{c}_{f,1} - \boldsymbol{s}^T \boldsymbol{A}_{f,1} = f(\boldsymbol{x})\lfloor p/2 \rceil + e_f \mod p$$

Next, we turn to reducing the norm of the decryption error, by applying the rounding (or modulus switch) technique in the HE literature [21]. Namely, to reduce the error norm by a factor of p/q for a $q < p$, we multiply $\boldsymbol{c}_{f,1}$ and $\boldsymbol{A}_{f,1}$ with q/p over the reals and then round to the nearest integer component wise. The rounding results satisfy the following equation

$$\lfloor \frac{q}{p} \boldsymbol{c}_{f,1} \rceil - \boldsymbol{s}^T \lfloor \frac{q}{p} \boldsymbol{A}_{f,1} \rceil = f(\boldsymbol{x})\lfloor q/2 \rceil + \lfloor \frac{q}{p} e_f \rceil + \mathsf{error} \mod p$$

where the rounding error error is bounded by $|\mathsf{hsk}|_1 + O(1)$, which is polynomially bounded as the secret is sampled from the LWE noise distribution instead of uniformly. This gives a SHE scheme for $\mathsf{P/Poly}$.

2.3 Putting Pieces Together

We now briefly explain the roles of PHFE and SHE in recent constructions of FE [6, 2, 39, 33] (which then gives $i\mathcal{O}$).

The FE constructions of [2, 39] use SHE to encode the input $\boldsymbol{x}$, and to homomorphically compute f to obtain an encoding hct_f of the output $\boldsymbol{y} = f(\boldsymbol{x})$. To decrypt hct_f, by the special decryption equation of SHE, it seems sufficient to compute just $\mathsf{hsk}_f = \langle \mathsf{hsk}, \mathsf{pp}_f \rangle$, which can be done using FE for inner products [1, 3]. However, revealing hsk_f, as noted above, reveals the perturbed output $\boldsymbol{y}\lfloor p/2 \rceil + e_f$, where the noise e_f is sensitive. To circumvent this, the constructions of [2, 39] partially hides e_f using *larger smudging noises* generated by a noise generator G. This boils down to compute $\langle \mathsf{hsk}, \mathsf{pp}_f \rangle + G(\mathsf{sd}_1, \mathsf{sd}_2)$. Since known candidate noise generators have constant degree in the public seed sd_1 and degree 2 in the private sd_2, this can be computed by PHFE for constant degree polynomials as in [33].

Following the blueprint of previous works, using our SHE for $\mathsf{P/Poly}$, we obtain a FE scheme for $\mathsf{P/Poly}$. Our schemes handles $\mathsf{P/Poly}$ circuits directly, without relying on bootstrapping techniques that first converts it into a NC^0 computation using randomized encoding, and then use FE for NC^0 and local PRG to compute the randomized encoding. This allows us to eliminate the use of local PRGs. Since our SHE builds on predicate encryption, it enjoys efficiency features of predicate encryption. In particular, we obtain the first FE scheme without multilinear maps where the size of the ciphertexts grow sublinearly in the output length of the functionality, rather than its size. Our PHFE scheme for NC^1 computation opens the door to using candidate noise generators with higher complexity, ones with perhaps NC^1 comptuation on the public seed and degree 2 on the private seed, to build $i\mathcal{O}$. Although at the moment, we do not have such candidates whose security is better founded than the current constant degree candidates, we believe that the additional flexibility offered by our PHFE scheme will be helpful in minimizing assumptions for $i\mathcal{O}$ beyond what we already achieve in this work.

Reader's Guide In Section 3, we recall definitions of functional encryption, partially hiding funtional encryption and perturbation resilient generators. In Section 4, we define and construct the notion of a special homomorphic encoding scheme. In Section 5, we present our construction of functional encryption for $\mathsf{P/Poly}$. In Section 6, we construct a partially hiding functional encryption scheme. Finally in Section 7, we stitch all the results to construct $i\mathcal{O}$.

3 Preliminaries

In this section, we describe preliminaries that are useful for rest of the paper. 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 $(s, \epsilon)-$ indistinguishable if for every probabilistic polynomial time adversary $\mathcal{A}$ of size bounded by $O(s)$ it holds that: $\left| \Pr_{x \leftarrow \mathcal{X}_\lambda}[\mathcal{A}(1^\lambda, x) = 1] - \Pr_{y \leftarrow \mathcal{Y}_\lambda}[\mathcal{A}(1^\lambda, y) = 1] \right| \leq \epsilon$ for every sufficiently large $\lambda \in \mathbb{N}$. We drop the notation (s, ϵ) from (s, ϵ)-indistinguishable when s is polynomial and ϵ is negligible.

For a field element $a \in \mathbb{Z}_p$ 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 $[-B, B]$.

Notation. We denote vectors and matrices using boldfaced characters. For example, for any domain $\mathcal{D}$, any vector is represented using lower case bold characters such as $\mathbf{v} \in \mathcal{D}^{n\times 1}$ or $\mathbf{v} \in \mathcal{D}^{1\times n}$. A matrix is represented as capital boldfaced characters, for example, $\mathbf{A} \in \mathcal{D}^{n\times m}$. In either case, we will address these elements as $\mathbf{v}[i, j]$ (with either $i = 1$ or $j = 1$ in case of vectors) and $\mathbf{A}[i, j]$ in case of matrices. Sometimes, whenever there is no conflicting circumstances, we will index these elements as simply $\mathbf{v}_{i,j}$ and $\mathbf{A}_{i,j}$. In case we need to also index on the vectors and matrices (for example when $\mathbf{A}_k$ is some matrix) we will stick to the previous notation (and denote the elements as $\mathbf{A}_k[i, j]$).

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* $| \Pr_{x \leftarrow \mathcal{X}_\lambda}[\mathcal{A}(1^\lambda, x) = 1] - \Pr_{y \leftarrow \mathcal{Y}_\lambda}[\mathcal{A}(1^\lambda, y) = 1]|$

3.1 Functional Encryption

In this section we define the notion of a secret key functional encryption scheme for a circuit class $\mathcal{C} = \{\mathcal{C}_\lambda\}_{\lambda \in [n]}$. Here $\mathcal{C}_\lambda$ is some family of polynomial sized circuits.

Syntax. A secret key functional encryption scheme (denoted by FE) for a message space $\chi = \{\chi_\lambda\}_{\lambda \in \mathbb{N}}$ and circuit class $\mathcal{C} = \{\mathcal{C}_\lambda\}_\lambda$ is a tuple of PPT algorithms with the following properties:

- **Setup,** $\mathsf{Setup}(1^\lambda)$: On input security parameter λ, it outputs the master secret key MSK.
- **Encryption,** $\mathsf{Enc}(\mathsf{MSK}, x)$: On input the encryption key MSK and a message $x \in \chi_\lambda$, it outputs a ciphertext CT.

- **Key Generation,** $\mathsf{KeyGen}(\mathsf{MSK}, C)$: On input the master secret key MSK and a circuit $C \in \mathcal{C}_\lambda$, it outputs a functional key sk_C.
- **Decryption,** $\mathsf{Dec}(sk_C, \mathsf{CT})$: On input functional key sk_C and a ciphertext CT, it outputs the result out.

We define correctness property below.

Correctness. Consider any function $C \in \mathcal{C}_\lambda$ and any plaintext $x \in \chi_\lambda$. Consider the following process:

- $\mathsf{MSK} \leftarrow \mathsf{Setup}(1^\lambda)$
- $sk_C \leftarrow \mathsf{KeyGen}(\mathsf{MSK}, C)$.
- $\mathsf{CT} \leftarrow \mathsf{Enc}(\mathsf{MSK}, x)$

The following should hold over the coins of the algorithm:

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

for some negligible function negl.

Sub-Linear Efficiency: We require that for any message $\boldsymbol{x} \in \chi_\lambda$ the following holds:

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

Let $s_{\mathcal{C}}$ denote the maximum size of the circuit in $\mathcal{C}_\lambda$. Then it should hold that the size CT is less than $s_{\mathcal{C}}^{1-\epsilon} \cdot \mathsf{poly}(\lambda, |x|)$. Here, poly is some fixed polynomial, $\epsilon > 0$ is some constant and $|x|$ is the length of the message x. This notion of sublinearity is useful to construct $i\mathcal{O}$ [19, 7].
A stronger notion of sublinearity is described below.

Output sublinearity: We require that for any message $\boldsymbol{x} \in \chi_\lambda$ the following holds:

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

Let d be the maximum depth circuit inside $\mathcal{C}_\lambda$ and $\ell_{\mathcal{C}}$ be the maximum output length of circuits in $\mathcal{C}_\lambda$. Then it should hold that the size CT is less than $\ell_{\mathcal{C}}^{1-\epsilon} \cdot \mathsf{poly}(d, \lambda, |x|)$. Here, poly is some fixed polynomial, $\epsilon > 0$ is some constant and $|x|$ is the length of the message x.

Remark: Once we fix d to be a fixed polynomial in λ, then size of a circuit gives a trivial upper bound on its output length and thus, the notion of output sublinearity implies the notion of sublinearity.

As in [6], we consider the notion of weak security as we can rely on amplification theorem proved in [6] to construct fully secure functional encryption/ indistinguishability obfuscation. Now we define the notion of $(s, \mathsf{adv})-$ security. Here adv is a parameter denoting advantage of adversary and s is the parameter denoting the size of the adversary.

3.1.1 Semi-functional Security

We define the following auxiliary algorithms.

Semi-functional Key Generation, $\mathsf{sfKG}(\mathsf{MSK}, C, \theta)$: On input the master secret key MSK, function $C \in \mathcal{C}_\lambda$ and a value θ, it computes the semi-functional key $sk_{C,\theta}$.

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

We define two security properties associated with the above auxiliary algorithms. The propoerties are:

- **Indistinguishability of Semi-Functional Keys:** This property says roughly that given ciphertexts that are generated using honest algorithm, then honestly generated keys are indistinguishable to semi-functional keys where the hardwired values θ_i are chosen by the adversary (independently of the secret key).
- **Indistinguishability of Semi-Functional Ciphertext:** This property says roughly that given the keys are generated using the semi-functional algorithm with hardwired values $\theta_i = f_i(M^*)$, where M^* is some challenge message and f_i are queried functions, then honest ciphertext encrypting M^* are *weakly* indistinguishable to a semi-functionally generated ciphertext even given other honestly generated ciphertexts.

Due to lack of space, formal details of these definitions can be found in Section A.

3.2 Partially Hiding Functional Encryption

Now we define the notion of a partially hiding functional encryption scheme PHFE considered in the works of [6, 39].

Function class of interest: The function class $\mathcal{F} = \{\mathcal{F}_{n,p}\}_{n \in \mathbb{N}, p \in \mathsf{PRIMES}}$ is indexed by a positive integer n and a prime p. Each function $f \in \mathcal{F}_{n,p}$ takes as input a vector of the form $(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}) \in \mathbb{F}_p^{3 \cdot n}$ (without loss of generality $\boldsymbol{x}, \boldsymbol{y}$ and $\boldsymbol{z}$ are assumed to be of equal length) and outputs $f(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z})$ which is of the form $L(g(\boldsymbol{x}), \boldsymbol{y} \otimes \boldsymbol{z})$. Here, g is a polynomial sized arithmetic circuit, and L is a degree two multilinear polynomial of the following kind. $L(\boldsymbol{a}, \boldsymbol{b}) = \Sigma_{i,j} c_{i,j} a_i \cdot b_j$ where each $c_{i,j} \in \mathbb{Z}_p$. Thus, as a polynomial, f has degree 1 in $g(\boldsymbol{x})$ and $\boldsymbol{y} \otimes \boldsymbol{z}$. The exact subclass of g and L will be described when we present our construction.

Syntax. A PHFE scheme for function class $\mathcal{F}$ consists of the following polynomial time algorithms.

- **Setup,** $\mathsf{Setup}(1^\lambda, 1^n)$: On input security parameter λ and an input length n, it outputs the master secret key MSK. It also output p, which denotes the field of computation. This is implicitly known to the algorithms below.
- **Encryption,** $\mathsf{Enc}(\mathsf{MSK}, (\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}))$: On input the encryption key MSK and a message vector $(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}) \in \mathbb{Z}_p^{3 \cdot n}$, it outputs a ciphertext CT. CT implicitly contains $\boldsymbol{x}$ in the clear.

- **Key Generation,** $\mathsf{KeyGen}(\mathsf{MSK}, f)$: On input the master secret key MSK and a circuit $f \in \mathcal{F}_{n,p}$, it outputs a functional key sk_f.
- **Decryption,** $\mathsf{Dec}(sk_f, 1^B, \mathsf{CT})$: On input functional key sk_f, a polynomial bound B and a ciphertext CT, it outputs the result *out*.

We define correctness property below.

Correctness. Consider any function $f \in \mathcal{F}_{n,p}$ and any plaintext $M = (\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}) \in \mathbb{Z}_p^{3 \cdot n}$ and any bound B. Consider the following process:

- $\mathsf{MSK} \leftarrow \mathsf{Setup}(1^\lambda, 1^n)$
- $sk_f \leftarrow \mathsf{KeyGen}(\mathsf{MSK}, f)$.
- $\mathsf{CT} \leftarrow \mathsf{Enc}(\mathsf{MSK}, M)$. Set $\theta = f(M)$ if $f(M) \in [-B, B]$, else 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 $(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}) \in \mathbb{Z}_p^{3 \cdot n}$ 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 the ciphertext CT is less than $n \cdot \mathsf{poly}(\lambda)$. Here poly is some polynomial independent of n.

Now we describe the security notion for the scheme.

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, $\mathsf{sfEnc}(\mathsf{MSK}, \boldsymbol{x}, 1^n)$: 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 2 (Indistinguishability of Semi-functional Ciphertexts)**.** *A* PHFE *scheme for a class of functions* $\mathcal{F}$ *is said to satisfy* **indistinguishability of semi-functional ciphertexts property** *if for any p.p.t adversary* $\mathcal{A}$*, the probability that* $\mathcal{A}$ *succeeds in the following experiment is* negl *for some negligible function.*

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

1. *Adversary on input* 1^λ *specifies* n.

2. *Run* $\mathsf{Setup}(1^\lambda, 1^n) \rightarrow \mathsf{MSK}$. *Send* p *to the adversary.*

3. $\mathcal{A}$ *specifies the following:*

   - *Challenge message* $M^* = (\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z})$. *Here each vector is in* $\mathbb{Z}_p^n$.
   - *It can also specify additional messages* $\{M_k = (\boldsymbol{x}_k, \boldsymbol{y}_k, \boldsymbol{z}_k)\}_{k \in [q]}$ *Here each vector is in* $\mathbb{Z}_p^n$.
   - *It also specifies functions* $f_1, \ldots, f_\eta \in \mathcal{F}_{n,p}$ *and hardwired values* $\theta_1, \ldots, \theta_\eta \in \mathbb{Z}_p$.

4. *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.*

5. *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* $\mathsf{CT}^* \leftarrow \mathsf{sfEnc}(\mathsf{MSK}, \boldsymbol{x}, 1^n)$. *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* $i \in [q]$.

6. *The challenger sends* $\left(\{\mathsf{CT}_i\}_{i \in [q]}, \mathsf{CT}^*, \{sk[f_k, \theta_k]\}_{k \in [\eta]}\right)$ *to* $\mathcal{A}$.

7. *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 3 (Indistinguishability of Semi-functional Keys)**.** *A* PHFE *for a class of functions* $\mathcal{F}$ *is said to satisfy* **indistinguishability of semi-functional keys property** *if for any p.p.t. adversary* $\mathcal{A}$ *the probability that* $\mathcal{A}$ *succeeds in the following experiment is* negl *for some negligible function.*

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

1. *Adversary on input* 1^λ *specifies* n.

2. *Run* $\mathsf{Setup}(1^\lambda, 1^n) \rightarrow \mathsf{MSK}$. *Send* p *to the adversary.*

3. $\mathcal{A}$ *specifies the following:*

   - *It can specify messages* $M_j = \{(\boldsymbol{x}_i, \boldsymbol{y}_i, \boldsymbol{z}_i)\}_{j \in [q]}$. *Here each vector is in* $\mathbb{Z}_p^n$
   - *It specifies functions* $f_1, \ldots, f_\eta \in \mathcal{F}_{n,p}$ *and hardwired values* $\theta_1, \ldots, \theta_\eta \in \mathbb{Z}_p$.

4. *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* $j \in [q]$.

5. *Challenger then sends* $\left(\{\mathsf{CT}_i\}_{i\in[q]}, \{sk[f_i]^*\}_{i\in[\eta]}\right)$ *to* $\mathcal{A}$.

6. $\mathcal{A}$ *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 PHFE scheme satisfies both the above definitions, then it is said to satisfy semi-functional security.

Definition 4 (Semi-functional Security)**.** *Consider a partially hiding FE scheme* PHFE *for a class of functions* $\mathcal{F}$. *We say that* PHFE *satisfies* **semi-functional security** *if it satisfies indistinguishability of semi-functional ciphertexts property (Definition 2) and indistinguishability of semi-functional keys property (Definition 3).*

Remark: The definition described above is for polynomial security. When we refer to subexponential security, we will require the above conditions to hold for all adversary of size bounded by 2^{λ^c} for some constant $c > 0$ with advantage at most $2^{-\lambda^c}$.

3.3 Perturbation Resilient Generator

Now we describe the notion of a Perturbation Resilient Generator ($\Delta\mathsf{RG}$ for short), proposed by [6]. 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 $\Delta\mathsf{RG}$.

A $\Delta\mathsf{RG}$ satisfies the following properties.

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

- $|\mathsf{Seed}| = n \cdot \mathsf{poly}(\lambda)$ for some polynomial poly independent of n. The size of Seed is linear in n.

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

$(s, \mathsf{adv})-$**Perturbation Resilience:** We require that for every 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 s and any $(a_1, .., a_\ell) \in [-B, B]^\ell$

$$|\Pr[D(x \overset{\$}{\leftarrow} \mathcal{D}_1) = 1] - \Pr[D(x \overset{\$}{\leftarrow} \mathcal{D}_2) = 1]| < \mathsf{adv}$$

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 $\mathcal{H}(\mathsf{pp}, \mathsf{Seed}) \to (h_1, ..., h_\ell)$. Output $(\mathsf{pp}, h_1, ..., h_\ell)$.

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

Now we describe the notion of Perturbation Resilient Generator implementable in a function class $\mathcal{F}$ ($\mathcal{F}$-$\Delta\mathsf{RG}$ for short.)

$\Delta\mathsf{RG}$ implementable in $\mathcal{F}$. A $\Delta\mathsf{RG}$ scheme implementable in function class $\mathcal{F} = \{\mathcal{F}_n\}_{n\in\mathbb{N}}$ ($\mathcal{F}$-$\Delta\mathsf{RG}$ for short) is a perturbation resilient generator with additional properties. We describe syntax again for a complete specification.

- $\mathsf{Setup}(1^\lambda, 1^n, B) \rightarrow (\mathsf{pp}, \mathsf{Seed})$. 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, $\mathsf{Seed} = (\mathsf{Seed.pub}, \mathsf{Seed.priv}(1), \mathsf{Seed.priv}(2))$ is a vector on $\mathbb{Z}_p$. Also, $\mathsf{pp} = (\mathsf{Seed.pub}(1), q_1, .., q_\ell)$. We require syntactically there exists two algorithms $\mathsf{SetupSeed}$ and $\mathsf{SetupPoly}$ such that Setup can be decomposed follows:

  1. $\mathsf{SetupSeed}(1^\lambda, 1^n, B) \rightarrow \mathsf{Seed}$. The $\mathsf{SetupSeed}$ algorithm outputs the seed.
  2. $\mathsf{SetupPoly}(1^\lambda, 1^n, B) \rightarrow q_1, ..., q_\ell$. The $\mathsf{SetupPoly}$ algorithm outputs $q_1, .., q_\ell$.

- $\mathsf{Eval}(\mathsf{pp}, \mathsf{Seed}) \rightarrow (h_1, ..., h_\ell)$, 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 $\mathcal{F}$-$\Delta\mathsf{RG}$. Here each q_i is in $\mathcal{F}_n$.

The security and efficiency requirements are same as before.
Remark: Few remarks are in order,

1. To construct $i\mathcal{O}$ we need the stretch of $\mathcal{F}$-$\Delta\mathsf{RG}$ to be equal to $\ell = n^{1+\epsilon}$ for some constant $\epsilon > 0$.

2. Looking ahead, we will use a $\mathcal{F}$-$\Delta\mathsf{RG}$ for a function class $\mathcal{F}$, that is also the function class for a PHFE scheme.

We discuss assumptions and candidates required to build $\Delta\mathsf{RG}$ sufficient for $i\mathcal{O}$ in Section B.

4 Special Homomorphic Encoding from GVW Predicate Encryption

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 integer linear combinations are exactly Λ. Every integer lattice is generated as the $\mathbb{Z}$-linear combination of linearly independent vectors $\mathbf{B} = \{\boldsymbol{b}_1, ..., \boldsymbol{b}_m\} \subset \mathbb{Z}^m$. For a matrix $\mathbf{A} \in \mathbb{Z}_p^{\mathsf{d}\times m}$, we define the "$p$-ary" integer lattices:

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

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

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 [47], who showed that solving it *on average* is as hard as (quantumly) solving several standard lattice problems *in the worst case.*

Definition 5 (LWE). *For an integer $p = p(\mathsf{d}) \geq 2$, and an error distribution $\chi = \chi(\mathsf{d})$ over $\mathbb{Z}_p$, the* Learning With Errors problem $\mathsf{LWE}_{\mathsf{d},m,p,\chi}$ *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 $\mathbf{A} \leftarrow \mathbb{Z}_q^{\mathsf{d} \times m}$, $\boldsymbol{s} \leftarrow \mathbb{Z}_p^{\mathsf{d}}$, $\boldsymbol{u} \leftarrow \mathbb{Z}_p^m$, and $\boldsymbol{x} \leftarrow \chi^m$.

Gadget matrix. The gadget matrix described below is proposed in [43].

Definition 6. *Let $m = \mathsf{d} \cdot \lceil \log p \rceil$, and define the gadget matrix $\mathbf{G} = \boldsymbol{g}_2 \otimes \mathbf{I}_{\mathsf{d}} \in \mathbb{Z}_p^{\mathsf{d} \times m}$, where the vector $\boldsymbol{g}_2 = (1, 2, 4, ..., 2^{\lfloor \log p \rfloor}) \in \mathbb{Z}_p^{\lceil \log p \rceil}$. We will also refer to this gadget matrix as "powers-of-two" matrix. We define the inverse function $\mathbf{G}^{-1} : \mathbb{Z}_p^{\mathsf{d} \times m} \to \{0,1\}^{m \times m}$ which expands each entry $a \in \mathbb{Z}_p$ of the input matrix into a column of size $\lceil \log p \rceil$ consisting of the bits of binary representations. We have the property that for any matrix $\mathbf{A} \in \mathbb{Z}_p^{\mathsf{d} \times m}$, it holds that $\mathbf{G} \cdot \mathbf{G}^{-1}(\mathbf{A}) = \mathbf{A}$.*

Next, we define the notion of a special homomorphic encoding (denoted by SHE)

4.1 Special Homomorphic Encoding

A special homomorphic encoding scheme for a class of circuits $\mathcal{C} = \{\mathcal{C}_{n,\lambda}\}_{n,\lambda \in \mathbb{N}}$ consists of the following algorithms. Here $\mathcal{C}_{n,\lambda}$ denote circuits with n bit inputs, depth λ and one output bit.

- $\mathsf{Setup}(1^\lambda, 1^n, p) \to \mathsf{PK}$. On input the security parameter and the number of input bits n and a modulus p, it outputs the public key PK.
- $\mathsf{Encode}(\mathsf{PK}, \boldsymbol{s}, m) \to \mathsf{CT}$. The encoding algorithm takes as input the message $m \in \{0,1\}^n$, a randomly chosen secret $\boldsymbol{s} \in \mathbb{Z}_p^{\dim_1 \times 1}$ from $\chi^{\dim_1 \times 1}$. Here χ is a polynomially bounded distribution and $\dim_1 = \mathsf{poly}(\lambda)$ independent of n. It outputs an encoding CT. Here $|\mathsf{CT}| \leq n \cdot \mathsf{poly}(\lambda, \log p)$.
- $\mathsf{EvalPK}(\mathsf{PK}, C)$: This deterministic algorithm takes as input PK along with a circuit $C \in \mathcal{C}_{n,\lambda}$ and outputs a vector $\boldsymbol{b}_C \in \mathbb{Z}_p^{\dim_1 \times 1}$
- $\mathsf{EvalCT}(\mathsf{PK}, C, \mathsf{CT})$: The EvalCT algorithm takes as input an encoding CT, a circuit $C \in \mathcal{C}_{n,\lambda}$, the public key PK and outputs a field element $\mathsf{CT}_C \in \mathbb{Z}_p$. This has the following structure:
  - First, $\mathsf{CT}_C = \langle \boldsymbol{s}, \boldsymbol{b}_C \rangle + e_C + C(m) \cdot Q$.
  - Here $|e_C| \leq \mathsf{poly}(\lambda, n)$
  - Q is a fixed number (depending on p) which is $\Theta(2^{\lambda^c})$ for some constant $c > 0$ and it is smaller than p
  - $\boldsymbol{b}_C \leftarrow \mathsf{Eval}(\mathsf{PK}, C)$

Security: Consider the following experiments for $b \in \{0,1\}$ and let $\mathcal{A}$ be any polynomial time adversary.

$\mathsf{Expt}_b(1^\lambda, 1^n, p)$:

- Adversary outputs $m_0, m_1 \in \{0,1\}^n$.
- Run $\mathsf{Setup}(1^\lambda, 1^n, p) \to \mathsf{PK}$.
- Sample a secret $\boldsymbol{s} \leftarrow \chi^{\dim_1}$. Compute $\mathsf{CT} \leftarrow \mathsf{Encode}(\mathsf{PK}, \boldsymbol{s}, m_b)$.
- Adversary on input PK, CT outputs $b' \in \{0,1\}$

Then, we require for any large enough security parameter λ, modulus $p = \theta(2^{\lambda^c})$ for any large enough constant $c > 0$, and any $n = n(\lambda)$

$$|\Pr[\mathsf{Expt}_0(1^\lambda, 1^n, p) = 1] - \Pr[\mathsf{Expt}_1(1^\lambda, 1^n, p) = 1]| < \mathsf{negl}(\lambda)$$

We say that the scheme is subexponentially secure if it holds for subexponential sized adversary. We now show how to construct such a SHE scheme assuming LWE.

4.2 Construction from GVW Predicate Encryption

Predicate Encryption. Now we recall the definition of predicate encryption scheme. A predicate encryption is a functional encryption scheme as described in Section 3. There are following differences.

- Encryptor encrypts messages of the form (attr, m).
- $\mathcal{C}$ consists of polynomially sized circuits C_P, where P is a predicate. C_P on input (attr, m) outputs m if $P(\mathsf{attr}) = 1$ and 0 otherwise.
- Security definition allows adversary to ask for any number of functional keys corresponding to predicates $P_1, ..., P_\eta$ as long as $P_i(\mathsf{attr}_0) = P_i(\mathsf{attr}_1) = 0$ where (attr_0, m_0) and (attr_1, m_1) are the challenge messages. In such a setting the adversary needs to distinguish between encryption of (attr_0, m_0) from encryption of (attr_1, m_1).

For a complete definition refer [30]. For our construction, we require some special properties from the predicate encryption scheme, such as efficiency, circuit homomorphism etc. All these properties are satisfied by the construction of [30], and we recall them next.

Properties of GVW Predicate Encryption Scheme. Let $n = \mathsf{poly}(\lambda)$ for any polynomial poly. For concreteness, let $\mathcal{P}_{n,\lambda}$ denote polynomially sized circuits with n bit inputs and one bit output. The depth of these circuits are bounded by λ. Considering $\mathcal{P}_{n,\lambda}$ as the class of predicates, we describe the properties below. We now describe various algorithms and associated properties of the GVW predicate encryption scheme. We denote the scheme by PE.

Setup. The setup algorithm takes as input security parameter λ and n and outputs a public key PK. Namely, $\mathsf{Setup}(1^\lambda, 1^n) \to (\mathsf{PK}, \mathsf{SK})$

- As a part of PK is the modulus p_1. Length of p_1 is $O(\mathsf{poly}_1(\lambda))$. It also outputs a dimensions $\dim_1 = \mathsf{poly}_2(\lambda)$ and $\dim_2 = \mathsf{poly}_3(\lambda)$. These are the dimensions of various matrices used in the scheme.
- PK consists of uniform matrices $\mathbf{B}_1, ..., \mathbf{B}_\ell, \mathbf{A}, \mathbf{D}$ where $\ell = n \cdot \mathsf{poly}(\lambda)$ for some polynomial poly. Each matrix is in $\mathbb{Z}_{p_1}^{\dim_1 \times \dim_2}$. This is also the space of the gadget matrix $\mathbf{G}$.

Encryption. The encryption algorithm takes as input public key PK, attribute $\mathsf{attr} \in \{0,1\}^n$ and a message $m \in \{0,1\}$ and does the following. $\mathsf{Enc}(\mathsf{PK}, \mathsf{attr}, m) \to (\mathsf{CT}_1, \mathsf{CT}_2)$, Now we describe in more detail.

- The encryption algorithm first samples a secret vector $\boldsymbol{s}$ from $\chi^{\dim_1 \times 1}$. Here χ is LWE error distribution used by the scheme. Then, it encodes attr to output $\widehat{\mathsf{attr}} \in \mathbb{Z}_{p_1}^{\ell}$.
- Now CT_1 is constructed as follows.
  - Compute $\boldsymbol{b}_i = \boldsymbol{s}^T(\mathbf{B}_i + \widehat{\mathsf{attr}}_i\mathbf{G}) + \mathbf{E}_i$ for $i \in [\ell]$. Here $\mathbf{E}_i \in \chi^{1 \times \dim_2}$.
  - Output $\mathsf{CT}_1 = (\boldsymbol{b}_1, \ldots, \boldsymbol{b}_\ell, \widehat{\mathsf{attr}})$
- Now CT_2 is constructed as follows.
  - Compute $\boldsymbol{a} = \boldsymbol{s}^T\mathbf{A} + \mathbf{E}$. Here $\mathbf{E} \in \chi^{1 \times \dim_2}$.
  - Compute $\boldsymbol{d} = \boldsymbol{s}^T\mathbf{D} + \mathbf{E}' + m\lfloor p_1/2 \rfloor [1, 0, ..., 0]$. Here $\mathbf{E}' \in \chi^{1 \times \dim_2}$.
  - Output $\mathsf{CT}_2 = (\boldsymbol{a}, \boldsymbol{d})$.
- By Enc_1 we denote the algorithm that takes as input PK and secret $\mathbf{s}$, attribute attr and outputs CT_1.
- Without loss of security we can assume $\boldsymbol{s}[1,1] = 1$ (first component of vector $\boldsymbol{s}$). This ensures that $\boldsymbol{v} = \boldsymbol{s}^T\mathbf{G}$ satisfies $\boldsymbol{v}[1,1] = 1$.
- In our construction, we will use Enc_1 algorithm instead of the encryption algorithm, thereby not computing CT_2 at all. This does not hamper security as we are just giving less information.

Evaluation. There are two algorithms: EvalPK and EvalCT. First we describe the $\mathsf{EvalPK}()$ algorithm. Formally, $\mathsf{EvalPK}(C, \mathbf{B}_1, ..., \mathbf{B}_\ell) \to \mathbf{B}_C$. On input $\mathbf{B}_1, \ldots, \mathbf{B}_\ell \in \mathbb{Z}_{p_1}^{\dim_1 \times \dim_2}$ and $C \in \mathcal{P}_{n,\lambda}$ outputs $\mathbf{B}_C \in \mathbb{Z}_{p_1}^{\dim_1 \times \dim_2}$.
EvalCT takes as input $\widehat{\mathsf{attr}}$, $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_\ell$ and $C \in \mathcal{P}_{n,\lambda}$. Formally, $\mathsf{EvalCT}(\mathsf{PK}, C, \widehat{\mathsf{attr}}, \boldsymbol{b}_1, \ldots, \boldsymbol{b}_\ell) \to \hat{\boldsymbol{b}}_C$. $\hat{\boldsymbol{b}}_C$ has the following structure:

$$\hat{\boldsymbol{b}}_C = \boldsymbol{s}^T(\mathbf{B}_C + (C(\mathsf{attr})\lfloor p_1/2 \rfloor + e)\mathbf{G}) + \mathbf{E}_C$$

Here $\|\mathbf{E}_C\|_\infty / p_1 < 2^{-\lambda^c}$ and $\|e\|/p_1 < 2^{-\lambda^c}$ for some constant $c > 0$. In fact $|e| < \mathsf{poly}(\lambda, n)$ for some polynomial.

Rounding-Evaluation. We now describe a procedure of rounding evaluation, which can be done publicly. We denote this by $\mathsf{RoundEval}$. $\mathsf{RoundEval}$ takes as input $\mathsf{PK}, \mathsf{CT}_1 = (\widehat{\mathsf{attr}}, \boldsymbol{b}_1, \ldots, \boldsymbol{b}_\ell)$, a circuit C, another modulus $p < p_1$.
More formally, $\mathsf{RoundEval}(\mathsf{PK}, C, \mathsf{CT}_1, p)$ does the following:

1. First run $\mathsf{EvalCT}(\mathsf{PK}, C, \mathsf{CT}_1) \to \hat{\boldsymbol{b}}_C$.
2. Now compute $\hat{\boldsymbol{b}}'_C = \lceil p/p_1 \cdot \hat{\boldsymbol{b}}_C \rfloor$. Namely multiply $\hat{\boldsymbol{b}}_C$ with p/p_1 over the reals and then take the nearest integer, component wise. $\hat{\boldsymbol{b}}'_C$ is now a vector over $\mathbb{Z}_p$.
3. Output $b'_C = \hat{\boldsymbol{b}}'_C[1,1]$, the first element of vector $\hat{\boldsymbol{b}}'_C$.

Now we observe something about b'_C. First observe $\hat{\boldsymbol{b}}_C[1,1]$ has the following structure $\hat{\boldsymbol{b}}_C[1,1] = \boldsymbol{s}^T \cdot \boldsymbol{B}_{C,1} + (C(\mathsf{attr})\lfloor p_1/2 \rfloor + e) \cdot \boldsymbol{v}[1,1]) + \mathbf{E}_C[1,1]$. Here $\boldsymbol{B}_{C,1}$ is the first column of $\mathbf{B}_C$ and $\boldsymbol{v} = \boldsymbol{s}^T \cdot \mathbf{G}$. Since $\boldsymbol{s}[1,1] = 1$, $\boldsymbol{v}[1,1] = 1$. Thus, $\hat{\boldsymbol{b}}_C[1,1] = \boldsymbol{s}^T \boldsymbol{B}_{C,1} + C(\mathsf{attr})\lfloor p_1/2 \rfloor + e + \mathbf{E}_C[1,1]$.
Let χ be a polynomially bounded distribution (bounded by $\mathsf{poly}_\chi(\lambda)$), then, we observe the following about b'_C relying on the theorems proven in [21] (see lemma 1 of the paper).

Theorem 5. *Assuming:*

- $\hat{\boldsymbol{b}}_C[1,1] = \boldsymbol{s}^T \boldsymbol{B}_{C,1} + C(\mathsf{attr})\lfloor p_1/2 \rfloor + e + \mathbf{E}_C[1,1]$
- χ *is a polynomially bounded distribution, bounded by,* $\mathsf{poly}_\chi(\lambda)$.

Then $b'_C = \boldsymbol{s}^T \cdot \mathbf{B}'_{C,1} + C(\mathsf{attr})\lfloor p_1/2 \rfloor' + e' + \mathbf{E}'_C[1,1] + \mathsf{error}$. *Here* $\mathbf{B}'_{C,1}$ *is the rounded version of* $\mathbf{B}_{C,1}$, e' *is a rounded version of* e, $\lfloor p_1/2 \rfloor'$ *is rounded version of* $\lfloor p_1/2 \rfloor$ *and* $\mathbf{E}'_C[1,1]$ *is rounded version of* $\mathbf{E}_C[1,1]$. error *is rounding error satisfying* $|\mathsf{error}| < \dim_1 \cdot \mathsf{poly}_\chi(\lambda) + 3$

Parameters. For our construction in Section 5, we need following setting of parameters. Let the class of circuits be $\mathcal{P}_{n,\lambda}$.

- $p > 2^{\lambda^c}$ for some constant $c > 0$.
- Error distribution χ (also the distribution for sampling secrets) is bounded by a polynomial $\mathsf{poly}_\chi(\lambda)$.
- p is set so that $|e'| + |\mathbf{E}'_C[1,1]| < \mathsf{poly}(\lambda, n)$.
- We set $\mathsf{Bound}_\Delta = max\{|e'| + |\mathbf{E}'_C[1,1]| + |\mathsf{error}|\}$
- p_1 is chosen so that $Q = \lceil p_1/2 \rfloor'$ is $\Theta(2^{\lambda^c})$ for some constant $c > 0$.

As shown in [30], these parameters can be instantiated using LWE with subexponential approximation factors.

Constructing SHE : With this, the construction of SHE becomes almost immediate.

- $\mathsf{Setup}(1^\lambda, 1^n, p)$: This algorithm runs $\mathsf{PE.Setup}(1^\lambda, 1^n) \to p_1, \mathbf{B}_1, ..., \mathbf{B}_\ell, \mathbf{A}, \mathbf{D}$. Output $\mathsf{PK} = (p, p_1, \mathbf{B}_1, ..., \mathbf{B}_\ell)$
- $\mathsf{Encode}(\mathsf{PK}, \boldsymbol{s}, m)$: Run $\mathsf{PE.Enc}_1(\mathsf{PK}, \boldsymbol{s}, m) \to \mathsf{CT}$. Output CT.
- $\mathsf{EvalPK}(\mathsf{PK}, C)$. Run $\mathsf{PE.EvalCT}(\mathsf{PK}, C)] \to \mathbf{B}_C$. Output $\boldsymbol{b}_C = \mathbf{B}'_{C,1}$. That is the first column of $\mathbf{B}_C$ which is rounded to p.
- $\mathsf{EvalCT}(\mathsf{PK}, C, \mathsf{CT}) \to \mathsf{CT}_C$. Run $\mathsf{PE.RoundEval}(\mathsf{PK}, C, \mathsf{CT}, p) \to \mathsf{CT}_C$. Output CT_C.

Security and structure properties are immediate from the properties of the PE scheme.

5 Construction: Functional Encryption

In this section we present our construction for a sublinear functional encryption scheme.

Function Class of Interest: The function class for which we will build is $\mathcal{C}_{\mathsf{FE},n,\lambda,\epsilon}$. This class contains all boolean circuits with depth bounded by λ, number of inputs equal to n, and number of outputs bounded by $n^{1+\epsilon}$ for some $\epsilon > 0$. The size of each circuit in the family is bounded by an arbitrary polynomial in n, λ. Looking ahead, in our construction ϵ is a function of the stretch of the perturbation resilient generator used in the scheme.

Ingredients: Following are the ingredients for our scheme.

- A special homomorphic encryption scheme SHE, which can be build using [30] predicate encryption scheme. The class of circuits for this scheme is $\mathcal{C}_{n,\lambda}$. Let the modulus used by the scheme be $p = O(2^{\lambda^c})$ for some constant $c > 0$ independent of n. Let χ be the error distribution.
- Partially Hiding Functional Encryption PHFE for a function class $\mathcal{F} = \mathcal{F}_n$. Recalling the definition in Section 3.2, each $f \in \mathcal{F}$ is of the following form: $f(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}) = L(g(\boldsymbol{x}), \boldsymbol{y} \otimes \boldsymbol{z})$. We require g to at least contain arbitrary degree one polynomials over $\mathbb{Z}_p$ and L to be the inner product functionality. The decryption procedure of PHFE is two staged. First stage is Dec_0, which computes $\mathbf{g}^{f(\boldsymbol{x},\boldsymbol{y},\boldsymbol{z})}$ where $\mathbf{g}$ is some canonical group generator of order p. The second stage Dec_1 takes as input some bound $\mathsf{Bound}_{\mathsf{PHFE}}$ and by brute force checks if the exponent is in range $[-\mathsf{Bound}_{\mathsf{PHFE}}, \mathsf{Bound}_{\mathsf{PHFE}}]$. If that is the case it outputs the exponent otherwise it outputs $\perp$.
- Perturbation resilient generators implementable in $\mathcal{F}$. We call them $\mathcal{F}$-$\Delta\mathsf{RG}$. If the stretch of $\mathcal{F}$-$\Delta\mathsf{RG}$ is $m = n^{1+\epsilon'}$, ϵ can be set as any constant less than ϵ'.
- We set two bounds. First bound is Bound_Δ. This bound is input to $\mathcal{F}$-$\Delta\mathsf{RG}$ algorithms and can be instantiated as described in Section 4.2. Second bound, $\mathsf{Bound}_{\mathsf{PHFE}}$, is given as input to PHFE decryption algorithm. This is instantiated as the sum of maximum norm of the $\mathcal{F}$-$\Delta\mathsf{RG}$ values and Bound_Δ. Both these bounds are guaranteed to be polynomials in λ and n.

Construction.

$\underline{\mathsf{Setup}(1^\lambda, 1^n)}$: The setup algorithm does the following:

1. Run $\mathsf{PHFE.Setup}(1^\lambda, 1^{n'}) \to \mathsf{PHFE.MSK}$. Here $n' = n \cdot \mathsf{poly}(\lambda)$ for some polynomial poly specified later. This polynomial is independent of n and will be specified later. Let p be the modulus of PHFE scheme.
2. Run $\mathsf{SHE.Setup}(1^\lambda, 1^n, p) \to \mathsf{SHE.PK}$.
3. Sample a secret vector $\boldsymbol{s}^* \in \chi^{\dim_1 \times 1}$. Set $\boldsymbol{s}^*[1,1] = 1$. This is only used in the semifunctional algorithms.
4. Compute $\mathsf{SHE.Encode}(\mathsf{SHE.PK}, \boldsymbol{s}^*, 0^n) \to \mathsf{SHE.CT}^*$. This is also only used in the semifunctional algorithms. Run $\mathcal{F}\text{-}\Delta\mathsf{RG.SetupSeed}(1^\lambda, 1^n, \mathsf{Bound}_\Delta) \to \mathsf{Seed}^*$. This is also used in the semifunctional algorithms.
5. Output $\mathsf{MSK} = (\mathsf{PHFE.MSK}, \mathsf{SHE.PK}, \boldsymbol{s}^*, \mathsf{SHE.CT}^*, \mathsf{Seed}^*)$

$\underline{\mathsf{Enc}(\mathsf{MSK}, m \in \{0,1\}^n)}$: The Encryption algorithm takes as input $\mathsf{MSK} = (\mathsf{PHFE.MSK}, \mathsf{SHE.PK},$ $\boldsymbol{s}^*, \mathsf{SHE.CT}^*, \mathsf{Seed}^*)$ and message $m \in \{0,1\}^n$ and does the following.

1. Sample a secret $\boldsymbol{s} \leftarrow \chi^{\dim_1 \times 1}$. Set $\boldsymbol{s}[1,1] = 1$.
2. Compute $\mathsf{SHE.Encode}(\mathsf{SHE.PK}, \boldsymbol{s}, m) \to \mathsf{SHE.CT}$.
3. Run $\mathcal{F}\text{-}\Delta\mathsf{RG.SetupSeed}(1^\lambda, 1^n, \mathsf{Bound}_\Delta) \to \mathsf{Seed}$. Parse $\mathsf{Seed} = (\mathsf{Seed.pub}, \mathsf{Seed.priv}(1), \mathsf{Seed.priv}(2))$.
4. Set $\boldsymbol{x} = (\mathsf{Seed.pub}, 1)$, $\boldsymbol{y} = (\mathsf{Seed.priv}(1), \boldsymbol{s})$ and $\boldsymbol{z} = (\mathsf{Seed.priv}(2), 1)$.
5. $\mathsf{PHFE.Enc}(\mathsf{MSK}, (\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z})) \to \mathsf{PHFE.CT}$. n' is set accordingly. n' is set as the maximum of length of $\boldsymbol{x}, \boldsymbol{y}$ and $\boldsymbol{z}$. Note that this is bounded by $n \cdot \mathsf{poly}(\lambda)$ for a fixed polynomial in λ due to the properties of $\mathcal{F}\text{-}\Delta\mathsf{RG}$ and SHE.
6. Output $(\mathsf{CT}_1 = \mathsf{SHE.CT}, \mathsf{CT}_2 = \mathsf{PHFE.CT})$

$\underline{\mathsf{KeyGen}(\mathsf{MSK}, C)}$: The keygen algorithm takes as input $\mathsf{MSK} = (\mathsf{PHFE.MSK}, \mathsf{SHE.PK}, \boldsymbol{s}^*, \mathsf{SHE.CT}^*, \mathsf{Seed}^*)$ and a circuit $C \in \mathcal{C}_{n,\lambda,\epsilon}$.

1. Parse $C = (C_1, \ldots, C_\eta)$ where C_i is the circuit that outputs i^{th} bit of C. Here $\eta \leq n^{1+\epsilon}$.
2. For each $i \in [\eta]$, compute $\mathsf{SHE.EvalPK}(\mathsf{PK}, C_i) \to \boldsymbol{b}_{C_i}$.
3. Sample $\mathcal{F}\text{-}\Delta\mathsf{RG.SetupPoly}(1^\lambda, 1^n, \mathsf{Bound}_\Delta) \to (q_1, ..., q_{n^{1+\epsilon'}})$.
4. Let G_i be the following arithmetic circuit. On input, vectors $(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z})$ where $\boldsymbol{x} = (\mathsf{Seed.pub}, 1)$, $\boldsymbol{y} = (\mathsf{Seed.priv}(1), \boldsymbol{s})$ and $\boldsymbol{z} = (\mathsf{Seed.priv}(2), 1)$, G_i outputs $\boldsymbol{s}^T \cdot \boldsymbol{b}_{C_i} - q_i(\mathsf{Seed})$. Note that the computations are done over $\mathbb{Z}_p$ and thus $G_i \in \mathcal{F}_{n'}$.
5. For each $i \in [\eta]$, compute $\mathsf{PHFE}.sk_{G_i} \leftarrow \mathsf{PHFE.KeyGen}(\mathsf{PHFE.MSK}, G_i)$.
6. Output $sk_C = (\mathsf{PHFE}.sk_{G_1}, \ldots, \mathsf{PHFE}.sk_{G_\eta})$

$\underline{\mathsf{Dec}(sk_C, \mathsf{CT}):}$ Parse $\mathsf{CT} = (\mathsf{SHE.CT}, \mathsf{PHFE.CT})$ and $sk_C = (\mathsf{PHFE}.sk_{G_1}, ..., \mathsf{PHFE}.sk_{G_\eta})$. Now proceed as follows. Recall that $C = (C_1, \ldots, C_\eta)$ where C_i outputs i^{th} bit of C.

1. Compute $\mathsf{SHE.EvalCT}(\mathsf{SHE.PK}, C_i, \mathsf{SHE.CT}) \to \mathsf{SHE.CT}_{C_i}$ for $i \in [\eta]$.
2. For each $i \in [\eta]$, compute $\mathsf{PHFE.Dec}_0(\mathsf{PHFE}.sk_{G_i}, \mathsf{PHFE.CT}) = \mathbf{g}^{\mathsf{Int}_i}$
3. Compute $\mathsf{Int}_{2,i} = \mathbf{g}^{-\mathsf{Int}_i} \cdot \mathbf{g}^{\mathsf{SHE.CT}_{C_i}}$ for $i \in [\eta]$.
4. For each $i \in [\eta]$, compute $\mathsf{Int}_{3,i} = \mathsf{PHFE.Dec}_1(\mathsf{Int}_{2,i}, \mathsf{Bound}_{\mathsf{PHFE}})$. Finally, if $\mathsf{Int}_{3,i}$ is $\perp$ then set $\mathsf{Int}_{4,i} = 1$, otherwise set $\mathsf{Int}_{4,i} = 0$.
5. Output $(\mathsf{Int}_{4,1}, \ldots, \mathsf{Int}_{4,\eta})$

Correctness. We now go over each steps of the decryption algorithm and argue why correctness holds.

Parse $\mathsf{CT} = (\mathsf{SHE.CT}, \mathsf{PHFE.CT})$ and $sk_C = (\mathsf{PHFE}.sk_{G_1}, ..., \mathsf{PHFE}.sk_{G_\eta})$. Now proceed as follows. Recall that $C = (C_1, \ldots, C_\eta)$ where C_i outputs i^{th} bit of C.

1. Compute $\mathsf{SHE.EvalCT}(\mathsf{SHE.PK}, C_i, \mathsf{SHE.CT}) \to \mathsf{SHE.CT}_{C_i}$ for $i \in [\eta]$. This ensures that $\mathsf{SHE.CT}_{C_i} = \boldsymbol{s}^T \cdot \boldsymbol{b}_{C_i} + C_i(m) \cdot Q + e_{C_i}$. Here Q is a large value described in the construction of SHE and $|e_{C_i}| < \mathsf{poly}(\lambda, n)$
2. For each $i \in [\eta]$, compute $\mathsf{PHFE.Dec}_0(\mathsf{PHFE}.sk_{G_i}, \mathsf{PHFE.CT}) = \mathbf{g}^{\mathsf{Int}_i}$. This ensures $\mathsf{Int}_i = \boldsymbol{s}^T \cdot \boldsymbol{b}_{C_i} - q_i(\mathsf{Seed})$.
3. Compute $\mathsf{Int}_{2,i} = \mathbf{g}^{-\mathsf{Int}_i} \cdot \mathbf{g}^{\mathsf{SHE.CT}_{C_i}}$ for $i \in [\eta]$. This ensures $\mathsf{Int}_{2,i} = \mathbf{g}^{C_i(m)Q + e_{C_i} + q_i(\mathsf{Seed})}$
4. For each $i \in [\eta]$, compute $\mathsf{Int}_{3,i} = \mathsf{PHFE.Dec}_1(\mathsf{Int}_{2,i}, \mathsf{Bound}_{\mathsf{PHFE}})$. This ensures that if $C_i(m) = 0$, $\mathsf{Int}_{3,i} = e_{C_i} + q_i(\mathsf{Seed})$ otherwise $\mathsf{Int}_{3,i} = \perp$.
5. Finally, if $\mathsf{Int}_{3,i}$ is $\perp$ then set $\mathsf{Int}_{4,i} = 1$, otherwise set $\mathsf{Int}_{4,i} = 0$.
6. Output $(\mathsf{Int}_{4,1}, \ldots, \mathsf{Int}_{4,\eta})$. This ensures output is $C(m)$.

Output-Sublinearity. Now we argue about the size of the ciphertext. Ciphertext contains two components: $(\mathsf{SHE.CT}, \mathsf{PHFE.CT})$.

- $|\mathsf{SHE.CT}| < n \cdot \mathsf{poly}(\lambda)$. As the depth of the circuit class is λ, this follows from the compactness properties of [30] Predicate Encryption scheme.
- $\mathsf{PHFE.CT}$ is a PHFE encryption of three vectors $\boldsymbol{x} = (\mathsf{Seed.pub}, 1)$, $\boldsymbol{y} = (\mathsf{Seed.priv}(1), \boldsymbol{s})$ and $\boldsymbol{z} = (\mathsf{Seed.priv}(2), 1)$. Now note that $|\mathsf{Seed}| < n \cdot \mathsf{poly}(\lambda)$ for some fixed polynomial in λ and $\boldsymbol{s}$ is a vector of dimension $\dim_1$, which is a polynomial in λ, independent of n. Thus by linear efficiency of PHFE, the size of this component is bounded by $n \cdot \mathsf{poly}(\lambda)$ for some fixed polynomial poly.
- For $\epsilon_1 > 0$, let $n^{1+\epsilon_1}$ be the stretch of $\mathcal{F}$-$\Delta\mathsf{RG}$. Thus the scheme allows to evaluate circuits of output length $\ell_{out} = n^{1+\epsilon_1}$. Thus, size of ciphertext is bounded by $\ell_{out}^{1/(1+\epsilon_1)}\mathsf{poly}(\lambda)$. This proves output sublinearity.

Due to lack of space we present the security proof for this construction in Section E.

6 Construction: Partially hiding FE

In this section we describe our construction for our PHFE scheme. Below we describe our function class of interest. First let us define a few operators to make our description simpler.

- On input any matrix $\mathbf{M} \in \mathbb{Z}_p^{k \times k}$ for any integer $k > 0$, $\mathsf{vec}(\mathbf{M})$ outputs $\boldsymbol{v}$, which is a row vector in $\mathbb{Z}_p^{1 \times k^2}$. This is computed by reading $\mathbf{M}$ into $\mathbf{v}$ row-wise.
- On input any vector $\boldsymbol{v} \in \mathbb{Z}_p^{1 \times k^2}$, $\mathsf{diag}(\boldsymbol{v})$ outputs a block diagonal matrix $\mathbf{M}' \in \mathbb{Z}_p^{k^3 \times k^3}$ where each diagonal block $\mathbf{M}'_{i,i} = v_i \cdot I_{k \times k}$ for $i \in [k^2]$
- Define $\mathsf{diagvec}$ as the the operator that takes in a matrix $\mathbf{M} \in \mathbb{Z}_p^{k \times k}$ and outputs the composition of $\mathsf{diag} \circ \mathsf{vec}$. This output can be denoted as $\mathbf{M}' \in \mathbb{Z}_p^{k^3 \times k^3}$.

Function Class of Interest: We consider the following class of functions denoted by $\mathcal{F}_{n,p,d}$. Here $d = \mathsf{poly}(n)$ is some polynomial in n and p is the prime modulus over which computations are done and $3 \cdot n$ is the number of inputs.
Each function $f \in \mathcal{F}_{n,p,d}$ takes as input three vectors $(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z})$ where each vector is in $\mathbb{Z}_p^n$. Now f can be described using $d+1$ linear functions $f^0, ..., f^d$. f^0 is a matrix in $\mathbb{Z}_p^{n^3 \times n^3}$. For $\ell \neq 0$, f^ℓ is a linear function from $\mathbb{Z}_p^n$ to $\mathbb{Z}_p^{n^3 \times n^3}$ generated as follows:

- For every $i, j \in [n^3], \ell \in [1, d]$, let $\boldsymbol{v}_{i,j}^\ell$ be a vector $\in \mathbb{Z}_p^n$. Then define $[f^\ell(\boldsymbol{x})]_{i,j} = \langle \boldsymbol{v}_{i,j}^\ell, \boldsymbol{x} \rangle$ for $\ell \leq d$.

Thus the function descriptions above define matrices $[f^\ell(\boldsymbol{x})]$ for $\ell \in [1, d]$. In each of these matrices, every entry is a linear polynomial in $\boldsymbol{x}$. Now we define f as follows.

$$f(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}) = \langle f^0, [f^1(\boldsymbol{x})] \cdot \ldots \cdot [f^d(\boldsymbol{x})] \cdot \mathsf{diagvec}([\boldsymbol{y} \otimes \boldsymbol{z}]) \rangle$$

Here, $[f^i] \cdot [f^j]$ denote a matrix multiplication between matrices $[f^i]$ and $[f^j]$. For any matrices $A, B \in \mathbb{Z}_p^{m \times m}$, define $\langle A, B \rangle = \Sigma_{i,j} A_{i,j} \cdot B_{i,j}$ as standard inner product on the set of matrices.
Observe that this class of functions satisfy the template $f(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}) = L(g(\boldsymbol{x}), \boldsymbol{y} \otimes \boldsymbol{z})$. Reflecting on this, if $f(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}) = L(g(\boldsymbol{x}), \boldsymbol{y} \otimes \boldsymbol{z})$, where L is a multilinear degree two polynomial (with degree one in $\boldsymbol{y} \otimes \boldsymbol{z}$) and g is arbitrary, the class of functions described above captures the case when g comes from the set of branching programs of length d over $\mathbb{Z}_p$. If d is allowed to be a sufficiently large polynomial in n and p is large enough, then via barrington's theorem, it also contains the case when g is chosen from set of boolean NC^1 circuits ($\boldsymbol{x}$ being restricted to be a binary vector in this case). Note that to realize boolean NC^1 we need each matrix f^i to be an affine function of $\boldsymbol{x}$. This can be ensured by setting, say $x_1 = 1$.

We now describe our construction.

6.1 Construction

Ingredients: Our main ingredient is a secret-key function hiding canonical function-hiding inner product functional encryption cIPE (refer Section C for a definition).
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 PHFE 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 $\mathsf{pp} = (e, G_1, G_2, G_T, g_1, g_2, \mathbb{Z}_p)$.
- Run cIPE setup as follows. $\mathsf{sk}_0 \leftarrow \mathsf{cIPE.Setup}(\mathsf{pp}, 1^2)$. Thus these keys are used to encrypt vectors in $\mathbb{Z}_p^2$.
- Then run cIPE setup algorithm, for vectors of size 1^{n+2}. That is, compute $\mathsf{sk}_1 \leftarrow \mathsf{cIPE.Setup}(\mathsf{pp}, 1^{n+2})$.
- Sample $\boldsymbol{\beta}, \boldsymbol{\gamma} \leftarrow \mathbb{Z}_p^n$.
- Output $\mathsf{MSK} = \{\mathsf{sk}_0, \mathsf{sk}_1, \boldsymbol{\beta}, \boldsymbol{\gamma}\}$

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

- Parse $\mathsf{MSK} = \{\mathsf{sk}_0, \mathsf{sk}_1, \boldsymbol{\beta}, \boldsymbol{\gamma}\}$
- Sample $r \leftarrow \mathbb{Z}_p$.
- Compute $\mathsf{CT}_0 = \mathsf{cIPE.Enc}(\mathsf{sk}_0, (r, 0))$.
- Sample $\mathsf{sk}' \leftarrow \mathsf{cIPE.Setup}(\mathsf{pp}, 1^4)$.
- Compute $\mathsf{CTC}_j \leftarrow \mathsf{cIPE.Enc}(\mathsf{sk}', (y_j, \beta_j, 0, 0))$ for $j \in [n]$
- Compute $\mathsf{CTK}_k \leftarrow \mathsf{cIPE.KeyGen}(\mathsf{sk}', (z_k, -r\gamma_k, 0, 0))$ for $k \in [n]$.
- Compute $\mathsf{CTX} = \mathsf{cIPE.Enc}(\mathsf{sk}_1, (rx_1, ..., rx_n, r, 0))$.
- Output $\mathsf{CT} = (\boldsymbol{x}, \mathsf{CT}_0, \mathsf{CTX}, \{\mathsf{CTC}_j, \mathsf{CTK}_k, \}_{j,k \in [n]})$

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

- Parse $\mathsf{MSK} = \{\mathsf{sk}_0, \mathsf{sk}_1, \boldsymbol{\beta}, \boldsymbol{\gamma}\}$
- Sample $\mathbf{A}^1, ..., \mathbf{A}^d$ randomly from $\mathbb{Z}_p^{n^3 \times n^3}$. Note that $\mathbf{A}^1, ..., \mathbf{A}^d$ are chosen freshly for each function query f, we drop the function in the subscript for notational simplicity.
- Parse $f(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}) = \langle f^0, [f^1(\boldsymbol{x})] \cdot \ldots \cdot [f^d(\boldsymbol{x})] \cdot [\boldsymbol{y} \otimes \boldsymbol{z}] \rangle$.
- Let $\theta_f = \langle f^0, \mathbf{A}^1 \rangle$.
- Compute $\mathsf{Key}_{0,f} = \mathsf{cIPE.KeyGen}(\mathsf{sk}_0, (\theta_f, 0))$.
- Define $\mathbf{A}^{d+1} = \mathsf{diagvec}(\boldsymbol{\beta} \otimes \boldsymbol{\gamma})$.
- For $\ell \in [1, d]$, define $\mathbf{M}_f^\ell$ as the following matrix.

  $$\mathbf{M}_f^\ell = r[f^\ell(\boldsymbol{x})]\mathbf{A}^{\ell+1} - r\mathbf{A}^\ell$$

  Observe that there exists a vector $\boldsymbol{w}_{i,j}^\ell \in \mathbb{Z}_p^{n+2}$ such that $\langle \boldsymbol{w}_{i,j}^\ell, (rx_1, ..., rx_n, r, 0) \rangle = \mathbf{M}_{f,i,j}^\ell$. Note that each entry $\boldsymbol{w}_{i,j}^\ell$ is some linear function of $\mathbf{A}^{\ell+1}$. Note that the last component (component $n+2$) is just 0 and the second last component is $-\mathbf{A}_{i,j}^\ell$.

- For $\ell \in [1, d]$ and $i, j \in [n^3]$, compute $\mathsf{Key}^{\ell}_{f,i,j} = \mathsf{cIPE.KeyGen}(\mathsf{sk}_1, \boldsymbol{w}^{\ell}_{i,j})$
- Output $sk_f = (\mathsf{Key}_{0,f}, \{\mathsf{Key}^{\ell}_{f,i,j}\}_{\ell \in [d], i,j \in [n^3]})$

$\underline{\mathsf{Dec}(1^B, sk_f, \mathsf{CT})}$: In the description below, by Mon, we will denote all elements being encoded to target group.

- Parse $sk_f = (\mathsf{Key}_{0,f}, \{\mathsf{Key}^{\ell}_{f,i,j}\}_{\ell \in [d], i,j \in [n^3]})$.
- Parse $f(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}) = \langle f^0, [f^1(\boldsymbol{x})] \cdot \ldots \cdot [f^d(\boldsymbol{x})] \cdot \mathsf{diagvec}(\boldsymbol{y} \otimes \boldsymbol{z}) \rangle$.
- Parse $\mathsf{CT} = (\boldsymbol{x}, \mathsf{CT}_0, \mathsf{CTX}, \{\mathsf{CTC}_j, \mathsf{CTK}_k\}_{j,k \in [n]})$
- For every $i, j \in [n]$, compute $\mathsf{Int}^{d+1}_{i,j} = \mathsf{cIPE.Dec}(\mathsf{CTK}_i, \mathsf{CTC}_j) = [y_i \cdot z_j - r\beta_i \cdot \gamma_j]_T$.
- Compute $\mathsf{Mon}_0 = \mathsf{cIPE.Dec}(\mathsf{Key}_{0,f}, \mathsf{CT}_0) = [r\langle f^0, \mathbf{A}^1 \rangle]_T$.
- Compute $\mathsf{Int}^{\ell}_{i,j} = \mathsf{cIPE.Dec}(\mathsf{Key}^{\ell}_{f,i,j}, \mathsf{CTX}) = [\mathbf{M}^{\ell}_{f,i,j}]_T$ for $\ell \in [d], i, j \in [n^3]$.
- Now observe the following. Let $\mathbf{A}^{d+1} = \mathsf{diagvec}(\boldsymbol{\beta} \otimes \boldsymbol{\gamma})$ and $\mathbf{M}^{d+1}_f = \mathsf{diagvec}(\boldsymbol{y} \otimes \boldsymbol{z}) - r\mathbf{A}^{d+1}$. Denote.

$$S = [f^1(\boldsymbol{x})] \cdot \ldots \cdot [f^{d-1}(\boldsymbol{x})] \cdot [f^d(\boldsymbol{x})]\mathbf{M}^{d+1}_f + \ldots + [f^1(\boldsymbol{x})] \cdot \mathbf{M}^2_f + \mathbf{M}^1_f$$

  Here
$$S = [f^1(\boldsymbol{x})] \cdot \ldots \cdot [f^d(\boldsymbol{x})] \cdot \mathsf{diagvec}(\boldsymbol{y} \otimes \boldsymbol{z}) - r\mathbf{A}^1$$

- Thus one can compute encoding $\mathsf{Mon}_{i,j} = [S_{i,j}]_T$ for $i, j \in [n^3]$ from the knowledge of $\mathsf{Int}^{\ell}_{i,j}$ and $\boldsymbol{x}$.
- Compute $\mathsf{Mon}_f = \mathsf{Mon}_0 \cdot \Pi_{i,j \in [n^3]} \mathsf{Mon}^{f_{0,i,j}}_{i,j}$
- Observe that

$$\Pi_{i,j \in [n^3]} \mathsf{Mon}^{f_{0,i,j}}_{i,j} = [\langle f_0, [f^1(\boldsymbol{x})] \cdot \ldots \cdot [f^d(\boldsymbol{x})] \cdot \mathsf{diagvec}(\boldsymbol{y} \otimes \boldsymbol{z}) \rangle - r\langle f^0, \mathbf{A}^1 \rangle]_T$$

- Thus, $\mathsf{Mon}_f = [\langle f_0, [f_1(\boldsymbol{x})] \cdot \ldots \cdot [f_d(\boldsymbol{x})] \cdot \mathsf{diagvec}(\boldsymbol{y} \otimes \boldsymbol{z}) \rangle]_T$.
- 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:

$$\mathsf{CT} = (\boldsymbol{x}, \mathsf{CT}_0, \mathsf{CTX}, \{\mathsf{CTC}_j, \mathsf{CTK}_k\}_{j,k \in [n]})$$

CT_0 is a ciphertext for vectors of length 2. CTC_j and CTC_k are cIPE ciphertexts and keys for vector of length 4. Finally CTX is a cIPE ciphertext for vector of length $n + 2$. Thus, linear efficiency follows due to linear efficiency of cIPE.

Security Proof Here is our theorem.

Theorem 6. *Assuming SXDH holds relative to* PPGen*, the construction described in Section 6 satisfies semi-functional security.*

Due to lack of space we describe our proof in Section D.

7 Summing Up

In Section 5 we prove the following theorem.

Theorem 7. *Let* λ *be the security parameter. Let* $s(\lambda)$ *be any size parameter larger than any polynomial. Then assuming,*

- *Security of LWE with subexponential approximation factors against adversaries of size* $O(s)$*.*
- PHFE *for* $\mathcal{F}$ *secure against adversaries of size* $O(s)$*.*
- $\mathcal{F}$-$\Delta\mathsf{RG}$ *satisfying* $(O(s), \mathsf{adv})-$*perturbation resilience.*

Then there exists a output-sublinear FE *scheme satisfying* $(O(s), \mathsf{adv} + \mathsf{negl})-$*semi-functional security for circuit class* $\mathcal{C}_{n,\lambda}$*.*

In Section 6, we construct a PHFE scheme for arithmetic branching programs assuming SXDH holds over bilinear groups. Note that [33] gave various candidates for perturbation resilient generators for this class (refer Section B for details).

Invoking the amplification theorem in [6] and the result of [7, 19, 17, 34], we obtain the following theorem:

Theorem 8. *Assuming the following:*

- *Security of LWE against subexponential sized circuits.*
- *SXDH over bilinear maps against subexponential sized circuits*
- *Perturbation resilient generators,* $\mathcal{F}$-$\Delta\mathsf{RG}$ *satisfying* $(1 - 1/\lambda)-$*perturbation resilience against circuits of subexponential size.*

There exists an indistinguishability obfuscation scheme for P/poly

References

[1] Abdalla, M., Bourse, F., Caro, A.D., Pointcheval, D.: Simple functional encryption schemes for inner products. In: PKC. pp. 733–751 (2015)

[2] Agrawal, S.: Indistinguishability obfuscation without multilinear maps: New methods for bootstrapping and instantiation. In: Ishai, Y., Rijmen, V. (eds.) EUROCRYPT 2019, Part I. LNCS, vol. 11476, pp. 191–225. Springer, Heidelberg (May 2019)

[3] Agrawal, S., Libert, B., Stehlé, D.: Fully secure functional encryption for inner products, from standard assumptions. In: Robshaw, M., Katz, J. (eds.) CRYPTO 2016, Part III. LNCS, vol. 9816, pp. 333–362. Springer, Heidelberg (Aug 2016)

[4] Agrawal, S., Rosen, A.: Functional encryption for bounded collusions, revisited. In: TCC. pp. 173–205 (2017)

[5] Ananth, P., Jain, A., Lin, H., Matt, C., Sahai, A.: Indistinguishability obfuscation without multilinear maps: New paradigms via low degree weak pseudorandomness and security amplification. In: Boldyreva, A., Micciancio, D. (eds.) CRYPTO 2019, Part III. LNCS, vol. 11694, pp. 284–332. Springer, Heidelberg (Aug 2019)

[6] Ananth, P., Jain, A., Sahai, A.: Indistinguishability obfuscation without multilinear maps: io from lwe, bilinear maps, and weak pseudorandomness. IACR Cryptology ePrint Archive 2018, 615 (2018)

[7] Ananth, P., Jain, A.: Indistinguishability obfuscation from compact functional encryption. In: Gennaro, R., Robshaw, M.J.B. (eds.) CRYPTO 2015, Part I. LNCS, vol. 9215, pp. 308–326. Springer, Heidelberg (Aug 2015)

[8] Ananth, P., Sahai, A.: Projective arithmetic functional encryption and indistinguishability obfuscation from degree-5 multilinear maps. In: Coron, J., Nielsen, J.B. (eds.) EUROCRYPT 2017, Part I. LNCS, vol. 10210, pp. 152–181. Springer, Heidelberg (Apr / May 2017)

[9] Applebaum, B.: Pseudorandom generators with long stretch and low locality from random local one-way functions. SIAM J. Comput. 42(5), 2008–2037 (2013)

[10] Applebaum, B.: Exponentially-hard gap-CSP and local PRG via local hardcore functions. In: Umans, C. (ed.) 58th FOCS. pp. 836–847. IEEE Computer Society Press (Oct 2017)

[11] Applebaum, B., Ishai, Y., Kushilevitz, E.: Cryptography with constant input locality. In: Menezes, A. (ed.) CRYPTO 2007. LNCS, vol. 4622, pp. 92–110. Springer, Heidelberg (Aug 2007)

[12] Applebaum, B., Ishai, Y., Kushilevitz, E.: How to garble arithmetic circuits. In: Ostrovsky, R. (ed.) 52nd FOCS. pp. 120–129. IEEE Computer Society Press (Oct 2011)

[13] Applebaum, B., Kachlon, E.: Sampling graphs without forbidden subgraphs and almost-explicit unbalanced expanders. Electronic Colloquium on Computational Complexity (ECCC) 26, 11 (2019)

[14] Applebaum, B., Lovett, S.: Algebraic attacks against random local functions and their countermeasures. SIAM J. Comput. 47(1), 52–79 (2018)

[15] Barak, B., Brakerski, Z., Komargodski, I., Kothari, P.K.: Limits on low-degree pseudorandom generators (or: Sum-of-squares meets program obfuscation). In: Nielsen, J.B., Rijmen, V. (eds.) EUROCRYPT 2018, Part II. LNCS, vol. 10821, pp. 649–679. Springer, Heidelberg (Apr / May 2018)

[16] Barak, B., Goldreich, O., Impagliazzo, R., Rudich, S., Sahai, A., Vadhan, S.P., Yang, K.: On the (im)possibility of obfuscating programs. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 1–18. Springer, Heidelberg (Aug 2001)

[17] Bitansky, N., Nishimaki, R., Passelègue, A., Wichs, D.: From cryptomania to obfustopia through secret-key functional encryption. In: Hirt, M., Smith, A.D. (eds.) TCC 2016-B, Part II. LNCS, vol. 9986, pp. 391–418. Springer, Heidelberg (Oct / Nov 2016)

[18] Bitansky, N., Paneth, O., Rosen, A.: On the cryptographic hardness of finding a Nash equilibrium. In: Guruswami, V. (ed.) 56th FOCS. pp. 1480–1498. IEEE Computer Society Press (Oct 2015)

[19] Bitansky, N., Vaikuntanathan, V.: Indistinguishability obfuscation from functional encryption. In: Guruswami, V. (ed.) 56th FOCS. pp. 171–190. IEEE Computer Society Press (Oct 2015)

[20] Bitansky, N., Vaikuntanathan, V.: A note on perfect correctness by derandomization. In: Coron, J., Nielsen, J.B. (eds.) EUROCRYPT 2017, Part II. LNCS, vol. 10211, pp. 592–606. Springer, Heidelberg (Apr / May 2017)

[21] Brakerski, Z., Gentry, C., Vaikuntanathan, V.: (leveled) fully homomorphic encryption without bootstrapping. In: Innovations in Theoretical Computer Science 2012, Cambridge, MA, USA, January 8-10, 2012. pp. 309–325 (2012)

[22] Brakerski, Z., Vaikuntanathan, V.: Fully homomorphic encryption from ring-LWE and security for key dependent messages. In: Rogaway, P. (ed.) CRYPTO 2011. LNCS, vol. 6841, pp. 505–524. Springer, Heidelberg (Aug 2011)

[23] Brzuska, C., Farshim, P., Mittelbach, A.: Indistinguishability obfuscation and UCEs: The case of computationally unpredictable sources. In: Garay, J.A., Gennaro, R. (eds.) CRYPTO 2014, Part I. LNCS, vol. 8616, pp. 188–205. Springer, Heidelberg (Aug 2014)

[24] Cohen, A., Holmgren, J., Nishimaki, R., Vaikuntanathan, V., Wichs, D.: Watermarking cryptographic capabilities. In: STOC (2016)

[25] Garg, S., Gentry, C., Halevi, S., Raykova, M., Sahai, A., Waters, B.: Candidate indistinguishability obfuscation and functional encryption for all circuits. In: 54th FOCS. pp. 40–49. IEEE Computer Society Press (Oct 2013)

[26] Garg, S., Pandey, O., Srinivasan, A.: Revisiting the cryptographic hardness of finding a nash equilibrium. In: Robshaw, M., Katz, J. (eds.) CRYPTO 2016, Part II. LNCS, vol. 9815, pp. 579–604. Springer, Heidelberg (Aug 2016)

[27] Goldreich, O.: Candidate one-way functions based on expander graphs. Electronic Colloquium on Computational Complexity (ECCC) 7(90) (2000)

[28] Goldwasser, S., Gordon, S.D., Goyal, V., Jain, A., Katz, J., Liu, F.H., Sahai, A., Shi, E., Zhou, H.S.: Multi-input functional encryption. In: Nguyen, P.Q., Oswald, E. (eds.) EUROCRYPT 2014. LNCS, vol. 8441, pp. 578–602. Springer, Heidelberg (May 2014)

[29] Goldwasser, S., Rothblum, G.N.: Securing computation against continuous leakage. In: Rabin, T. (ed.) CRYPTO 2010. LNCS, vol. 6223, pp. 59–79. Springer, Heidelberg (Aug 2010)

[30] Gorbunov, S., Vaikuntanathan, V., Wee, H.: Predicate encryption for circuits from LWE. In: Gennaro, R., Robshaw, M.J.B. (eds.) CRYPTO 2015, Part II. LNCS, vol. 9216, pp. 503–523. Springer, Heidelberg (Aug 2015)

[31] Hofheinz, D., Jager, T., Khurana, D., Sahai, A., Waters, B., Zhandry, M.: How to generate and use universal samplers. In: Cheon, J.H., Takagi, T. (eds.) ASIACRYPT 2016, Part II. LNCS, vol. 10032, pp. 715–744. Springer, Heidelberg (Dec 2016)

[32] Hohenberger, S., Sahai, A., Waters, B.: Full domain hash from (leveled) multilinear maps and identity-based aggregate signatures. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part I. LNCS, vol. 8042, pp. 494–512. Springer, Heidelberg (Aug 2013)

[33] Jain, A., Lin, H., Matt, C., Sahai, A.: How to leverage hardness of constant-degree expanding polynomials overa $\mathbb{R}$ to build $i\mathcal{O}$. In: Ishai, Y., Rijmen, V. (eds.) EUROCRYPT 2019, Part I. LNCS, vol. 11476, pp. 251–281. Springer, Heidelberg (May 2019)

[34] Kitagawa, F., Nishimaki, R., Tanaka, K.: Obfustopia built on secret-key functional encryption. In: Nielsen, J.B., Rijmen, V. (eds.) EUROCRYPT 2018, Part II. LNCS, vol. 10821, pp. 603–648. Springer, Heidelberg (Apr / May 2018)

[35] Koppula, V., Lewko, A.B., Waters, B.: Indistinguishability obfuscation for turing machines with unbounded memory. In: STOC (2015)

[36] Kothari, P.K., Mori, R., O'Donnell, R., Witmer, D.: Sum of squares lower bounds for refuting any CSP. In: Hatami, H., McKenzie, P., King, V. (eds.) 49th ACM STOC. pp. 132–145. ACM Press (Jun 2017)

[37] Lin, H.: Indistinguishability obfuscation from constant-degree graded encoding schemes. In: Fischlin, M., Coron, J.S. (eds.) EUROCRYPT 2016, Part I. LNCS, vol. 9665, pp. 28–57. Springer, Heidelberg (May 2016)

[38] Lin, H.: Indistinguishability obfuscation from SXDH on 5-linear maps and locality-5 PRGs. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017, Part I. LNCS, vol. 10401, pp. 599–629. Springer, Heidelberg (Aug 2017)

[39] Lin, H., Matt, C.: Pseudo flawed-smudging generators and their application to indistinguishability obfuscation. IACR Cryptology ePrint Archive 2018, 646 (2018)

[40] Lin, H., Tessaro, S.: Indistinguishability obfuscation from trilinear maps and block-wise local PRGs. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017, Part I. LNCS, vol. 10401, pp. 630–660. Springer, Heidelberg (Aug 2017)

[41] Lin, H., Vaikuntanathan, V.: Indistinguishability obfuscation from DDH-like assumptions on constant-degree graded encodings. In: Dinur, I. (ed.) 57th FOCS. pp. 11–20. IEEE Computer Society Press (Oct 2016)

[42] Lombardi, A., Vaikuntanathan, V.: Limits on the locality of pseudorandom generators and applications to indistinguishability obfuscation. In: Kalai, Y., Reyzin, L. (eds.) TCC 2017, Part I. LNCS, vol. 10677, pp. 119–137. Springer, Heidelberg (Nov 2017)

[43] Micciancio, D., Peikert, C.: Trapdoors for lattices: Simpler, tighter, faster, smaller. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 700–718. Springer, Heidelberg (Apr 2012)

[44] Mossel, E., Shpilka, A., Trevisan, L.: On e-biased generators in NC0. In: FOCS. pp. 136–145 (2003)

[45] Mossel, E., Shpilka, A., Trevisan, L.: On e-biased generators in NC0. In: 44th FOCS. pp. 136–145. IEEE Computer Society Press (Oct 2003)

[46] O'Donnell, R., Witmer, D.: Goldreich's PRG: evidence for near-optimal polynomial stretch. In: CCC. pp. 1–12 (2014)

[47] Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. In: STOC. pp. 84–93 (2005)

[48] Sahai, A., Waters, B.: How to use indistinguishability obfuscation: deniable encryption, and more. In: STOC. pp. 475–484 (2014)

[49] Wee, H.: Attribute-hiding predicate encryption in bilinear groups, revisited. In: TCC. pp. 206–233 (2017)

A Semi-functional Security for Functional Encryption

We define the following auxiliary algorithms.

Semi-functional Key Generation, $\mathsf{sfKG}(\mathsf{MSK}, C, \theta)$: On input the master secret key MSK, function $C \in \mathcal{C}_\lambda$ and a value θ, it computes the semi-functional key $sk_{C,\theta}$.

Semi-functional Encryption, $\mathsf{sfEnc}(\mathsf{MSK}, 1^\lambda)$: On input the master encryption key MSK, and the length 1^λ, 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 key property.

Definition 7 (s-Indistinguishability of Semi-functional Key)**.** *An FE scheme for circuits* FE *for a class of functions* $\mathcal{C} = \{\mathcal{C}_\lambda\}_{\lambda \in \mathbb{N}}$ *is said to satisfy* $s-$**indistinguishability of semi-functional key property** *if for sufficiently large* $\lambda \in \mathbb{N}$*, for any adversary* $\mathcal{A}$ *of size* s*, the probability that* $\mathcal{A}$ *succeeds in the following experiment bounded by* negl*.*
$\mathsf{Expt}(1^\lambda, \mathbf{b})$*:*

1. $\mathcal{A}$ *specifies the following:*
   - *It can specify messages* $M_j = \{x_j\}$ *for* $j \in [q]$ *for any polynomial* q*. Here each* $M_j \in \chi_\lambda$*.*
   - *It specifies function queries as follows:*
     - *It specifies* $C \in \mathcal{C}_\lambda$*.*
     - *It specifies values* θ *in output space of* C*.*

2. *The challenger computes the following:*
   - $\mathsf{MSK} \leftarrow \mathsf{Setup}(1^\lambda)$
   - $\mathsf{CT}_j \leftarrow \mathsf{Enc}(\mathsf{MSK}, M_j)$*, for every* $j \in [q]$*.*
   - *If* $\mathbf{b} = 0$*, compute* $sk^*_C \leftarrow \mathsf{KeyGen}(\mathsf{MSK}, C)$*. Otherwise, compute* $sk^*_C \leftarrow \mathsf{sfKG}(\mathsf{MSK}, C, \theta_i)$*.*
3. *Challenger sends* $\{\mathsf{CT}_i\}_{i \in q}$ *and* $\{sk^*_C\}$ *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 8 (Indistinguishability of Semi-functional Ciphertexts)**.** *For an FE scheme* FE *for a class of functions* $\mathcal{C} = \{\mathcal{C}_\lambda\}_{\lambda \in \mathbb{N}}$ *, the* $(s.\mathsf{adv})-$ **indistinguishability of semi-functional ciphertexts property** *is associated with two experiments. The experiments are parameterised with* $\mathsf{aux} = (1^\lambda, \Gamma, M_i = \{x_i\}_{i \in \Gamma}, M^* = x, C)$

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

1. *The challenger sets* $\theta = C(x)$*. The challenger computes the following:*
2. *Compute* $\mathsf{MSK} \leftarrow \mathsf{Setup}(1^\lambda)$*.*
3. *Compute* $sk_{C,\theta} \leftarrow \mathsf{sfKG}(\mathsf{MSK}, C, \theta)$*.*
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* $\mathsf{CT}^* \leftarrow \mathsf{sfEnc}(\mathsf{MSK}, 1^\lambda)$*.*
7. *Output the following:*
   (a) CT_i *for* $i \in \Gamma$ *and* CT^**.*
   (b) $sk_{C,\theta}$*.*
   (c) M^* *and* $\{M_i\}_{i \in \Gamma}$
   (d) C

An FE scheme FE *associated with plaintext space* χ *is said to satisfy* (s, adv)**-indistinguishability of semi-functional ciphertexts property** *if the following happens:* $\forall \lambda > \lambda_0$*, any polynomial* Γ*, messages* $\{M_i\}_{i \in \Gamma} \in \chi_\lambda$*,* $M^* \in \chi_\lambda$ *,* $C \in \mathcal{C}_\lambda$ *and any adversary* $\mathcal{A}$ *of size* s*:*

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

where $\mathsf{aux} = (1^\lambda, \Gamma, M_i = \{x_i\}_{i \in \Gamma}, M^* = x, C)$

Definition 9 (Semi-functional Security)**.** *Consider an FE scheme* FE *for a class of circuits* $\mathcal{C}_\lambda$*. We say that* FE *satisfies* $(s, \mathsf{adv})-$**semi-functional security** *if it satisfies* $(s, \mathsf{adv})-$*indistinguishability of semi-functional ciphertexts property (Definition 8) and* $s-$*indistinguishability of semi-functional key property (Definition 7).*

Remark: We now remark that for $i\mathcal{O}$, we need size s to be greater than 2^{λ^c} for some constant $c > 0$, and adv to be at most $1 - 1/\mathsf{poly}(\lambda)$ for any fixed polynomial poly independent of the output length of $\mathcal{C}$. This is because of the amplification theorem proved in [6]. In order to construct standard sublinear FE, we need adv security to hold against any polynomial time adversary.

B Pseudorandomness Assumptions Implying Obfuscation

Now, we recall the new pseudorandomness assumptions of perturbation resilient generator (ΔRG) that were made initially by the work of [6, 5] and then developed in the work of [33]. In particular, [6, 5] made degree-3 version of the assumption described below. Later [33], obtained obfuscation if the assumption below holds for any constant degree $D \geq 3$. Below we present an assumption that is an instantiation of the general assumption with specific parameters so that the assumption is simplified with least number of parameters. This assumption is sufficient to imply $i\mathcal{O}$. The assumption has the following specification:

- The distinguishing advantage is set to be bounded by 0.99 as opposed to $1 - 1/\mathsf{poly}(\lambda)$.
- The magnitude of the perturbations handled are bounded by n. We can set parameters for our SHE scheme so that this sufficies.
- We set $D \geq 4$.

Definition 10 (Constant-Degree Multivariate Polynomial Samplers)**.** *For any integer constant $D \geq 4$, $\epsilon > 0$ and a parameter n, we let $\mathcal{Q}[D, n, \epsilon]$ be shorthand for an arbitrary "polynomial sampler" that outputs $\lfloor n^{1+\epsilon} \rfloor$ polynomials*

$$q_1, \ldots, q_{\lfloor n^{1+\epsilon} \rfloor} \leftarrow \mathcal{Q}[D, n, \epsilon]$$

where each polynomial q_k takes in n variables $e_1, \ldots, e_n$, and each monomial has total degree (at most) D over the e_i variables.

We will typically use the shorthand $\boldsymbol{e} = (e_1, \ldots, e_n)$ and write these polynomials as $q_k(\boldsymbol{e})$. When the inputs are suppressed, q_k will denote the vector of coefficients of q_k.

Definition 11 (LWE with leakage)**.** *For any $p, \chi, n, \epsilon, D, \mathcal{Q}$, let $\mathsf{Dist}_{leak}[p, \chi, n, \epsilon, D, \mathcal{Q}]$ be the following sampling procedure. For each $i \in [n]$, sample*

- *LWE coefficient vector $\boldsymbol{a}_i \leftarrow \mathbb{Z}_p^{n^{2/D}}$,*
- *LWE error $e_i \leftarrow \chi$.*

Sample a random secret vector $\boldsymbol{s} \leftarrow \mathbb{Z}_p^{n^{2/D}}$. Next, sample "leakage" polynomials $(q_1, \ldots, q_{\lfloor n^{1+\epsilon} \rfloor}) \leftarrow \mathcal{Q}[D, n, \epsilon]$. Let $\boldsymbol{e} := (e_1, \ldots, e_n)$, Output

$$\{\boldsymbol{a}_i, \langle \boldsymbol{a}_i, \boldsymbol{s} \rangle + e_i \ mod\ p\}_{i \in [n]}, \{q_k, q_k(\boldsymbol{e})\}_{k \in [n^{1+\epsilon}]}.$$

The following distribution is defined almost identically to Dist_{leak}, except that it additionally takes perturbation values $\{\delta_i\}_{i \in [n^{1+\epsilon}]}$. The differences between definition 11 are highlighted in red.

Definition 12 (LWE with δ-perturbed leakage)**.** *For any $p, \chi, n, \epsilon, D, \mathcal{Q}$ and bounded integer perturbations[5] $\{\delta_i\}_{i \in [n^{1+\epsilon}]}$ where $|\delta_i| < n$ for all $i \in [n^{1+\epsilon}]$, let $\mathsf{Dist}_{pert}[p, \chi, n, \epsilon, \mathcal{Q}, \{\delta_i\}]$ be the following sampling procedure. For each $i \in [n]$, sample*

[5]In our SHE construction, parameters can be set so that perturbation resilience is only required for magintudes upto n as opposed to a general polynomial.

- *LWE coefficient vector* $\boldsymbol{a}_i \leftarrow \mathbb{Z}_p^{n^{2/D}}$,
- *LWE error* $e_i \leftarrow \chi$.

Sample a random secret vector $\boldsymbol{s} \leftarrow \mathbb{Z}_p^{n^{2/D}}$. *Next, sample "leakage" polynomials* $(q_1, \ldots, q_{\lfloor n^{1+\epsilon} \rfloor}) \leftarrow \mathcal{Q}[D, n, \epsilon]$. *Let* $\mathbf{e} := (e_1, \ldots, e_n)$. *Output*

$$\{\boldsymbol{a}_i, \langle \boldsymbol{a}_i, \boldsymbol{s} \rangle + e_i \ mod\ p\}_{i \in [n]}, \{q_k, q_k(\boldsymbol{e}) + 2 \cdot \delta_k\}_{k \in [n^{1+\epsilon}]}.$$

We can now state our main assumption.

Main Assumption. Let $\epsilon > 0$ be some constant. Consider the following setting of parameters:

1. Let χ be the discrete gaussian random variable with mean 0 and variance n^2.
2. p is a $O(2^{n^{\epsilon/2}})$ prime.

The assumption states that for all degree $D \geq 4$, there exists an $\epsilon > 0$ such that for all large enough natural number $n > N_{D,\epsilon}$ and any $\delta_1, ..., \delta_{n^{1+\epsilon}} \in [-n, n]$ and for any adversary $\mathcal{A}$ of size $2^{n^{O(\epsilon)}}$,

$$\left| \Pr_{Z \xleftarrow{\$} \mathcal{D}_{leak}[p,\chi,n,\epsilon,\mathcal{Q}]} [\mathcal{A}(Z) = 1] - \Pr_{Z \xleftarrow{\$} \mathcal{D}_{pert}[p,\chi,n,\epsilon,\mathcal{Q},\{\delta_i\}]} [\mathcal{A}(Z) = 1] \right| \leq 0.99.$$

Below we recall two concrete candidates for instantiating polynomial sampler that were proposed by [33].

B.1 Candidates

Let $n \in \mathbb{N}$ be a natural number. We will consider multivariate polynomials $f : \{-1, +1\}^n \to \mathbb{R}$ which can be represented as $f(\boldsymbol{x}) = \sum_{S \subseteq [n]} c_S \prod_{i \in S} x_i$. In this notation, $\boldsymbol{x} = (x_1, ..., x_n)$ is the input. We will refer by $X_S = \prod_{i \in S} x_i$. Therefore in this notation $f(\boldsymbol{x}) = \sum_S c_S \cdot X_S$. We also refer to c_S for any set $S \subset [n]$ as the fourier coefficient of f with respect to S. One can compute it as: $c_S = \frac{1}{2^n} \cdot \sum_{\boldsymbol{x} \in \{-1,1\}^n} f(\boldsymbol{x}) \cdot X_S$.
Next, we describe a linear transform $\phi : \{-1, +1\} \to \{0, 1\}$ given by $\phi(x) \to (1 - x)/2$. By $\tilde{x}$ we will denote $\phi(x)$. Observe that for any input $\boldsymbol{x} = (x_1, ..., x_n) \in \{-1, +1\}^n$. it holds that $\phi(X_S) = \bigoplus_{i \in S} \tilde{x}_i$. The intuition is really simple and relates the boolean operation of $\oplus$ over inputs from $\{0, 1\}$ to the product of inputs from $\{+1, -1\}$. The idea is that there is a natural correspondance between:

- 0 in the *xor* notation (boolean variable False) with 1 in $\{+1, -1\}$ notation.
- 1 in the *xor* notation (boolean variable True) with -1 in $\{+1, -1\}$ notation.

B.1.1 D-XOR Based Candidate

Now we recall the $D-$XOR based candidate proposed by [33]. For a more natural definition, we treat $\boldsymbol{x}$ as the variable representing input to the polynomials and $\boldsymbol{e}$ as the planted input.
Polynomial Sampler- D-XOR Based:

- On input 1^n and $\epsilon > 0$, sample randomly sets $S_{i,j} \subset [n]$ of size $|S_{i,j}| = D$ for $i \in [n^{1+\epsilon}]$, $j \in [n^{D/2-1-2\epsilon}]$.
- For all $i \in [n^{1+\epsilon}]$, define $q_i(\boldsymbol{x}) = \sum_{j\in[n^{D/2-1-2\cdot\epsilon}]} c_{i,j} \prod_{\ell\in S_{i,j}} x_\ell$, where coefficients $c_{i,j}$ are chosen at random from $\{+1,-1\}$.
- Output $q_1, ..., q_{n^{1+\epsilon}}$. Observe that we must have $1+2\epsilon < D/2$ in order to instantiate this candidate.

The reason it is called D-XOR based candidate is that each polynomial is a signed sum of random degree $D-$monomials, and, as noted above, each monomial has a correspondance to *xor* of D inputs in the boolean $\{0,1\}$ notation. Although, this (as well as the next candidate) is inspired from boolean functions, for constructing indistinguishability obfuscation, they are evaluated over some set like the set of integers $\mathbb{Z}$ or reals $\mathbb{R}$. We view these candidates as polynomials being evaluated over $\mathbb{R}$ depending on the context.

B.1.2 TSA Based Candidate

[33] also proposed a candidate that is inspired from the *Tri-Sum-And* predicate given by:

$$\mathsf{TSA} : \{0,1\}^5 \rightarrow \{0,1\}$$
$$\mathsf{TSA}(\tilde{x}_1, ..., \tilde{x}_5) = \tilde{x}_1 \oplus \tilde{x}_2 \oplus \tilde{x}_3 \oplus \tilde{x}_4 \cdot \tilde{x}_5$$

This predicate has been extensively studied, and is used to instantiate goldreich's PRG. For an input of length n, it can provide a stretch of upto $n^{1.5-\epsilon}$ for any constant $\epsilon > 0$ [45, 46, 14].
We can also write the predicate above in $\{-1,+1\}$ notation as follows:

$$\mathsf{TSA} : \{-1,+1\}^5 \rightarrow \{-1,+1\}$$
$$\mathsf{TSA}(x_1,x_2,x_3,x_4,x_5) = x_1 \cdot x_2 \cdot x_3 \cdot \left(1 - \frac{(1-x_4)\cdot(1-x_5)}{2}\right)$$

Note that:

$$2 \cdot \mathsf{TSA}(x_1,x_2,x_3,x_4,x_5) = x_1 \cdot x_2 \cdot x_3 \cdot (1 - x_4 - x_5 + x_4 \cdot x_5)$$

We call this function as $\mathsf{2TSA}$. For any ordered set $S \subset [n]$ of size 5, we denote by $\mathsf{2TSA}(\{x_i\}_{i\in S})$, the predicate evaluated on inputs $\{x_i\}_{i\in S}$. If the inputs are not in {-1,+1} but rather lie in $\mathbb{Z}$, from the context it will mean that the polynomial is evaluated over the reals.
Polynomial Sampler- TSA Based:

- On input 1^n and $\epsilon > 0$, sample randomly (ordered) sets $S_{i,j} \subset [n]$ of size $|S_{i,j}| = 5$ for $i \in [n^{1+\epsilon}]$, $j \in [n^{0.5-2\epsilon}]$ [6].
- For all $i \in [n^{1+\epsilon}]$, define $q_i(\boldsymbol{x}) = \sum_{j\in[n^{0.5-2\cdot\epsilon}]} c_{i,j} \cdot \mathsf{2TSA}(\{x_\ell\}_{\ell\in S_{i,j}})$, where coefficients $c_{i,j}$ are chosen at random from $\{+1,-1\}$.

[6]This bound comes from [36] lowerbound.

- Output $q_1, ..., q_{n^{1+\epsilon}}$.

The difference between this and the previous candidate is that the monomials are now replaced with arithmetized versions of the $\mathsf{2TSA}$ function. Similarly, one can propose more boolean PRG inspired candidates by replacing $\mathsf{2TSA}$ with some other PRG predicate.

C Canonical Function Hiding Inner Product FE

We now describe the notion of a canonical function hiding inner product FE proposed by [38]. 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 $\mathbb{Z}_p$.
- $\mathsf{Setup}(\mathsf{pp}, 1^n) \to \mathsf{sk}$. The setup algorithm takes as input the length of vector 1^n and parameters pp and outputs a secret key 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).
- $\mathsf{Enc}(\mathsf{sk}, \boldsymbol{x}) \to \mathsf{CT}$. The encryption algorithm takes as input a vector $x \in \mathbb{Z}_p^n$ and outputs a ciphertext CT.
- $\mathsf{KeyGen}(\mathsf{sk}, \boldsymbol{y}) \to \mathsf{sk}_{\boldsymbol{y}}$. The key generation algorithm on input the master secret key sk and a function vector $y \in \mathbb{Z}_p^n$ and outputs a function key $\mathsf{sk}_{\boldsymbol{y}}$
- $\mathsf{Dec}(1^B, \mathsf{sk}_{\boldsymbol{y}}, \mathsf{CT}) \to m^*$. The decryption algorithm takes as input a ciphertext CT, a function key $\mathsf{sk}_{\boldsymbol{y}}$ 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. $\mathsf{Setup}(\mathsf{pp}, 1^n) \to \mathsf{sk}$. Fix any $x, y \in \mathbb{Z}_p^n$
3. $\mathsf{KeyGen}(\mathsf{sk}, \boldsymbol{y}) \to \mathsf{sk}_{\boldsymbol{y}}$.
4. $\mathsf{Enc}(\mathsf{sk}, \boldsymbol{x}) \to \mathsf{CT}$
5. $\mathsf{Dec}(1^B, \mathsf{sk}_{\boldsymbol{y}}, \mathsf{CT}) = \theta$

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 $(\boldsymbol{x}, \boldsymbol{y}) \in \mathbb{Z}_p^n$ the following happens:

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

The size of the circuit computing CT and $\mathsf{sk}_{\boldsymbol{y}}$ is less than $n \log_2 p \cdot \mathsf{poly}'(\lambda) < n\mathsf{poly}(\lambda)$. 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 $\mathbb{Z}_p$ 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 [38].

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 $\mathsf{PPSetup}(1^\lambda) \rightarrow \mathsf{pp}$.
2. Run $\mathsf{Setup}(\mathsf{pp}, 1^n) \rightarrow (\mathsf{pp}, \mathsf{sk})$.
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 $L = \mathsf{poly}(\lambda)$. Here each vector is in $\mathbb{Z}_p^n$. 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 $\mathsf{CT}_i = \mathsf{Enc}(\mathsf{sk}, \boldsymbol{x}_i^b)$ for $i \in [L]$ and $\mathsf{sk}_i = \mathsf{KeyGen}(\mathsf{sk}, \boldsymbol{y}_i^b)$ for $i \in [L]$.
5. Output $\{\mathsf{CT}_i, \mathsf{sk}_i\}_{i \in [L]}$.

Theorem 9 (Imported Theorem [38])**.** *Assuming subexponential SXDH holds relative to* PPGen*, there exists a subexponential canonical function hiding inner product functional encryption scheme.*

D Security Proof: PHFE Construction

Here is our theorem.

Theorem 10. *Assuming SXDH holds relative to* PPGen*, the construction described in Section 6 satisfies semi-functional security.*

We now describe semi-functional algorithms.

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**].

On input the master secret key MSK and function f,

- Parse $\mathsf{MSK} = \{\mathsf{sk}_0, \mathsf{sk}_1, \boldsymbol{\beta}, \boldsymbol{\gamma}\}$
- Sample $\mathbf{A}^1, ..., \mathbf{A}^d$ randomly from $\mathbb{Z}_p^{n^3 \times n^3}$
- Parse $f(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}) = \langle f^0, [f^1(\boldsymbol{x})] \cdot \ldots \cdot [f^d(\boldsymbol{x})] \cdot \mathsf{diagvec}(\boldsymbol{y} \otimes \boldsymbol{z})\rangle$.
- Let $\theta_f = \langle f^0, \mathbf{A}^1\rangle$.
- [**Change**] Compute $\mathsf{Key}_{0,f} = \mathsf{cIPE.KeyGen}(\mathsf{sk}_0, (\theta_f, \Delta))$.
- Define $\mathbf{A}^{d+1} = \mathsf{diagvec}(\boldsymbol{\beta} \otimes \boldsymbol{\gamma})$.
- For $\ell \in [1, d]$, define $\mathbf{M}_f^\ell$ as the following matrix.

  $$\mathbf{M}_f^\ell = r[f^\ell(\boldsymbol{x})]\mathbf{A}^{\ell+1} - r\mathbf{A}^\ell$$

  Observe that there exists a vector $\boldsymbol{w}_{i,j}^\ell \in \mathbb{Z}_p^{n+2}$ such that $\langle \boldsymbol{w}_{i,j}^\ell, (rx_1, ..., rx_n, r, 0)\rangle = \mathbf{M}_{f,i,j}^\ell$. Note that each entry $\boldsymbol{w}_{i,j}^\ell$ is some linear function of $\mathbf{A}^{\ell+1}$. Note that the last component (component $n+2$) is just 0 and the second last component is $-\mathbf{A}_{i,j}^\ell$.
- For $\ell \in [1, d]$ and $i, j \in [n]$, compute $\mathsf{Key}_{f,i,j}^\ell = \mathsf{cIPE.KeyGen}(\mathsf{sk}_1, \boldsymbol{w}_{i,j}^\ell)$
- Output $sk_f = (\mathsf{Key}_{0,f}, \{\mathsf{Key}_{f,i,j}^\ell\}_{\ell \in [d], i,j \in [n^3]})$

We now describe semi-functional encryption algorithm:

$\underline{\mathsf{sfEnc}(\mathsf{MSK}, \boldsymbol{x}, 1^n)}$: On input the public attribute $\boldsymbol{x} \in \mathbb{Z}_p^n$ and MSK. Perform the following operations. The change from regular encryption algorithm is marked with boldfaced [**Change**].

- Parse $\mathsf{MSK} = \{\mathsf{sk}_0, \mathsf{sk}_1, \boldsymbol{\beta}, \boldsymbol{\gamma}\}$
- Sample $r \leftarrow \mathbb{Z}_p$.
- [**Change**] Compute $\mathsf{CT}_0 = \mathsf{cIPE.Enc}(\mathsf{sk}_0, (r, 1))$.
- Sample $\mathsf{sk}' \leftarrow \mathsf{cIPE.Setup}(\mathsf{pp}, 1^4)$.
- [**Change**] Compute $\mathsf{CTC}_j \leftarrow \mathsf{cIPE.Enc}(\mathsf{sk}', (0, \beta j, 0, 0))$ for $j \in [n]$

- [**Change**] Compute $\mathsf{CTK}_k \leftarrow \mathsf{cIPE.KeyGen}(\mathsf{sk}', (0, -r\gamma_k, 0, 0))$ for $k \in [n]$.
- Compute $\mathsf{CTX} = \mathsf{cIPE.Enc}(\mathsf{sk}_1, (rx_1, ..., rx_n, r, 0))$.
- Output $\mathsf{CT} = (\boldsymbol{x}, \mathsf{CT}_0, \mathsf{CTX}, \{\mathsf{CTC}_j, \mathsf{CTK}_k, \}_{j,k \in [n]})$

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 6, 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 following form. The only difference lies in the component $\mathsf{Key}_{0,f}$. If challenge bit is 0, it is generated as $\mathsf{cIPE.KeyGen}(\mathsf{sk}_0, (\theta_f, 0))$ otherwise as $\mathsf{cIPE.KeyGen}(\mathsf{sk}_0, (\theta_f, \Delta))$. 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. $\square$

Theorem 11. *Assuming SXDH holds relative to* PPGen*, the scheme described in Section 6, satisfies indistinguishability of semi-functional ciphertexts property.*

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. The change in each intermediate hybrid is marked with boldfaced [**Change**].

Hybrid$_0$:

1. The challenger does setup for the scheme as follows:
   - Sample $\mathsf{pp} \leftarrow \mathsf{cIPE.PPSetup}(1^\lambda)$. Let us assume $\mathsf{pp} = (e, G_1, G_2, G_T, g_1, g_2, \mathbb{Z}_p)$.
   - Run cIPE setup as follows. $\mathsf{sk}_0 \leftarrow \mathsf{cIPE.Setup}(\mathsf{pp}, 1^2)$. Thus these keys are used to encrypt vectors in $\mathbb{Z}_p^2$.
   - Then run cIPE setup algorithm, for vectors of size 1^{n+2}. That is, compute $\mathsf{sk}_1 \leftarrow \mathsf{cIPE.Setup}(\mathsf{pp}, 1^{n+2})$.
   - Sample $\boldsymbol{\beta}, \boldsymbol{\gamma} \leftarrow \mathbb{Z}_p^n$.
   - Output $\mathsf{MSK} = \{\mathsf{sk}_0, \mathsf{sk}_1, \boldsymbol{\beta}, \boldsymbol{\gamma}\}$
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 $\mathbb{Z}_p^{3 \cdot n}$. We now describe how the challenge ciphertext CT is generated. Other ciphertexts $\mathsf{CT}(i)$ for $i \in [L]$ are generated similarly. Note that for ciphertext for message $m(i)$, the components will be denoted as: $\mathsf{CT}(i)_0, \mathsf{CTX}(i), \{\mathsf{CTC}(i)_j, \mathsf{CTK}(i)_k\}_{j,k}$
   - Parse $\mathsf{MSK} = \{\mathsf{sk}_0, \mathsf{sk}_1, \boldsymbol{\beta}, \boldsymbol{\gamma}\}$

- Sample $r \leftarrow \mathbb{Z}_p$.
- Compute $\mathsf{CT}_0 = \mathsf{cIPE.Enc}(\mathsf{sk}_0, (r, 0))$.
- Sample $\mathsf{sk}' \leftarrow \mathsf{cIPE.Setup}(\mathsf{pp}, 1^4)$.
- Compute $\mathsf{CTC}_j \leftarrow \mathsf{cIPE.Enc}(\mathsf{sk}', (y_j, \beta j, 0, 0))$ for $j \in [n]$
- Compute $\mathsf{CTK}_k \leftarrow \mathsf{cIPE.KeyGen}(\mathsf{sk}', (z_k, -r\gamma_k, 0, 0))$ for $k \in [n]$.
- Compute $\mathsf{CTX} = \mathsf{cIPE.Enc}(\mathsf{sk}_1, (rx_1, ..., rx_n, r, 0))$.
- Output $\mathsf{CT} = (\boldsymbol{x}, \mathsf{CT}_0, \mathsf{CTX}, \{\mathsf{CTC}_j, \mathsf{CTK}_k, \}_{j,k\in[n]})$

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 associated with value Δ.

   - Parse $\mathsf{MSK} = \{\mathsf{sk}_0, \mathsf{sk}_1, \boldsymbol{\beta}, \boldsymbol{\gamma}\}$
   - Sample $\mathbf{A}^1, ..., \mathbf{A}^d$ randomly from $\mathbb{Z}_p^{n\times n}$. This step is done for each query. We omit subscript of the function (e.g. $\mathbf{A}^\ell_f$) untill necessary.
   - Parse $f(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}) = \langle f^0, [f^1(\boldsymbol{x})] \cdot \ldots \cdot [f^d(\boldsymbol{x})] \cdot \mathsf{diagvec}(\boldsymbol{y} \otimes \boldsymbol{z})\rangle$.
   - Let $\theta_f = \langle f^0, \mathbf{A}^1\rangle$.
   - Compute $\mathsf{Key}_{0,f} = \mathsf{cIPE.KeyGen}(\mathsf{sk}_0, (\theta_f, \Delta))$.
   - Define $\mathbf{A}^{d+1} = \mathsf{diagvec}(\boldsymbol{\beta} \otimes \boldsymbol{\gamma})$.
   - For $\ell \in [1, d]$, define $\mathbf{M}^\ell_f$ as the following matrix.

   $$\mathbf{M}^\ell_f = r[f^\ell(\boldsymbol{x})]\mathbf{A}^{\ell+1} - r\mathbf{A}^\ell$$

     Observe that there exists a vector $\boldsymbol{w}^\ell_{i,j} \in \mathbb{Z}_p^{n+2}$ such that $\langle \boldsymbol{w}^\ell_{i,j}, (rx_1, ..., rx_n, r, 0)\rangle = \mathbf{M}^\ell_{f,i,j}$. Note that each entry $\boldsymbol{w}^\ell_{i,j}$ is some linear function of $\mathbf{A}^{\ell+1}$. Note that the last component (component $n+2$) is just 0 and the second last component is $-\mathbf{A}^\ell_{i,j}$.
   - For $\ell \in [1, d]$ and $i, j \in [n^3]$, compute $\mathsf{Key}^\ell_{f,i,j} = \mathsf{cIPE.KeyGen}(\mathsf{sk}_1, \boldsymbol{w}^\ell_{i,j})$
   - Output $sk_f = (\mathsf{Key}_{0,f}, \{\mathsf{Key}^\ell_{f,i,j}\}_{\ell\in[d],i,j\in[n]})$

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

Hybrid$_1$: This hybrid is the same as the previous hybrid except that the randomness r used for challenge ciphertext is chosen at Setup. At the same time ciphertext component CT_0 and key components $\mathsf{Key}_{0,f}$ is generated differently. CT_0 now encrypts $(0, 1)$. $\mathsf{Key}_{0,f}$ is now computed to ensure that inner products remain the same.

1. The challenger does setup for the scheme as follows:

   - Sample $\mathsf{pp} \leftarrow \mathsf{cIPE.PPSetup}(1^\lambda)$. Let us assume $\mathsf{pp} = (e, G_1, G_2, G_T, g_1, g_2, \mathbb{Z}_p)$.
   - Run cIPE setup as follows. $\mathsf{sk}_0 \leftarrow \mathsf{cIPE.Setup}(\mathsf{pp}, 1^2)$. Thus these keys are used to encrypt vectors in $\mathbb{Z}_p^2$.

- Then run cIPE setup algorithm, for vectors of size 1^{n+2}. That is, compute $\mathsf{sk}_1 \leftarrow \mathsf{cIPE.Setup}(\mathsf{pp}, 1^{n+2})$.
- Sample $\boldsymbol{\beta}, \boldsymbol{\gamma} \leftarrow \mathbb{Z}_p^n$.
- Output $\mathsf{MSK} = \{\mathsf{sk}_0, \mathsf{sk}_1, \boldsymbol{\beta}, \boldsymbol{\gamma}\}$

2. [**Change**] Challenger samples randomness r from uniformly.

3. 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 $\mathbb{Z}_p^{3 \cdot n}$. We now describe how the challenge ciphertext CT is generated. Other ciphertexts $\mathsf{CT}(i)$ for $i \in [L]$ are generated as in the previous hybrid. Note that for ciphertext for message $m(i)$, the components will be denoted as: $\mathsf{CT}(i)_0, \mathsf{CTX}(i), \{\mathsf{CTC}(i)_j, \mathsf{CTK}(i)_k\}_{j,k}$

   - Parse $\mathsf{MSK} = \{\mathsf{sk}_0, \mathsf{sk}_1, \boldsymbol{\beta}, \boldsymbol{\gamma}\}$
   - [**Change**] Compute $\mathsf{CT}_0 = \mathsf{cIPE.Enc}(\mathsf{sk}_0, (0, 1))$.
   - Sample $\mathsf{sk}' \leftarrow \mathsf{cIPE.Setup}(\mathsf{pp}, 1^4)$.
   - Compute $\mathsf{CTC}_j \leftarrow \mathsf{cIPE.Enc}(\mathsf{sk}', (y_j, \beta j, 0, 0))$ for $j \in [n]$
   - Compute $\mathsf{CTK}_k \leftarrow \mathsf{cIPE.KeyGen}(\mathsf{sk}', (z_k, -r\gamma_k, 0, 0))$ for $k \in [n]$.
   - Compute $\mathsf{CTX} = \mathsf{cIPE.Enc}(\mathsf{sk}_1, (rx_1, ..., rx_n, r, 0))$.
   - Output $\mathsf{CT} = (\boldsymbol{x}, \mathsf{CT}_0, \mathsf{CTX}, \{\mathsf{CTC}_j, \mathsf{CTK}_k, \}_{j,k \in [n]})$

4. 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 associated with value Δ.

   - Parse $\mathsf{MSK} = \{\mathsf{sk}_0, \mathsf{sk}_1, \boldsymbol{\beta}, \boldsymbol{\gamma}\}$
   - Sample $\mathbf{A}^1, ..., \mathbf{A}^d$ randomly from $\mathbb{Z}_p^{n^3 \times n^3}$. This step is done for each query. We omit subscript of the function (e.g. $\mathbf{A}_f^\ell$) untill necessary.
   - Parse $f(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}) = \langle f^0, [f^1(\boldsymbol{x})] \cdot \ldots \cdot [f^d(\boldsymbol{x})] \cdot \mathsf{diagvec}(\boldsymbol{y} \otimes \boldsymbol{z}) \rangle$.
   - Let $\theta_f = \langle f^0, \mathbf{A}^1 \rangle$.
   - [**Change**] Compute $\mathsf{Key}_{0,f} = \mathsf{cIPE.KeyGen}(\mathsf{sk}_0, (\theta_f, r\theta_f))$.
   - Define $\mathbf{A}^{d+1} = \mathsf{diagvec}(\boldsymbol{\beta} \otimes \boldsymbol{\gamma})$.
   - For $\ell \in [1, d]$, define $\mathbf{M}_f^\ell$ as the following matrix.

     $$\mathbf{M}_f^\ell = r[f^\ell(\boldsymbol{x})]\mathbf{A}^{\ell+1} - r\mathbf{A}^\ell$$

     Observe that there exists a vector $\boldsymbol{w}_{i,j}^\ell \in \mathbb{Z}_p^{n+2}$ such that $\langle \boldsymbol{w}_{i,j}^\ell, (rx_1, ..., rx_n, r, 0) \rangle = \mathbf{M}_{f,i,j}^\ell$. Note that each entry $\boldsymbol{w}_{i,j}^\ell$ is some linear function of $\mathbf{A}^{\ell+1}$. Note that the last component (component $n+2$) is just 0 and the second last component is $-\mathbf{A}_{i,j}^\ell$.
   - For $\ell \in [1, d]$ and $i, j \in [n]$, compute $\mathsf{Key}_{f,i,j}^\ell = \mathsf{cIPE.KeyGen}(\mathsf{sk}_1, \boldsymbol{w}_{i,j}^\ell)$
   - Output $sk_f = (\mathsf{Key}_{0,f}, \{\mathsf{Key}_{f,i,j}^\ell\}_{\ell \in [d], i,j \in [n^3]})$

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

Lemma 2. *Assuming* cIPE *is secure, then, for all p.p.t. adversaries* $\mathcal{A}$, $|\Pr[\mathcal{A}(\mathbf{Hybrid}_0) = 1] - \Pr[\mathcal{A}(\mathbf{Hybrid}_1) = 1]| \leq \mathsf{negl}(\lambda)$. *If* cIPE *is subexponentially secure, this gap is subexponentially small.*

Proof. The only difference between the two hybrids is the way that the challenge ciphertext component CT_0 is generated and the functional key components $\mathsf{Key}_{0,f}$ are generated. In both the hybrids, the decryption of CT_0 from $\mathsf{Key}_{0,f}$ is $r\theta_f$. The decryption of component $\mathsf{CT}(i)_0$ (corresponding to other ciphertext queries) for $i \in [L]$ is also the same- $r(i)\theta_f$. Thus the claim follows from security of cIPE scheme. $\square$

$\mathbf{Hybrid}_2$: This hybrid is the same as the previous hybrid except the challenge ciphertext component CTX and key components $\mathsf{Key}^{\ell}_{f_i,j,k}$ are generated for $i \in [L]$, $\ell \in [d]$ and $j,k \in [n^3]$. They are done in such a way that decryption of all components $\mathsf{CTX}(i)$ with keys $\mathsf{Key}^{\ell}_{g,j,k}$ remain the same.

1. The challenger does setup for the scheme as follows:
   - Sample $\mathsf{pp} \leftarrow \mathsf{cIPE.PPSetup}(1^{\lambda})$. Let us assume $\mathsf{pp} = (e, G_1, G_2, G_T, g_1, g_2, \mathbb{Z}_p)$.
   - Run cIPE setup as follows. $\mathsf{sk}_0 \leftarrow \mathsf{cIPE.Setup}(\mathsf{pp}, 1^2)$. Thus these keys are used to encrypt vectors in $\mathbb{Z}_p^2$.
   - Then run cIPE setup algorithm, for vectors of size 1^{n+2}. That is, compute $\mathsf{sk}_1 \leftarrow \mathsf{cIPE.Setup}(\mathsf{pp}, 1^{n+2})$.
   - Sample $\boldsymbol{\beta}, \boldsymbol{\gamma} \leftarrow \mathbb{Z}_p^n$.
   - Output $\mathsf{MSK} = \{\mathsf{sk}_0, \mathsf{sk}_1, \boldsymbol{\beta}, \boldsymbol{\gamma}\}$
2. Challenger samples randomness r from uniformly.
3. 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 $\mathbb{Z}_p^{3 \cdot n}$. We now describe how the challenge ciphertext CT is generated. Other ciphertexts $\mathsf{CT}(i)$ for $i \in [L]$ are generated as in the previous hybrid. Note that for ciphertext for message $m(i)$, the components will be denoted as: $\mathsf{CT}(i)_0, \mathsf{CTX}(i), \{\mathsf{CTC}(i)_j, \mathsf{CTK}(i)_k\}_{j,k}$
   - Parse $\mathsf{MSK} = \{\mathsf{sk}_0, \mathsf{sk}_1, \boldsymbol{\beta}, \boldsymbol{\gamma}\}$
   - Compute $\mathsf{CT}_0 = \mathsf{cIPE.Enc}(\mathsf{sk}_0, (0,1))$.
   - Sample $\mathsf{sk}' \leftarrow \mathsf{cIPE.Setup}(\mathsf{pp}, 1^4)$.
   - Compute $\mathsf{CTC}_j \leftarrow \mathsf{cIPE.Enc}(\mathsf{sk}', (y_j, \beta j, 0, 0))$ for $j \in [n]$
   - Compute $\mathsf{CTK}_k \leftarrow \mathsf{cIPE.KeyGen}(\mathsf{sk}', (z_k, -r\gamma_k, 0, 0))$ for $k \in [n]$.
   - [**Change**] Compute $\mathsf{CTX} = \mathsf{cIPE.Enc}(\mathsf{sk}_1, (0, ...0, 1))$.
   - Output $\mathsf{CT} = (\boldsymbol{x}, \mathsf{CT}_0, \mathsf{CTX}, \{\mathsf{CTC}_j, \mathsf{CTK}_k, \}_{j,k \in [n]})$
4. 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 associated with value Δ.

- Parse $\mathsf{MSK} = \{\mathsf{sk}_0, \mathsf{sk}_1, \boldsymbol{\beta}, \boldsymbol{\gamma}\}$
- Sample $\mathbf{A}^1, ..., \mathbf{A}^d$ randomly from $\mathbb{Z}_p^{n^3 \times n^3}$. This step is done for each query. We omit subscript of the function (e.g. $\mathbf{A}_f^\ell$) until necessary.
- Parse $f(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}) = \langle f^0, [f^1(\boldsymbol{x})] \cdot \ldots \cdot [f^d(\boldsymbol{x})] \cdot \mathsf{diagvec}(\boldsymbol{y} \otimes \boldsymbol{z})\rangle$.
- Let $\theta_f = \langle f^0, \mathbf{A}^1\rangle$.
- Compute $\mathsf{Key}_{0,f} = \mathsf{cIPE.KeyGen}(\mathsf{sk}_0, (\theta_f, r\theta_f))$.
- Define $\mathbf{A}^{d+1} = \mathsf{diagvec}(\boldsymbol{\beta} \otimes \boldsymbol{\gamma})$.
- [**Change**] For $\ell \in [1, d]$, define $\mathbf{M}_f^\ell$ as the following matrix.

  $$\mathbf{M}_f^\ell = r[f^\ell(\boldsymbol{x})]\mathbf{A}^{\ell+1} - r\mathbf{A}^\ell$$

  Let $\boldsymbol{w}_{i,j}^\ell \in \mathbb{Z}_p^{n+2}$ be the vector computed as in the previous hybrid. We define $\boldsymbol{w'}_{i,j}^\ell$ be the vector that equals $\boldsymbol{w}_{i,j}^\ell$ in its first $n+1$ components and is equal to $\mathbf{M}_{f,i,j}^\ell$ in the component $n+2$.
- [**Change**] For $\ell \in [1, d]$ and $i, j \in [n]$, compute $\mathsf{Key}_{f,i,j}^\ell = \mathsf{cIPE.KeyGen}(\mathsf{sk}_1, \boldsymbol{w'}_{i,j}^\ell)$
- Output $sk_f = (\mathsf{Key}_{0,f}, \{\mathsf{Key}_{f,i,j}^\ell\}_{\ell \in [d], i,j \in [n]})$

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

Lemma 3. *Assuming* cIPE *is secure, then, for all p.p.t. adversaries* $\mathcal{A}$, $|\Pr[\mathcal{A}(\mathbf{Hybrid}_1) = 1] - \Pr[\mathcal{A}(\mathbf{Hybrid}_2) = 1]| \le \mathsf{negl}(\lambda)$. *If* cIPE *is subexponentially secure, this gap is subexponentially small.*

Proof. The only difference between the two hybrids is the way that the challenge ciphertext component CTX is generated and the functional key components $\mathsf{Key}_{f,i,j}^\ell$ are generated for all queried functions f and $i, j \in [n^3]$. In both the hybrids, the decryption of CTX from $\mathsf{Key}_{f,i,j}^\ell$ is $\mathbf{M}_{i,j}^\ell$ for $i, j \in [n^3]$. In $\mathbf{Hybrid}_1$ CTX is generated as an encryption of $(rx_1, ..., rx_n, r, 0)$, while in $\mathbf{Hybrid}_2$ it is generated as an encryption of $(0, ..., 0, 1)$. In $\mathbf{Hybrid}_1$, $\mathsf{Key}_{f,i,j}^\ell$ is generated as a cIPE key for $\boldsymbol{w}_{f,i,j}^\ell$, while in $\mathbf{Hybrid}_2$ it is generated as a cIPE key for vector $\boldsymbol{w'}_{f,i,j}^\ell$. Note that this ensures that the decryptions of $\mathsf{CTX}(i)$ and CTX with all keys $\mathsf{Key}_{f,i,j}^\ell$ remain the same. Thus the claim follows from security of cIPE scheme. □

We now define a series of hybrids.

$\mathbf{Hybrid}_{3,a}$: For $a \in [n]$, this hybrid is the same as the previous hybrid except the challenge ciphertext component CTC_a is generated differently. The CTC_j encrypt 0 for $j \le a$, and y_j otherwise. In order to account for this change, we change we change $\mathsf{Key}_{0,f}$ to be the key for $(\theta_f, r\theta_f + f(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}) - f(\boldsymbol{x}, \boldsymbol{y}_a, \boldsymbol{z}))$. Here $\boldsymbol{y}_a = (0, ..., 0, y_{a+1}, ..., y_n)$.

1. The challenger does setup for the scheme as follows:

   - Sample $\mathsf{pp} \leftarrow \mathsf{cIPE.PPSetup}(1^\lambda)$. Let us assume $\mathsf{pp} = (e, G_1, G_2, G_T, g_1, g_2, \mathbb{Z}_p)$.
   - Run cIPE setup as follows. $\mathsf{sk}_0 \leftarrow \mathsf{cIPE.Setup}(\mathsf{pp}, 1^2)$. Thus these keys are used to encrypt vectors in $\mathbb{Z}_p^2$.

- Then run cIPE setup algorithm, for vectors of size 1^{n+2}. That is, compute $\mathsf{sk}_1 \leftarrow \mathsf{cIPE.Setup}(\mathsf{pp}, 1^{n+2})$.
- Sample $\boldsymbol{\beta}, \boldsymbol{\gamma} \leftarrow \mathbb{Z}_p^n$.
- Output $\mathsf{MSK} = \{\mathsf{sk}_0, \mathsf{sk}_1, \boldsymbol{\beta}, \boldsymbol{\gamma}\}$

2. Challenger samples randomness r from uniformly.

3. 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 $\mathbb{Z}_p^{3 \cdot n}$. We now describe how the challenge ciphertext CT is generated. Other ciphertexts $\mathsf{CT}(i)$ for $i \in [L]$ are generated as in the previous hybrid. Note that for ciphertext for message $m(i)$, the components will be denoted as: $\mathsf{CT}(i)_0, \mathsf{CTX}(i), \{\mathsf{CTC}(i)_j, \mathsf{CTK}(i)_k\}_{j,k}$

   - Parse $\mathsf{MSK} = \{\mathsf{sk}_0, \mathsf{sk}_1, \boldsymbol{\beta}, \boldsymbol{\gamma}\}$
   - Compute $\mathsf{CT}_0 = \mathsf{cIPE.Enc}(\mathsf{sk}_0, (0, 1))$.
   - Sample $\mathsf{sk}' \leftarrow \mathsf{cIPE.Setup}(\mathsf{pp}, 1^4)$.
   - [**Change**] Compute $\mathsf{CTC}_j \leftarrow \mathsf{cIPE.Enc}(\mathsf{sk}', (0, \beta j, 0, 0))$ for $j \leq a$
   - Compute $\mathsf{CTC}_j \leftarrow \mathsf{cIPE.Enc}(\mathsf{sk}', (y_j, \beta_j, 0, 0))$ for $j \in [a+1, n]$.
   - Compute $\mathsf{CTK}_k \leftarrow \mathsf{cIPE.KeyGen}(\mathsf{sk}', (z_k, -r\gamma_k, 0, 0))$ for $k \in [n]$.
   - Compute $\mathsf{CTX} = \mathsf{cIPE.Enc}(\mathsf{sk}_1, (0, ...0, 1))$.
   - Output $\mathsf{CT} = (\boldsymbol{x}, \mathsf{CT}_0, \mathsf{CTX}, \{\mathsf{CTC}_j, \mathsf{CTK}_k, \}_{j,k \in [n]})$

4. 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 associated with value Δ.

   - Parse $\mathsf{MSK} = \{\mathsf{sk}_0, \mathsf{sk}_1, \boldsymbol{\beta}, \boldsymbol{\gamma}\}$
   - Sample $\mathbf{A}^1, ..., \mathbf{A}^d$ randomly from $\mathbb{Z}_p^{n^3 \times n^3}$. This step is done for each query. We omit subscript of the function (e.g. $\mathbf{A}_f^\ell$) untill necessary.
   - Parse $f(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}) = \langle f^0, [f^1(\boldsymbol{x})] \cdot \ldots \cdot [f^d(\boldsymbol{x})] \cdot \mathsf{diagvec}(\boldsymbol{y} \otimes \boldsymbol{z}) \rangle$.
   - Let $\theta_f = \langle f^0, \mathbf{A}^1 \rangle$.
   - [**Change**] Compute $\mathsf{Key}_{0,f} = \mathsf{cIPE.KeyGen}(\mathsf{sk}_0, (\theta_f, r\theta_f + \Delta - f(\boldsymbol{x}, \boldsymbol{y}_a, \boldsymbol{z})))$. Here $\boldsymbol{y}_a = (0, .., 0, y_{a+1}, ..., y_n)$. Note that for $a = n$, the hardwiring $\Delta - f(\boldsymbol{x}, \boldsymbol{y}_a, \boldsymbol{z})$ is just equal to Δ.
   - Define $\mathbf{A}^{d+1} = \mathsf{diagvec}(\boldsymbol{\beta} \otimes \boldsymbol{\gamma})$.
   - For $\ell \in [1, d]$, define $\mathbf{M}_f^\ell$ as the following matrix.

   $$\mathbf{M}_f^\ell = r[f^\ell(\boldsymbol{x})]\mathbf{A}^{\ell+1} - r\mathbf{A}^\ell$$

   Let $\boldsymbol{w}_{i,j}^\ell \in \mathbb{Z}_p^{n+2}$ be the vector computed as in the previous hybrid. We define $\boldsymbol{w'}_{i,j}^\ell$ be the vector that equals $\boldsymbol{w}_{i,j}^\ell$ in its first $n+1$ components and is equal to $\mathbf{M}_{f,i,j}^\ell$ in the component $n+2$.

- For $\ell \in [1,d]$ and $i,j \in [n^3]$, compute $\mathsf{Key}^{\ell}_{f,i,j} = \mathsf{cIPE.KeyGen}(\mathsf{sk}_1, \boldsymbol{w'}^{\ell}_{i,j})$
- Output $sk_f = (\mathsf{Key}_{0,f}, \{\mathsf{Key}^{\ell}_{f,i,j}\}_{\ell \in [d], i,j \in [n^3]})$

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

Let $\mathbf{Hybrid}_2 = \mathbf{Hybrid}_{3,-1}$. We defer indistinguishability of $\mathbf{Hybrid}_{3,i}$ from $\mathbf{Hybrid}_{3,i+1}$ for $i \in [-1, n-1]$ is argued later in Section D.1.

$\mathbf{Hybrid}_4$: This hybrid is the same as the previous hybrid except that in the challenge ciphertext components CTK_j for $j \in [n]$ are generated by replacing z_j with 0. This can be done because the output of the decryption of CTC_j with CTK_k do not change.

1. The challenger does setup for the scheme as follows:
   - Sample $\mathsf{pp} \leftarrow \mathsf{cIPE.PPSetup}(1^{\lambda})$. Let us assume $\mathsf{pp} = (e, G_1, G_2, G_T, g_1, g_2, \mathbb{Z}_p)$.
   - Run cIPE setup as follows. $\mathsf{sk}_0 \leftarrow \mathsf{cIPE.Setup}(\mathsf{pp}, 1^2)$. Thus these keys are used to encrypt vectors in $\mathbb{Z}_p^2$.
   - Then run cIPE setup algorithm, for vectors of size 1^{n+2}. That is, compute $\mathsf{sk}_1 \leftarrow \mathsf{cIPE.Setup}(\mathsf{pp}, 1^{n+2})$.
   - Sample $\boldsymbol{\beta}, \boldsymbol{\gamma} \leftarrow \mathbb{Z}_p^n$.
   - Output $\mathsf{MSK} = \{\mathsf{sk}_0, \mathsf{sk}_1, \boldsymbol{\beta}, \boldsymbol{\gamma}\}$

2. Challenger samples randomness r from uniformly.

3. 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 $\mathbb{Z}_p^{3 \cdot n}$. We now describe how the challenge ciphertext CT is generated. Other ciphertexts $\mathsf{CT}(i)$ for $i \in [L]$ are generated as in the previous hybrid. Note that for ciphertext for message $m(i)$, the components will be denoted as: $\mathsf{CT}(i)_0, \mathsf{CTX}(i), \{\mathsf{CTC}(i)_j, \mathsf{CTK}(i)_k\}_{j,k}$
   - Parse $\mathsf{MSK} = \{\mathsf{sk}_0, \mathsf{sk}_1, \boldsymbol{\beta}, \boldsymbol{\gamma}\}$
   - Compute $\mathsf{CT}_0 = \mathsf{cIPE.Enc}(\mathsf{sk}_0, (0,1))$.
   - Sample $\mathsf{sk}' \leftarrow \mathsf{cIPE.Setup}(\mathsf{pp}, 1^4)$.
   - Compute $\mathsf{CTC}_j \leftarrow \mathsf{cIPE.Enc}(\mathsf{sk}', (0, \beta j, 0, 0))$ for $j \in [n]$
   - [**Change**] Compute $\mathsf{CTK}_k \leftarrow \mathsf{cIPE.KeyGen}(\mathsf{sk}', (0, -r\gamma_k, 0, 0))$ for $k \in [n]$.
   - Compute $\mathsf{CTX} = \mathsf{cIPE.Enc}(\mathsf{sk}_1, (0, ...0, 1))$.
   - Output $\mathsf{CT} = (\boldsymbol{x}, \mathsf{CT}_0, \mathsf{CTX}, \{\mathsf{CTC}_j, \mathsf{CTK}_k,\}_{j,k \in [n]})$

4. 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 associated with value Δ.
   - Parse $\mathsf{MSK} = \{\mathsf{sk}_0, \mathsf{sk}_1, \boldsymbol{\beta}, \boldsymbol{\gamma}\}$
   - Sample $\mathbf{A}^1, ..., \mathbf{A}^d$ randomly from $\mathbb{Z}_p^{n \times n}$. This step is done for each query. We omit subscript of the function (e.g. $\mathbf{A}^{\ell}_f$) until necessary.

- Parse $f(\boldsymbol{x},\boldsymbol{y},\boldsymbol{z}) = \langle f^0, [f^1(\boldsymbol{x})]\cdot \ldots \cdot [f^d(\boldsymbol{x})]\cdot \mathsf{diagvec}(\boldsymbol{y}\otimes \boldsymbol{z})\rangle$.
- Let $\theta_f = \langle f^0, \mathbf{A}^1\rangle$.
- Compute $\mathsf{Key}_{0,f} = \mathsf{cIPE.KeyGen}(\mathsf{sk}_0, (\theta_f, r\theta_f + \Delta))$.
- Define $\mathbf{A}^{d+1} = \mathsf{diagvec}(\boldsymbol{\beta}\otimes\boldsymbol{\gamma})$.
- For $\ell \in [1,d]$, define $\mathbf{M}_f^\ell$ as the following matrix.

  $$\mathbf{M}_f^\ell = r[f^\ell(\boldsymbol{x})]\mathbf{A}^{\ell+1} - r\mathbf{A}^\ell$$

  Let $\boldsymbol{w}_{i,j}^\ell \in \mathbb{Z}_p^{n+2}$ be the vector computed as in the previous hybrid. We define $\boldsymbol{w'}_{i,j}^{\ell}$ be the vector that equals $\boldsymbol{w}_{i,j}^\ell$ in its first $n+1$ components and is equal to $\mathbf{M}_{i,j}^\ell$ in the component $n+2$.
- For $\ell \in [1,d]$ and $i,j \in [n^3]$, compute $\mathsf{Key}_{f,i,j}^\ell = \mathsf{cIPE.KeyGen}(\mathsf{sk}_1, \boldsymbol{w'}_{i,j}^{\ell})$
- Output $sk_f = (\mathsf{Key}_{0,f}, \{\mathsf{Key}_{f,i,j}^\ell\}_{\ell\in[d],i,j\in[n^3]})$

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

Lemma 4. *Assuming* cIPE *is secure, then, for all p.p.t. adversaries* $\mathcal{A}$, $|\Pr[\mathcal{A}(\mathbf{Hybrid}_{3,n}) = 1] - \Pr[\mathcal{A}(\mathbf{Hybrid}_4) = 1]| \le \mathsf{negl}(\lambda)$. *If* cIPE *is subexponentially secure, this gap is subexponentially small.*

Proof. The only difference between the two hybrids is the way that the challenge ciphertext component CTK_j for $j \in [n]$ is generated. They are generated as function key for vector $(z_j, -r\gamma_k, 0, 0)$ in $\mathbf{Hybrid}_{3,n}$, where as in $\mathbf{Hybrid}_4$, they are generated as function keys for $(0, -r\gamma_j, 0, 0)$. Note that in both hybrids when decrypted using CTC_k, they both result in $[-r\beta_k\gamma_j]_T$. Thus the claim follows from security of cIPE scheme. □

Hybrid$_5$: This hybrid is the same as the previous one except that now we switch CT_0. We ensure that the decryption of CT_0 as well as $\mathsf{CT}(i)_0$ with key components $\mathsf{Key}_{0,f}$ remain the same.

1. The challenger does setup for the scheme as follows:
   - Sample $\mathsf{pp} \leftarrow \mathsf{cIPE.PPSetup}(1^\lambda)$. Let us assume $\mathsf{pp} = (e, G_1, G_2, G_T, g_1, g_2, \mathbb{Z}_p)$.
   - Run cIPE setup as follows. $\mathsf{sk}_0 \leftarrow \mathsf{cIPE.Setup}(\mathsf{pp}, 1^2)$. Thus these keys are used to encrypt vectors in $\mathbb{Z}_p^2$.
   - Then run cIPE setup algorithm, for vectors of size 1^{n+2}. That is, compute $\mathsf{sk}_1 \leftarrow \mathsf{cIPE.Setup}(\mathsf{pp}, 1^{n+2})$.
   - Sample $\boldsymbol{\beta},\boldsymbol{\gamma} \leftarrow \mathbb{Z}_p^n$.
   - Output $\mathsf{MSK} = \{\mathsf{sk}_0, \mathsf{sk}_1, \boldsymbol{\beta}, \boldsymbol{\gamma}\}$
2. Challenger samples randomness r from uniformly.
3. 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 $\mathbb{Z}_p^{3\cdot n}$. We now describe how the challenge ciphertext CT is generated. Other ciphertexts $\mathsf{CT}(i)$ for $i \in [L]$ are generated as in the previous hybrid. Note that for ciphertext for message $m(i)$, the components will be denoted as: $\mathsf{CT}(i)_0, \mathsf{CTX}(i), \{\mathsf{CTC}(i)_j, \mathsf{CTK}(i)_k\}_{j,k}$

- Parse $\mathsf{MSK} = \{\mathsf{sk}_0, \mathsf{sk}_1, \boldsymbol{\beta}, \boldsymbol{\gamma}\}$
- [**Change**] Compute $\mathsf{CT}_0 = \mathsf{cIPE.Enc}(\mathsf{sk}_0, (r, 1))$.
- Sample $\mathsf{sk}' \leftarrow \mathsf{cIPE.Setup}(\mathsf{pp}, 1^4)$.
- Compute $\mathsf{CTC}_j \leftarrow \mathsf{cIPE.Enc}(\mathsf{sk}', (0, \beta j, 0, 0))$ for $j \in [n]$
- Compute $\mathsf{CTK}_k \leftarrow \mathsf{cIPE.KeyGen}(\mathsf{sk}', (0, -r\gamma_k, 0, 0))$ for $k \in [n]$.
- Compute $\mathsf{CTX} = \mathsf{cIPE.Enc}(\mathsf{sk}_1, (0, ...0, 1))$.
- Output $\mathsf{CT} = (\boldsymbol{x}, \mathsf{CT}_0, \mathsf{CTX}, \{\mathsf{CTC}_j, \mathsf{CTK}_k, \}_{j,k\in[n]})$

4. 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 associated with value Δ.

   - Parse $\mathsf{MSK} = \{\mathsf{sk}_0, \mathsf{sk}_1, \boldsymbol{\beta}, \boldsymbol{\gamma}\}$
   - Sample $\mathbf{A}^1, ..., \mathbf{A}^d$ randomly from $\mathbb{Z}_p^{n^3 \times n^3}$. This step is done for each query. We omit subscript of the function (e.g. $\mathbf{A}_f^\ell$) until necessary.
   - Parse $f(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}) = \langle f^0, [f^1(\boldsymbol{x})] \cdot \ldots \cdot [f^d(\boldsymbol{x})] \cdot \mathsf{diagvec}(\boldsymbol{y} \otimes \boldsymbol{z})\rangle$.
   - Let $\theta_f = \langle f^0, \mathbf{A}^1\rangle$.
   - Compute $\mathsf{Key}_{0,f} = \mathsf{cIPE.KeyGen}(\mathsf{sk}_0, (\theta_f, \Delta))$.
   - Define $\mathbf{A}^{d+1} = \boldsymbol{\beta} \otimes \boldsymbol{\gamma}$.
   - For $\ell \in [1, d]$, define $\mathbf{M}_f^\ell$ as the following matrix.

     $$\mathbf{M}_f^\ell = r[f^\ell(\boldsymbol{x})]\mathbf{A}^{\ell+1} - r\mathbf{A}^\ell$$

     Let $\boldsymbol{w}_{i,j}^\ell \in \mathbb{Z}_p^{n+2}$ be the vector computed as in the previous hybrid. We define $\boldsymbol{w'}_{i,j}^\ell$ be the vector that equals $\boldsymbol{w}_{i,j}^\ell$ in its first $n+1$ components and is equal to $\mathbf{M}_{f,i,j}^\ell$ in the component $n+2$.
   - For $\ell \in [1, d]$ and $i, j \in [n^3]$, compute $\mathsf{Key}_{f,i,j}^\ell = \mathsf{cIPE.KeyGen}(\mathsf{sk}_1, \boldsymbol{w'}_{i,j}^\ell)$
   - Output $sk_f = (\mathsf{Key}_{0,f}, \{\mathsf{Key}_{f,i,j}^\ell\}_{\ell\in[d],i,j\in[n^3]})$

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

Lemma 5. *Assuming* cIPE *is secure, then, for all p.p.t. adversaries* $\mathcal{A}$, $|\Pr[\mathcal{A}(\mathbf{Hybrid}_4) = 1] - \Pr[\mathcal{A}(\mathbf{Hybrid}_5) = 1]| \leq \mathsf{negl}(\lambda)$. *If* cIPE *is subexponentially secure, this gap is subexponentially small.*

Proof. The proof of this is similar to the proof of lemma 2 □

Hybrid$_6$: This hybrid is the same as the previous one except that now we switch CTX and $\mathsf{Key}_{f,i,j}^\ell$ for all functions f. We ensure that the decryption of CTX as well as $\mathsf{CTX}(i)$ with key components $\mathsf{Key}_{f,i,j}^\ell$ remain the same for $\ell \in [d]$ and $i, j \in [n^3]$.

1. The challenger does setup for the scheme as follows:

- Sample $\mathsf{pp} \leftarrow \mathsf{cIPE.PPSetup}(1^\lambda)$. Let us assume $\mathsf{pp} = (e, G_1, G_2, G_T, g_1, g_2, \mathbb{Z}_p)$.
- Run cIPE setup as follows. $\mathsf{sk}_0 \leftarrow \mathsf{cIPE.Setup}(\mathsf{pp}, 1^2)$. Thus these keys are used to encrypt vectors in $\mathbb{Z}_p^2$.
- Then run cIPE setup algorithm, for vectors of size 1^{n+2}. That is, compute $\mathsf{sk}_1 \leftarrow \mathsf{cIPE.Setup}(\mathsf{pp}, 1^{n+2})$.
- Sample $\boldsymbol{\beta}, \boldsymbol{\gamma} \leftarrow \mathbb{Z}_p^n$.
- Output $\mathsf{MSK} = \{\mathsf{sk}_0, \mathsf{sk}_1, \boldsymbol{\beta}, \boldsymbol{\gamma}\}$

2. Challenger samples randomness r from uniformly.

3. 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 $\mathbb{Z}_p^{3 \cdot n}$. We now describe how the challenge ciphertext CT is generated. Other ciphertexts $\mathsf{CT}(i)$ for $i \in [L]$ are generated as in the previous hybrid. Note that for ciphertext for message $m(i)$, the components will be denoted as: $\mathsf{CT}(i)_0, \mathsf{CTX}(i), \{\mathsf{CTC}(i)_j, \mathsf{CTK}(i)_k\}_{j,k}$

   - Parse $\mathsf{MSK} = \{\mathsf{sk}_0, \mathsf{sk}_1, \boldsymbol{\beta}, \boldsymbol{\gamma}\}$
   - Compute $\mathsf{CT}_0 = \mathsf{cIPE.Enc}(\mathsf{sk}_0, (r, 1))$.
   - Sample $\mathsf{sk}' \leftarrow \mathsf{cIPE.Setup}(\mathsf{pp}, 1^4)$.
   - Compute $\mathsf{CTC}_j \leftarrow \mathsf{cIPE.Enc}(\mathsf{sk}', (0, \beta j, 0, 0))$ for $j \in [n]$
   - Compute $\mathsf{CTK}_k \leftarrow \mathsf{cIPE.KeyGen}(\mathsf{sk}', (0, -r\gamma_k, 0, 0))$ for $k \in [n]$.
   - [**Change**] Compute $\mathsf{CTX} = \mathsf{cIPE.Enc}(\mathsf{sk}_1, (rx_1, ... rx_n, r, 0))$.
   - Output $\mathsf{CT} = (\boldsymbol{x}, \mathsf{CT}_0, \mathsf{CTX}, \{\mathsf{CTC}_j, \mathsf{CTK}_k, \}_{j,k \in [n]})$

4. 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 associated with value Δ.

   - Parse $\mathsf{MSK} = \{\mathsf{sk}_0, \mathsf{sk}_1, \boldsymbol{\beta}, \boldsymbol{\gamma}\}$
   - Sample $\mathbf{A}^1, ..., \mathbf{A}^d$ randomly from $\mathbb{Z}_p^{n \times n}$. This step is done for each query. We omit subscript of the function (e.g. $\mathbf{A}_f^\ell$) until necessary.
   - Parse $f(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}) = \langle f^0, [f^1(\boldsymbol{x})] \cdot \ldots \cdot [f^d(\boldsymbol{x})] \cdot \mathsf{diagvec}(\boldsymbol{y} \otimes \boldsymbol{z}) \rangle$.
   - Let $\theta_f = \langle f^0, \mathbf{A}^1 \rangle$.
   - Compute $\mathsf{Key}_{0,f} = \mathsf{cIPE.KeyGen}(\mathsf{sk}_0, (\theta_f, \Delta))$.
   - Define $\mathbf{A}^{d+1} = \mathsf{diagvec}(\boldsymbol{\beta} \otimes \boldsymbol{\gamma})$.
   - For $\ell \in [1, d]$, define $\mathbf{M}_f^\ell$ as the following matrix.

$$\mathbf{M}_f^\ell = r[f^\ell(\boldsymbol{x})]\mathbf{A}^{\ell+1} - r\mathbf{A}^\ell$$

   - [**Change**] For $\ell \in [1, d]$ and $i, j \in [n]$, compute $\mathsf{Key}_{f,i,j}^\ell = \mathsf{cIPE.KeyGen}(\mathsf{sk}_1, \boldsymbol{w}_{i,j}^\ell)$
   - Output $sk_f = (\mathsf{Key}_{0,f}, \{\mathsf{Key}_{f,i,j}^\ell\}_{\ell \in [d], i,j \in [n^3]})$

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

Lemma 6. *Assuming* cIPE *is secure, then, for all p.p.t. adversaries* $\mathcal{A}$, $|\Pr[\mathcal{A}(\mathbf{Hybrid}_5) = 1] - \Pr[\mathcal{A}(\mathbf{Hybrid}_6) = 1]| \leq \mathsf{negl}(\lambda)$. *If* cIPE *is subexponentially secure, this gap is subexponentially small.*

Proof. The proof of this is similar to the proof of lemma 3 □

Note that $\mathbf{Hybrid}_6$ corresponds to the case with semi-functional challenge ciphertext and semi-functional function keys.

Now we argue indistinguishability of $\mathbf{Hybrid}_{3,i}$ with $\mathbf{Hybrid}_{3,i+1}$ for $i \in [-1, n-1]$

D.1 Indistinguishability of $\mathbf{Hybrid}_{3,i}$ and $\mathbf{Hybrid}_{3,i+1}$

We will prove indistinguishability between $\mathbf{Hybrid}_2$ and $\mathbf{Hybrid}_{3,0}$. For successive hybrids, indistinguishability argument is similar. Here is a brief description of the intermediate hybrids. We will elaborate on them.

- $\mathbf{Hybrid}_{2,0}$: This hybrid is the same as the previous one except that now we change CTC_1, (also $\mathsf{CTC}(i)_1$) to encrypt $(0,0,1,0)$. There is a corresponding change in CTK_k (and $\mathsf{CTK}(i)_k$) for $k \in [n]$. CTK_k is generated as a key for the vector $(z_k, -r\gamma_k, y_1 z_k - r\beta_1\gamma_k, 0)$
- $\mathbf{Hybrid}_{2,1}$: This hybrid is the same as the previous one except that now we use SXDH to replace $r\beta_1\gamma_k$ with randomly chosen $v_{1,k}$. At the same time, matrices $r\mathbf{A}_f^\ell$ for $\ell \in [d]$ (corresponding to all queried functions f) are replaced with randomly chosen matrices $\mathbf{B}_f^\ell$.
- $\mathbf{Hybrid}_{2,2}$: Now we can replace y_1 with 0. This change is information theoretic.
- $\mathbf{Hybrid}_{2,3}$: Now we use SXDH again to undo the change and replace $\mathbf{B}_f^\ell$ and $v_{1,k}$.
- $\mathbf{Hybrid}_{2,4}$: This is the same as $\mathbf{Hybrid}_{3,0}$.

Now we describe the hybrids.

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

1. The challenger does setup for the scheme as follows:
   - Sample $\mathsf{pp} \leftarrow \mathsf{cIPE.PPSetup}(1^\lambda)$. Let us assume $\mathsf{pp} = (e, G_1, G_2, G_T, g_1, g_2, \mathbb{Z}_p)$.
   - Run cIPE setup as follows. $\mathsf{sk}_0 \leftarrow \mathsf{cIPE.Setup}(\mathsf{pp}, 1^2)$. Thus these keys are used to encrypt vectors in $\mathbb{Z}_p^2$.
   - Then run cIPE setup algorithm, for vectors of size 1^{n+2}. That is, compute $\mathsf{sk}_1 \leftarrow \mathsf{cIPE.Setup}(\mathsf{pp}, 1^{n+2})$.
   - Sample $\boldsymbol{\beta}, \boldsymbol{\gamma} \leftarrow \mathbb{Z}_p^n$.
   - Output $\mathsf{MSK} = \{\mathsf{sk}_0, \mathsf{sk}_1, \boldsymbol{\beta}, \boldsymbol{\gamma}\}$
2. Challenger samples randomness r from uniformly.

3. 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 $\mathbb{Z}_p^{3 \cdot n}$. We now describe how the challenge ciphertext CT is generated. Other ciphertexts $\mathsf{CT}(i)$ for $i \in [L]$ are generated as in the previous hybrid. Except that the change is described below. Note that for ciphertext for message $m(i)$, the components will be denoted as: $\mathsf{CT}(i)_0, \mathsf{CTX}(i), \{\mathsf{CTC}(i)_j, \mathsf{CTK}(i)_k\}_{j,k}$

   Here is how CT is generated.

   - Parse $\mathsf{MSK} = \{\mathsf{sk}_0, \mathsf{sk}_1, \boldsymbol{\beta}, \boldsymbol{\gamma}\}$
   - Compute $\mathsf{CT}_0 = \mathsf{cIPE.Enc}(\mathsf{sk}_0, (0, 1))$.
   - Sample $\mathsf{sk}' \leftarrow \mathsf{cIPE.Setup}(\mathsf{pp}, 1^4)$.
   - [**Change**] Compute $\mathsf{CTC}_j \leftarrow \mathsf{cIPE.Enc}(\mathsf{sk}', (y_j, \beta_j, 0, 0))$ for $j \in [n] \setminus 1$
   - [**Change**] Compute $\mathsf{CTC}_1 \leftarrow \mathsf{cIPE.Enc}(\mathsf{sk}', (0, 0, 1, 0))$.
   - [**Change**] Compute $\mathsf{CTK}_k \leftarrow \mathsf{cIPE.KeyGen}(\mathsf{sk}', (z_k, -r\gamma_k, y_1 z_k - r\beta_1\gamma_k, 0))$ for $k \in [n]$.
   - Compute $\mathsf{CTX} = \mathsf{cIPE.Enc}(\mathsf{sk}_1, (0, ...0, 1))$.
   - Output $\mathsf{CT} = (\boldsymbol{x}, \mathsf{CT}_0, \mathsf{CTX}, \{\mathsf{CTC}_j, \mathsf{CTK}_k, \}_{j,k \in [n]})$

   The only change in generating $\mathsf{CT}(i)$ for $i \in [L]$ from the previous hybrid is marked below.

   - [**Change**] Compute $\mathsf{CTC}(i)_1 \leftarrow \mathsf{cIPE.Enc}(\mathsf{sk}'(i), (0, 0, 1, 0))$.
   - [**Change**] Compute $\mathsf{CTK}(i)_k \leftarrow \mathsf{cIPE.Enc}(\mathsf{sk}'(i), (z(i)_k, -r(i)\gamma_k, y(i)_1 z(i)_k - r(i)\beta_1\gamma_k, 0))$ for $k \in [n]$.

4. 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 associated with value Δ.

   - Parse $\mathsf{MSK} = \{\mathsf{sk}_0, \mathsf{sk}_1, \boldsymbol{\beta}, \boldsymbol{\gamma}\}$
   - Sample $\mathbf{A}_f^1, ..., \mathbf{A}_f^d$ randomly from $\mathbb{Z}_p^{n^3 \times n^3}$. This step is done for each function query.
   - Parse $f(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}) = \langle f^0, [f^1(\boldsymbol{x})] \cdot \ldots \cdot [f^d(\boldsymbol{x})] \cdot \mathsf{diagvec}(\boldsymbol{y} \otimes \boldsymbol{z}) \rangle$.
   - Let $\theta_f = \langle f^0, \mathbf{A}_f^1 \rangle$.
   - Compute $\mathsf{Key}_{0,f} = \mathsf{cIPE.KeyGen}(\mathsf{sk}_0, (\theta_f, r\theta_f))$.
   - Define $\mathbf{A}_f^{d+1} = \mathsf{diagvec}(\boldsymbol{\beta} \otimes \boldsymbol{\gamma})$.
   - For $\ell \in [1, d]$, define $\mathbf{M}_f^\ell$ as the following matrix.

     $$\mathbf{M}_f^\ell = r[f^\ell(\boldsymbol{x})]\mathbf{A}_f^{\ell+1} - r\mathbf{A}_f^\ell$$

     Let $\boldsymbol{w}_{i,j}^\ell \in \mathbb{Z}_p^{n+2}$ be the vector computed as in the previous hybrid. We define $\boldsymbol{w'}_{i,j}^\ell$ be the vector that equals $\boldsymbol{w}_{i,j}^\ell$ in its first $n+1$ components and is equal to $\mathbf{M}_{f,i,j}^\ell$ in the component $n+2$.
   - For $\ell \in [1, d]$ and $i, j \in [n^3]$, compute $\mathsf{Key}_{f,i,j}^\ell = \mathsf{cIPE.KeyGen}(\mathsf{sk}_1, \boldsymbol{w'}_{i,j}^\ell)$
   - Output $sk_f = (\mathsf{Key}_{0,f}, \{\mathsf{Key}_{f,i,j}^\ell\}_{\ell \in [d], i,j \in [n]})$

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

Lemma 7. *Assuming* cIPE *is secure, then, for all p.p.t. adversaries* $\mathcal{A}$, $|\Pr[\mathcal{A}(\mathbf{Hybrid}_2) = 1] - \Pr[\mathcal{A}(\mathbf{Hybrid}_{2,0}) = 1]| \leq \mathsf{negl}(\lambda)$*. If* cIPE *is subexponentially secure, this gap is subexponentially small.*

Proof. The only difference between the two hybrids is the way CTC_1 (and $\mathsf{CTC}(i)_1$) and CTC_k (and CTK_k) for $k \in [n]$ are generated. CTK_k are generated so that decryption of CTC_j for $j \in [n]$ is the same in both hybrids- $y_j z_k - r\beta_j\gamma_k$.

The security thus follows from the security of cIPE scheme.

□

Hybrid$_{2,1}$: In this hybrid, we rely on SXDH to switch elements.

1. The challenger does setup for the scheme as follows:

   - Sample $\mathsf{pp} \leftarrow \mathsf{cIPE.PPSetup}(1^\lambda)$. Let us assume $\mathsf{pp} = (e, G_1, G_2, G_T, g_1, g_2, \mathbb{Z}_p)$.
   - Run cIPE setup as follows. $\mathsf{sk}_0 \leftarrow \mathsf{cIPE.Setup}(\mathsf{pp}, 1^2)$. Thus these keys are used to encrypt vectors in $\mathbb{Z}_p^2$.
   - Then run cIPE setup algorithm, for vectors of size 1^{n+2}. That is, compute $\mathsf{sk}_1 \leftarrow \mathsf{cIPE.Setup}(\mathsf{pp}, 1^{n+2})$.
   - Sample $\boldsymbol{\beta}, \boldsymbol{\gamma} \leftarrow \mathbb{Z}_p^n$.
   - Output $\mathsf{MSK} = \{\mathsf{sk}_0, \mathsf{sk}_1, \boldsymbol{\beta}, \boldsymbol{\gamma}\}$

2. [**Change**] Challenger samples randomness r from uniformly. Also, sample $u_{1,k}$ for $k \in [n]$ randomly from $\mathbb{Z}_p$. This will be used to replace $r\beta_1\gamma_k$.

3. 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 $\mathbb{Z}_p^{3 \cdot n}$. We now describe how the challenge ciphertext CT is generated. Other ciphertexts $\mathsf{CT}(i)$ for $i \in [L]$ are generated as in the previous hybrid. Note that for ciphertext for message $m(i)$, the components will be denoted as: $\mathsf{CT}(i)_0, \mathsf{CTX}(i), \{\mathsf{CTC}(i)_j, \mathsf{CTK}(i)_k\}_{j,k}$

   Here is how CT is generated.

   - Compute $\mathsf{CT}_0 = \mathsf{cIPE.Enc}(\mathsf{sk}_0, (0,1))$.
   - Sample $\mathsf{sk}' \leftarrow \mathsf{cIPE.Setup}(\mathsf{pp}, 1^4)$.
   - Compute $\mathsf{CTC}_j \leftarrow \mathsf{cIPE.Enc}(\mathsf{sk}', (y_j, \beta_j, 0, 0))$ for $j \in [n] \setminus 1$
   - Compute $\mathsf{CTC}_1 \leftarrow \mathsf{cIPE.Enc}(\mathsf{sk}', (0, 0, 1, 0))$.
   - Compute $\mathsf{CTK}_k \leftarrow \mathsf{cIPE.KeyGen}(\mathsf{sk}', (z_k, -r\gamma_k, y_1 z_k - u_{1,k}, 0))$ for $k \in [n]$.
   - Compute $\mathsf{CTX} = \mathsf{cIPE.Enc}(\mathsf{sk}_1, (0, ...0, 1))$.
   - Output $\mathsf{CT} = (\boldsymbol{x}, \mathsf{CT}_0, \mathsf{CTX}, \{\mathsf{CTC}_j, \mathsf{CTK}_k, \}_{j,k \in [n]})$

4. 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 associated with value Δ.

- Parse $\mathsf{MSK} = \{\mathsf{sk}_0, \mathsf{sk}_1, \boldsymbol{\beta}, \boldsymbol{\gamma}\}$
- Sample matrices $\mathbf{A}_f^\ell$ randomly for $\ell \in [d]$ from $\mathbb{Z}_p^{n^3 \times n^3}$.
- [**Change**] For every function f, sample $\mathbf{B}_f^\ell$ for $\ell \in [d]$ randomly from $\mathbb{Z}_p^{n^3 \times n^3}$.
- Parse $f(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}) = \langle f^0, [f^1(\boldsymbol{x})] \cdot \ldots \cdot [f^d(\boldsymbol{x})] \cdot \mathsf{diagvec}(\boldsymbol{y} \otimes \boldsymbol{z})\rangle$.
- Let $\theta_f = \langle f^0, \mathbf{A}_f^1\rangle$.
- [**Change**] Let $\theta'_f = \langle f^0, \mathbf{B}_f^1\rangle$.
- Compute $\mathsf{Key}_{0,f} = \mathsf{cIPE.KeyGen}(\mathsf{sk}_0, (\theta_f, \theta'_f))$.
- Define $\mathbf{A}_f^{d+1} = \mathsf{diagvec}(\boldsymbol{\beta} \otimes \boldsymbol{\gamma})$. Define $\mathbf{B}_f^{d+1}$ as the matrix $r\mathbf{A}^{d+1}$, except that $r\beta_1\gamma_k$ is replaced with random element $u_{1,k}$.
- [**Change**] For $\ell \in [1, d]$, define $\mathbf{M}_f^\ell$ as the following matrix.

$$\mathbf{M}_f^\ell = r[f^\ell(\boldsymbol{x})]\mathbf{A}_f^{\ell+1} - r\mathbf{A}_f^\ell$$

  Let $\boldsymbol{w}_{i,j}^\ell \in \mathbb{Z}_p^{n+2}$ be the vector computed as in the previous hybrid. We define $\boldsymbol{w}'^\ell_{i,j}$ be the vector that equals $\boldsymbol{w}_{i,j}^\ell$ in its first $n+1$ components and is equal to $\mathbf{M}'^\ell_{f,i,j}$ in the component $n+2$, where $\mathbf{M}'^\ell_f$ is defined below for $\ell \in [d]$.

$$\mathbf{M}'^\ell_f = [f^\ell(\boldsymbol{x})]\mathbf{B}_f^{\ell+1} - \mathbf{B}_f^\ell$$

- For $\ell \in [1, d]$ and $i, j \in [n]$, compute $\mathsf{Key}_{f,i,j}^\ell = \mathsf{cIPE.KeyGen}(\mathsf{sk}_1, \boldsymbol{w}'^\ell_{i,j})$
- Output $sk_f = (\mathsf{Key}_{0,f}, \{\mathsf{Key}_{f,i,j}^\ell\}_{\ell \in [d], i,j \in [n]})$

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

Lemma 8. *If SXDH holds relative to* PPGen*, then, for all polynomial time adversaries* $\mathcal{A}$, $|\Pr[\mathcal{A}(\mathbf{Hybrid}_{2,0}) = 1] - \Pr[\mathcal{A}(\mathbf{Hybrid}_{2,1}) = 1]| \leq \mathsf{negl}(\lambda)$. *If SXDH is subexponentially hard, this gap is subexponentially small.*

Proof. The only difference between the two hybrids is the way hardcodings using $r\beta_1\gamma_k$ for $k \in [n]$ and $r\mathbf{A}_f^\ell$ is generated for $\ell \in [d]$. In $\mathbf{Hybrid}_{2,1}$, the hardcoding $r\beta_1\gamma_k$ used in generating CTK_k is replaced with a random value $v_{1,k}$. Note that this can be done due to canonical structure of cIPE. Exponent r, β_1 and γ_k for all $k \in [n]$, only appear in group G_2. Hence, this follows from SXDH.

At the same time, hardcodings correponding to $r\mathbf{A}_f^\ell$ that appear in $\mathsf{Key}_{f,i,j}^\ell$ is computed by using a random matrix $\mathbf{B}_f^\ell$ instead of $r\mathbf{A}^\ell$ for $\ell \in [d]$ and $i, j \in [n^3]$. Since exponents $\mathbf{A}_f^\ell$ and r appear only in G_2 (due to canonical structure of cIPE), this can also be done due to SXDH. □

$\mathbf{Hybrid}_{2,2}$: In this hybrid, we replace CTK_k to hardcode $0 - u_{1,k}$ instead of $y_1 z_k - u_{1,k}$. This change is accounted for by adding an offset in the hardcoded value.

1. The challenger does setup for the scheme as follows:

- Sample $\mathsf{pp} \leftarrow \mathsf{cIPE.PPSetup}(1^\lambda)$. Let us assume $\mathsf{pp} = (e, G_1, G_2, G_T, g_1, g_2, \mathbb{Z}_p)$.
- Run cIPE setup as follows. $\mathsf{sk}_0 \leftarrow \mathsf{cIPE.Setup}(\mathsf{pp}, 1^2)$. Thus these keys are used to encrypt vectors in $\mathbb{Z}_p^2$.

- Then run cIPE setup algorithm, for vectors of size 1^{n+2}. That is, compute $\mathsf{sk}_1 \leftarrow \mathsf{cIPE.Setup}(\mathsf{pp}, 1^{n+2})$.
- Sample $\boldsymbol{\beta}, \boldsymbol{\gamma} \leftarrow \mathbb{Z}_p^n$.
- Output $\mathsf{MSK} = \{\mathsf{sk}_0, \mathsf{sk}_1, \boldsymbol{\beta}, \boldsymbol{\gamma}\}$

2. [**Change**] Challenger samples randomness r from uniformly. Let $\mathbf{A}^{d+1} = \mathsf{diagvec}(\boldsymbol{\beta} \otimes \boldsymbol{\gamma})$. Also, sample $u_{1,k}$ for $k \in [n]$ randomly from $\mathbb{Z}_p$. This will be used to replace $r\beta_1\gamma_k$.

3. 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 $\mathbb{Z}_p^{3 \cdot n}$. We now describe how the challenge ciphertext CT is generated. Other ciphertexts $\mathsf{CT}(i)$ for $i \in [L]$ are generated as in the previous hybrid. Note that for ciphertext for message $m(i)$, the components will be denoted as: $\mathsf{CT}(i)_0, \mathsf{CTX}(i), \{\mathsf{CTC}(i)_j, \mathsf{CTK}(i)_k\}_{j,k}$

   Here is how CT is generated.

   - Compute $\mathsf{CT}_0 = \mathsf{cIPE.Enc}(\mathsf{sk}_0, (0,1))$.
   - Sample $\mathsf{sk}' \leftarrow \mathsf{cIPE.Setup}(\mathsf{pp}, 1^4)$.
   - Compute $\mathsf{CTC}_j \leftarrow \mathsf{cIPE.Enc}(\mathsf{sk}', (y_j, \beta_j, 0, 0))$ for $j \in [n] \setminus 1$
   - Compute $\mathsf{CTC}_1 \leftarrow \mathsf{cIPE.Enc}(\mathsf{sk}', (0, 0, 1, 0))$.
   - [**Change**] Compute $\mathsf{CTK}_k \leftarrow \mathsf{cIPE.KeyGen}(\mathsf{sk}', (z_k, -r\gamma_k, -u_{1,k}, 0))$ for $k \in [n]$.
   - Compute $\mathsf{CTX} = \mathsf{cIPE.Enc}(\mathsf{sk}_1, (0, ...0, 1))$.
   - Output $\mathsf{CT} = (\boldsymbol{x}, \mathsf{CT}_0, \mathsf{CTX}, \{\mathsf{CTC}_j, \mathsf{CTK}_k, \}_{j,k \in [n]})$

4. 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 associated with value Δ.

   - Parse $\mathsf{MSK} = \{\mathsf{sk}_0, \mathsf{sk}_1, \boldsymbol{\beta}, \boldsymbol{\gamma}\}$
   - Sample $\mathbf{A}_f^1, ..., \mathbf{A}_f^d$ randomly from $\mathbb{Z}_p^{n^3 \times n^3}$.
   - Sample $\mathbf{B}_f^1, ..., \mathbf{B}_f^d$ randomly from $\mathbb{Z}_p^{n^3 \times n^3}$.
   - Parse $f(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}) = \langle f^0, [f^1(\boldsymbol{x})] \cdot \ldots \cdot [f^d(\boldsymbol{x})] \cdot \mathsf{diagvec}(\boldsymbol{y} \otimes \boldsymbol{z}) \rangle$.
   - Let $\theta_f = \langle f^0, \mathbf{A}_f^1 \rangle$.
   - [**Change**] Let $\theta'_f = \langle f^0, \mathbf{B}_f^1 \rangle + f(\boldsymbol{x}, \boldsymbol{y}_0, \boldsymbol{z})$. Here $\boldsymbol{y}_0 = (y_1, 0, ..., 0)$.
   - Compute $\mathsf{Key}_{0,f} = \mathsf{cIPE.KeyGen}(\mathsf{sk}_0, (\theta_f, \theta'_f))$.
   - Define $\mathbf{A}_f^{d+1} = \mathsf{diagvec}(\boldsymbol{\beta} \otimes \boldsymbol{\gamma})$. Define $\mathbf{B}_f^{d+1}$ as the matrix $r\mathbf{A}_f^{d+1}$, except that $r\beta_1\gamma_k$ is replaced with random element $u_{1,k}$.
   - For $\ell \in [1, d]$, define $\mathbf{M}_f^\ell$ as the following matrix.

$$\mathbf{M}_f^\ell = r[f^\ell(\boldsymbol{x})]\mathbf{A}_f^{\ell+1} - r\mathbf{A}_f^\ell$$

Let $\boldsymbol{w}^{\ell}_{i,j} \in \mathbb{Z}_p^{n+2}$ be the vector computed as in the previous hybrid. We define $\boldsymbol{w'}^{\ell}_{i,j}$ be the vector that equals $\boldsymbol{w}^{\ell}_{i,j}$ in its first $n+1$ components and is equal to $\mathbf{M'}^{\ell}_{f,i,j}$ in the component $n+2$, where $\mathbf{M'}^{\ell}_{f}$ is defined below for $\ell \in [d]$.

$$\mathbf{M'}^{\ell}_{f} = [f^{\ell}(\boldsymbol{x})]\mathbf{B}^{\ell+1}_{f} - \mathbf{B}^{\ell}_{f}$$

- For $\ell \in [1,d]$ and $i,j \in [n]$, compute $\mathsf{Key}^{\ell}_{f,i,j} = \mathsf{cIPE.KeyGen}(\mathsf{sk}_1, \boldsymbol{w'}^{\ell}_{i,j})$
- Output $sk_f = (\mathsf{Key}_{0,f}, \{\mathsf{Key}^{\ell}_{f,i,j}\}_{\ell\in[d],i,j\in[n^3]})$

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

Lemma 9. *For any adversary $\mathcal{A}$,* $|\Pr[\mathcal{A}(\mathbf{Hybrid}_{2,1}) = 1] - \Pr[\mathcal{A}(\mathbf{Hybrid}_{2,2})]| = 0$.

Proof. To prove this, we consider following distributions. First distribution corresponds to hard-codings corresponding to $\mathbf{Hybrid}_{2,1}$ and the last one corresponds to $\mathbf{Hybrid}_{2,2}$.

Distribution 1:

- Set $u_{i,j} = r\beta_i\gamma_j$ for $i \in [2,n]$ and $j \in [n]$ and $u_{1,j}$ as a randomly sampled element in $\mathbb{Z}_p$ for $j \in [n]$. Let $\mathbf{U}$ be the matrix denoted by the entries $u_{i,j}$.
- Denote $-\tilde{u}_{1,k} = y_1 z_k - u_{1,k}$ for $k \in [n]$. Output the following.
- $-\tilde{u}_{1,k}$ for $k \in [n]$.
- $\mathbf{M}^{\ell}_{f} = [f^{\ell}(\boldsymbol{x})]\mathbf{B}^{\ell+1}_{f} - \mathbf{B}^{\ell}_{f}$ for $\ell \in [d]$, and all queried functions f.
- $\theta'_f = \langle f_0, \mathbf{B}^1_f\rangle$ for all queried f.

Rewriting a bit differently and substuiting the following we get the next distribution.
Set $\tilde{\mathbf{B}}^{d+1}_{f} = \mathbf{B}^{d+1}_{f} - \mathsf{diagvec}(\boldsymbol{y}_1 \otimes \boldsymbol{z})$.
For $\ell \in [d]$, $\tilde{\mathbf{B}}^{\ell}_{f} = \mathbf{B}^{\ell}_{f} - [f^{\ell}(\boldsymbol{x})] \cdot \ldots \cdot [f^{1}(\boldsymbol{x})]\mathsf{diagvec}(\boldsymbol{y}_1 \otimes \boldsymbol{z})$.
Observe that $\mathbf{M}^{\ell}_{f} = [f^{\ell}(\boldsymbol{x})]\tilde{\mathbf{B}}^{\ell+1}_{f} - \tilde{\mathbf{B}}^{\ell}_{f}$ for $\ell \in [d]$

Distribution 2:

- $-\tilde{u}_{1,k}$ for $k \in [n]$.
- $\mathbf{M}^{\ell}_{f} = [f^{\ell}(\boldsymbol{x})]\tilde{\mathbf{B}}^{\ell+1}_{f} - \tilde{\mathbf{B}}^{\ell}_{f}$ for $\ell \in [d]$, and all queried functions f.
- $\theta'_f = \langle f_0, \mathbf{B}^1_f\rangle = \langle f_0, \tilde{\mathbf{B}}^1_f + [f_1(\boldsymbol{x})] \cdot \ldots \cdot [f_d(\boldsymbol{x})] \cdot \mathsf{diagvec}(\boldsymbol{y_1} \otimes \boldsymbol{z})\rangle$ for all queried f. Thus, $\theta'_f = \langle f_0, \tilde{\mathbf{B}}^1_f\rangle + f(\boldsymbol{x}, \boldsymbol{y}_1, \boldsymbol{z})$.

Now observe that $\mathbf{B}^{\ell}_{f}$ is uniformly chosen and thus the distribution $\{\tilde{\mathbf{B}}^{\ell}_{f} = \mathbf{B}^{\ell}_{f} - [f^{\ell}(\boldsymbol{x})] \cdot \ldots \cdot [f^{1}(\boldsymbol{x})]\mathsf{diagvec}(\boldsymbol{y}_1 \otimes \boldsymbol{z})\}_{\ell\in[d+1]}$ is identical to $\{\mathbf{B}^{\ell}_{f}\}_{\ell\in[d+1]}$. Thus writing it differently we get the third distribution.

Distribution 3:

- $0 - u_{1,k}$ for $k \in [n]$.
- $\mathbf{M}_f^\ell = [f^\ell(\boldsymbol{x})]\mathbf{B}_f^{\ell+1} - \mathbf{B}_f^\ell$ for $\ell \in [d]$
- $\theta'_f = \langle f_0, \mathbf{B}_f^1 \rangle + f(\boldsymbol{x}, \boldsymbol{y}_1, \boldsymbol{z})$.

□

Hybrid$_{2,3}$: In this hybrid, we replace $u_{1,k} = r\beta_1\gamma_k$ and $\mathbf{B}_f^\ell = r\mathbf{A}_f^\ell$ for $k \in [n]$ and $\ell \in [d]$.

1. The challenger does setup for the scheme as follows:
   - Sample $\mathsf{pp} \leftarrow \mathsf{cIPE.PPSetup}(1^\lambda)$. Let us assume $\mathsf{pp} = (e, G_1, G_2, G_T, g_1, g_2, \mathbb{Z}_p)$.
   - Run cIPE setup as follows. $\mathsf{sk}_0 \leftarrow \mathsf{cIPE.Setup}(\mathsf{pp}, 1^2)$. Thus these keys are used to encrypt vectors in $\mathbb{Z}_p^2$.
   - Then run cIPE setup algorithm, for vectors of size 1^{n+2}. That is, compute $\mathsf{sk}_1 \leftarrow \mathsf{cIPE.Setup}(\mathsf{pp}, 1^{n+2})$.
   - Sample $\boldsymbol{\beta}, \boldsymbol{\gamma} \leftarrow \mathbb{Z}_p^n$.
   - Output $\mathsf{MSK} = \{\mathsf{sk}_0, \mathsf{sk}_1, \boldsymbol{\beta}, \boldsymbol{\gamma}\}$
2. Challenger samples randomness r from uniformly. Let $\mathbf{A}^{d+1} = \mathsf{diagvec}(\boldsymbol{\beta} \otimes \boldsymbol{\gamma})$.
3. 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 $\mathbb{Z}_p^{3 \cdot n}$. We now describe how the challenge ciphertext CT is generated. Other ciphertexts $\mathsf{CT}(i)$ for $i \in [L]$ are generated as in the previous hybrid. Note that for ciphertext for message $m(i)$, the components will be denoted as: $\mathsf{CT}(i)_0, \mathsf{CTX}(i), \{\mathsf{CTC}(i)_j, \mathsf{CTK}(i)_k\}_{j,k}$

   Here is how CT is generated.
   - Compute $\mathsf{CT}_0 = \mathsf{cIPE.Enc}(\mathsf{sk}_0, (0, 1))$.
   - Sample $\mathsf{sk}' \leftarrow \mathsf{cIPE.Setup}(\mathsf{pp}, 1^4)$.
   - Compute $\mathsf{CTC}_j \leftarrow \mathsf{cIPE.Enc}(\mathsf{sk}', (y_j, \beta_j, 0, 0))$ for $j \in [n] \setminus 1$
   - Compute $\mathsf{CTC}_1 \leftarrow \mathsf{cIPE.Enc}(\mathsf{sk}', (0, 0, 1, 0))$.
   - [**Change**] Compute $\mathsf{CTK}_k \leftarrow \mathsf{cIPE.KeyGen}(\mathsf{sk}', (z_k, -r\gamma_k, -r\beta_1 \cdot \gamma_k, 0))$ for $k \in [n]$.
   - Compute $\mathsf{CTX} = \mathsf{cIPE.Enc}(\mathsf{sk}_1, (0, ...0, 1))$.
   - Output $\mathsf{CT} = (\boldsymbol{x}, \mathsf{CT}_0, \mathsf{CTX}, \{\mathsf{CTC}_j, \mathsf{CTK}_k, \}_{j,k \in [n]})$
4. 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 associated with value Δ.
   - Parse $\mathsf{MSK} = \{\mathsf{sk}_0, \mathsf{sk}_1, \boldsymbol{\beta}, \boldsymbol{\gamma}\}$
   - Sample $\mathbf{A}_f^1, ..., \mathbf{A}_f^d$ randomly from $\mathbb{Z}_p^{n \times n}$

- Parse $f(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}) = \langle f^0, [f^1(\boldsymbol{x})] \cdot \ldots \cdot [f^d(\boldsymbol{x})] \cdot \mathsf{diagvec}(\boldsymbol{y} \otimes \boldsymbol{z})\rangle$.
- Let $\theta_f = \langle f^0, \mathbf{A}_f^1 \rangle$.
- [**Change**] Let $\theta'_f = \theta_f + f(\boldsymbol{x}, \boldsymbol{y}_0, \boldsymbol{z})$. Here $\boldsymbol{y}_0 = (y_1, 0, ..., 0)$.
- Compute $\mathsf{Key}_{0,f} = \mathsf{cIPE.KeyGen}(\mathsf{sk}_0, (\theta_f, \theta'_f))$.
- Define $\mathbf{A}_f^{d+1} = \mathsf{diagvec}(\boldsymbol{\beta} \otimes \boldsymbol{\gamma})$.
- For $\ell \in [1, d]$, define $\mathbf{M}_f^\ell$ as the following matrix.

$$\mathbf{M}_f^\ell = r[f^\ell(\boldsymbol{x})]\mathbf{A}_f^{\ell+1} - r\mathbf{A}_f^\ell$$

  Let $\boldsymbol{w}_{i,j}^\ell \in \mathbb{Z}_p^{n+2}$ be the vector computed as in the previous hybrids.
- [**Change**] For $\ell \in [1, d]$ and $i, j \in [n]$, compute $\mathsf{Key}_{f,i,j}^\ell = \mathsf{cIPE.KeyGen}(\mathsf{sk}_1, \boldsymbol{w}_{i,j}^\ell)$
- Output $sk_f = (\mathsf{Key}_{0,f}, \{\mathsf{Key}_{f,i,j}^\ell\}_{\ell \in [d], i,j \in [n^3]})$

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

The proof of this is similar to the proof of lemma 8

Lemma 10. *If SXDH holds relative to* PPGen*, then, for all polynomial time adversaries* $\mathcal{A}$*,* $|\Pr[\mathcal{A}(\mathbf{Hybrid}_{2,2}) = 1] - \Pr[\mathcal{A}(\mathbf{Hybrid}_{2,3}) = 1]| \leq \mathsf{negl}(\lambda)$*. If SXDH is subexponentially hard, this gap is subexponentially small.*

The proof of the following lemma is similar to lemma 7

Lemma 11. *Assuming* cIPE *is secure, then, for all p.p.t. adversaries* $\mathcal{A}$*,* $|\Pr[\mathcal{A}(\mathbf{Hybrid}_{2,3}) = 1] - \Pr[\mathcal{A}(\mathbf{Hybrid}_3) = 1]| \leq \mathsf{negl}(\lambda)$*. If* cIPE *is subexponentially secure, this gap is subexponentially small.*

E Security Proof for FE Construction

We now present semi-functional encryption and key generation algorithms and then argue indistinguishability. First we define semi-functional encryption algorithm:

$\underline{\mathsf{sfEnc}(\mathsf{MSK}, 1^n)}$: The encryption algorithm takes as input $\mathsf{MSK} = (\mathsf{PHFE.MSK}, \mathsf{SHE.PK}, \boldsymbol{s}^*, \mathsf{SHE.CT}^*, \mathsf{Seed}^*)$ and does the following.

1. Parse $\mathsf{Seed}^* = (\mathsf{Seed}^*.\mathsf{pub}, \mathsf{Seed}^*.\mathsf{priv}(1), \mathsf{Seed}^*.\mathsf{priv}(2))$.
2. Set $\boldsymbol{x} = (\mathsf{Seed}^*.\mathsf{pub}, 1)$
3. $\mathsf{PHFE.sfEnc}(\mathsf{MSK}, \boldsymbol{x}, 1^{n'}) \rightarrow \mathsf{PHFE.ct}_{\mathsf{sf}}$.
4. Output $(\mathsf{CT}_1 = \mathsf{SHE.CT}^*, \mathsf{CT}_2 = \mathsf{PHFE.ct}_{\mathsf{sf}})$

Now we describe our semi-functional key generation algorithm:

$\underline{\mathsf{sfKG}(\mathsf{MSK}, C, \boldsymbol{\theta} = (\theta_1, ..., \theta_\eta))}$: The semi-functional keygen algorithm takes as input $\mathsf{MSK} = (\mathsf{PHFE.MSK}, \mathsf{SHE.PK}, \boldsymbol{s}^*, \mathsf{SHE.CT}^*, \mathsf{Seed}^*)$ and a circuit $C \in \mathcal{C}_{n,\lambda,\epsilon}$ along with value $\boldsymbol{\theta} = (\theta_1, ..., \theta_\eta)$. Here η is the length of the output of C.

1. Parse $C = (C_1, \ldots, C_\eta)$ where C_i is the circuit that outputs i^{th} bit of C. Here $\eta \leq n^{1+\epsilon}$.
2. For each $i \in [\eta]$, compute $\mathsf{SHE.EvalCT}(\mathsf{SHE.PK}, C_i, \mathsf{SHE.CT}^*,) \rightarrow \mathsf{CT}^*_{C_i}$.
3. Sample $\mathcal{F}\text{-}\Delta\mathsf{RG.SetupPoly}(1^\lambda, 1^n, \mathsf{Bound}_\Delta) \rightarrow (q_1, ..., q_{n^{1+\epsilon'}})$.
4. Run $\mathcal{H}(q_1, ..., q_{n^{1+\epsilon'}}, \mathsf{Seed}^*) \rightarrow (h_1, ..., h_{n^{1+\epsilon'}})$
5. Compute $\gamma_i \leftarrow \mathsf{CT}^*_{C_i} - \theta_i \cdot Q - h_i$.
6. Let G_i be the following arithmetic circuit. On input, vectors $(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z})$ where $\boldsymbol{x} = (\mathsf{Seed.pub}, 1)$, $\boldsymbol{y} = (\mathsf{Seed.priv}(1), \boldsymbol{S})$ and $\boldsymbol{z} = (\mathsf{Seed.priv}(2), 1)$, G_i outputs $\boldsymbol{s}^T \cdot \boldsymbol{b}_{C_i} - q_i(\mathsf{Seed})$. Note that the computations are done over $\mathbb{Z}_p$ and thus $G_i \in \mathcal{F}$.
7. For each $i \in [\eta]$, compute $\mathsf{PHFE}.sk[G_i, \gamma_i] \leftarrow \mathsf{PHFE.sfKG}(\mathsf{PHFE.MSK}, G_i, \gamma_i)$.
8. Output $sk_C = (\mathsf{PHFE}.sk[G_1, \gamma_1], \ldots, \mathsf{PHFE}.sk[G_\eta, \gamma_\eta])$

Observe that the decryption of a normal ciphertext m with a semi-functional key for circuit C with hardwired value $\boldsymbol{\theta}$, outputs $C(m)$. Whereas, the decryption of a semi-functional ciphertext with such a key will output $\boldsymbol{\theta}$.

Now observe that it is trivial to argue indistinguishability of the semi-functional key property. The only difference between the honest key generation algorithm and semi-functional key generation algorithm is that to generate PHFE keys for circuits G_i, semi-functional key generation algorithm relies on semi-functional key generation algorithm of PHFE scheme. Note that hardwirings γ_i is independent of PHFE.MSK. Thus, this property follows directly from the security of PHFE scheme.

Here is the theorem:

Theorem 12. *Let $s(\lambda)$ be a parameter larger than any polynomial. Assuming* PHFE *satisfies indistinguishability of semi-functional key properties for adversaries of size $O(s)$, then the scheme described above satisfies indistinguishability of semi-functional key property against adversaries of size $O(s)$.*

Now we show that FE indistinguishability of semi-functional ciphertext property. We will prove the following:

Theorem 13. *Let $s(\lambda)$ be any size parameter larger than any polynomial. Then assuming,*

- *Security of LWE with subexponential approximation factors against adversaries of size $O(s)$.*
- PHFE *for $\mathcal{F}$ secure against adversaries of size $O(s)$.*
- $\mathcal{F}$-ΔRG *satisfying $(O(s), \mathsf{adv})-$perturbation resilience.*

Then, FE *scheme described above satisfies $(O(s), \mathsf{adv} + \mathsf{negl})-$indistinguishability of semi-functional ciphertexts property.*

We now present hybrids and argue indistinguishability between them. The first hybrid corresponds to the case when the challenge ciphertext is honestly encrypted, where as the last hybrid correspond to the case when it is semi-functionally encrypted. Function key is semi-functionally generated in both these hybrids. Then, we analyze the indistinguishability between them.

Hybrid$_0$: In this hybrid, all ciphertexts are generated honestly using honest encryption algorithm. Function key is generated semi-functionally.

1. Adversary outputs length 1^n, challenge message $\overline{m}$ along with messages $m_1, ..., m_\Gamma$. It also outputs a circuit $C \in \mathcal{C}_{\lambda,n,\epsilon}$. Let $C = (C_1, ..., C_\eta)$ where each sub-circuit C_i outputs i^{th} bit of C.

2. Challenger does setup. We recall the algorithm below.

   (a) Run $\mathsf{PHFE.Setup}(1^\lambda, 1^{n'}) \to \mathsf{PHFE.MSK}$. Here $n' = n \cdot \mathsf{poly}(\lambda)$.

   (b) Run $\mathsf{SHE.Setup}(1^\lambda, 1^n) \to \mathsf{SHE.PK}$.

   (c) Sample a secret vector $\boldsymbol{s}^* \in \chi^{\dim_1 \times 1}$. Set $\boldsymbol{s}^*[1,1] = 1$.

   (d) Compute $\mathsf{SHE.Encode}(\mathsf{SHE.PK}, \boldsymbol{s}^*, 0^n) \to \mathsf{SHE.CT}^*$. Run $\mathcal{F}\text{-}\Delta\mathsf{RG.SetupSeed}(1^\lambda, 1^n, \mathsf{Bound}_\Delta) \to \mathsf{Seed}^*$.

   (e) Output $\mathsf{MSK} = (\mathsf{PHFE.MSK}, \mathsf{SHE.PK}, \boldsymbol{s}^*, \mathsf{SHE.CT}^*, \mathsf{Seed}^*)$

3. Challenger encrypts all messages using honest encryption algorithm and hands over the ciphertexts to the adversary. These ciphertexts are denoted $\overline{\mathsf{CT}}, \mathsf{CT}[1], ..., \mathsf{CT}[\Gamma]$. We describe below, how challenge ciphertext is encrypted.

   (a) Sample a secret $\boldsymbol{s} \leftarrow \chi^{\dim_1 \times 1}$. Set $\boldsymbol{s}[1,1] = 1$.

   (b) Compute $\mathsf{SHE.Encode}(\mathsf{SHE.PK}, \boldsymbol{s}, \overline{m}) \to \mathsf{SHE}.\overline{\mathsf{CT}}$.

   (c) Run $\mathcal{F}\text{-}\Delta\mathsf{RG.SetupSeed}(1^\lambda, 1^n, \mathsf{Bound}_\Delta) \to \mathsf{Seed}$. Parse $\mathsf{Seed} = (\mathsf{Seed.pub}, \mathsf{Seed.priv}(1), \mathsf{Seed.priv}(2))$.

   (d) Set $\boldsymbol{x} = (\mathsf{Seed.pub}, 1)$, $\boldsymbol{y} = (\mathsf{Seed.priv}(1), \boldsymbol{s})$ and $\boldsymbol{z} = (\mathsf{Seed.priv}(2), 1)$.

   (e) $\mathsf{PHFE.Enc}(\mathsf{MSK}, (\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z})) \to \mathsf{PHFE}.\overline{\mathsf{CT}}$.

   (f) Output $\overline{\mathsf{CT}} = (\overline{\mathsf{CT}}_1 = \mathsf{SHE}.\overline{\mathsf{CT}}, \overline{\mathsf{CT}}_2 = \mathsf{PHFE}.\overline{\mathsf{CT}})$

4. Function key for circuit C is generated as follows using the semi-functional key generation algorithm. Let $\theta_i = C_i(\overline{m})$ for $i \in [\eta]$.

   (a) Parse $C = (C_1, \ldots, C_\eta)$ where C_i is the circuit that outputs i^{th} bit of C. Here $\eta \leq n^{1+\epsilon}$.

   (b) For each $i \in [\eta]$, compute $\mathsf{SHE.EvalCT}(\mathsf{SHE.PK}, C_i, \mathsf{SHE.CT}^*) \to \mathsf{CT}^*_{C_i}$.

   (c) Sample $\mathcal{F}\text{-}\Delta\mathsf{RG.SetupPoly}(1^\lambda, 1^n, \mathsf{Bound}_\Delta) \to (q_1, ..., q_{n^{1+\epsilon'}})$. Run $\mathcal{H}(q_1, ..., q_{n^{1+\epsilon}}, \mathsf{Seed}^*) = (h_1, ..., h_{n^{1+\epsilon}})$.

   (d) Compute $\gamma_i \leftarrow \mathsf{CT}^*_{C_i} - \theta_i \cdot Q - h_i$.

   (e) Let G_i be the following arithmetic circuit. On input, vectors $(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z})$ where $\boldsymbol{x} = (\mathsf{Seed.pub}, 1)$, $\boldsymbol{y} = (\mathsf{Seed.priv}(1), \boldsymbol{s})$ and $\boldsymbol{z} = (\mathsf{Seed.priv}(2), 1)$, G_i outputs $\boldsymbol{s}^T \cdot \boldsymbol{b}_{C_i} - q_i(\mathsf{Seed})$. Note that the computations are done over $\mathbb{Z}_p$ and thus $G_i \in \mathcal{F}$.

   (f) For each $i \in [\eta]$, compute $\mathsf{PHFE}.sk[G_i, \gamma_i] \leftarrow \mathsf{PHFE.sfKG}(\mathsf{PHFE.MSK}, G_i, \gamma_i)$.

   (g) Output $sk_C = (\mathsf{PHFE}.sk[G_1, \gamma_1], \ldots, \mathsf{PHFE}.sk[G_\eta, \gamma_\eta])$

5. Adversary is handed over the function key. Finally, adversary outputs a guess $\rho \in \{0, 1\}$.

Hybrid$_1$: This hybrid is the same as the previous hybrid except that hardwirings γ_i used in generating semi-functional keys are generated differently. The change from the previous hybrid is marked in boldfaced [**Change**].

1. Adversary outputs length 1^n, challenge message $\overline{m}$ along with messages $m_1, ..., m_\Gamma$. It also outputs a circuit $C \in \mathcal{C}_{\lambda,n,\epsilon}$. Let $C = (C_1, ..., C_\eta)$ where each sub-circuit C_i outputs i^{th} bit of C.

2. Challenger does setup. We recall the algorithm below.

   (a) Run $\mathsf{PHFE.Setup}(1^\lambda, 1^{n'}) \to \mathsf{PHFE.MSK}$. Here $n' = n \cdot \mathsf{poly}(\lambda)$.
   (b) Run $\mathsf{SHE.Setup}(1^\lambda, 1^n) \to \mathsf{SHE.PK}$.
   (c) Sample a secret vector $\boldsymbol{s}^* \in \chi^{\dim_1 \times 1}$. Set $\boldsymbol{s}^*[1,1] = 1$.
   (d) Compute $\mathsf{SHE.Encode}(\mathsf{SHE.PK}, \boldsymbol{S}^*, 0^n) \to \mathsf{SHE.CT}^*$. Run $\mathcal{F}\text{-}\Delta\mathsf{RG.SetupSeed}(1^\lambda, 1^n, \mathsf{Bound}_\Delta) \to \mathsf{Seed}^*$.
   (e) Output $\mathsf{MSK} = (\mathsf{PHFE.MSK}, \mathsf{SHE.PK}, \boldsymbol{s}^*, \mathsf{SHE.CT}^*, \mathsf{Seed}^*)$

3. Challenger encrypts all messages using honest encryption algorithm and hands over the ciphertexts to the adversary. These ciphertexts are denoted $\overline{\mathsf{CT}}, \mathsf{CT}[1], ..., \mathsf{CT}[\Gamma]$. We describe below, how challenge ciphertext is encrypted.

   (a) Sample a secret $\boldsymbol{s} \leftarrow \chi^{\dim_1 \times 1}$. Set $\boldsymbol{s}[1,1] = 1$.
   (b) Compute $\mathsf{SHE.Encode}(\mathsf{SHE.PK}, \boldsymbol{s}, \overline{m}) \to \mathsf{SHE}.\overline{\mathsf{CT}}$.
   (c) Run $\mathcal{F}\text{-}\Delta\mathsf{RG.SetupSeed}(1^\lambda, 1^n, \mathsf{Bound}_\Delta) \to \mathsf{Seed}$. Parse $\mathsf{Seed} = (\mathsf{Seed.pub}, \mathsf{Seed.priv}(1), \mathsf{Seed.priv}(2))$.
   (d) Set $\boldsymbol{x} = (\mathsf{Seed.pub}, 1)$, $\boldsymbol{y} = (\mathsf{Seed.priv}(1), \boldsymbol{s})$ and $\boldsymbol{z} = (\mathsf{Seed.priv}(2), 1)$.
   (e) $\mathsf{PHFE.Enc}(\mathsf{MSK}, (\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z})) \to \mathsf{PHFE}.\overline{\mathsf{CT}}$.
   (f) Output $\overline{\mathsf{CT}} = (\overline{\mathsf{CT}}_1 = \mathsf{SHE}.\overline{\mathsf{CT}}, \overline{\mathsf{CT}}_2 = \mathsf{PHFE}.\overline{\mathsf{CT}})$

4. Function key for circuit C is generated as follows using the semi-functional key generation algorithm. Let $\theta_i = C_i(\overline{m})$ for $i \in [\eta]$.

   (a) Parse $C = (C_1, \ldots, C_\eta)$ where C_i is the circuit that outputs i^{th} bit of C. Here $\eta \leq n^{1+\epsilon}$.
   (b) [**Change**] For $i \in [\eta]$, compute $\mathsf{SHE.EvalPK}(\mathsf{PK}, C_i) \to \boldsymbol{b}_{C_i}$.
   (c) Sample $\mathcal{F}\text{-}\Delta\mathsf{RG.SetupPoly}(1^\lambda, 1^n, \mathsf{Bound}_\Delta) \to (q_1, ..., q_{n^{1+\epsilon'}})$.
   (d) [**Change**] Compute $\gamma_i \leftarrow \boldsymbol{s}^T \cdot \boldsymbol{b}_{C_i} - q_i(\mathsf{Seed})$. Set $\boldsymbol{x} = (\mathsf{Seed.pub}, 1)$, $\boldsymbol{y} = (\mathsf{Seed.Priv}(1), \boldsymbol{s})$ and $\boldsymbol{z} = (\mathsf{Seed.Priv}(2), 1)$. Note that $G_i(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}) = \gamma_i$.
   (e) For each $i \in [\eta]$, compute $\mathsf{PHFE}.sk[G_i, \gamma_i] \leftarrow \mathsf{PHFE.sfKG}(\mathsf{PHFE.MSK}, G_i, \gamma_i)$.
   (f) Output $sk_C = (\mathsf{PHFE}.sk[G_1, \gamma_1], \ldots, \mathsf{PHFE}.sk[G_\eta, \gamma_\eta])$

5. Adversary is handed over the function key. Finally, adversary outputs a guess $\rho \in \{0, 1\}$.

Lemma 12. *Let $s(\lambda)$ be a parameter greater than any polynomial in λ. If* PHFE *satisfies indistinguishability of semi-functional keys property against adversaries of size $O(s)$, then for any adversary $\mathcal{A}$ of size $O(s)$,* $|\Pr[\mathcal{A}(\mathbf{Hybrid}_0) = 1] - \Pr[\mathcal{A}(\mathbf{Hybrid}_1) = 1]| \leq \mathsf{negl}(\lambda)$.

Proof. The only difference between the hybrids is that in $\mathbf{Hybrid}_0$, for each $i \in [\eta]$, the hardwirings γ_i are generated as in semi-functional key generation algorithm, where as in $\mathbf{Hybrid}_1$, they are generated as $G_i(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z})$ where $\mathsf{PHFE}.\overline{\mathsf{CT}}$ encrypts $(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z})$. Note that in both cases γ_i is independent of $\mathsf{PHFE}.\mathsf{MSK}$. Indistinguishability now follows from the security of PHFE scheme. $\square$

Hybrid$_2$: This hybrid is the same as the previous hybrid except that the challenge ciphertext component $\mathsf{PHFE}.\overline{\mathsf{CT}}$ is semi-functionally generated.

1. Adversary outputs length 1^n, challenge message $\overline{m}$ along with messages $m_1, ..., m_\Gamma$. It also outputs a circuit $C \in \mathcal{C}_{\lambda,n,\epsilon}$. Let $C = (C_1, ..., C_\eta)$ where each sub-circuit C_i outputs i^{th} bit of C.

2. Challenger does setup. We recall the algorithm below.

   (a) Run $\mathsf{PHFE.Setup}(1^\lambda, 1^{n'}) \to \mathsf{PHFE.MSK}$. Here $n' = n \cdot \mathsf{poly}(\lambda)$.

   (b) Run $\mathsf{SHE.Setup}(1^\lambda, 1^n) \to \mathsf{SHE.PK}$.

   (c) Sample a secret vector $\boldsymbol{s}^* \in \chi^{\dim_1 \times 1}$. Set $\boldsymbol{s}^*[1,1] = 1$.

   (d) Compute $\mathsf{SHE.Encode}(\mathsf{SHE.PK}, \boldsymbol{S}^*, 0^n) \to \mathsf{SHE.CT}^*$. Run $\mathcal{F}\text{-}\Delta\mathsf{RG.SetupSeed}(1^\lambda, 1^n, \mathsf{Bound}_\Delta)$ $\to \mathsf{Seed}^*$.

   (e) Output $\mathsf{MSK} = (\mathsf{PHFE.MSK}, \mathsf{SHE.PK}, \boldsymbol{s}^*, \mathsf{SHE.CT}^*, \mathsf{Seed}^*)$

3. Challenger encrypts all messages and hands over the ciphertexts to the adversary. These ciphertexts are denoted $\overline{\mathsf{CT}}, \mathsf{CT}[1], ..., \mathsf{CT}[\Gamma]$. We describe below, how challenge ciphertext is encrypted. Other ciphertexts are generated as in the previous hybrid.

   (a) Sample a secret $\boldsymbol{s} \leftarrow \chi^{\dim_1 \times 1}$. Set $\boldsymbol{s}[1,1] = 1$.

   (b) Compute $\mathsf{SHE.Encode}(\mathsf{SHE.PK}, \boldsymbol{s}, \overline{m}) \to \mathsf{SHE}.\overline{\mathsf{CT}}$.

   (c) Run $\mathcal{F}\text{-}\Delta\mathsf{RG.SetupSeed}(1^\lambda, 1^n, \mathsf{Bound}_\Delta) \to \mathsf{Seed}$. Parse $\mathsf{Seed} = (\mathsf{Seed.pub}, \mathsf{Seed.priv}(1), \mathsf{Seed.priv}(2))$.

   (d) Set $\boldsymbol{x} = (\mathsf{Seed.pub}, 1)$, $\boldsymbol{y} = (\mathsf{Seed.priv}(1), \boldsymbol{s})$ and $\boldsymbol{z} = (\mathsf{Seed.priv}(2), 1)$.

   (e) [**Change**] $\mathsf{PHFE.sfEnc}(\mathsf{MSK}, (\boldsymbol{x}, 1^{n'})) \to \mathsf{PHFE}.\overline{\mathsf{CT}}$.

   (f) Output $\overline{\mathsf{CT}} = (\overline{\mathsf{CT}}_1 = \mathsf{SHE}.\overline{\mathsf{CT}}, \overline{\mathsf{CT}}_2 = \mathsf{PHFE}.\overline{\mathsf{CT}})$

4. Function key for circuit C is generated as follows using the semi-functional key generation algorithm. Let $\theta_i = C(\overline{m})$ for $i \in [\eta]$.

   (a) Parse $C = (C_1, \ldots, C_\eta)$ where C_i is the circuit that outputs i^{th} bit of C. Here $\eta \leq n^{1+\epsilon}$.

   (b) For $i \in [\eta]$, compute $\mathsf{SHE.EvalPK}(\mathsf{SHE.PK}, C_i) \to \boldsymbol{b}_{C_i}$.

   (c) Sample $\mathcal{F}\text{-}\Delta\mathsf{RG.SetupPoly}(1^\lambda, 1^n, \mathsf{Bound}_\Delta) \to (q_1, ..., q_{n^{1+\epsilon'}})$.

   (d) Compute $\gamma_i \leftarrow \boldsymbol{s}^T \cdot \boldsymbol{b}_{C_i} - q_i(\mathsf{Seed})$. Set $\boldsymbol{x} = (\mathsf{Seed.pub}, 1)$, $\boldsymbol{y} = (\mathsf{Seed.Priv}(1), \boldsymbol{s})$ and $\boldsymbol{z} = (\mathsf{Seed.Priv}(2), 1)$. Note that $G_i(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}) = \gamma_i$.

   (e) For each $i \in [\eta]$, compute $\mathsf{PHFE}.sk[G_i, \gamma_i] \leftarrow \mathsf{PHFE.sfKG}(\mathsf{PHFE.MSK}, G_i, \gamma_i)$.

   (f) Output $sk_C = (\mathsf{PHFE}.sk[G_1, \gamma_1], \ldots, \mathsf{PHFE}.sk[G_\eta, \gamma_\eta])$

5. Adversary is handed over the function key. Finally, adversary outputs a guess $\rho \in \{0, 1\}$.

Lemma 13. *Let $s(\lambda)$ be a parameter greater than any polynomial in λ. If* PHFE *satisfies indistinguishability of semi-functional ciphertext property against adversaries of size $O(s)$, then for any adversary $\mathcal{A}$ of size $O(s)$,* $|\Pr[\mathcal{A}(\mathbf{Hybrid}_1) = 1] - \Pr[\mathcal{A}(\mathbf{Hybrid}_2) = 1]| \leq \mathsf{negl}(\lambda)$.

Proof. The only difference between the hybrids is that in $\mathbf{Hybrid}_1$, for each $i \in [\eta]$, the ciphertext $\mathsf{PHFE}.\overline{\mathsf{CT}}$ are generated using $\mathsf{PHFE.Enc}$, where as in $\mathbf{Hybrid}_2$, it is generated using $\mathsf{PHFE.sfEnc}$ algorithm. Note that in both cases $\gamma_i = G_i(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z})$ where $\mathsf{PHFE}.\overline{\mathsf{CT}}$ encrypts $(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z})$ in $\mathbf{Hybrid}_1$. Indistinguishability now follows from the security of PHFE scheme. $\square$

Hybrid$_3$: This hybrid is the same as the previous hybrid except that hardwirings γ_i is generated differently by invoking the security of $\mathcal{F}$-ΔRG.

1. Adversary outputs length 1^n, challenge message $\overline{m}$ along with messages $m_1, ..., m_\Gamma$. It also outputs a circuit $C \in \mathcal{C}_{\lambda,n,\epsilon}$. Let $C = (C_1, ..., C_\eta)$ where each sub-circuit C_i outputs i^{th} bit of C.
2. Challenger does setup. We recall the algorithm below.
   (a) Run $\mathsf{PHFE.Setup}(1^\lambda, 1^{n'}) \rightarrow \mathsf{PHFE.MSK}$. Here $n' = n \cdot \mathsf{poly}(\lambda)$.
   (b) Run $\mathsf{SHE.Setup}(1^\lambda, 1^n) \rightarrow \mathsf{SHE.PK}$.
   (c) Sample a secret vector $\boldsymbol{s}^* \in \chi^{\dim_1 \times 1}$. Set $\boldsymbol{s}^*[1,1] = 1$.
   (d) Compute $\mathsf{SHE.Encode}(\mathsf{SHE.PK}, \boldsymbol{s}^*, 0^n) \rightarrow \mathsf{SHE.CT}^*$. Run $\mathcal{F}$-$\Delta\mathsf{RG.SetupSeed}(1^\lambda, 1^n, \mathsf{Bound}_\Delta) \rightarrow \mathsf{Seed}^*$.
   (e) Output $\mathsf{MSK} = (\mathsf{PHFE.MSK}, \mathsf{SHE.PK}, \boldsymbol{s}^*, \mathsf{SHE.CT}^*, \mathsf{Seed}^*)$
3. Challenger encrypts all messages and hands over the ciphertexts to the adversary. These ciphertexts are denoted $\overline{\mathsf{CT}}, \mathsf{CT}[1], ..., \mathsf{CT}[\Gamma]$. We describe below, how challenge ciphertext is encrypted. Other ciphertexts are generated as in the previous hybrid.
   (a) Sample a secret $\boldsymbol{s} \leftarrow \chi^{\dim_1 \times 1}$. Set $\boldsymbol{s}[1,1] = 1$.
   (b) Compute $\mathsf{SHE.Encode}(\mathsf{SHE.PK}, \boldsymbol{s}, \overline{m}) \rightarrow \mathsf{SHE.}\overline{\mathsf{CT}}$.
   (c) Run $\mathcal{F}$-$\Delta\mathsf{RG.SetupSeed}(1^\lambda, 1^n, \mathsf{Bound}_\Delta) \rightarrow \mathsf{Seed}$. Parse $\mathsf{Seed} = (\mathsf{Seed.pub}, \mathsf{Seed.priv}(1), \mathsf{Seed.priv}(2))$.
   (d) Set $\boldsymbol{x} = (\mathsf{Seed.pub}, 1)$, $\boldsymbol{y} = (\mathsf{Seed.priv}(1), \boldsymbol{s})$ and $\boldsymbol{z} = (\mathsf{Seed.priv}(2), 1)$.
   (e) $\mathsf{PHFE.sfEnc}(\mathsf{MSK}, (\boldsymbol{x}, 1^{n'})) \rightarrow \mathsf{PHFE.}\overline{\mathsf{CT}}$.
   (f) Output $\overline{\mathsf{CT}} = (\overline{\mathsf{CT}}_1 = \mathsf{SHE.}\overline{\mathsf{CT}}, \overline{\mathsf{CT}}_2 = \mathsf{PHFE.}\overline{\mathsf{CT}})$
4. Function key for circuit C is generated as follows using the semi-functional key generation algorithm. Let $\theta_i = C_i(\overline{m})$ for $i \in [\eta]$.
   (a) Parse $C = (C_1, \ldots, C_\eta)$ where C_i is the circuit that outputs i^{th} bit of C. Here $\eta \leq n^{1+\epsilon}$.
   (b) [**Change**] For $i \in [\eta]$, compute $\mathsf{SHE.EvalCT}(\mathsf{SHE.PK}, C_i, \mathsf{SHE.}\overline{\mathsf{CT}}) \rightarrow \mathsf{SHE.}\overline{\mathsf{CT}}_{C_i}$.
   (c) [**Change**] Sample $\mathcal{F}$-$\Delta\mathsf{RG.SetupPoly}(1^\lambda, 1^n, \mathsf{Bound}_\Delta) \rightarrow (q_1, ..., q_{n^{1+\epsilon'}})$. Compute $(h_1, .., h_{n^{1+\epsilon'}}) \leftarrow \mathcal{H}(q_1, ..., q_{n^{1+\epsilon'}}, \mathsf{Seed})$
   (d) [**Change**] Compute $\gamma_i \leftarrow \overline{\mathsf{CT}}_{C_i} - \theta_i \cdot Q - h_i$. Set $\boldsymbol{x} = (\mathsf{Seed.pub}, 1)$, $\boldsymbol{y} = (\mathsf{Seed.Priv}(1), \boldsymbol{s})$ and $\boldsymbol{z} = (\mathsf{Seed.Priv}(2), 1)$. Note that $G_i(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}) \neq \gamma_i$.
   (e) For each $i \in [\eta]$, compute $\mathsf{PHFE.}sk[G_i, \gamma_i] \leftarrow \mathsf{PHFE.sfKG}(\mathsf{PHFE.MSK}, G_i, \gamma_i)$.
   (f) Output $sk_C = (\mathsf{PHFE.}sk[G_1, \gamma_1], \ldots, \mathsf{PHFE.}sk[G_\eta, \gamma_\eta])$
5. Adversary is handed over the function key. Finally, adversary outputs a guess $\rho \in \{0, 1\}$.

Lemma 14. *Let $s(\lambda)$ be a parameter greater than any polynomial in λ. If $\mathcal{F}$-ΔRG satisfies $(O(s), \mathsf{adv})-$security, then for any adversary $\mathcal{A}$ of size $O(s)$,* $|\Pr[\mathcal{A}(\mathbf{Hybrid}_2) = 1] - \Pr[\mathcal{A}(\mathbf{Hybrid}_3) = 1]| \leq \mathsf{adv}$.

Proof. The only difference between the hybrids is that for each $i \in [\eta]$, hardwirings γ_i are generated differently. In $\mathbf{Hybrid}_2$, it is generated as $\gamma_i = \boldsymbol{s}^T \cdot \boldsymbol{b}_{C_i} - q_i(\mathsf{Seed})$. In $\mathbf{Hybrid}_3$, it is generated as $\mathsf{SHE}.\overline{\mathsf{CT}}_{C_i} - C_i(\overline{m}) \cdot Q - h_i$. Due to structure of SHE encodings the following holds,

$$\mathsf{SHE}.\overline{\mathsf{CT}} = \boldsymbol{s}^T \cdot \boldsymbol{b}_{C_i} + C(\overline{m}) \cdot Q + \mathsf{error}$$

Where $|\mathsf{error}| \leq \mathsf{Bound}_{\Delta}$. Since $\mathsf{Seed.priv}(1)$ and $\mathsf{Seed.priv}(2)$ are hidden, hardwirings γ_i are indistinguishable due to security of $\mathcal{F}$-$\Delta\mathsf{RG}$. □

Hybrid$_4$: This hybrid is the same as the previous hybrid except that now $\mathsf{SHE}.\overline{\mathsf{CT}}$ is generated as an encryption of 0^n.

1. Adversary outputs length 1^n, challenge message $\overline{m}$ along with messages $m_1, ..., m_\Gamma$. It also outputs a circuit $C \in \mathcal{C}_{\lambda,n,\epsilon}$. Let $C = (C_1, ..., C_\eta)$ where each sub-circuit C_i outputs i^{th} bit of C.

2. Challenger does setup. We recall the algorithm below.

   (a) Run $\mathsf{PHFE.Setup}(1^\lambda, 1^{n'}) \to \mathsf{PHFE.MSK}$. Here $n' = n \cdot \mathsf{poly}(\lambda)$.

   (b) Run $\mathsf{SHE.Setup}(1^\lambda, 1^n) \to \mathsf{SHE.PK}$.

   (c) Sample a secret vector $\boldsymbol{s}^* \in \chi^{\dim_1 \times 1}$. Set $\boldsymbol{s}^*[1,1] = 1$.

   (d) Compute $\mathsf{SHE.Encode}(\mathsf{SHE.PK}, \boldsymbol{s}^*, 0^n) \to \mathsf{SHE.CT}^*$. Run $\mathcal{F}\text{-}\Delta\mathsf{RG.SetupSeed}(1^\lambda, 1^n, \mathsf{Bound}_\Delta) \to \mathsf{Seed}^*$.

   (e) Output $\mathsf{MSK} = (\mathsf{PHFE.MSK}, \mathsf{SHE.PK}, \boldsymbol{s}^*, \mathsf{SHE.CT}^*, \mathsf{Seed}^*)$

3. Challenger encrypts all messages using and hands over the ciphertexts to the adversary. These ciphertexts are denoted $\overline{\mathsf{CT}}, \mathsf{CT}[1], ..., \mathsf{CT}[\Gamma]$. We describe below, how challenge ciphertext is encrypted. Other ciphertexts are generated as in the previous hybrid.

   (a) Sample a secret $\boldsymbol{s} \leftarrow \chi^{\dim_1 \times 1}$. Set $\boldsymbol{s}[1,1] = 1$.

   (b) [**Change**] Compute $\mathsf{SHE.Encode}(\mathsf{SHE.PK}, \boldsymbol{s}, 0^n) \to \mathsf{SHE}.\overline{\mathsf{CT}}$.

   (c) Run $\mathcal{F}\text{-}\Delta\mathsf{RG.SetupSeed}(1^\lambda, 1^n, \mathsf{Bound}_\Delta) \to \mathsf{Seed}$. Parse $\mathsf{Seed} = (\mathsf{Seed.pub}, \mathsf{Seed.priv}(1), \mathsf{Seed.priv}(2))$.

   (d) Set $\boldsymbol{x} = (\mathsf{Seed.pub}, 1)$, $\boldsymbol{y} = (\mathsf{Seed.priv}(1), \boldsymbol{s})$ and $\boldsymbol{z} = (\mathsf{Seed.priv}(2), 1)$.

   (e) $\mathsf{PHFE.sfEnc}(\mathsf{MSK}, (\boldsymbol{x}, 1^{n'})) \to \mathsf{PHFE}.\overline{\mathsf{CT}}$.

   (f) Output $\overline{\mathsf{CT}} = (\overline{\mathsf{CT}}_1 = \mathsf{SHE}.\overline{\mathsf{CT}}, \overline{\mathsf{CT}}_2 = \mathsf{PHFE}.\overline{\mathsf{CT}})$

4. Function key for circuit C is generated as follows using the semi-functional key generation algorithm. Let $\theta_i = C_i(\overline{m})$ for $i \in [\eta]$.

   (a) Parse $C = (C_1, \ldots, C_\eta)$ where C_i is the circuit that outputs i^{th} bit of C. Here $\eta \leq n^{1+\epsilon}$.

   (b) For $i \in [\eta]$, compute $\mathsf{SHE.EvalCT}(\mathsf{SHE.PK}, C_i, \mathsf{SHE}.\overline{\mathsf{CT}}) \to \mathsf{SHE}.\overline{\mathsf{CT}}_{C_i}$.

   (c) Sample $\mathcal{F}\text{-}\Delta\mathsf{RG.SetupPoly}(1^\lambda, 1^n, \mathsf{Bound}_\Delta) \to (q_1, ..., q_{n^{1+\epsilon'}})$. Compute $(h_1, .., h_{n^{1+\epsilon'}}) \leftarrow \mathcal{H}(q_1, ..., q_{n^{1+\epsilon'}}, \mathsf{Seed})$

   (d) Compute $\gamma_i \leftarrow \overline{\mathsf{CT}}_{C_i} - \theta_i \cdot Q - h_i$. Set $\boldsymbol{x} = (\mathsf{Seed.pub}, 1)$, $\boldsymbol{y} = (\mathsf{Seed.Priv}(1), \boldsymbol{s})$ and $\boldsymbol{z} = (\mathsf{Seed.Priv}(2), 1)$. Note that $G_i(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}) \neq \gamma_i$.

   (e) For each $i \in [\eta]$, compute $\mathsf{PHFE}.sk[G_i, \gamma_i] \leftarrow \mathsf{PHFE.sfKG}(\mathsf{PHFE.MSK}, G_i, \gamma_i)$.

   (f) Output $sk_C = (\mathsf{PHFE}.sk[G_1, \gamma_1], \ldots, \mathsf{PHFE}.sk[G_\eta, \gamma_\eta])$

5. Adversary is handed over the function key. Finally, adversary outputs a guess $\rho \in \{0, 1\}$.

Lemma 15. *Let $s(\lambda)$ be a parameter greater than any polynomial in λ. If* SHE *is secure against adversaries of size $O(s)$, then for any adversary $\mathcal{A}$ of size $O(s)$,* $|\Pr[\mathcal{A}(\mathbf{Hybrid}_3) = 1] - \Pr[\mathcal{A}(\mathbf{Hybrid}_4) = 1]| \leq \mathsf{negl}$.

Proof. The only difference between $\textbf{Hybrid}_3$ and $\textbf{Hybrid}_4$ is the way $\mathsf{SHE}.\overline{\mathsf{CT}}$ is generated. In $\textbf{Hybrid}_3$ it is generated as an encoding of $\overline{m}$, while in $\textbf{Hybrid}_4$, it is generated as 0^n. Note that the randomness used to generate the encodings is nowhere else used. Thus indistinguishability holds due to security of SHE. $\square$

Hybrid$_5$: This hybrid is the same as the previous hybrid except that it has syntactic differences.

1. Adversary outputs length 1^n, challenge message $\overline{m}$ along with messages $m_1, ..., m_\Gamma$. It also outputs a circuit $C \in \mathcal{C}_{\lambda,n,\epsilon}$. Let $C = (C_1, ..., C_\eta)$ where each sub-circuit C_i outputs i^{th} bit of C.

2. Challenger does setup. We recall the algorithm below.

   (a) Run $\mathsf{PHFE.Setup}(1^\lambda, 1^{n'}) \rightarrow \mathsf{PHFE.MSK}$. Here $n' = n \cdot \mathsf{poly}(\lambda)$.

   (b) Run $\mathsf{SHE.Setup}(1^\lambda, 1^n) \rightarrow \mathsf{SHE.PK}$.

   (c) Sample a secret vector $\boldsymbol{s}^* \in \chi^{\dim_1 \times 1}$. Set $\boldsymbol{s}^*[1,1] = 1$.

   (d) Compute $\mathsf{SHE.Encode}(\mathsf{SHE.PK}, \boldsymbol{s}^*, 0^n) \rightarrow \mathsf{SHE.CT}^*$. Run $\mathcal{F}\text{-}\Delta\mathsf{RG.SetupSeed}(1^\lambda, 1^n, \mathsf{Bound}_\Delta) \rightarrow \mathsf{Seed}^*$.

   (e) Output $\mathsf{MSK} = (\mathsf{PHFE.MSK}, \mathsf{SHE.PK}, \boldsymbol{s}^*, \mathsf{SHE.CT}^*, \mathsf{Seed}^*)$

3. Challenger encrypts the challenge ciphertext using semi-functional encryption algorithm. These ciphertexts are denoted $\overline{\mathsf{CT}}, \mathsf{CT}[1], ..., \mathsf{CT}[\Gamma]$. We describe below, how challenge ciphertext is encrypted.

   (a) Set $\boldsymbol{x} = (\mathsf{Seed}^*.\mathsf{pub}, 1)$, $\boldsymbol{y} = (\mathsf{Seed}^*.\mathsf{priv}(1), \boldsymbol{s}^*)$ and $\boldsymbol{z} = (\mathsf{Seed}^*.\mathsf{priv}(2), 1)$.

   (b) $\mathsf{PHFE.sfEnc}(\mathsf{MSK}, (\boldsymbol{x}, 1^{n'})) \rightarrow \mathsf{PHFE}.\overline{\mathsf{CT}}$.

   (c) Output $\overline{\mathsf{CT}} = (\overline{\mathsf{CT}}_1 = \mathsf{SHE.CT}^*, \overline{\mathsf{CT}}_2 = \mathsf{PHFE}.\overline{\mathsf{CT}})$

4. Function key for circuit C is generated as follows using the semi-functional key generation algorithm. Let $\theta_i = C_i(\overline{m})$ for $i \in [\eta]$.

   (a) Parse $C = (C_1, \ldots, C_\eta)$ where C_i is the circuit that outputs i^{th} bit of C. Here $\eta \leq n^{1+\epsilon}$.

   (b) For $i \in [\eta]$, compute $\mathsf{SHE.EvalCT}(\mathsf{SHE.PK}, C_i, \mathsf{SHE}.\overline{\mathsf{CT}}) \rightarrow \mathsf{SHE}.\overline{\mathsf{CT}}_{C_i}$.

   (c) Sample $\mathcal{F}\text{-}\Delta\mathsf{RG.SetupPoly}(1^\lambda, 1^n, \mathsf{Bound}_\Delta) \rightarrow (q_1, ..., q_{n^{1+\epsilon'}})$. Compute $(h_1, .., h_{n^{1+\epsilon'}}) \leftarrow \mathcal{H}(q_1, ..., q_{n^{1+\epsilon'}}, \mathsf{Seed})$

   (d) Compute $\gamma_i \leftarrow \overline{\mathsf{CT}}_{C_i} - \theta_i \cdot Q - h_i$. Note that $G_i(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}) \neq \gamma_i$.

   (e) For each $i \in [\eta]$, compute $\mathsf{PHFE}.sk[G_i, \gamma_i] \leftarrow \mathsf{PHFE.sfKG}(\mathsf{PHFE.MSK}, G_i, \gamma_i)$.

   (f) Output $sk_C = (\mathsf{PHFE}.sk[G_1, \gamma_1], \ldots, \mathsf{PHFE}.sk[G_\eta, \gamma_\eta])$

5. Adversary is handed over the function key. Finally, adversary outputs a guess $\rho \in \{0, 1\}$.

Lemma 16. *For all adversaries* $\mathcal{A}$ $|\Pr[\mathcal{A}(\mathbf{Hybrid}_4) = 1] - \Pr[\mathcal{A}(\mathbf{Hybrid}_5) = 1]| = 0$.

Proof. These hybrids are identical. □