Indistinguishability Obfuscation from DDH-like Assumptions on Constant-Degree Graded Encodings[*]

Huijia Lin[†]
UC Santa Barbara

Vinod Vaikuntanathan[‡]
MIT

Abstract

All constructions of general purpose indistinguishability obfuscation (IO) rely on either meta-assumptions that encapsulate an exponential family of assumptions (e.g., Pass, Seth and Telang, CRYPTO 2014 and Lin, EUROCRYPT 2016), or polynomial families of assumptions on graded encoding schemes with a high polynomial degree/multilinearity (e.g., Gentry, Lewko, Sahai and Waters, FOCS 2014).

We present a new construction of IO, with a security reduction based on two assumptions: (a) a *DDH-like* assumption — called the *joint-SXDH assumption* — on *constant degree* graded encodings, and (b) the existence of polynomial-stretch pseudorandom generators (PRG) in NC^0. Our assumption on graded encodings is simple, has constant size, and does not require handling composite-order rings. This narrows the gap between the mathematical objects that exist (bilinear maps, from elliptic curve groups) and ones that suffice to construct general purpose indistinguishability obfuscation.

[*] A preliminary version of this paper appeared in the *57th IEEE Annual Symposium on Foundations of Computer Science*, FOCS 2016. This is the full version.

[†] `rachel.lin@cs.ucsb.edu`. Huijia Lin was partially supported by NSF grants CNS-1528178 and CNS-1514526.

[‡] `vinodv@csail.mit.edu`. Research supported in part by NSF Grants CNS-1350619 and CNS-1414119, Alfred P. Sloan Research Fellowship, Microsoft Faculty Fellowship, the NEC Corporation, a Steven and Renee Finn Career Development Chair from MIT. This work was also sponsored in part by the Defense Advanced Research Projects Agency (DARPA) and the U.S. Army Research Office under contracts W911NF-15-C-0226 and W911NF-15-C-0236. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of theDARPA and ARO.

Contents

1 Introduction

Indistinguishability obfuscation (IO) is a probabilistic polynomial-time algorithm $\mathcal{O}$ that takes as input a circuit C and outputs an (obfuscated) circuit $C' = \mathcal{O}(C)$ satisfying two properties:

(a) *functionality*: C and C' compute the same function; and

(b) *security*: for any two circuits C_1 and C_2 that compute the same function (and have the same size), $\mathcal{O}(C_1)$ and $\mathcal{O}(C_2)$ are computationally indistinguishable.

IO is a surprisingly powerful cryptographic notion. Defined first in the seminal work of Barak, Goldreich, Impagliazzo, Rudich, Sahai, Vadhan and Yang [BGI$^+$01a], it was largely unnoticed (with the singular exception of [GR07]) until the recent work of Garg, Gentry, Halevi, Raykova, Sahai and Waters [GGH$^+$13c] who demonstrated a *candidate* construction of indistinguishability obfuscation, and the work of Sahai and Waters [SW14] who showed that despite appearing somewhat useless to an untrained eye, IO has enormous power, so much so that it is virtually "crypto-complete". Starting from [SW14], we now know that IO gives us a treasure-chest of cryptographic constructions, solutions to a number of open problems (see, e.g., [SW14, GGH$^+$13c, GGHR14, GHRW14, CHN$^+$15, GP15, BPW16, BGJ$^+$16] and many more) and even has implications in complexity theory [BPR15, HY16].

In the tradition of theoretical cryptography, one *defines* a useful cryptographic object (a definition involves a notion of functionality and one of security), demonstrates a *construction* of this object using mathematics, and finally, *proves its security* from computational hardness assumptions (such as the hardness of factoring, discrete logarithms, or learning with errors). Garg et al. [GGH$^+$13c] showed a construction of IO using ideal lattices (which they abstracted into a framework called cryptographic multilinear maps [GGH13a]) but the construction came *as-is* with *no* security proof. There have since been a series of attempts at security proofs for IO under assumptions of varying complexity (which we will review in detail in the sequel). This state of affairs motivates one of the most important questions in cryptography today:

Does indistinguishability obfuscation exist, and under which cryptographic assumptions?

Let us cut to the chase: in this work, we show a construction of IO from the *joint SXDH* assumption on prime-order multilinear maps with *constant multi-linearity*. This narrows the gap between the mathematical objects that exist (bilinear maps) and ones that suffice for IO (O(1)-linear maps). We now describe what each of these terms mean through the lens of existing IO constructions.

Constructions and Proofs of Indistinguishability Obfuscation Over the last three years, there has been a great deal of work trying to construct IO schemes and prove their security. The mathematics underlying all IO constructions, broadly speaking, arises from the geometry of numbers (or the theory of integer lattices), but this has been abstracted out into the framework of *graded encoding schemes* (also called *cryptographic multilinear maps*) [BS02, GGH13a].[1]

In a nutshell, a graded encoding scheme for a ring $\mathcal{R}$ provides us with a (potentially exponentially large) collection of groups (written multiplicatively) of order q together with a relation $\mathsf{pairable}$ on them such that for any two groups G_i and G_j such that $\mathsf{pairable}(G_i, G_j) = G_k$, we have $g_i^{\alpha} \otimes g_j^{\beta} = g_k^{\alpha\beta}$ where $\otimes$ is a pairing function and $\alpha, \beta \in \mathbb{Z}_q$ are scalars. If $\mathsf{pairable}(G_i, G_j) = \perp$,

[1] In reality, we do not have any instantiations of graded encoding schemes, but rather only *noisy* graded encodings which turn out to suffice for functionality.

then we say that G_i and G_j are *not pairable*, and otherwise they are *pairable*. (In the literature, such graded encoding schemes are referred to as *clean* graded encodings, and can be generalized to *noisy* graded encodings. For most part of this introduction, we will use the interface of clean graded encodings; see Section 1.3 for a discussion on noisy graded encodings towards the end of the introduction.)

We can use these groups to compute multivariate polynomials in the exponent. That is, given a sequence of elements $g_j^{\alpha_j}$, compute $g_k^{p(\alpha_1,\ldots,\alpha_n)}$ where p is an n-variate polynomial and g_k is an element in the appropriate group, provided that p can be computed using a sequence of group operations in the same group and pairing operations over different groups as specified by **pairable**. The maximum degree of a multivariate polynomial that can be computed in the exponent is called the *multilinearity* of the collection. We call the number of groups in the collection the *universe size* (which could be constant, polynomial or exponential in the security parameter). The order of the group is q, and we will differentiate between prime-order and composite-order groups. In this language, the well-known bilinear maps have multilinearity 2 and universe size 2 (or 3 in the case of asymmetric pairing groups).

- **Proofs in Ideal Models:** Several works [GGH+13c, CV13, BR14, BGK+14a, AB15, Zim15] showed proofs of security for obfuscation in the so-called *ideal multilinear group* model. Roughly speaking, these models postulate that the only way an adversary can operate on group elements is through the legal group interface (namely, group operations between two elements in the same group and the pairing operation between two elements in pairable groups). Restricting the power of the adversary in such a way is unrealistic, and is underscored by the fact that in this model, one can actually get virtual black-box obfuscation (which by [BGI+01a] does not exist for general programs).

- **Concrete Assumptions on Graded Encodings (GES):** Pass, Seth and Telang [PST14] postulated an *uber-assumption* on multilinear maps which, roughly speaking, say that any attack on a collection of group elements can be translated into an attack in the *ideal multilinear group model*. Gentry, Lewko, Sahai and Waters [GLSW15], following the work of Gentry, Lewko and Waters [GLW14], took the first step in simplifying the assumption and came up with a construction under the *multilinear subgroup elimination* assumption on *composite-order groups*. Bitansky and Vaikuntanathan [BV15] and Ananth and Jain [AJ15], showed how to convert any functional encryption scheme into an IO scheme. Together with the FE construction of Garg, Gentry, Halevi and Zhandry [GGHZ16], this gives us an IO scheme based on similar assumptions on composite-order groups.

  The main deficiency of all these constructions is that they require graded encoding schemes with large multilinearity – either $\mathrm{poly}(|C|, n, \lambda)$ stand-alone, or at least $\mathsf{poly}(n, \lambda)$ after applying the bootstrapping theorem of Canetti, Lin, Tessaro and Vaikuntanathan [CLTV15].[2] In addition, they all either rely on a very complicated uber-assumption, or rely on composite-order graded encodings. Furthermore, the universe size in all these cases is at least $\mathsf{poly}(n, \lambda)$.

- **Towards Constant Multilinearity:** The closest in spirit to this work is the recent result of Lin [Lin16] who showed that constant-degree multilinear maps suffice for IO. (This is in spite of recent implausibility results [PS16, MMN16a, BV15, MMN+16b] showing that construction of IO in the *ideal constant-degree multilinear map* model, implies construction of IO

[2] There are other IO bootstrapping theorems in the literature, notably that of [GGH+13c]. However, they do not appear to help in reducing the multilinearity.

in the plain model; in other words, constant-degree multilinear map does not "help" black-box construction of IO. [Lin16] circumvents this by making non-black-box use of the graded encodings.) Unfortunately, her work has the following drawbacks: First, the concrete complexity assumption was a complicated über-assumption (borrowed from [PST14]); and secondly, her graded encoding collection has composite order and large, polynomial, universe size, namely $\mathsf{poly}(n, \lambda)$.

	Assumption	Multilinearity	Universe	Composite Order?
[BR14] [BGK$^+$14a, AGIS14] [AB15, Zim15]	ideal multilinear model	$\mathsf{poly}(n, \lambda)^{\ddagger}$	$\mathsf{poly}(n, \lambda)^{\ddagger}$	no
[PST14]	über-multilinear	$\mathsf{poly}(n, \lambda)^{\ddagger}$	$\mathsf{poly}(n, \lambda)^{\ddagger}$	no
[GLSW15]	multilinear subgroup elimination	$\mathsf{poly}(n, \lambda)^{\ddagger}$	$\mathsf{poly}(n, \lambda)^{\ddagger}$	yes
[BV15, AJ15, AJS15] +[GGHZ16]	similar to multilinear subgroup elimination	$\mathsf{poly}(n, \lambda)$	$\mathsf{poly}(n, \lambda)$	yes
[Lin16]	über-multilinear	$O(1)$	$\mathsf{poly}(\lambda)$	yes
This Work	Joint SXDH	$O(1)$	$O(1)$	no

Figure 1: A Summary of Known IO Constructions. ‡ denotes the fact that the complexity (multilinearity or universe size) was originally $\mathsf{poly}(|C|, n, \lambda)$ but can be brought down to $\mathsf{poly}(n, \lambda)$ using bootstrapping.

See Figure 1 for a summary.

In short, all assumptions used in the construction of IO are either a complicated uber-assumption that encodes the computation in the assumption itself, or an assumption on composite order graded encodings (GES) with polynomial universe size. The gap between bilinear maps and these objects is rather large. Even the construction of Lin [Lin16] requires a collection of $\mathsf{poly}(n, \lambda)$ groups with a complex interaction between them (through the uber-assumption), even though the multilinearity is constant.

With the aim of narrowing the gap between the mathematical objects that exist (bilinear maps) and ones that suffice for IO, we seek to:

> *Construct IO from a simple assumption on prime-order GES with $O(1)$ multilinearity and universe size.*

1.1 Our Results

Joint-SXDH Assumption on Graded Encodings. Our joint-SXDH assumption on graded encodings is a natural generalization of the standard symmetric external Diffie-Hellman (SXDH) assumption on (asymmetric) bilinear pairing groups. In short, SXDH states that the decisional Diffie-Hellman assumption holds in every source group. That is, let G_0 and G_1 be a pair of source groups, whose elements can be paired to produce elements in a target group G_T.

SXDH over bilinear maps: $\forall l \in \{0, 1\}$, $\left\{g_0,\ g_1,\ g_T,\ g_l^a,\ g_l^b,\ g_l^{ab}\right\} \approx \left\{g_0,\ g_1,\ g_T\ g_l^a,\ g_l^b,\ g_l^r\right\}$

where g_i is the generator for group G_i and a, b, r are random exponents. Note that SXDH possibly holds in asymmetric bilinear groups because elements in the same source group do not pair; otherwise, one can easily distinguish the above distributions by checking the equality $e(g_l^a, g_l^b) = e(g_l^{ab}, g_l)$.

The SXDH assumption naturally generalizes to graded encodings with a collection of groups $\{G_l\}_l$: It postulates that the distribution of g_l^a, g_l^b, g_l^{ab} in any group l should be indistinguishable to that of g_l^a, g_l^b, g_l^r, provided that elements in G_l cannot be paired with themselves. Here, because graded encodings allow for a richer computation structure, it is not only necessary that elements cannot be paired directly (*i.e.* $\mathbf{pairable}(G_l, G_l) = \perp$), but they also do not pair "indirectly", via a sequence of pairings with elements in other groups. This leads to the notion of the closure of the **pairable** function, denoted as **pairable***, which roughly speaking indicates whether two groups G_{l_1}, G_{l_2} can ever be paired via any sequence of pairing. More precisely, $\mathbf{pairable}^*(G_{l_1}, G_{l_2}) = 1$ if there are groups G_{l_3} and G_{l_4} such that $\mathbf{pairable}^*(G_{l_1}, G_{l_3}) = 1$, $\mathbf{pairable}^*(G_{l_2}, G_{l_4}) = 1$, and $\mathbf{pairable}(G_{l_3}, G_{l_4}) \neq \perp$; otherwise, $\mathbf{pairable}^*(G_{l_1}, G_{l_2}) = 0$. Then,

SXDH over graded encodings: $\forall l$ s.t. $\mathbf{pairable}^*(G_l, G_l) = 0$,

$$\Big\{\{g_i\},\ g_l^a,\ g_l^b,\ g_l^{ab}\Big\} \approx \Big\{\{g_i\},\ g_l^a,\ g_l^b,\ g_l^r\Big\} \text{ where } \{g_i\} \text{ is the set of generators of all groups.}$$

Finally, joint-SXDH further generalizes SXDH. It considers the joint distribution of elements $(g_l^a, g_l^b, g_l^{ab})_{l \in S}$ in a set S of groups, with the *same* exponents a, b, ab. By the same argument above, if any two groups G_{l_1}, G_{l_2} in the set are pairable, directly or indirectly, one can distinguish the joint distribution from the distribution of $(g_l^a, g_l^b, g_l^r)_{l \in S}$ with random exponents a, b, r. Otherwise, in the same spirit as SXDH, the distributions are possibly indistinguishable — this is exactly our joint-SXDH assumption.

Joint-SXDH over Graded Encodings:

$$\forall \text{ Set } S \text{ satisfies } \forall\, l_1, l_2 \in S,\ \mathbf{pairable}^*(G_{l_1}, G_{l_2}) = 0,$$
$$\Big\{\{g_i\},\ \Big\{g_l^a,\ g_l^b,\ g_l^{ab}\Big\}_{l \in S}\Big\} \approx \Big\{\{g_i\},\ \Big\{g_l^a,\ g_l^b,\ g_l^r\Big\}_{l \in S}\Big\}$$

Furthermore, the subexponential joint-SXDH assumption requires the above distributions to have subexponentially small distinguishing gap to all polynomial time distinguishers. (See Section 4 for the precise statement of the assumption.)

IO from joint-SXDH on Constant-Degree Graded Encodings. We are now ready to state our main theorem.

Theorem 1 (Main Theorem, Informal)**.** *Assume the existence of a sub-exponentially secure* $n^{1+\alpha}$-stretch *pseudorandom generator (PRG) in* NC^0 *for any positive constant* $\alpha > 0$*. Then, IO for* $\mathsf{P/poly}$ *is implied by the sub-exponential joint-SXDH assumption on a constant-degree graded encoding scheme, with prime order and constant universe size.*

Tree-GES: The graded encoding scheme that our main theorem relies on has a specific pairable function that allows computing arithmatic circuits of layers of additions and multiplications; we refer to such a scheme a *tree-structured* graded encoding scheme, or *tree-GES* for short. Roughly speaking, a tree-GES consists of a set of groups arranged at the nodes of a 4-ary tree, together with a $\mathsf{pairable}$ function defined in the following way. If $G_{l_0}, G_{l_1}, G_{l_2}$ and G_{l_3} are the (groups in the) four children of a node G_l, then $\mathbf{pairable}(G_{l_0}, G_{l_1}) = \mathbf{pairable}(G_{l_2}, G_{l_3}) = G_l$, whereas all other combinations are not pairable (*e.g.*, $\mathbf{pairable}(G_{l_0}, G_{l_2}) = \perp$, and so on). Naturally, given $g_{l_i}^{a_i}$ for $i \in \{0, 1, 2, 3\}$, one can compute $g_l^{a_0 a_1 + a_2 a_3}$ (one layer of multiplications followed by additions), and cannot compute (via the honest interface) any quadratic polynomial containing monomials $a_i a_j$ for $i = j$ or $i \leq 1 < j$.

One of the nice features of this general interface is that it captures directly the computation structure we need from GES, and it can be instantiated from both set-based and graph-based (prime- as well as composite-order) multilinear maps, which gives it a great deal of flexibility. For example, while none of the previous abstract constructions [PST14, GLSW15, GGHZ16, Lin16] can be instantiated based on the graph-based multilinear maps of [GGH15], our IO scheme will admit such an instantiation.

In the language of tree-GES, our IO construction relies on the joint-SXDH assumption on tree-GES with a tree of constant depth. Carrying this over to the set-multilinear map setting, this translates to constant multilinearity and constant universe size. Our main theorem also relies on a sub-exponentially secure polynomial stretch PRG. See Section 2.3 for a discussion on this assumption.

Our Approach via Bootstrapping FE for NC^0 to IO: Our approach towards constructing IO from constant-depth tree-GES is through a bootstrapping step showing that assuming the existence of polynomial-stretch pseudorandom generators (PPRG) in NC^0, a (collusion-resistant) Functional Encryption (FE) scheme for NC^0 implies indistinguishability obfuscation for all of P; the FE scheme for NC^0 needs to have linear efficiency, in the sense that, encryption time depends linearly in the message length. Then, we use constant-depth tree-GES to implement a such FE scheme.

We invite the reader to pause for a moment and note that while common sense would dictate that obfuscation is more powerful than functional encryption, obfuscation for NC^0 circuits is completely trivial (namely, for each output bit of the circuit, publish the truth table of the circuit that generates it, and the constant number of input bits that the output bit depends on) and yet, FE for NC^0 is far from trivial. Indeed, the theorem below says that FE for NC^0 is powerful enough to imply indistinguishability obfuscation for all of P.

Theorem 2 (Informal, following [BV15, AJS15, Lin16])**.** *Assume the existence of a sub-exponentially secure* $n^{1+\alpha}$-stretch *pseudorandom generator in* NC^0 *for any positive constant* $\alpha > 0$*. Then, IO for* P/poly *is implied by FE schemes for* NC^0 *with encryption time linear in the input length.*

The theorem follows from combining observations in previous works [BV15, AJS15, Lin16], in particular, combining randomized encoding in NC^0 and PRG in NC^0 to transform any NC^1 computation into an NC^0 computation. See Section 3 for more details.

Constructing FE for NC^0 from Constant-Degree GES: Our main technical contribution is the construction of an FE scheme for NC^0 with linear efficiency under the joint-SXDH assumption on tree-GES for constant-depth trees.

Theorem 3 (Informal)**.** *Assuming the existence of a prime-order tree-GES for depth-$O(1)$ trees with the joint-SXDH assumption, there is a (collusion-resistant) FE scheme for all* NC^0 *circuits, with encryption time linear in message length.*

Thus, put together, we get IO for P/poly, assuming joint-SXDH and the existence of polynomial-stretch PRGs in NC^0. We now proceed to describe the techniques behind the FE construction.

1.2 Technical Overview

Since Theorem 2 follows from observations in previous works, we focus on the question:

> *How does one construct a collusion-resistant functional encryption scheme for* NC^0*, with linear efficiency?*

The State-of-the-Art of Collusion Resistant FE. In the literature, the only constructions of collusion-resistant FE from standard assumptions are for computing inner products, referred to as Inner Product Encryption (IPE). Roughly speaking, a (public key or secret key) IPE scheme allows to encode vectors $\mathbf{y}$ and $\mathbf{x}$ in a ring $\mathcal{R}$, in a function key $\mathsf{sk}_{\mathbf{y}}$ and ciphertext $\mathsf{ct}_{\mathbf{x}}$ respectively, and decryption computes the inner product $\langle \mathbf{y}, \mathbf{x} \rangle \in \mathcal{R}$. Abdalla, Bourse, De Caro and Pointcheval (ABCP) [ABDP15, ABCP16] came up with a public key IPE scheme based on one of a variety of assumptions, such as the decisional Diffie-Hellman assumption, the Paillier assumption and the learning with errors assumption. Following that, Bishop, Jain and Kowalczyk [BJK15] (BJK) constructed a secret-key scheme based on the SXDH assumption over asymmetric bilinear maps; their scheme achieves the stronger security property of *weak function-hiding* (explained below). Both the ABCP and BJK schemes do not compute the inner product $\langle \mathbf{y}, \mathbf{x} \rangle$ in the clear, but computes it in the exponent $g^{\langle \mathbf{y}, \mathbf{x} \rangle}$; the BJK scheme in fact computes the inner product $\theta \langle \mathbf{y}, \mathbf{x} \rangle$ masked by a scalar θ in the exponent; see more discussion later.[3]

Given IPE schemes, it is trivial to implement FE for quadratic polynomials: Simply write a quadratic function f as a linear function over quadratic monomials $f(x) = \Sigma_{i,j} c_{i,j} x_i x_j = \langle \mathbf{c}, \mathbf{x} \otimes \mathbf{x} \rangle$, where $\otimes$ is tensor product. Then, use an IPE scheme to generate a ciphertext $\mathsf{ct}_{\mathbf{x} \otimes \mathbf{x}}$ and a function key $\mathsf{sk}_{\mathbf{c}}$, which produce $f(x)$. However, the ciphertext size scales *quadratically* in $n = |\mathbf{x}|$. This idea easily generalizes and gives a FE scheme for NC^0 with encryption time n^d, where d is the degree of the computation. Unfortunately, improving these FE schemes to have encryption time linear in the input length under standard assumptions (*e.g.* bilinear maps) has proved elusive.

Coming from the "other side", Garg, Gentry, Halevi and Zhandry (GGHZ) [GGHZ16] proposed a general-purpose FE scheme from polynomial-degree GES (with composite-order). A natural next attempt would be to try to specialize their FE scheme to NC^0 circuits, in the hope that we can pull off the construction using only constant-degree GES. This wishful thinking runs into trouble. Very roughly speaking, the GGHZ construction works with a universal branching program and requires GES with multilinearity that is $O(\ell)$ where ℓ is the length of the branching program. Now, even if we only want to handle NC^0 circuits that take n bits of input, converting them into a universal branching program results in a program of size $\Omega(n)$.

One might hope to get around this problem by representing the NC^0 circuit for each output bit as a constant-sized branching program; however, in this case, it is not clear how each function key can "index" the right input bits in the n-bit input to compute on. This "indexing problem" prevents us from tweaking the construction to support NC^0 circuits with constant multilinearity.

In this work, we come up with a completely different FE construction that not only gives us constant multilinearity, but also relies on GES with prime order and constant universe size, and the simple joint-SXDH assumption.

[3] There has been a long line of work on "inner product *testing* functional encryption" or "zero-testing IPE" (see, e.g., [KSW08, LOS+10] and many others) which is different from what we need here. In IPE, we require that function key $\mathsf{sk}_{\mathbf{y}}$ and ciphertext $\mathsf{ct}_{\mathbf{x}}$ produce the inner product in $\mathcal{R}$ in the exponent. In contrast, in zero-testing IPE, one can only compute whether $\langle \mathbf{x}, \mathbf{y} \rangle \overset{?}{=} 0$ in $\mathcal{R}$. In particular, they do not produce the inner product in the exponent, in a way that allows for further computation. Hence, they are insufficient for our construction of FE for NC^0.

Overview Towards constructing FE for NC^0 with linear efficiency from constant-degree GES, our first observation is that functionality is easy to achieve, since NC^0 circuits f can be represented as constant-degree arithmetic circuits or polynomials, and constant-degree GES supports evaluating constant-degree polynomials in the exponent. Once the output $y = f(x)$ is computed in the exponent g^y, it can be extracted as it is Boolean. Thus, the main challenge lies in achieving security, ensuring that the input and all intermediate computation results are hidden.

To hide the input and computation, the first tool that comes in mind is Randomized Encodings (RE). An RE scheme allows one to use randomness to encode a function f and an input x, $\Pi \overset{\$}{\leftarrow} \mathbf{RE}(f, x; r)$, so that:

1. The encoding algorithm is simple: Each element of Π is of the form $x_{\pi(i)} \cdot p_i(r) + q_i(r)$, where π is an input-mapping function, and p_i, q_i are polynomial functions of the randomness r. That is, a linear function of a single input bit (and a polynomial function of the randomness r);

2. The encoding Π reveals the output $z = f(x)$ of the computation and nothing more.

The key difference of RE from FE in that RE cannot be reused, whereas the ciphertexts (respectively, function keys) of a FE scheme can be reused across an unbounded number of function keys (respectively, ciphertexts).

The First Idea and Challenges. Our first and foremost idea is *to combine the re-usability of IPE schemes with the capability of hiding inputs and computations of RE schemes*, by designing techniques to use an IPE scheme to compute randomized encodings. More specifically,

Outline of Our FE scheme

- *Key Generation:* To create a key sk_f for $f \in \mathsf{NC}^0$, first encode f in a set of vectors $\{\mathbf{u}_k\}$, and then publish IPE function keys $\mathsf{sk}_f = \{\mathsf{sk}^k_{\mathbf{u}_k}\}$ for these vectors, using independently sampled master keys.
- *Encryption:* Similarly, to encrypt an input $x \in \{0,1\}^n$, encode x in a set of vectors $\{\mathbf{v}_k\}$, and encrypt them in IPE ciphertexts $\mathsf{ct}_x = \{\mathsf{ct}^k_{\mathbf{v}_k}\}$ with corresponding master keys.

The vectors $\mathbf{u}_k$ and $\mathbf{v}_k$ are set up in a way so that their inner products $\langle \mathbf{u}_k, \mathbf{v}_k \rangle = \Pi_k$ produce exactly the k^{th} element Π_k in the randomized encoding for f, x. Thus, the IPE scheme ensures that evaluating $\mathsf{sk}^k_{\mathbf{u}_k}$ and $\mathsf{ct}^k_{\mathbf{v}_k}$ produces Π_k in the exponent $g_{l_k}^{\Pi_k}$ in some group G_{l_k}.

In the literature, the idea of using FE for a weak function class, to compute the randomized encodings of a stronger function class has been used in bootstrapping FE for NC^1 to FE for $\mathsf{P/poly}$ [ABSV15]. In some sense, our construction can be viewed as bootstrapping FE for inner products to FE for NC^0. Here, unique challenges arise due to the fact that we can only compute inner products.

- **Challenge 1:** *How to generate randomized encodings using only inner products?*

  To do so, we crucially rely on *affine* randomized encodings, where each element Π_k in the encoding of a computation f, x depends *linearly* on each bit in x. The idea is then to represent each element Π_k as the inner product between some coefficient vectors (depending on f) and input vectors (depending on x), so that, Π_k can be computed using IPE.

  In particular, we will use the arithmetic randomized encodings for NC^1 of Applebaum, Ishai and Kushilevitz [AIK14], which is affine and has many other useful properties.

- **Challenge 2:** *How to generate the randomness for randomized encodings?*

  Consider a scenario where our FE for NC^0 scheme is used to publish m function keys $\{\mathsf{sk}_{f_j}\}$ and m ciphertexts $\{\mathsf{ct}_{x_i}\}$. Every pair of key and ciphertext $\mathsf{sk}_{f_j}, \mathsf{ct}_{x_i}$ computes a randomized encoding $\Pi_{j,i} \in \mathbf{RE}(f_j, x_i)$ (in the exponent), which requires using fresh (at least, "computationally fresh") randomness r_{ji}. Note that we need in total m^2 "pieces" of randomness, but has only m function keys and ciphertexts — r_{ji}'s can only be pseudorandom.

  In the case of bootstrapping FE for NC^1 to FE for $\mathsf{P/poly}$, this problem is easily resolved using Pseudo Random Functions (PRFs): One can simply encrypt a PRF seed s together with the input x, and the function keys evaluate the PRF on s to expand pseudorandomness for computing the randomized encoding. However, in our case, the functionality of IPE does not support PRF evaluation. Not even extremely strong local PRGs can help here, since any quadratic-stretch PRGs (from $O(m)$ bits to m^2 bits) has at least degree 3.

  We resolve this problem by, instead, relying on built-in pseudorandomness assumption, namely *joint-SXDH*, in GES. Indeed, the SXDH assumption w.r.t. a group G_l guarantees that given a set of $2m$ random elements in the exponent $\{g_l^{s_j}, g_l^{t_i}\}_{j,l\in[m]}$, the set of m^2 products in the exponent $\{g_l^{s_j t_i}\}$ are indistinguishable to elements $\{g_l^{r_{ji}}\}$ with truly random exponents. The r_{ji}'s in the exponent will be the randomness for generating RE. They can be computed from short, length-$2m$, seeds $\{s_j, t_i\}$ in degree 2; just that they must reside in the exponent.

Before going into details on how to resolve the above two challenges, we first complete the construction outline.

Achieving Functionality. Given that we can use IPE to compute randomized encodings in the exponent $\{g_{l_k}^{\Pi_k}\}$, it is tempting to think that one can simply extract Π if the encodings are binary, and compute the output y in the clear. If this could be done, we would have obtained FE for NC^0 from only bilinear maps, and thus IO from bilinear maps. The catch is that we have to use arithmetic randomized encodings (where the elements that compose the randomized encoding, namely Π_k, live in a large field) and cannot use binary randomized encodings. Roughly speaking, the culprit is our solution to Challenge 2. As mentioned above, the elements of the randomized encodings are generated pseudo-randomly. The randomness used for generating the randomized encoding lives in the exponent ring $\mathcal{R}$, and can only produce pseudorandomness *in the exponent* through the joint-SXDH assumption. In turn, as a result of this, we need to use *arithmetic* randomized encodings (in particular, [AIK14]), which cannot be extracted from the exponent, unless discrete logarithm is easy. In fact, extracting these arithmetic randomized encodings would lead to attacks on the joint-SXDH assumption.

Therefore, we rely on constant-degree GES to achieve *functionality*, by evaluating the arithmetic randomized encoding Π in the exponent. Evaluation produces the output y in the exponent, which can be extracted since it is binary. More specifically, recall that we use a tree-structured GES that supports evaluating arithmetic circuits with a constant number of layers of multiplications and additions, in particular, the RE evaluation circuit for NC^0 computation. We will carefully instantiate different IPE instances $(\mathsf{sk}^k_{\mathbf{u}_k}, \mathsf{ct}^k_{\mathbf{v}_k})$ using different groups in the tree-GES so that IPE evaluation produces the randomized encoding $g_{l_k}^{\Pi_k}$ in appropriate groups l_k, on which RE evaluation can be performed.

With these insights, let's now circle back and resolve challenges 1 and 2.

Resolving Challenge 1. Our key tool is the *affine* AIK arithmetic randomized encodings (ARE) [AIK14], which depends linearly in the input. More specifically, the AIK arithmetic randomized encoding for an (arithmetic) NC^1 function f and input $\mathbf{x} \in \mathcal{R}^n$ is computed using a set of $m = \text{poly}(n)$ *fixed* linear functions L_k as follows:

$$\left\{ \Pi_k = L_k(\mathbf{x}, \mathbf{r}) = p_k(\mathbf{r}) x_{\pi(k)} + q_k(\mathbf{r});\ \pi : [m] \to [n] \right\}$$

Here, each randomized encoding element Π_k depends on a single input bit $x_{\pi(k)}$, determined by an input mapping function π. The coefficients of the linear functions $p_k(\mathbf{r})$ and $q_k(\mathbf{r})$ are *fixed* multi-linear polynomials that act on the randomness $\mathbf{r}$. The only part that depends on the function f is the input mapping function.

To use IPE to compute such arithmetic randomized encodings, the idea is that the FE key generation algorithm encodes the coefficients $p_k(\mathbf{r}), q_k(\mathbf{r})$ and the input mapping function π, and the FE encryptor encrypts $\mathbf{x}$; they together compute the affine functions $L_k(\mathbf{x}, \mathbf{r})$. More precisely,

Our FE scheme, version 1

- *Key Generation:* To generate a key sk_f for f, sample randomness $\mathbf{r}$, and publish IPE keys $\mathsf{sk}_f = \{\mathsf{sk}^k_{\mathbf{u}_k}\}$ for vectors $\mathbf{u}_k = (p_k(\mathbf{r}) || q_k(\mathbf{r})) \otimes \mathbf{e}_{\pi(k)}$ (using independently sampled master keys).
- *Encryption:* To encrypt x, publish IPE ciphertexts $\mathsf{ct}_x = \{\mathsf{ct}^k_{\mathbf{v}_k}\}$ for vectors $\mathbf{v}_k = (x_i || 1)_{i \in [n]}$ (using corresponding master keys).

It is easy to verify that $\langle \mathbf{u}_k, \mathbf{v}_k \rangle = \Pi_k$. In other words, we achieve the goal of computing AIK arithmetic randomized encodings using IPE. The above scheme is, however, insecure: In particular, the randomness $\mathbf{r}$ for generating randomized encodings is hardcoded in the secret key, meaning that the randomized encodings for the same function f and different inputs $x_1, x_2, \cdots$ share the same randomness, which renders them insecure. This leads us back to resolving the second challenge of generating the randomness for randomized encodings.

Resolving Challenge 2. We rely on joint-SXDH to generate randomness. What we need is that for every pair of key and ciphertext, the randomized encoding should use fresh (at least, "computationally fresh") randomness. We accomplish this by (re-)writing the affine functions as

$$\left\{ \Pi_k = L_k(\mathbf{x}, \mathbf{r}, \mathbf{s}) = p_k(\mathbf{rs}) x_{\pi(k)} + q_k(\mathbf{rs}) \right\}$$

The randomness in use is the coordinate-wise multiplication of $\mathbf{r}$ and $\mathbf{s}$. We will put one multiplicative "share" $\mathbf{r}$ in the key, and the other $\mathbf{s}$ in the ciphertext. To see how to compute such a thing, note that if

$$p_k(\mathbf{r}) = \sum_j M_{kj}(\mathbf{r}) \qquad \text{and} \qquad q_k(\mathbf{r}) = \sum_j M'_{kj}(\mathbf{r})$$

where the M_{kj} and M'_{kj} are monomials, then

$$p_k(\mathbf{rs}) = \sum_j M_{kj}(\mathbf{r}) M_{kj}(\mathbf{s}) \qquad \text{and} \qquad q_k(\mathbf{rs}) = \sum_j M'_{kj}(\mathbf{r}) M'_{kj}(\mathbf{s})$$

We modify our FE scheme as below:

Our FE scheme, version 2

- *Key Generation:* To generate sk_f for f, sample $\mathbf{r}$ and publish IPE keys $\mathsf{sk}_f = \{\mathsf{sk}^k_{\mathbf{u}_k}\}$ for vectors

$$\mathbf{u}_k = \Big(M_{kj}(\mathbf{r}), M'_{kj}(\mathbf{r}) \Big)_j \otimes \mathbf{e}_{\pi(k)} \tag{1}$$

- *Encryption:* To encrypt $x \in \{0,1\}^n$, sample $\mathbf{s}$ and publish IPE ciphertexts $\mathsf{ct}_x = \{\mathsf{ct}^k_{\mathbf{v}_k}\}$ for vectors

$$\mathbf{v}_k = \Big(M_{kj}(\mathbf{s})x_1, M'_{kj}(\mathbf{s}) \Big)_j \Big|\Big| \cdots \Big(M_{kj}(\mathbf{s})x_i, M'_{kj}(\mathbf{s}) \Big)_j \cdots \Big|\Big| \Big(M_{kj}(\mathbf{s})x_n, M'_{kj}(\mathbf{s}) \Big)_j \tag{2}$$

Now, the inner product $\langle \mathbf{u}_k, \mathbf{v}_k \rangle$ is the randomized encoding element Π_k generated using randomness $\mathbf{r}\mathbf{s}$. Moreover, the AIK randomized encoding has the property that the total number of monomials $M_{k,j}, M'_{k,j}$ is bounded by $2^{O(d)}$, where d is the depth of the arithmetic circuit computing f. Thus for NC^0 computations, the vectors $\mathbf{u}_k, \mathbf{v}_k$ are of length $O(n)$, linear in the input length, giving us the desired linear efficiency property.

Overview of Security Proof FE security states that the ciphertexts ct_{x^0} and ct_{x^1} of inputs x^0 and x^1 should be indistinguishable, even in the presence of keys $\{\mathsf{sk}_{f_j}\}$ as long as they satisfy that $f_j(x^0) = f_j(x^1)$ for every j. We want to reduce this indistinguishability to the security of randomized encodings — that encodings $\{\Pi^0_j\}$ for f_j, x^0, and encodings $\{\Pi^1_j\}$ for f_j, x^1 are indistinguishable. But, before invoking RE security, we must first argue that the input x^b is hidden, and the randomness $\{\mathbf{r}_j\mathbf{s}\}$ for generating Π^b_j is jointly pseudo-random. This is certainly not the case w.r.t. honestly generated keys and ciphertexts: First x^b is embeded in the ciphertext, and second it seems impossible to argue that the products $\{\mathbf{r}_j\mathbf{s}\}$ are pseudorandom, when $\mathbf{r}_j$ and $\mathbf{s}$ reside respectively in IPE keys and ciphertexts that can be paired together.

To resolve this conundrum, our idea is leveraging the function hiding property of a secret-key IPE scheme, in order to "move" the input x^b and $\mathbf{s}$ into the function keys in security hybrids. Let us explain. The function hiding property guarantees that IPE keys and ciphertexts for two sets of vectors $\{\mathbf{a}_i, \mathbf{b}_i\}$ and $\{\mathbf{a}'_i, \mathbf{b}'_i\}$ are indistinguishable if they produce identical inner products $\langle \mathbf{a}_i, \mathbf{b}_j \rangle = \langle \mathbf{a}'_i, \mathbf{b}'_j \rangle$. We now further modify the FE scheme to encode vectors with some trailing zeros.

Our FE scheme, version 3

- *Key Generation:* To generate sk_f for f, sample $\mathbf{r}$ and publish IPE keys $\mathsf{sk}_f = \{\mathsf{sk}^k_{\mathbf{u}_k}\}$ for vectors $\mathbf{u}'_k = \mathbf{u}_k \;||\; 0$, where $\mathbf{u}_k$ is described in Equation (1).
- *Encryption:* To encrypt $x \in \{0,1\}^n$, sample $\mathbf{s}$ and publish IPE ciphertexts $\mathsf{ct}_x = \{\mathsf{ct}^k_{\mathbf{v}_k}\}$ for vectors $\mathbf{v}'_k = \mathbf{v}_k \;||\; 0$, where $\mathbf{v}_k$ is described in Equation (2).

The trailing zeros do not affect the functionality. But, in the security proof, they provide the crucial "space" for hardwiring the randomized encoding Π^b in the function key sk_f, without computing it. More specifically, in the proof, we move to a hybrid, encoding vectors of form $\mathbf{u}''_k = \mathbf{u}_k || \Pi_k$, and $\mathbf{v}''_k = \mathbf{0}||1$. Since $\langle \mathbf{u}''_k, \mathbf{v}''_k \rangle = \langle \mathbf{u}'_k, \mathbf{v}'_k \rangle = \langle \mathbf{u}_k, \mathbf{v}_k \rangle$, by function hiding of IPE, this hybrid is indistinguishable to the honest execution. Notice that in this hybrid, the ciphertext contains no information of the input x^b, and the key for a function f_j has the corresponding randomized encoding Π^b_j (for f_j, x^b) hardwired in. Furthermore, the fact that the randomness share $\mathbf{s}$ disappears eventually allows us to argue that $\{\mathbf{r}_j\mathbf{s}\}$ used for generating $\{\Pi_j^b\}$ are pseudorandom. Then, we can finally invoke the RE security, that $\{\Pi_j^0\}$ and $\{\Pi_j^1\}$ are indistinguishable, to argue that FE security holds.

The proof strategy of using computational assumptions to reduce the FE security to RE security resembles that of many FE and IO schemes in the literature in a high level (e.g., [GLSW15]), but the details of how we make this approach go through are very different.

Additional Challenges Additional challenges must be addressed in order to make the above security proof overview go through. First, applying joint-SXDH to argue the pseudorandomness of $\{\mathbf{rs}\}$ is tricky. This is because we (have to) compute elements Π_k in a randomized encoding Π in different groups $g_{l_k}^{\Pi_k}$ in order to further evaluate Π in the exponent. But, the collection of elements $\{\Pi_k\}$ are correlated through shared randomness $\mathbf{rs}$. An attacker can potentially leverage this correlation, and through computation over different groups, distinguish Π_k generated from $\mathbf{rs}$ and that from true randomness. It turns out that the structure of the tree-GES, together with the join-SXDH assumption is exactly what we need to prevent all attacks that arise out of such correlations.

Second, the above proof relies on a secret key IPE that is fully function hiding. Looking back into the literature, we see that the BJK secret key IPE [BJK15] is only weak function hiding. A followup work [DDM16] constructed fully function hiding secret-key IPE. In this work, we show how to generically transform any weak function hiding IPE to full function hiding IPE; our transformation is black-box, extremely simple and of independent interest.

A further issue is that these function hiding secret-key IPE schemes do not produce the inner product in the exponent directly $g^{\langle \mathbf{u},\mathbf{v}\rangle}$, but produce the inner product masked by a scalar $(g^{\langle \mathbf{u},\mathbf{v}\rangle\theta}, g^{\theta})$, where the scalar θ is determined by the randomness used in key generation and encryption. This creates the problem that randomized encoding elements computed using different IPE instances are masked by distinct scalars $(g_{l_k}^{\Pi_k\theta_k}, g^{\theta_k})$, preventing RE evaluation in the exponent. To resolve this, in our FE scheme, the secret-key IPE instances $\{\mathsf{sk}^k_{\mathbf{u}_k}, \mathsf{ct}^k_{\mathbf{v}_k}\}$ are generated using different master secret keys, but the *same randomness*. Thus, they produce randomized encoding elements masked by the same scalar $(g_{l_k}^{\Pi_k\theta}, g^{\theta})$ and then evaluation can be done as before. As a result, we need function hiding secret-key IPE that allows sharing randomness among instances generated using different master secret keys. It turns out that our function hiding secret-key IPE derived from the BJK scheme has this property.

Stitching all pieces together, we obtain a secret-key FE for NC^0 with linear efficiency, from constant-depth tree-GES.

Slotted IPE and Public-key FE. We have to go one step further to construct a public-key FE for NC^0 with linear efficiency. The natural first idea is to use, instead of a secret-key function-hiding IPE scheme, a public-key function-hiding IPE. However, a moment's reflection tells us that such an object cannot possibly exist: that is, the properties of function-hiding and being public-key do not play well with each other.

Our solution to this issue is to construct a "hybrid" encryption scheme that we call a *slotted inner product encryption* (or slotted IPE) scheme. Roughly speaking, a slotted IPE scheme generates keys for vectors $\mathbf{y}_{pub}||\mathbf{y}_{priv}$, encrypts vectors $\mathbf{x}_{pub}||\mathbf{x}_{priv}$, and given the functional secret key, computes an inner product between them. Crucially, a slotted IPE scheme has the following seemingly contradictory properties:

- *Public Key for the first Slot:* Anyone can encrypt vectors of the form $\mathbf{x}_{pub}||\mathbf{0}$. (However, it is computationally hard to encrypt any vector with a non-zero component in the second slot.) The usual notion of semantic security holds, that is, encryption of $\mathbf{x}_{pub}||\mathbf{x}_{priv}$ and $\mathbf{x'}_{pub}||\mathbf{x'}_{priv}$ are indistinguishable if all published function keys do not separate them.

- *Function Hiding for the second Slot:* We require hiding for the second component of the vector in the secret key. That is, the following two worlds are indistinguishable: In the first world, one gets the secret key for $\mathbf{y}_{pub}||\mathbf{y}_{priv}$ and the ciphertext for $0||\mathbf{x}_{priv}$, and in the second world, one gets the secret key for $\mathbf{y}_{pub}||\mathbf{y}'_{priv}$ and the ciphertext for $0||\mathbf{x}'_{priv}$ such that $\langle \mathbf{x_{priv}}, \mathbf{y_{priv}} \rangle = \langle \mathbf{x_{priv}}', \mathbf{y_{priv}}' \rangle$.

It turns out that this notion is the right combination of public-key and function-hiding FE which is both achievable and useful. Slotted IPE is similar in spirit to objects defined in [KSW08] and also similar to (but simpler than) the slotted FE definition of [GGHZ16]. We refer the reader to Section 6 for the definition and construction of the slotted IPE scheme. Replacing the secret-key IPE with a slotted IPE in our construction yields a public-key FE for NC^0 with linear efficiency.

Constructing Slotted IPE from Joint SXDH. The final piece of the puzzle is to construct a slotted IPE scheme. We do this by combining our secret key function hiding IPE scheme, derived from the BJK scheme [BJK15], and the public-key IPE scheme of Abdalla et al. [ABDP15]. We refer the reader to Section 7 for the construction of the FE scheme.

1.3 Noisy Graded Encodings

So far, we described our constructions and security proof in the language of *clean* graded encodings, however all the instantiations [GGH13a, CLT13, GGH15] are for *noisy* graded encodings. (see Section 4 for more details). Our constructions of FE for NC^0 and IO can be instantiated with noisy graded encodings. However, the security reduction to the joint-SXDH assumption does not go through, as the reduction performs scalar multiplication in order to "re-purpose" elements in the joint-SXDH assumption, to elements in the security experiments of FE for NC^0. Known noisy instantiations, when modified to support scalar multiplication, succumb to attacks. Nevertheless, our FE for NC^0 scheme when instantiated with *ideal* noisy graded encodings is secure. We leave it as an interesting open question whether our construction of FE or IO is secure (or can be made secure) in the recently proposed *weakly ideal* multilinear model [MSZ16a, MSZ16b, GMS16] which seems to capture all known attacks against noisy graded encodings.

1.4 Local PRGs

We briefly survey constructions of low depth PRGs. See Applebaum's book [App14] for more references and discussions. The existence of PRG in TC^0 follows from a variety of hardness assumption including intractability of factoring, discrete logarithm, or lattice problems (e.g. [NR95, NR97, NRR00, BPR12]). More works focused on PRG in NC^0. On the negative side, it was shown that there is no PRG in NC^0_4 (with output locality 4) achieving super-linear stretch [CM01, MST03]. On the positive side, Applebaum, Ishai, and Kushilevitz [AIK04] showed that any PRG in NC^1 can be efficiently "compiled" into a PRG in NC^0 using randomized encodings, but with only *sub-linear* stretch. Unfortunately, to the best of our knowledge, there is no construction of PRG in NC^0 with super-linear stretch from well-known assumptions. But, there are candidate constructions.

The authors of [AIK04] constructed a *linear-stretch* PRG in NC^0 under a specific intractability assumption related to the hardness of decoding "sparsely generated" linear codes [AIK08], previously conjectured by Alekhnovich [Ale03]. Goldreich's one-way functions $f : \{0,1\}^n \to \{0,1\}^m$ where each bit of output is a fixed predicate P of a constant number d of input bits chosen at random, is also a candidate PRG when $m > n$. Several works investigated the (in)security of Goldreich's OWFs and PRGs: So far, there are no successful attacks when the choice of the predicate P avoids certain degenerate cases [CEMT09, BQ12, OW14, AL15].

1.5 Organization

We show the construction of IO from FE for NC^0 in Section 3. Section 5 shows a function-hiding secret-key IPE scheme and Section 6 defines the notion of slotted IPE and uses it to construct a public-key IPE scheme with the right notion of function hiding. In Section 4, we define our notion of tree-structured GES schemes and the joint-SXDH assumption on them. We also show how to use tree-GES to implement other common forms of graded encoding schemes such as set-multilinear maps and graph-based multilinear maps. Finally, in Section 7, we put these components together to construct our FE scheme.

2 Preliminaries

Let $\mathbb{Z}$ and $\mathbb{N}$ denote the set of integers, and positive integers, respectively. Let $[n]$ denote the set $\{1, 2, \ldots, n\}$, $\mathcal{R}$ denote a ring with $\mathbf{0}$ and $\mathbf{1}$ being the additive and multiplicative identities.

We denote by PPT probabilistic polynomial time Turing machines. The term *negligible* is used for denoting functions that are (asymptotically) smaller than any inverse polynomial. More precisely, a function $\nu(\star)$ from non-negative integers to reals is called *negligible* if for every constant $c > 0$ and all sufficiently large n, it holds that $\nu(n) < n^{-c}$.

2.1 μ-Indistinguishability

Definition 1 (μ-indistinguishability)**.** *Let $\mu : \mathbb{N} \to [0,1]$ be a function. A pair of distribution ensembles $\{X_\lambda\}_{\lambda\in\mathbb{N}}$, $\{Y_\lambda\}_{\lambda\in\mathbb{N}}$ are μ-indistinguishable if for every non-uniform PPT distinguisher D, every sufficiently large security parameter $\lambda \in \mathbb{N}$, and auxiliary input $z \in \{0,1\}^{\mathrm{poly}(\lambda)}$, it holds that*

$$|\Pr[x \xleftarrow{\$} X_\lambda : D(1^\lambda, x, z) = 1] - \Pr[y \xleftarrow{\$} Y_\lambda : D(1^\lambda, y, z) = 1]| \le \mu(\lambda)$$

Definition 2 (Computational and Sub-exponential Indistinguishability)**.** *A pair of distribution ensembles $\{X_\lambda\}_{\lambda\in\mathbb{N}}$, $\{Y_\lambda\}_{\lambda\in\mathbb{N}}$ are computationally indistinguishable if they are $1/p$-indistinguishable for every polynomial p, and are sub-exponentially indistinguishable if they are μ-indistinguishable for some sub-exponentially small $\mu(\lambda) = 2^{\lambda^\varepsilon}$ with a constant $\varepsilon > 0$.*

Note that the above definition of sub-exponential indistinguishability is weaker than standard sub-exponential hardness assumptions that consider distinguishers running in sub-exponential *time*.

Below, we provide definitions of standard cryptographic primitives using the terminology of μ-indistinguishability, which implicitly defines variants with polynomial or sub-exponential security. As a matter of convention, we will drop μ when μ is a negligible function, and say sub-exponential security when μ is a sub-exponentially small function.

2.2 Indistinguishability Obfuscation

We recall the notion of indistinguishability obfuscation for a class of circuit defined by [BGI+01b], adding the new dimension that the class of circuits may have restricted domains $\{\mathbb{D}_\lambda\}_{\lambda\in\mathbb{N}}$.

Definition 3 (Indistinguishability Obfuscator ($i\mathcal{O}$) for a circuit class). *A uniform* PPT *machine $i\mathcal{O}$ is an indistinguishability obfuscator for a class of circuits $\{\mathcal{C}_\lambda\}_{\lambda\in\mathbb{N}}$ (with potentially restricted domains $\{\mathbb{D}_\lambda\}_{\lambda\in\mathbb{N}}$), if the following conditions are satisfied:*

Correctness: *For all security parameters $\lambda \in \mathbb{N}$, for every $C \in \mathcal{C}_\lambda$, and every input x (in $\mathbb{D}_\lambda$), we have that*

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

where the probability is taken over the coin-tosses of the obfuscator $i\mathcal{O}$.

μ-Indistinguishability: *For every ensemble of pairs of circuits $\{C_{0,\lambda}, C_{1,\lambda}\}_{\lambda\in\mathbb{N}}$ satisfying that $C_{b,\lambda} \in \mathcal{C}_\lambda$, $|C_{0,\lambda}| = |C_{1,\lambda}|$, and $C_{0,\lambda}(x) = C_{1,\lambda}(x)$ for every x (in $\mathbb{D}_\lambda$), the following ensembles of distributions are μ-indistinguishable:*

$$\left\{C_{1,\lambda}, C_{2,\lambda}, i\mathcal{O}(1^\lambda, C_{1,\lambda})\right\}_{\lambda\in\mathbb{N}}$$
$$\left\{C_{1,\lambda}, C_{2,\lambda}, i\mathcal{O}(1^\lambda, C_{2,\lambda})\right\}_{\lambda\in\mathbb{N}}$$

Definition 4 (IO for P/poly). *A uniform* PPT *machine $i\mathcal{O}_{\mathsf{P/poly}}(\star,\star)$ is an indistinguishability obfuscator for* P/poly *if it is an indistinguishability obfuscator for the class $\{\mathcal{C}_\lambda\}_{\lambda\in\mathbb{N}}$ of circuits of size at most λ.*

2.3 Pseudorandom Generator

Definition 5 (Pseudo-Random Generator (PRG)). *Let ℓ be a polynomial-bounded function. A deterministic polynomial-time uniform machine $\mathbf{PRG}$ is a $\ell(\lambda)$-stretch pseudorandom generator if the following conditions are satisfied:*

Syntax *For every $\lambda \in \mathbb{N}$ and every $r \in \{0,1\}^\lambda$, $\mathbf{PRG}(r)$ outputs $r' \in \{0,1\}^{\ell(\lambda)}$*

μ-Indistinguishability: *The following ensembles are μ-indistinguishable*

$$\left\{r \xleftarrow{\$} \{0,1\}^\lambda : \mathbf{PRG}(r)\right\} \approx_\mu \left\{r' \xleftarrow{\$} \{0,1\}^{\ell(\lambda)}\right\}$$

For every $\lambda \in \mathbb{N}$, let $\mathbf{PRG}_\lambda : \{0,1\}^\lambda \to \{0,1\}^{\ell(\lambda)}$ denote the total Boolean function corresponding to $\mathbf{PRG}$ for λ-bit inputs. We say that $\mathbf{PRG}$ has degree d (for a universal constant d), if for every λ, every output bit of $\mathbf{PRG}_\lambda$ is a degree d function of the input bits.

Since every $\mathbf{PRG}_\lambda$ is a total function, by the Nisan-Szegedy result [NS94], a constant-degree pseudorandom generator $\mathbf{PRG}$ must belong to NC^0.

Claim 1. *Every constant-degree $\mathbf{PRG}$ is in NC^0.*

2.4 Randomized Encodings

In this section, we recall the traditional definition of randomized encodings with simulation security [IK02, AIK06].

Definition 6 (Randomized Encoding Scheme for Circuits)**.** *A Randomized Encoding scheme* **RE** *consists of two PPT algorithms,*

- $(\hat{C}, \hat{x}) \xleftarrow{\$} \mathsf{REnc}(1^\lambda, C, x)$*: On input a security parameter* 1^λ*, circuit* C*, and input* x*,* REnc *generates an encoding* $\hat{C}_x$*.*
- $y = \mathsf{REval}(\hat{C}_x)$*: On input* $\hat{C}_x$ *produced by* REnc*,* REval *outputs* y*.*

Correctness: *The two algorithms* REnc *and* REval *satisfy the following correctness condition: For all security parameters* $\lambda \in \mathbb{N}$*, circuit* C*, input* x*, it holds that,*

$$\Pr[\hat{C}_x \xleftarrow{\$} \mathsf{REnc}(1^\lambda, C, x) : \ \mathsf{Eval}(\hat{C}_x) = C(x)] = 1$$

μ**-Simulation Security:** *There exists a PPT algorithm* RSim*, such that, for every ensemble* $\{C_\lambda, x_\lambda\}$ *where* $|C_\lambda|, |x_\lambda| \leq \mathrm{poly}(\lambda)$*, the following ensembles are* μ*-indistinguishable for all* $\lambda \in N$*.*

$$\left\{\hat{C}_x \xleftarrow{\$} \mathsf{REnc}(1^\lambda, C, x) : \hat{C}_x\right\}$$
$$\left\{\hat{C}_x \xleftarrow{\$} \mathsf{RSim}(1^\lambda, C(x), 1^{|C|}, 1^{|x|}) : \hat{C}_x\right\}$$

where $C = C_\lambda$ *and* $x = x_\lambda$*.*

Furthermore, let $\mathcal{C}$ *be a complexity class, we say that randomized encoding scheme* **RE** *is in* $\mathcal{C}$*, if the encoding algorithm* REnc *can be implemented in that complexity class.*

2.4.1 Program-Decomposable Randomized Encodings

Roughly speaking, a program-decomposable randomized encoding (PDRE) scheme is a randomized encoding (RE) scheme with a special syntax, but the same security and correctness properties (See Section 2.4 for a definition of randomized encoding schemes). Roughly speaking, a PDRE scheme is an RE scheme whose encodings consist of many components — polynomial in the size of the circuit— where each component can be generated in time *independent of* the complexity of the computation. Perhaps the most famous example of a PDRE scheme is Yao's garbled circuits [Yao86] where the components are the "garbled gates" which can each be generated very quickly.

Formally, a program-decomposable randomized encoding scheme **PDRE** consists of three algorithms $(\mathsf{Decomp}, \mathsf{REnc}, \mathsf{REval})$. For any $\lambda \in \mathbb{N}$, any function $f \in \mathcal{F}_{\lambda,N,D,S}$ described as (C, ρ), and input $x \in \{0,1\}^N$, the algorithms proceed as follows. (We denote by $\ell = \ell(\lambda, N, S)$ the upperbound on the length of the encodings as well as the number of random bits needed for generating the encoding.)

- *Program Decomposition:* $\mathsf{Decomp}(1^\lambda, f)$ "decomposes" f into ℓ components $\{(F_i, S_i, I_i)\}_{i\in[\ell]}$, where F_i describes a part of the function f, $S_i \subseteq [N(\lambda)]$ is a set of indices of input bits (generated depending on ρ), and $I_i \subseteq [\ell]$ is a set of indices of random bits.

- *Encoding:* REnc on input $x \in \{0,1\}^N$, and the "function decomposition" $\{(F_i, S_i, I_i)\}_{i\in[\ell]}$ samples a random string $r \xleftarrow{\$} \{0,1\}^\ell$, and produces each bit of the encoding $\{\widehat{f(x)}_i\}_{i\in[\ell]}$ independently using only a single function component:

$$\left\{\widehat{f(x)}_i = \mathsf{renc}(F_i, x|_{S_i}, r|_{I_i})\right\}_{i\in[\ell]}$$

- *Evaluation:* REval on input $\widehat{f(x)}$ outputs a value $y = f(x)$.

2.4.2 Linear Efficiency

In [AJS15], it suffices to require that the upperbound ℓ on the size of the randomized encoding, and the number of random bits needed for encoding to be $\mathrm{poly}(\lambda, S, N)$, and each output bit can be computed in a fixed polynomial size $\mathrm{poly}(\lambda)$ independent of S and N. Here, we consider stronger efficiency requirements:

- The upper bound ℓ is linear in the size of the circuit, and independent of the input length. That is,

$$\ell(\lambda, N(\lambda), S(\lambda)) = p(\lambda) \cdot S(\lambda)$$

  Such a requirement makes sense for NC^0 circuits where the input may be much longer than the circuit description. *This property is achieved by Yao's garbled circuit, where the encoding can be divided into exactly $S(\lambda)$ parts, each depending on a single gate in the circuit, and can be computed in a fixed polynomial time and in logarithmic depth, assuming PRG in NC^1.*

- Each output bit can be computed in constant size. That is, $\mathsf{renc} \in \mathsf{NC}^0$.

  This second property is achieved by the information theoretically secure randomized encoding for NC^1 by [AIK04] (AIK), which is in NC^0 with input locality 4 and depth 3.

By the composability of randomized encodings [AIK04], we can achieve achieve both efficiency properties simultaneously, by encoding the computation of each part in Yao's garbling of a computation, using the AIK randomized encoding. We denote the composed scheme the Yao-AIK scheme.

2.5 Functional Encryption

We provide the definition of a public-key functional encryption (FE) scheme with indistinguishability-based security which originally appeared in [BSW12, O'N10]. Below we define public key functional encryption first, and then note the difference with secret key functional encryption.

2.5.1 Public-Key Functional Encryption

Syntax. Let $\{\mathcal{X}_\lambda\}_{\lambda\in\mathbb{N}}$ and $\{\mathcal{Y}_\lambda\}_{\lambda\in\mathbb{N}}$ be ensembles of sets. Let $\{\mathcal{F}_\lambda\}_{\lambda\in\mathbb{N}}$, where every function in the set $\mathcal{F}_\lambda$ maps inputs in $\mathcal{X}_\lambda$ to outputs in $\mathcal{Y}_\lambda$.

A public-key functional encryption (FE) scheme **FE** for $\{\mathcal{F}_\lambda\}_{\lambda\in\mathbb{N}}$ consists of four PPT algorithms $(\mathsf{FE.Setup}, \mathsf{FE.KeyGen}, \mathsf{FE.Enc}, \mathsf{FE.Dec})$.

- *Setup:* $\mathsf{FE.Setup}(1^\lambda)$ is an algorithm that on input a security parameter outputs a master public key and a master secret key $(\mathsf{mpk}, \mathsf{msk})$.

- *Key Generation:* $\mathsf{FE.KeyGen}(\mathsf{msk}, f)$ on input the master secret key and the description of a function $f \in \mathcal{F}_\lambda$, outputs a functional secret key sk_f.
- *Encryption:* $\mathsf{FE.Enc}(\mathsf{mpk}, x)$ on input the master public key and a message $x \in \mathcal{X}_\lambda$, outputs an encryption ct of x.
- *Decryption:* $\mathsf{FE.Dec}(\mathsf{sk}, \mathsf{ct})$ on input the secret key associated with f and an encryption of x, outputs $y \in \mathcal{Y}_\lambda$.

Correctness: We define perfect correctness here. For every λ, $f \in \mathcal{F}_\lambda$, $x \in \mathcal{X}_\lambda$, it holds that,

$$\Pr\left[\begin{array}{c} (\mathsf{mpk}, \mathsf{msk}) \overset{\$}{\leftarrow} \mathsf{FE.Setup}(1^\lambda) \\ \mathsf{ct} \overset{\$}{\leftarrow} \mathsf{FE.Enc}(\mathsf{mpk}, x) \\ \mathsf{sk} \overset{\$}{\leftarrow} \mathsf{FE.KeyGen}(\mathsf{msk}, f) \end{array} : f(x) = \mathsf{FE.Dec}(\mathsf{sk}, \mathsf{ct})\right] = 1$$

Indistinguishability Security. Indistinguishability security of a functional encryption requires that no adversary can distinguish the FE encryption of one input x_0 from that of another x_1, if the adversary only obtains secret keys sk_f for functions f that yield the same output on x_0 and x_1, that is, $f(x_0) = f(x_1)$. In the adaptive setting, the two challenge inputs (x_0, x_1) and all functions f are chosen adaptively by the adversary. In the weaker selective setting, the adversary is restricted to choose (x_0, x_1) and all functions f statically.

Definition 7 (IND-Security). *A public-key FE scheme* $\mathbf{FE} = (\mathsf{FE.Setup}, \mathsf{FE.KeyGen}, \mathsf{FE.Enc}, \mathsf{FE.Dec})$ *for* $\{\mathcal{F}_\lambda\}_{\lambda \in \mathbb{N}}$ *is* μ-IND-secure, *if for every* PPT *adversary* A*, and every sufficiently large security parameter* $\lambda \in \mathbb{N}$*, the adversary's advantage in the following games is bounded by* $\mu(\lambda)$

$$\mathsf{Advt}_A^{\mathbf{FE}} = \left|\Pr[\mathsf{Exp}_A^{\mathbf{FE}}(1^\lambda, 0) = 1] - \Pr[\mathsf{Exp}_A^{\mathbf{FE}}(1^\lambda, 1) = 1]\right| \le \mu(\lambda)$$

$\mathsf{Exp}_A^{\mathbf{FE}}(1^\lambda, b)$ *proceeds as follows:*

1. **Key Generation.** *The challenger* CH *samples* $(\mathsf{mpk}, \mathsf{msk}) \overset{\$}{\leftarrow} \mathsf{FE.Setup}(1^\lambda)$ *and sends* mpk *to the adversary.*
2. **Function Queries.** *Repeat the following for an arbitrary number of times determined by* A*: Upon* A *choosing a function query* $f \in \mathcal{F}_\lambda$*,* CH *sends* A *a function key* $\mathsf{sk}_f \overset{\$}{\leftarrow} \mathsf{FE.KeyGen}(\mathsf{msk}, f)$*.*
3. **Message Queries.** *Upon* A *choosing a pair of messages* (x_0, x_1)*,* CH *sends* A *a ciphertext* $\mathsf{ct} \overset{\$}{\leftarrow} \mathsf{FE.Enc}(\mathsf{mpk}, x_b)$*.*
4. **Function Queries** *Repeat the second step, again for an arbitrary number of times determined by* A*.*
5. *Finally* A *outputs a bit* b' *which is also the output of the experiment.*

Restriction: *Every function query* f *must satisfy that* $f(x_0) = f(x_1)$*.*

Definition 8 (Selective Security). *We say that* $\mathbf{FE}$ *is* μ-selectively secure *if the condition in Definition 7 holds for modified experiments* $\mathsf{Sel.Exp}_A^{\mathbf{FE}}(1^\lambda, b)$ *where the adversaries choose challenge messages* (x_0, x_1) *immediately after receiving the master public key and before requesting any function keys.*

Note that our notion of selective security is stronger than other notions in the literature, which requires the adversaries to choose x_0, x_1 at the beginning of the experiments before receiving the master public key, or even requiring the adversaries to choose all function queries also at the beginning of the experiment. Our construction of FE scheme later achieves this stronger form of selective security.

Definition 9 (1-Key FE)**.** *We say that* $\mathbf{FE}$ *is a* μ*-secure (or* μ*-selectively secure) 1-key FE scheme if it satisfies the security requirements in Definition 7 (or, respectively, Definition 8) against adversaries that ask for at most one function key query.*

2.5.2 FE Schemes for $\mathsf{P/poly}$, NC^1 and NC^0

Consider the following function class parameterized by (efficiently computable) polynomials N, D, and S, which denote upper bounds on, respectively, the length of the inputs, the depth of the circuits, and the size of the circuits computing the function class.

Function Class $\{\mathcal{F}_{\lambda,N,D,S}\}$**.** The domain is the set of all $N(\lambda)$-bit strings $\mathcal{X}_\lambda = \{0,1\}^{N(\lambda)}$. Every function f in set $\mathcal{F}_{\lambda,N,D,S}$ is described by a tuple (C, ρ): C is a circuit of depth at most $D(\lambda)$ and size at most $S(\lambda)$, and ρ is a subset of indexes of $[N(\lambda)]$. (C, ρ) satisfies that for every $x \in \{0,1\}^{N(\lambda)}$, $f(x) = C(x|_\rho)$, where $x|_\rho$ is the projection of the bits of x in the subset ρ.

Note that $f(x) = C(x|_\rho)$ effectively means that f depends only on a subset of the input bits. Therefore, the size $S(\lambda)$ of the circuit C may be much smaller than the input length $N(\lambda)$. This allows us to consider, for example, NC^0 functions that computes over a constant number of bits in a polynomal length input. Additionally, the pair (C, ρ) can be described in $\mathrm{poly}(S(\lambda), N(\lambda))$ bits.

Definition 10 (Families of FE schemes)**.** *Let* $\mathcal{N}, \mathcal{D}, \mathcal{S}$ *be subsets of polynomial functions. We say that* $\mathcal{FE} = \{\mathbf{FE}_{N,D,S}\}$ *is a family of (1-key) FE schemes for* $\mathcal{N}, \mathcal{D}, \mathcal{S}$ *with* μ*-security (or* μ*-selective security) if for every polynomials* $N \in \mathcal{N}, D \in \mathcal{D}, S \in \mathcal{S}$, $\mathbf{FE}_{N,D,S}$ *is a* μ*-secure (or* μ*-selectively secure, resp.) (1-key) FE scheme for the function class* $\{\mathcal{F}_{\lambda,N,D,S}\}$.

Moreover, define the following special cases:

- *When* $\mathcal{N}, \mathcal{D}, \mathcal{S}$ *are the sets of all polynomials,* $\mathcal{FE}$ *is a family of FE schemes for* $\mathsf{P/poly}$.
- *When* $\mathcal{N}, \mathcal{S}$ *are the sets of all polynomials and* $\mathcal{D}$ *is the set of all logarithmic functions,* $\mathcal{FE}$ *is a family of FE schemes for* NC^1.
- *When* $\mathcal{N}, \mathcal{S}$ *are the sets of all polynomials and* $\mathcal{D}$ *is the set of all constant functions,* $\mathcal{FE}$ *is a family of FE schemes for* NC^0.

2.5.3 Compactness

In the above definition of families of FE schemes, algorithms in scheme $\mathbf{FE}_{N,D,S}$ could run in polynomial time depending on polynomials N, D, S. In the literature, stronger efficiency requirements have been considered. In particular, the works of [AJ15, BV15] defined compact FE schemes for NC^1, which requires the encryption time to be independent of the circuit size S of the functions.

Definition 11 (Compactness of FE schemes for NC^1)**.** *Let* $\mathcal{FE} = \{\mathbf{FE}_{N,D,S}\}$ *be a family of FE schemes for* NC^1.

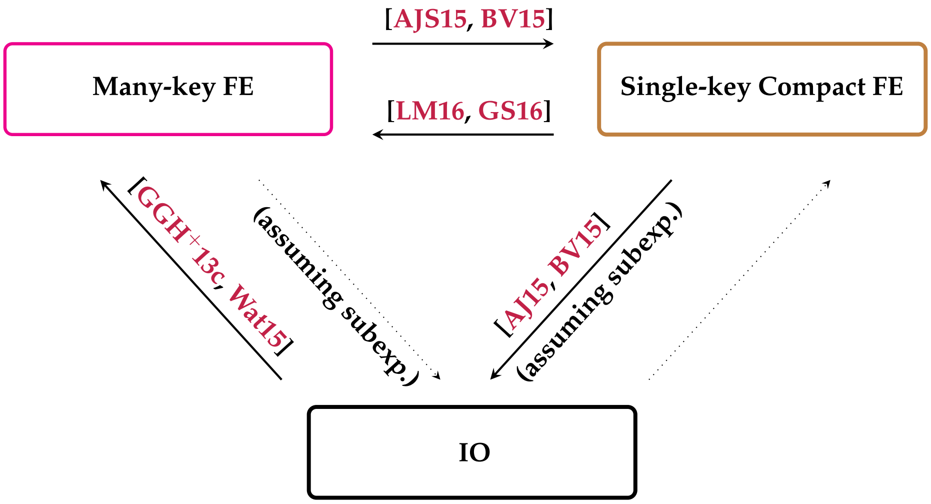

Figure 2: Solid lines indicate direct implications, and dotted lines indicate implications that follow as corollaries by combining the solid lines. By default, all FE schemes referred to here are selectively secure. Whenever there is an implication of the form $X \rightarrow Y$, the reader should interpret that as X for P/poly implies Y for P/poly. However, the actual theorems are somewhat stronger. For example, we know that assuming the existence of weak PRFs in NC^1, many-key FE for NC^1 implies single-key compact FE for all of P/poly [AJS15, BV15].

Compactness: *We say that the functional encryption scheme $\mathcal{FE}$ is compact if for every logarithmic function D, there is a polynomial p, such that, for every polynomials N, S, the encryption algorithm of $\mathbf{FE}_{N,D,S}$ runs in time $p(\lambda, N(\lambda), \log S(\lambda))$.*

$(1-\varepsilon)$-Sublinear Compactness: *We say that $\mathcal{FE}$ is $(1-\varepsilon)$-sublinearly compact, if for every logarithmic function D, there is a polynomial p, such that, for every polynomials N, S, the encryption algorithm of $\mathbf{FE}_{N,D,S}$ runs in time $p(\lambda, N(\lambda)) \cdot S(\lambda)^{1-\varepsilon}$.*

3 Indistinguishability Obfuscation from NC^0-Functional Encryption

Recent results show a strong connection between Indistinguishability obfuscation (IO) and Functional encryption (FE). Here, as before, we distinguish between two different notions of functional encryption: the standard notion of many-key secure (or, collusion-resistant) functional encryption and single-key secure compact functional encryption.[4] On the one hand, [GGH+13c] showed that IO for NC^1 implies many-key FE for P/poly. On the other hand, [AJ15, BV15] showed that (subexponentially secure) single-key compact FE for NC^1 implies IO for P/poly. Finally, [BV15, AJS15] showed that many-key FE for NC^1 implies single-key compact FE for P/poly. Put together, this shows that all three notions – IO, single-key compact FE, and many-key FE – are equivalent upto subexponential security considerations. See Figure 2 for a detailed description of these connections.

Theorem 4. *Assume the existence of a weak PRF in NC^1. Then:*

[4] In the rest of our exposition, when we talk of the security of functional encryption, we will mean selective security. Ananth, Brakerski, Segev and Vaikuntanathan [ABSV15] have recently showed how to transform selectively secure many-key FE schemes into adaptively (fully) secure many-key FE schemes.

- [AJ15, BV15] *For any constant $\epsilon > 0$, the existence of a single-key FE scheme $\mathcal{FE} = \{\mathbf{FE}_{N,D,S}\}$ for* NC^1 *with sub-exponential, selective, security and $(1-\varepsilon)$-sublinear-compactness implies IO for* $\mathsf{P/poly}$.
- [AJS15, BV15] *The existence of a many-key FE scheme $\mathcal{FE}' = \{\mathbf{FE}'_{N',D',S'}\}$ for* NC^1 *with sub-exponential, selective, security implies the existence of a single-key FE scheme $\mathcal{FE} = \{\mathbf{FE}_{N,D,S}\}$ for* NC^1 *with sub-exponential, selective, security and $(1-\varepsilon)$-sublinear-compactness.*

Put together, if there is a many-key secure (but not necessarily compact) FE scheme for NC^1, *then there is an IO scheme for* $\mathsf{P/poly}$.

The authors of [AJS15] further noted that it suffices to use a PRG instead of a PRF in their construction, which still suffices for constructing compact 1-key FE scheme for polynomially bounded computations.

In this section, we show that by carefully instantiating the cryptographic primitives underlying the construction of [BV15, AJS15], we obtain that IO for $\mathsf{P/poly}$ is implied by FE for NC^0 functions, assuming the existence of subexponentially secure $\mathbf{PRG}$ in NC^0.

Theorem 5 (From NC^0-FE to IO). *Let $\alpha > 0$ be a positive constant, and let $\mathbf{PRG}$ be a sub-exponentially secure* $n^{1+\alpha}$-stretch *pseudo-random generator computable by circuits of depth $d = d(\lambda)$. Let $\mathcal{FE} = \{\mathbf{FE}_{N,D,S}\}$ be a many-key functional encryption scheme for circuits of depth $D(\lambda) = d(\lambda) + 5$ with sub-exponential selective security and the following encryption efficiency:*

> *Encryption efficiency: the encryption time of $\mathbf{FE}_{N,D,S}$ is linear in the length of the input, that is, bounded by $p(\lambda, S(\lambda)) \cdot N(\lambda)$ for a fixed polynomial p.*

Then, there exists an indistinguishability obfuscator $\mathbf{IO}$ for $\mathsf{P/poly}$.

In particular, if $\mathbf{PRG}$ is in NC^0, *then FE schemes for* NC^0 *with encryption time linear in the input length imply IO for* $\mathsf{P/poly}$.

The proof of the theorem is essentially identical to that of Theorem 4 by [AJS15]. Below we provide a proof sketch, and focus only on showing how carefully choosing the primitives underlying the result [BV15, AJS15] affects the parameters of the FE scheme needed for IO.

3.1 Proof of Theorem 5

To construct an indistinguishability obfuscator (IO) for all functions in $\mathsf{P/poly}$ starting from a many-key functional encryption (FE) scheme, Bitansky and Vaikuntanathan [BV15] and Ananth, Jain and Sahai [AJS15] proceed in two steps.

Step 1. Construct sub-exponentially secure single-key FE schemes $\mathbf{CFE} = \{\mathbf{CFE}_{N,D,S}\}$ with $(1-\varepsilon)$-sublinear compactness for NC^1 circuits; and

Step 2. Invoke the result of [AJ15, BV15] (resp. [BNPW16] in the case of secret-key FE).

We recall the transformation of [BV15, AJS15] below, and instantiate their components carefully to obtain Theorem 5.

Tools Used for Obtaining Theorem 5 We will choose the following instantiations of the three tools used in [AJS15]:

- A pseudorandom generator **PRG** computable in depth $d = d(\lambda)$ with $n^{1+\alpha}$-stretch for any constant α. We assume implicitly that $d = O(\log \lambda)$.
- Selectively secure (collusion resistant) FE schemes **CRFE** $= \{\mathbf{CRFE}^{N',D',S'}\}$ for circuits with fixed depth $D' = D'(\lambda) = d(\lambda) + 5$ and encryption time linear in input length, namely $\mathrm{poly}(\lambda, S') \cdot N'$.
- A specific PDRE scheme **PDRE** $= (\mathsf{Decomp}, \mathsf{REnc}, \mathsf{REval})$ which is the composition of Yao's garbling scheme and the AIK randomized encoding scheme in NC^0. This scheme satisfies what we call *linear efficiency* (described below), and each output bit is computable in depth 3.

3.2 The Construction

For any polynomial functions N, D, S, the scheme

$$\mathbf{CFE}_{N,D,S} = (\mathsf{CFE.Setup}, \mathsf{CFE.KeyGen}, \mathsf{CFE.Enc}, \mathsf{CFE.Dec})$$

invokes **PRG**, **PDRE** and the $\mathbf{CRFE}_{N',D',S'}$ for N', D', S' set below. For any λ and any function $f \in \mathcal{F}_{\lambda,N,D,S}$ and input $x \in \{0,1\}^N$, the algorithms proceed as follows:

Single-key Compact FE Scheme CFE

- *Setup:* $\mathsf{CFE.Setup}(1^\lambda)$ samples $(\mathsf{mpk}, \mathsf{msk}) \xleftarrow{\$} \mathsf{CRFE.Setup}(1^\lambda)$.
- *Key Generation:* $\mathsf{CFE.KeyGen}(\mathsf{msk}, f)$ does the following:
  - Decompose function f to $\{(F_i, S_i, I_i)\}_{i \in [\ell]} \leftarrow \mathsf{Decomp}(1^\lambda, f)$.
  - For each index $i \in [\ell]$, sample a uniformly random bit CT_i and generate a key $\mathsf{sk}_i \xleftarrow{\$} \mathsf{CRFE.KeyGen}(\mathsf{msk}, f_i)$ for f_i defined as: On input x of length N, two PRG seeds s_1, s_2 each of length $\ell^{1/(1+\alpha)}$ and a bit b.

$$f_i(x, s_1, s_2, b) = \begin{cases} \mathsf{renc}(F_i, x|_{S_i}, \mathbf{PRG}(s_1)|_{I_i}) & \text{if } b = 0 \\ CT_i \oplus \mathbf{PRG}(s_2)|_i & \text{if } b = 1 \end{cases}$$

  (We show below in Claim 2 how to represent f_i as a circuit and index-set pair (C_i, ρ_i) such that $f_i(x, s_1, s_2, b) = C_i((x, s_1, s_2, b)|_{\rho_i})$, and analyze various bounds N', D', S' related to f_i.)
- *Encryption:* $\mathsf{CFE.Enc}(\mathsf{mpk}, x)$ samples $s_1, s_2 \xleftarrow{\$} \{0,1\}^{\ell^{1/(1+\alpha)}}$, and generates

$$\mathsf{ct} \xleftarrow{\$} \mathsf{CRFE.Enc}(\mathsf{mpk}, (x, s_1, s_2, 0))$$

- *Decryption:* $\mathsf{CFE.Dec}(\mathsf{sk}, \mathsf{ct})$ computes for each $i \in [\ell]$ the i^{th} bit of the encoding $\widehat{f(x)}_i = \mathsf{CRFE.Dec}(\mathsf{sk}_i, \mathsf{ct})$, and then evaluates the output $f(x) = \mathsf{REval}(\widehat{f(x)})$.

Figure 3: Caption

3.2.1 Compactness of CFE

We show that **CFE** is a compact FE scheme, which requires a new analysis different from [AJS15, BV15], since we use tools with different parameters. In particular, our PRG has only $n^{(1+\alpha)}$-stretch, as opposed to large polynomial stretch; to compensate for this, the Yao-AIK PDRE has linear efficiency.

We start by showing that each of the component functions f_i (see Figure 3) can be computed by circuits whose depth and size are completely independent of those of the original function f. Then, in Lemma 1, we show that **CFE** is compact.

Claim 2. *For every $\lambda \in \mathbb{N}$, the functions f_i used in* $\mathsf{CFE.Enc}$ *(Figure 3) belong to the function class $\mathcal{F}_{\lambda,N',D',S'}$, where*

$$N'(\lambda) = N(\lambda) + 2\ell(\lambda) + 1 \qquad D'(\lambda) = d(\lambda) + 5, \qquad S'(\lambda) = O(2^{D'(\lambda)})$$

Proof. To analyze the bounds N', D', S' on the input length, circuit depth, and circuit size of f_i, we first show how to present it as (C_i, ρ_i), such that, $f_i(x, s_1, s_2, b) = C_i((x, s_1, s_2, b)|_{\rho_i})$.

Towards this, let $(\mathbf{PRG}_j, \pi_j)$ be the pair describing the function $\mathbf{PRG}(s)_j$ that computes the j^{th} bit of the output of $\mathbf{PRG}$, that is, $\mathbf{PRG}_j(s|_{\pi_j}) = \mathbf{PRG}(s)|_j$. We first describe ρ_i and then the circuit C_i.

To compute f_i, the following bits in the entire input $X = (x, s_1, s_2, b)$ are needed.

- S_i contains the indexes of bits in x needed for computing f_i; let $x' = X|_{S_i}$.
- For every $j \in I_i$, $K_{i,j}$ contains the indexes of bits in s_1 needed for computing $\mathbf{PRG}(s_1)|_j$.
$$K_{i,j} = \{N + k \mid k \in \pi_j\}$$
Let $s'_{1,j} = X|_{K_{i,j}}$.
- L_i contains the indexes of bits in s_2 needed for computing $\mathbf{PRG}(s_2)|_i$.
$$L_i = \left\{N + \ell^{1/(1+\alpha)} + k \mid k \in \pi_i\right\}$$
Let $s'_2 = X|_{L_i}$
- The bit b has index $N + 2\ell^{1/(1+\alpha)} + 1$.

Therefore, we set
$$\rho_i = S_i \cup \left(\cup_{j\in[I_i]} K_{i,j}\right) \cup L_i \cup \{N + 2\ell^{1/(1+\alpha)} + 1\}$$

The circuit C_i on input $X' = X|_{\rho_i}$ can parse the input as $(x', \{s'_{1,j}\}_{j\in I_i}, s'_2, b)$, and compute $f_i(x)$ by computing the relevant PRG output bits, and the i^{th} bit of the encoding. To do so, the circuit C_i contains the circuits $\mathbf{PRG}_j$ for every $j \in I_i$ and $j = i$, and the circuit for computing renc. The procedure of C_i is described in Figure 4.

<u>*Parameters N', D', S'*</u> Clearly, the input (x, s_1, s_2, b) of f_i has length $N' = N + 2\ell^{1/(1+\alpha)} + 1$. The depth D' of the circuit C_i is dominated by the depth d of $\mathbf{PRG}$. Once the relevant PRG output bits are computed, the rest of the computation, namely, computing renc of the Yao-AIK scheme and branching depending on the mode b can be performed in depth 5 (the former can be done in depth 3, and the latter in 2). Therefore $D' = d + 5$. Finally, the size S' of C_i is again dominated by size of the circuits $\mathbf{PRG}_j$ computing individual output bits of $\mathbf{PRG}$, since the rest of computation (renc and branching depending on the mode) can be done in constant size. Hence, $S' = O(\max_j |PRG_j|)$. Since $\mathbf{PRG} \in \mathsf{NC}^1$, $|\mathbf{PRG}_j| \le 2^d$ for all j, and we can set $S' = O(2^d)$. □

Circuit $C[\{\mathbf{PRG}_j\}, \mathbf{PRG}_i, \mathsf{renc}, F, CT](X')$

Hardwired Content: A set of circuits $\{\mathbf{PRG}_j\}_{j \in J}$, $\mathbf{PRG}_i$ for computing a subset $J \cup \{i\}$ of output bits through $\mathbf{PRG}$ (in a valid circuit $J = I_i$), and the circuit for computing renc. A string F and a bit CT.

Input: A string X'. (A valid input $X' = X|_{\rho_i}$ for some input $X = (x, s_1, s_2, b)$ and $i \in [\ell]$.)

Procedure:

- Parse $X' = (x', \{s'_{1,j}\}_{j \in J}, s'_2, b))$. (Note that parsing can be done trivally through wiring the input bits appropriately)
- If $b = 0$
  1. For every $j \in J$, compute $r_j = \mathbf{PRG}_j(s'_{1,j})$.
  2. Compute $z = \mathsf{renc}(F, x', \{r_j\})$.
- If $b = 1$
  1. Compute $\sigma = \mathbf{PRG}_i(s'_2)$.
  2. Compute $z = CT \oplus \sigma$.

Output: The bit z.

Figure 4: Circuit C

Lemma 1. *Let $\alpha > 0$ be a constant and let $\epsilon := \alpha/(1 + \alpha)$. If $\mathbf{PRG}$ has $n^{1+\alpha}$-stretch and $\mathbf{PDRE}$ has linear efficiency, then the functional encryption scheme $\mathbf{CFE}$ is $(1 - \epsilon)$-sublinearly compact.*

Proof. The efficiency of $\mathbf{CRFE}_{N',D',S'}$ guarantees that when encrypting a string $X = (x, s_1, s_2, b)$ of length N' the encryption time is bounded by

$$\mathsf{Time}_{\mathsf{CRFE.Enc}}(\mathsf{mpk}, X) = \mathrm{poly}(\lambda, S') \cdot N' \,.$$

By the fact that $N' = N + 2\ell^{1/(1+\alpha)} + 1$, $S' = O(2^d)$ and $d = O(\log \lambda)$, and that the encryption time of $\mathbf{CFE}_{N,D,S}$ is dominated by the encryption time of $\mathbf{CRFE}_{N',D',S'}$, we have

$$\mathsf{Time}_{\mathsf{CFE.Enc}} = \mathrm{poly}(\lambda, N) \cdot \ell^{1/(1+\alpha)} \,.$$

Recall that $\ell = \ell(\lambda, N, S)$ is an upperbound on the length of the randomized encoding (and the number of random bits needed for generating it) for a computation with input length N, and circuit size S. By the linear efficiency of the Yao-AIK scheme, $\ell(\lambda, N, S) = \mathrm{poly}(\lambda)S$. Plugging this into the above equation for $\mathsf{Time}_{\mathsf{CFE.Enc}}$,

$$\mathsf{Time}_{\mathsf{CFE.Enc}} = \mathrm{poly}(\lambda, N)(\mathrm{poly}(\lambda)S)^{1/(1+\alpha)} = \mathrm{poly}(\lambda, N)S^{1/(1+\alpha)}$$

Therefore, $\mathbf{CFE}$ is $(1 - \alpha/(1 + \alpha))$-sublinearly compact. □

3.2.2 Security of CFE

Since the above construction of $\mathbf{CFE}$ can be viewed as a special case of that in [AJS15], using specific tools, it follows from their analysis that.

Lemma 2. $\mathbf{CFE}$ *defined above is a single-key selectively secure FE scheme for* NC^1.

3.3 IO from FE for Constant Degree Polynomials

As an interesting corollary of the bootstrapping technique, we can show how to get IO starting from FE for constant degree polynomials. We will not pursue this direction further in this paper, but offer this section as an observation of independent interest.

Theorem 5 directly implies that assuming the existence of subexponentially secure $n^{1+\alpha}$-stretch PRG in NC^0, it suffices to construct FE for Boolean NC^0, or equivalently, FE for constant degree polynomials. In this section, we give a more refinded analysis, and show that if the underlying PRG can be computed in degree $\deg$, then it suffice to have FE for degree $(2\deg+2)$ polynomials. In particular, if there exists a $n^{1+\alpha}$-stretch PRG with degree 2 over $GF(2)$, then FE for degree 6 polynomials over $GF(2)$ suffices.

Corollary 1 (IO from FE for Constant-Degree Polynomials). *Assume the existence of a sub-exponentially secure $n^{1+\alpha}$-stretch PRG $\mathbf{PRG}$ computable in degree $\deg$ for any positive constant α. Then, IO for $\mathsf{P/poly}$ is implied by (collusion resistant) FE schemes $\mathcal{FE}$ for degree $(2\deg+2)$ polynomials over $GF(2)$ with subexponential selective security and encryption time linear in the input length.*

Proof. Towards the corollary, we need to show that the function f_i used in the encryption algorithm $\mathsf{CFE.Enc}$ can be computed in degree $2\deg+2$. This follows since the computation in mode 0 has degree $2\deg+1$, and the computation in mode 1 has degree $\deg+1$. Then since branching based on mode can be done in degree 1, the overall degree is $2\deg+2$. In mode 0, one bit in the randomized encoding of the Yao-AIK scheme is computed. Since every encoding bit of Yao-AIK scheme has degree 2 in the random bits and degree 1 in the input bit, the degree of the output bit is exactly $2\deg+2$. In mode 1, it essentially computes a random bit using $\mathbf{PRG}$ and XOR it with a hardwired bit CT, and hence the degree is bounded by $\deg+1$. □

4 Graded Encoding with the Joint-SXDH Assumption

Our definition of Graded Encoding (GE) schemes for prime order rings (namely, rings whose additive group has prime order) is based on the definition of GE schemes from [GGH13b] and follow-up works. In a nutshell, a GES for a ring $\mathcal{R}$ consists of a collection of encodings that we denote $[a]_\ell$, where $a \in \mathcal{R}$ is a ring element, and ℓ is a "label" that comes from a label space $\mathcal{L}$. One can efficiently perform *constrained homomorphic computation* on these encodings. That is, (1) Encodings with the *same* label can be homomorphically added; and (2) Encodings with labels that satisfy certain *"pairable"* constraints can be homomorphically multiplied.

We also consider Noisy Graded Encoding (NGE) schemes which work as above except that there is a set of encodings of a ring element $a \in \mathcal{R}$ under a label $\ell \in \mathcal{L}$, and not a single one as with GE schemes. Correspondingly, an NGE scheme also comes with a *zero-testing* procedure that checks whether encodings with a special *zero-testing label* $\ell^\star$ belong to the set $[0]_{\ell^\star}$.

In the literature, there are many variants of GE (and NGE) schemes. Different variants use different types of labels (*e.g.*, integers, sets, paths in a graph), and correspondingly different criteria of being "pairable".

In Section 4.1, we first present a general formalization of GES that uses a generic label set $\mathcal{L}$, and a general function "**pairable**" to determine which two labels $l_1, l_2 \in \mathcal{L}$ can be paired together. Previous variants of GESs are special cases of this general formulation. In Section 4.4, we define the concrete label set $\mathcal{L}$ and the concrete function **pairable** used in this work, for the construction of FE schemes. We call this a *tree-GES* scheme. Additionally, we show that tree-GES can be implemented using either the graph-based or the set-based GESs from the literature [GGH13b, BGK+13, GGH15]. However, we prefer to use the tree-GES formulation as it expresses directly the properties we need in the construction. That is, we decouple the construction of FE schemes from our new GES variant, namely tree-GES, from the implementation of the new GES variant from more standard versions.

In Section 4.3, we consider a simple and concrete hardness assumption over GES, namely the *joint-SXDH assumption*. (It naturally corresponds to the joint-SXDH assumption over jointly sampled asymmetric bilinear maps in Section 5.5.)

Finally, looking ahead, we remark that in addition to homomorphic addition and multiplication, we also require efficient homomorphic scalar multiplication (in both our clean and noisy encoding definitions). Scalar multiplication is important for reducing the security of our FE scheme to the simple, instance independent, joint-SXDH assumption. The need for efficient homomorphic scalar multiplication is shared by other IO and FE constructions from simple assumptions such as [GLSW15, GLW14, GGHZ16].

4.1 Clean Graded Encoding Schemes

We first define (clean) graded encoding schemes and then generalize to define noisy ones.

Definition 12 (Clean Graded Encoding)**.** *Let $\mathcal{R}$ be a ring, $\mathcal{L} \subseteq \{0,1\}^*$ be a set of labels, and* $\mathbf{pairable} : \mathcal{L} \times \mathcal{L} \to \mathcal{L} \cup \{\bot\}$ *be a function. A graded encoding scheme* GES *for* $(\mathcal{R}, \mathcal{L}, \mathbf{pairable})$ *is associated with a tuple of* PPT *algorithms* $(\mathsf{InstGen}, \mathsf{Encode}, \mathsf{Add}, \mathsf{SMult}, \mathsf{Mult})$ *which behave as follows:*

- $(\mathsf{pp}, \mathsf{ep}) \overset{\$}{\leftarrow} \mathsf{InstGen}(1^\lambda)$. *The p.p.t. instance generation algorithm* $\mathsf{InstGen}$ *outputs a public parameter* pp *and and encoding parameter* ep *that describe an* $(\mathcal{R}, \mathcal{L}, \mathbf{pairable})$*-graded encoding scheme.*

- $A \leftarrow \mathsf{Encode}(\mathsf{ep}, \alpha \in \mathcal{R}, l \in \mathcal{L})$. *The* deterministic *encoding algorithm* Encode *takes as input the encoding parameter* ep*, a ring element α and a label l and outputs an encoding A. We will also use the shorthand $[\alpha]_l$ for the unique encoding of $\alpha \in \mathcal{R}$ under label $l \in \mathcal{L}$, namely the output of* $\mathsf{Encode}(\mathsf{pp}, \alpha \in \mathcal{R}, l \in \mathcal{L})$.

- $B \leftarrow \mathsf{Add}(\mathsf{pp}, A_1, A_2)$. *The p.p.t. addition algorithm takes as input the public parameters, encodings $A_1 = [\alpha_1]_{l_1}$ and $A_2 = [\alpha_2]_{l_2}$, and outputs*

$$B = A_1 \oplus A_2 = [\alpha_1 + \alpha_2]_l$$

  if $l_1 = l_2 = l$ and $\bot$ otherwise.

- $B \leftarrow \mathsf{SMult}(\mathsf{pp}, \beta \in \mathcal{R}, A)$. *The p.p.t. scalar multiplication algorithm takes as input the public parameters, an encoding $A = [\alpha]_l$ and a scalar $\beta \in \mathcal{R}$, and outputs*

$$B = \beta \odot A = [\beta\alpha]_l$$

- $B \leftarrow \mathsf{Mult}(\mathsf{pp}, A_1, A_2)$. *The p.p.t. multiplication algorithm takes as input the public parameters, encodings $A_1 = [\alpha_1]_{l_1}$ and $A_2 = [\alpha_2]_{l_2}$, and outputs*

$$B = A_1 \otimes A_2 = [\alpha_1\alpha_2]_{l_3}$$

  if $\mathbf{pairable}(l_1, l_2) = l_3 \in \mathcal{L}$ *and $\bot$ otherwise.*

Note that in the above definition, the ring $\mathcal{R}$, the label set $\mathcal{L}$, and the function **pairable** are fixed, independent of the security parameter. We can also consider a stronger version of GES for an ensemble of triples $\{(\mathcal{R}_\lambda, \mathcal{L}_\lambda, \mathbf{pairable}_\lambda)\}_{\lambda\in\mathbb{N}}$ depending on the security parameters.

Definition 13. *A graded encoding scheme* GES *for* $\{(\mathcal{R}_\lambda, \mathcal{L}_\lambda, \mathbf{pairable}_\lambda)\}_\lambda$ *is defined identically as in Definition 12, except that for every* $\lambda \in \mathbb{N}$*, the p.p.t. algorithm* InstGen *outputs a description of* $(\mathcal{R}_\lambda, \mathcal{L}_\lambda, \mathbf{pairable}_\lambda)$ *and the algorithms* (InstGen, Encode, Add, SMult, Mult) *behaves w.r.t. the ring* $\mathcal{R} = \mathcal{R}_\lambda$*, the label set* $\mathcal{L} = \mathcal{L}_\lambda$*, and function* $\mathbf{pairable} = \mathbf{pairable}_\lambda$*.*

For convenience of notation, in the rest of the paper, we will simply use binary operators $\oplus$, $\odot$ and $\otimes$ in place of the efficient algorithms Add, SMult and Mult.

4.2 Noisy Graded Encoding Schemes

We generalize the definition above to consider noisy encodings, where there are multiple encodings for the same ring element and label. This necessitates two changes to the definition, providing additional capabilities that were implicit in the definition of clean graded encodings.

Definition 14 (Noisy Graded Encoding)**.** *A noisy graded encoding scheme* GES *for* $(\mathcal{R}, \mathcal{L}, l^\star, \mathbf{pairable})$ *where* $l^\star \in \mathcal{L}$ *is associated with the* PPT *algorithms* (InstGen, Encode, Add, SMult, Mult, Eq) *just as in the clean graded encoding from Definition 12, with the addition of the equality testing algorithm* Eq*. We describe the additional properties of a noisy graded encoding (on top of what Definition 12 requires) below.*

- **Probabilistic Encoding Algorithm:** *The encoding algorithm* Encode(ep, α, l) *is a* probabilistic *algorithm (as opposed to deterministic in the definition of clean encodings). Consequently, we define* $[\alpha]_l$ *to be the set of all encodings of* $\alpha \in \mathcal{R}$ *under label* $l \in \mathcal{L}$ *produced by* Encode*.*

- **Disjoint Encodings:** *The sets of encodings of different ring elements under different labels are disjoint. That is, for every* $\alpha, \alpha' \in \mathcal{R}$ *and* $l, l' \in \mathcal{L}$*, if* $(\alpha, l) \neq (\alpha', l')$*,* $[\alpha]_l \cap [\alpha']_{l'} = \emptyset$*.*

  When there is no cause for confusion, we will slightly abuse notation and denote $A = [\alpha]_l$ *when instead we should be writing* $A \in [\alpha]_l$*.*

- **Equality Testing:** *There is an additional p.p.t. algorithm* Eq(pp, A_1, A_2) *that takes as input the public parameter* pp*, encodings* $A_1 = [\alpha_1]_{l^\star}$ *and* $A_2 = [\alpha_2]_{l^\star}$*, outputs 1 iff* $\alpha_1 = \alpha_2$*. We call* $l^\star$ *the equality-testing label. Note that for clean graded encoding structures, equality testing can be done efficiently for every label, by simply testing the equality of the encodings as strings.*

- μ**-Indistinguishable Linear-homomorphic Operations:** *For a linear function* $L : \mathcal{R}^n \to \mathcal{R}$*, we abuse notation and denote by* $L([\mathbf{x}]_l)$ *the operation of applying the linear function L to the encodings* $[x_1]_l, [x_2]_l, \ldots, [x_n]_l$*. This can be done using a combination of* Add *and* SMult*.*

  The indistinguishable-linear homomorphic operations guarantee that performing linear operations to the encodings of a vector produce indistinguishable encodings as directly encoding the outputs of the linear operations. In the case of clean graded encoding schemes, since encodings are unique, this property is automatically fulfilled. With this property, a noisy graded encoding scheme behaves like a clean graded encoding scheme. In the rest of the paper, for convenience, we abuse notation to write $L([\mathbf{x}]_l) = [L(\mathbf{x})]_l$*.*

  Formally, for every PPT *adversary* A*, we require the following games for* $b = 0$ *or* 1 *to be* μ*-indistinguishable.*

  <u>$\mathsf{Hom}_A^{\mathsf{GES}}(1^\lambda, b)$ *proceeds as follows:*</u>

- **Public Parameter.** *The challenger CH sends A the public and encoding parameters* $(\mathsf{pp}, \mathsf{ep}) \xleftarrow{\$} \mathsf{InstGen}(1^\lambda)$*, which describe a* $(\mathcal{R}, \mathcal{L}, l^\star, \mathbf{pairable})$*-noisy GES structure.*
- **Linear Function Query.** *Repeat the following for an arbitrary number of times determined by A. Upon A choosing a linear function L and a vector $\mathbf{x} \in \mathcal{R}^m$ of arbitrary length, and a label $l \in \mathcal{L}$, CH does:*
  - *If $b = 0$, sample* $[\mathbf{x}]_l \xleftarrow{\$} \mathsf{Encode}(\mathsf{ep}, \mathbf{x}, l)$ *and send A the encoding $X_0 = L([\mathbf{x}]_l)$.*
  - *If $b = 1$, send A the encoding* $X_1 \xleftarrow{\$} \mathsf{Encode}(\mathsf{ep}, L(\mathbf{x}), l)$*.*
- *Finally A outputs a bit b'.*

4.3 The joint-SXDH Assumption

We observe that encodings of different ring elements under the same label l "acts" as a group: $G_l = \{[\alpha]_l : \alpha \in \mathcal{R}\}$ with operation $\oplus$ that achieves $[\alpha_1]_l \oplus [\alpha_2]_l = [\alpha_1 + \alpha_2]_l$. (In the case of clean GES schemes, G_l is actually a group). Moreover, encodings under different labels l_1, l_2 can be paired together whenever $\mathbf{pairable}(l_1, l_2) = l_3 \neq \perp$. Therefore, it is natural to view encodings under labels l_1, l_2 that are pairable as a noisy generalization of the classical asymmetric bilinear groups, and GES as a further generalization that enables "a richer computation structure". We now naturally generalize the symmetric external Diffie-Hellman (SXDH) assumption over jointly sampled asymmetric bilinear groups to the setting of (noisy) GES.

Before we define the joint-SXDH assumption, we need to define the notion of the closure of the **pairable** relation, which we denote $\mathbf{pairable}^*$. $\mathbf{pairable}^*(l_1, l_2) = 1$ if there are labels l_3 and l_4 such that $\mathbf{pairable}^*(l_1, l_3) = 1$, $\mathbf{pairable}^*(l_2, l_4) = 1$, and $\mathbf{pairable}(l_3, l_4) \neq \perp$. Otherwise, $\mathbf{pairable}^*(l_1, l_2) = 0$.

The joint-SXDH Assumption The joint-SXDH assumption over GES requires that encodings of the same DDH tuple (a, b, ab) under a set S of labels to be indistinguishable to that of (a, b, r) under the same set, as long as no two labels in S are in $\mathbf{pairable}^*$, the cloure of the pairable relation. Formally, for every ensemble $\{S_\lambda\}$ of polynomial-sized sets satisfying that for every λ and every $i, j \in S_\lambda$, $\mathbf{pairable}^*_\lambda(l_i, l_j) = \perp$, the following two ensembles are μ-indistinguishable:

$$\left\{(\mathsf{pp}, \mathsf{ep}) \xleftarrow{\$} \mathsf{InstGen}(1^\lambda),\ a, b \xleftarrow{\$} \mathcal{R} \ :\ \mathsf{pp},\ \{[a]_l, [b]_l, [ab]_l\}_{l \in S_\lambda}\right\}_\lambda$$
$$\left\{(\mathsf{pp}, \mathsf{ep}) \xleftarrow{\$} \mathsf{InstGen}(1^\lambda),\ a, b, r \xleftarrow{\$} \mathcal{R} \ :\ \mathsf{pp},\ \{[a]_l, [b]_l, [r]_l\}_{l \in S_\lambda}\right\}_\lambda$$

When $|S_\lambda| = 1$, the above gives the SXDH assumption over GES. When $|S_\lambda| > 1$, clearly if any two labels l_i, l_j are pairable, the above ensembles are *distinguishable*. In a similar vein, if two labels l_i, l_j are in the closure of pairable, then again the above ensembles are *distinguishable*. However, when no pairs of labels are in the closure of pairable, the joint-SXDH assumption possibly holds.

4.4 Tree-GES: Graded Encoding for Depth-D 4-ary Trees

We now specify the parameters, $\{(\mathcal{R}_\lambda, \mathcal{L}_\lambda, l^\star_\lambda, \mathbf{pairable}_\lambda)\}_{\lambda \in \mathbb{N}}$, of the GES we use. Roughly speaking, we require the GES scheme to support a "computation structure" that corresponds to a depth-D (complete) 4-ary tree. When D is a constant, such a scheme can be instantiated with a GES for a fixed tuple $(\mathcal{L}, l^\star, \mathbf{pairable})$, and when D is a logarithmic function, it can be instantiated with GES for an ensemble $\{(\mathcal{L}_\lambda, l^\star_\lambda, \mathbf{pairable}_\lambda)\}_{\lambda \in \mathbb{N}}$. In both cases, we consider arbitrary prime order rings $\mathcal{R} \cong \mathbb{Z}_p$.

For convenience, we call such GES schemes, *tree-GES schemes*. The reason that we consider GES for both constant and logarithmic depth 4-ary tree is that the former suffices for our FE construction later if assuming the existence of a sub-exponentially secure PRG in NC^0, and the latter is needed if assuming sub-exponentially secure PRG in NC^1.

GES for Constant-Depth 4-ary Tree. Let D be a constant. Consider a 4-ary tree $\mathcal{T}$ of depth D, where the root is at layer 0 and the leaves are at layer D. Every node in layer $0 \leq d \leq D$ is labeled with a length-d 4-ary string, $l \in \{0,1,2,3\}^d$. In particular, the root is labeled with the empty string ε. A GES scheme for depth-D 4-ary tree is a GES scheme for a fixed tuple $(\mathcal{L}, l^\star, \mathbf{pairable})$ defined as follows:

- The label set $\mathcal{L} = \{0,1,2,3\}^{\leq D}$ is the set of nodes in a depth D 4-ary tree.

  Note: It will be instructive to think of encodings with label $l \in \{0,1,2,3\}^{\leq D}$ as being associated with node l in $\mathcal{T}$.

- The equality testing label $l^\star = \varepsilon$, corresponding to the root of $\mathcal{T}$.

- The function **pairable** is defined as follows: we allow the first two children $l||0$ and $l||1$, or the second two children $l||2$ and $l||3$ of any node l, to be paired together and produce output label l. That is,

$$\mathbf{pairable}^\star(l_1, l_2) = \begin{cases} l & \text{if } (l_1, l_2) = (l||0, l||1) \text{ or } (l||2, l||3) \\ \perp & \text{otherwise} \end{cases}$$

  We will let $\mathbf{pairable}^*$ to be the closure of **pairable**, as defined in Section 4.3. That is, we allow labels l_1, l_2 that are descendants of such two children to be paired together, still producing label l. Without loss of generality, assume that $l_1 \leq l_2$.

$$\mathbf{pairable}^*(l_1, l_2) = \begin{cases} l & \text{if } l_1, l_2 \text{ are descendants of } l_1', l_2' \text{ s.t. } (l_1', l_2') = (l||0, l||1) \text{ or } (l||2, l||3) \\ \perp & \text{otherwise} \end{cases}$$

GES for Logarithmic-Depth 4-ary Tree Let D be a logarithmic function. GES for Depth-D 4-ary Tree is a GES for the ensemble $\{(\mathcal{L}, l^\star, \mathbf{pairable})\}$ defined as follows: For every $\lambda \in \mathbb{N}$,

- The label set $\mathcal{L} = \mathcal{L}_\lambda \, \{0,1,2,3\}^{\leq D}$ is the set of labels of all nodes in the depth $D = D(\lambda) = c \log \lambda$ 4-ary tree.

- The equality testing label $l^\star = l^\star_\lambda = \varepsilon$.

- The function **pairable** is defined identically as above, over the bigger set of labels.

4.5 Connection with Set-based and Graph-based GES

We show that our GES for depth-D 4-ary trees can be implemented using both set-based and graph-based GES. Towards this, we first express set-based and graph-based GES in our framework, as GES for $(\mathcal{L}, l^\star, \mathbf{pairable})$. Then, we say that GES for $(\mathcal{L}, l^\star, \mathbf{pairable})$ is implemented by GES for $(\mathcal{L}', l^{\star\prime}, \mathbf{pairable}')$, if there exists a mapping function $\kappa : \mathcal{L} \to \mathcal{L}'$, such that,

$$\forall l_1, l_2 \in \mathcal{L},\ \mathbf{pairable}_t(l_1, l_2) \neq \perp \text{ iff } \mathbf{pairable}_g(\kappa(l_1), \kappa(l_2)) \neq \perp \qquad (3)$$

In this case, we say that $\mathbf{pairable}$ and $\mathbf{pairable}'$ are consistent under mapping κ.

4.5.1 Set-based GES

Set-based graded encodings [GGH13b, BR14, BGK+14b] consider a label set $\mathcal{L} \subseteq 2^U$ that consists of subsets of a universe set U. Zero testing can be done at the special label $l^\star = U$ that is exactly the universe set ($U \in \mathcal{L}$). Finally, for any two sets $s_0, s_1 \in \mathcal{L}$, they can be paired iff they are disjont; the output label is their union. More precisely,

$$\mathbf{pairable}_s(s_0, s_1) = \begin{cases} s_0 \cup s_1 & \text{if } s_0 \cap s_1 = \emptyset \\ \perp & \text{otherwise} \end{cases}$$

Implementing GES for Depth-D 4-ary Trees using Set-based GES To implement GES for depth-D 4-ary tree using set-based GES, we need to specify: 1) the universe set U_D, 2) a mapping κ_D that maps every label $l \in \{0,1,2,3\}^{\leq D}$ to a subset of U_D. Then we need to show that under κ_D, the pairable function $\mathbf{pairable}_s$ of the set-based GES and the pairable function $\mathbf{pairable}_t$ for depth-D 4-ary tree are consistent. We do so via induction over depth d. For all depth d, the universe U_d contains alphanumeric strings.

- *Base case when $d = 1$:* The universe $U_1 = \{a, b, c, d\}$. The mapping function κ_1 is defined as

  $$\kappa^1(\varepsilon) = U_1, \quad \kappa_1(0) = \{a, c\}, \quad \kappa_1(1) = \{b, d\}, \quad \kappa_1(2) = \{a, d\}, \quad \kappa_1(3) = \{b, c\}$$

  It is easy to verify that the sets corresponding to labels 0 and 1 are disjoint, as well as the sets to labels 2 and 3. But, the sets for any other pair of labels are not disjoint. Therefore, $\mathbf{pairable}_s$ and $\mathbf{pairable}_t$ are consistent under κ_1.

  We additionally keep the invariant that for every d, any element in the universe U_d appears at most in the sets of two distinct leaves $l_1, l_2 \in \{0,1,2,3\}^d$. This holds for $d = 1$.

- *Induction case from d to $d+1$:* The induction hypothesis states that for depth d, there is a universe U_d and mapping function κ_d from $\{0,1,2,3\}^{\leq d}$ to subsets of U_d, such that, $\mathbf{pairable}_t$ is consistent with $\mathbf{pairable}_s$ and the above invariant holds. We establish the same for depth $d+1$.

  Define the following $\mathbf{extend}$ function that on input an alphanumeric string A outputs a set of strings that appends to A different suffixes,

  $$\mathbf{extend}(A) = A||0||\{a, b, c, d\} \cup A||1||\{a, b, c, d\} \cup \ A||2||\{e, f, g, h\}||\{a, b, c, d\}$$

  where $S||\{a, b, c, d\} = \{Xa, Xb, Xc, Xd \ : \ X \in S\}$ is the set of strings obtained by appending every string in set S with one of the 4 letters a, b, c, d, and for a single sting $X||\{a, b, c, d\} = \{X\}||\{a, b, c, d\}$. Moreover, let $\mathbf{extend}(S)$ on input a set S outputs the union of the strings obtained by extending each string in S, that is, $\mathbf{extend}(S) = \cup_{A \in S} \mathbf{extend}(A)$. We observe that for any sets S_1, S_2, $\mathbf{extend}(S_1)$ and $\mathbf{extend}(S_2)$ are disjoint iff S_1 and S_2 are.

  The universe $U^{d+1} = \mathbf{extend}(U^d)$. The mapping function κ^{d+1} is defined as follows:

  - For $l \in \{0,1,2,3\}^{\leq d}$, $\kappa^{d+1}(l) = \mathbf{extend}(\kappa^d(l))$.

By the observation above, for any $l_1, l_2 \in \{0,1,2,3\}^{\leq d}$,

$$\mathbf{pairable}_s(\kappa^{d+1}(l_1), \kappa^{d+1}(l_2)) \neq \perp \text{ iff } \mathbf{pairable}_s(\kappa^{d}(l_1), \kappa^{d}(l_2)) \neq \perp \text{ iff } \mathbf{pairable}_t(l_1, l_2)$$

where the second iff follows from the induction hypothesis. Therefore, $\mathbf{pairable}_s$ and $\mathbf{pairable}_t$ are consistent under κ^{d+1} for these labels of length no longer than d.

- For $l \in \{0,1,2,3\}^{d+1}$, let $l = l'||x$ for $l' \in \{0,1,2,3\}^d$. Starting from the empty set, we grow the set $\kappa^{d+1}(l)$ by add multiple strings corresponding to each $A \in \kappa^d(l')$. For each $A \in \kappa^d(l')$, the strings we add are different in two cases: Recall that by the invariant, $A \in \kappa^d$ appears at most in the sets of two labels $l'_0 < l'_1$ of length d. If $l' = l'_0$ set the flag $\beta = 0$, and otherwise $\beta = 1$. (In the case A appears in only one set, set $\beta = 0$). We now specify the strings $B_1, \cdots B_6$ to be added corresponding to A depending on the flag β.
  - ∗ Add B_1, B_2 as

$$B_1, B_2 = \begin{cases} A\beta a, A\beta c & \text{If } x = 0 \\ A\beta b, A\beta d & \text{If } x = 1 \\ A\beta a, A\beta d & \text{If } x = 2 \\ A\beta b, A\beta c & \text{If } x = 3 \end{cases} \tag{4}$$

  - ∗ If $\beta = 0$, add $B_3 \cdots B_6$ as

$$B_3, B_4, B_5, B_6 = \begin{cases} (Ae, Ag) \times (a, c) & \text{If } x = 0 \\ (Ae, Ag) \times (b, d) & \text{If } x = 1 \\ (Af, Ah) \times (a, c) & \text{If } x = 2 \\ (Af, Ah) \times (b, d) & \text{If } x = 3 \end{cases} \tag{5}$$

    where $(A, B) \times (C, D)$ are the four strings $A||C$, $A||D$, $B||C$, $B||D$.
    On the other hand, if $\beta = 1$, add $B_3 \cdots B_6$ as

$$B_3, B_4, B_5, B_6 = \begin{cases} (Ae, Ah) \times (a, d) & \text{If } x = 0 \\ (Ag, Af) \times (b, c) & \text{If } x = 1 \\ (Ae, Ah) \times (a, d) & \text{If } x = 2 \\ (Ag, Af) \times (b, c) & \text{If } x = 3 \end{cases} \tag{6}$$

  Note that the above are all the strings related to A to be added to sets $\kappa^{d+1}(l'_0||x)$ and $\kappa^{d+1}(l'_1||x)$ for any $x \in \{0,1,2,3\}$. Observe that the invariant holds — every string appears at most twice.

It remains to show that for any l_0, l_1 such that one of them, say l_0, has length $d+1$, $\mathbf{pairable}_s$ and $\mathbf{pairable}_t$ are consistent under the above defined κ^{d+1}. We again consider cases:

- *Case 1: l_0 and l_1 are both children of $l' \in \{0,1,2,3\}^d$.* Locally among the four children of l', for every $A \in \kappa^d(l')$ with flag β, the strings B_0, B_1 added to different children according to equation (4) allows children 0 and 1, as well as 2 and 3, to be paired together, while preventing all other pairs of children from being paired (like in the base case). On the other hand, strings $B_3, \cdots, B_6$ in both case $\beta = 0$ and $\beta = 1$ still permits children 0 and 1, as well as 2 and 3 to be paired. Therefore, $\mathbf{pairable}_s$ and $\mathbf{pairable}_t$ are consistent locally for the 4 children of every node l' at layer d.

- *Case 2: If $l_0 = l'_0||x_0$ and $l_1 = l'_1||x_1$ for different $l'_0, l'_1 \in \{0,1,2,3\}^d$.* Under $\mathbf{pairable}_t$, l_0 and l_1 are pairable iff l'_0 and l'_1 are. We show that under $\mathbf{pairable}_s$, $\kappa^{d+1}(l_0)$ and $\kappa^{d+1}(l_1)$ are pairable iff $\kappa^d(l'_0)$ and $\kappa^d(l'_1)$ are. Then by the induction hypothesis, $\mathbf{pairable}_t$ and $\mathbf{pairable}_s$ are consistent for such l_0, l_1.

  First, if $\kappa^d(l'_0)$ and $\kappa^d(l'_1)$ are disjoint, then so are $\kappa^{d+1}(l_0)$ and $\kappa^{d+1}(l_1)$.

  It remains to show that if the former two sets are not disjoint, then neither are the latter two. Let A be a string in both $\kappa^d(l'_0)$ and $\kappa^d(l'_1)$. Assume w.l.o.g. that $l'_0 < l'_1$. Thus, $l'_0||x_0$ is associated with flag 0 and $l'_1||x_1$ with flag 1. No matter what x_0 and x_1 are, there is a letter $y \in \{e,f,g,h\}$ and a letter $z \in \{a,b,c,d\}$, such that, string Ayz is added to both $\kappa^{d+1}(l_0)$ and $\kappa^{d+1}(l_1)$, which are hence not disjoint.
- *Case 3: If $l_0 = l'_0||x_0$ and $l_1 = l'_1 \in \{0,1\}^{\leq d}$.* Under $\mathbf{pairable}_t$, l_0 and l_1 are pairable iff l'_0 and l'_1 are. By induction hypothesis, $\mathbf{pairable}_s(\kappa^d(l'_0), \kappa^d(l'_1)) = \perp$ iff $\mathbf{pairable}_t(l'_0, l'_1) = \perp$. It suffices to show that $\mathbf{pairable}_s(\kappa^{d+1}(l_0), \kappa^{d+1}(l_1)) = \perp$ iff $\mathbf{pairable}_s(\kappa^d(l'_0), \kappa^d(l'_1)) = \perp$. First, if $\kappa^d(l'_0)$ and $\kappa^d l'_1$ are disjoint, then so are $\kappa^{d+1}(l_0)$ and $\kappa^{d+1}(l_1)$. If $\kappa^d(l'_0)$ and $\kappa^d(l'_1)$ are not disjoint, let A be a string in both sets. Then all the strings extenasded from A in $\kappa^{d+1}(l_0)$ are contained in $\kappa^{d+1}(l_1)$. Thus they are not disjoint either.

In summary, we conclude that $\mathbf{pairable}_s$ and $\mathbf{pairable}_t$ are consistent under κ^{d+1}.

4.5.2 Graph-based GES

Graph-based graded encodings [GGH15] are associated with an underlying directed graph $G = (V, E)$. They have a label set $\mathcal{L}$ that consists of all paths $p = u \rightsquigarrow v$ in G, denoted as $\mathcal{L} = \mathsf{path}(G)$. Zero testing can be done at any path. Finally, for any two paths $p_0, p_1 \in \mathcal{L}$, they can be paired iff they are consecutive paths like $p_0 = u \rightsquigarrow v$ and $p_1 = v \rightsquigarrow w$ and the output label is the concatenation of the two paths $p = u \rightsquigarrow w$. More precisely,

$$\mathbf{pairable}_g(p_0, p_1) = \begin{cases} u \rightsquigarrow w & \text{if } \exists b,\ p_b = u \rightsquigarrow v,\ p_{1+b} = v \rightsquigarrow w \\ \perp & \text{otherwise} \end{cases}$$

We note that the graph-based GES scheme can be extended to work with the following more permissive $\mathbf{pairable}_g$ function, when encodings of 1 under every label is released.

$$\mathbf{pairable}_g(p_0, p_1) = \begin{cases} u \rightsquigarrow w & \text{if } \exists b,\ p_b = u \rightsquigarrow v,\ p_{1-b} = v' \rightsquigarrow w \text{ and } v \rightsquigarrow v' \in \mathsf{path}(G) \\ \perp & \text{otherwise} \end{cases}$$

Below, we work with this $\mathbf{pairable}_g$ function.

Implementing GES for Depth-D 4-ary Trees using Graph-based GES To implement GES for depth-D 4-ary tree using graph-based GES, we need to specify: 1) the underlying graph G^D, 2) a mapping κ^D that maps every lable $l \in \{0,1,2,3\}^{\leq D}$ to a path p in G^D. Then we need to show that under κ^D, the pairable function $\mathbf{pairable}_g$ of the graph-based GES and the pairable function $\mathbf{pairable}_t$ for depth-D 4-ary tree are consistent. We do so via induction over depth d.

- *Base case when $d = 1$:* The graph G^1 contains two paths $p_0 = s \to v_0 \to t$ and $p_1 = s \to v_1 \to t$. The mapping function κ^1 is defined as

$$\kappa^1(\varepsilon) = s \rightsquigarrow t, \quad \kappa^1(0) = s \to v_0, \quad \kappa^1(1) = v_0 \to t, \quad \kappa^1(2) = s \to v_1, \quad \kappa^1(3) = v_1 \to t$$

It is easy to verify that under mapping κ^1, $\mathbf{pairable}_t$ and $\mathbf{pairable}_g$ are consistent.

We additionally keep the invariant that for every d, labels l of length exactly d are mapped to a direct edge in G^d. This invariant clearly holds for $d = 1$.

- *Induction case from d to $d+1$:* The induction hypothesis states that for depth d, there is a graph G^d and mapping function κ^d from $\{0,1,2,3\}^{\leq d}$ to paths in G^d, such that, $\mathbf{pairable}_t$ is consistent with $\mathbf{pairable}_g$ and the above invariant holds. We want to establish the same for depth $d+1$.

  G^{d+1} is constructed by replacing every edge $u \to v$ in G^d with the graph G^1, that is, two new nodes ν_0 and ν_1 are added and edge $u \to v$ is replaced with edges $u \to \nu_0$, $\nu_0 \to v$, $u \to \nu_1$ and $\nu_1 \to v$. Note that every node and path in G^d is still a node and path in G^{d+1}, and if there is no path from u to v in G^d, there is no such path in G^{d+1} either.

  The new mapping function κ^{d+1} from $\{0,1,2,3\}^{\leq d+1}$ to paths in G^{d+1} are defined identically as κ^d on the sub-domain $\{0,1,2,3\}^{\leq d}$. We now define its output on labels in $\{0,1,2,3\}^{d+1}$. Let l be such a label of length d, and $\kappa^{d+1}(l) = \kappa^d(l) = u \to v$.

  $$\kappa^{d+1}(l||0) = u \to \nu_0, \quad \kappa^{d+1}(l||1) = \nu_0 \to v, \quad \kappa^{d+1}(l||2) = u \to \nu_1, \quad \kappa^{d+1}(l||3) = \nu_1 \to v$$

  Note that the above invarant holds w.r.t. κ^{d+1}.

  We show that $\mathbf{pairable}_t$ and $\mathbf{pairable}_g$ are consistent with each other through κ^{d+1}. Since κ^{d+1} and κ^d agrees on every label of length no larger than d, it suffices to argue that $\mathbf{pairable}_t$ and $\mathbf{pairable}_g$ are consistent for two labels l_1, l_2 where at least one of them has length $d+1$. Consider the following cases:

  - $|l_1| = d+1$ and $|l_2| \leq d$ (or the other way around). In this case, $l_1 = l_1'||c_1$ for some length-d label l_1', and $\mathbf{pairable}_t(l_1, l_2) = \mathbf{pairable}_t(l_1', l_2)$. Next we show that

    $$\mathbf{pairable}_g(\kappa^{d+1}(l_1), \kappa^{d+1}(l_2)) = \mathbf{pairable}_g(\kappa^d(l_1'), \kappa^d(l_2)) \neq \perp \text{ iff } \mathbf{pairable}_t(l_1', l_2) \neq \perp$$

    The iff follows from the induction hypothesis. To see the first equation, consider the example $c_1 = 0$. In this case, if $\kappa^d(l_1') = u \to v$, then $\kappa^{d+1}(l) = u \to \nu_0$. Let $(u', v') = \kappa^d(l_2') = \kappa^{d+1}(l_2)$. By construction of G^{d+1}, if $u \to v$ and $u' \rightsquigarrow v'$ are connected via some path $v \rightsquigarrow u'$ (or via some path $v' \rightsquigarrow u$) in G^d, then $u \to \nu_0$ and $u' \rightsquigarrow v'$ are connected via path $\nu_0 \to v \rightsquigarrow u'$ (or via $v' \rightsquigarrow u$) in G^{d+1}, and vice versa. Similarly, the first equation holds for any other value of c_1.
  - $|l_1| = |l_2| = d+1$. In this case, $l_1 = l_1'||c_1$ and $l_2 = l_2'||c_2$. 1) If $l_1' = l_2'$, it follows from the same analysis as in the base case that $\mathbf{pairable}_t$ and $\mathbf{pairable}_g$ are consistent for such l_1, l_2. 2) If $l_1' \neq l_2'$, it follows from similar analysis as above that $\mathbf{pairable}_g(\kappa^{d+1}(l_1), \kappa^{d+1}(l_2)) = \mathbf{pairable}_g(\kappa^d(l_1'), \kappa^d(l_2'))$. The latter equals to $\mathbf{pairable}_t(l_1', l_2')$ by the induction hypothesis, which in turn equals to $\mathbf{pairable}_t(l_1, l_2)$ in the setting for tree.

5 Function-Hiding Secret-Key IPE

In this section, we construct a function-hiding secret-key inner-product functional encryption (IPE) scheme. We first recall in Section 5.3 the IPE scheme proposed by Bishop, Jain and Kowalczyk (BJK) [BJK15] which is based on the SXDH assumption on asymmetric bilinear groups and satisfies a *weak notion of function-hiding*. We build on the BJK scheme in two important ways.

First, we require our scheme to be secure even when the randomness for generating the function keys and ciphertexts are "shared" between different instances of the IPE scheme. This strengthened notion called *multi-instance function hiding* (with shared randomness) is introduced in Section 5.4. We observe in Section 5.5 that the weak function-hiding property of the BJK scheme is robust to sharing randomness as required in the multi-instance setting.

Second, in Section 5.7, we generically (without making any other assumptions) bootstrap any weak function-hiding IPE scheme that is multi-instance secure to a (strong) function-hiding IPE scheme that is multi-instance secure. We remark that even setting aside the multi-instance aspect of the IPE scheme, our scheme gives a construction of function-hiding IPE secure under the SXDH assumption on asymmetric *bilinear* groups. As we described in the introduction, strong function-hiding IPE was previously constructed in the work of Datta, Dutta and Mukhopadhyay [DDM16]; however, their construction is complex and non-generic, whereas we show a general transformation from any weak function-hiding IPE scheme.

We start by introducing secret-key IPE in Section 5.1 and asymmetric bilinear groups in Section 5.2.

5.1 Secret-Key Inner Product Functional Encryption

A secret-key inner product functional encryption (SKIPE) scheme for length-N inner products is a secret key FE scheme for the function class of computing inner products of length-N vectors over some ring $\mathcal{R}$. In this work, we consider a slight generalization, and allow the setup algorithm of the secret key IPE scheme to receive as input some public parameter pp, which in particular specifies the ring $\mathcal{R}$ over which inner products are computed. This generalization allows us to flexibly consider running the same scheme with different public parameters. For example, both the BJK IPE scheme and our secret key IPE scheme use a public parameter pp that describes a pair of asymmetric bilinear groups of prime order p, and computes inner products over $\mathcal{R} = \mathbb{Z}_p$. Due to our generalization, we can consider instantiating these schemes with different distributions of asymmetric bilinear groups. In particular, in our FE construction later, we will instantiate them with graded encodings with different labels that behave like asymmetric bilinear groups.

Syntax Let $\mathcal{G}$ be a public parameter generator and $N = N(\lambda)$. A secret-key inner product functional encryption (IPE) scheme skIPE for length-N inner products consists of four PPT algorithms $(\mathsf{skIPE.Setup}, \mathsf{skIPE.KeyGen}, \mathsf{skIPE.Enc}, \mathsf{skIPE.Dec})$. When instantiated with $\mathcal{G}$, the algorithms proceed as follows:

- *Setup:* $\mathsf{skIPE.Setup}(1^\lambda, \mathsf{pp})$ is an algorithm that on input a security parameter 1^λ and a public parameter pp in the support of $\mathcal{G}(1^\lambda)$ outputs a master secret key msk. pp contains the description of a ring $\mathcal{R}$.

- *Key Generation:* $\mathsf{skIPE.KeyGen}(\mathsf{msk}, \mathbf{y})$ on input the master secret key and the description of a vector $\mathbf{y} \in \mathcal{R}^N$, outputs a secret key sk.

- *Encryption:* $\mathsf{skIPE.Enc}(\mathsf{msk}, \mathbf{x})$ on input the master secret key and a message $\mathbf{x} \in \mathcal{R}^N$, outputs an encryption ct of $\mathbf{x}$.

- *Decryption:* $\mathsf{skIPE.Dec}(\mathsf{sk}, \mathsf{ct})$ computes $\mathbf{x} \cdot \mathbf{y}$ over $\mathcal{R}$.

Correctness: We define perfect correctness of decryption here, although our definition can be generalized to allow a negligible decryption error. For every λ, every pp, and every $\mathbf{x}, \mathbf{y} \in \mathcal{R}^N$, the following holds

$$\Pr\left[\begin{array}{c} \mathsf{msk} \xleftarrow{\$} \mathsf{skIPE.Setup}(1^\lambda, \mathsf{pp}) \\ \mathsf{ct} \xleftarrow{\$} \mathsf{skIPE.Enc}(\mathsf{msk}, \mathbf{x}) \\ \mathsf{sk} \xleftarrow{\$} \mathsf{skIPE.KeyGen}(\mathsf{msk}, \mathbf{y}) \end{array} : \mathsf{skIPE.Dec}(\mathsf{sk}, \mathsf{ct}) = \mathbf{x} \cdot \mathbf{y} \text{ over } \mathcal{R}\right] = 1$$

5.1.1 Function Hiding and Weak Function Hiding

In the secret-key setting, researchers have considered a very strong security property of functional encryption, namely input- and function-hiding. The combination of input- and function-hiding provides privacy guarantees not only for the encrypted inputs but also for the encoded functions. More specifically, in the setting of inner product encryption, the security definition requires that the function keys and ciphertexts of vectors $\{\mathbf{y}_j^0\}$ and $\{\mathbf{x}_i^0\}$ are indistinguishable from that of $\{\mathbf{y}_j^1\}$ and $\{\mathbf{x}_i^1\}$, as long as the inner products of every input and function pair are identical. Namely,

$$\mathbf{x}_i^0 \cdot \mathbf{y}_j^0 = \mathbf{x}_i^1 \cdot \mathbf{y}_j^1$$

Definition 15 (Function Hiding for Secret Key IPE). *Let $\mathcal{G}$ be a public parameter generator and N a polynomial. We say that a secret key IPE* $\mathsf{skIPE} = (\mathsf{skIPE.Setup}, \mathsf{skIPE.KeyGen}, \mathsf{skIPE.Enc}, \mathsf{skIPE.Dec})$ *for computing length-N inner products is μ-*function hiding *w.r.t. $\mathcal{G}$, if for every* PPT *adversary A, and every sufficiently large security parameter $\lambda \in \mathbb{N}$, the adversary's advantage in the following games is bounded by $\mu(\lambda)$:*

$$\mathsf{Advt}_{A,\mathcal{G}}^{\mathsf{skIPE}} = \left|\Pr[\mathsf{FH.Exp}_{A,\mathcal{G}}^{\mathsf{skIPE}}(1^\lambda, 0) = 1] - \Pr[\mathsf{FH.Exp}_{A,\mathcal{G}}^{\mathsf{skIPE}}(1^\lambda, 1) = 1]\right| \leq \mu(\lambda)$$

The game $\mathsf{FH.Exp}_{A,\mathcal{G}}^{\mathsf{skIPE}}(1^\lambda, b)$ *proceeds as follows:*

- **Public Parameter and Key Generation.** *The challenger CH samples a public parameter* $\mathsf{pp} \xleftarrow{\$} \mathcal{G}$, *which contains the description of a ring $\mathcal{R}$, and a master secret key* $\mathsf{msk} \xleftarrow{\$} \mathsf{skIPE.Setup}(1^\lambda, \mathsf{pp})$.
- **Function and Message Queries.** *Repeat the following for an arbitrary number of times determined by A: In iteration i,*
  - *Upon A choosing a pair of challenge functions $\mathbf{y}_i^0, \mathbf{y}_i^1 \in \mathcal{R}^{N(\lambda)}$. CH sends A a function key* $\mathsf{sk}_i \xleftarrow{\$} \mathsf{skIPE.KeyGen}(\mathsf{msk}, \mathbf{y}_i^b)$.
  - *Upon A choosing a pair of challenge messages $\mathbf{x}_i^0, \mathbf{x}_i^1 \in \mathcal{R}^{N(\lambda)}$. CH sends A a ciphertext* $\mathsf{ct}_i \xleftarrow{\$} \mathsf{skIPE.Enc}(\mathsf{msk}, \mathbf{x}_i^b)$.
- *Finally A outputs a bit b'.*

Restriction R: *Every message query $\mathbf{x}_i^0, \mathbf{x}_i^1$ and every function query $\mathbf{y}_j^0, \mathbf{y}_j^1$ must satisfy that $\mathbf{x}_i^0 \cdot \mathbf{y}_j^0 = \mathbf{x}_i^1 \cdot \mathbf{y}_j^1$.*

We call the above definition of security function-hiding security, or sometimes strong function-hiding security, to distinguish from the weaker security guarantee below.

Definition 16 (Weak Function Hiding for Secret Key IPE)**.** *We say that a secret key IPE* skIPE *for computing length-N inner products is μ-*weak function hiding *w.r.t. $\mathcal{G}$, if the condition in Definition 15 holds for a modified experiment* $\mathsf{wFH.Exp}^{\mathsf{skIPE}}_{A,\mathcal{G}}(1^\lambda, b)$ *which is identical to* $\mathsf{FH.Exp}^{\mathsf{skIPE}}_{A,\mathcal{G}}(1^\lambda, b)$ *except that the restriction $\mathbf{R}$ is replaced with the restriction $\mathbf{R}'$ below.*

Restriction $\mathbf{R}'$: *Every message query $\mathbf{x}_i^0, \mathbf{x}_i^1$ and every function query $\mathbf{y}_j^0, \mathbf{y}_j^1$ must satisfy that $\mathbf{x}_i^0 \cdot \mathbf{y}_j^0 = \mathbf{x}_i^0 \cdot \mathbf{y}_j^1 = \mathbf{x}_i^1 \cdot \mathbf{y}_j^1$.*

5.2 Asymmetric Bilinear Groups

Let $\mathcal{G}$ denote a group generator that on input 1^λ outputs $(p, G_0, G_1, G_T, \otimes)$, where G_0, G_1, G_T are groups with prime order p. G_0 and G_1 are referred to as the source groups and G_T as the target group. Assume without loss of generality that the description of the source groups contain generators g_0, g_1 of G_0, G_1. In addition, the following properties hold.

- *Admissible:* $\otimes : G_0 \times G_1 \to G_T$ is efficiently computable and $g_T = g_0 \otimes g_1$ generates G_T.
- *Bilinear:* For any $a, b \in \mathbb{Z}_p$, $g_0^a \otimes g_1^b = (g_0 \otimes g_1)^{ab} = g_T^{ab}$.

The Bracket Notation To be consistent with the notations for graded encodings later, we use the following bracket notations to denote group elements.

$$[a]_0 = g_0^a, \quad [b]_1 = g_0^b, \quad [c]_T = g_T^c$$

Under this notation, the generators in group $x \in \{0, 1, T\}$ is represented as $[1]_x = g_x$. We also use the following vector notation to represent vectors of group elements succinctly: For any $\mathbf{v} = (v_1, \cdots, v_m) \in \mathbb{Z}_p^m$, and $x \in \{0, 1, T\}$:

$$[\mathbf{v}]_x = [v_1]_x \cdots [v_m]_x$$

Homomorphic Operations Informally, $[a]_x$ can be viewed as an encoding of a in group G_x. Then, using multiplication and exponentiation in each group, we can perform addition "$\oplus$" and scalor multiplication "$\odot$" to vectors encoded in the same group x. Formally, for any $\mathbf{v}, \mathbf{w} \in \mathbb{Z}_p^m$, and $\alpha \in \mathbb{Z}_p$,

$$[\mathbf{v}]_x \oplus [\mathbf{w}]_x := [\mathbf{v} + \mathbf{w}]_x = ([v_1 + w_1]_x \cdots [v_m + w_m]_x)$$
$$\alpha \odot [\mathbf{v}]_x := ([\mathbf{v}]_x)^\alpha = [\alpha \mathbf{v}]_x = ([\alpha v_1]_x \cdots [\alpha v_m]_x)$$

In particular, this means we can homomorphically evaluate any linear function L in $\mathbb{Z}_p$, over encoded vectors. We conveniently write

$$L([\mathbf{v}]_x) = [L(\mathbf{v})]_x$$

Using pairing, we can compute inner product "$\cdot$" of two vectors encoded in different source groups.

$$[\mathbf{v}]_0 \cdot [\mathbf{w}]_1 := \bigoplus_{i \in [m]} [v_i]_0 \otimes [w_i]_1 = [\mathbf{v} \cdot \mathbf{w}]_T$$

The SXDH Assumption The SXDH assumption states that the standard DDH assumption holds in each of the source groups. Formally, for each source group G_x for $x \in \{0,1\}$, the following two ensembles are μ-indistinguishable.

$$\left\{(p, G_0, G_1, G_T, \otimes) \overset{\$}{\leftarrow} \mathcal{G}(1^\lambda),\ a, b \overset{\$}{\leftarrow} \mathbb{Z}_p\ :\ (p, G_0, G_1, G_T, \otimes),\ [a]_x, [b]_x, [ab]_x\right\}_\lambda$$
$$\left\{(p, G_0, G_1, G_T, \otimes) \overset{\$}{\leftarrow} \mathcal{G}(1^\lambda),\ a, b, r \overset{\$}{\leftarrow} \mathbb{Z}_p\ :\ (p, G_0, G_1, G_T, \otimes),\ [a]_x, [b]_x, [r]_x\right\}_\lambda$$

5.3 BJK Weak Function Hiding Secret Key IPE

Bishop, Jain and Kowalczyk constructed a secret key IPE with the weak function hiding property using asymmetric bilinear groups in which the SXDH assumption holds. We recall their scheme.

Dual Pairing Vector Space The BJK construction employs the concept of dual pairing vector space of Okamoto and Takashima [OT08, OT09]. The algorithm $\mathrm{Dual}(\mathbb{Z}_p^m)$ samples two random sets of vectors $B = \{\mathbf{b}_1, \cdots \mathbf{b}_m\}$ and $B^\star = \{\mathbf{b}_1^\star \cdots \mathbf{b}_m^\star\}$ of $\mathbb{Z}_p^m$, subject to the constraint that they are "dual orthonormal" meaning that $B^\star = (B^T)^{-1}$, that is,

$$\mathbf{b}_i \cdot \mathbf{b}_i^\star = 1 \pmod p \text{ for all } i$$
$$\mathbf{b}_i \cdot \mathbf{b}_j^\star = 0 \pmod p \text{ for all } i \neq j$$

For convenience, we will denote by $B_1 = [\mathbf{b}_1 \| \cdots \| \mathbf{b}_{m/2}]$ and $B_2 = [\mathbf{b}_{m/2+1} \| \cdots \| \mathbf{b}_m]$ in $\mathbb{Z}_p^{m \times (m/2)}$ the first and the second halves of columns in B, when m is even.

Overview of the BJK Scheme Fix any input length polynomial $N = N(\lambda)$. The BJK scheme for computing length-N inner products, which we will denote as $\overline{\mathsf{skIPE}}^N$, treats the plaintext vector $\mathbf{x}$ and the key vector $\mathbf{y}$ completely symmetrically. To achieve weak function hiding, they encode $\mathbf{x}$ in source group G_0 and $\mathbf{y}$ in source group G_1. The inner product can then be computed using pairing between G_0 and G_1. The scheme cannot directly publish $[\mathbf{x}]_0$ and $[\mathbf{y}]_1$, since the SXDH assumption does not imply that encoded (non-random) vectors would be hidden. Instead, $\mathbf{x}, \mathbf{y}$ are first "embedded" into dual orthonormal spaces, like $B^\star\mathbf{x}$ and $B\mathbf{y}$, before encoding. The dual orthonormal spaces are sampled at random, and hence the embeddings are random, subject to preserving the inner product $\mathbf{x} \cdot \mathbf{y}$. For the proof to go through, another key ingredient in the scheme is embedding the vectors twice and additionally masking the embedding with random scalars. More specifically, the scheme samples $B, B^\star \overset{\$}{\leftarrow} \mathrm{Dual}(\mathbb{Z}_p^{2m})$ and $\alpha, \alpha^\star, \beta, \beta^\star \overset{\$}{\leftarrow} \mathbb{Z}_p$, and embeds $\mathbf{x}, \mathbf{y}$ as follows:

$$\mathbf{v}_x = \alpha^\star B_1^\star \mathbf{x} + \beta^\star B_2^\star \mathbf{x} \qquad \mathbf{v}_y = \alpha B_1 \mathbf{y} + \beta B_2 \mathbf{y}$$
$$\text{Clearly, } \mathbf{v}_x \cdot \mathbf{v}_y = (\alpha\alpha^\star + \beta\beta^\star)\mathbf{x} \cdot \mathbf{y}$$

(To remove the effect of $\theta = (\alpha\alpha^\star + \beta\beta^\star)$ from the final output, the scheme separately embeds the scalars in smaller dual orthonormal spaces $D, D^\star \overset{\$}{\leftarrow} \mathrm{Dual}(\mathbb{Z}_p^2)$.) We refer to the first embedding using $B_1, B_1^\star$ as the first slot, and that using $B_2, B_2^\star$ the second slot. The use of double embedding and random scalar masks provide enough ambiguity, so that, in a hybrid security proof, $\mathbf{v}_x$ (or $\mathbf{v}_y$) can be information theoretically "repurposed" to embed a different $x^\star$ in one of the two slots, w.r.t. a different dual orthonormal space $F, F^\star$. This allows us to gradually change the embedded vectors in different slots of different function keys and ciphertexts one by one in the hybrid security proof.

Because of the importance of this double embedding, we will explicitly write it as:

$$\begin{gathered}\mathsf{dE}(B, \mathbf{v}; \alpha, \beta) = \alpha B_1 \mathbf{v} + \beta B_2 \mathbf{v} \text{ , and} \\ \mathsf{dE}(B, \mathbf{v}; \alpha, \beta) \cdot \mathsf{dE}(B^\star, \mathbf{w}; \alpha^\star, \beta^\star) = \theta(\mathbf{v} \cdot \mathbf{w}) \text{ , where } \theta = \alpha\alpha^\star + \beta\beta^\star\end{gathered}$$

Remark 1 (Linearity of the Double Embedding Function)**.** *We remark that the double embedding function* dE *is lineally homomorphic in the sense that, for any* $\mathbf{w}_1 = \mathsf{dE}(B, \mathbf{v}_1; \alpha, \beta)$ *and* $\mathbf{w}_2 = \mathsf{dE}(B, \mathbf{v}_2; \alpha, \beta)$*, the following holds:*

$$\begin{gathered}\mathbf{w}_1 + \mathbf{w}_2 = \mathsf{dE}(B, \mathbf{v}_1 + \mathbf{v}_2; \alpha, \beta) \\ \gamma \mathbf{w}_1 = \mathsf{dE}(B, \gamma \mathbf{v}_1; \alpha, \beta)\end{gathered}$$

This linearity will be very useful later when constructing our slotted public-key IPE scheme.

The BJK Schemes We now formally describe the BJK scheme $\overline{\mathsf{skIPE}}^N$ for computing length-N inner products. Let $\mathsf{pp} = (p, G_0, G_1, G_T, \otimes)$ be the public parameters describing asymmetric bilinear groups with order p.

- *Setup:* $\overline{\mathsf{skIPE}}.\mathsf{Setup}(1^\lambda, \mathsf{pp})$ samples $(B, B^\star) \xleftarrow{\$} \mathrm{Dual}(\mathbb{Z}_p^{2N})$ and $(D, D^\star) \xleftarrow{\$} \mathrm{Dual}(\mathbb{Z}_p^2)$. It sets $\mathsf{msk} = (B, B^\star, D, D^\star)$.
- *Key Generation:* $\overline{\mathsf{skIPE}}.\mathsf{KeyGen}(\mathsf{msk}, \mathbf{y})$ on input the master secret key $\mathsf{msk} = (B, B^\star, D, D^\star)$ and vector $\mathbf{y} \in \mathbb{Z}_p^N$. It samples two random scalars $\alpha, \beta \xleftarrow{\$} \mathbb{Z}_p$, and then computes

$$\begin{gathered}\mathsf{sk}[1] = [\mathbf{c}]_1 = [\mathsf{dE}(B, \mathbf{y}; \alpha, \beta)]_1 \\ \mathsf{sk}[2] = [\mathbf{e}]_1 = [\mathsf{dE}(D, 1; \alpha, \beta)]_1\end{gathered}$$

  It outputs $\mathsf{sk} = [\mathbf{c}]_1, [\mathbf{e}]_1$.

- *Encryption:* $\overline{\mathsf{skIPE}}.\mathsf{Enc}(\mathsf{msk}, \mathbf{x})$ on input the master secret key $\mathsf{msk} = (B, B^\star, D, D^\star)$ and vector $\mathbf{x} \in \mathbb{Z}_p^m$, samples two random scalars $\alpha^\star, \beta^\star \xleftarrow{\$} \mathbb{Z}_p$, and then computes

$$\begin{gathered}\mathsf{ct}[1] = [\mathbf{c}^\star]_0 = [\mathsf{dE}(B^\star, \mathbf{x}; \alpha^\star, \beta^\star)]_0 \\ \mathsf{ct}[2] = [\mathbf{e}^\star]_0 = [\mathsf{dE}(D^\star, 1; \alpha^\star, \beta^\star)]_0\end{gathered}$$

  It outputs $\mathsf{ct} = [\mathbf{c}^\star]_0, [\mathbf{e}^\star]_0$.

- *Decryption:* $\overline{\mathsf{skIPE}}.\mathsf{Dec}(\mathsf{sk}, \mathsf{ct})$ on input $\mathsf{sk} = [\mathbf{c}]_1, [\mathbf{e}]_1$ and $\mathsf{ct} = [\mathbf{c}^\star]_0, [\mathbf{e}^\star]_0$ computes

$$\begin{aligned}[\mathbf{e}^\star]_0 \otimes [\mathbf{e}]_1 &= [\alpha\alpha^\star + \beta\beta^\star]_T = [\theta]_T \\ [\mathbf{c}^\star]_0 \otimes [\mathbf{c}]_1 &= [\mathsf{dE}(B^\star, \mathbf{x}; \alpha^\star, \beta^\star) \cdot \mathsf{dE}(B, \mathbf{y}; \alpha, \beta)]_T \\ &= [(\alpha\alpha^\star + \beta\beta^\star)(\mathbf{x} \cdot \mathbf{y})]_T \\ &= [\theta(\mathbf{x} \cdot \mathbf{y})]_T\end{aligned}$$

  If $z = \mathbf{x} \cdot \mathbf{y}$ is bounded by some polynomial bound $\Gamma = \Gamma(\lambda)$. Then z can be extracted through trying all possible values $i \in [\Gamma]$, and outputting the value i satisfying

$$[\theta(\mathbf{x} \cdot \mathbf{y})]_T = i \odot [\theta]_T$$

It is easy to verify that the BJK scheme is correct for computing inner products whose value fall into a polynomial range.

Weak Function Hiding In [BJK15], it is shown that their scheme guarantees the indistinguishability of experiments $\mathsf{FH.Exp}(1^\lambda, 0)$ and $\mathsf{FH.Exp}(1^\lambda, 1)$ if the adversaries choose only message and function queries $\{(\mathbf{x}_i^0, \mathbf{x}_i^1)\}$ and $\{(\mathbf{y}_j^0, \mathbf{y}_j^1)\}$, such that,

$$\mathbf{x}_i^0 \cdot \mathbf{y}_j^0 = \underline{\mathbf{x}_i^0 \cdot \mathbf{y}_j^1} = \underline{\mathbf{x}_i^1 \cdot \mathbf{y}_j^0} = \mathbf{x}_i^1 \cdot \mathbf{y}_j^1$$

This restriction is even more stringent than the restriction in our weak function hiding definition. However, a careful examination of their security proof reveals that either the second or the third equation (they are symmetric) is redundant (namely, their security proof goes through without it). In other words, the BJK scheme satisfies the weak function-hiding security definition.

Lemma 3 ([BJK15])**.** *Assume that the SXDH assumption holds in the asymmetric bilinear groups generated by $\mathcal{G}$. Then, for any polynomial N, the BJK secret key IPE scheme $\overline{\mathsf{skIPE}}^N$ is weak function hiding w.r.t. $\mathcal{G}$.*

Proof Idea. The proof of this lemma appears in the work of Bishop, Jain and Kowalczyk [BJK15]. Here, we describe the intuition behind the proof, highlighting the reason why they only achieve weak function-hiding.

The BJK proof strategy uses two sequence of hybrids to show the indistinguishability of the two experiments, $\mathsf{wFH.Exp}^{\mathsf{IPE}}_{A,\mathcal{G}}(1^\lambda, 0)$ and $\mathsf{wFH.Exp}^{\mathsf{IPE}}_{A,\mathcal{G}}(1^\lambda, 1)$, where the former generates function keys and ciphertexts for vectors $\{\mathbf{y}_j^0\}$ and $\{\mathbf{x}_i^0\}$, and the latter generates keys and ciphertexts for $\{\mathbf{y}_j^1\}$ and $\{\mathbf{x}_i^1\}$. The first sequence of hybrids switches the ciphertexts from encrypting $\{\mathbf{x}_i^0\}$ to encrypting $\{\mathbf{x}_i^1\}$ one by one, while keeping the function keys always the same, encoding vectors $\{\mathbf{y}_j^0\}$. The indistinguishability of neighboring hybrids in this sequence relies on the fact that

$$\mathbf{x}_i^0 \cdot \mathbf{y}_j^0 = \mathbf{x}_i^1 \cdot \mathbf{y}_j^0 \tag{7}$$

for all i, j. (In fact, $\mathbf{y}_j^1$ never appears in these hybrids and cannot play any role in their indistinguishbability proof.)

Following the first sequence, the second sequence switches the function keys from encoding $\{\mathbf{y}_j^0\}$ to encoding $\{\mathbf{y}_j^1\}$, while keeping the ciphertexts the same, encrypting $\{\mathbf{x}_i^1\}$. For the same reason, the indistinguishability of neighboring hybrids in the second sequence only relies on the fact that

$$\mathbf{x}_i^1 \cdot \mathbf{y}_j^0 = \mathbf{x}_i^1 \cdot \mathbf{y}_j^1 \tag{8}$$

for all i, j. Therefore, the indistinguishability of $\mathsf{wFH.Exp}^{\mathsf{IPE}}_{A,\mathcal{G}}(1^\lambda, b)$ for $b = 0, 1$ holds as long as $\mathbf{x}_i^0 \cdot \mathbf{y}_j^0 = \mathbf{x}_i^1 \cdot \mathbf{y}_j^0 = \mathbf{x}_i^1 \cdot \mathbf{y}_j^1$. In other words, the BJK scheme satisfies weak function hiding. Indeed, this particular way of proving security seems to require both conditions 7 and 8 above, resulting in weak function-hiding.

We will show in the sequel a general transformation that takes any weak function-hiding scheme (in particular, the BJK scheme) for length-$2N$ inner products and converts it into a strong function-hiding scheme for length-N inner products.

For completeness, we give the full proof of security of the [BJK15] scheme in Appendix **??**.

5.4 Multi-Instance Function Hiding

In this work, as we noted above, we need (and will construct) secret key IPE schemes with the full function hiding property, as opposed to weak function hiding achieved by the BJK scheme. Furthermore, we strengthen the security property to consider *multi-instance function hiding*. By an instance of the IPE scheme, we refer to a master secret key generated w.r.t. some public parameter, and the function keys and ciphertexts generated using this master secret key. We require function hiding to hold even when an adversary sees the functional keys and ciphertexts of multiple instances. Clearly, if all instances are independent, then multi-instance function hiding follows from the function hiding of each instance via a simple hybrid argument. We here consider a two-fold correlation between different instances: First, the function keys and ciphertexts of different instances are generated using shared randomness. Second, different instances are w.r.t. public parameters sampled jointly by a multi-parameter generator. These two features will be important for our construction of FE from graded encoding schemes later.

Multi-Parameter Generator: Let $\mathcal{MG}$ denote a multi-parameter generator that on input 1^λ outputs a set of polynomial number $m = m(\lambda)$ of public parameters $\{\mathsf{pp}_i\}_{i \in [m]}$. We consider the restricted case where all public parameters specify the same ring $\mathcal{R}$ for computing inner products. An example of such a generator we consider is a multi-group generator that samples parameters $\mathsf{pp}_i = (p, G_{0,i}, G_{1,i}, G_{T,i}, \otimes_i)$ describing asymmetric bilinear groups where $\otimes_i : G_{0,i} \times G_{1,i} \to G_{T,i}$.

Definition 17 (Multi-Instance Function Hiding). *Let $\mathcal{MG}$ be a multi-parameter generator, and $N = N(\lambda)$ and $m = m(\lambda)$ be polynomial functions. We say that a secret key IPE* skIPE *for computing length-N inner products is m-*multi-instance *μ-*function hiding *w.r.t. $\mathcal{MG}$, if for every* PPT *adversary A, and every sufficiently large security parameter $\lambda \in \mathbb{N}$, the adversary's advantage in the following games is bounded by $\mu(\lambda)$:*

$$\mathsf{Advt}^{\mathsf{skIPE}}_{A,\mathcal{MG}} = \left| \Pr[\mathsf{MFH}^{\mathsf{skIPE}}_{A,\mathcal{MG}}(1^\lambda, 0) = 1] - \Pr[\mathsf{MFH}^{\mathsf{skIPE}}_{A,\mathcal{MG}}(1^\lambda, 1) = 1] \right| \leq \mu(\lambda)$$

The experiment $\mathsf{MFH}^{\mathsf{skIPE}}_{A,\mathcal{MG}}(1^\lambda, b)$ proceeds as follows:

- **Public Parameter and Key Generation.** *Public parameters $(\mathsf{pp}_1, \cdots, \mathsf{pp}_m) \xleftarrow{\$} \mathcal{MG}(1^\lambda)$ are sampled, which contain the description of a ring $\mathcal{R}$. The challenger CH samples master secret keys $\{\mathsf{msk}_k \xleftarrow{\$} \mathsf{skIPE.Setup}(1^\lambda, \mathsf{pp}_k)\}_{k \in [m]}$.*
- **Function and Message Queries.** *Interact with A in an arbitrary number of iterations, where in each iteration, A has four options.*

  Below every $r^\star_i$ and r_j are random coins shared across the generation of multiple function keys and ciphertexts, sampled independently and randomly by CH.

  - *Upon A sending message query* $(\mathsf{mesg}\ (\mathbf{x}^0_{k,i}$ *or* $\mathbf{x}^1_{k,i}), k, i)$ *for $k \leq m$, CH sends A ciphertexts* $\mathsf{ct}_{k,i} \xleftarrow{\$} \mathsf{skIPE.Enc}(\mathsf{msk}_k, \mathbf{x}^b_{k,i};\ r^\star_i)$.
  - *Upon A sending function query* $(\mathsf{func}\ (\mathbf{y}^0_{k,j}$ *or* $\mathbf{y}^1_{k,j}), k, j)$ *for $k \leq m$, CH sends A function keys* $\mathsf{sk}_{k,i} \xleftarrow{\$} \mathsf{skIPE.KeyGen}(\mathsf{msk}_k, \mathbf{y}^b_{k,i};\ r_i)$.
  - *Upon A sending auxiliary message query* $(\mathsf{aux\text{-}mesg}\ \{\mathbf{x}_{l,k,i}\}_l, k, i)$ *for $k > m$, CH sends A ciphertexts* $\{\mathsf{ct}_{l,k,i} \xleftarrow{\$} \mathsf{skIPE.Enc}(\mathsf{msk}_k, \mathbf{x}_{l,k,i};\ r^\star_i)\}_l$.
  - *Upon A sending auxiliary function query* $(\mathsf{aux\text{-}func}\ \{\mathbf{y}_{1,k,j}\}_l, k, j)$ *for $k > m$, CH sends A function keys* $\{\mathsf{sk}_{l,k,j} \xleftarrow{\$} \mathsf{skIPE.KeyGen}(\mathsf{msk}_k, \mathbf{y}_{l,k,j};\ r_j)\}_l$.

  For every k, i, j, queries of form $(\star, k, i)$ can only be submitted once.

- *Finally A outputs a bit b'.*

Restriction: *For every $k \in [m]$, challenge message query $\mathbf{x}^0_{k,i}, \mathbf{x}^1_{k,i}$ and function query $\mathbf{y}^0_{k,j}, \mathbf{y}^1_{k,j}$ must satisfy that $\mathbf{x}^0_{k,i} \cdot \mathbf{y}^0_{k,j} = \mathbf{x}^1_{k,i} \cdot \mathbf{y}^1_{k,j}$.*

Note that the in the above definition, there are two types of quries, the challenge ones (mesg $(\mathbf{x}^0_{k,i}$ or $\mathbf{x}^1_{k,i}), k, i)$ and (func $(\mathbf{y}^0_{k,j}$ or $\mathbf{y}^1_{k,j}), k, j)$, and the auxilary ones ($\mathsf{aux\text{-}mesg}$ $\{\mathbf{x}_{l,k,i}\}_l, k, i)$ and ($\mathsf{aux\text{-}func}$ $\{\mathbf{y}_{1,k,j}\}_l, k, j)$. For the challenge queries, the information which message or function vectors are encoded must be hidden. In this case, the same randomness $r^\star_i, r_i$ can not be reused within the same instance. On the other hand, for the auxiliary queries, no hiding guarantees are required and $r^\star_i, r_i$ can be reused even within the same instance.

5.5 Multi-Instance Weak Function Hiding

We can similarly consider weak function hiding in the multi-instance setting.

Definition 18 (Multi-Instance Weak Function Hiding)**.** *Let $\mathcal{MG}$ be a multi-parameter generator and N a polynomial. We say that a secret key IPE $\overline{\mathsf{skIPE}}$ for computing length-N inner products is μ-*multi-instance weak function hiding *w.r.t. $\mathcal{MG}$ if the condition in Definition 17 holds with a more stringent restriction.*

- *Restriction: For every $k \in [m]$, every challenge message query $\mathbf{x}^0_{k,i}, \mathbf{x}^1_{k,i}$ and challenge function query $(\mathbf{y}^0_{k,j}, \mathbf{y}^1_{k,j})$ must satisfy that $\mathbf{x}^0_{k,i} \cdot \mathbf{y}^0_{k,j} = \mathbf{x}^1_{k,i} \cdot \mathbf{y}^0_{k,j} = \mathbf{x}^1_{k,i} \cdot \mathbf{y}^1_{k,j}$.*

As discussed above, the BJK scheme is function hiding w.r.t. a $\mathcal{G}$ that samples asymmetric bilinear groups where SXDH holds. In the multi-instance setting, when different instances share the random coins for generating function keys and ciphertexts, we show that the BJK scheme is multi-instance function hiding w.r.t. any multi-group generator $\mathcal{MG}$ that genenrates many asymmetric bilinear groups where SXDH holds jointly, formalized below.

The Joint-SXDH Assumption Consider $(\mathsf{pp}_1, \cdots, \mathsf{pp}_m) \xleftarrow{\$} \mathcal{MG}(1^\lambda)$ generated by $\mathcal{MG}$, where $\mathsf{pp}_i = (p, G_{i,0}, G_{i,1}, G_{i,T}, \mathbf{pair}_i)$ describes the i^{th} pair of asymmetric bilinear groups. Like before, we denote elements in $G_{i,x}$ using the braket notation $[a]_{ix}$. The joint-SXDH assumption states that for any x, encodings of the same DDH tuple a, b, ab in all x^{th} source group $\{G_{i,x}\}$ are indistinguishable from that of a, b, r. Formally, for each $x \in \{0, 1\}$, the following two ensembles are μ-indistinguishable.

$$\Big\{(\mathsf{pp}_1, \cdots, \mathsf{pp}_m) \xleftarrow{\$} \mathcal{MG}(1^\lambda),\ a, b \xleftarrow{\$} \mathbb{Z}_p\ :\ (\mathsf{pp}_1, \cdots, \mathsf{pp}_m),\ \underline{\{[a]_{ix}, [b]_{ix}, [ab]_{ix}\}_{i\in[m]}}\Big\}_\lambda$$

$$\Big\{(\mathsf{pp}_1, \cdots, \mathsf{pp}_m) \xleftarrow{\$} \mathcal{MG}(1^\lambda),\ a, b, r \xleftarrow{\$} \mathbb{Z}_p\ :\ (\mathsf{pp}_1, \cdots, \mathsf{pp}_m),\ \underline{\{[a]_{ix}, [b]_{ix}, [r]_{ix}\}_{i\in[m]}}\Big\}_\lambda$$

When $m = 1$, the joint-SXDH assumption is exactly the SXDH assumption.

5.6 BJK is Multi-Instance Weak Function Hiding

Lemma 4. *Assume that the joint-SXDH assumption holds in the asymmetric bilinear groups generated by $\mathcal{MG}$. Then, for any polynomial N, the BJK secret key IPE scheme $\overline{\mathsf{skIPE}}^N$ is multi-instance weak function hiding w.r.t. $\mathcal{MG}$.*

Proof Sketch. Towards showing the indistinguishability of $\mathsf{MFH}^{\mathsf{sIPE}}_{A,\mathcal{MG}}(1^\lambda, 0)$ and $\mathsf{MFH}^{\mathsf{sIPE}}_{A,\mathcal{MG}}(1^\lambda, 1)$ for *any* PPT A, it suffices to show their indistinguishability against a class of weaker adversaries that only send challenge queries $(\mathsf{mesg}\ (\mathbf{x}^0_{k^\star,i} \text{ or } \mathbf{x}^1_{k^\star,i}), k^\star, i)$ and $(\mathsf{func}\ (\mathbf{y}^0_{k^\star,j} \text{ or } \mathbf{y}^1_{k^\star,j}), k^\star, j)$ in a single instance $k^\star \leq m$, and in all other instances $k \neq k^\star$ it only asks for auxiliary queries $(\mathsf{aux\text{-}mesg}\ \{\mathbf{x}_{l,k,i}\}_l, k, i)$ and $(\mathsf{aux\text{-}func}\ \{\mathbf{y}_{1,k,j}\}_l, k, j)$ where the inputs and functions do not change. This holds since security against all PPT adversaries is implied by security against the weaker adversaries via a simple hybrid argument, where the k^{th} hybrid (for $j \leq m$) is identical to $\mathsf{MFH}^{\mathsf{sIPE}}_{A,\mathcal{MG}}(1^\lambda, 0)$, except that input and function vectors encoded in the first k instance are $\mathbf{x}^1_{k',i}, \mathbf{y}^1_{k',j}$ (as opposed to $\mathbf{x}^0_{k',i}, \mathbf{y}^0_{k',j}$) for $k' \leq k$.

Fix such a weaker adversary A, and security parameter λ. Assume w.l.o.g. that A submits challenge queires only in the first instance $k = 1$ and submits auxiliary queries in all other instances $k > 1$. Call k the challenge instance and all others auxiliary instances. Security against A is almost the same as single-instance weak function hiding, except from the following differences:

- the random scalars used in generating the challenge function keys and ciphertexts—α_j, β_j for $\mathsf{sk}_{1,j}$ and $\alpha^\star_i, \beta^\star_i$ for $\mathsf{ct}_{1,i}$, are also used for generating many auxiliary function keys $\mathsf{sk}_{l,k,j}$ and ciphertexts $\mathsf{ct}_{l,k,i}$ in auxiliary instances $k > 1$.

Therefore, security against A follows from essentially the same proof of single-instance weak function hiding in BJK, except from the need to simulate the auxiliary instances for A. Below, we briefly review the BJK proof and show how to adapt it to simulate auxiliary instances.

Let Q_1, Q_2 be respectively upper bounds on the number of ciphertexts and function keys that A obtains in each instance. The BJK proof calls a normal ciphertext of $\mathbf{x}$ a type-$(\mathbf{x}, \mathbf{x})$ ciphertext, since $\mathbf{x}$ is embedded twice. It also uses the notion of a type-$(\mathbf{x}, 0)$ and type-$(0, \mathbf{x})$ ciphertext, where $\mathbf{x}$ is only embedded once as $\alpha^\star B^\star_1 \mathbf{x}$ or $\beta^\star B^\star_2 \mathbf{x}$. Similarly, there are type-$(\mathbf{y}, \mathbf{y})$, type-$(\mathbf{y}, 0)$ and type-$(0, \mathbf{y})$ function keys.

The BJK proof goes through two sequences of hybrids:

- Sequence $\mathbf{Game}_{0,Z} \cdots \mathbf{Game}_{Q_1,Z}$ changes the input vectors being encoded. In $\mathbf{Game}_{j,z}$ the first j ciphertexts in the challenge instance are of type-$(\mathbf{x}^1_j, \mathbf{x}^1_j)$ and the rest ciphertexts are of type-$(\mathbf{x}^0_j, \mathbf{x}^0_j)$, while all keys are of type-$(\mathbf{y}^0_j, \mathbf{y}^0_j)$.

- Sequence $\mathbf{Game}_{O,0} = \mathbf{Game}_{Q_1,Z} \cdots \mathbf{Game}_{O,j}$ changes the function vectors being encoded. In $\mathbf{Game}_{O,j}$ the first j function keys in the challenge instance are of type-$(\mathbf{y}^1_j, \mathbf{y}^1_j)$ and the rest are of type-$(\mathbf{y}^0_j, \mathbf{y}^0_j)$, while all ciphertexts are of type-$(\mathbf{x}^1_j, \mathbf{x}^1_j)$.

In our context, each hybrid additionally generates function keys of type-$(\mathbf{y}_{l,k,j}, \mathbf{y}_{l,k,j})$ and ciphertexts of type-$(\mathbf{x}_{l,k,j}, \mathbf{x}_{l,k,j})$ in auxiliary instances. Note that they remain the same in all hybrids.

Since in the BJK IPE scheme, the function key and ciphertexts are completely symmetric, the indistinguishability proof for the neighboring hybrids in the first and second sequences are identical. Therefore it suffice to prove the indistinguishability between $\mathbf{Game}_{j-1,Z}$ and $\mathbf{Game}_{j,Z}$, which in turn is proven using another sequence of hybrids $\mathbf{Game}^0_{j,Z} = \mathbf{Game}_{j-1,z}, \mathbf{Game}^1_{j,Z}, \cdots, \mathbf{Game}^7_{j,Z} = \mathbf{Game}_{j,Z}$ as follows.

- $\mathbf{Game}^1_{j,Z}$ is the same as $\mathbf{Game}^0_{j,Z}$ except that all challenge ciphertexts $i < j$ are of type-$(0, \mathbf{x}^1_i)$, the j^{th} is of type-$(0, \mathbf{x}^0_j)$, and all others $i > j$ are of type-$(0, \mathbf{x}^0_i)$. And every auxiliary-ciphertexts $\mathsf{ct}_{l,k,i}$ is of type-$(0, \mathbf{x}_{l,k,i})$.

- $\mathbf{Game}^2_{j,Z}$ is the same as $\mathbf{Game}^1_{j,Z}$, except that the j^{th} challenge ciphertext is of type-$(\mathbf{x}^0_j, \mathbf{x}^0_j)$. And all auxiliary-ciphertexts $\mathsf{ct}_{l,k,j}$ for the same j are of type-$(\mathbf{x}_{l,k,j}, \mathbf{x}_{l,k,j})$.

- $\mathbf{Game}^3_{j,Z}$ is the same as $\mathbf{Game}^2_{j,Z}$, except that the j^{th} challenge ciphertext is of type-$(\mathbf{x}^0_j, \mathbf{x}^1_j)$.
- $\mathbf{Game}^4_{j,Z}$ is the same as $\mathbf{Game}^3_{j,Z}$, except that all challenge ciphertexts $i < j$ are of type-$(\mathbf{x}^1_i, \mathbf{x}^1_i)$, and all others $i > j$ are of type-$(\mathbf{x}^0_i, \mathbf{x}^0_i)$. And all auxiliary-ciphertexts $\mathsf{ct}_{l,k,i}$ for the $i \neq j$ are of type-$(\mathbf{x}_{l,k,i}, \mathbf{x}_{l,k,i})$.
- $\mathbf{Game}^5_{j,Z}$ is the same as $\mathbf{Game}^4_{j,Z}$, except that all challenge ciphertexts $i < j$ are of type-$(\mathbf{x}^1_i, 0)$, and all others $i > j$ are of type-$(\mathbf{x}^0_i, 0)$. And all auxiliary-ciphertexts $\mathsf{ct}_{l,k,i}$ for the $i \neq j$ are of type-$(\mathbf{x}_{l,k,i}, 0)$.
- $\mathbf{Game}^6_{j,Z}$ is the same as $\mathbf{Game}^5_{j,Z}$, except that the j^{th} challenge ciphertext is of type-$(\mathbf{x}^1_j, \mathbf{x}^1_j)$.
- $\mathbf{Game}^7_{j,Z}$ is the same as $\mathbf{Game}^6_{j,Z}$, except that all challenge ciphertexts $i < j$ are of type-$(\mathbf{x}^1_i, \mathbf{x}^1_i)$, and all others $i > j$ are of type-$(\mathbf{x}^0_i, \mathbf{x}^0_i)$. And all auxiliary-ciphertexts $\mathsf{ct}_{l,k,i}$ for the $i \neq j$ are of type-$(\mathbf{x}_{l,k,i}, \mathbf{x}_{l,k,i})$.

To show the indistinguishability of neighboring hybrids, BJK uses two types of arguments. We argue that these arguments work even at the presence of auxiliary instances of types defined above.

- $\mathbf{Game}^2_{j,Z} \approx \mathbf{Game}^3_{j,Z}$ and $\mathbf{Game}^5_{j,Z} \approx \mathbf{Game}^6_{j,Z}$ follow from the an information theoretic argument. This means the marginal distributions $\mathcal{D}_2$ and $\mathcal{D}_3$ of function keys and ciphertexts in the challenge instance in $\mathbf{Game}^2_{j,Z}$ and $\mathbf{Game}^3_{j,Z}$ are statistically close. Furthermore, given a sample from $\mathcal{D}_2$ or $\mathcal{D}_3$, the auxiliary instances in $\mathbf{Game}^2_{j,Z}$ or $\mathbf{Game}^3_{j,Z}$ respectively can be sampled (using unbounded time) conditioned on sharing the random scalars used in the challenge instance appropriately. Therefore, $\mathbf{Game}^2_{j,Z}$ and $\mathbf{Game}^3_{j,Z}$ are statistically close even when there are auxiliary instances. The same holds for $\mathbf{Game}^5_{j,Z}$ and $\mathbf{Game}^6_{j,Z}$.
- In BJK, the indistinguishability of other neighboring hybrids follows from the SXDH assumption of the 0^{th} source group G_0 of the asymmetric bilinear groups. Consider $\mathbf{Game}^0_{j,z} \approx \mathbf{Game}^1_{j,z}$ for example (other indistinguishability follows from similar arguments). We claim that when there are auxiliary instances, it suffices to replace the SXDH assumption with the joint-SXDH assumption over the 0^{th} source groups of all instances.

  The BJK proof provides a reduction from the indistinguishability of the challenge instance in $\mathbf{Game}^0_{j,z}$ and $\mathbf{Game}^1_{j,z}$ to the indistinguishability of the DDH tuple $[a]_{1,0}, [b]_{1,0}, [ab + r]_{1,0}$ where r is random or zero. (Recall that the ciphertexts of the challenge instance exist in group $G_{1,0}$.) The reduction provides a way to generate from $[a]_{1,0}, [b]_{1,0}, [ab + r]_{1,0}$ any ciphertext $\mathsf{ct}_{1,i}$ or any function keys $\mathsf{sk}_{1,j}$ in the challenge instance, using correlated randomness that determines the master secret key msk_1. The resulting ciphertext $\mathsf{ct}_{1,i}$ uses random scalar $\alpha^\star_i = \alpha'_i r$, $\beta^\star_i = \beta'_i + \alpha'_i b$, and the function key $\mathsf{sk}_{1,j}$ uses random scalar $\alpha_j = \alpha''_j$, $\beta_j = \alpha''_j a + \beta''_j$, where $\alpha'_i, \beta'_i, \alpha''_j, \beta''_j$ are random scalars sampled by the reduction.

  We can use the BJK reduction to simulate any auxiliary ciphertext $\mathsf{ct}_{l,k,i}$ or function key $\mathsf{sk}_{l,k,j}$ in instance k, which share $\alpha^\star_i, \beta^\star_i$ and α_j, β_j. Simple use the method provided by the BJK reduction to generate from $[a]_{k,0}, [b]_{k,0}, [ab + r]_{k,0}$ (now in group $G_{k,0}$) ciphertext $\mathsf{ct}_{l,k,i}$ or key $\mathsf{sk}_{l,k,j}$ using correlated randomness that determines msk_k. As long as the same random scalars $\alpha'_i, \beta'_i, \alpha''_j, \beta''_j$ are used, the resulting ciphertext and function key use exactly scalars $\alpha^\star_i, \beta^\star_i$ and α_j, β_j as desired. Therefore, by the joint-SXDH assumption over groups $\{G_{k,0}\}_k$, $\mathbf{Game}^0_{j,z} \approx \mathbf{Game}^1_{j,z}$.

Finally, we note that when proving the indistinguishability of $\mathbf{Game}_{O,j-1}$ and $\mathbf{Game}_{O,j}$, the joint-SXDH assumption over groups $\{G_{k,1}\}_k$ is needed. □

5.7 Our Multi-Instance (Strongly) Function-Hiding Secret-Key IPE

We show how to generically construct a multi-instance *strongly* function-hiding secret-key IPE skIPE^N for computing inner products of length-N vectors, starting from any multi-instance *weakly* function-hiding secret-key IPE scheme $\overline{\mathsf{skIPE}}^{2N}$ for computing the inner product of length-$2N$ vectors. The idea enabling this bootstrapping is (extremely) simple.

A function key $\mathsf{sk}(\mathbf{y})$ for a vector $\mathbf{y}$ (respectively, ciphertext $\mathsf{ct}(\mathbf{x})$ for $\mathbf{x}$) of skIPE^N is simply a function key $\overline{\mathsf{sk}}(\mathbf{y}||0^N)$ for the vector $\mathbf{y}||0^N$ (respectively, ciphertext $\overline{\mathsf{ct}}(\mathbf{x}||0^N)$ for the vector $\mathbf{x}||0^N$) of $\overline{\mathsf{skIPE}}^N$. That is,

$$\forall \mathbf{y} \in \mathbb{Z}_p^N,\ \mathsf{sk}(\mathbf{y}^0) = \overline{\mathsf{sk}}(\mathbf{y}^0||0^N) \quad \text{and} \quad \forall \mathbf{x} \in \mathbb{Z}_p^N,\ \mathsf{ct}(\mathbf{x}^0) = \overline{\mathsf{ct}}(\mathbf{x}^0||0^N)$$

To see why this simple idea works, consider showing the function hiding property w.r.t. a single challenge function and message query $(\mathbf{y}^0, \mathbf{y}^1)$ and $(\mathbf{x}^0, \mathbf{x}^1)$ satisfying $\mathbf{y}^0 \cdot \mathbf{x}^0 = \mathbf{y}^1 \cdot \mathbf{x}^1$ ($\neq \mathbf{x}^0 \cdot \mathbf{y}^1$ and $\neq \mathbf{x}^1 \cdot \mathbf{y}^0$). We want to show that:

$$\left\{\mathsf{ct}(\mathbf{x}^0), \mathsf{sk}(\mathbf{y}^0)\right\} \approx \left\{\mathsf{ct}(\mathbf{x}^1), \mathsf{sk}(\mathbf{y}^1)\right\}$$

That is, we want to show that:

$$\left\{\overline{\mathsf{ct}}(\mathbf{x}^0||0^N), \overline{\mathsf{sk}}(\mathbf{y}^0||0^N)\right\} \approx \left\{\overline{\mathsf{ct}}(\mathbf{x}^1||0^N), \overline{\mathsf{sk}}(\mathbf{y}^1||0^N)\right\}$$

Because of the redundancy in the encoded vectors, we can go through the following sequence of hybrids, encoding using $\overline{\mathsf{skIPE}}^{2N}$ intermediate vectors $\mathbf{X}_i, \mathbf{Y}_i$ satisfying the more stringent condition that $\mathbf{X}_i \cdot \mathbf{Y}_i = \mathbf{X}_i \cdot \mathbf{Y}_{i+1} = \mathbf{X}_{i+1} \cdot \mathbf{Y}_{i+1}$. By weak function hiding, neighboring hybrids are indistinguishable, which concludes the function hiding of skIPE^N. In particular, we show that:

$$\begin{aligned}&\left\{\overline{\mathsf{ct}}(\mathbf{x}^0||0^N),\ \overline{\mathsf{sk}}(\mathbf{y}^0||0^N)\right\}\\ &\qquad\approx \left\{\overline{\mathsf{ct}}(0^N||\mathbf{x}^1),\ \overline{\mathsf{sk}}(\mathbf{y}^0||\mathbf{y}^1)\right\} \approx \left\{\overline{\mathsf{ct}}(\mathbf{x}^1||0^N),\ \overline{\mathsf{sk}}(\mathbf{y}^1||\mathbf{y}^1)\right\}\\ &\qquad\qquad\approx \left\{\overline{\mathsf{ct}}(\mathbf{x}^1||0^N),\ \overline{\mathsf{sk}}(\mathbf{y}^1||0^N)\right\}\end{aligned}$$

The reader can easily verify that the indistinguishability of each adjacent pair of hybrids follows from weak function-hiding of $\overline{\mathsf{skIPE}}$ together with the fact that $\mathbf{y}^0 \cdot \mathbf{x}^0 = \mathbf{y}^1 \cdot \mathbf{x}^1$. For example, referring to the first and second hybrids, note that

$$(\mathbf{x}^0||0^N) \cdot (\mathbf{y}^0||0^N) = (\mathbf{x}^0||0^N) \cdot (\mathbf{y}^0||\mathbf{y}^1) = (0^N||\mathbf{x}^1) \cdot (\mathbf{y}^0||\mathbf{y}^1)$$

which satisfies the pre-condition necessary to invoke weak function-hiding. (Here, as usual, $\cdot$ refers to the inner product operation and $||$ refers to the concatenation of two vectors.)

We now formally present our multi-instance function hiding IPE scheme skIPE^N, using a multi-instance weak function hiding IPE scheme $\overline{\mathsf{skIPE}}^{2N}$.

The Scheme skIPE^N. Let $\mathsf{pp} = (p, G_0, G_1, G_T, \otimes)$ be the public parameter that describes a pair of asymmetric bilinear groups with order p.

- *Setup:* $\mathsf{skIPE}.\mathsf{Setup}(1^\lambda, 1^N, \mathsf{pp})$ samples $\mathsf{msk} \xleftarrow{\$} \overline{\mathsf{skIPE}}.\mathsf{Setup}(1^\lambda, 1^{2N}, \mathsf{pp})$.
- *Key Generation:* $\mathsf{skIPE}.\mathsf{KeyGen}(\mathsf{msk}, \mathbf{y})$ samples $\mathsf{sk} \xleftarrow{\$} \overline{\mathsf{skIPE}}.\mathsf{KeyGen}(\mathsf{msk}, \mathbf{y}||0^N)$.

- *Encryption:* $\mathsf{skIPE.Enc}(\mathsf{msk}, \mathbf{x})$ samples $\mathsf{ct} \xleftarrow{\$} \overline{\mathsf{skIPE}}.\mathsf{Enc}(\mathsf{msk}, \mathbf{x}||0^N)$.
- *Decryption:* $\mathsf{skIPE.Dec}(\mathsf{sk}, \mathsf{ct})$ outputs $z = \overline{\mathsf{skIPE}}.\mathsf{Dec}(\mathsf{sk}, \mathsf{ct})$.

The correctness of skIPE^N follows directly from that of $\overline{\mathsf{skIPE}}^{2N}$ and the fact that $(\mathbf{x}||0^N) \cdot (\mathbf{y}||0^N) = \mathbf{x} \cdot \mathbf{y}$. Next we analyze its security.

Lemma 5. *Let $\mathcal{MG}$ be a multi-group generator and N a polynomial. If $\overline{\mathsf{skIPE}}^{2N}$ is μ-multi-instance weakly function hiding w.r.t. $\mathcal{MG}$, then skIPE^N is 3μ-multi-instance function hiding w.r.t. $\mathcal{MG}$.*

Proof. We want to show that for every PPT adversary A, and every sufficiently large security parameter $\lambda \in \mathbb{N}$, the adversary's advantage is bounded by $3\mu(\lambda)$.

$$\mathsf{Advt}^{\mathsf{skIPE}}_{A,\mathcal{MG}} = \left|\Pr[\mathsf{MFH}^{\mathsf{skIPE}}_{A,\mathcal{MG}}(1^\lambda, 0) = 1] - \Pr[\mathsf{MFH}^{\mathsf{skIPE}}_{A,\mathcal{MG}}(1^\lambda, 1) = 1]\right| \leq 3\mu(\lambda)$$

Towards this, we consider a sequence of hybrids, $H_0, \cdots, H_3$, where H_0 and H_3 are identical to experiments $\mathsf{MFH}^{\mathsf{skIPE}}_{A,\mathcal{MG}}(1^\lambda, 0)$ and $\mathsf{MFH}^{\mathsf{skIPE}}_{A,\mathcal{MG}}(1^\lambda, 1)$ respectively, and show that the probabilities that A outputs 1 in neighboring hybrids differ at most by $\mu(\lambda)$.

- *Hybrid H_1* proceeds identically to H_0 except that the challenger CH generates challenge function keys and ciphertexts as follows:
  - For every $(\mathsf{mesg}\ (\mathbf{x}^0_{k,i}, \mathbf{x}^1_{k,i}), k, i)$ query, it returns $\mathsf{ct}_{k,i} \xleftarrow{\$} \overline{\mathsf{skIPE}}.\mathsf{Enc}(\mathsf{msk}, 0^N||\mathbf{x}^1_{k,i};\ r^\star_i)$, as opposed to $\mathsf{ct}_{k,i} \xleftarrow{\$} \overline{\mathsf{skIPE}}.\mathsf{Enc}(\mathsf{msk}, \mathbf{x}^0_{k,i}||0^N;\ r^\star_i)$ in H_0.
  - For every $(\mathsf{func}\ (\mathbf{y}^0_{k,i}, \mathbf{y}^1_{k,i}), k, i)$ query, it returns $\mathsf{sk}_{k,i} \xleftarrow{\$} \overline{\mathsf{skIPE}}.\mathsf{KeyGen}(\mathsf{msk}, \mathbf{y}^0_{k,i}||\mathbf{y}^1_{k,i};\ r_i)$, as opposed to $\mathsf{sk}_{k,i} \xleftarrow{\$} \overline{\mathsf{skIPE}}.\mathsf{KeyGen}(\mathsf{msk}, \mathbf{y}^0_{k,i}||0^N;\ r_i)$ in H_0.

  For every $k \in [m]$, every pair of challenge function key and ciphertext satisfies,

  $$\forall i, j,\ \mathbf{x}^0_{k,i}||0^N \cdot \mathbf{y}^0_{k,j}||0^N = \mathbf{x}^0_{k,i}||0^N \cdot \mathbf{y}^0_{k,j}||\mathbf{y}^1_{k,j} = 0^N||\mathbf{x}^1_{k,i} \cdot \mathbf{y}^0_{k,j}||\mathbf{y}^1_{k,j}\ ,$$

  since $\mathbf{x}^0_{k,i} \cdot \mathbf{y}^0_{k,j} = \mathbf{x}^1_{k,i} \cdot \mathbf{y}^1_{k,j}$. Then it follows directly from the μ-multi-instance weak function hiding of $\overline{\mathsf{skIPE}}^{2N}$ that H_0 and H_1 are μ-indistinguishable to A.

- *Hybrid H_2* proceeds identically to H_1 except that the challenger CH generates challenge function keys and ciphertexts as follows:
  - For every $(\mathsf{mesg}\ (\mathbf{x}^0_{k,i}, \mathbf{x}^1_{k,i}), k, i)$ query, it returns $\mathsf{ct}_{k,i} \xleftarrow{\$} \overline{\mathsf{skIPE}}.\mathsf{Enc}(\mathsf{msk}, \mathbf{x}^1_{k,i}||0^N;\ r^\star_i)$, as opposed to $\mathsf{ct}_{k,i} \xleftarrow{\$} \overline{\mathsf{skIPE}}.\mathsf{Enc}(\mathsf{msk}, 0^N||\mathbf{x}^1_{k,i};\ r^\star_i)$ in H_1.
  - For every $(\mathsf{func}\ (\mathbf{y}^0_{k,i}, \mathbf{y}^1_{k,i}), k, i)$ query, it returns $\mathsf{sk}_{k,i} \xleftarrow{\$} \overline{\mathsf{skIPE}}.\mathsf{KeyGen}(\mathsf{msk}, \mathbf{y}^1_{k,i}||\mathbf{y}^1_{k,i};\ r_i)$, as opposed to $\mathsf{sk}_{k,i} \xleftarrow{\$} \overline{\mathsf{skIPE}}.\mathsf{KeyGen}(\mathsf{msk}, \mathbf{y}^0_{k,i}||\mathbf{y}^1_{k,i};\ r_i)$ in H_1.

  For every $k \in [m]$, every pair of challenge function key and ciphertext satisfies,

  $$\forall i, j,\ 0^N||\mathbf{x}^1_{k,i} \cdot \mathbf{y}^0_{k,j}||\mathbf{y}^1_{k,j} = 0^N||\mathbf{x}^1_{k,i} \cdot \mathbf{y}^1_{k,j}||\mathbf{y}^1_{k,j} = \mathbf{x}^1_{k,i}||0^N \cdot \mathbf{y}^1_{k,j}||\mathbf{y}^1_{k,j}\ ,$$

  since $\mathbf{x}^0_{k,i} \cdot \mathbf{y}^0_{k,j} = \mathbf{x}^1_{k,i} \cdot \mathbf{y}^1_{k,j}$. Then it follows directly from the μ-multi-instance weak function hiding of $\overline{\mathsf{skIPE}}^{2N}$ that H_1 and H_2 are μ-indistinguishable to A.

- (The final) Hybrid $H_3 = \mathsf{MFH}^{\mathsf{skIPE}}_{A,\mathcal{MG}}(1^\lambda, 1)$ proceeds identically to H_2 except that the challenge ciphertexts and function keys are generated as follows:
  - For every $(\mathsf{mesg}\ (\mathbf{x}^0_{k,i}, \mathbf{x}^1_{k,i}), k, i)$ query, it returns $\mathsf{ct}_{k,i} \xleftarrow{\$} \overline{\mathsf{skIPE}}.\mathsf{Enc}(\mathsf{msk}, \mathbf{x}^1_{k,i}||0^N;\ r^\star_i)$, identical to H_2.
  - For every $(\mathsf{func}\ (\mathbf{y}^0_{k,i}, \mathbf{y}^1_{k,i}), k, i)$ query, it returns $\mathsf{sk}_{k,i} \xleftarrow{\$} \overline{\mathsf{skIPE}}.\mathsf{KeyGen}(\mathsf{msk}, \mathbf{y}^1_{k,i}||0^N;\ r_i)$, as opposed to $\mathsf{sk}_{k,i} \xleftarrow{\$} \overline{\mathsf{skIPE}}.\mathsf{KeyGen}(\mathsf{msk}, \mathbf{y}^1_{k,i}||\mathbf{y}^1_{k,i};\ r_i)$ in H_2.

  For every $k \in [m]$, every pair of challenge function key and ciphertext satisfies,

$$\forall i, j,\ \mathbf{x}^1_{k,i}||0^N \cdot \mathbf{y}^1_{k,j}||\mathbf{y}^1_{k,j} = \mathbf{x}^1_{k,i}||0^N \cdot \mathbf{y}^1_{k,j}||0^N\ ,$$

  It follows again from the μ-multi-instance weak function hiding of $\overline{\mathsf{skIPE}}^{2N}$ that H_2 and H_3 are μ-indistinguishable to A.

Therefore, by a hybrid argument, H_0 and H_3 are 3μ-indistinguishable to A, which concludes the lemma. □

6 Slotted Public Key IPE

6.1 Public Key IPE

A public-key IPE scheme IPE^N for computing length-N inner products consists of four PPT algorithms ($\mathsf{IPE.Setup}$, $\mathsf{IPE.KeyGen}$, $\mathsf{IPE.Enc}$, $\mathsf{IPE.Dec}$) that have the same syntax as that of a secret-key IPE scheme skIPE except that (i) the setup algorithm $\mathsf{IPE.Setup}(1^\lambda, \mathsf{pp})$ outputs a master public key mpk and a master secret key msk, and (ii) the encryption algorithm $\mathsf{IPE.Enc}(\mathsf{mpk}, \mathbf{x})$ on input the master public key mpk and vector $\mathbf{x}$ outputs a ciphertext ct. Like in the secret key setting, the public parameter pp specifies the ring $\mathcal{R}$ over which inner products are computed.

The security of a public key IPE is simply the indistinguishability security of public key FE scheme, restricted to inner product computations. Note that this notion refers to hiding the input that is encrypted, and *not* the function for which keys are generated.

Definition 19. *Let $\mathcal{G}$ be a public parameter generator and N be any polynomial. We say that a public-key IPE scheme IPE^N for computing length-N inner products is μ-secure w.r.t. $\mathcal{G}$, if it satisfies the condition in Definition 7 with the setup algorithm $\mathsf{IPE.Setup}(1^\lambda, \mathsf{pp})$ for $\mathsf{pp} \xleftarrow{\$} \mathcal{G}(1^\lambda)$, and function class $\mathcal{F}_\lambda$ consisting of all inner product function $f_{\mathbf{y}}(\mathbf{x}) = \mathbf{y} \cdot \mathbf{x}$ over $\mathcal{R}$ specified by pp.*

6.2 ABDP Public Key IPE

In [ABDP15], Abdalla, Bourse, De Caro, and Pointcheval constructed public key IPE schemes with indistinguishability security, based on the DDH assumption. We recall their scheme, which is very similar to the ElGamal encryption scheme.

The Decisional Diffie-Hellman (DDH) Assumption Let $\mathcal{G}$ denote a group generator that on input 1^λ outputs (p, G), where G is a group of prime order p, and g is a generator of G contained in its description. As above, we use the bracket notation to denote elements in G: $[v] = g^v$ and

$[\mathbf{x}] = g^{\mathbf{x}} = g^{x_1}, \cdots, g^{x_m}$ for $v \in \mathbb{Z}_p, \mathbf{x} \in \mathbb{Z}_p^m$. The DDH assumption states that the following two ensembles are μ-indistinguishable for a negligible function μ:

$$\left\{(p, G) \xleftarrow{\$} \mathcal{G}(1^\lambda),\ a, b \xleftarrow{\$} \mathbb{Z}_p\ :\ (p, G),\ [a], [b], [ab]\right\}_\lambda$$
$$\left\{(p, G) \xleftarrow{\$} \mathcal{G}(1^\lambda),\ a, b, r \xleftarrow{\$} \mathbb{Z}_p\ :\ (p, G),\ [a], [b], [r]\right\}_\lambda$$

Overview of the ABDP Scheme Recall that the basic ElGamal encryption scheme for message space $\mathbb{Z}_p$ is as follows:

$$\mathsf{sk} \xleftarrow{\$} \mathbb{Z}_p, \qquad \mathsf{pk} = g^{\mathsf{sk}}, \qquad \mathsf{ct} = (g^r,\ \mathsf{pk}^r g^m) = (g^r,\ g^{(r\ \mathsf{sk} + x)}) \text{ for } r \xleftarrow{\$} \mathbb{Z}_p$$

Note that in this scheme, decryption can be done when x is small.

Under our bracket notation this is written as:

$$\mathsf{sk} \xleftarrow{\$} \mathbb{Z}_p, \qquad \mathsf{pk} = [\mathsf{sk}], \qquad \mathsf{ct} = [r],\ (r \odot \mathsf{pk}) \oplus [x] = [r \mid\mid (r\ \mathsf{sk} + x)]$$

(Recall that "$\odot$" and "$\oplus$" are respectively the homomorphic scalar multiplication and addition operations over encodings.) The ElGamal encryption can be easily modified to encrypt vectors $\mathbf{x} \in \mathbb{Z}_p^N$, while sharing the random scalar r, and maintaining security under the same (DDH) assumption.

$$\mathsf{msk} \xleftarrow{\$} \mathbb{Z}_p^N, \quad \mathsf{mpk} = [\mathsf{msk}], \quad \mathsf{ct} = [-r],\ (r \odot \mathsf{mpk}) \oplus [\mathbf{x}] = [(-r \mid\mid r\ \mathsf{msk} + \mathbf{x})]$$

(We encode $-r$ instead of r for convenience.) To turn the above scheme into an IPE scheme, observe that given a vector $\mathbf{y} \in \mathbb{Z}_p^N$ and the inner product $\mathbf{y} \cdot \mathsf{msk}$ in the clear, one can homomorphically evaluate,

$$(\mathbf{y} \cdot \mathsf{msk} \mid\mid \mathbf{y}) \cdot \mathsf{ct} = (\mathbf{y} \cdot \mathsf{msk} \mid\mid \mathbf{y}) \cdot [-r \mid\mid (r(\mathsf{msk} + \mathbf{x}))] = [-r\mathbf{y} \cdot \mathsf{msk} + r\mathbf{y} \cdot \mathsf{msk} + \mathbf{x} \cdot \mathbf{y}] = [(\mathbf{x} \cdot \mathbf{y})]\ .$$

Therefore, it suffice to release $\mathbf{y} \cdot \mathsf{msk} \mid\mid \mathbf{y}$ as the function key for computing the inner product.

The ABDP Scheme We now formally describe the ABDP public key IPE scheme IPE^N. Let $\mathsf{pp} = (p, G)$ be a public parameter that describes a prime order group G; the inner product is computed over $\mathbb{Z}_p^N$.

- *Setup:* $\mathsf{IPE.Setup}(1^\lambda, \mathsf{pp})$ samples $\mathbf{s} \xleftarrow{\$} \mathbb{Z}_p^N$, and outputs master public key $\mathsf{mpk} = [\mathbf{s}]$ and master secret key $\mathsf{msk} = \mathbf{s}$.
- *Key Generation:* $\mathsf{IPE.KeyGen}(\mathsf{msk}, \mathbf{y})$ on input the master secret key $\mathsf{msk} = \mathbf{s}$ and vector $\mathbf{y}$ both in $\mathbb{Z}_p^N$, simply outputs $\mathsf{sk} = \mathbf{y} \cdot \mathbf{s} \mid\mid \mathbf{y}$.
- *Encryption:* $\mathsf{IPE.Enc}(\mathsf{mpk}, \mathbf{x})$ on input the master public key $\mathsf{mpk} = [\mathbf{s}]$ and vector $\mathbf{x} \in \mathbb{Z}_p^N$, samples a random scalar $r \xleftarrow{\$} \mathbb{Z}_p$ and outputs

$$\mathsf{ct} = [-r] \mid\mid (r \odot \mathsf{mpk}) \oplus [\mathbf{x}] = [-r \mid\mid r\mathbf{s} + \mathbf{x}]$$

- *Decryption:* $\mathsf{IPE.Dec}(\mathsf{sk}, \mathsf{ct})$ on input $\mathsf{sk} = (\mathbf{y} \cdot \mathbf{s} \,||\, \mathbf{y})$ and $\mathsf{ct} = [-r \,||\, r\mathbf{s} + \mathbf{x}]$ homomorphically computes the inner product between sk and ct.

  $$\mathsf{sk} \cdot \mathsf{ct} = [(\mathbf{y} \cdot \mathbf{s} \,||\, \mathbf{y}) \cdot (-r \,||\, r\mathbf{s} + \mathbf{x})] = [\mathbf{x} \cdot \mathbf{y}]$$

  If the output value $z = \mathbf{x} \cdot \mathbf{y}$ is bounded by some polynomial bound $\Gamma = \Gamma(\lambda)$. Then z can be extracted by trying all possible values $i \in [\Gamma]$, and outputting the value i satisfying $[(\mathbf{x} \cdot \mathbf{y})] = i \odot [1]$.

Correctness of the scheme is easy to see. The security proof is, however, non-trivial. Abdalla et al. [ABDP15] showed that the scheme in fact satisfies simulation-based security, which implies the notion of indistinguishability-based security considered in this work.

Lemma 6. *Assume that the DDH assumption holds in the groups generated by $\mathcal{G}$. Then, for any polynomial N, the ABDP public key IPE scheme IPE^N is secure w.r.t. $\mathcal{G}$.*

Remark 2 (Linearity of the ABDP Schemes)**.** *We note that the algorithms of the ABDP schemes are "linear" in the following sense.*

- *Key Generation is deterministic, and the algorithm* $\mathsf{IPE.KeyGen}(\mathsf{msk}, \mathbf{y})$ *is linear over* $\mathbf{y}$.

- *Encryption uses a random scalar r, and the algorithm* $\mathsf{IPE.Enc}(\mathsf{mpk}, \mathbf{x};\ r)$ *is linear over* $\mathbf{x}$*, and elements in* mpk *and* $\mathbf{x}$ *are never multiplied together.*

- *Decryption is deterministic, and the algorithm* $\mathsf{IPE.Dec}(\mathsf{sk}, \mathsf{ct})$ *is linear over* sk*, as well as over the vector encoded in* ct.

Jumping ahead, later when constructing a slotted public key IPE, we will replace $\mathbf{y}$ *and* $\mathbf{x}$ *with their encoded versions, and thanks to the linearity properties, the above algorithms can still be evaluated homomorphically.*

6.3 Definition of Output-Encoded Slotted-IPE

Syntax An output-encoded slotted IPE scheme sIPE^N for computing length-N inner products consists of the following six PPT algorithms, and is associated with an output encoding function OEnc. When instantiated with a public parameter generator $\mathcal{G}$, the algorithms proceed as follows:

- *Setup:* $\mathsf{sIPE.Setup}(1^\lambda, \mathsf{pp};\ r_S)$ is an algorithm that on input a security parameter, a public parameter pp in the support of $\mathcal{G}$ that contains the descrption of a ring $\mathcal{R}$, and (partial) random coins r_S, outputs a master public key mpk and a master secret key msk.

  Note: sIPE.Setup *could sample additional random coins besides r_S. When we write directly* $(\mathsf{mpk}, \mathsf{msk}) \xleftarrow{\$} \mathsf{sIPE.Setup}(1^\lambda, \mathsf{pp})$*, we mean r_S is randomly sampled. The same applies to following algorithms.*

- *Key Generation:* $\mathsf{sIPE.KeyGen}(\mathsf{msk}, \mathbf{y}_1, \mathbf{y}_2;\ r_K)$ on input the master secret key, *two vectors* $\mathbf{y}_1, \mathbf{y}_2 \in \mathcal{R}^{N(\lambda)}$ and (partial) random coins r_K, outputs a secret key $\mathsf{sk} = \mathsf{sk}(\mathbf{y}_1, \mathbf{y}_2)$ with $\mathbf{y}_1$ in the first slot and $\mathbf{y}_2$ in the second slot.

- *Secret Encryption:* $\mathsf{sIPE.sEnc}(\mathsf{msk}, \mathbf{x}_1, \mathbf{x}_2;\ r_E)$ on input the master secret key, two messages $\mathbf{x}_1, \mathbf{x}_2 \in \mathcal{R}^{N(\lambda)}$ and (partial) random coins r_E, outputs an encryption $\mathsf{ct} = \mathsf{ct}(\mathbf{x}_1, \mathbf{x}_2)$ with $\mathbf{x}_1$ in the first slot and $\mathbf{x}_2$ in the second slot.

- *Public Encryption:* $\mathsf{sIPE.Enc}(\mathsf{mpk}, \mathbf{x};\ r_E)$ on input the master public key a message $\mathbf{x} \in \mathcal{R}^{N(\lambda)}$ and (partial) random coins r_E, outputs an encryption $\mathsf{ct} = \mathsf{ct}(\mathbf{x}, 0^N)$ of $\mathbf{x}$ in the first slot.

- *Decryption:* $\mathsf{sIPE.Dec}(\mathsf{sk}, \mathsf{ct})$ outputs an encoding $Z = \mathsf{OEnc}(z, \mathsf{pp}, r_S, r_K, r_E)$ of the actual output z via OEnc. The encoding also depends on the partial random coins r_S, r_K, r_E and the public parameter pp.

  Note: In some cases, the actual output z can be extracted from the encoding Z. For example, in the BJK secret key IPE, our secret key IPE, and the ABDP public key IPE schemes, the decryption algorithms first produce an encoding of the output in some groups, and then the output can be extracted if its value falls in some polynomial-sized range.

Normal Public Key Mode In the normal public key mode, only the first slot in the ciphertexts and function keys are needed, meaning that only the public encryption algorithm $\mathsf{sIPE.Enc}$ is used, and the second vector $\mathbf{y}_2$ to the algorithm $\mathsf{sIPE.KeyGen}$ is always set to an arbitrary non-zero vector say 1^N.

Correctness: There is a negligible function μ, such that, for every λ, pp in the support of $\mathcal{G}$, and $\mathbf{x}_1, \mathbf{x}_2, \mathbf{y}_1, \mathbf{y}_2 \in \mathcal{R}^N$, the following holds conditions holds.

$$\Pr\left[\begin{array}{c} (\mathsf{mpk}, \mathsf{msk}) \xleftarrow{\$} \mathsf{sIPE.Setup}(1^\lambda, \mathsf{pp};\ r_S) \\ \mathsf{ct} \xleftarrow{\$} \mathsf{sIPE.sEnc}(\mathsf{mpk}, \mathbf{x}_1, \mathbf{x}_2;\ r_E) \\ \mathsf{sk} \xleftarrow{\$} \mathsf{sIPE.KeyGen}(\mathsf{msk}, \mathbf{y}_1, \mathbf{y}_2;\ r_K) \end{array} : \begin{array}{c} z = \mathbf{x}_1 \cdot \mathbf{y}_1 + \mathbf{x}_2 \cdot \mathbf{y}_2 \\ \mathsf{OEnc}(z, \mathsf{pp}, r_S, r_E, r_K) = \mathsf{sIPE.Dec}(\mathsf{sk}, \mathsf{ct}) \end{array}\right] \geq 1 - \mu(\lambda)$$

Statistically Indistinguishable Public Encryption. We require that ciphertexts produced by the public encryption to be statistically close to ciphertexts produced by the secret encryption when the second slot is set to 0^N. That is, there is a negligible function μ, such that, for every $\lambda, \mathsf{pp}, \mathbf{x}$, the following two distributions are at most $\mu(\lambda)$ far.

$$\left(\begin{array}{c} (\mathsf{mpk}, \mathsf{msk}) \xleftarrow{\$} \mathsf{sIPE.Setup}(1^\lambda, \mathsf{pp}) \\ \mathsf{ct} \xleftarrow{\$} \mathsf{sIPE.Enc}(\mathsf{mpk}, \mathbf{x}) \end{array} : \mathsf{mpk}, \mathsf{ct}\right), \left(\begin{array}{c} (\mathsf{mpk}, \mathsf{msk}) \xleftarrow{\$} \mathsf{sIPE.Setup}(1^\lambda, \mathsf{pp}) \\ \mathsf{ct} \xleftarrow{\$} \mathsf{sIPE.sEnc}(\mathsf{mpk}, \mathbf{x}, 0^N) \end{array} : \mathsf{mpk}, \mathsf{ct}\right)$$

Input Indistinguishability This security property corresponds to the basic IND-security of public key IPE, that is, encryption of vectors $\mathbf{x}_1^0, \mathbf{x}_2^0$ in the first and second slots is indistinguishable from encryption of vectors $\mathbf{x}_1^1, \mathbf{x}_2^1$, as long as, all function keys correspond to functions $\{\mathbf{y}_{i,1}, \mathbf{y}_{i,2}\}$ that have the same output on these inputs $(\mathbf{y}_{i,1}||\mathbf{y}_{i,2}) \cdot (\mathbf{x}_1^0||\mathbf{x}_2^0) = (\mathbf{y}_{i,1}||\mathbf{y}_{i,2}) \cdot (\mathbf{x}_1^1||\mathbf{x}_2^1)$. Here again, we consider multi-instance security as in the definition of multi-instance function hiding of secret key IPE (Definition 17), where the adversary sees ciphertexts and function keys of different instances, that are correlated through shared randomness and the public parameters. Like before, the public parameters are sampled jointly via a multi-parameter generator $\mathcal{MG}$. Different from before, the function keys and ciphertexts of different instances do not share all random coins, but rather only share (partial) random coins $r_S, \{r_{K,i}\}, r_E$ that affects the output encoding at decryption time. Formally,

Definition 20 (Multi-Instance Input Indistinguishability)**.** *An output-encoded slotted IPE scheme* sIPE^N *for computing length-N inner products has* μ-multi-instance input indistinguishability *w.r.t.* $\mathcal{MG}$*, if for every* PPT *adversary A, the following games* $\mathsf{Int}^{\mathsf{sIPE}}_{A,\mathcal{MG}}(1^\lambda, 0)$ *and* $\mathsf{Int}^{\mathsf{sIPE}}_{A,\mathcal{MG}}(1^\lambda, 1)$ *are μ-indistinguishable.*

$\mathsf{Int}^{\mathsf{sIPE}}_{A,\mathcal{MG}}(1^\lambda, b)$ *proceeds as follow:*

- **Public Parameter and Key Generation.** *The challenger does:*

- *sample public parameters* $(\mathsf{pp}_1, \cdots, \mathsf{pp}_m) \xleftarrow{\$} \mathcal{MG}(1^\lambda)$*, which contain the description of a ring* $\mathcal{R}$*,*
- *sample (partial) randomness* r_S*,*
- *sample* m *master public and secret key* $\{(\mathsf{mpk}_k, \mathsf{msk}_k) \xleftarrow{\$} \mathsf{sIPE.Setup}(1^\lambda, \mathsf{pp}_k;\ r_S)\}_{k\in[m]}$*, and sends* A *all master public keys* $\{\mathsf{mpk}_k\}$*.*

- **Function Queries.** *Repeat the following for an arbitrary number of times determined by* A*: In iteration* i*, upon* A *choosing challenge functions* $\{\mathbf{y}_{1,k,i}, \mathbf{y}_{2,k,i}\}_{k\in[m]}$ *in* $\mathcal{R}^{N(\lambda)}$*,* CH *does:*
  - *sample (shared) random string* $r_{K,i}$*; and*
  - *send* A *challenge function keys* $\{\mathsf{sk}_{k,i} \xleftarrow{\$} \mathsf{sIPE.KeyGen}(\mathsf{msk}_k, \mathbf{y}_{1,k,i}, \mathbf{y}_{2,k,i};\ r_{K,i})\}_k$*;*
- **Message Queries:** *Upon* A *choosing challenge messages* $\{(\mathbf{x}^0_{1,k}, \mathbf{x}^0_{2,k}), (\mathbf{x}^1_{1,k}, \mathbf{x}^1_{2,k)}\}_k$ *in* $\mathcal{R}^{N(\lambda)}$*,* CH *does:*
  - *sample (shared) random string* r_E*; and*
  - *send* A *ciphertexts* $\{\mathsf{ct}_k \xleftarrow{\$} \mathsf{sIPE.sEnc}(\mathsf{msk}_k, \mathbf{x}^b_{1,k}, \mathbf{x}^b_{2,k};\ r_E)\}_k$*;*
- **Function Queries Again:** *Repeat the "function queries" stage above.*
- *Finally* A *outputs a bit* b'*.*

Restriction: *For every* $k \in [m]$*, every function query* $\mathbf{y}_{1,k,i}, \mathbf{y}_{2,k,i}$ *from* A *must satisfy that* $(\mathbf{y}_{1,k,i}||\mathbf{y}_{2,k,i}) \cdot (\mathbf{x}^0_{1,k}||\mathbf{x}^0_{1,k}) = (\mathbf{y}_{1,k,i}||\mathbf{y}_{2,k,i}) \cdot (\mathbf{x}^1_{1,k}||\mathbf{x}^1_{2,k})$*.*

Input Indistinguishability in Normal Mode. We note that the above input indistinguishability directly implies that in the normal public key mode, where $\mathbf{x}_{2,k}$ and $\mathbf{y}_{2,k,i}$ are all set to 0^N, the usual input indistinguishability property of public key functional encryption schemes holds.

Second-Slot Function Hiding One can encrypt any value in the first slot using the master public key, whereas can only encrypt in the second slot using the master secret key. Therefore, the second slot acts like a secret key IPE. We require that the stronger function hiding property to hold w.r.t. the second slot, (again) in the multi-instance setting.

Definition 21 (Multi-Instance Second Slot Function Hiding)**.** *An output-encoded slotted IPE scheme* sIPE^N *for computing length-*N *inner products has* μ-multi-instance second-slot function hiding *w.r.t.* $\mathcal{MG}$*, if for every* PPT *adversary* A*, the following games* $\mathsf{SdFH}^{\mathsf{sIPE}}_{A,\mathcal{MG}}(1^\lambda, 0)$ *and* $\mathsf{SdFH}^{\mathsf{sIPE}}_{A,\mathcal{MG}}(1^\lambda, 1)$ *are* μ*-indistinguishable.*

$\mathsf{SdFH}^{\mathsf{sIPE}}_{A,\mathcal{MG}}(1^\lambda, b)$ *proceeds as follow:*

- **Public Parameter and Key Generation.** *The same as in experiment* $\mathsf{Exp}^{\mathsf{sIPE}}_{A,\mathcal{MG}}(1^\lambda, b)$ *of Definition 20.*
- **Function Queries.** *Repeat the following for an arbitrary number of times determined by* A*: In iteration* i*, upon* A *choosing challenge functions* $\{\mathbf{y}_{1,k,i}, \mathbf{y}^0_{2,k,i}, \mathbf{y}^1_{2,k,i}\}_k$ *in* $\mathcal{R}^{N(\lambda)}$*,* CH *does:*
  - *sample (shared) random string* $r_{K,i}$*; and*
  - *send* A *challenge function keys* $\mathsf{sk}_{k,i} \xleftarrow{\$} \mathsf{sIPE.KeyGen}(\mathsf{msk}_k, \mathbf{y}_{1,k,i}, \mathbf{y}^b_{2,k,i}\ ;\ r_{K,i})$*;*

- **Message Queries:** *Upon A choosing challenge messages $\{\mathbf{x}_{1,k}, \mathbf{x}^0_{2,k}, \mathbf{x}^1_{2,k}\}_k$ in $\mathcal{R}^{N(\lambda)}$, CH does:*
  - *sample (shared) random string r_E, and*
  - *send A challenge ciphertexts $\{\mathsf{ct}_k \xleftarrow{\$} \mathsf{sIPE.sEnc}(\mathsf{msk}_k, \mathbf{x}_{1,k}, \mathbf{x}^b_{2,k}\ ;\ r_E)\}_k$.*
- **Function Queries Again:** *Repeat the "function queries" stage above.*
- *Finally A outputs a bit b'.*

Restriction: *For every $k \in [m]$, every function query $\mathbf{y}_{1,k,i}, \mathbf{y}^0_{2,k,i}, \mathbf{y}^1_{2,k,i}$ from A must satisfy that $\mathbf{y}^0_{2,k,i} \cdot \mathbf{x}^0_{2,k} = \mathbf{y}^1_{2,k,i} \cdot \mathbf{x}^1_{2,k}$.*

Joint Indistinguishability We show that combining input indistinguishability and second-slot function hiding gives a stronger security property, called *joint indistinguishability*, defined below. We will use this security property in our construction of FE schemes later.

Definition 22 (Multi-Instance Joint Indistinguishability). *An output-encoded slotted IPE scheme sIPE^N for computing length-N inner products satisfies μ-*multi-instance joint-indistinguishability *w.r.t. $\mathcal{MG}$, if for every* PPT *adversary A, the following games $\mathsf{JInd}^{\mathsf{sIPE}}_{A,\mathcal{MG}}(1^\lambda, 0)$ and $\mathsf{JInd}^{\mathsf{sIPE}}_{A,\mathcal{MG}}(1^\lambda, 1)$ are μ-indistinguishable.*

$\mathsf{JInd}^{\mathsf{sIPE}}_{A,\mathcal{MG}}(1^\lambda, b)$ *proceeds as follow:*

- **Public Parameter and Key Generation.** *The same as in experiment $\mathsf{Exp}^{\mathsf{sIPE}}_{A,\mathcal{MG}}(1^\lambda, b)$ of Definition 20.*
- **Function Queries.** *Repeat the following for an arbitrary number of times determined by A: In iteration i, upon A choosing challenge functions $\{\mathbf{y}_{1,k,i}, \mathbf{y}^0_{2,k,i}, \mathbf{y}^1_{2,k,i}\}_k$ in $\mathcal{R}^{N(\lambda)}$, CH does:*
  - *sample (shared) random string $r_{K,i}$, and*
  - *send A challenge function keys $\{\mathsf{sk}_{k,i} \xleftarrow{\$} \mathsf{sIPE.KeyGen}(\mathsf{msk}_k, \mathbf{y}_{1,k,i}, \mathbf{y}^b_{2,k,i};\ r_{K,i})\}_k$.*
- **Message Queries:** *Upon A choosing challenge messages $\{(\mathbf{x}^0_{1,k}, \mathbf{x}^0_{2,k} = 0^N), (\mathbf{x}^1_{1,k} = 0^N, \mathbf{x}^1_{2,k})\}_k$ in $\mathcal{R}^{N(\lambda)}$, CH does:*
  - *sample (shared) random string r_E, and*
  - *send A challenge ciphertext $\{\mathsf{ct}_k \xleftarrow{\$} \mathsf{sIPE.sEnc}(\mathsf{msk}_k, \mathbf{x}^b_{1,k}, \mathbf{x}^b_{2,k};\ r_E)\}_k$.*
- **Function Queries Again:** *Repeat the "function queries" stage above.*
- *Finally A outputs a bit b'.*

Restriction: *For every $k \in [m]$, every function query $\mathbf{y}_{1,k,i}, \mathbf{y}^0_{2,k,i}, \mathbf{y}^1_{2,k,i}$ from A must satisfy that $\mathbf{y}_{1,k,i} \cdot \mathbf{x}^0_{1,k} = \mathbf{y}^1_{2,k,i} \cdot \mathbf{x}^1_{2,k}$. (Note that $\mathbf{x}^0_{2,k} = \mathbf{x}^1_{1,k} = 0^N$)*

Lemma 7. *Let $\mathcal{MG}$ be a multi-parameter generator and N any polynomial. If an output-encoded slotted IPE scheme sIPE^N satisfies both multi-instance input indistinguishability and second-slot function hiding w.r.t. $\mathcal{MG}$, then it also satisfies multi-instance joint indistinguishability w.r.t. $\mathcal{MG}$.*

Proof of Lemma 2. Fix any PPT adversary A, and security parameter λ. Let $m = m(\lambda)$ be an upper bound on the number of public parameters sampled by $\mathcal{MG}$. We show the indistinguishability of experiments $\mathsf{JInd}^{\mathsf{sIPE}}_{A,\mathcal{MG}}(1^\lambda, 0)$ and $\mathsf{JInd}^{\mathsf{sIPE}}_{A,\mathcal{MG}}(1^\lambda, 1)$ through a sequence of hybrids H_0 to H_3, where the former is identical to $\mathsf{JInd}^{\mathsf{sIPE}}_{A,\mathcal{MG}}(1^\lambda, 0)$ and the latter to $\mathsf{JInd}^{\mathsf{sIPE}}_{A,\mathcal{MG}}(1^\lambda, 1)$. Recall that

Hybrid H_1: H_1 proceeds identically to H_0 except that,

- upon A choosing a function query $\{\mathbf{y}_{1,k,i}, \mathbf{y}^0_{1,k,i}, \mathbf{y}^1_{2,k,i}\}_{k\in[m]}$, send A function keys $\{\mathsf{sk}_{k,i}\}_k$ that encode vector $(\mathbf{y}_{1,k,i}, \mathbf{y}_{1,k,i})$ as opposed to $(\mathbf{y}_{1,k,i}, \mathbf{y}^0_{2,k,i})$ as in H_0.

Note that because the only difference between H_0 and H_1 is the vectors encoded in the second slot of the function keys, and it holds that:

$$\forall k, i,\ 0^N \cdot \mathbf{y}^0_{2,k,i} = 0^N \cdot \mathbf{y}_{1,k,i}$$

It follows from the multi-instance second-slot function hiding property of sIPE that H_0 and H_1 are indistinguishable.

Hybrid H_2: H_2 proceeds identically to H_1 except that,

- upon A choosing a message query $\{(\mathbf{x}^0_{1,k}, 0^N), (0^N, \mathbf{x}^1_{2,k}\}_{k\in[m]}$, send A ciphertext $\{\mathsf{ct}_k\}_k$ that encodes vector $(0^N, \mathbf{x}^0_{1,k})$ as opposed to $(\mathbf{x}^0_{1,k}, 0^N)$ as in H_1.

Note that because the only difference between H_1 and H_2 is the vector encoded in the ciphertext, and it holds that:

$$\forall k, i,\ (\mathbf{x}^0_{1,k}, 0^N) \cdot (\mathbf{y}_{1,k,i}, \mathbf{y}_{1,k,i}) = (0^N, \mathbf{x}^0_{1,k}) \cdot (\mathbf{y}_{1,k,i}, \mathbf{y}_{1,k,i})$$

It follows from the multi-instance input indistinguishability property of sIPE that H_1 and H_2 are indistinguishable.

Hybrid H_3: H_3 proceeds identically to H_2 except that,

- upon A choosing a function query $\{\mathbf{y}_{1,k,i}, \mathbf{y}^0_{1,k,i}, \mathbf{y}^1_{2,k,i}\}_{k\in[m]}$, send A function keys $\{\mathsf{sk}_{k,i}\}_k$ that encodes vector $(\mathbf{y}_{1,k,i}, \mathbf{y}^1_{2,k,i})$ as opposed to $(\mathbf{y}_{1,k,i}, \mathbf{y}_{1,k,i})$ as in H_2.
- upon A choosing a message query $\{(\mathbf{x}^0_{1,k}, 0^N), (0^N, \mathbf{x}^1_{2,k}\}_{k\in[m]}$, send A ciphertext $\{\mathsf{ct}_k\}_k$ that encodes vector $(0^N, \mathbf{x}^1_{2,k})$ as opposed to $(0^N, \mathbf{x}^0_{1,k})$ as in H_2.

Note that because the only difference between H_2 and H_3 is the vectors encoded in the second slots of the function keys and ciphertext, and it holds that:

$$\forall k, i,\ \mathbf{x}^0_{1,k} \cdot \mathbf{y}_{1,k,i} = \mathbf{x}^1_{2,k} \cdot \mathbf{y}^1_{2,k,i}$$

It follows from the multi-instance second-slot function hiding property of sIPE that H_2 and H_3 are indistinguishable.

□

6.4 Our Output-Encoded Slotted-IPE

We construct output-encoded slotted-IPE schemes sIPE^N using asymmetric bilinear groups. Scheme sIPE^N combines our function hiding secret-key IPE scheme skIPE^N in Section 5.7 and the ABDP public-key IPE scheme IPE^M for some polynomial M related to N that is specified below. We in-line the construction with correctness analysis in italic font. Let $\mathsf{pp} = (p, G_0, G_1, G_T, \mathbf{pair})$ be a public parameter that describes asymmetric bilinear groups.

- **Setup:** $\mathsf{slPE.Setup}(1^\lambda, \mathsf{pp})$ samples
  - *first-slot secret key* $\widehat{\mathsf{msk}} = (\widehat{B}, \widehat{B}^\star, \widehat{D}, \widehat{D}^\star)$ where $(\widehat{B}, \widehat{B}^\star) \xleftarrow{\$} \mathrm{Dual}(\mathbb{Z}_p^{4N})$ and $(\widehat{D}, \widehat{D}^\star) \xleftarrow{\$} \mathrm{Dual}(\mathbb{Z}_p^2)$ as $\mathsf{skIPE.Setup}(1^\lambda)$ samples.
  - *second-slot secret key* $\widetilde{\mathsf{msk}} = (\widetilde{B}, \widetilde{B}^\star, \widetilde{D}, \widetilde{D}^\star)$ sampled as above.
  - *first-component* IPE *keys* $(\mathsf{mpk}_1, \mathsf{msk}_1)$ where $\mathbf{s}_1 \xleftarrow{\$} \mathbb{Z}_p^N$, $\mathsf{msk}_1 = \mathbf{s}_1$ and $\mathsf{mpk}_1 = [\mathbf{s}_1]_0$ as $\mathsf{IPE.Setup}(1^\lambda)$ samples.
  - *second-component* IPE *keys* $(\mathsf{mpk}_2, \mathsf{msk}_2)$ where $\mathbf{s}_2 \xleftarrow{\$} \mathbb{Z}_p^N$, and $\mathsf{msk}_2 = \mathbf{s}_2$ and $\mathsf{mpk}_2 = [\mathbf{s}_2]_0$ as above.
  - *random scalars* $r_E = \alpha^\star, \beta^\star \xleftarrow{\$} \mathbb{Z}_p^2$.
  - *first-slot seed ciphertexts* $\{\widehat{\mathsf{ct}}_i \xleftarrow{\$} \mathsf{skIPE.Enc}(\widehat{\mathsf{msk}}, \mathbf{u}_i;\ \alpha^\star, \beta^\star)\}_{i \in [M]}$ for all "unit" vectors $\mathbf{u}_i \in \mathbb{Z}_p^N$ that has 1 at the i^{th} location and 0 everywhere else.

  It sets $\mathsf{sl.msk} = (\widehat{\mathsf{msk}}, \widetilde{\mathsf{msk}}, \mathsf{msk}_1, \mathsf{msk}_2, \alpha^\star, \beta^\star)$, $\mathsf{sl.mpk} = (\mathsf{mpk}_1, \mathsf{mpk}_2, \{\widehat{\mathsf{ct}}_i\}_{i \in [M]})$.

- **Key Generation:** $\mathsf{slPE.KeyGen}(\mathsf{sl.msk}, \mathbf{y}_1, \mathbf{y}_2)$ on input the master secret key $\mathsf{sl.msk}$ and vectors $\mathbf{y}_1, \mathbf{y}_2 \in \mathbb{Z}_p^N$ samples
  - *random scalars* $r_K = (\alpha, \beta) \xleftarrow{\$} \mathbb{Z}_p^2$*;*
  - *first-slot and second-slot function keys of* skIPE*,* $\widehat{\mathsf{sk}} \xleftarrow{\$} \mathsf{skIPE.KeyGen}(\widehat{\mathsf{msk}}, \mathbf{y}_1;\ \alpha, \beta)$ and $\widetilde{\mathsf{sk}} \xleftarrow{\$} \mathsf{skIPE.KeyGen}(\widetilde{\mathsf{msk}}, \mathbf{y}_2;\ \alpha, \beta)$, where

	FIRST COMPONENTS	SECOND COMPONENTS
FIRST SLOT	$\widehat{\mathsf{sk}}[1] = [\widehat{\mathbf{c}}]_1 = [\mathsf{dE}(\widehat{B}, \mathbf{y}_1 \| 0^N; \alpha, \beta)]_1$	$\widehat{\mathsf{sk}}[2] = [\widehat{\mathbf{e}}]_1 = [\mathsf{dE}(\widehat{D}, 1; \alpha, \beta)]_1$
SECOND SLOT	$\widetilde{\mathsf{sk}}[1] = [\widetilde{\mathbf{c}}]_1 = [\mathsf{dE}(\widetilde{B}, \mathbf{y}_2 \| 0^N; \alpha, \beta)]_1$	~~$\widetilde{\mathsf{sk}}[2] = [\widetilde{\mathbf{e}}]_1 = [\mathsf{dE}(\widetilde{D}, 1; \alpha, \beta)]_1$~~

    Discard $\widetilde{\mathsf{sk}}[2]$.
    Note: By the construction of skIPE^N*,* $\widehat{\mathbf{c}}, \widetilde{\mathbf{c}} \in \mathbb{Z}_p^{4N}$ *and* $\widehat{\mathbf{e}} \in \mathbb{Z}_p^2$*.*
  - *first-components and second-components function keys of* IPE*,* $[\mathsf{sk}_1]_1 \xleftarrow{\$} \mathsf{IPE.KeyGen}(\mathsf{msk}_1, [\widehat{\mathbf{c}}]_1 \| [\widetilde{\mathbf{c}}]_1)$ and $[\mathsf{sk}_2]_1 \xleftarrow{\$} \mathsf{IPE.KeyGen}(\mathsf{msk}_2, [\widehat{\mathbf{e}}]_1)$.
    We note that the key generation can be done despite of the fact that the vectors $\widehat{\mathbf{c}} \| \widetilde{\mathbf{c}}$ and $\widehat{\mathbf{e}}$ are already encoded in G_1 is because the key generation algorithm $\mathsf{IPE.KeyGen}$ is linear in the encoded vector (see Remark 2) and we can homomorphically compute linear functions over encodings in G_1.

    $$[\mathsf{sk}_1]_1 = [(\mathbf{s}_1 \cdot (\widehat{\mathbf{c}} \| \widetilde{\mathbf{c}}))\ \|\ (\widehat{\mathbf{c}} \| \widetilde{\mathbf{c}})]_1 \qquad [\mathsf{sk}_2]_1 = [(\mathbf{s}_2 \cdot \widehat{\mathbf{e}})\ \|\ \widehat{\mathbf{e}}]_1$$

    Note: Since $\widehat{\mathbf{c}} \| \widetilde{\mathbf{c}} \in \mathbb{Z}_p^{8N}$ *and* $\widehat{\mathbf{e}} \in \mathbb{Z}_p^2$*, it suffices to set the length of the vectors that* IPE *handles to* $M = 8N$*.*

  It outputs $\mathsf{sl.sk} = ([\mathsf{sk}_1]_1, [\mathsf{sk}_2]_1)$.

- **Secret Encryption:** $\mathsf{slPE.sEnc}(\mathsf{msk}, \mathbf{x}_1, \mathbf{x}_2)$ on input the master secret key $\mathsf{sl.msk}$ and vectors $\mathbf{x}_1, \mathbf{x}_2 \in \mathbb{Z}_p^N$ does:

- Set random scalars $r_E = r_S = (\alpha^\star, \beta^\star)$;
- Generate the *first-slot and second-slot ciphertexts of* skIPE*:* $\widehat{\mathsf{ct}}[1] = \oplus_{i\in[N]}(x_i \odot \widehat{\mathsf{ct}}_i[1])$ where $\{\widehat{\mathsf{ct}}_i\}$ are the ciphertexts in $\mathsf{sl.mpk}$, $\widehat{\mathsf{ct}}[2] = \widehat{\mathsf{ct}}_1[2]$, and $\widetilde{\mathsf{ct}} \xleftarrow{\$} \mathsf{skIPE.Enc}(\widetilde{\mathsf{msk}}, \mathbf{x}_2;\ \alpha^\star, \beta^\star)$. We have (analysis of $\widehat{\mathsf{ct}}$ is below):

	FIRST COMPONENTS	SECOND COMPONENTS
FIRST SLOT	$\widehat{\mathsf{ct}}[1] = \oplus_{i\in[N]}(x_{1,i} \odot \widehat{\mathsf{ct}}_i[1])$ $= [\widehat{\mathbf{c}}^\star]_0 = [\mathsf{dE}(\widehat{B}^\star, \mathbf{x}_1\|\|0^N; \alpha^\star, \beta^\star)]_0$	$\widehat{\mathsf{ct}}[2] = \widehat{\mathsf{ct}}_1[2]$ $= [\widehat{\mathbf{e}}^\star]_0 = [\mathsf{dE}(\widehat{D}^\star, 1; \alpha^\star, \beta^\star)]_0$
SECOND SLOT	$\widetilde{\mathsf{ct}}[1] = [\widetilde{\mathbf{c}}^\star]_0 = [\mathsf{dE}(\widetilde{B}^\star, \mathbf{x}_2\|\|0^N; \alpha^\star, \beta^\star)]_0$	~~$\widetilde{\mathsf{ct}}[2] = [\widetilde{\mathbf{e}}^\star]_0 = [\mathsf{dE}(\widetilde{D}^\star, 1; \alpha^\star, \beta^\star)]_0$~~

Discard $\widetilde{\mathsf{ct}}[2]$.

Analysis of $\widehat{\mathsf{ct}}$*: Recall that* $\{\widehat{\mathsf{ct}}_i\}$ *in* $\mathsf{sl.mpk}$ *are encryption of the unit vectors* $\{\mathbf{u}_i\}$ *sharing the same random scalars* $\alpha^\star, \beta^\star$*.*

$$\widehat{\mathsf{ct}}_i[1] = [\widehat{\mathbf{c}}_i^\star]_0 = [\mathsf{dE}(\widehat{B}^\star, \mathbf{u}_i\|0^N; \alpha^\star, \beta^\star)]_0 \qquad \widehat{\mathsf{ct}}_i[2] = [\widehat{\mathbf{e}}^\star]_0 = [\mathsf{dE}(\widehat{D}^\star, 1; \alpha^\star, \beta^\star)]_0$$

Note that all the second components are identical. Hence, $\widehat{\mathsf{ct}}[2] = [\mathsf{dE}(\widehat{D}^\star, 1; \alpha^\star, \beta^\star)]_0$*.* *On the other hand, expand the computation of* $\widehat{\mathsf{ct}}[1]$*:*

$$\begin{aligned}\widehat{\mathsf{ct}}[1] &= \oplus_{i\in[N]}(x_{1,i} \odot \widehat{\mathsf{ct}}_i[1]) = \oplus_{i\in[N]}(x_{1,i} \odot [\mathsf{dE}(\widehat{B}^\star, \mathbf{u}_i\|0^N; \alpha^\star, \beta^\star)]_0) \\ &= \oplus_{i\in[N]}[\mathsf{dE}(\widehat{B}^\star, x_{1,i}\mathbf{u}_i\|0^N; \alpha^\star, \beta^\star)]_0 = [\mathsf{dE}(\widehat{B}^\star, \Sigma_{i\in[N]}x_{1,i}\mathbf{u}_i\|0^N; \alpha^\star, \beta^\star)]_0 \\ &\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad = [\mathsf{dE}(\widehat{B}^\star, \mathbf{x}_1\|0^N; \alpha^\star, \beta^\star)]_0\end{aligned}$$

where the third and fourth equalities follow from the linearity of the double embedding dE *function (see Remark 1). Therefore* $\widehat{\mathsf{ct}} = \widehat{\mathsf{ct}}[1], \widehat{\mathsf{ct}}[2]$ *is an* skIPE *encryption of* $\mathbf{x}_1$*.*

- Generate *first-components and second-components function encryptions of* IPE*,* $[\mathsf{ct}_1]_0 \xleftarrow{\$} \mathsf{IPE.Enc}(\mathsf{mpk}_1, [\widehat{\mathbf{c}}^\star]_0\|[\widetilde{\mathbf{c}}^\star]_0)$ and $[\mathsf{ct}_2]_0 \xleftarrow{\$} \mathsf{IPE.Enc}(\mathsf{mpk}_2, [\widehat{\mathbf{e}}^\star]_0)$.
  We note that the encryption can be done despite of the fact that the vectors $\widehat{\mathbf{c}}^\star\|\widetilde{\mathbf{c}}^\star$ and $\widehat{\mathbf{e}}^\star$ are already encoded in G_1, because the key generation algorithm $\mathsf{IPE.Enc}$ is multilinear, and it never multiples elements in the master public key with that in the plaintext vectors (see Remark 2).

$$[\mathsf{ct}_1]_0 = [-r_1 \,\|\, (r_1\mathbf{s}_1 + (\widehat{\mathbf{c}}^\star\|\widetilde{\mathbf{c}}^\star))]_0 \qquad\qquad [\mathsf{ct}_2]_0 = [-r_2 \,\|\, (r_2\mathbf{s}_2 + \widehat{\mathbf{e}}^\star)]_0$$

It outputs $\mathsf{sl.ct} = ([\mathsf{ct}_1]_0, [\mathsf{ct}_2]_0)$.

- **Public Encryption:** $\mathsf{sIPE.Enc}(\mathsf{sl.mpk}, \mathbf{x})$ on input the master public key $\mathsf{sl.mpk} = (\mathsf{mpk}_1, \mathsf{mpk}_2, \{\widehat{\mathsf{ct}}_i\})$ and vector $\mathbf{x} \in \mathbb{Z}_p^N$ proceeds identically as $\mathsf{sIPE.sEnc}(\mathsf{sl.msk}, \mathbf{x}, 0^N)$. This can be done since the only secret information used in $\mathsf{sIPE.sEnc}$ is the second slot master secret key $\widetilde{\mathsf{msk}}$ and the scalars $\alpha^\star, \beta^\star$ for generating second slot ciphertext, when the second vector is set to $\mathbf{x}_2 = 0^N$, $\widetilde{\mathsf{ct}}[1] = [\widetilde{\mathbf{c}}^\star]_0 = [0^{4N}]_0$ and $\widetilde{\mathsf{msk}}, \alpha^\star, \beta^\star$ are not needed. All other steps rely only on public information.

  Note: By construction, the output of public encryption is identical to $\mathsf{sIPE.sEnc}(\mathsf{sl.msk}, \mathbf{x}, 0^N)$*.*

Note: The above algorithm requires the capability to encode zero in group G_0. We can modify it so that the algorithm does not need to generate encoding of zero. Since $\widetilde{\mathbf{c}}^\star = 0^N$, instead of encoding it and then homomorphically generate the first-components ciphertext, we can generate the first-components ciphertext as follows:

$$[\mathsf{ct}_1]_0 = [-r_1]_0 \mid\mid [r_1(\mathbf{s}_{1,1\cdots N}) + \widehat{\mathbf{c}}^\star]_0 \mid\mid [r_1(\mathbf{s}_{1,N+1\cdots 2N})]_0$$

The middle part is still produced homomorphically over $\widehat{\mathsf{ct}}[1]$, while the last part can be produced by directly encoding $r_1\mathbf{s}_{1,N+1\cdots 2N}$, which is non-zero with overwhelming probability.

- **Decryption:** $\mathsf{sIPE.Dec}(\mathsf{sI.sk}, \mathsf{sI.ct})$ on input $\mathsf{sI.sk} = ([\mathsf{sk}_1]_1, [\mathsf{sk}_2]_1)$ and $\mathsf{sI.ct} = ([\mathsf{ct}_1]_0, [\mathsf{ct}_2]_0)$ does:
  - Decrypt the first-components and second-components ciphertexts: $[z_b]_T = \mathsf{IPE.Dec}([\mathsf{sk}_b]_1,\ [\mathsf{ct}_b]_0)$ for $b \in \{1, 2\}$. Note that $\mathsf{IPE.Dec}$ can be evaluated despite that sk_b is encoded in G_1 because the algorithm requires taking inner product between sk_b and $[\mathsf{ct}_b]_0$, which can be done homomorphically over $[\mathsf{sk}_b]_1$ using pairing. More specifically

    $$\begin{aligned} [z_1]_T &= [\mathsf{sk}_1]_1 \cdot [\mathsf{ct}_1]_0 = [(\widehat{\mathbf{c}}||\widetilde{\mathbf{c}}) \cdot (\widehat{\mathbf{c}}^\star||\widetilde{\mathbf{c}}^\star)]_T = [\widehat{\mathbf{c}} \cdot \widehat{\mathbf{c}}^\star + \widetilde{\mathbf{c}} \cdot \widetilde{\mathbf{c}}^\star]_T \\ [z_2]_T &= [\mathsf{sk}_2]_1 \cdot [\mathsf{ct}_2]_0 = [\widehat{\mathbf{e}} \cdot \widehat{\mathbf{e}}^\star]_T \end{aligned}$$

    It outputs $[z_1]_T, [z_2]_T$
  - The *output encoding function* is $\mathsf{OEnc}(z, \mathsf{mpk}, r_S, r_K, r_E) = \mathsf{OEnc}(z, \mathsf{mpk}, \alpha, \alpha^\star, \beta, \beta^\star) = [\theta z]_T, [\theta]_T$, where $\theta = \alpha\alpha^\star + \beta\beta^\star$.

Note: Here again if z is a value in a polynomially sized range, then it can be extracted by trying all possible values.

Correctness of Computation: Note that $[\widehat{\mathbf{c}}]_1 \cdot [\widehat{\mathbf{c}}^\star]_0 = [\widehat{\mathbf{c}} \cdot \widehat{\mathbf{c}}^\star]_T$ corresponds exactly to decrypting the first-slot ciphertext with the first-slot function key of skIPE*, and $[\widetilde{\mathbf{c}}]_1 \cdot [\widetilde{\mathbf{c}}^\star]_0 = [\widetilde{\mathbf{c}} \cdot \widetilde{\mathbf{c}}^\star]_T$ corresponds to decrypting the second-slot ciphertext and key. Therefore, this gives*

$$\begin{aligned} [z_1]_T &= [(\alpha\alpha^\star + \beta\beta^\star)(\mathbf{x}_1 \cdot \mathbf{y}_1) + (\alpha\alpha^\star + \beta\beta^\star)(\mathbf{x}_2 \cdot \mathbf{y}_2)]_T = [\theta(\mathbf{x}_1 \cdot \mathbf{y}_1 + \mathbf{x}_2 \cdot \mathbf{y}_2)]_T \\ [z_2]_T &= [(\alpha\alpha^\star + \beta\beta^\star)]_T = [\theta]_T \end{aligned}$$

Thus, for any λ, $\mathbf{x}_1, \mathbf{x}_2, \mathbf{y}_1, \mathbf{y}_2 \in \mathcal{R}^N$, the probability that decryption of sI.sk, sI.ct *yields* $\mathsf{OEnc}((\mathbf{x}_1 \cdot \mathbf{y}_1 + \mathbf{x}_2 \cdot \mathbf{y}_2), \mathsf{mpk}, \alpha, \alpha^\star, \beta, \beta^\star)$ *is 1 (over the randomness of the scalars, the generation of* sI.mpk, sI.msk, sI.sk *and* sI.ct*).*

6.5 Security

We next proceed to analyze the security of the scheme sIPE^N. At a high-level, it follows from the input indistinguishability of the public key IPE scheme IPE^M that sIPE^N also satisfies input indistinguishability, and it follows from the function hiding property of the secret key IPE scheme skIPE^N (and the fact that the second-slot master secret key is completely hidden) that sIPE^N is second-slot function hiding. Note that in sIPE^N, the random coins r_S, r_K, r_E that affect the output encoding are exactly the random scalars $\alpha, \alpha^\star, \beta, \beta^\star$ used by invocations of skIPE^N. On the other hand, the random coins used by invocations of IPE^M are independently and randomly sampled in different instances. Therefore, the input indistinguishability property of sIPE^M holds in the multi-instance setting since all invocations of IPE^M are independent, assuming that the SXDH assumption holds in every asymmetric bilinear group sampled by $\mathcal{MG}$. The second-slot function hiding property also holds in the multi-instance setting because the scheme skIPE^N is multi-instance function hiding, assuming that the joint-SXDH assumption holds in asymmetric bilinear groups $\mathcal{MG}$. Formally,

Lemma 8. *Assume that the joint-SXDH assumption holds in the asymmetric bilinear groups generated by $\mathcal{MG}$. Then, for any polynomial N, the ouput-encoded slotted IPE scheme sIPE^N satisfies multi-instance input indistinguishability w.r.t. $\mathcal{MG}$.*

Lemma 9. *Assume that the joint-SXDH assumption holds in the asymmetric bilinear groups generated by $\mathcal{MG}$. Then, for any polynomial N, the ouput-encoded slotted IPE scheme sIPE^N satisfies multi-instance second-slot function hiding w.r.t. $\mathcal{MG}$.*

As a corollary of Lemma 2, we obtain that sIPE^N also satisfies multi-instance joint indistinguishability.

Corollary 2. *Assume that the joint-SXDH assumption holds in the asymmetric bilinear groups generated by $\mathcal{MG}$. Then, for any polynomial N, the ouput-encoded slotted IPE scheme sIPE^N satisfies multi-instance joint indistinguishability w.r.t. $\mathcal{MG}$.*

Proof of Lemma 8. Fix any PPT adversary A, and security parameter λ. Let $m = m(\lambda)$ be an upper bound on the number of public parameters sampled by $\mathcal{MG}$. We show the indistinguishability of experiments $\mathsf{Int}^{\mathsf{sIPE}}_{A,\mathcal{MG}}(1^\lambda, 0)$ and $\mathsf{Int}^{\mathsf{sIPE}}_{A,\mathcal{MG}}(1^\lambda, 1)$ through a sequence of hybrids H_0 to H_m, where the former is identical to $\mathsf{Int}^{\mathsf{sIPE}}_{A,\mathcal{MG}}(1^\lambda, 0)$ and the latter to $\mathsf{Int}^{\mathsf{sIPE}}_{A,\mathcal{MG}}(1^\lambda, 1)$. By definition of experiment Int, in both H_0 and H_m, A sees m instances of the scheme sIPE^N. In each instance, A receives the master secret key mpk_k, an arbitrary number of function keys $\{\mathsf{sk}_{k,i}\}$ of vectors $\mathbf{y}_{1,k,i}, \mathbf{y}_{2,k,i}$, and a ciphertext ct_k of vectors $\mathbf{x}^0_{1,k}, \mathbf{x}^0_{2,k}$ in H_0 and $\mathbf{x}^1_{1,k}, \mathbf{x}^1_{2,k}$ in H_m. Then, the intermediate hybrid H_j for $j \in [m]$ is defined as follows:

- *Hybrid H_j* proceeds identically to H_0 except that, in the first j instances, A receives ciphertexts ct_k of $\mathbf{x}^1_{1,k}, \mathbf{x}^1_{2,k}$, as opposed to $\mathbf{x}^0_{1,k}, \mathbf{x}^0_{2,k}$, for $k \le j$. The only difference between H_{j-1} and H_j is that the ciphertext ct_j of the j^{th} instance switches from encrypting $\mathbf{x}^0_{1,j}, \mathbf{x}^0_{2,j}$ to encrypting $\mathbf{x}^1_{1,j}, \mathbf{x}^1_{2,j}$.

Next, we reduce the indistinguishability of H_{j-1} and H_i to the (single-instance) input indistinguishability of the ABDP public key IPE scheme IPE. Then, by a hybrid, we obtain the lemma.

Formally, suppose for contradiction that for infinitely many $\lambda \in \mathbb{N}$, A distinguishes H_0 and H_m with advantage $\nu(\lambda)$, then for every such λ, there exists a $j \in [m]$, such that, A distinguishes hybrid H_{j-1} and H_j with advantage $\nu(\lambda)/m(\lambda)$. Then we can build another adversary B that violates the input indistinguishability of IPE when instantiated with the j^{th} public parameter pp_j sampled by $\mathcal{MG}$, which contradicts with the fact the SXDH assumption holds w.r.t. the asymmetric bilinear groups described by pp_j, generated by $\mathcal{MG}$.

The adversary B interacts with the challenger CH in experiment $\mathsf{Exp}^{\mathsf{IPE}}_B(1^\lambda, b)$ for $b \in \{0, 1\}$, and internally emulates the execution of hybrid H_{j-1+b} with A, as follows:

- *Public Parameter Generation:* Public Parameters $(\mathsf{pp}_1, \cdots, \mathsf{pp}_m) \xleftarrow{\$} \mathcal{MG}$ are sampled.

- *Key Generation:* Upon receiving an IPE master public key mpk^* from CH generated w.r.t. the j^{th} public parameter pp_j, B generates the master public and secret keys $\{\mathsf{sl.mpk}_k, \mathsf{sl.msk}_k\}$ of m sIPE instances as follows:
  - sample (shared) random string $r_S = \alpha^\star, \beta^\star$,
  - generate m instances $(\mathsf{sl.mpk}_k, \mathsf{sl.msk}_k) \xleftarrow{\$} \{\mathsf{sIPE.Setup}(1^\lambda, \mathsf{pp}_k; r_S)\}_{k\in[m]}$,
  - replace the first-component IPE master public key of the j^{th} instance with mpk^* from CH, and replace the corresponding first-component IPE master secret key with $\perp$, unknown. That is, $\mathsf{sl.mpk}_j = (\underline{\mathsf{mpk}^*}, \mathsf{mpk}_{2,j}, \{\widehat{\mathsf{ct}}_{l,j}\}_{l\in[M]})$ and $\mathsf{sl.msk}_j = (\widehat{\mathsf{msk}}_j, \widetilde{\mathsf{msk}}_j, \underline{\perp}, \mathsf{msk}_{2,j}, \alpha^\star, \beta^\star)$.

  It sends A master public keys $\{\mathsf{sl.mpk}_k\}$.

- *Function Queries:* Repeat the following for an arbitrary number of times determined by A: In iteration i, upon A choosing challenge functions $\{\mathbf{y}_{1,k,i}, \mathbf{y}_{2,k,i}\}_k$, B does:
  - sample (shared) random string $r_{K,i} = \alpha_i, \beta_i$,
  - for the every instance $k \neq j$, generate function key $\mathsf{sk}_{k,i} \xleftarrow{\$} \mathsf{sIPE.KeyGen}(\mathsf{sl.msk}_k, \mathbf{y}_{1,k,i}, \mathbf{y}_{2,k,i};\ r_{K,i})$.
  - for the j^{th} instance, B is missing the first-component IPE master secret key in $\mathsf{sl.msk}_j = (\widehat{\mathsf{msk}}_j, \widetilde{\mathsf{msk}}_j, \perp, \mathsf{msk}_{2,j}, \alpha^\star, \beta^\star)$. Thus, to generate the secret key $\mathsf{sk}_{j,i}$ of $\mathbf{y}_{1,j,i}, \mathbf{y}_{2,j,i}$, B simulates the steps $\mathsf{sIPE.KeyGen}(\mathsf{sl.msk}_j, \mathbf{y}_{1,j,i}, \mathbf{y}_{2,j,i};\ r_{K,i})$ does, and obtains the first-component secret key using its challenger CH.

    More precisely, B computes first the vectors to be encoded in the first- and second-slot function keys $\widehat{\mathsf{sk}}_{j,i}, \widetilde{\mathsf{sk}}_{j,i}$ of skIPE.

	FIRST COMPONENTS	SECOND COMPONENTS
FIRST SLOT	$\widehat{\mathbf{c}}_{j,i} = \mathsf{dE}(\widehat{B}_j, \mathbf{y}_{1,j,i}\|\|0^N; \alpha_i, \beta_i)$	$\widehat{\mathbf{e}}_{j,i} = \mathsf{dE}(\widehat{D}_j, 1; \alpha_i, \beta_i)$
SECOND SLOT	$\widetilde{\mathbf{c}}_{j,i} = \mathsf{dE}(\widetilde{B}_j, \mathbf{y}_{2,j,i}\|\|0^N; \alpha_i, \beta_i)$	

    where $\widehat{\mathsf{msk}}_j = (\widehat{B}_j, \widehat{B}^\star_j, \widehat{D}_j, \widehat{D}^\star_j)$ and $\widetilde{\mathsf{msk}}_j = (\widetilde{B}_j, \widetilde{B}^\star_j, \widetilde{D}_j, \widetilde{D}^\star_j)$.

    To obtain the first-component IPE function key under key mpk^*, B sends CH the function query $\mathbf{y}^*_i = \widehat{\mathbf{c}}_{j,i}||\widetilde{\mathbf{c}}_{j,i}$ and receives back an IPE function key sk^*_i of $\mathbf{y}^*_i$ under mpk^*. It sets the first-component function key to $[\mathsf{sk}_{1,j,i}]_{j,1} = [\mathsf{sk}^*_i]_{j,1}$ and generates the second-component function key itself $[\mathsf{sk}_{2,j,i}]_{j,1} \xleftarrow{\$} \mathsf{IPE.KeyGen}(\mathsf{msk}_{2,j}, [\widehat{\mathbf{e}}_{j,i}]_{j,1})$.

    This yields the function key $\mathsf{sk}_{j,i} = [\mathsf{sk}^*_i]_{j,1}, [\mathsf{sk}_{2,j,i}]_{j,1}$.

  It sends A function keys $\{\mathsf{sk}_{k,i}\}$.

- *Message Queries:* Upon A choosing challenge messages $\{\mathbf{x}^0_{1,k}, \mathbf{x}^0_{1,k}, \mathbf{x}^1_{2,k}, \mathbf{x}^1_{1,k}\}_k$, B does:
  - (the (shared) random string r_E is empty.)
  - for every instance $k < j$, generate ciphertext $\mathsf{ct}_k \xleftarrow{\$} \mathsf{sIPE.sEnc}(\mathsf{sl.msk}_k, \underline{\mathbf{x}^1_{1,k}, \mathbf{x}^1_{2,k}})$.
  - for every instance $k > j$, generate ciphertext $\mathsf{ct}_k \xleftarrow{\$} \mathsf{sIPE.sEnc}(\mathsf{sl.msk}_k, \underline{\mathbf{x}^0_{1,k}, \mathbf{x}^0_{2,k}})$.
  - for the j^{th} instance, B generates the ciphertext ct_j of $\underline{\mathbf{x}^b_{1,j}, \mathbf{x}^b_{2,j}}$, by simulating the steps $\mathsf{sIPE.sEnc}(\mathsf{sl.msk}_j, \mathbf{x}^b_{1,j}, \mathbf{x}^b_{2,j})$ does, obtaining the first-component ciphertext using CH.

More precisely, B computes first the vectors to be encoded in the skIPE first- and second-slot ciphertexts $\widehat{\mathsf{ct}}^b_j, \widetilde{\mathsf{ct}}^b_j$ of vectors $\mathbf{x}^b_{1,j}, \mathbf{x}^b_{2,j}$ for both $b = 0, 1$.

	FIRST COMPONENTS	SECOND COMPONENTS
FIRST SLOT	$\widehat{\mathbf{c}}^{\star b}_j = \mathsf{dE}(\widehat{B}^\star_j, \mathbf{x}^b_{1,j} \| 0^N; \alpha^\star_j, \beta^\star_j)$	$\widehat{\mathbf{e}}^\star_j = \mathsf{dE}(\widehat{D}^\star_j, 1; \alpha^\star_j, \beta^\star_j)$
SECOND SLOT	$\widetilde{\mathbf{c}}^{\star b}_j = \mathsf{dE}(\widetilde{B}^\star_j, \mathbf{x}^b_{2,j} \| 0^N; \alpha^\star_j, \beta^\star_j)$	

To obtain the first-component IPE ciphertext under key mpk^*, B sends CH the message query $\mathbf{x}^{*,0} = \widehat{\mathbf{c}}^{\star 0}_j \| \widetilde{\mathbf{c}}^{\star 0}_j$ and $\mathbf{x}^{*,1} = \widehat{\mathbf{c}}^{\star 1}_j \| \widetilde{\mathbf{c}}^{\star 1}_j$, and receives back an IPE ciphertext ct^* of $\mathbf{x}^{*,b}$ under mpk^*. It sets the first-component function key to $[\mathsf{ct}_{1,j}]_{j,0} = [\mathsf{ct}^*]_{j,0}$ and generates the second-component ciphertext itself $[\mathsf{ct}_{2,j}]_{j,0} \xleftarrow{\$} \mathsf{IPE.Enc}(\mathsf{mpk}_{2,j}, [\widehat{\mathbf{e}}^\star_j]_{j,0})$.
This yields the function key $\mathsf{ct}_j = [\mathsf{ct}^*]_{j,0}, [\mathsf{ct}_{2,j}]_{j,0}$.

It sends A challenge ciphertext $\{\mathsf{ct}_k\}$.

- *Function Queries Again:* Repeat the "function queries" stage above.
- Finally B outputs what A outputs.

We argue that when B participates in $\mathsf{Exp}^{\mathsf{IPE}}_B(1^\lambda, b)$, it internally emulates for A hybrid H_{j-1+b} perfectly. By construction, all instances other than j are generated exactly as the challenger in both hybrid H_{j-1} and H_i does. Thus it suffices to show that B emulates the j^{th} instance perfectly for A. B embeds the external IPE master public key mpk^* as the first-component public key in the j^{th} instance, which is correctly distributed. Then to generate the first-component function key and ciphertexts correctly, B generates the vectors $\{\widehat{\mathbf{c}}_{j,i}, \widetilde{\mathbf{c}}_{j,i}\}$ and $\{\widehat{\mathbf{c}}^{\star b}_j, \widetilde{\mathbf{c}}^{\star b}_j\}$ to be encoded in the first- and second-slot function keys and ciphertexts of skIPE correctly. In the key generation and encryption algorithms $\mathsf{sIPE.KeyGen}$ and $\mathsf{sIPE.sEnc}$, the first-component function key and ciphertext are evaluated homomorphically over the first- and second-slot function keys and ciphertexts, which are $[\mathbf{y}^*_i]_{j,1}$ for $\mathbf{y}^*_i = \widehat{\mathbf{c}}_{j,i} \| \widetilde{\mathbf{c}}_{j,i}$ and $[\mathbf{x}^{*,b}]_{j,0}$ for $\mathbf{x}^{*,b} = \widehat{\mathbf{c}}^{\star b}_j \| \widetilde{\mathbf{c}}^{\star b}_j$.

$$[\mathsf{sk}^*_i]_{j,1} \xleftarrow{\$} \mathsf{IPE.KeyGen}(\mathsf{msk}^*, [\mathbf{y}^*_i]_{j,1}) \qquad [\mathsf{ct}^{*,b}]_{j,0} \xleftarrow{\$} \mathsf{IPE.Enc}(\mathsf{mpk}^*, [\mathbf{x}^{*,b}]_{j,0})$$

Instead, B sends the plaintext vectors $\{\mathbf{y}^*_i\}$ and $\mathbf{x}^{*,0}, \mathbf{x}^{*,1}$ as function and message queries to its challenger, obtaining $\{\mathsf{sk}^*_i\}$ and $\mathsf{ct}^{*,b}$ in the plaintext. It then internally encodes them to obtain $\{[\mathsf{sk}^*_i]_{j,1}\}$ and $[\mathsf{ct}^{*,b}]_{j,0}$. Therefore, it generates the first-component function key and ciphertext perfectly as in $\mathsf{sIPE.KeyGen}$ and $\mathsf{sIPE.sEnc}$. Furthermore, B also generates the second-component function key and ciphertext perfectly. Overall, B emulates the generation of the j^{th} instance perfectly. This concludes the lemma. □

Remark 3. *We remark that in the proof above, we rely on the fact that for any vector* $\mathbf{x}$ *and linear function* L*. The following two procedures produce the same encoding: 1) generate the encoding of* $\mathbf{x}$ *first in some group* G_l*, and then apply the linear function* L *homomorphically over the encodings and 2) evaluate* $L(\mathbf{x})$ *first, and then encode the output in* G_l*. That is,* $[L(\mathbf{x})]_l = L([\mathbf{x}]_l)$*. This is true in asymmetric bilinear groups since encodings of any element is unique.*

We stress that the property of unique encoding is sufficient but not necessary. Even if encodings are not unique, as long as they satisfy that the distribution of $[L(\mathbf{x})]_l$ *and that of* $L([\mathbf{x}]_l)$ *are computationally indistinguishable, the above proof still go through, as well as the security proof of the ABDP public key IPE* IPE *and our secret key IPE* skIPE*. Thus, the above lemmas still hold. As we will see later, this fact helps us to instantiate the slotted IPE scheme* sIPE *using graded encoding schemes with noisy encodings.*

Proof of Lemma 9. Suppose for contradiction that there exists a PPT adversary A, that for infinitely many $\lambda \in \mathbb{N}$ distinguishes $\mathsf{SdFH}^{\mathsf{sIPE}}_{A,\mathcal{MG}}(1^\lambda, 0)$ and $\mathsf{SdFH}^{\mathsf{sIPE}}_{A,\mathcal{MG}}(1^\lambda, 1)$ with advantage $\nu(\lambda)$. Then we show that there is a another adversary B that violates the multi-instance function hiding property of skIPE w.r.t. a related group generator $\mathcal{MG}^*$.

- $\mathcal{MG}^*(1^\lambda)$ samples $(\mathsf{pp}_1, \cdots, \mathsf{pp}_m) \xleftarrow{\$} \mathcal{MG}$ outputs $(\mathsf{pp}_1, \cdots, \mathsf{pp}_m), (\mathsf{pp}_1, \cdots, \mathsf{pp}_m)$.

By that joint-SXDH holds w.r.t. $\mathcal{MG}$, joint-SXDH also holds w.r.t. $\mathcal{MG}^*$. The first m public parameters corresponds to the second-slot skIPE function keys and ciphertexts in the m instances, while the second m parameters corresponds to the first-slot skIPE function keys and ciphertexts.

Fix a security parameter λ. The adversary B interacts with the challenger CH in experiment $\mathsf{MFH}^{\mathsf{skIPE}}_{B,\mathcal{MG}^*}(1^\lambda, b)$ for $b \in \{0, 1\}$, and internally emulates the execution of $\mathsf{SdFH}^{\mathsf{sIPE}}_{A,\mathcal{MG}}(1^\lambda, b)$ as follows. Steps that belong to the MFH experiment are underlined.

- *Public Parameter Generation:* Public Parameters $\underline{(\mathsf{pp}_1, \cdots, \mathsf{pp}_m), (\mathsf{pp}_1, \cdots, \mathsf{pp}_m) \xleftarrow{\$} \mathcal{MG}^*}$ are sampled in the MFH experiment. B uses $(\mathsf{pp}_1, \cdots, \mathsf{pp}_m)$ as the public parameters for A.

- *Key Generation:* $\underline{\text{The challenger } CH \text{ samples } 2m \text{ master secret keys of skIPE.}}$ The the first m corresponds to the m second-slot master secret keys $\widetilde{\mathsf{msk}}_1, \cdots \widetilde{\mathsf{msk}}_m$, and second m corresponds to the m first-slot master secret keys $\widehat{\mathsf{msk}}_1, \cdots \widehat{\mathsf{msk}}_m$, in the m instances of sIPE emulated by B internally for A.

  B receives nothing from CH, and needs to emulate for A the m sIPE master public keys $\{\mathsf{sI.mpk}_k\}$, where each $\mathsf{sI.mpk}_k = (\mathsf{mpk}_{1,k}, \mathsf{mpk}_{2,k}, \{\widehat{\mathsf{ct}}_{l,k}\})$. It does:

  – Generate the first-component and second-component keys $(\mathsf{mpk}_{1,k}, \mathsf{msk}_{1,k}) \xleftarrow{\$} \mathsf{IPE.Setup}(1^\lambda)$ and $(\mathsf{mpk}_{2,k}, \mathsf{msk}_{2,k}) \xleftarrow{\$} \mathsf{IPE.Setup}(1^\lambda)$.

  – For every $k \in [m]$, to obtain the first-slot ciphertexts $\{\widehat{\mathsf{ct}}_{l,k}\}_l$ of skIPE that encrypt the unit vectors $\{\mathbf{u}_l\}$, $\underline{B \text{ sends query } (\mathsf{aux\text{-}mesg}\ \{\mathbf{u}_l\}, m + k, 1)}$ to its challenger CH, who replies with ciphertexts $\{\widehat{\mathsf{ct}}_{l,k} \xleftarrow{\$} \mathsf{skIPE.Enc}(\widehat{\mathsf{msk}}_k, \mathbf{u}_l; \alpha_1^\star, \beta_1^\star)\}_l$, where $\alpha_1^\star, \beta_1^\star$ are the shared randomness corresponding to $r_1^\star$ (in the MFH experiment).

  B sends A master public keys $\{\mathsf{sI.mpk}_k\}$.

- *Function Queries:* Repeat the following for an arbitrary number of times determined by A: In iteration i, upon A choosing challenge functions $\{\mathbf{y}_{1,k,i}, \mathbf{y}^0_{2,k,i}, \mathbf{y}^1_{2,k,i}\}_{k\in[m]}$, B needs to generate the sIPE function keys $\{\mathsf{sk}_{k,i}\}_k$ of vector $\{\mathbf{y}_{1,k,i}, \mathbf{y}^b_{2,k,i}\}_k$ for A. For every k, it does the following:

  – To generate the first-slot function key $\widehat{\mathsf{sk}}_{k,i}$, $\underline{B \text{ sends query } (\mathsf{aux\text{-}func}\ \mathbf{y}_{1,k,i}, m + k, i)}$ to its challenger CH, who replies with function key $\widehat{\mathsf{sk}}_{k,i} \xleftarrow{\$} \mathsf{skIPE.KeyGen}(\widehat{\mathsf{msk}}_k, \mathbf{y}_{1,k,i}; \alpha_i, \beta_i)$, where α_i, β_i are the shared randomness corresponding to r_i.

  – To generate the second-slot function key $\widetilde{\mathsf{sk}}_{k,i}$, $\underline{B \text{ sends query } (\mathsf{func}\ (\mathbf{y}^0_{2,k,i} \text{ or } \mathbf{y}^1_{2,k,i}), k, i)}$ to its challenger CH, who replies with function key $\widetilde{\mathsf{sk}}_{k,i} \xleftarrow{\$} \mathsf{skIPE.KeyGen}(\widetilde{\mathsf{msk}}_k, \mathbf{y}^b_{2,k,i}; \alpha_i, \beta_i)$.

  – Complete the key generation by computing the first- and second-component function keys using $\mathsf{msk}_{1,k}$ and $\mathsf{msk}_{2,k}$ as in $\mathsf{sIPE.KeyGen}$, obtaining $\mathsf{sk}_{k,i}$.

  Send A function keys $\{\mathsf{sk}_{k,i}\}_k$.

- *Message Queries:* Upon A choosing challenge messages $\{\mathbf{x}_{1,k}, \mathbf{x}^0_{2,k}, \mathbf{x}^1_{1,k}\}_k$, B needs to generate the sIPE ciphertexts $\{\mathsf{ct}_k\}_k$ of vector $\{\mathbf{x}_{1,k}, \mathbf{x}^b_{2,k}\}_k$ for A. For every k, it does the following:
  - Generate the first-slot ciphertext $\widehat{\mathsf{ct}}_k$ of vector $\mathbf{x}_{1,k}$ using $\{\widehat{\mathsf{ct}}_{l,k}\}_k$ exactly as in sIPE.Enc.
  - To generate the second-slot ciphertext $\widetilde{\mathsf{ct}}_k$, B sends query $(\mathsf{mesg}\ (\mathbf{x}^0_{2,k} \text{ or } \mathbf{x}^1_{2,k}), k, 1)$ to its challenger CH, who replies with ciphertext $\widetilde{\mathsf{ct}}_k \xleftarrow{\$} \mathsf{skIPE.KeyGen}(\widetilde{\mathsf{msk}}_k, \mathbf{x}^b_{2,k}; \alpha^\star_1, \beta^\star_1)$ (using random scalars $\alpha^\star_1, \beta^\star_1$ corresponding to $r^\star_1$.)
  - Complete the key generation by computing the first- and second-component ciphertexts using $\mathsf{mpk}_{1,k}$ and $\mathsf{mpk}_{2,k}$ as in sIPE.Enc, obtaining ct_k.

  Send A ciphertexts $\{\mathsf{ct}_k\}_k$.

- *Function Queries Again:* Repeat the "function queries" stage above.

- Finally B outputs what A outputs.

We argue that when B participates in $\mathsf{MFH}^{\mathsf{skIPE}}_{B,\mathcal{MG}^*}(1^\lambda, b)$, it internally emulates $\mathsf{SdFH}^{\mathsf{sIPE}}_{A,\mathcal{MG}}(1^\lambda, b)$ perfectly. To see this, observe that the first- and second-slot function keys and ciphertexts that B obtains from CH are all correctly distributed. Furthermore, given them, B generates the first- and second-components keys and ciphertexts exactly as algorithms sIPE.KeyGen and sIPE.Enc do. Therefore, if A distinguishes the two SdFH experiments with advantage $\nu(\lambda)$, so does B. □

6.6 Output-Encoded Slotted-IPE using GES

In this section, we instantiate the output-encoded slotted-IPE sIPE^N in Section 6.4 using GES for depth-D 4-ary tree. (The instantiation is the same for constant or logarithmic D.)

Recall that sIPE^N is built upon asymmetric bilinear maps. As discussed before, encodings under each label $G_{l'} = \{[\alpha]_{l'} : \alpha \in \mathcal{R}\}$ "acts" as a group under the $\oplus$ operator. In particular, for any label l in the second last layer $D-1$ of the tree $\mathcal{T}$, $G_{l||0}$ and $G_{l||1}$ act as groups. By the definition of **pairable**, encodings in $G_{l||0}$ and $G_{l||1}$ are pairable producing encodings in G_l. Therefore, $G_{l||0}$ and $G_{l||1}$ "act" as the source groups of asymmetric bilinear groups and G_l "acts" as the target group. We can use them to instantiate our output-encoded slotted IPE. More precisely, for $\mathcal{R} \cong \mathbb{Z}_p$, consider a public parameter $\mathsf{pp} = (p, G_{l||0}, G_{l||1}, G_l, \mathbf{pair})$ that describes encodings under label $l||0, l||1, l$.

- The $\mathsf{sIPE.Setup}(1^\lambda, \mathsf{pp})$ on input the security parameter and such a public parameter pp outputs sl.mpk, sl.msk as before.

- In all algorithms, every encoding $[a]_0$ in G_0 of the asymmetric bilinear groups is replaced with an encoding of the same value under label $l||0$ using GES, $[a]_{l||0}$, and every encoding $[a]_1$ in G_1 is replaced with encodings under label $l||1$, $[a]_{l||1}$. Correspondingly, addition and scalar multiplication in G_b for $b \in \{0, 1\}$ are replaced with addition and scalar multiplication between encodings with label $l||b$, and pairing between G_0 and G_1 to G_T is replaced with pairing between encodings with label $l||0$ and $l||1$, to encodings with label l.

Security When instantiated with encodings of GES, the scheme sIPE^N satisfies the same security property, multi-instance joint-indistinguishability, provided that the joint-SXDH assumption holds w.r.t. these encodings used to instantiate the scheme. We define *a multi-group generator $\mathcal{MG}$ w.r.t.* GES.

- $\mathcal{MG}(1^\lambda)$ samples $\mathsf{pp}' \xleftarrow{\$} \mathsf{InstGen}(1^\lambda)$, which describes a graded encoding structure for ring $\mathcal{R} \cong \mathbb{Z}_p$, and outputs $\{\mathsf{pp}_i = (p, G_{l_i||0}, G_{l_i||1}, G_{l_i}, \mathbf{pair})\}_{i\in[m]}$ satisfying that for every $i, j \in [m]$, $l_i \in \{0,1,2,3\}^{D-1}$, and $\mathbf{pairable}(l_i, l_j) = \bot$.

Lemma 10. *Let* GES *be a GES scheme for depth-D 4-ary tree and $\mathcal{MG}$ a multi-group generator w.r.t. it. Assume that the joint-SXDH assumption holds for* GES*. Then, for any polynomial N, the ouput-encoded slotted IPE scheme* sIPE^N *satisfies multi-instance joint-indistinguishability w.r.t. $\mathcal{MG}$.*

Proof. We show that if joint-SXDH holds w.r.t. GES, then joint-SXDH holds w.r.t. $\mathcal{MG}$. This follows because for every $x \in \{0,1\}$, no two "groups" $G_{l_i||x}$ and $G_{l_j||x}$ in the collection of x^{th} "source groups" $\{G_{l_i||x}\}$ can be paired together, that is $\mathbf{pairable}(l_i||x, l_j||x) = \bot$, because $\mathbf{pairable}(l_i, l_j) = \bot$. It follows from the joint-SXDH assumption w.r.t. GES, that the following two ensembles are indistinguishable.

$$\left\{(\mathsf{pp}_1, \cdots, \mathsf{pp}_m) \xleftarrow{\$} \mathcal{MG}(1^\lambda),\ a, b \xleftarrow{\$} \mathbb{Z}_p\ :\ (\mathsf{pp}_1, \cdots, \mathsf{pp}_m),\ \left\{[a]_{l_i||x}, [b]_{l_i||x}, [ab]_{l_i||x}\right\}_{i\in[m]}\right\}_\lambda$$
$$\left\{(\mathsf{pp}_1, \cdots, \mathsf{pp}_m) \xleftarrow{\$} \mathcal{MG}(1^\lambda),\ a, b, r \xleftarrow{\$} \mathbb{Z}_p\ :\ (\mathsf{pp}_1, \cdots, \mathsf{pp}_m),\ \left\{[a]_{l_i||x}, [b]_{l_i||x}, [r]_{l_i||x}\right\}_{i\in[m]}\right\}_\lambda$$

This is exactly the joint-SXDH assumption w.r.t. $\mathcal{MG}$. Then by corollary 2, sIPE^N satisfies multi-instance joint-indistinguishability w.r.t. $\mathcal{MG}$. □

7 FE from the joint-SXDH Assumption on Graded Encodings

We finally describe our functional encryption scheme for NC^0 which is secure under the joint-SXDH assumption on tree-graded encodings. We start in Section 7.1 by introducing our central information-theoretic tool, namely the notion and construction of (affine) arithmetic randomized encodings (ARE) by Applebaum, Ishai and Kushilevitz [AIK14]. In Section 7.2, we use AREs and all the machinery developed in the previous sections to construct our FE scheme, and in Section 7.3, we provide a proof of security.

7.1 Affine Randomized Encoding for NC^1

Syntax An (affine) Arithmetic Randomized Encoding (ARE) scheme produces randomized encodings that depend linearly on the input bits; more specifically, the encodings consist of the outputs of a set of linear functions

$$\widehat{f(\mathbf{x})} = \left\{\mathsf{L}_i(x_{\pi(i)}; \mathbf{r})\right\}_{i\in[\ell]}$$

over a ring $\mathcal{R}$, each evaluated on a single input bit specified by an input mapping function π and shared randomness $\mathbf{r}$. The encoding procedure first evaluates a program encoding algorithm RPenc that depends only on f (but not $\mathbf{x}$) and outputs the function π, and then evaluates an input encoding algorithm RIenc that evaluates the set of linear functions on appropriate input bits to obtain $\widehat{f(\mathbf{x})}$.

Formally, an ARE scheme $\mathsf{ARE}_{N,D,S}$ for the class of functions $\mathcal{F}_{\lambda,N,D,S}$ is associated with the following:

- an ensemble of rings $\{\mathcal{R}_\lambda\}_{\lambda\in\mathbb{N}}$, and

- an ensemble of sets of functions $\left\{ \{\mathsf{L}_i\}_{i\in[\ell]} \right\}_{\lambda\in\mathbb{N}}$ and a polynomial M, such that for every λ, $\ell = \ell(\lambda)$, $M = M(\lambda)$, and $i \in [\ell]$, $\mathsf{L}_i : \{0,1\} \times \mathcal{R}^M \to \mathcal{R}$ is linear in its first input.

$$\mathsf{L}_i(b, \mathbf{r}) = bp_i(\mathbf{r}) + q_i(\mathbf{r}) \text{ where } p_i : \mathcal{R}^M \to \mathcal{R},\ q_i : \mathcal{R}^M \to \mathcal{R}$$

The scheme $\mathsf{ARE}^{N,D,S}$ consists of three PPT algorithms (RPenc, RIenc, REval). For any λ, function $f \in \mathcal{F}_\lambda^{N,D,S}$ described as (C, ρ), and input $\mathbf{x} \in \{0,1\}^N$, the algorithms proceed as follows:

- *Encoding:* $\mathsf{REnc}(1^\lambda, f, \mathbf{x})$ runs the following two algorithms in sequence and outputs what RIenc outputs.

  1. *Program Encoding:* $\mathsf{RPenc}(1^\lambda, f)$ outputs an input mapping function $\pi : [\ell] \to [N]$ that specifies which input bit $x_{\pi(i)}$ the i^{th} linear function L_i should be applied to.
  2. *Input Encoding:* $\mathsf{RIenc}(1^\lambda, \mathbf{x}, \pi)$ samples $\mathbf{r} \xleftarrow{\$} \mathcal{R}^M$, and evaluates L_i on input bit $x_{\pi(i)}$ and $\mathbf{r}$ to obtain the i^{th} element in the randomized encoding $\Pi[i] = \mathsf{L}_i(x_{\pi(i)};\ \mathbf{r})$. It outputs $\widehat{f(\mathbf{x})} = \Pi$.

- *Evaluation:* $\mathsf{REval}(\widehat{f(\mathbf{x})})$ on input an encoding $\widehat{f(\mathbf{x})}$ outputs a value $y = f(\mathbf{x})$.

The correctness and security requirements are identical to regular randomized encoding schemes as in Definition 6.

The AIK Affine Randomized Encoding Schemes for NC^1 In [AIK11] (AIK), a very simple affine randomized encoding scheme for Boolean NC^1 is proposed, whose time complexity grows exponentially in the depth of the functions. Their main novelty is extending this basic scheme to work with $\mathsf{P/poly}$. For our purpose, the simple scheme for NC^1 suffices. At a high-level, the AIK schemes for NC^1 generates randomized encodings inductively depending on the depth of the circuit. From randomized encodings of depth d circuits to that of depth $d+1$ circuits, the following simple Π randomized encoding for quadratic functions of form $f(x_1, x_2) = c_1x_1x_2 + c_2$ is used.

$$\begin{pmatrix} \Pi[0] \\ \Pi[2] \end{pmatrix} = \begin{pmatrix} 1 \\ -r_1 \end{pmatrix} x_1 + \begin{pmatrix} r_0 \\ r_2 \end{pmatrix} \qquad \begin{pmatrix} \Pi[1] \\ \Pi[3] \end{pmatrix} = \begin{pmatrix} 1 \\ -r_0 \end{pmatrix} c_1x_2 + \begin{pmatrix} r_1 \\ -r_2 - r_0r_1 + c_2 \end{pmatrix} \tag{9}$$

The above randomized encoding Π can be evalutate as $\Pi[0]\Pi[1] + \Pi[2] + \Pi[3] = f(x_1, x_2)$.

Below we recall their scheme. Fix any polynomials N, S, and a logarithmic function D. The AIK ARE scheme $\mathsf{ARE}^{N,D,S}$ for the class of Boolean functions in $\mathcal{F}_\lambda^{N,D,S}$ works with arbitrary rings $\{\mathcal{R}\}_\lambda$, and a sets of functions $\{\{\mathsf{L}_i\}_{i\in[\ell]}\}_\lambda$ specified below. For any Boolean function $f = (C, \rho) \in \mathcal{F}_\lambda^{N,D,S}$, we assume w.l.o.g. that C contains $D = D(\lambda)$ layers of NAND gates. We use the following notations:

- C^d represents the circuit consisting of the first d layers of C, with a set of w^d input wires $\mathbf{x}^d$ (i.e., output wires of layer $d+1$). In particular, when $d = 0$, C^0 is a circuit with no gate and simply output the input bit $\mathbf{x}^0$. When $d = D$, $C^D = C$ and $\mathbf{x}^D = \mathbf{x}|\rho$. (Recall that C is evaluated on the subset ρ of the bits of $\mathbf{x}$.)

Using the above notations, we now describe the algorithms in the AIK ARE scheme $\mathsf{ARE}^{N,D,S}$:

- *Program Encoding:* $\mathsf{RPenc}(1^\lambda, f = (C, \rho))$ inductively define the randomized encoding $\Pi^d(\mathbf{x}^d)$ for C^d treating $\mathbf{x}^d$ as formal variables, from $d = 0$ to D. We denote by $\{\mathsf{L}_i^d\}_{i\in[\ell^d]}$ the set of linear functions for computing the randomized encoding of depth d circuits, and ℓ^d the number of these functions.

  - BASE CASE $d = 0$: The randomized encoding for $C^0(x_1^0) = x_1^0$ is simply $\Pi^0(x_1^0) = x_1^0$. The corresponding linear function and input mapping function are:

    - $*$ the set of functions $\left\{\mathsf{L}_1^0\right\}$ contains a single function $\mathsf{L}_1^0(b, \mathbf{r}^0 = \varepsilon) = b$, and
    - $*$ the input mapping function $\pi^0 : [1] \to [1]$ is $\pi^0(1) = 1$.

    This way,

$$\left\{\Pi^0(\mathbf{x}^0)[1] = \mathsf{L}_1^0(x_1^0, \mathbf{r}^0) = x_1^0\right\}$$

  - INDUCTIVE CASE FROM d TO $d+1$: Given the randomized encoding for C^d,

$$\left\{\Pi^d(\mathbf{x}^d)[i] = \mathsf{L}_i^d(x_{k_i}^d, \mathbf{r}^{\le d}) = p_i^d(\mathbf{r}^{\le d})x_{k_i}^d + q_i^d(\mathbf{r}^{\le d})\right\}_{i\in[\ell^d]} \text{ where } k_i = \pi^d(i).$$

    Towards defining the randomized encoding for C^{d+1}, observe that every bit x_k^d equals to the NAND of two input bits $x_{\tau(k,1)}^{d+1}, x_{\tau(k,2)}^{d+1}$ for layer $d+1$, where τ is determined by the wiring in layer $d+1$ of C. Then, every element $\Pi^d(\mathbf{x}^d)[i]$ in the randomized encoding Π^d is a quadratic function over two input bits in $\mathbf{x}^{d+1}$.

$$\begin{aligned}\Pi^d(\mathbf{x}^d)[i] &= p_i^d(\mathbf{r}^{\le d})\left(1 - x_{\tau(k_i,1)}^{d+1}x_{\tau(k_i,2)}^{d+1}\right) + q_i^d(\mathbf{r}^{\le d}) \\ &= -p_i^d(\mathbf{r}^{\le d})\left(x_{\tau(k_i,1)}^{d+1}x_{\tau(k_i,2)}^{d+1}\right) + \left(p_i^d(\mathbf{r}^{\le d}) + q_i^d(\mathbf{r}^{\le d})\right) \text{ , where } k_i = \pi^d(i).\end{aligned}$$

    Compute the randomized encoding for this quadratic function, as described in equation (9).

$$\begin{aligned}\begin{pmatrix}\Pi^{d+1}(\mathbf{x}^{d+1})[4i+0] \\ \Pi^{d+1}(\mathbf{x}^{d+1})[4i+2]\end{pmatrix} &= \begin{pmatrix}1 \\ -r_{i,1}^{d+1}\end{pmatrix} x_{\tau(k_i,1)}^{d+1} + \begin{pmatrix}r_{i,0}^{d+1} \\ r_{i,2}^{d+1}\end{pmatrix} \\ \begin{pmatrix}\Pi^{d+1}(\mathbf{x}^{d+1})[4i+1] \\ \Pi^{d+1}(\mathbf{x}^{d+1})[4i+3]\end{pmatrix} &= \begin{pmatrix}1 \\ -r_{i,0}^{d+1}\end{pmatrix} - p_i^d(\mathbf{r}^{\le d})x_{\tau(k_i,2)}^{d+1} + \begin{pmatrix}r_{i,1}^{d+1} \\ \nu\end{pmatrix} \\ &\text{where } \nu = -r_{i,2}^{d+1} - r_{i,0}^{d+1}r_{i,1}^{d+1} + \left(p_i^d(\mathbf{r}^{\le d}) + q_i^d(\mathbf{r}^{\le d})\right)\end{aligned}$$

    Then, the randomized encoding of $C^{d+1}(\mathbf{x}^{d+1})$ is the collection of randomized encodings of every quadratic function, each corresponding to an element in the randomized encoding of $C^d(\mathbf{x}^d)$, that is, $\Pi^{d+1}(\mathbf{x}^{d+1}) = \left\{\Pi^{d+1}(\mathbf{x}^{d+1})[4i+c]\right\}_{i\in[\ell^d],c\in\{0,1,2,3\}}$. In total, there are $\ell^{d+1} = 4\ell^d = 4^{d+1}$ elements in Π^{d+1} and the corresponding linear functions $\{\mathsf{L}_{4i+c}\}$ and π^{d+1} are:

$$\begin{aligned}\pi^{d+1}(4i+0) = \pi^{d+1}(4i+2) = \tau(k_i, 1) = \tau(\pi^d(i), 1) \\ \pi^{d+1}(4i+1) = \pi^{d+1}(4i+3) = \tau(k_i, 2) = \tau(\pi^d(i), 2)\end{aligned}$$

$$\begin{aligned}
&\mathsf{L}^{d+1}_{4i+0}(x^{d+1}_{\tau(k_i,1)}, \mathbf{r}^{\le d+1}) = x^{d+1}_{\tau(k_i,1)} + r^{d+1}_{i,0}\\
&\mathsf{L}^{d+1}_{4i+2}(x^{d+1}_{\tau(k_i,1)}, \mathbf{r}^{\le d+1}) = -r^{d}_{i,1} x^{d+1}_{\tau(k_i,1)} + r^{d+1}_{i,2}\\
&\mathsf{L}^{d+1}_{4i+1}(x^{d+1}_{\tau(k_i,2)}, \mathbf{r}^{\le d+1}) = -p^{d}_{i}(\mathbf{r}^{\le d}) x^{d+1}_{\tau(k_i,2)} + r^{d+1}_{i,1}\\
&\mathsf{L}^{d+1}_{4i+3}(x^{d+1}_{\tau(k_i,2)}, \mathbf{r}^{\le d+1}) = r^{d+1}_{i,0} p^{d}_{i}(\mathbf{r}^{\le d}) x^{d+1}_{\tau(k_i,2)} + \Big(-r^{d+1}_{i,2} - r^{d+1}_{i,0} r^{d+1}_{i,1} + p^{d}_{i}(\mathbf{r}^{\le d}) + q^{d}_{i}(\mathbf{r}^{\le d})\Big)
\end{aligned}$$

where $\mathbf{r}^{\le d+1}$ consists of $\mathbf{r}^{\le d}$ and all the new random elements $\{r^{d+1}_{i,t}\}_{i\in[\ell^d], t\in\{0,1,2\}}$.

The induction ends at $d = D$, procuring the randomized encoding for $C^D(x^D) = C(x|_\rho)$:

$$\left\{\Pi^D(\mathbf{x}^D)[i] = \mathsf{L}^D_i(x^D_{k_i}, \mathbf{r}^{\le D})\right\}_{i\in[\ell^D]} \text{ where } k_i = \pi^D(i) \text{ and } \ell^D = 4^D$$

To transform the input mapping function from working with domain x^D to domain x, RPenc sets the final input mapping function π so that

$$\forall i \in [\ell^D],\ x_{\pi(i)} = (x|_\rho)_{\pi^D(i)}$$

It outputs π.

- *Input Encoding:* $\mathsf{RIenc}(1^\lambda, x, \pi)$ samples random $\mathbf{r}^{\le D}$ and outputs $\left\{\Pi[i] = \mathsf{L}^D_i(x_{\pi(i)}; \mathbf{r}^{\le D})\right\}_{i\in[\ell^D]}$.
- *Evaluation:* $\mathsf{REval}(\Pi)$ on input an encoding Π of $f(x)$ inductively computes the randomized encoding Π^d for $C^d(x^d)$ from $d = D$ to 0. The final output is exactly $y = \Pi^0$.
  - Base Case $d = D$: Π^D for $C^D(x^D)$ is exactly Π, and has length $\ell^D = 4^D$.
  - Inductive step from $d+1$ to d: Given the randomized encoding Π^{d+1} for $C^{d+1}(x^{d+1})$, compute the randomized encoding Π^d for $C^d(x^d)$ as follows:
    Since every element $i \in [\ell^d]$ in Π^d is encoded by $\Pi^{d+1}[4i+0, \cdots, 4i+3]$, we obtain

$$\Pi^d[i] = \Pi^{d+1}[4i+0]\Pi^{d+1}[4i+1] + \Pi^{d+1}[4i+2] + \Pi^{d+1}[4i+3]$$

Lemma 11 ([AIK11]). *For every polynomials N and S and logarithmic function D, the AIK scheme $\mathsf{ARE}^{N,D,S}$ is a perfectly secure affine randomized encoding for Boolean functions in the class $\mathcal{F}^{N,D,S}$.*

7.1.1 Additional Properties of the AIK Affine Randomized Encoding

We observe the following properties of the AIK ARE scheme $\mathsf{ARE}^{N,D,S}$, which will be instrumental for our construction of FE scheme later. Let $\{\mathsf{L}^D_i\}_{i\in[\ell^D]}$ be the set of linear functions associated with $\mathsf{ARE}^{N,D,S}$, where each function $\mathsf{L}^D_i(b, \mathbf{r}) = p^D_i(\mathbf{r})b + q^D_i(\mathbf{r})$.

- *Multilinear coefficient functions:* All the coefficient functions $\{q^D_i, p^D_i\}$ are multilinear in elements of $\mathbf{r}$.
- *Bounded number of monomials:* Let $\mathsf{Mnml}(\{q^D_i, p^D_i\})$ be the set of monomials (over elements of $\mathbf{r}$) in p^D_i's and q^D_i's. The total number of monomials is bounded by a fixed polynomial, in particular, $|\mathsf{Mnml}(\{q^D_i, p^D_i\})| = O(\lambda^2)$.

  This can be shown via the following recurrence relation: Let $M^d = |\mathsf{Mnml}(\{q^d_i, p^d_i\})|$. $M^{d+1} = M^d + 5\ell^d$, and thus $M^D = O(\lambda^2)$.

- *Multiplication with independent randomness:* For every $d+1 \leq D$, evaluation of the randomized encoding Π^{d+1} for $C^{d+1}(x^{d+1})$ involves computing $\Pi^d[i] = \Pi^{d+1}[4i+0]\Pi^{d+1}[4i+1]+\Pi^{d+1}[4i+2] + \Pi^{d+1}[4i+3]$ for every $i \in [\ell^d]$. Note that, the two elements $\Pi^{d+1}[4i+0]$ and $\Pi^{d+1}[4i+1]$ that are multiplied together depend on completely independent random coins, the former on $r_{i,0}^{d+1}$ and the latter on $\mathbf{r}^{\leq d}$ and $r_{i,1}^{d+1}$. They both share random coins with the other two terms, which however are only added together.

These property are crucial for our FE construction next.

7.2 Construction

We now construct a FE scheme $\mathbf{FE}^{N,D,S}$ for the class of functions $\mathcal{F}^{N,D,S}$, where N, S are any polynomials and D is any logarithmic function. The construction uses the following building blocks:

- a GES scheme GES for depth-$(D(\lambda)+1)$ 4-ary tree $\mathcal{T}$. Let $\mathcal{R}$ and $\{(\mathcal{L}, l^\star, \mathbf{pairable})\}$ be the parameters associated with GES. (When D is a constant, the ensemble is replace with a single tuple $(\mathcal{L}, l^\star, \mathbf{pairable})$),
- the ouput-encoded slotted IPE scheme sIPE^M instantiated with GES of Lemma 12, and
- the AIK affine randomized encoding scheme $\mathsf{ARE}^{N,D,S}$.

Overview Recall the following facts about our GES scheme associated with a depth-$(D(\lambda)+1)$ 4-ary tree $\mathcal{T}$. It has a label set $\mathcal{L} = \{0,1,2,3\}^{\leq D+1}$ consisting of the labels of all nodes in $\mathcal{T}$. Equality testing can be performed over encodings labeled with the root ε, and pairing can be done between descendants (l_1, l_2) of two labels (l'_1, l'_2) that are either the first two children $(l||0, l||1)$ of a node l, or the second two $(l||2, l||3)$.

To compute $f(x)$, our FE scheme generates a secret key sk of f and an encryption ct of x using multiple instances $\{\mathsf{sI.sk}_i, \mathsf{sI.ct}_i\}$ (under different master public and secret keys) of the slotted IPE scheme sIPE. Different instances of sIPE are instantiated with encodings labeled with different leaves $l||0, l||1 \in \{0,1,2,3\}^{D+1}$ of $\mathcal{T}$, and the vectors encoded in $\{\mathsf{sI.sk}_i, \mathsf{sI.ct}_i\}$ are set up carefully so that their inner products produce exactly the randomized encoding $\widehat{f(x)} = \Pi$ of $f(x)$ (up to some technicality discussed below). Here, we crucially rely on the fact that the randomized encoding is affine in order to compute it using only inner product. Then, evaluating sk and ct yields GES-encodings of Π labeled with different nodes at (the second last) layer D. Using the tree computation structure of GES, GES-encodings of Π can be homomorphically evaluated, eventually yielding a GES-encoding of the output $y = f(x)$ at the root. Since the final output is Boolean, y can be extracted using equality testing at the root.

Two technical challenges need to be addressed for the above high-level plan to go through. First, where does the random coins for generating the randomized encoding Π come from? One naive attempt is embedding some fixed random elements $\mathbf{r}^{\leq D}$ in either the secret key sk or the ciphertext ct. But, this would lead to reusing the same random elements across all randomized encodings produced using that secret key or that ciphertext, rendering these randomized encodings insecure. To overcome this problem, we embed independent random elements $\mathbf{r}_{sk}^{\leq D}$ and $\mathbf{r}_{ct}^{\leq D}$ in sk and ct respectively, so that, the randomized encoding Π produced by sk, ct corresponds to random elements $\mathbf{r}^{\leq D} = \mathbf{r}_{sk}^{\leq D}\mathbf{r}_{ct}^{\leq D}$ (where multiplication is coordinate-wise). This way, randomized encodings produced using different pairs of secret key and ciphertext would not have identical random elements, but still correlated random elements. Then, in the security proof, we first move to a hybrid experiment where the randomized encodings Π for $f(x)$ is hardwired in the function key sk for f, and Π is generated using $\mathbf{r}^{\leq D}\mathbf{r}_{sk}^{\leq D}\mathbf{r}_{ct}^{\leq D}$. Then, we move to another hybrid experiment where the random elements $\mathbf{r}^{\leq D}$ are randomly and independently sampled, by relying on the joint-SXDH assumption of GES. Finally, in this hybrid, we can leverage the perfect security of the randomized encoding scheme ARE to argue that it is indistinguishable to switch from one challenge input $\mathbf{x}^0$ to another $\mathbf{x}^1$, since they produce the same outputs w.r.t. f.

One technicality is that our slotted IPE scheme sIPE when evaluated does not directly produce an encoding of the output $[\Pi]_l$. Instead, for each element $\Pi[i]$, it produces encodings $[\Pi[i]\theta_i]_l$ and $[\theta_i]_l$ w.r.t. some mask θ_i (which depends on the random scalars $\alpha, \alpha^\star, \beta, \beta^\star$ used in key generation and encryption of sIPE). If different elements $\Pi[i]$'s are masked with distinct θ_i's, Π can no longer be evaluated. To see this, consider the simple randomized encoding $\Pi[0, \cdots, 3]$ for the quadratic function $q(x_1, x_2) = c_1 x_1 x_2 + c_2$. The evaluation involves computing $y = \Pi[0]\Pi[1] + \Pi[2] + \Pi[3]$, which cannot be done if each $\Pi[c]$ is masked with a different θ_c. One naive fix is to mask all $\Pi[i]$ with the same θ. This however is insecure for the following reason. Sharing the mask θ corresponds to sharing the same random scalars $\alpha, \alpha^\star, \beta, \beta^\star$ across *all* instances of sIPE. Though our slotted IPE satisfies multi-instance security with shared random scalars, this only holds for instances instantiated with encodings that cannot be paired together (otherwise, the joint-SXDH assumption does not hold). Thus, we cannot use a universal mask θ. Instead, we design a way to partition elements in Π into groups, such that, it is secure to share the same mask θ within each group, and Π can still be evaluated though different groups use different masks.

Formal Construction Fix a security parameter λ, a function $f = (C, \rho) \in \mathcal{F}^{N,D,S}$ and an input $x \in \{0,1\}^N$. The scheme $\mathbf{FE}^{N,D,S}$ associates every layer $d \in \{0, \cdots, D\}$ of the three $\mathcal{T}$ with the randomized encoding Π^d of $C^d(x^d)$. In particular, the root is associated with the output $y = \Pi^0$ and the second last layer D is associated with the randomized encoding Π^D of $C^D(x^D)$, which is the randomized encoding of $f(x)$. The last layer $D+1$ is used to instantiate different instances of sIPE, which when evaluated produce the encodings of Π^D at layer D. Within each layer $d \in \{0, \cdots, D\}$, different nodes $l \in \{0,1,2,3\}^d$ are associated with different *disjoint* subsets $S_l^d \subseteq [\ell^d]$ of the elements in Π^d, such that, $\cup_l S_l^d = [\ell^d]$. The scheme guarantees that for all element $\Pi^d[i]$ associated with a node l (i.e., $i \in S_l^d$), evaluation would produce their encodings masked by some value θ_l under the label l, that is, $[\Pi^d[i]\theta_l]_l$ and $[\theta_l]_l$.

We first specify inductively the assignment of subsets $\{S_l^d\}$.

- *Base Case for $d = 0$:* $S_\varepsilon^0 = \{1\}$, that is, the root corresponds to Π_1^0.
- *Inductive Case from d to $d+1$:* For every $l \in \{0,1,2,3\}^d$ in layer d associated with subset S_l^d, its four children at layer $d+1$ are associated with subsets:

  $$S_{l||0}^{d+1} = \{4i \mid \forall i \in S_l^d\} \ , \ S_{l||1}^{d+1} = \{4i+1 \mid \forall i \in S_l^d\} \ , \ S_{l||2}^{d+1} = \{4i+2, 4i+3 \mid \forall i \in S_l^d\} \ , \ S_{l||3}^{d+1} = \emptyset$$

  That is, if element $\Pi^d[i]$ is assigned to node l, then the four elements $\Pi^{d+1}[4i+0, \cdots 4i+3]$ in Π^{d+1} that computes $\Pi^d[i]$ are assigned to l's four children: $\Pi^d[4i+0]$ to child $l||0$, $\Pi^d[4i+1]$ to $l||1$, and $(\Pi^d[4i+2], \Pi^d[4i+3])$ to $l||2$. Node $l||3$ is not assigned with any element of Π^{d+1}, but is used for computing encoding of the mask.

With the above assignment of S_l^d, we now describe the algorithms of $\mathbf{FE}^{N,D,S}$.

- *Setup:* $\mathsf{FE.Setup}(1^\lambda)$ does the following:

- Sample the *public parameter* $\mathsf{pp} \xleftarrow{\$} \mathsf{InstGen}(1^\lambda)$ which describes a $(\mathcal{R}, \mathcal{L}, \mathbf{pairable})$-GES with $\mathcal{R} \cong \mathbb{Z}_p$ for some prime p.

  For every $l \in \{0,1,2,3\}^D$, let $\mathsf{pp}_l = (p, G_{l||0}, G_{l||1}, G_l, \mathbf{pair})$ be the public parameter that describes encodings under label $l||0, l||1, l$.
- Jointly sample a pair of scalars for every node in layer D $\{\alpha^\star_l, \beta^\star_l\}_{l\in\{0,1,2,3\}^D}$ from the scalar distribution $\mathcal{D}(\mathcal{R}, D)$:
  - $*$ For every $l \in \{0,1\}^D$, sample $\alpha^\star_l, \beta^\star_l \xleftarrow{\$} \mathcal{R}$ at random.
  - $*$ For every $l \in \{0,1,2,3\}^D \setminus \{0,1\}^D$, $\alpha^\star_l, \beta^\star_l = \alpha^\star_{\tau(l)}, \beta^\star_{\tau(l)}$, where $\tau(l)$ replaces every letter $l_i \in \{0,1,2,3\}$ with $l_i \bmod 2 \in \{0,1\}$.
- for every node in the second last layer $l \in \{0,1,2,3\}^D$ and every $i \in S^D_l \cup \{0\}$, sample a pair of master public and secret keys of sIPE, using public parameter pp_l and random scalars $\alpha^\star_l, \beta^\star_l$.

$$\left\{(\mathsf{sI.mpk}_{l,i}, \mathsf{sI.msk}_{l,i}) \xleftarrow{\$} \mathsf{sIPE.Setup}(1^\lambda, \mathsf{pp}_l;\ \alpha^\star_l, \beta^\star_l)\right\}_{l\in\{0,1,2,3\}^D, i\in S^D_l\cup\{0\}}$$

  Each $\mathsf{sI.mpk}_{l,i}, \mathsf{sI.msk}_{l,i}$ defines an instance of sIPE instantiated with encodings under labels $l||0$, $l||1$ and l.

It outputs master public key $\mathsf{mpk} = \{\mathsf{sI.mpk}_{l,i}\}$ and master secret key $\mathsf{msk} = \{\mathsf{sI.msk}_{l,i}\}$.

- *Key Generation:* $\mathsf{FE.KeyGen}(\mathsf{msk}, f = (C, \rho))$ does the following:
  - Jointly sample a pair of scalars for every leaf $\{\alpha_l, \beta_l\}_{l\in\{0,1,2,3\}^D}$ from the distribution $\mathcal{D}(\mathcal{R}, D)$ described in FE.Setup.
  - Sample elements of $\mathbf{r}^{\le D}_k$ randomly from $\mathcal{R}$.
  - Let $\{\mathsf{L}^D_i = p^D_i(\mathbf{r}^{\le D})b + q^D_i(\mathbf{r}^{\le D})\}_{i\in[\ell^D]}$ be the set of functions associated with $\mathsf{ARE}^{N,D,S}$, and $Mn = \mathsf{Mnml}(\{q^D_i, p^D_i\}) = \{m_j\}_{j\in[Q]}$ the set of monomials in the coefficient polynomials (recall that $Q = |Mn| = O(\lambda^2)$). Then, p^D_i, q^D_i can be decomposed to:

$$q^D_i(\mathbf{r}^{\le D}) = \Sigma_{j\in[Q]}\ \gamma_{i,j} m_j(\mathbf{r}^{\le D}) \qquad p^D_i(\mathbf{r}^{\le D}) = \Sigma_{j\in[Q]}\ \delta_{i,j} m_j(\mathbf{r}^{\le D})$$

    For every $i \in [\ell^D]$, set V_i to the following length-$2Q$ vector:

$$\begin{aligned} V_i = \gamma_{i,1}m_1(\mathbf{r}^{\le D}_{sk})||\cdots||\gamma_{i,j}m_j(\mathbf{r}^{\le D}_{sk})||\cdots||\gamma_{i,Q}m_Q(\mathbf{r}^{\le D}_{sk})& \\ ||\delta_{i,1}m_1(\mathbf{r}^{\le D}_{sk})||\cdots||\delta_{i,j}m_j(\mathbf{r}^{\le D}_{sk})||\cdots||\delta_{i,Q}m_Q(\mathbf{r}^{\le D}_{sk})& \end{aligned}$$

  - Generate the input mapping function $\pi \xleftarrow{\$} \mathsf{RPenc}(1^\lambda, f = (C, \rho))$. For every $i \in [\ell^D] \cup \{0\}$, set Y_i to the following length-M vector where $M = 2QN$.

$$Y_i = \begin{cases} V_i \otimes \mathbf{u}_{\pi(i)} = (0^{2Q})^{\pi(i)-1}||V_i||(0^{2Q})^{N-\pi(i)} & \text{if } i \in [\ell^D] \\ 0^{M-1}1 & \text{if } i = 0 \end{cases}$$

    where $\mathbf{u}_{\pi(i)}$ is the unit vector of length-N that has a single 1 at position $\pi(i)$.

- for every $l \in \{0,1,2,3\}^D$ and $i \in S_l^D \cup \{0\}$, sample a sIPE function key $\mathsf{sl.sk}_{l,i}$ of Y_i, using $\mathsf{sl.msk}_{l,i}$ and the random scalars α_l, β_l:

$$\mathsf{sl.sk}_{l,i} \xleftarrow{\$} \mathsf{sIPE.KeyGen}(\mathsf{sl.msk}_{l,i},\ Y_i,\ 1^M;\ \alpha_l, \beta_l)$$

  where the second slot is set to a dummy value 1^M.

Output $\mathsf{sk} = \{\mathsf{sl.sk}_{l,i}\}$.

- *Encryption:* $\mathsf{FE.Enc}(\mathsf{mpk}, x)$ does the following:

  - Sample elements of $\mathbf{r}_{ct}^{\leq D}$ randomly from $\mathcal{R}$.
  - Set U and X to the following length-Q and length-$2N$ vectors

$$\begin{aligned} U &= m_1(\mathbf{r}_{ct}^{\leq D})||m_j(\mathbf{r}_{ct}^{\leq D})|| \cdots ||m_Q(\mathbf{r}_{ct}^{\leq D}) \\ X &= x_1||1||x_2||1 \cdots ||x_N||1 \end{aligned}$$

  - for every $l \in \{0,1,2,3\}^D$ and $i \in S_l^D \cup \{0\}$, set X_i to the following length $M = 2QN$ vector.

$$X_i = \begin{cases} U \otimes X = x_1U||U||x_2U||U \cdots ||x_nU||U & \text{if } i \in [\ell^D] \\ 0^{M-1}1 & \text{if } i = 0 \end{cases}$$

  - for every $l \in \{0,1,2,3\}^D$ and $i \in S_l^D \cup \{0\}$, sample a sIPE ciphertext $\mathsf{sl.ct}_{l,i}$ using $\mathsf{sl.mpk}_{l,i}$: $\mathsf{sl.ct}_{l,i} \xleftarrow{\$} \mathsf{sIPE.Enc}(\mathsf{sl.mpk}_{l,i}, X_i)$.

It outputs $\mathsf{ct} = \{\mathsf{sl.ct}_{l,i}\}$.

- *Decryption:* $\mathsf{FE.Dec}(\mathsf{sk}, \mathsf{ct})$ does the following:

  - *Evaluate the IPE secret keys and ciphertexts:* For every $l \in \{0,1,2,3\}^D$ and $i \in S_l^D \cup \{0\}$, evaluate $\mathsf{sl.sk}_{l,i}$ and $\mathsf{sl.ct}_{l,i}$ to obtain $Z_{l,i}^D = \mathsf{sIPE.Dec}(\mathsf{sl.sk}_{l,i}, \mathsf{sl.ct}_{l,i})$.

By the correctness of sIPE*,*

$Z_{l,i}^D = \mathsf{OEnc}((X_i \cdot Y_i),\ \mathsf{sl.mpk}_{l,i}, \alpha_l, \alpha_l^\star, \beta_l, \beta_l^\star) = [(X_i \cdot Y_i)\theta_l]_l, [\theta_l]_l$ *where* $\theta_l = \alpha_l\alpha_l^\star + \beta_l\beta_l^\star$

By definition of X_i and Y_i,

$$\begin{aligned} X_0 \cdot Y_0 &= (0^{M-1}1) \cdot (0^{M-1}1) = 1 \\ X_i \cdot Y_i &= (V_i \otimes \mathbf{u}_{\pi(i)}) \cdot (U \otimes X) = V_i \cdot \left(x_{\pi(i)}U||U\right) \\ &= \left(\Sigma_{j\in[Q]}\gamma_{i,j}m_j(\mathbf{r}_{sk}^{\leq D})m_j(\mathbf{r}_{ct}^{\leq D})\right) x_{\pi(i)} + \left(\Sigma_{j\in[Q]}\delta_{i,j}m_j(\mathbf{r}_{sk}^{\leq D})m_j(\mathbf{r}_{ct}^{\leq D})\right) \qquad (*) \\ &= \left(\Sigma_{j\in[Q]}\gamma_{i,j}m_j(\mathbf{r}_{sk}^{\leq D}\mathbf{r}_{ct}^{\leq D})\right) x_{\pi(i)} + \left(\Sigma_{j\in[Q]}\delta_{i,j}m_j(\mathbf{r}_{sk}^{\leq D}\mathbf{r}_{ct}^{\leq D})\right) \\ &= p_i^D(\mathbf{r}^{\leq D})x_{\pi(i)} + q_i^D(\mathbf{r}^{\leq D}) \qquad \textit{where } \mathbf{r}^{\leq D} = \mathbf{r}_{sk}^{\leq D}\mathbf{r}_{ct}^{\leq D} \\ &= \Pi^D[i](\mathbf{r}^{\leq D}) = \Pi^D[i] \end{aligned}$$

We note that equality (*) holds because all $\{p_i^D, q_i^D\}$ are multiplier and hence so are all monomials $\{m_j\}$. (See Section 7.1.1.)

For convenience of notation, we think of Π^d being pre-pended with a 1, that is, $\Pi^d[0] = 1$ for all $d \in [D]$. Then, evaluating $\mathsf{sl.sk}_{l,i}$ and $\mathsf{sl.ct}_{l,i}$ gives GES-encodings of the randomized encoding Π^D of $f(x)$, more precisely,

$$\left\{Z_{l,i}^D[1] = [\Pi^D[i]\theta_l]_l\right\}_{l\in[\ell^D], i\in[S_l^D]\cup\{0\}} \quad \textit{where } \Pi^D[0] = 1 \tag{10}$$

– *Evaluate the randomized encoding Π^D homomorphically:* Starting from the encodings in Equation 10 at layer $D-1$, inductively evaluate the encodings to obtain encodings of Π^d at higher layers:

$$\left\{[\Pi^d[i]\theta_l]_l\right\}_{l\in[\ell^d], i\in[S_l^d]\cup\{0\}} \quad \text{where } \theta_l = \theta_{l||0}\theta_{l||1}$$

while maintaining the invariant that $\theta_l = \theta_{\tau(l)}$.

The inductive step from encodings at layer $d+1$, $\{[\Pi^{d+1}[i]\theta_l]_l\}$ for $l \in \{0,1,2,3\}^{d+1}$, $i \in [S_l^{d+1}] \cup \{0\}$, to encodings at layer d, $\{[\Pi^d[i]\theta_l]_l\}$ for $l \in \{0,1,2,3\}^d$, $i \in [S_l^d] \cup \{0\}$, proceeds as follows:

  - For every $l \in \{0,1,2,3\}^d$ and $i = 0$,

$$\begin{aligned}[\Pi^d[0]\theta_l]_l = [\theta_l]_l &= \mathbf{pair}([\theta_{l||0}]_{l||0}, [\theta_{l||1}]_{l||1}) \\ &= \mathbf{pair}([\Pi^{d+1}[0]\theta_{l||0}]_{l||0}, [\Pi^{d+1}[0]\theta_{l||1}]_{l||1})\end{aligned} \tag{11}$$

    Invariant: By the induction hypothesis, $\theta_{l||0} = \theta_{\tau(l)||0}$ and $\theta_{l||1} = \theta_{\tau(l)||1}$, and thus $\theta_l = \theta_{\tau(l)}$.

  - For every $l \in [\ell^d]$ and $i \in S_l^d$, since $\Pi^d[i] = \Pi^{d+1}[4i+0]\Pi^{d+1}[4i+1] + \Pi^{d+1}[4i+2] + \Pi^{d+1}[4i+3]$,

$$\begin{aligned}[\Pi^d[i]\theta_l]_l = \mathbf{pair}&([\Pi^{d+1}[4i+0]\theta_{l||0}]_{l||0}, [\Pi^{d+1}[4i+1]\theta_{l||1}]_{l||1}) \\ &\oplus \mathbf{pair}([\Pi^{d+1}[4i+2]\theta_{l||2}]_{l||2}, [\Pi^{d+1}[0]\theta_{l||3}]_{l||3}) \\ &\oplus \mathbf{pair}([\Pi^{d+1}[4i+3]\theta_{l||2}]_{l||2}, [\Pi^{d+1}[0]\theta_{l||3}]_{l||3})\end{aligned} \tag{12}$$

    Invariant: The induction hypothesis guarantees that for any c, $\theta_{l||c} = \theta_{\tau(l||c)}$, and thus $\theta_l = \theta_{l||0}\theta_{l||1} = \theta_{l||2}\theta_{l||3} = \theta_{\tau(l)||0}\theta_{\tau(l)||1} = \theta_{\tau(l)}$.

At the end of the induction, obtain encodings $[\Pi^0[0]\theta_\varepsilon]_\varepsilon = [\theta_\varepsilon]_\varepsilon$, and $[\Pi^0[1]\theta_\varepsilon]_\varepsilon = [y\theta_\varepsilon]_\varepsilon$. Since $y \in \{0,1\}$, output $y = 1$ iff $\mathsf{Eq}([\theta_\varepsilon]_\varepsilon, [y\theta_\varepsilon]_\varepsilon) = 1$.

To show that the final output y is computed correctly, it suffices to show that the above induction step from $d+1$ to d computes elements $\Pi^d[i]$ correctly, if elements $\Pi^{d+1}[i]$ are correct. Then combined with the fact that $\Pi^D[i]$'s are computed correctly by the analysis above, we conclude that $y = \Pi^0[1]$ is also correct.

The correctness of the induction step follows from the correctness of the AIK ARE scheme, the invariant on θ's and the fact that the assignment of subsets $\{S_l^{d+1}\}$ ensures that all encodings on the right hand side of equations (11) and (12) can be found in $\{[\Pi^{d+1}[i]\theta_l]_l\}_{l\in[\ell^{d+1}], i\in[S_l^{d+1}]\cup\{0\}}$.

7.3 Security

Lemma 12. *Let D be any logarithmic function and* GES *a GES scheme for depth-$D(\lambda) + 1$ 4-ary tree. Assume that the joint-SXDH assumption holds for* GES*. Then, for any polynomials N, S, the scheme $\mathbf{FE}^{N,D,S}$ presented above is a selectively secure FE scheme for the class of Boolean functions in $\mathcal{F}^{N,D,S}$.*

Proof. Fix any PPT adversary A, and security parameter λ. We need to show that experiments $\mathsf{Sel.Exp}_A^{\mathbf{FE}}(1^\lambda, 0)$ and $\mathsf{Sel.Exp}_A^{\mathbf{FE}}(1^\lambda, 1)$ are indistinguishability to A. Recall that in $\mathsf{Sel.Exp}_A^{\mathbf{FE}}(1^\lambda, b)$, the adversary A after receiving a master public key mpk, immediately chooses two challenge inputs (x^0, x^1) and obtains a ciphertext ct of x^b. Then, it can request for many function keys $\{\mathsf{sk}^j\}$ of arbitrary functions $\{f^j\}$ of his choice, with the only restriction that $f^j(x^0) = f^j(x^1)$ for all j. Let J be an upper bound on the number of function keys that A asks. By construction, one instance of $\mathbf{FE}$ consists of many instances of the slotted IPE scheme sIPE instantiated using encodings under different labels $l||0, l||1$, for $l \in \{0, 1, 2, 3\}^D$ in the second last layer. That is,

$$\mathsf{mpk}, \mathsf{msk} = \{\mathsf{sl.mpk}_{l,i}\}, \{\mathsf{sl.msk}_{l,i}\}, \quad \mathsf{sk} = \{\mathsf{sl.sk}_{l,i}\}, \quad \mathsf{ct} = \{\mathsf{sl.ct}_{l,i}\}$$

where $l \in \{0, 1, 2, 3\}^D$ and $i \in S_l^D \cup \{0\}$. We refer to $\mathsf{sl.mpk}_{l,i}, \mathsf{sl.msk}_{l,i}, \mathsf{sl.sk}_{l,i}, \mathsf{sl.ct}_{l,i}$ the $(l, i)^{\text{th}}$ instance of sIPE.

At a high-level, we show the indistinguishability of the two experiments in the following steps:

- *Step 1: Hardwire the randomized encodings.* First, we go through a sequence of 2^D hybrids $H_0^b = \mathsf{Sel.Exp}_A^{\mathbf{FE}}(1^\lambda, b), H_1^b, \cdots, H_{2^D}^b$, to switch every function keys $\mathsf{sk}^j = \{\mathsf{sl.sk}_{l,i}^j\}$ and challenge ciphertext $\mathsf{ct} = \{\mathsf{sl.ct}_{l,i}\}$ from encoding vectors $\{(Y_i^j, 1^M)\}$ and $\{(X_i, 0^M)\}$ (where 1^M and 0^M are encoded in the second slot) to encoding $\{(Y_i^j, 0^{M-1}||\Pi^j[i])\}$ and $\{(0^M, 0^{M-1}||1)\}$, where $\Pi^j[i] = Y_i^j \cdot X_i$. Here, we rely on the fact that A chooses his challenge inputs x^0, x^1 before sending any function queries to ensure that at the time of generating function key sk^j, Π^j is known. The indistinguishability of neighboring hybrids reduces to the multi-instance joint indistinguishability of sIPE.

  In the final hybrid $H_{2^D}^b$, the ciphertext ct contains no information of x^b, and the secret keys $\mathsf{sk}^j = \{\mathsf{sl.sk}_{l,i}^j\}_{l,i}$ has Π^j hardwired inside which is a randomized encoding $\widehat{f^j(x^b)}$. We would like to apply the perfect security of the AIK ARE scheme ARE to claim that $H_{2^D}^0$ is identically distributed as $H_{2^D}^1$. This, however, fails because different Π^j are generated using correlated randomness. In particular, the randomness $\mathbf{r}^j$ for generating Π^j is equal to $\mathbf{r}^j = \mathbf{r}_{\mathsf{sk}}^j \mathbf{r}_{ct}$, where $\mathbf{r}_{ct}$ is embedded in ct, and $\mathbf{r}_{sk}^j$ is embedded in sk^j.

- *Step 2: Switch to truly random elements.* To address the above issue, we use another sequence of hybrids $G_{1,0}^b = H_{2^D}^b, \cdots G_{J,T}^b$ to switch, for each function query j, $\mathbf{r}^j$ from being the product $\mathbf{r}_{sk}^j \mathbf{r}_{ct}$ to a uniformly random vector $\mathbf{w}^j \xleftarrow{\$} \mathcal{R}^T$, (where T is an upper bound on $|\mathbf{r}^j|$). This step relies on the joint-SXDH assumption of GES and crucially the fact that the same random element r_t^j would never appear at two nodes l_1, l_2 that are pairable.

  At the final hybrid $G_{J,T}^b$, the randomized encoding Π^j hardwired in the function key sk^j is computed using independent and random vector $\mathbf{w}^j$.

- *Step 3: Switch the challenge input.* Then it follows from the perfect security of ARE, $G_{J,T}^0$ and $G_{J,T}^1$ are identically distributed. By a hybrid argument, we conclude that experiments $H_0^b = \mathsf{Sel.Exp}_A^{\mathbf{FE}}(1^\lambda, b)$ for $b = 0$ and 1 are indistinguishable.

Next, we describe formally the hybrids $\{H^b_s\}$ and $\{G^b_{j,t}\}$ and analyze the indistinguishability of neighboring hybrids.

Hybrid H^b_s: Hybrid H^b_s for $s \in [2^D]$ proceeds identically to $H^b_0 = \mathsf{Sel.Exp}^{\mathbf{FE}}_A(1^\lambda, b)$, except that, all the sIPE instances in the set $\Gamma(< s) = \{(l, i) : \tau(l) < s,\ i \in S^D_l\}$ encode different function and input vectors:

$$\text{In } H^b_0,\ \forall\,(l,i) \in \Gamma(<s) \quad \mathsf{sI.sk}^j_{l,i} \text{ encodes } (Y^j_i, 1^M) \qquad \mathsf{ct}_{l,i} \text{ encodes } (X_i, 0^M) \tag{13}$$

$$\text{In } H^b_s,\ \forall\,(l,i) \in \Gamma(<s) \quad \mathsf{sI.sk}^j_{l,i} \text{ encodes } (Y^j_i, 0^{M-1}||\Pi^j[i]) \quad \mathsf{ct}_{l,i} \text{ encodes } (0^M, 0^{M-1}||1) \tag{14}$$

where $\Pi^j[i] = Y^j_i \cdot X_i$. All other instances outside the set $\Gamma(< s)$ encode identical vectors as in line (13).

Recall that by construction of **FE**, $\Pi^j[i]$ is the i^{th} element of the randomized encoding Π^j for the computation $f^j(x^b)$. In other words, in H^b_s, for every $i \in \cup_{\tau(l)<s} S^D_l$, the i^{th} randomized encoding element $\Pi^j[i]$ is directly hardwired into sk^j, as opposed to being computed via $Y^j_i \cdot X_i$.

Indistinguishability between H^b_{s-1} and H^b_s: We claim that for every s, hybrids H^b_{s-1} and H^b_s are indistinguishable to A. Note that the only difference between them is that instances in the set $\Gamma(s-1) = \{(l,i) : \tau(l) = s-1,\ i \in S^D_l\}$ encode vectors in line (13) in H^b_{s-1} and vectors in line (14) in H^b_s. By the construction of **FE**, all instances $(l,i) \in \Gamma(s-1)$ use random scalars $\alpha^\star_{s-1}, \beta^\star_{s-1}$ in the setup phase for generating the master public and secret keys $(\mathsf{sI.mpk}_{l,i}, \mathsf{sI.msk}_{l,i})$, and use random scalars $\alpha^j_{s-1}, \beta^j_{s-1}$ during the j^{th} key generation for generating $\mathsf{sI.sk}^j_{l,i}$. Moreover, instances in $\Gamma(s-1)$ are the only instances that use these random scalars. In other words, instances outside $\Gamma(s-1)$ are generated completely independently of instances in $\Gamma(s-1)$, and are identically distributed in H^b_{s-1} and H^b_s. Therefore, to show the indistinguishability of H^b_{s-1} and H^b_s, it suffices to show the indistinguishability of the joint distributions of instances in $\Gamma(s-1)$ in H^b_{s-1} and H^b_s, which we show follows from the multi-instance joint-indistinguishabilty of sIPE.

Note that for any labels $l_1 < l_2$ such that $\tau(l_1) = \tau(l_2) = s-1$, they are not pairable, that is, $\mathbf{pairable}(l_1, l_2) = \bot$. This is because for $\tau(l_1) = \tau(l_2)$ to hold, it must be that l_1 and l_2 are decedents of ($l||0$, $l||2$) or ($l||1$, $l||3$), and hence are not pairable. Therefore, by that joint-SXDH holds in GES, joint-SXDH holds over encodings with labels $\{l||0 : \tau(l) = s-1\}$, as well as over encodings with labels $\{l||1 : \tau(l) = s-1\}$. In other words, joint-SXDH holds in encodings used to instantiate sIPE instances in $\Gamma(s-1)$. Hence, by Lemma 12, the multi-instance joint-indistinguishability property holds for instances in $\Gamma(s-1)$. Observe additionally that the function and input vectors encoded in instances in $\Gamma(s-1)$ have identical inner products in H^b_{s-1} and H^b_s.

$$(Y^j_i||1^M) \cdot (X_i||0^M) = X_i \cdot Y^j_i = \Pi^j[i] = (Y^j_i, 0^{M-1}||\Pi^j[i]) \cdot (0^M, 0^{M-1}||1)$$

Therefore, instances in $\Gamma(s-1)$ are indistinguishable in H^b_{s-1} and H^b_s, and hence so are H^b_{s-1} and H^b_s.

Analysis of $H^b_{2^D}$ In the last hybrid $H^b_{2^D}$, the ciphertext $\mathsf{ct} = \{\mathsf{sI.ct}_{l,i}\}_{l,i}$ encodes the vector $(0^M, 0^{M-1}||1)$, containing no information of x^b. For every function query f^j, the secret key $\mathsf{sk}^j = \{\mathsf{sI.sk}^j_{l,i}\}_{l,i}$ contains in the second slots a randomized encoding Π^j of the computation $f^j(x)$. Towards showing the indistinguishability of $H^0_{2^D}$ and $H^1_{2^D}$, we will eventually apply the perfect security of the randomized encoding scheme ARE, to argue that a randomized encoding Π^j of the computation $f^j(x^0)$ is identically distributed as a randomized encoding of the computation $f^j(x^1)$, since $f^j(x^0) = f^j(x^1)$. But for the perfect security of ARE to hold, it must be the case that randomized encodings Π^j for different function queries are generated using independent random elements. However, this is not true in $H^b_{2^D}$. By construction of $\mathbf{FE}$, the j^{th} randomized encoding Π^j is generated using a vector of random elements $\mathbf{r}^j = \mathbf{r}^j_{\mathsf{sk}}\mathbf{r}_{ct}$, where $\mathbf{r}_{ct}$ are random elements sampled at encryption time and embedded in ct, and $\mathbf{r}^j_{sk}$ are random elements sampled at key generation time and embedded in sk^j. The distributions of $\mathbf{r}^j = \mathbf{r}^j_{\mathsf{sk}}\mathbf{r}_{ct}$ for different j are correlated. This motivates the next sequence of hybrids.

Hybrid $G^b_{j,t}$ Hybrid $G^b_{j,t}$ for $j \in [J]$ and $t \in \{0\} \cup [T]$ (where J is an upper bound on the number of function queries and t an upper bound on the number of random elements for generating AIK randomized encodings), is identical to $G^b_{1,0} = H^b_{2^D}$ except that $\{\Pi^j\}$ are computed with the following random elements.

- For all $h < j$, $\{\Pi^h\}$ is generated using independent random vector $\mathbf{w}^h \xleftarrow{\$} \mathcal{R}^T$, during the key generation for function f^h.
- For all $h > j$, $\{\Pi^h\}$ is generated using a vector $\mathbf{r}^h = \mathbf{r}^h_{sk}\mathbf{r}_{ct}$, where $\mathbf{r}_{ct}$ is the random elements sampled during the encryption of x^b and $\mathbf{r}^h_{sk}$ is sampled during the key generation for function f^h.
- For $h = j$, $\{\Pi^h\}$ is generated using a vector $\mathbf{r}^h$ such that the first t elements are random $\mathbf{r}^h_{\leq t} \xleftarrow{\$} \mathcal{R}^t$ while the rest are the products $\mathbf{r}^h_{>t} = \mathbf{r}^h_{sk,>t}\mathbf{r}_{ct,>t}$.

By construction, for any j, hybrids $G^b_{j-1,T}$ and $G^b_{j,0}$ are identically distributed. Therefore, it suffices to show the indistinguishability between $G^b_{j,t-1}$ and $G^b_{j,t}$.

Indistinguishability of $G^b_{j,t-1}$ and $G^b_{j,t}$ We claim that for every j, t, hybrids $G^b_{j,t-1}$ and $G^b_{j,t}$ are indistinguishable to A. Note that the only difference between them is that in the former, the t^{th} random element r^j_t is $r^j_t = r^j_{sk,t}r_{ct,t}$, whereas in the latter $r^j_t = w^j_t \xleftarrow{\$} \mathcal{R}$ is random.

Recall that by construction of ARE, the i^{th} element in a randomized encoding Π for some computation $f(x)$ using random vector $\mathbf{r}$ is computed as,

$$\Pi[i] = p_i(\mathbf{r})x_{\pi(i)} + q_i(\mathbf{r}) \qquad \text{where } p_i(\mathbf{r}) = \Sigma_{l\in[Q]}\gamma_{i,l}m_l(\mathbf{r}),\ p_i(\mathbf{r}) = \Sigma_{l\in[Q]}\delta_{i,l}m_l(\mathbf{r})$$

where $\{m_l\}_{l\in[Q]}$ is the set of monomials in all $\{p_i, q_i\}_i$. Moreover, p_i, q_i are multilinear and so are all monomial m_l's. Therefore, every element $\Pi[i]$ (viewed as a function over $f, x, \mathbf{r}$) is multilinear over $\mathbf{r}$. Different p_i, q_i functions can depend on different subsets of elements in $\mathbf{r}$. Let $\Lambda(t)$ be the set of indexes i such that p_i or q_i depends on the t^{th} random element r_t. In other words, $\Lambda(t)$ is the set of RE elements $\Pi[i]$ that depends on r_t. (Note that $\Lambda(t)$ is determined by the scheme ARE, and independent of $f, x, \mathbf{r}$.)

By the construction of $\mathbf{FE}$, the i^{th} RE element $\Pi[i]$ is assigned to a unique node l s.t. $i \in S^D_l$. Let $\Psi(t) = \{(i,l) \mid i \in \Lambda(t) \text{ and } i \in S^D_l\}$ be the set of sIPE instances designated for computing RE elements in $\Lambda(t)$. Therefore, $\Psi(t)$ is the set of instances depend on r_t.

Combing back to hybrids $G_{j,t-1}$ and $G_{j,t}$, for every instance $(l,i) \in \Psi(t)$, and every function query f^j, the function key $\mathsf{sl.sk}^j_{l,i}$ encoding vector $(Y^j_i, 0^{M-1}||\Pi^j[i])$, where $Y^j_i = V^j_i \otimes \mathbf{u}_{\pi^j(i)}$ and

$$V^j_i = \gamma_{i,1}m_1(\mathbf{r}^j_{sk})||\cdots||\gamma_{i,Q}m_Q(\mathbf{r}^j_{sk})||\delta_{i,1}m_1(\mathbf{r}^j_{sk})||\cdots||\delta_{i,Q}m_Q(\mathbf{r}^j_{sk})$$

Thus, Y_i^j depends linearly on $r_{sk,t}^j$ and as discussed above $\Pi^j[i]$ depends linearly on r_t^j as well. Furthermore, by construction of sIPE, the function key $\mathsf{sl.sk}_{l,i}^j$ depends linearly on the encoded vectors Y_i^j and $\Pi^j[i]$, meaning that there exist linear functions L_1^j, L_2^j such that $\mathsf{sl.sk}_{l,i}^j$ can be written as $[L_1^j(r_{sk,t}^j)]_l, [L_2^j(r_t^j)]_l$, where L_1^j, L_2^j depends on the master secret key $\mathsf{sl.msk}_{l,i}$ and other random elements for instance (l, i). Furthermore, only function keys $\{\mathsf{sl.sk}_{l,i}^j\}$ of instances in $\Psi(t)$ depend on $r_{sk,t}^j$ and r_t^j. All other objects — master keys, other function keys, and ciphertexts — are generated identically in in $G_{j,t-1}$ and $G_{j,t}$.

Therefore, to show the indistinguishabilty of $G_{j,t-1}$ and $G_{j,t}$, it suffices to show the indistinguishabilty of function keys $\{\mathsf{sl.sk}_{l,i}^j\}$ of instances $(l, i) \in \Psi(t)$, which in turn is implied by the indistinguishabilty of the following distributions

$$\left\{ r_{sk,t}^j,\ r_{ct,t} \xleftarrow{\$} \mathcal{R},\ r_t^j = r_{sk,t}^j r_{ct,t} :\ \left\{ [r_{sk,t}^j]_{l||1}, [r_{ct,t}]_{l||1}, [r_t^j]_{l||1} \right\}_{(l,i)\in\Psi(t)} \right\}$$
$$\left\{ r_{sk,t}^j,\ r_{ct,t},\ w_t^j \xleftarrow{\$} \mathcal{R},\ r_t^j = w_t^j\ :\ \left\{ [r_{sk,t}^j]_{l||1}, [r_{ct,t}]_{l||1}, [r_t^j]_{l||1} \right\}_{(l,i)\in\Psi(t)} \right\}$$

All keys $\mathsf{sl.sk}_{l,i}^j$ in $G_{j,t-1}$ can be generated from the first distribution homomorphically (in an indistinguishable way), and all keys in $G_{j,t}$ can be generated from the second distribution homomorphically. Moreover, the above two distributions are indistinguishable as long as the joint-SXDH assumption holds for encodings under labels $\{l||1\}_{(l,i)\in\Psi(t)}$, which is implied by the joint-SXDH assumption of GES and the following claim. Therefore $G_{j,t-1}$ and $G_{j,t}$ are indistinguishable.

Claim 3. *For every t, every $(l_1, i_1), (l_2, i_2) \in \Psi(t)$, it holds that $\mathbf{pairable}(l_1, l_2) = \bot$.*

Proof. Proof by induction over the depth d of the circuit. When $d = 0$, the claim holds vacuously since $\Pi^0[1] = x_1$ does not depend on any random elements.

Assume that the claim is true for depth-d circuits. That is, for every element $r_t^{\leq d}$ in the random vector $\mathbf{r}^{\leq d}$ for computing Π^d, for every $(l_1, i_1), (l_2, i_2) \in \Psi^d(t)$, it holds that $\mathbf{pairable}(l_1, l_2) = \bot$, where $l_1, l_2 \in \{0, 1, 2, 3\}^d$ and i_1, i_2 are indexes of elements in Π^d. Then we show that the claim holds for all depth-$(d+1)$ circuits. Recall that the random vector $\mathbf{r}^{\leq d+1}$ for computing Π^{d+1} consists of two parts $\mathbf{r}^{\leq d}$ and $\mathbf{r}^{d+1}$. Consider two cases:

- *Case 1: $r_t^{\leq d+1}$ is an element in $\mathbf{r}^{d+1}$.* Recall that by construction of ARE, when encoding a depth-$(d + 1)$ computation, for every $i \in [\ell^d]$, three random vectors $r_{i,0}^{d+1}, r_{i,1}^{d+1}, r_{i,2}^{d+1}$ are sampled, and used respectively in the generation of the following RE elements $(\Pi^{d+1}[4i + 0], \Pi^{d+1}[4i+3])$, $(\Pi^{d+1}[4i+1], \Pi^{d+1}[4i+2], \Pi^{d+1}[4i+3])$, and $(\Pi^{d+1}[4i+2], \Pi^{d+1}[4i+3])$. The four RE elements are respectively assigned to node $l||0$, $l||1$, $l||2$ and $l||2$ for some $l \in \{0, 1\}^d$. Hence $\Psi(i, 0) = \{(l||0, 4i + 0), (l||2, 4i + 3)\}$, $\Psi(i, 1) = \{(l||1, 4i + 1), (l||2, 4i + 2), (l||2, 4i + 3)\}$ and $\Psi(i, 3) = \{(l||2, 4i + 2), (l||2, 4i + 3)\}$. It is easy to verify that for every $t = (i, c)$, the claim holds.

- *Case 2: $r_t^{\leq d+1}$ is an element in $\mathbf{r}^{\leq d}$.* Let $r_t^{\leq d+1} = r_{t'}^{\leq d}$ for some t'. By the induction hypothesis, no two labels in $\Psi^d(t')$ are pairable. Additionally, by the construction of ARE, for every $(l, i) \in \Psi^d(t')$, the RE element $\Pi^d[i]$ is encoded in 4 elements $\Pi^{d+1}[4i \cdots 4i + 3]$ in Π^{d+1}. Moreover, only $\Pi^{d+1}[4i + 1]$ and $\Pi^{d+1}[4i + 3]$ depend on random elements in $\mathbf{r}^{\leq d}$ (the other two elements depend only on elements in $\mathbf{r}^{d+1}$.) Therefore, $\Psi^{d+1}(t) = \{(l||1, 4i + 1),\ (l||2, 4i + 3) \mid (l, i) \in \Psi^d(t')\}$. By the fact that no two labels in $\Psi^d(t')$ are pairable, the same holds for $\Psi^{d+1}(t)$.

□

□

Acknowledgements. The authors thank Hoeteck Wee for being the intellectual inspiration behind this work, Stefano Tessaro for many helpful inputs and insights, and Benny Applebaum for being a quick and reliable oracle. Huijia Lin was partially supported by NSF grants CNS-1528178 and CNS-1514526. Vinod Vaikuntanathan was supported by NSF Grants CNS-1350619 and CNS-1414119 and by the U.S. Army Research Office under contract W911NF-15-C-0226.

References

[AB15] Benny Applebaum and Zvika Brakerski. Obfuscating circuits via composite-order graded encoding. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *TCC 2015, Part II*, volume 9015 of *LNCS*, pages 528–556, Warsaw, Poland, March 23–25, 2015. Springer, Heidelberg, Germany.

[ABCP16] Michel Abdalla, Florian Bourse, Angelo De Caro, and David Pointcheval. Better security for functional encryption for inner product evaluations. *IACR Cryptology ePrint Archive*, 2016:11, 2016.

[ABDP15] Michel Abdalla, Florian Bourse, Angelo De Caro, and David Pointcheval. Simple functional encryption schemes for inner products. In Jonathan Katz, editor, *PKC 2015*, volume 9020 of *LNCS*, pages 733–751, Gaithersburg, MD, USA, March 30 – April 1, 2015. Springer, Heidelberg, Germany.

[ABSV15] Prabhanjan Ananth, Zvika Brakerski, Gil Segev, and Vinod Vaikuntanathan. From selective to adaptive security in functional encryption. In Rosario Gennaro and Matthew Robshaw, editors, *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part II*, volume 9216 of *Lecture Notes in Computer Science*, pages 657–677. Springer, 2015.

[AGIS14] Prabhanjan Vijendra Ananth, Divya Gupta, Yuval Ishai, and Amit Sahai. Optimizing obfuscation: Avoiding Barrington's theorem. In Gail-Joon Ahn, Moti Yung, and Ninghui Li, editors, *ACM CCS 14*, pages 646–658, Scottsdale, AZ, USA, November 3–7, 2014. ACM Press.

[AIK04] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. Cryptography in nc^0. In *FOCS*, pages 166–175, 2004.

[AIK06] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. Computationally private randomizing polynomials and their applications. *Computational Complexity*, 15(2):115–162, 2006.

[AIK08] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. On pseudorandom generators with linear stretch in nc^0. *Computational Complexity*, 17(1):38–69, 2008.

[AIK11] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. How to garble arithmetic circuits. In Rafail Ostrovsky, editor, *52nd FOCS*, pages 120–129, Palm Springs, California, USA, October 22–25, 2011. IEEE Computer Society Press.

[AIK14] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. How to garble arithmetic circuits. *SIAM J. Comput.*, 43(2):905–929, 2014.

[AJ15] Prabhanjan Ananth and Abhishek Jain. Indistinguishability obfuscation from compact functional encryption. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part I*, volume 9215 of *LNCS*, pages 308–326, Santa Barbara, CA, USA, August 16–20, 2015. Springer, Heidelberg, Germany.

[AJS15] Prabhanjan Ananth, Abhishek Jain, and Amit Sahai. Achieving compactness generically: Indistinguishability obfuscation from non-compact functional encryption. *IACR Cryptology ePrint Archive*, 2015:730, 2015.

[AL15] Benny Applebaum and Shachar Lovett. Algebraic attacks against random local functions and their countermeasures. *Electronic Colloquium on Computational Complexity (ECCC)*, 22:172, 2015.

[Ale03] Michael Alekhnovich. More on average case vs approximation complexity. In *44th Symposium on Foundations of Computer Science (FOCS 2003), 11-14 October 2003, Cambridge, MA, USA, Proceedings*, pages 298–307. IEEE Computer Society, 2003.

[App14] Benny Applebaum. *Cryptography in Constant Parallel Time*. Information Security and Cryptography. Springer, 2014.

[BGI$^+$01a] Boaz Barak, Oded Goldreich, Rusell Impagliazzo, Steven Rudich, Amit Sahai, Salil Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. In *Advances in Cryptology CRYPTO 2001*, pages 1–18. Springer, 2001.

[BGI$^+$01b] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 1–18, Santa Barbara, CA, USA, August 19–23, 2001. Springer, Heidelberg, Germany.

[BGJ$^+$16] Nir Bitansky, Shafi Goldwasser, Abhishek Jain, Omer Paneth, Vinod Vaikuntanathan, and Brent Waters. Time-lock puzzles from randomized encodings. In Madhu Sudan, editor, *Proceedings of the 2016 ACM Conference on Innovations in Theoretical Computer Science, Cambridge, MA, USA, January 14-16, 2016*, pages 345–356. ACM, 2016.

[BGK$^+$13] Boaz Barak, Sanjam Garg, Yael Tauman Kalai, Omer Paneth, and Amit Sahai. Protecting obfuscation against algebraic attacks. In *EuroCrypt'14*, 2013.

[BGK$^+$14a] Boaz Barak, Sanjam Garg, Yael Tauman Kalai, Omer Paneth, and Amit Sahai. Protecting obfuscation against algebraic attacks. In Phong Q. Nguyen and Elisabeth Oswald, editors, *Advances in Cryptology - EUROCRYPT 2014 - 33rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Copenhagen, Denmark, May 11-15, 2014. Proceedings*, volume 8441 of *Lecture Notes in Computer Science*, pages 221–238. Springer, 2014.

[BGK$^+$14b] Boaz Barak, Sanjam Garg, Yael Tauman Kalai, Omer Paneth, and Amit Sahai. Protecting obfuscation against algebraic attacks. In Phong Q. Nguyen and Elisabeth Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 221–238, Copenhagen, Denmark, May 11–15, 2014. Springer, Heidelberg, Germany.

[BJK15] Allison Bishop, Abhishek Jain, and Lucas Kowalczyk. Function-hiding inner product encryption. In Tetsu Iwata and Jung Hee Cheon, editors, *Advances in Cryptology - ASIACRYPT 2015 - 21st International Conference on the Theory and Application of Cryptology and Information Security, Auckland, New Zealand, November 29 - December 3, 2015, Proceedings, Part I*, volume 9452 of *Lecture Notes in Computer Science*, pages 470–491. Springer, 2015.

[BNPW16] Nir Bitansky, Ryo Nishimaki, Alain Passelègue, and Daniel Wichs. From cryptomania to obfustopia through secret-key functional encryption. *IACR Cryptology ePrint Archive*, 2016:558, 2016.

[BPR12] Abhishek Banerjee, Chris Peikert, and Alon Rosen. Pseudorandom functions and lattices. In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 719–737, Cambridge, UK, April 15–19, 2012. Springer, Heidelberg, Germany.

[BPR15] Nir Bitansky, Omer Paneth, and Alon Rosen. On the cryptographic hardness of finding a nash equilibrium. In Guruswami [Gur15], pages 1480–1498.

[BPW16] Nir Bitansky, Omer Paneth, and Daniel Wichs. Perfect structure on the edge of chaos - trapdoor permutations from indistinguishability obfuscation. In *Theory of Cryptography - 13th International Conference, TCC 2016-A, Tel Aviv, Israel, January 10-13, 2016, Proceedings, Part I*, pages 474–502, 2016.

[BQ12] Andrej Bogdanov and Youming Qiao. On the security of goldreich's one-way function. *Computational Complexity*, 21(1):83–127, 2012.

[BR14] Zvika Brakerski and Guy N. Rothblum. Virtual black-box obfuscation for all circuits via generic graded encoding. In Yehuda Lindell, editor, *TCC 2014*, volume 8349 of *LNCS*, pages 1–25, San Diego, CA, USA, February 24–26, 2014. Springer, Heidelberg, Germany.

[BS02] Dan Boneh and Alice Silverberg. Applications of multilinear forms to cryptography. *IACR Cryptology ePrint Archive*, 2002:80, 2002.

[BSW12] Dan Boneh, Amit Sahai, and Brent Waters. Functional encryption: a new vision for public-key cryptography. *Commun. ACM*, 55(11):56–64, 2012.

[BV15] Nir Bitansky and Vinod Vaikuntanathan. Indistinguishability obfuscation from functional encryption. In *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 171–190, 2015.

[CEMT09] James Cook, Omid Etesami, Rachel Miller, and Luca Trevisan. Goldreich's one-way function candidate and myopic backtracking algorithms. In *TCC*, pages 521–538, 2009.

[CHN$^+$15] Aloni Cohen, Justin Holmgren, Ryo Nishimaki, Vinod Vaikuntanathan, and Daniel Wichs. Watermarking cryptographic capabilities. *IACR Cryptology ePrint Archive*, 2015:1096, 2015. To Appear in ACM STOC 2016.

[CLT13] Jean-Sébastien Coron, Tancrède Lepoint, and Mehdi Tibouchi. Practical multilinear maps over the integers. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part I*, volume 8042 of *LNCS*, pages 476–493, Santa Barbara, CA, USA, August 18–22, 2013. Springer, Heidelberg, Germany.

[CLTV15] Ran Canetti, Huijia Lin, Stefano Tessaro, and Vinod Vaikuntanathan. Obfuscation of probabilistic circuits and applications. In Dodis and Nielsen [DN15], pages 468–497.

[CM01] M. Cryan and P. B. Miltersen. On pseudorandom generators in nc0. In Proc. 26th MFCS, 2001.

[CV13] Ran Canetti and Vinod Vaikuntanathan. Obfuscating branching programs using black-box pseudo-free groups. *IACR Cryptology ePrint Archive*, 2013:500, 2013.

[DDM16] Pratish Datta, Ratna Dutta, and Sourav Mukhopadhyay. Functional encryption for inner product with full function privacy. In Chen-Mou Cheng, Kai-Min Chung, Giuseppe Persiano, and Bo-Yin Yang, editors, *Public-Key Cryptography - PKC 2016 - 19th IACR International Conference on Practice and Theory in Public-Key Cryptography, Taipei, Taiwan, March 6-9, 2016, Proceedings, Part I*, volume 9614 of *Lecture Notes in Computer Science*, pages 164–195. Springer, 2016.

[DN15] Yevgeniy Dodis and Jesper Buus Nielsen, editors. *Theory of Cryptography - 12th Theory of Cryptography Conference, TCC 2015, Warsaw, Poland, March 23-25, 2015, Proceedings, Part II*, volume 9015 of *Lecture Notes in Computer Science*. Springer, 2015.

[GGH13a] Sanjam Garg, Craig Gentry, and Shai Halevi. Candidate multilinear maps from ideal lattices. In Thomas Johansson and Phong Q. Nguyen, editors, *Advances in Cryptology - EUROCRYPT 2013, 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings*, volume 7881 of *Lecture Notes in Computer Science*, pages 1–17. Springer, 2013.

[GGH13b] Sanjam Garg, Craig Gentry, and Shai Halevi. Candidate multilinear maps from ideal lattices. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 1–17, Athens, Greece, May 26–30, 2013. Springer, Heidelberg, Germany.

[GGH^{+}13c] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA*, pages 40–49. IEEE Computer Society, 2013.

[GGH15] Craig Gentry, Sergey Gorbunov, and Shai Halevi. Graph-induced multilinear maps from lattices. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *TCC 2015, Part II*, volume 9015 of *LNCS*, pages 498–527, Warsaw, Poland, March 23–25, 2015. Springer, Heidelberg, Germany.

[GGHR14] Sanjam Garg, Craig Gentry, Shai Halevi, and Mariana Raykova. Two-round secure MPC from indistinguishability obfuscation. In Yehuda Lindell, editor, *Theory of Cryptography - 11th Theory of Cryptography Conference, TCC 2014, San Diego, CA, USA, February 24-26, 2014. Proceedings*, volume 8349 of *Lecture Notes in Computer Science*, pages 74–94. Springer, 2014.

[GGHZ16] Sanjam Garg, Craig Gentry, Shai Halevi, and Mark Zhandry. Functional encryption without obfuscation. In Eyal Kushilevitz and Tal Malkin, editors, *Theory of Cryptography - 13th International Conference, TCC 2016-A, Tel Aviv, Israel, January 10-13, 2016, Proceedings, Part II*, volume 9563 of *Lecture Notes in Computer Science*, pages 480–511. Springer, 2016.

[GHRW14] Craig Gentry, Shai Halevi, Mariana Raykova, and Daniel Wichs. Outsourcing private RAM computation. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pages 404–413. IEEE Computer Society, 2014.

[GLSW15] Craig Gentry, Allison Bishop Lewko, Amit Sahai, and Brent Waters. Indistinguishability obfuscation from the multilinear subgroup elimination assumption. In Guruswami [Gur15], pages 151–170.

[GLW14] Craig Gentry, Allison B. Lewko, and Brent Waters. Witness encryption from instance independent assumptions. In Juan A. Garay and Rosario Gennaro, editors, *Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part I*, volume 8616 of *Lecture Notes in Computer Science*, pages 426–443. Springer, 2014.

[GMS16] Sanjam Garg, Pratyay Mukherjee, and Akshayaram Srinivasan. Obfuscation without the vulnerabilities of multilinear maps. *IACR Cryptology ePrint Archive*, 2016:390, 2016.

[GP15] Sanjam Garg and Antigoni Polychroniadou. Two-round adaptively secure MPC from indistinguishability obfuscation. In Dodis and Nielsen [DN15], pages 614–637.

[GR07] Shafi Goldwasser and Guy N. Rothblum. On best-possible obfuscation. In Salil P. Vadhan, editor, *Theory of Cryptography, 4th Theory of Cryptography Conference, TCC 2007, Amsterdam, The Netherlands, February 21-24, 2007, Proceedings*, volume 4392 of *Lecture Notes in Computer Science*, pages 194–213. Springer, 2007.

[GS16] Sanjam Garg and Akshayaram Srinivasan. Unifying security notions of functional encryption. *IACR Cryptology ePrint Archive*, 2016:524, 2016.

[Gur15] Venkatesan Guruswami, editor. *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*. IEEE Computer Society, 2015.

[HY16] Pavel Hubácek and Eylon Yogev. Hardness of continuous local search: Query complexity and cryptographic lower bounds. *Electronic Colloquium on Computational Complexity (ECCC)*, 23:63, 2016.

[IK02] Yuval Ishai and Eyal Kushilevitz. Perfect constant-round secure computation via perfect randomizing polynomials. In *ICALP*, pages 244–256, 2002.

[KSW08] Jonathan Katz, Amit Sahai, and Brent Waters. Predicate encryption supporting disjunctions, polynomial equations, and inner products. In *EUROCRYPT 2008*, pages 146–162, 2008.

[Lin16] Huijia Lin. Indistinguishability obfuscation from constant-degree graded encoding schemes, 2016. To Appear in Eurocrypt'16.

[LM16] Baiyu Li and Daniele Micciancio. Compactness vs collusion resistance in functional encryption. *IACR Cryptology ePrint Archive*, 2016:561, 2016.

[LOS$^+$10] Allison B. Lewko, Tatsuaki Okamoto, Amit Sahai, Katsuyuki Takashima, and Brent Waters. Fully secure functional encryption: Attribute-based encryption and (hierarchical) inner product encryption. In Henri Gilbert, editor, *EUROCRYPT 2010*, volume 6110 of *LNCS*, pages 62–91, French Riviera, May 30 – June 3, 2010. Springer, Heidelberg, Germany.

[MMN16a] Mohammad Mahmoody, Ameer Mohammed, and Soheil Nematihaji. On the impossibility of virtual black-box obfuscation in idealized models. In *Theory of Cryptography - 13th International Conference, TCC 2016-A, Tel Aviv, Israel, January 10-13, 2016, Proceedings, Part I*, pages 18–48, 2016.

[MMN$^+$16b] Mohammad Mahmoody, Ameer Mohammed, Soheil Nematihaji, Rafael Pass, and Abhi Shelat. Lower bounds on assumptions behind indistinguishability obfuscation. In *Theory of Cryptography - 13th International Conference, TCC 2016-A, Tel Aviv, Israel, January 10-13, 2016, Proceedings, Part I*, pages 49–66, 2016.

[MST03] Elchanan Mossel, Amir Shpilka, and Luca Trevisan. On e-biased generators in NC0. In *44th Symposium on Foundations of Computer Science (FOCS 2003), 11-14 October 2003, Cambridge, MA, USA, Proceedings*, pages 136–145, 2003.

[MSZ16a] Eric Miles, Amit Sahai, and Mark Zhandry. Annihilation attacks for multilinear maps: Cryptanalysis of indistinguishability obfuscation over GGH13. *IACR Cryptology ePrint Archive*, 2016:147, 2016.

[MSZ16b] Eric Miles, Amit Sahai, and Mark Zhandry. Secure obfuscation in a weak multilinear map model: A simple construction secure against all known attacks. *IACR Cryptology ePrint Archive*, 2016:588, 2016.

[NR95] Moni Naor and Omer Reingold. Synthesizers and their application to the parallel construction of pseudo-random functions. In *36th FOCS*, pages 170–181, Milwaukee, Wisconsin, October 23–25, 1995. IEEE Computer Society Press.

[NR97] Moni Naor and Omer Reingold. Number-theoretic constructions of efficient pseudorandom functions. In *38th FOCS*, pages 458–467, Miami Beach, Florida, October 19–22, 1997. IEEE Computer Society Press.

[NRR00] Moni Naor, Omer Reingold, and Alon Rosen. Pseudo-random functions and factoring (extended abstract). In *32nd ACM STOC*, pages 11–20, Portland, Oregon, USA, May 21–23, 2000. ACM Press.

[NS94] Noam Nisan and Mario Szegedy. On the degree of boolean functions as real polynomials. *Computational Complexity*, 4:301–313, 1994.

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

[OT08] Tatsuaki Okamoto and Katsuyuki Takashima. Homomorphic encryption and signatures from vector decomposition. In Steven D. Galbraith and Kenneth G. Paterson, editors, *Pairing-Based Cryptography - Pairing 2008, Second International Conference, Egham, UK, September 1-3, 2008. Proceedings*, volume 5209 of *Lecture Notes in Computer Science*, pages 57–74. Springer, 2008.

[OT09] Tatsuaki Okamoto and Katsuyuki Takashima. Hierarchical predicate encryption for inner-products. In Mitsuru Matsui, editor, *Advances in Cryptology - ASIACRYPT 2009, 15th International Conference on the Theory and Application of Cryptology and Information Security, Tokyo, Japan, December 6-10, 2009. Proceedings*, volume 5912 of *Lecture Notes in Computer Science*, pages 214–231. Springer, 2009.

[OW14] Ryan O'Donnell and David Witmer. Goldreich's PRG: evidence for near-optimal polynomial stretch. In *IEEE 29th Conference on Computational Complexity, CCC 2014, Vancouver, BC, Canada, June 11-13, 2014*, pages 1–12, 2014.

[PS16] Rafael Pass and Abhi Shelat. Impossibility of VBB obfuscation with ideal constant-degree graded encodings. In *Theory of Cryptography - 13th International Conference, TCC 2016-A, Tel Aviv, Israel, January 10-13, 2016, Proceedings, Part I*, pages 3–17, 2016.

[PST14] Rafael Pass, Karn Seth, and Sidharth Telang. Indistinguishability obfuscation from semantically-secure multilinear encodings. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part I*, volume 8616 of *LNCS*, pages 500–517, Santa Barbara, CA, USA, August 17–21, 2014. Springer, Heidelberg, Germany.

[SW14] Amit Sahai and Brent Waters. How to use indistinguishability obfuscation: deniable encryption, and more. In David B. Shmoys, editor, *46th ACM STOC*, pages 475–484, New York, NY, USA, May 31 – June 3, 2014. ACM Press.

[Wat15] Brent Waters. A punctured programming approach to adaptively secure functional encryption. In Rosario Gennaro and Matthew Robshaw, editors, *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part II*, volume 9216 of *Lecture Notes in Computer Science*, pages 678–697. Springer, 2015.

[Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *FOCS*, pages 162–167, 1986.

[Zim15] Joe Zimmerman. How to obfuscate programs directly. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 439–467, Sofia, Bulgaria, April 26–30, 2015. Springer, Heidelberg, Germany.