Projective Arithmetic Functional Encryption and Indistinguishability Obfuscation From Degree-5 Multilinear Maps

Prabhanjan Ananth[*]
prabhanjan@cs.ucla.edu

Amit Sahai[†]
sahai@cs.ucla.edu

Abstract

In this work, we propose a variant of functional encryption called *projective arithmetic functional encryption* (PAFE). Roughly speaking, our notion is like functional encryption for arithmetic circuits, but where secret keys only yield partially decrypted values. These partially decrypted values can be linearly combined with known coefficients and the result can be tested to see if it is a small value.

We give a *degree-preserving* construction of PAFE from multilinear maps. That is, we show how to achieve PAFE for arithmetic circuits of degree d using only degree-d multilinear maps. Our construction is based on an assumption over such multilinear maps, that we justify in a generic model. We then turn to applying our notion of PAFE to one of the most pressing open problems in the foundations of cryptography: building secure indistinguishability obfuscation ($\mathsf{i}\mathcal{O}$) from simpler building blocks.

$\mathsf{i}\mathcal{O}$ from degree-5 multilinear maps. Recently, the works of Lin [Eurocrypt 2016] and Lin-Vaikuntanathan [FOCS 2016] showed how to build $\mathsf{i}\mathcal{O}$ from constant-degree multilinear maps. However, no explicit constant was given in these works, and an analysis of these published works shows that the degree requirement would be in excess of 30. The ultimate "dream" goal of this line of work would be to reduce the degree requirement all the way to 2, allowing for the use of well-studied bilinear maps, or barring that, to a low constant that may be supportable by alternative secure low-degree multilinear map candidates. We make substantial progress toward this goal by showing how to leverage PAFE for degree-5 arithmetic circuits to achieve $\mathsf{i}\mathcal{O}$, thus yielding the first $\mathsf{i}\mathcal{O}$ construction from degree-5 multilinear maps.

[*]Affiliated with Center for Encrypted Functionalities and Department of Computer Science, UCLA. This work is partially supported by grant #360584 from the Simons Foundation and the grants under Amit Sahai.

[†]Affiliated with Center for Encrypted Functionalities and Department of Computer Science, UCLA.Research supported in part from a DARPA/ARL SAFEWARE award, NSF Frontier Award 1413955, NSF grants 1619348, 1228984, 1136174, and 1065276, a Xerox Faculty Research Award, a Google Faculty Research Award, an equipment grant from Intel, and an Okawa Foundation Research Grant. This material is based upon work supported by the Defense Advanced Research Projects Agency through the ARL under Contract W911NF-15-C-0205. The views expressed are those of the author and do not reflect the official policy or position of the Department of Defense, the National Science Foundation, or the U.S. Government.

Contents

1 Introduction

Functional encryption (FE), introduced by Sahai and Waters [SW05, SW08], allows for the creation of secret keys sk_f corresponding to functions f, such that when such a secret key sk_f is applied to an encryption of x, decryption yields $f(x)$ but, intuitively speaking, nothing more is revealed about x. In this work, we will focus on the secret-key variant of FE where knowledge of the master secret key is needed to perform encryption. Functional encryption has proven to be remarkably versatile: it captures as special cases efficient applications like attribute-based encryption for formulas [GPSW06, BSW07] and predicate encryption for inner products [KSW08] from bilinear maps. At the same time, the general notion of functional encryption implies remarkably powerful primitives, including most notably indistinguishability obfuscation (i$\mathcal{O}$) [AJ15, BV15, AJS15, BNPW16].

In this work, we continue the study of functional encryption notions, constructions, and implications. As a byproduct of our study, we tackle the one of the most pressing open problems in theoretical cryptography: building secure i$\mathcal{O}$ from simpler building blocks. In particular, we give the first construction of i$\mathcal{O}$ using only degree-5 multilinear maps.

FE in the arithmetic context. For a number of cryptographic objects that deal with general computations, *arithmetic* circuits have been considered in addition to boolean circuits. The primary motivation for this arises when we wish to apply these objects to cryptographic computations, since many cryptographic computations can be better expressed as arithmetic circuits rather than boolean circuits. For example, zero-knowledge proofs [GMR89] for arithmetic circuits (e.g. [GS08] in the bilinear setting) have been influential because they allow for the construction of zero-knowledge protocols whose structure and complexity more closely match the structure and complexity of algebraic cryptographic algorithms.

In a similar spirit, we study general FE in the context where secret keys should correspond to arithmetic circuits. Notably however, our motivation will not (primarily) be efficiency, but rather achieving new feasibility results, as we will elaborate below.

Previous work has studied FE for arithmetic circuits in two special cases: The work of Boneh et al. [BNS13, BGG$^+$14] studied attribute-based encryption for arithmetic circuits from the LWE assumption. (Our work will diverge technically from this.) Another line of work started with the work of Katz, Sahai, and Waters [KSW08], studying FE where secret keys corresponded to arithmetic inner product computations, using bilinear groups as the underlying cryptographic tool. There has been several followup papers on FE for inner products [ABCP15, AAB$^+$15, BJK15, ABCP16, DDM16, LV16] with various security notions and correctness properties. An issue that will be important to us, and that arises already in the context of inner products, concerns the *correctness* property of the FE scheme. Ideally, a secret key for an arithmetic circuit C, when applied to an encryption of x, should allow the decryptor to learn $C(x)$. However, FE constructions typically store values "in the exponent," and thus the difficulty of discrete logarithms in bilinear groups implies that if $C(x)$ is superpolynomial, it will be difficult to recover. This issue has been dealt with in the past either by requiring that decryption only reveals whether $C(x) = 0$, as in [KSW08], or by requiring that decryption only reveals $C(x)$ if $C(x)$ is polynomially bounded, such as in the works of Abdalla et al. and others [ABCP15, BJK15, ABCP16, DDM16]. We will diverge from past work when dealing with this issue, in order to provide greater flexibility, and in so doing, we introduce our notion of *projective*[1]

[1]We call our notion *projective* FE because, roughly speaking, a user holding a collection of keys $\{sk_C\}_C$ for several arithmetic circuits C can only learn information about various *linear projections* $\sum_C \alpha_C C(x)$ for known small coefficients $\{\alpha_C\}_C$. We discuss this in more detail below. Our name is also loosely inspired by the notion of projective hash functions, introduced by Cramer and Shoup [CS02], where keys (called projective keys) only allow one to evaluate the hash function on inputs x in some NP language, but not on all strings. In our setting, as well, our keys are similarly only "partially functional" in that they only allow the user to learn information about various

arithmetic FE.

1.1 Our Contributions

Projective Arithmetic FE (PAFE). In projective arithmetic FE, like in FE, encrypting a value x yields a ciphertext c. Also like in (arithmetic) FE, in PAFE each secret key sk_C is associated with an *arithmetic* circuit[2] C. However, unlike in FE, in PAFE when the secret key sk_C is applied to the ciphertext c, it does not directly yield the decrypted value $C(x)$, but rather this yields a partial decryption p_C. We call this process *projective decryption*. We envision a party holding a collection of secret keys $\{sk_C\}_C$ would apply projective decryption using these secret keys to the ciphertext c to obtain a collection of partial decryptions $\{p_C\}_C$. Finally, this party can choose any collection of small coefficients $\{\alpha_C\}_C$ arbitrarily, and then call a different efficient *recovery* algorithm which is given all the partial decryptions $\{p_C\}_C$ and coefficients $\{\alpha_C\}_C$. The recovery algorithm then outputs a bit that indicates whether $\sum_C \alpha_C C(x) = 0$ or not. (More generally, we can allow the user to recover the value of $\sum_C \alpha_C C(x)$ as long as it is bounded by a polynomial.)

Thus, projective arithmetic FE can be seen as relaxing the correctness guarantee that would be provided by the standard notion of FE when applied to arithmetic circuits over fields of superpolynomial size (which is not known to be achievable). Of course, if decryption actually allowed a user to learn $\{C(x)\}_C$ for several arithmetic circuits C, then the user would be able to compute $\sum_C \alpha_C C(x)$ for any set of small coefficients $\{\alpha_C\}_C$ of her choice. Note that our notion is more permissive than *only* revealing whether $C(x) = 0$, as in the original work for FE for inner products [KSW08], or only revealing $C(x)$ if it is polynomially bounded, such as in other works on FE for inner products [ABCP15, BJK15, ABCP16, DDM16]. With regard to security, our notion will, intuitively speaking, only require indistinguishability of encryptions of x from encryptions of y, if $C(x) = C(y)$ for all secret keys sk_C obtained by the adversary. However, for our application of PAFE to iO, we require a stronger notion of security that we call *semi-functional security*. We give an intuitive explanation of this notion in the technical overview.

Degree-preserving construction of PAFE from multilinear maps. The first main technical contribution of our work is a construction of (secret-key) PAFE for degree-d arithmetic circuits, from degree-d *asymmetric* multilinear maps[3]. Furthermore, it suffices that the groups over which the multilinear maps are defined are prime order. Our construction is based on an explicit pair of assumptions over such multilinear maps, that we can justify in the standard generic multilinear model.

Theorem 1 (Informal). *There exists secret-key PAFE for degree-d arithmetic circuits from degree-d prime order asymmetric multilinear maps under Assumptions #1 and #2 (see Section 7.2.1).*

Our assumptions do not require any low-level encodings of 0 to be given to the adversary, and we thus believe them to be instantiable using existing candidate multilinear maps. Indeed, because of some pseudorandomness properties of our construction and generic proof of security, we believe that our assumptions can be proven secure in the Weak MMap model considered in the works of Miles et al. and Garg et al. [MSZ16, GMM+16], which would give further evidence of its instantiability. Because we want to posit instantiable assumptions, we do not formulate

linear projections, and they do not in general reveal the full information that should be learned by obtaining all $C(x)$ values. However, to the best of our knowledge, only this loose relationship exists between projective hash functions and our notion of projective FE.

[2]We only are interested in arithmetic circuits of fan-in 2.

[3]Roughly speaking, asymmetric multilinear maps disallows pairing of elements from the same group structure.

a succinct version of our assumption together with a reduction of security as was done in the works of Gentry et al. or Lin and Vaikuntanathan [GLSW15, LV16], because unfortunately no existing candidate multilinear map construction is known to securely support such reductions, and indeed the assumptions of [GLSW15, LV16] are broken when instantiated with existing candidates. We stress that, like in the recent work of [Lin16, LV16], if the degree d is constant, then our pair of assumptions would only involve a constant-degree multilinear map.

Our construction can be seen as a generalizing FE for inner products (degree 2 functions) from bilinear maps, to higher degrees in a degree preserving manner. Thus, our construction can be applied to cryptographic computations that are naturally represented as arithmetic functions of low degree, but not as inner products. In more detail, we introduce the notion of slotted encodings that has the same flavor of multilinear maps defined over composite order groups. We then show how to emulate slotted encodings using prime-order multilinear maps. However, this emulation strategy only works in the case of constant degree. We hope that this technique will be useful to transform constructions based on constant degree composite order multilinear maps (for example [Lin16]) to constructions based on constant degree prime order multilinear maps.

$\mathsf{i}\mathcal{O}$ from degree-5 multilinear maps. Our motivation for building PAFE for arithmetic circuits in a degree-preserving manner is to achieve new feasibility results for $\mathsf{i}\mathcal{O}$ from low-degree multilinear maps. The concept of $\mathsf{i}\mathcal{O}$ was first defined by Barak et al. [BGI$^+$01]. Informally speaking, $\mathsf{i}\mathcal{O}$ converts a program (represented by a boolean circuit) into a "pseudo-canonical form." That is, for any two equivalent programs P_0, P_1 of the same size, we require that $\mathsf{i}\mathcal{O}(P_0)$ is computationally indistinguishable from $\mathsf{i}\mathcal{O}(P_1)$. The first candidate construction of $\mathsf{i}\mathcal{O}$ was given by Garg et al. [GGH$^+$13b], and especially since the introduction of punctured programming techniques of Sahai and Waters [SW14], $\mathsf{i}\mathcal{O}$ has found numerous applications, with numerous papers published since 2013 that use $\mathsf{i}\mathcal{O}$ to accomplish cryptographic tasks that were not known to be feasible before (see, e.g., [GGH$^+$13b, SW14, GGHR14, HSW14, GGG$^+$14, BPR15, BP15, CHN$^+$16, BGJ$^+$16]). However, it is still not known how to build $\mathsf{i}\mathcal{O}$ from standard cryptographic assumptions. Given the enormous applicability of $\mathsf{i}\mathcal{O}$ to a wide variety of cryptographic problems, one of the most pressing open problems in the foundations of cryptography is to find ways to construct $\mathsf{i}\mathcal{O}$ from simpler building blocks. Indeed, while there have been dozens of papers published showing how to use $\mathsf{i}\mathcal{O}$ to accomplish amazing things, only a handful of papers have explored simpler building blocks that suffice for constructing $\mathsf{i}\mathcal{O}$.

One line of work toward this objective is by Lin [Lin16] and Lin and Vaikuntanathan [LV16], who showed how to build $\mathsf{i}\mathcal{O}$ from constant-degree multilinear maps. Unfortunately, no explicit constant was given in these works, and an analysis of these published works shows that the degree requirement would be in excess of 100. The ultimate "dream" goal of this line of work would be to reduce the degree requirement all the way to 2, allowing for the use of well-studied bilinear maps, or barring that, to a low constant that may be supportable by alternative secure low-degree multilinear map candidates.

We make substantial progress toward this goal by showing how to achieve $\mathsf{i}\mathcal{O}$ starting from PAFE. Specifically, we first construct ε-sublinear secret key functional encryption for NC^1 circuits, with constant $\varepsilon < 1$, starting from PAFE[4] for degree-d arithmetic circuits and a specific type of degree d-randomizing polynomials [IK00, AIK06][5]. We require that the randomizing polynomials satisfy some additional properties such as the encoding polynomials should be homogenous, the randomness complexity[6] is ε-sub-linear in the circuit size and the decod-

[4]We additionally require that PAFE has encryption complexity to be multiplicative overhead in the message size. Our construction of PAFE satisfies this property.

[5]The degree of a randomizing polynomial is defined to be the maximum degree of the polynomials computing the encoding function.

[6]Randomness complexity in this context refers to the size of the random string used in the encoding algorithm.

ing algorithm should be executed as a sequence of linear functions. We call a scheme that satisfies these additional properties as homogenous randomizing polynomials with ε-sub-linear randomness complexity. As we will see later, we can achieve ε-sub-linear randomness complexity property for free by employing an appropriate pseudorandom generator of $\frac{1}{\varepsilon'}$-stretch, where constant $\varepsilon' > 1$ is related to ε. Hence, we only care about constructing homogenous randomizing polynomials (without sublinear property) and we provide an information theoretic construction achieving the same.

Once we construct ε-sublinear secret key functional encryption, we can then invoke the result of [BNPW16] and additionally assume learning with errors to obtain iO. For this transformation, we are required to assume that the underlying FE scheme and learning with errors is sub-exponentially secure. Thus,

Theorem 2 (Informal)**.** *We construct an indistinguishability obfuscation scheme for P/poly assuming the following: for some constant d,*

1. *Sub-exponentially secure PAFE scheme for degree d arithmetic circuits with multiplicative overhead in encryption complexity. From Theorem 1, this can be based on sub-exponentially secure Assumptions #1 and #2 (Section 7.2.1).*
2. *Sub-exponentially secure degree d homogenous randomizing polynomials with ε-sub-linear randomness complexity. This can be based on sub-exponentially secure pseudorandom generators of stretch $\frac{1}{\varepsilon'}$, where constant $\varepsilon' > 1$ is related to ε.*
3. *Sub-exponentially secure learning with errors.*

Instantiation: We show how to leverage PAFE for degree-5 arithmetic circuits to achieve $\mathsf{i}\mathcal{O}$, thus yielding the first $\mathsf{i}\mathcal{O}$ construction from degree-5 multilinear maps. The crucial step in this transformation is to first construct homogenous randomizing polynomials with sub-linear randomness complexity of degree 15. We first identify that the work of [AIK06] satisfies the required properties of a degree-3 homogenous randomizing polynomials scheme. To achieve sub-linear randomness complexity, we assume an explicit degree-2 pseudo-random generator (PRGs) achieving super-linear stretch in the boolean setting, and a related explicit degree-3 PRG achieving super-quadratic stretch in the arithmetic setting. In particular we use a boolean PRG of stretch 1.49 and an algebraic PRG of stretch 2.49 [OW14] (see also [AL16]). We then observe that for a special class of circuits $\mathcal{C}$, the degree of the above polynomials can be reduced to 5 if we additionally allow for pre-processing of randomness. Also, we show how to remove the algebraic PRG part in the construction of randomizing polynomials for $\mathcal{C}$.

As alluded to above, the fact that our PAFE can directly deal with an arithmetic PRG in a degree-preserving manner is critical to allowing us to achieve $\mathsf{i}\mathcal{O}$ with just degree-5 mutlilinear maps.

Theorem 3 (Informal)**.** *We construct an indistinguishability obfuscation scheme for P/poly assuming the following: for some constant d,*

1. *Sub-exponentially secure PAFE scheme for degree 5 arithmetic circuits with multiplicative overhead in encryption complexity. From Theorem 1, this can be based on sub-exponentially secure Assumptions #1 and #2 (Section 7.2.1).*
2. *Sub-exponentially secure degree 5 homogenous randomizing polynomials for $\mathcal{C}$ with ε-sub-linear randomness complexity. This can be based on sub-exponentially secure boolean PRG of stretch 1.01 (see Section 4.4).*
3. *Sub-exponentially secure learning with errors.*

Concurrent Work. In a concurrent work[7], Lin obtains a new IO construction with a security reduction to 1) L-linear maps with the subexponential symmetric external Diffie-Hellman

[7]We have engaged in several amicable exchanges with Lin about our respective results over the few weeks preceding

(SXDH) assumption, 2) subexponentially secure locality-L PRG, and 3) subexponential LWE. When using a locality 5 PRG, 5-linear maps with the SXDH assumption suffice. The L-linear maps consist of L source groups $G_1, \cdots, G_L$, whose elements $g_1^{a_1}, \cdots, g_L^{a_L}$ can be "paired" together to yield an element in a target group $g_T^{a_1 \cdots a_L}$. The SXDH assumption on such multilinear maps is a natural generalization of the SXDH assumption on bilinear maps: It postulates that the DDH assumption holds in every source group G_d, that is, elements g_d^a, g_d^b, g_d^{ab} are indistinguishable from g_d^a, g_d^b, g_d^r, for random a, b and r.

To obtain IO, she first constructs collusion-resistant FE schemes for computing degree-L polynomials from L-linear maps, and then bootstraps such FE schemes to IO for P, assuming subexponentially secure locality-L PRG and LWE.

We now give a technical overview of our approach.

1.2 Technical Overview

We give an informal description of the algorithms of projective arithmetic functional encryption (PAFE). We focus on secret-key setting in this work.

- **Setup**: It outputs secret key MSK.
- **Key Generation**: On input an arithmetic circuit C and master secret key, it produces a functional key sk_C.
- **Encryption**: On input message x, it outputs a ciphertext CT.
- **Projective Decryption**: On input a functional key sk_C and ciphertext CT, it produces a partial decrypted value ι.
- **Recover**: On input many partial decrypted values $\{\iota_i\}$ and a linear function (specified as co-efficients), it outputs the result of applying the linear function on the values contained in $\{\iota_i\}$.

We first show how to achieve iO starting from secret-key PAFE. Later, we show how to obtain PAFE for degree **D** polynomials starting from degree **D** multilinear maps.

iO from Secret-Key PAFE: We start with the goal of constructing a sub-linear secret-key FE scheme for NC^1 starting from PAFE for constant degree arithmetic circuits. Our goal is to minimize the degree of arithmetic circuits that suffices us to achieve sub-linear FE.

We start with the standard tool of randomizing polynomials to implement NC^1 using a constant degree arithmetic circuit. We use randomizing polynomials with a special decoder: the decoder is a sequence of linear functions chosen adaptively [8]. At a high level the construction proceeds as follows: let the randomizing polynomial of circuit C, input x and randomness r be of the form $p_1(x; r), \ldots, p_N(x; r)$. The sub-linear FE functional key corresponding to a circuit C are a collection of PAFE keys for $p_1, \ldots, p_N$. The encryption of x w.r.t sublinear FE scheme is a PAFE encryption of (x, r). To obtain $C(x)$, first execute the projective decryption algorithm on key of p_i and ciphertext of (x, r) to obtain partial decrypted values corresponding to $p_i(x, r)$.

the initial posting of our papers. During this time, with regard to the minimum level of multilinearity needed for constructing iO, certain milestones were reached at different times. In particular, our group had the first paper claiming iO from degree-15 maps. After that, Lin had the first paper claiming iO from degree-5 maps, and thereafter our group was able to modify our paper to claim iO from degree-5 maps (this work). Both groups independently relied on PRGs of locality 5 to achieve these results, and the common bottleneck at degree 5 reflects this. Prior to posting, however, the groups did not exchange any manuscripts, and worked independently. There are several other differences in our results, most notably with regard to the assumptions. In addition, the techniques are substantially different.

[8] That is, choice of every linear function could depend on the output of the previously chosen linear functions on the encoding of computation.

Now, execute the recover algorithm on input a linear function and the above partial decrypted values, where the linear function is chosen by the decoder of the randomizing polynomials scheme. Depending on the output of the recover algorithm, the decoder picks a new linear function. This process is repeated until we finally recover the output of the circuit C.

Before we justify why this scheme is secure, we remark as to why this scheme satisfies the sub-linear efficiency property. In order to achieve sub-linear efficiency, we require that $|r| = |C|^{1-\varepsilon}$ for some $\varepsilon > 0$. Thus, we require randomizing polynomials with sub-linear randomness complexity. We remark later how to achieve this.

The next goal is to argue security: prior works either employ function privacy properties [BS15] or Trojan techniques [CIJ+13, ABSV15] to make the above approach work. However, going through these routes is going to increase the degree of arithmetic circuits required to achieve sub-linear FE. Instead, we start with a PAFE scheme with a stronger security guarantee called *semi-functional* security. This notion is inspired by the dual system methodology introduced by Waters [Wat09] in different context and later employed by several other works (see for example, [LOS+10, GGHZ14]). Associated with this notion, there are two types of objects:

- *Semi-Functional Keys:* A semi-functional key is associated with an arithmetic circuit C and a hardwired value v.
- *Semi-Functional Ciphertexts:* A semi-functional ciphertext is generated just using the master secret key.

We define how honestly generated keys, honestly generated ciphertexts and semi-functional keys, semi-functional ciphertexts are required to behave with each other in Table 1. Honestly generated key or ciphertext refers to generation of key or ciphertext according to the description of the scheme.

	Honestly Generated Keys	Semi-Functional Keys
Honestly Generated Ciphertexts	Honest decryption	Honest decryption
Semi-Functional Ciphertexts	Not Defined	Output Hardwired Value

Table 1: We consider four possibilities of decryption: (a) honestly generated keys correctly decrypts honestly generated ciphertexts (from correctness property), (b) semi-functional keys also correctly decrypts honestly generated ciphertexts, (c) there is no correctness guarantee on the decryption of honestly generated keys on semi-functional ciphertexts, (d) Finally, the decryption of semi-functional keys on semi-functional ciphertexts yields the hardwired value associated with the key.

A PAFE scheme is said to satisfy semi-functional security if both the following definitions are satisfied:

- *Indistinguishability of Semi-functional keys*: It should be hard to distinguish an honestly generated functional key of C from a semi-functional key of C associated with any hardwired value v.
- *Indistinguishability of Semi-functional Ciphertexts*: It should be hard to distinguish an honestly generated ciphertext of x from a semi-functional ciphertext if every functional key of C issued is a semi-functional key associated with hardwired value $C(x)$.

Once we have a secret key PAFE scheme that satisfies semi-functional security then we can prove the security as follows: we consider a simple case when the adversary only submits one message query (x_0, x_1).

- We first turn the functional key associated with an arithmetic circuit C into a semi-functional key with the hardwired value $C(x_0)$.
- Once all the functional keys are semi-functional, we can now switch the ciphertext of x_0 to semi-functional ciphertext.
- Since $C(x_0) = C(x_1)$, we can switch back the semi-functional keys to be honestly generated functional keys.
- Finally, we switch back the ciphertext from semi-functional to honestly generated ciphertext of x_1.

If the adversary requests multiple message queries, then the above process is to be repeated one message query at a time.

Choice of Randomizing Polynomials with Sub-linear Randomness: The next question is what randomizing polynomials do we choose to instantiate the above approach. As we will see later, if we choose randomizing polynomials with sub-linear randomness complexity of degree $\mathbf{D}$ then it suffices build PAFE from degree $\mathbf{D}$ multilinear maps. Also, we will require the polynomials to be homogenous.

Hence, our goal is to choose a homogenous randomizing polynomials with minimal degree and also satisfying (i) linear decodability and (ii) sub-linear randomness complexity properties. We achieve this in the following steps:

1. First, build randomizing polynomials with minimal degree. We start with [AIK06] for NC^1, where the polynomials are of degree 3. In spirit, this is essentially information theoretic Yao with the wire keys being elements over $\mathbb{F}_\mathbf{p}$ and every wire key is associated with a random mask (which is represented as a bit) that helps in figuring out which of the four entries to be decoded for the next gate.
2. The above scheme already satisfies linear decodability property. This is because the decryption of every garbled gate is a linear operation. The linear function chosen to decrypt one garbled gate now depends on the linear functions chosen to decrypt its children gates.
3. Next, we tackle sub-linear randomness complexity: we generate the wire keys and the random masks as the output of a PRG. The total length of all the wire keys is roughly square the size of the NC^1 circuit. This is because, the size of the wire keys at the bottom most (input) layer are proportional to the size of the circuit. We use an algebraic PRG of stretch $(2+\varepsilon)$ to generate the wire keys and we use a boolean PRG to generate the random masks. The degree of the algebraic PRG over $\mathbb{F}_\mathbf{p}$ is 3 while the degree of the boolean PRG represented over $\mathbb{F}_\mathbf{p}$ is 5. When the above PRGs are plugged into the randomizing polynomials construction from the above step, we get the degree of the polynomials to be 15.
4. Finally, we show how to make the above randomizing polynomials homogenous. This is done using a standard homogenization argument: add dummy variables to the polynomials such that the degree of all the terms in the polynomials are the same. While evaluating these polynomials, set all these dummy variables to 1. This retains the functionality and at the same time ensures homogeneity.

We can now use the above randomizing polynomials scheme to instantiate the above approach. After partial decryption, we get partial decrypted values associated with $\{p_i(x;r)\}$. Now, since the decoding is composed of many linear functions, we can execute the Recover algorithm (multiple times) to recover the output.

Reducing the Degree: We can apply some preprocessing to reduce the degree of the above polynomials further. We remark how to reduce the degree to 5. Later, in the technical sections, we explore alternate ways of reducing the degree, as well.

Suppose we intend to construct sublinear FE for a specific class of circuits $\mathcal{C}$. In this case, we are required to construct randomizing polynomials only for $\mathcal{C} \in NC^1$.

We define $\mathcal{C}$ as follows: every circuit $C \in \mathcal{C}$ of output length $\mathbf{N}$ is of the form $C = (C_1, \ldots, C_{\mathbf{N}})$, where (i) C_i outputs the i^{th} output bit of C, (ii) $|C_i| = \text{poly}(\lambda)$ for a fixed polynomial poly, (iii) Depth of C_i is $c \cdot \log(\lambda)$, where c is a constant independent of $|C|$ and, (iv) C_i for every $i \in [\mathbf{N}]$ has the same topology – what is different, however, are the constants associated with the wires. We show later that it suffices to build sublinear FE for $\mathcal{C}$ to obtain iO. We now focus on obtain randomizing polynomials for $\mathcal{C}$.

We start with the randomizing polynomials scheme that we described above. Recall that it involved generating a garbled table for every gate in the circuit C. Moreover, the randomness to generate this garbled table is derived from an algebraic and a boolean PRG. We make the following useful changes: let $C = (C_1, \ldots, C_{\mathbf{N}})$ such that C_i outputs the i^{th} output bit of C. Let $w_1^i, \ldots, w_{\mathsf{nw}}^i$ be the set of wires in C_i and $G_1^i, \ldots, G_{\mathsf{ng}}^i$ be the set of gates in C_i.

- We invoke nw number of instantiations of boolean PRGs $\mathsf{bPRG}_1^w, \ldots, \mathsf{bPRG}_{\mathsf{nw}}^w$ and $\mathsf{bPRG}_1^r, \ldots, \mathsf{bPRG}_{\mathsf{nw}}^r$. All these PRGs have the same structure (i.e., same predicates is used) and have degree 5 over arbitrary field (with slightly superlinear stretch $1+\varepsilon$). Pseudorandom generator bPRG_j^w is used to generate wire keys for wires $w_j^1, \ldots, w_j^{\mathbf{N}}$. Recall that earlier we were using an algebraic PRG of quadratic stretch. This is because the size of wire keys was proportional to exponential in depth, which could potentially be linear in the size of the circuit. However, since we are considering the specific circuit class $\mathcal{C}$, the depth of every circuit is $c \log(\lambda)$. And thus the size of the wire keys is independent of the security parameter. This is turn allows us to use just a PRG of superlinear stretch $1+\varepsilon$. Finally, bPRG_j^r is used to generate random masks for the wires $w_j^1, \ldots, w_j^{\mathbf{N}}$.

- We now consider the [AIK06] randomizing polynomials associated with circuit C. As before, we substitute the variables associated with wire keys and random masks with the polynomials associated with the appropriate PRGs. The formal variables in the PRG polynomials are associated with the seed.

- The result of the above process is the encoding of C consisting of polynomials $p_1, \ldots, p_N$ with variables associated with the seeds of PRGs. Note that the degree of these polynomials is still 15.

- We then observe that there are polynomials $q_1, \ldots, q_T$ in seed variables such that $p_1, \ldots, p_N$ can be rewritten in terms of $q_1, \ldots, q_T$ and moreover, the degree of p_i in the new variables $\{q_i\}$ is 5. The advantage of doing this is that the polynomials $\{q_i\}$ can be evaluated during the encryption phase[9]. The only thing we need to be wary of is the fact that T could be as big as $|C|$. If this is the case then the encryption complexity would be at least linear in $|C|$, which violate the sublinearity of the FE scheme. We show how to carefully pick $q_1, \ldots, q_T$ such that T is sub-linear in $|C|$ and the above properties hold. We refer the reader to the technical sections for more details.

The only missing piece here is to show that sublinear FE for this special class of circuits $\mathcal{C}$ with sub-exponential security loss implies iO. To show this, it suffices to show that sublinear FE for $\mathcal{C}$ implies sublinear FE for all circuits. Consider the transformation from FE for NC^1 to FE for all circuits by [ABSV15] – the same transformation also works for single-key sublinear secret key FE. We consider a variant of their transformation. In this transformation, a sublinear FE

[9]This idea is similar in spirit to the recent work of Bitansky et al. [BLP16], who introduced degree reduction techniques in a different context.

key for circuit C' is generated by constructing a circuit C that has hardwired into it C' and value v. Circuit C takes as input x, PRF key K and mode b. If $b = 0$ it outputs a Yao's garbled circuit of (C, x) computed w.r.t randomness derived from K. If $b = 1$ it outputs the value v. We can re-write C as being composed of sub-circuits $C_1, \ldots, C_{\mathbf{N}}$ such that each of C_i is in NC^1, $|C_i| = \text{poly}(\lambda)$ and depth of C_i is $c \cdot \log(\lambda)$ for a fixed polynomial poly and fixed constant c. Intuitively, C_i, has hardwired into it gate G_i of C' and i^{th} block of v. It computes a garbled table corresponding to G_i if $b = 0$, otherwise it outputs the i^{th} block of v.

Constructing PAFE: We now focus on building PAFE from multilinear maps. The first attempt to encrypt the input $x = (x_1, \ldots, x_{\ell_{\mathsf{inp}}})$ would be to just encode every x_i separately. Now, during evaluation of circuits $C_1, \ldots, C_N$ on these encodings will yield a top level encoding of $C_i(x)$. This homomorphic evaluation would correspond to projective decryption operation. The recover algorithm would just compute a linear function on all the top level encodings of $C_i(x)$ and using zero test parameters, recover the answer if the output of the linear function is 0.

However, we cannot allow the adversary to evaluate recover outputs for circuits C_i of his choice. We should ensure that he recovers outputs only for circuits corresponding to which he has been issued functional keys. The main challenge in designing a functional key for C is to guarantee authenticity – how do we ensure that if the adversary, given a functional key corresponding to C, can only evaluate C on these inputs? To ensure this, we introduce a parallel branch of computation: we instead encode (x_i, α_i) where $\{\alpha_i\}$ are random elements determined during the setup. Then as part of the functional key associated with C, we give out an encoding of $C(\{\alpha_i\})$ at the top level that will allow us to cancel the α_i part after computing C on encodings of $\{(x_i, \alpha_i)\}$ and in the end, just get an encoding of $C(x)$. However, to implement this, we need to make sure that the computation of C on $\{x_i\}$ and $\{\alpha_i\}$ are done separately even though x_i and α_i are encoded together.

The work of [Zim15, AB15] used the above idea in the context of designing iO. As we will discuss below, we extend their techniques in several ways, to deal with the problem of mixing ciphertext components and achieving the semi-functional security properties we need from our PAFE scheme. However, before we discuss these difficulties, we note that the work of [Zim15, AB15] implement parallel branches by using composite order multilinear maps. Composite order multilinear maps allow for jointly encoding for a vector of elements such that addition and multiplication operations can be homomorphically performed on every component of the vector separately.

However, one of the primary motivations for this line of work on building constructions for i$\mathcal{O}$ from low-degree multilinear maps is to enable the use of future candidate low-degree multilinear maps, where achieving composite order may not be possible. Indeed, current instantiations of composite order multlinear maps [CLT13] have poorly understood security properties, and have been subject to efficient cryptanalytic attacks in some settings (see, e.g., [CHL+15, CGH+15]). Thus, instead of relying on composite order multilinear maps, we do the following: we introduce a primitive called a slotted encoding scheme, that allows for the same functionality as offered by composite order multilinear maps. This then helps us in implementing the idea of [Zim15, AB15] using a slotted encoding scheme. We later show how to realize a constant degree slotted encoding scheme using prime order multilinear maps. We define slotted encodings next.

Slotted Encoding: A slotted encoding scheme, parameterized by L (number of slots), has the following algorithms: (i) Setup: this generates the secret parameters, (ii) Encode: it takes as input $(a_1, \ldots, a_L)$ and outputs an encoding of it, (iii) Arithmetic operations: it takes two encodings of $(a_1, \ldots, a_L)$ and $(b_1, \ldots, b_L)$ and performs arithmetic operations on every component separately. For instance, addition of encoding of $(a_1, \ldots, a_L)$ and $(b_1, \ldots, b_L)$ would lead to encoding of $(a_1 + b_1, \ldots, a_L + b_L)$, (iv) Zero Testing: It outputs success if the encoding of

$(a_1, \ldots, a_L)$ is such that $a_i = 0$ for every i.

In this work, we will be interested in asymmetric slotted encodings, where the slotted encodings is associated with a tree T such that every encoding is associated with a node in T and two encodings can be paired only if their associated nodes are siblings. The degree of slotted encodings is defined to be the maximum degree of polynomials the scheme lets us evaluate.

Constant Degree Slotted Encoding From Prime Order MMaps: We start with the simple case when degree of slotted encodings is 2 (the bilinear case). The idea of dual vector spaces were introduced by [OT08] and further developed as relevant to us by [OT09, BJK15] to address this problem for bilinear maps. In this framework, there is an algorithm that generates $2n$ vectors $(\mu_1, \ldots, \mu_n), (\nu_1, \ldots, \nu_n)$ of dimension n such that: (i) inner product, $\langle \mu_i, \nu_i \rangle = 1$ and, (ii) inner product, $\langle \mu_i, \nu_j \rangle = 0$ when $i \neq j$. Using this, we can encode $(a_1, \ldots, a_n)$ associated with some node u in the tree as follows: encode every element of the vector $a_1\mu_1 + \cdots + a_n\mu_n$. The encoding of $(b_1, \ldots, b_n)$ associated with a node v, which is a sibling of u, will be encodings of the vector $b_1\nu_1 + \cdots + b_n\nu_n$. Now, computing inner product of both these encodings will lead to an encoding of $a_1 \cdot b_1 + \cdots + a_n \cdot b_n$.

This idea doesn't suffice for degree 3. So our idea is to work modularly, and consider multiple layers of vectors. The encoding of $(a_1, \ldots, a_n)$ under node u will be encodings of the vector $(a_1\mu_1 \otimes \mu'_1 + \cdots + a_n\mu_n \otimes \mu'_n)$[10], where $\{\mu'_i\}$ is a basis of a vector space associated with the parent of node u. Now, when this is combined with encoding of $b_1\nu_1 + \cdots + b_n\nu_n$, computed under node v, we get encoding of $(a_1b_1\mu'_1 + \cdots a_nb_n\mu'_n)$. Using this we can then continue for one more level.

To generalize this for higher degrees we require tensoring of multiple vectors (potentially as many as the depth of the tree). This means that the size of the encodings at the lower levels is exponential in the depth and thus, we can only handle constant depth trees. Implementing our tensoring idea for multiple levels is fairly technical, and we refer the reader to the relevant technical section for more details.

PAFE from Slotted Encodings: Using slotted encodings, we make a next attempt in constructing PAFE:

- To encrypt $x = (x_1, \ldots, x_{\ell_{\mathsf{inp}}})$, we compute a slotted encoding of (x_i, α_i), where α_i are sampled uniformly at random during the setup phase.
- A functional key of C consists of a slotted encoding of $(0, C(\{\alpha_i\}))$ at the top level.

The partial decryption first homomorphically evaluates C on slotted encodings of (x_i, α_i) to get a slotted encoding of $(C(\{x_i\}), C(\{\alpha_i\}))$. The second slot can be 'canceled' using top level encoding of $(0, C(\{\alpha_i\}))$ to get an encoding of $(C(\{x_i\}), 0)$. The hope is that if the evaluator uses a different circuit C' then the second slot will not get canceled and hence, he would be unable to get a zero encoding.

However, choosing a different C' is not the only thing an adversary can do. He could also mix encodings from different ciphertexts and try to compute C on it – the above approach does not prevent such attacks. In order to handle this, we need to ensure that the evaluation of ciphertexts can never be mixed. In order to solve this problem, we use a mask γ that be independently sampled for every ciphertext. Every encoding will now be associated with this mask. Implementing this idea will crucially make use of the fact that the polynomial computed by the arithmetic circuit is a homogenous polynomial.

Yet another problem arises is in the security proof: for example, to design semi-functional keys, we need to hardwire a value in the functional key. In order to enable this, we introduce a third slot. With this new modification, we put forward a template of our construction. Our actual construction involves more details which we skip to keep this section informal.

[10]Here, $\mu_i \otimes \mu_j$ denotes the tensoring of μ_i and μ_j.

- To encrypt $x = (x_1, \ldots, x_{\ell_{\mathsf{inp}}})$, we compute a slotted encoding of $(x_i, \alpha_i, 0)$, where α_i are sampled uniformly at random during the setup phase. Additionally, you give out encoding of $(0, S, 0)$ at one level lower than the top level, where S is also picked at random in the setup phase.
- A functional key of C consists of a slotted encoding of $(0, C(\{\alpha_i\}) \cdot S^{-1}, 0)$ at the top level.

The decryption proceeds as before, except that the encodings of $(0, C(\{\alpha_i\}) \cdot S^{-1}, 0)$ and $(0, S, 0)$ are paired together before we proceed.

Note that in both the ciphertext and the functional key, the third slot is not used at all. The third slot helps in the security proof. To see how we describe the semi-functional parameters at a high level as follows:

- *Semi-functional Ciphertexts*: To encrypt $x = (x_1, \ldots, x_{\ell_{\mathsf{inp}}})$, we compute a slotted encoding of $(0, \alpha_i, 0)$, where α_i is computed as before. Additionally, you give out encoding of $(0, S, 1)$ at one level lower than the top level, where S is also picked at random in the setup phase. Note that the third slot now contains 1 which signals that it is activated.
- *Semi-functional Keys*: A functional key of C consists of a slotted encoding of $(0, C(\{\alpha_i\}), v)$ at the one level lower than top level, where v is the hardwired value associated with the semi-functional key.

During the decryption of semi-functional key with honestly generated ciphertext, the third slot will not be used since it will be deactivated in the ciphertext. So the decryption proceeds normally. However, during the decryption of semi-functional key with semi-functional ciphertexts, the third slot is used since the third slot is activated in the ciphertext. We argue the security of our construction in the ideal multilinear map model.

Comparison With [LV16]. We now compare our work with the recent exciting work of [LV16], in order to illustrate some differences that allow us to achieve lower degree. The work of [LV16] first defines FE for NC^0 with a non-trivial efficiency property and give a new bootstrapping theorem[11] to achieve compact FE. They then show how to achieve FE for NC^0 from constant degree multilinear maps[12]. Interestingly, they use arithmetic randomizing polynomials within their construction of FE for NC^0 – this will be important as we note below.

In contrast, we do not build FE for NC^0, but rather show how to proceed directly from projective arithmetic FE for degree-5 arithmetic circuits to iO (without additional use of multilinear maps). Furthermore, our construction of PAFE is *degree preserving*, so to achieve PAFE for degree-5 arithmetic circuits, we only need degree-5 multilinear maps. In contrast, in [LV16], to build FE for NC^0, their work has to "pay" in degree not only based on the depth of the NC^0 circuit that underlies each secret key, but also for the arithmetic randomizing polynomial that they apply to the NC^0 circuit. This adds a significant overhead in the constant degree their multilinear map must support. Our approach is simpler, as our randomizing polynomials are only used in the path from PAFE to iO, which does not use multilinear maps in any additional way. There are, of course, many other technical differences between our work and [LV16], as well. Another conceptual idea that we introduce, and that is different from [LV16], is the notion of slotted encodings, an abstraction of composite order multilinear maps, and our method for emulating slotted encodings using prime order multilinear maps without increasing the degree.

[11]Their bootstrapping theorem also works if we start with FE for constant degree polynomials over $\mathbb{F}_2$.

[12]Note that, in particular, the security of their scheme reduces to a succinct assumption called the multilinear joint SXDH assumption. As we noted earlier, unfortunately this assumption is not known to be instantiable with existing multilinear map candidates. However, one can posit a different assumption that directly assumes their FE for NC^0 scheme to be secure, and we do not know of any attacks on that (non-succinct) assumption.

1.3 Organization

1. We start with preliminaries in Section 2.
2. The concept of projective arithmetic functional encryption is defined in Section 3.
3. We then show how to construct iO starting from PAFE. To do that, we construct sublinear secret key FE. This is done in two steps:
   - First, we introduce the notion of (homogenous) (T, Φ)-randomizing polynomials in Section 4. This will be a tool used in the construction of iO.
   - Once this is done, we construct sublinear secret key FE from PAFE and homogenous (T, Φ)-randomizing polynomials in Section 5.
4. Finally, we demonstrate how to construct PAFE from multilinear maps. This is accomplished in the following steps:
   - We first define an abstraction of composite order multilinear maps, that we call slotted encodings. This is done in Section 6. In the same section, we show how to implement slotted encodings from prime order multilinear maps.
   - We construct PAFE from slotted encodings in Section 7. The security of PAFE is based on concrete assumptions on slotted encodings.
   - Finally in Section 8, we show how to justify the assumptions on slotted encodings in the ideal multilinear map model.

2 Preliminaries

We denote the security parameter by λ. Two distributions $\mathcal{D}_1$ and $\mathcal{D}_2$, defined on the same sample space, are said to be computationally indistinguishable, denoted by $\mathcal{D}_1 \cong_c \mathcal{D}_2$ if the following holds: for any PPT adversary $\mathcal{A}$ and sufficiently large security parameter $\lambda \in \mathbb{N}$,

$$\left|\mathsf{Pr}[1 \leftarrow \mathcal{A}(1^\lambda, s_1) \ : \ s_1 \xleftarrow{\$} \mathcal{D}_1] - \mathsf{Pr}[1 \leftarrow \mathcal{A}(1^\lambda, s_2) \ : \ s_2 \xleftarrow{\$} \mathcal{D}_2(1^\lambda)]\right| \leq \mathsf{negl}(\lambda),$$

where negl is some negligible function.

2.1 Indistinguishability Obfuscation (iO)

The notion of indistinguishability obfuscation (iO), first conceived by Barak et al. [BGI+01], guarantees that the obfuscation of two circuits are computationally indistinguishable as long as they both are equivalent circuits, i.e., the output of both the circuits are the same on every input. Formally,

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

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

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

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

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

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

2.2 Secret-Key Functional Encryption

A secret-key functional encryption (FE) scheme FE over a message space $\mathcal{X} = \{\mathcal{X}_\lambda\}_{\lambda \in \mathbb{N}}$ and a function space $\mathcal{C} = \{\mathcal{C}_\lambda\}_{\lambda \in \mathbb{N}}$ is a tuple $(\mathsf{FE.Setup}, \mathsf{FE.KeyGen}, \mathsf{FE.Enc}, \mathsf{FE.Dec})$ of PPT algorithms with the following properties:

- $\mathsf{FE.Setup}(1^\lambda)$: The setup algorithm takes as input the unary representation of the security parameter, and outputs a secret key $\mathsf{FE.MSK}$.
- $\mathsf{FE.KeyGen}(\mathsf{FE.MSK}, f)$: The key-generation algorithm takes as input the secret key $\mathsf{FE.MSK}$ and a function $f \in \mathcal{C}_\lambda$, and outputs a functional key $\mathsf{FE.SK}_f$.
- $\mathsf{FE.Enc}(\mathsf{FE.MSK}, x)$: The encryption algorithm takes as input the secret key $\mathsf{FE.MSK}$ and a message $x \in \mathcal{X}_\lambda$, and outputs a ciphertext CT.
- $\mathsf{FE.Dec}(\mathsf{FE.SK}_f, \mathsf{CT})$: The decryption algorithm takes as input a functional key $\mathsf{FE.SK}_f$ and a ciphertext CT, and outputs out.

In terms of correctness, we require that there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all sufficiently large $\lambda \in \mathbb{N}$, for every message $x \in \mathcal{X}_\lambda$, and for every function $f \in \mathcal{C}_\lambda$ it holds that

$$\mathsf{FE.Dec}\left(\mathsf{FE.KeyGen}(\mathsf{FE.MSK},\ f),\ \mathsf{FE.Enc}(\mathsf{FE.MSK},\ x)\right) = f(x)$$

with probability at least $1 - \mathsf{negl}(\lambda)$, where $\mathsf{FE.MSK} \leftarrow \mathsf{FE.Setup}(1^\lambda)$, and the probability is taken over the random choices of all algorithms.

Definition 2 (Security). *A secret-key functional encryption scheme $\Pi = (\mathsf{FE.Setup}, \mathsf{FE.KeyGen}, \mathsf{FE.Enc}, \mathsf{FE.Dec})$ over a function space $\mathcal{C} = \{\mathcal{C}_\lambda\}_{\lambda \in \mathbb{N}}$ and a message space $\mathcal{X} = \{\mathcal{X}_\lambda\}_{\lambda \in \mathbb{N}}$ is* secure *if for any PPT adversary $\mathcal{A}$ there exists a negligible function $\mathsf{negl}(\cdot)$ such that*

$$\mathsf{Advtge}^{\mathsf{FE}}_{\Pi,\mathcal{A}}(\lambda) = \left|\Pr[\mathsf{Expt}^{\mathsf{FE}}_{\Pi,\mathcal{A}}(\lambda, 0) = 1] - \Pr[\mathsf{Expt}^{\mathsf{FE}}_{\Pi,\mathcal{A}}(\lambda, 1) = 1]\right| \leq \mathsf{negl}(\lambda),$$

for all sufficiently large $\lambda \in \mathbb{N}$, where for each $b \in \{0, 1\}$ and $\lambda \in \mathbb{N}$ the experiment $\mathsf{Expt}^{\mathsf{FE}}_{\Pi,\mathcal{A}}(1^\lambda, b)$, modeled as a game between the adversary $\mathcal{A}$ and a challenger, is defined as follows:

1. **Setup phase:** *The challenger samples $\mathsf{FE.MSK} \leftarrow \mathsf{FE.Setup}(1^\lambda)$.*
2. **Message queries:** *On input 1^λ the adversary submits $((x_1^{(0)}, \ldots, x_{\ell_\mathbf{x}}^{(0)}), (x_1^{(1)}, \ldots, x_{\ell_\mathbf{x}}^{(1)}))$ for some polynomial $\ell_\mathbf{x} = \ell_\mathbf{x}(\lambda)$. The challenger replies with $(\mathsf{CT}_1, \ldots, \mathsf{CT}_{\ell_\mathbf{x}})$, where $\mathsf{CT}_i \leftarrow \mathsf{FE.Enc}(\mathsf{FE.MSK}, x_i^{(b)})$ for every $i \in [\ell_\mathbf{x}]$.*
3. **Function queries:** *The adversary queries the challenger with any function $f \in \mathcal{F}_\lambda$ such that $f(x_i^{(0)}) = f(x_i^{(1)})$ for every $i \in [\ell_\mathbf{x}]$. The adversary makes $\ell_\mathbf{f}$ such queries. For each such query, the challenger replies with $\mathsf{FE}.sk_f \leftarrow \mathsf{FE.KeyGen}(\mathsf{FE.MSK}, f)$.*
4. **Output phase:** *The adversary outputs a bit b' which is defined as the output of the experiment.*

t-overhead. An FE scheme is said to have t-overhead if the size of a functional key, associated with function f, has size at most $(|f| \cdot \lambda)^t$. More formally,

Definition 3. *Consider a constant $t \in \mathbb{N}$. A FE scheme $\mathsf{FE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{ProjectDec})$ for a class of functions $\mathcal{C}$ is said to have* t-**overhead** *if $|sk_f| \leq (|f| \cdot \lambda)^t$ for every $f \in \mathcal{C}$, where (i) $\mathsf{MSK} \leftarrow \mathsf{Setup}(1^\lambda)$ and, (ii) $sk_f \leftarrow \mathsf{KeyGen}(\mathsf{MSK}, f)$.*

Sublinear Secret Key FE. A secret key functional encryption scheme is said to be sublinear if the encryption complexity is sublinear in the complexity of the function family. More formally,

Definition 4 ([BV15, AJ15]). *Let p be a polynomial. A secret key FE scheme* $\mathsf{FE} = (\mathsf{FE.Setup}, \mathsf{FE.KeyGen}, \mathsf{FE.Enc}, \mathsf{FE.Dec})$ *for $\mathcal{C}$ is said to be* sublinear *if the running time of* $\mathsf{FE.Enc}(\mathsf{FE.MSK}, m)$ *is* $(\ell_f)^{1-\epsilon} \cdot p(\lambda, |m|)$ *, where* $\mathsf{FE.MSK} \leftarrow \mathsf{FE.Setup}(1^\lambda)$ *and* $\ell_f = \max_{C \in \mathcal{C}_{|m|}} \{C\}$ *with $\mathcal{C}_{|m|}$ consisting of all circuits in $\mathcal{C}$ with inputs of length $|m|$.*

2.3 Exponentially-Efficient iO (XiO)

We recall the definition of XiO introduced by the work of Lin, Pass, Seth and Telang [LPST16].

Exponentially-Efficient iO (XiO). XiO is an indistinguishability obfuscation with the weaker efficiency requirement that dictates that the size of the obfuscated circuit should be sublinear in the size of the truth table associated with the circuit.

Definition 5. *(Exponentially-Efficient iO (XiO)) For a constant $\gamma < 1$, a machine* XiO *is a γ-compressing exponentially-efficient indistinguishability obfuscator (XiO) for a circuit class $\{\mathcal{C}_\lambda\}_{\lambda \in \mathbb{N}}$ if it satisfies the functionality and indistinguishability in Definition 1* and the following *efficiency requirements:*

- **Non-trivial efficiency***: For every $\lambda \in \mathbb{N}$, every $C \in \mathcal{C}_\lambda$, we have that $|\mathsf{i}\mathcal{O}(\lambda, C)| \leq 2^{n\gamma}\text{poly}(\lambda, C)$, where n is the input length of C.*

3 Projective Arithmetic Functional Encryption

In this section, we introduce the notion of projective arithmetic functional encryption scheme. There are two main differences from a (standard) functional encryption scheme:

- Functional keys are associated with arithmetic circuits.
- The projective decryption algorithm only outputs partial decrypted values. There is a recover algorithm that computes on the partial decrypted values and produces an output.

3.1 Definition

We can consider either a public key projective arithmetic FE scheme or a secret key projective arithmetic secret key FE scheme. In this work, we define and construct a secret key projective arithmetic FE scheme.

A secret-key projective arithmetic functional encryption (FE) scheme PAFE over field $\mathbb{F}_{\mathbf{p}}$ is associated with a message space $\mathcal{X} = \{\mathcal{X}_\lambda\}_{\lambda \in \mathbb{N}}$ and a arithmetic circuit class $\mathcal{C} = \{\mathcal{C}_\lambda\}_{\lambda \in \mathbb{N}}$ over $\mathbb{F}_{\mathbf{p}}$. Here, $\mathcal{X}$ comprises of strings with every symbol in the string belongs to $\mathbb{F}_{\mathbf{p}}$.

PAFE comprises of a tuple $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{ProjectDec})$ of PPT algorithms with the following properties:

- $\mathsf{Setup}(1^\lambda)$: The setup algorithm takes as input the unary representation of the security parameter, and outputs a secret key MSK.
- $\mathsf{KeyGen}(\mathsf{MSK}, C)$: The key-generation algorithm takes as input the secret key MSK and a arithmetic circuit $C \in \mathcal{C}_\lambda$, over $\mathbb{F}_{\mathbf{p}}$, and outputs a functional key sk_C.
- $\mathsf{Enc}(\mathsf{MSK}, x)$: The encryption algorithm takes as input the secret key MSK and a message $x \in \mathcal{X}_\lambda$, and outputs a ciphertext CT.

- $\mathsf{ProjectDec}(sk_C, \mathsf{CT})$: The projective decryption algorithm takes as input a functional key sk_C and a ciphertext CT, and outputs a partial decrypted value ι.
- $\mathsf{Recover}(c_1, \iota_1, \ldots, c_{\ell_\mathbf{f}}, \iota_{\ell_\mathbf{f}})$: The recover algorithm takes as input co-efficients $c_1, \ldots, c_{\ell_\mathbf{f}} \in \mathbb{F}_\mathbf{p}$, partial decrypted values $\iota_1, \ldots, \iota_{\ell_\mathbf{f}}$ and outputs out.

We first define the correctness property and later, define the security property.

B-Correctness. The correctness is parameterized by a set $B \subseteq \mathbb{F}_\mathbf{p}$. We emphasize that B is a set of polynomial size, i.e., $|B| = \mathrm{poly}(\lambda)$. Consider an honestly generated ciphertext CT of input x. Consider honestly generated keys $sk_{C_1}, \ldots, sk_{C_{\ell_\mathbf{f}}}$. Denote the corresponding decrypted values to be $\iota_1, \ldots, \iota_{\ell_\mathbf{f}}$. If it holds that $\sum_{i=1}^{\ell_\mathbf{f}} c_i \cdot C_i(x) = \mathsf{out}^* \in B$ then we require that $\mathsf{Recover}(c_1, \iota_1, \ldots, c_{\ell_\mathbf{f}}, \iota_{\ell_\mathbf{f}})$, where $c_i \in \mathbb{F}_\mathbf{p}$, always outputs out^*.

Remark 1. *Our construction only supports the case when $B = \{0\}$ when implemented by multilinear maps that only allows for zero testing at the final level. However, if encodings of 1 are given out at the top level, then B can be defined to be the set $\{0, \ldots, \mathrm{poly}(\lambda)\}$, where* poly *is a fixed polynomial.*

Remark 2 ((B, B')-Correctness)**.** *We can also consider a property that we call (B, B')-correctness. It is the same as B-correctness except that the co-efficients c_i input to the above evaluation algorithm has to be in the set $B' \subseteq \mathbb{F}_\mathbf{p}$.*

Remark 3 (Alternate Notation of Evaluation)**.** *Instead of feeding coefficients to the evaluation algorithm, we can directly feed in the description of the linear function. That is, if $\mathsf{out}^* \leftarrow \mathsf{Recover}(\mathfrak{f}, (\iota_1, \ldots, \iota_{\ell_{\ell_\mathbf{f}}}))$ with $\mathfrak{f}$ being a linear function then we require that $\mathfrak{f}(C_1(x), \ldots, C_{\ell_{\ell_\mathbf{f}}}) = \mathsf{out}^*$, where ι_i is obtained by decrypting a functional key of C_i with x.*

Remark 4 (General Recover Functions)**.** *Here, we are considering recover algorithms that compute a linear function on the outputs of the arithmetic circuits. This suffices for our work. However, we can envision more general type of evaluation functions which compute an arbitrary polynomial on the outputs of arithmetic circuits.*

3.2 Semi-Functional Security

We introduce a notion of semi-functional security associated with projective arithmetic FE. We refer the reader to the technical overview for an informal intuition behind the notion of semi-functional security.

We define the following two auxiliary algorithms.

Semi-Functional Key Generation, $\mathsf{sfKG}(\mathsf{MSK}, C, \theta)$: On input master secret key MSK, arithmetic circuit C, value θ, it outputs a semi-functional key sk_C.

Semi-Functional Encryption, $\mathsf{sfEnc}(\mathsf{MSK}, 1^{\ell_{\mathsf{inp}}})$: On input master secret key MSK and ℓ_{inp}, it outputs a semi-functional ciphertext CT.

We now introduce two security properties. We start with the first property, namely indistinguishability of semi-functional keys.

This property states that it should be hard for an efficient adversary to distinguish a semi-functional key associated with circuit C and value v from an honestly generated key associated with C. Additionally, the adversary can request for other semi-functional keys or honestly generated keys. The ciphertexts will be honestly generated.

Definition 6 (Indistinguishability of Semi-Functional Keys)**.** *Consider a projective arithmetic functional encryption scheme* $\mathsf{PAFE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{ProjectDec}, \mathsf{Recover})$. *We say that*

PAFE *satisfies* **indistinguishability of semi-functional keys** *with respect to* sfKG *if for any PPT adversary $\mathcal{A}$ there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that*

$$\mathsf{Advtge}_{\mathcal{A}}^{\mathsf{PAFE}}(\lambda) = \left|\Pr[\mathsf{Expt}_{\mathcal{A}}^{\mathsf{PAFE}}(\lambda, 0) = 1] - \Pr[\mathsf{Expt}_{\mathcal{A}}^{\mathsf{PAFE}}(\lambda, 1) = 1]\right| \leq \mathsf{negl}(\lambda),$$

for all sufficiently large $\lambda \in \mathbb{N}$, where for each $b \in \{0,1\}$ and $\lambda \in \mathbb{N}$ the experiment $\mathsf{Expt}_{\mathcal{A}}^{\mathsf{PAFE}}(1^\lambda, b)$*, modeled as a game between the adversary $\mathcal{A}$ and a challenger, is defined as follows:*

1. **Setup phase:** *The challenger samples* $\mathsf{MSK} \leftarrow \mathsf{Setup}(1^\lambda)$.
2. **Message queries:** *On input 1^λ the adversary submits $(x_1, \ldots, x_{\ell_{\mathbf{x}}})$ for some polynomial $\ell_{\mathbf{x}} = \ell_{\mathbf{x}}(\lambda)$.*
3. **Function queries:** *The adversary also submits arithmetic circuit queries to the challenger. There are three tuples the adversary submits:*
   - *This comprises of circuits and values associated with every circuit; $(C_1^0, \theta_1, \ldots, C_{\ell_{\mathbf{f}}}^0, \theta_{\ell_{\mathbf{f}}})$. Here, $\theta_j \in \mathbb{F}_{\mathbf{p}}$.*
   - *This comprises of just circuits; $(C_1^1, \ldots, C_{\ell'_{\mathbf{f}}}^1)$.*
   - *This corresponds to a challenge circuit pair query (C^*, θ^*)*
4. **Challenger's response:** *The challenger replies with $(\mathsf{CT}_1, \ldots, \mathsf{CT}_{\ell_{\mathbf{x}}})$, where $\mathsf{CT}_i \leftarrow \mathsf{Enc}(\mathsf{MSK}, x_i)$ for every $i \in [\ell_{\mathbf{x}}]$. It also sends the following functional keys: for every $j \in [\ell_{\mathbf{f}}]$,*
   - *$sk_{C_j^0} \leftarrow \mathsf{sfKG}(\mathsf{MSK}, C_j^0, \theta_j)$.*
   - *$sk_{C_j^1} \leftarrow \mathsf{KeyGen}(\mathsf{MSK}, C_j^1)$.*
   - *If $b = 0$, generate $sk_{C^*} \leftarrow \mathsf{sfKG}(\mathsf{MSK}, C^*, \theta^*)$. Otherwise generate $sk_{C^*} \leftarrow \mathsf{KeyGen}(\mathsf{MSK}, C^*)$.*
5. **Output phase:** *The adversary outputs a bit b' which is defined as the output of the experiment.*

The second property is indistinguishability of semi-functional ciphertexts. This property states that it should be hard for an efficient adversary to distinguish honestly generated ciphertext of x from a semi-functional ciphertext. In this experiment, it is required that the adversary only gets semi-functional keys associated with circuits C_i and value v_i such that $v_i = C_i(x)$.

Definition 7 (Indistinguishability of Semi-Functional Ciphertexts)**.** *Consider a projective arithmetic functional encryption scheme* $\mathsf{PAFE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{ProjectDec}, \mathsf{Recover})$*. We say that* PAFE *satisfies* **indistinguishability of semi-functional ciphertexts** *with respect to* sfEnc *if for any PPT adversary $\mathcal{A}$ there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that*

$$\mathsf{Advtge}_{\mathcal{A}}^{\mathsf{PAFE}}(\lambda) = \left|\Pr[\mathsf{Expt}_{\mathcal{A}}^{\mathsf{PAFE}}(\lambda, 0) = 1] - \Pr[\mathsf{Expt}_{\mathcal{A}}^{\mathsf{PAFE}}(\lambda, 1) = 1]\right| \leq \mathsf{negl}(\lambda),$$

for all sufficiently large $\lambda \in \mathbb{N}$, where for each $b \in \{0,1\}$ and $\lambda \in \mathbb{N}$ the experiment $\mathsf{Expt}_{\mathcal{A}}^{\mathsf{PAFE}}(1^\lambda, b)$*, modeled as a game between the adversary $\mathcal{A}$ and a challenger, is defined as follows:*

1. **Setup phase:** *The challenger samples* $\mathsf{MSK} \leftarrow \mathsf{Setup}(1^\lambda)$.
2. **Message queries:** *On input 1^λ the adversary submits $(x_1, \ldots, x_{\ell_{\mathbf{x}}})$ for some polynomial $\ell_{\mathbf{x}} = \ell_{\mathbf{x}}(\lambda)$ and it also sends the challenge query x^* .*
3. **Function queries:** *The adversary also submits arithmetic circuit queries to the challenger. The query is of the form $(C_1, \theta_1, \ldots, C_{\ell_{\mathbf{f}}}, \theta_{\ell_{\mathbf{f}}})$. It should hold that $\theta_j = C_j(x^*)$ for every $j \in [\ell_{\mathbf{f}}]$. If it does not hold, the experiment is aborted.*

4. **Challenger's response:** *The challenger replies with* $(\mathsf{CT}_1, \ldots, \mathsf{CT}_{\ell_{\mathbf{x}}})$*, where* $\mathsf{CT}_i \leftarrow \mathsf{Enc}(\mathsf{MSK}, x_i)$ *for every* $i \in [\ell_{\mathbf{x}}]$*. It sends* $\mathsf{CT}^* \leftarrow \mathsf{Enc}(\mathsf{MSK}, x^*)$ *only if* $b = 0$*, otherwise it sends* $\mathsf{CT}^* \leftarrow \mathsf{sfEnc}\left(\mathsf{MSK}, 1^{|x^*|}\right)$*. Finally, it sends the following functional keys: for every* $j \in [\ell_{\mathbf{f}}]$*, compute* $sk_{C_j} \leftarrow \mathsf{sfKG}(\mathsf{MSK}, C_j, \theta_j)$*.*
5. **Output phase:** *The adversary outputs a bit* b' *which is defined as the output of the experiment.*

Remark 5. *One can also define a stronger property where instead of submitting one challenge message* x^**, the challenger submits a challenge message pair* (x_0^*, x_1^*) *and the requirement that for every circuit* C_j *query,* $C_j(x_0^*) = C_j(x_1^*)$*. The reduction, in response, encrypts* x_b^* *where* b *is the challenge bit. It can be seen that this stronger security property is implied by the above property.*

We now define semi-functional security property.

Definition 8. *We say that a projective arithmetic FE scheme, over* $\mathbb{F}_{\mathbf{p}}$*, is said to be* **semi-functionally secure** *if it satisfies both (i) indistinguishability of semi-functional keys property and, (ii) indistinguishability of semi-functional ciphertexts property.*

3.3 Other Notions

We also consider the following two notions of projective arithmetic FE.

Constant Degree Projective Arithmetic FE. In this work, we are interested in projective arithmetic FE for circuits that compute constant degree arithmetic circuits. In particular, we consider constant degree arithmetic circuits over arbitrary field $\mathbb{F}_{\mathbf{p}}$.

Multiplicative Overhead in Encryption Complexity. We say that a projective arithmetic FE scheme, over field $\mathbb{F}_{\mathbf{p}}$, satisfies multiplicative overhead in encryption complexity property if the complexity of encrypting x is $|x| \cdot \mathrm{poly}(\lambda, \log(\mathbf{p}))$. That is,

Definition 9 (Multiplicative Overhead in Encryption Complexity)**.** *Consider a projective arithmetic FE scheme* $\mathsf{PAFE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{ProjectDec})$*, over field* $\mathbb{F}_{\mathbf{p}}$*. We say that* PAFE *satisfies multiplicative overhead in encryption complexity if* $|\mathsf{Enc}(\mathsf{MSK}, x)| = |x| \cdot \mathrm{poly}(\lambda, \log(\mathbf{p}))$*, where* MSK *is the secret key generated during setup.*

Circuits versus Polynomials. Often in this manuscript, we interchangeably use arithmetic circuits over $\mathbb{F}_{\mathbf{p}}$ with polynomials computed over $\mathbb{F}_{\mathbf{p}}$. If there is a polynomial p over $\mathbb{F}_{\mathbf{p}}$ having $\mathrm{poly}(\lambda)$ number of terms then there is a $\mathrm{poly}'(\lambda)$-sized arithmetic circuit over $\mathbb{F}_{\mathbf{p}}$, where poly and poly$'$ are polynomials. However, the reverse in general need not be true: if there is a $\mathrm{poly}'(\lambda)$-sized arithmetic circuit over $\mathbb{F}_{\mathbf{p}}$ then the associated polynomial could have exponentially many terms. For example: $(x_1 + x_2) \cdots (x_{2n-1} + x_{2n})$ has a succinct circuit representation but when expanded as a polynomial has exponential number of terms.

In this work, we are only interested in arithmetic circuits which can be expressed as polynomials efficiently. In particular, we consider arithmetic circuits of constant fan-in and constant depth.

4 (T, Φ)-Randomizing Polynomials

In this section, we define the notion of (T, Φ)-randomizing polynomials. We start by defining the notion of $\left(T, \vec{\phi}\right)$-respecting property for polynomials.

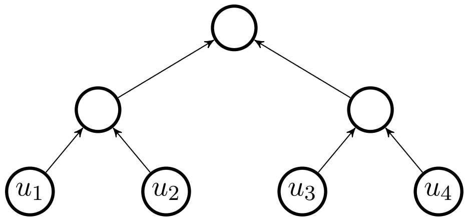

Figure 1: Tree T_1

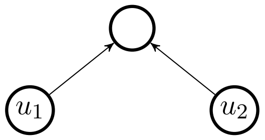

Figure 2: Tree T_2

Example: Consider a polynomial ring of 10 variables $x_1, \ldots, x_{10}$. A monomial $t = x_1x_2x_3x_4$ is (T_1, ϕ)-respecting, where $\phi(u_i) = i$. However, it is not respecting with respect to (T_1, ϕ')-respecting, where $\phi'(u_1) = 5$. Similarly, t is not (T_2, ϕ)-respecting with respect to any ϕ.

4.1 (T, ϕ)-Respecting Polynomials

Consider a binary tree T and a set of maps $\vec{\phi} = (\phi_1, \ldots, \phi_N)$ with $\phi_i : [V] \to [n]$. We introduce a property associated with n-variate polynomials, termed as $(T, \vec{\phi})$-*respecting*. To do that, we first introduce the (T, ϕ)-respecting property in the context of monomials. Once we do that, we define a polynomial p over $\mathbb{F}_{\mathbf{p}}$ to be $(T, \vec{\phi})$-respecting if every monomial t_i is (T, ϕ_i)-respecting, where $p = \sum_{i=1}^{N} c_i \cdot t_i$ and $c_i \in \mathbb{F}_{\mathbf{p}}$.

Definition 10 ((T, ϕ)-Respecting Monomials)**.** *Let $T = (V, E)$ be a tree and let $\phi : [V] \to [n]$. We say that a monomial $t \in \mathbb{F}_{\mathbf{p}}[y_1, \ldots, y_n]$ is a (T, ϕ)-respecting monomial over $\mathbb{F}_{\mathbf{p}}$ if it is a monomial and is computed as follows:*

1. *If u is a leaf node then the monomial associated with u is $t_u = x_{\phi(u)}$.*
2. *If u is a non-leaf node and let v, w are children of u then $t_u = t_v \cdot t_w$.*
3. *If $\mathbf{rt}$ is the root of T then $t = t_{\mathbf{rt}}$ is the monomial associated with $\mathbf{rt}$.*

We are now ready to define $(T, \vec{\phi})$-respecting polynomials. We also define the notion of *homogeneity* in this context. A polynomial is said to be $(T, \vec{\phi})$-respecting homogenous polynomial if firstly, it is $(T, \vec{\phi})$-respecting and secondly, it is homogenous i.e., every monomial in the polynomial has the same degree.

Definition 11 ($(T, \vec{\phi})$-Respecting (Homogenous) Polynomials)**.** *Let $T = (V, E)$ be a tree and let $\vec{\phi} = (\phi_1, \ldots, \phi_N)$, where $\phi_i : [V] \to [n]$. A polynomial $p \in \mathbb{F}_{\mathbf{p}}[y_1, \ldots, y_n]$ is a $(T, \vec{\phi})$-respecting polynomial over $\mathbb{F}_{\mathbf{p}}$ and is computed as*

$$p = \sum_{i=1}^{N} c_i t_i$$

- $c_i \in \mathbb{F}_{\mathbf{p}}$
- *t_i is a (T, ϕ_i)-respecting monomial.*

Furthermore, we define p to be a homogenous polynomial if t_i, for every $i \in [N]$, has the same degree.

Remark 6. *For every polynomial p, there exists a tree T and a set of maps $\vec{\phi}$ such that p satisfies $(T, \vec{\phi})$-respecting property.*

Finally, we define the notion of degree of (T, Φ)-respecting polynomials.

Definition 12 (Degree)**.** *The* **degree** *of a (T, Φ)-respecting polynomial $p \in \mathbb{F}_{\mathbf{p}}[y_1, \ldots, y_n]$, defined to be same as the degree of p, will be denoted by $\deg[\mathbf{p}](p)$. We say that the degree of a tuple of polynomials $(p_1, \ldots, p_N)$ is D if $D = \max\{\deg[\mathbf{p}](p_1), \ldots, \deg[\mathbf{p}](p_N)\}$.*

4.2 Definition of (T, Φ)-Randomizing Polynomials

We now define (T, Φ)-randomizing polynomials. We later show how to construct (T, Φ)-randomizing polynomials with sub-linear randomness complexity.

The general approach is to combine existing randomizing polynomials with algebraic pseudorandom generators to get the desired result. We later show how to instantiate this approach by considering specific instantiations of randomizing polynomials and algebraic pseudorandom generators. As a result, we get low degree (T, Φ)-randomizing polynomials with sub-linear randomness complexity.

(T, Φ)-(Homogenous) Randomizing Polynomials. We first recall the definitions of randomizing polynomials [IK00, AIK06].

Definition 13 (Randomizing Polynomials)**.** *A randomizing polynomials scheme* $\mathsf{RP} = (\mathsf{E}, \mathsf{D})$ *for a class of circuits* $\mathcal{C}$ *has the following syntax:*

- *Encoding,* $\mathsf{E}(1^\lambda, C, x)$*: On input security parameter* λ*, circuit* C*, input* x*, it outputs an encoding of* $C \in \mathcal{C}$ *and* x*,* $\langle (C, x) \rangle$*.*
- *Decoding,* $\mathsf{D}(\langle (C, x) \rangle)$*: On input encoding of* C *and* x*, it output the decoded value* α*.*

RP *is required to satisfy the following properties:*

- Correctness*: For every security parameter* $\lambda \in \mathbb{N}$*, circuit* C *and input* x*,* $C(x) = \mathsf{D}(\mathsf{E}(1^\lambda, C, x))$*.*
- Efficiency*: The typical efficiency we require is that the depth complexity of* E *should be independent of size/ depth complexity of* C*.*
- Security*: For every PPT adversary* $\mathcal{A}$*, for large enough security parameter* $\lambda \in \mathbb{N}$*, circuit* C *and input* x*, there exists a simulator* Sim *such that:*

$$\left\{ \mathsf{E}(1^\lambda, C, x) \right\} \cong \left\{ \mathsf{Sim}(1^\lambda, 1^{|C|}, C(x)) \right\}$$

The indistinguishability can either be computational, statistical or perfect.

We now give the definition of (T, Φ)-randomizing polynomials. Its a randomizing polynomials scheme with two additional requirements: (i) the polynomials associated with the encoding algorithm are (T, Φ)-respecting and, (ii) the decoding algorithm should be of a specific form – in particular, it should be composed of many linear functions (computed adaptively). Furthermore, we also consider the notion of (T, Φ)-*homogenous* randomizing polynomials: the polynomials in the encode procedure are homogenous.

We define the special decoding property, namely composable linear decoding property, that is necessary for our construction.

B-Composable Linear Decoding: We are interested in the following decoding property, parameterized by a constant $B \in \mathbb{F}_\mathbf{p}$. Consider a circuit C and input x. Let L be a list such that the i^{th} element in L is $\langle (C, x) \rangle_i \in \mathbb{F}_\mathbf{p}$, where $\langle (C, x) \rangle_i$ is the i^{th} element in $\langle (C, x) \rangle$. The decoding algorithm D proceeds as follows:

- The decoder has oracle access to elements in L. It proceeds in a sequence of T steps.
- In the i^{th} step, it submits a linear function $\mathfrak{f}_i$ to L and in turn receives a value $v_i \in \{0, \ldots, B\}$ which is the output of $\mathfrak{f}_i(L)$. If $\mathfrak{f}_i(L) \notin \{0, \ldots, B\}$ then oracle outputs $\perp$.
- Let v_T be the output of the oracle in the T^{th} step.

The output of this process is v_T. Such a decoder will be called B-*composable linear decoder.*

Example: Information theoretic Yao's garbling schemes [IK02] for NC^1 circuits is an example of a randomizing polynomial scheme which has a composable linear decoder. This is similar to Yao's garbling schemes [Yao86, BHR12] except that the encryption used in the garbled table is a *one-time encryption*. This means that the length of the keys used in every layer grows exponentially from the output gate to the input layer.

At a high level, the randomizing polynomial of a circuit C and input x consists of the following: (i) for every gate G in C, garbled table with four entries, one for each entry of the truth table of the gate, (ii) wire keys corresponding to the input x. Every entry in the garbled table is a double one-time encryption of the appropriate output wire key appended with a suitable fixed mask (to indicate the successful decryption) of length λ. The two keys used to perform the double encryption are the appropriate input wire keys of the i^{th} gate. For every output gate, the wire keys K_w^0 and K_w^1 are set to all zeroes and all ones respectively.

The decoding algorithm roughly proceeds as follows: to decode the i^{th} gate, it first has to decode its children gates (these are the gates whose output wires are fed to the i^{th} gate). Once it obtains the input wire keys of i^{th} gate, it tries to decrypt all the entries of the table. This decryption is a linear operation since we are using a one time encryption. The decryption is successful for one of the entries in the table – the decryption is successful if the decoder recovers a fixed mask. We set the fixed mask to be all zeroes string.

This scheme has a B-composable linear decoder for $B = \{0\}$. This can be seen as follows: the linear function submitted to the oracle corresponding to the i^{th} gate is used to indicate which of the four entries yielded a successful decryption query. Also this linear function is computed on the input wire keys as well as all the gates that are in the sub-circuit of the i^{th} gate. More formally, the decoder proceeds in the following steps:

- The set of all wires in C is denoted by $\mathcal{W}$. The length of a wire key associated with wire $\mathbf{w}$ is $L_{\mathbf{w}}$. Let the input wires of C be $w_1, \ldots, w_\ell$. As part of encoding of (C, x), the wire keys $K_{w_1}, \ldots, K_{w_\ell}$ are given out.
- Let $G_1, \ldots, G_n$ be the gates in C. Let the garbled table associated with G_i be T_i. Recall that T_i consists of four entries $T_i[1], \ldots, T_i[4]$. We say that $T_i[j]$ is a *valid entry* if it is decrypted during the evaluation of the garbled circuit.
- Recursively, the decoder constructs linear functions $\{\widehat{\mathfrak{f}^k_{\mathbf{w}}}\}_{\mathbf{w} \in \mathcal{W}, k \in [\lambda+1, L_{\mathbf{w}}+\lambda]}$ and $\{\mathfrak{f}_i^{j.k}\}_{i \in [n], j \in [4], k \in [\lambda]}$ such that $\hat{\mathfrak{f}^k_{\mathbf{w}}}$ on input the encoding, outputs the k^{th} least significant bit of the wire key of $\mathbf{w}$ obtained during evaluation and $\mathfrak{f}_i^{j.k}$ outputs 0 if and only if the k^{th} least significant bit of $T_i[j]$ is 0. Note that the decoder itself do not submit the functions $\{\widehat{\mathfrak{f}_{\mathbf{w}}}\}$ to the oracle, but instead use them to construct functions $\{\mathfrak{f}_i^{j.k}\}$ which are submitted to the oracle.
  - For every input wire $\mathbf{w}$, $\widehat{\mathfrak{f}^k_{\mathbf{w}}}$ outputs the k^{th} least significant bit of the wire key $K_{\mathbf{w}}$.
  - For every gate G_i with input wires $\mathbf{w}_1$ and $\mathbf{w}_2$, $\mathfrak{f}_i^{j.k}$ first recovers the wire keys $K_{\mathbf{w}_1}$ and $K_{\mathbf{w}_2}$ using functions $\{\widehat{\mathfrak{f}^{k'}_{\mathbf{w}_1}}\}$ and $\{\widehat{\mathfrak{f}^{k'}_{\mathbf{w}_2}}\}$ for every $k' \in [\lambda+1, L_{\mathbf{w}} + \lambda]$. It then outputs 0 if k^{th} least significant bit of $T_i[j] - (K_{\mathbf{w}_1} + K_{\mathbf{w}_2})$ is 0.

  Suppose $\mathbf{w}_3$ is the output wire of G_i. If for some j, $\mathfrak{f}_i^{j.k}$ outputs 0 for every $k \in [\lambda]$ then $\widehat{\mathfrak{f}^{k'}_{\mathbf{w}_3}}$ outputs $(k')^{th}$ least significant bit of $(T_i[j] - (K_{\mathbf{w}_1} + K_{\mathbf{w}_2}))_{k'}$ for $k' \in [\lambda+1, L_{\mathbf{w}} + \lambda]$.
- If the output of the linear function corresponding to the output gate G_i^{out} is all zeroes then the corresponding output bit is set to 0, otherwise the output bit is set to 1.

This concludes the decoding process.

We now define the notion of homogenous randomizing polynomials.

Definition 14 ((T, Φ)-(Homogenous) Randomizing Polynomials)**.** *Let $T = (V, E)$ be a tree and let $\Phi = \left(\overrightarrow{\phi_1}, \ldots, \overrightarrow{\phi_N}\right)$. A scheme* $\mathsf{RP} = (\mathsf{E}, \mathsf{D})$ *is said to be a (T, Φ)-randomizing polynomials* *scheme over $\mathbb{F}_{\mathbf{p}}$ for a class of circuits $\mathcal{C} = \{\mathcal{C}_\lambda\}$ if:*

1. (E, D) *is a randomizing polynomial scheme for $\mathcal{C}$.*
2. D *is a B-composable linear decoder for some constant $B \in \mathbb{F}_{\mathbf{p}}$.*
3. *For every circuit C and input x, we have $\mathsf{E}(C, x; r) = (p_1(x, r), \ldots, p_N(x, r))$, where $p_i \in \mathbb{F}_{\mathbf{p}}[y_1, \ldots, y_n]$ is a $(T, \overrightarrow{\phi_i})$-respecting polynomial.*

Furthermore if every $i \in [n]$, p_i is **homogenous** *and of the same degree then we term* RP *to be a homogenous (T, Φ)-randomizing polynomial scheme* RP*. We say that the* **degree** *of homogenous (T, Φ)-randomizing polynomial scheme* RP *is D if for every $i \in [n]$, $\deg[\mathbf{p}](p_i) = D$.*

Definition 15 ((T, Φ)-Randomizing Polynomials with ε-Sub-linear Randomness Complexity)**.** *A (T, Φ)-randomizing polynomials scheme for $\mathcal{C}$ is said to have ε-sub-linear randomness complexity if the size of the randomness used to encode a circuit C and input x is $|C|^\varepsilon \cdot \mathrm{poly}(\lambda)$.*

We now switch to constructing (T, Φ)-RP with sub-linear randomness. We first start by sketching the general approach and later show how to instantiate it.

4.3 (T, Φ)-RP with Sub-linear Randomness: General Approach

Existing constructions of randomizing polynomials already yield (T, Φ)-randomizing polynomials – although we still need to argue that they satisfy the composable linear decoder property. To ensure sub-linear randomness complexity, we use algebraic pseudorandom generators of appropriate stretch. We begin by describing algebraic pseudorandom generators.

Algebraic PRGs. We first define the notion of algebraic PRGs.

Definition 16 (p-Algebraic PRGs)**.** *An arithmetic circuit* $\mathsf{PRG} : \mathbb{Z}_p^n \to \mathbb{Z}_p^m$ *over $\mathbb{Z}_p$, where $m > n$, is said to be an algebraic pseudorandom generator if the following holds for every PPT distinguisher D, for sufficiently large $\lambda \in \mathbb{N}$,*

$$\left| \Pr_{s \xleftarrow{\$} \mathbb{Z}_p^n} [1 \leftarrow D(\mathsf{PRG}(s))] - \Pr_{r \xleftarrow{\$} \mathbb{Z}_p^m} [1 \leftarrow D(r)] \right| \leq \mathsf{negl}(\lambda)$$

where negl *is a negligible function.*

Approach. We start with a randomizing polynomials scheme $\mathsf{RP} = (\mathsf{E}, \mathsf{D})$ for a class of circuits $\mathcal{C} = \{\mathcal{C}_\lambda\}_{\lambda \in \mathbb{N}}$. Let the encoding of circuit C, given by $\mathsf{E}(C, \cdot; \cdot) = (p_1(\cdot; \cdot), \ldots, p_N(\cdot; \cdot))$.

Let the randomness complexity required to encode C be ℓ_R. We use an algebraic pseudorandom generator $\mathsf{PRG} : \mathbb{Z}_p^{\ell_R^\varepsilon} \to \mathbb{Z}_p^{\ell_R}$ for some constant $\varepsilon < 1$.

We now define the new randomizing polynomials scheme $(\mathsf{E}', \mathsf{D}')$ that enjoys ε-sub-linear randomness complexity.

$\mathsf{E}'(C, x; r)$: On input circuit C, input x and randomness r; first compute $\mathsf{PRG}(r)$ to get R. Output $\mathsf{E}(C, x; R)$.

The decode algorithm D' is defined to be identical to D.

Thus we have the following theorem.

Theorem 4. *The randomizing polynomials scheme* $(\mathsf{E}', \mathsf{D}')$ *is secure assuming the security of* (E, D) *and security of* PRG*.*

Ensuring (T, Φ)-respecting property. We now remark how to ensure (T, Φ)-respecting property for randomizing polynomials. We start with a randomizing polynomials scheme (E, D).

Consider a tree T. As long as the number of leaves in T is the same as the degree of homogeneity of the polynomials $\{p_i\}_{i \in [N]}$, with $p_i \in \mathbb{F}_{\mathbf{p}}[x_1, \ldots, x_n]$ associated with the encode function $\mathsf{E}(C, \cdot; \cdot)$ in the randomizing polynomials scheme, we can find a set of maps $\vec{\phi_i}$ such that p_i is $(T, \vec{\phi_i})$-respecting. Let $p_i = \sum_{j=1}^{K_i} c_{i,j} t_{i,j}$, where $c_{i,j} \in \mathbb{F}_{\mathbf{p}}$. We can always find $\phi_{i,j} : [V] \to [n]$ such that when $x_{\phi_{i,j}(u)}$ is assigned to node u and the tree T is evaluated to yield a monomial $t_{i,j}$.

This in itself is not sufficient to ensure that the randomizing polynomials scheme is (T, Φ)-respecting, where $\Phi = (\vec{\phi}_1, \ldots, \vec{\phi}_N)$, since Φ could depend on C. However, there are randomizing polynomials schemes (demonstrated next), where for every polynomial $p_i = \sum_{j=1}^{K_i} c_{i,j} t_{i,j}$ derived from encoding of C, only the coefficients depend on C and in particular, the monomials are independent of C. This means that for every circuit C, the polynomials $\{p_i\}_{i \in [N]}$ derived from the encoding of C are (T, Φ)-respecting, where $\Phi = (\vec{\phi}_1, \ldots, \vec{\phi}_N)$. Thus, the randomizing polynomials scheme is (T, Φ)-respecting.

4.4 Tools to Instantiate General Approach

We instantiate the general approach sketched in the previous section by using specific algebraic PRGs and low degree randomizing polynomials.

I. Instantiations of Algebraic PRGs. We use a low degree pseudorandom generator with polynomial stretch. In particular, we use a candidate proposed by Goldreich [Gol11]: the pseudorandom generator $\mathsf{PRG} : \mathbb{Z}_p^n \to \mathbb{Z}_p^m$ is designed by picking m indexing c-local predicates $P_i : \mathbb{Z}_p^c \to \mathbb{Z}_p$ and indexing functions $\chi_1, \ldots, \chi_m$ at random, with $\chi_i : [c] \to [n]$. The i^{th} output bit of $\mathsf{PRG}(x)$, for $i \in [m]$, is defined to be:

$$\mathsf{PRG}_i(s) = P_i \left(s_{\chi_i(1)}, \ldots, s_{\chi_i(c)} \right)$$

In this work, all the predicates are fixed ahead of time and in fact all of them have the same description. In particular, define the following predicate considered by O'Donnell and Witmer [OW14]:

$$P_i(x_1, \ldots, x_c) = x_1 + \ldots + x_t + Q(x_{t+1}, \ldots, x_c) \mod \mathbf{p},$$

where Q is any $(c-t)$-ary predicate. We set m to be $O(n^{\frac{t}{2}-\epsilon})$ for any constant $\epsilon > 0$.

O'Donnell and Witmer showed evidence that the above predicate is a secure PRG by suitably instantiating Q. In particular, they gave evidence of security for the following candidate.

1.49PRGAssumption: *Set $m = O(n^{1.49})$, $c = 5$ and $\mathbf{p} = 2$. Pick indexing functions $\chi_1, \ldots, \chi_m$ at random, with $\chi_i : [c] \to [n]$. We define the pseudorandom generator $\mathsf{1.49PRG} : \{0,1\}^n \to \{0,1\}^m$ such that i^{th} bit of PRG, on input seed s, is computed as follows:*

$$\mathsf{1.49PRG}_i(s) = P(s_{\chi_i(1)}, \ldots, s_{\chi_i(c)}),$$

where $P(x_1, x_2, x_3, x_4, x_5) = x_1 + x_2 + x_3 + x_4 x_5 \mod 2$.

Yet another PRG we are interested is an algebraic PRG with stretch $2 + \varepsilon$. We state the following two assumptions depending on which field we are operating upon.

2.49PRG[2]**Assumption**: *Set $m = O(n^{2.49})$, $c = 8$ and $\mathbf{p} = 2$. Pick indexing functions $\chi_1, \ldots, \chi_m$ at random, with $\chi_i : [c] \to [n]$. We define the pseudorandom generator $\mathsf{2.49[2]PRG} : \{0,1\}^n \to \{0,1s\}^m$ such that i^{th} bit of PRG, on input seed s, is computed as follows:*

$$\mathsf{2.49[2]PRG}_i(s) = P(s_{\chi_i(1)}, \ldots, s_{\chi_i(c)}),$$

where $P(x_1, x_2, x_3, x_4, x_5) = x_1 + x_2 + x_3 + x_4 + x_5 + x_6x_7x_8 \mod 2$.

2.49PRG[p]Assumption: *Set* $m = O(n^{2.49})$, $c = 8$ *and large enough prime* $\mathbf{p}$. *Pick indexing functions* $\chi_1, \ldots, \chi_m$ *at random, with* $\chi_i : [c] \to [n]$. *We define the pseudorandom generator* $\mathsf{2.49[\mathbf{p}]PRG} : \mathbb{Z}_p^n \to \mathbb{Z}_p^m$ *such that* i^{th} *bit of PRG, on input seed* s*, is computed as follows:*

$$\mathsf{2.49[\mathbf{p}]PRG}_i(s) = P(s_{\chi_i(1)}, \ldots, s_{\chi_i(c)}),$$

where $P(x_1, x_2, x_3, x_4, x_5) = x_1 + x_2 + x_3 + x_4 + x_5 + x_6x_7x_8 \mod \mathbf{p}$.

Remark 7. *For all our applications that use PRG with stretch* $n^{1.49}$ *can be replaced by any other locality-5 PRG of stretch* $n^{1+\varepsilon}$ *for constant* $\varepsilon > 0$. *Similarly, for the case of algebraic PRGs, we only require PRGs of stretch* $n^{2+\varepsilon}$ *for constant* $\varepsilon > 0$.

We note that we are currently unaware of any work that provides evidence about the security of either $\mathsf{2.49[2]PRG}$ or the $\mathsf{2.49[\mathbf{p}]PRG}$ candidate.

II. Low Degree Randomizing Polynomials [AIK06]. We consider the information theoretic version of the randomizing polynomials construction of [AIK06] defined for the class of log-depth and poly-sized formulas which corresponds to the class NC^1. More specifically, we consider formulas which when transformed, yields a balanced NC^1 circuit. If the size of the original poly-sized formula is a polynomial in λ and if the depth of the new NC^1 circuit is d, we require that 2^d is polynomial in λ[13].

On input log-depth formula C and input x, assign two random keys to every wire in C, associated with bits 0 and 1. We denote the key assigned to wire w and associated with bit b as K_w^b. In turn, K_w^b can be written as $(K_w^{b,0}||K_w^{b,1})$. If w is a output wire then $K_w^b = b$. We remark later about the size of the keys. Also associated with wire w is a random bit r_w.

The encoding of (C, x) comprises of garbled gates $\widetilde{G_1}, \ldots, \widetilde{G_\kappa}$ along with input keys $\mathbf{W}_x$, computed as a function of the keys assigned to all the wires in C. Further, for every $i \in [\kappa]$, the garbled gate $\widetilde{G_i} = (\widetilde{G_i}[a_1, a_2])_{a_1 \in \{0,1\}, a_2 \in \{0,1\}}$.

For $i \in [\kappa]$, let w_A and w_B denote the input wires of G_i and let w_C denote its output wire. For $a_1, a_2 \in \{0, 1\}$, we have:

$$\widetilde{G_i}[a_1, a_2] = K_{w_A}^{a_1 \oplus r_{w_A}, a_2} \oplus K_{w_B}^{a_2 \oplus r_{w_B}, a_1} \oplus \left(K_{w_C}^{G_i(a_1 \oplus r_{w_A}, a_2 \oplus r_{w_B})} \mid\mid r_{w_C} \oplus G_i(a_1 \oplus r_{w_A}, a_2 \oplus r_{w_B})\right)$$

For input wires $w_1, \ldots, w_n$, the keys corresponding to input x are $\mathbf{W}_x = (K_{w_1}^{x_1}||x_1 \oplus r_{w_1}, \ldots, K_{w_n}^{x_n}||x_n \oplus r_{w_n})$.

Furthermore, the decoder of the above randomizing polynomial is a B-composable linear decoder with $B = \{0\}$. The linear function for the i^{th} gates incorporates all the keys recovered in the sub-circuit of the i^{th} gate. The output of the linear function is the mask recovered at the i^{th} gate. We argue this formally below.

Claim 1. *The information-theoretic variant of [AIK06] as described above satisfies* B*-composable linear decoder property for* $B = \{0\}$.

Proof. The decoder is designed along the same lines as the decoder of the information-theoretic Yao.

[13]The type of circuits that we encode later using the randomizing polynomials already satisfy this property. Specifically, the circuit has many parallel copies of gate garbling circuit. Each of these gate garbling circuits perform some PRG computations. By making sure that PRG has a balanced NC^1 circuit, we can ensure the overall circuit is balanced.

- The set of all wires in C is denoted by $\mathcal{W}$. The length of a wire key associated with wire $\mathbf{w}$ is $L_{\mathbf{w}}+1$ – "+1" part is for the random mask. Let the input wires of C be $w_1, \ldots, w_\ell$. As part of encoding of (C, x), the wire keys $(K_{w_1}, \mathbf{r}_1), \ldots, (K_{w_\ell}, \mathbf{r}_\ell)$ are given out.
- Let $G_1, \ldots, G_n$ be the gates in C. Let the garbled table associated with G_i be T_i. Recall that T_i consists of four entries $T_i[0], \ldots, T_i[3]$. We say that $T_i[j]$ is a *valid entry* if it is decrypted during the evaluation of the garbled circuit.
- Recursively, the decoder constructs linear functions $\{\widehat{\mathfrak{f}_{\mathbf{w}}^{k}}\}_{\mathbf{w}\in\mathcal{W}, k\in[2,L_{\mathbf{w}}+1]}$ and $\{\mathfrak{f}_i\}_{i\in[n]}$ such that $\mathfrak{f}_{\mathbf{w}}^{\hat{k}}$ on input the encoding, outputs the k^{th} least significant bit of the wire key of $\mathbf{w}$ obtained during evaluation and $\mathfrak{f}_i^{j}$ outputs 0 if and only if the least significant bit (corresponding to random mask) of the wire key obtained during the evaluation of gate G_i is 0. Note that the decoder itself do not submit the functions $\{\widehat{\mathfrak{f}_{\mathbf{w}}}\}$ to the oracle, but instead uses them to construct functions $\{\mathfrak{f}_i\}$ which are submitted to the oracle.
  - For every input wire $\mathbf{w}$, $\widehat{\mathfrak{f}_{\mathbf{w}}^{k}}$ outputs the k^{th} least significant bit of the wire key $K_{\mathbf{w}}$.
  - For every gate G_i with input wires $\mathbf{w}_1$ and $\mathbf{w}_2$, do the following: $\mathfrak{f}_i$, with j hardwired, first recover the wire keys $K_{\mathbf{w}_1}$ and $K_{\mathbf{w}_2}$ using functions $\{\widehat{\mathfrak{f}_{\mathbf{w}_1}^{k'}}\}$ and $\{\widehat{\mathfrak{f}_{\mathbf{w}_2}^{k'}}\}$ for every $k' \in [2, L_{\mathbf{w}}+1]$. It then outputs 0 if k^{th} least significant bit of $T_i[j] - (K_{\mathbf{w}_1} + K_{\mathbf{w}_2})$ is 0. Else it outputs 1.

  Suppose $\mathbf{w}_3$ is the output wire of G_i. Let the output wire of $G_{i'}$ be $\mathbf{w}_1$ and output wire of $G_{i''}$ be $\mathbf{w}_2$. Let b_0 and b_1 be the bits output by $\mathfrak{f}_{i'}$ and $\mathfrak{f}_{i''}$ respectively. Set j to be the integer representation of $b_0 b_1$. Now, the function $\widetilde{\mathfrak{f}_{\mathbf{w}_3}^{k'}}$ outputs $(k')^{th}$ least significant bit of $(T_i[j] - (K_{\mathbf{w}_1}||\mathbf{r}_1 + K_{\mathbf{w}_2}||\mathbf{r}_2))_{k'}$ for $k' \in [2, L_{\mathbf{w}}+1]$.
- If the output of the linear function corresponding to the output gate G_i^{out} is 0 then the corresponding output bit is set to 0, otherwise the output bit is set to 1.

This concludes the decoding process. □

Size of Keys. We now remark about the size of the keys. To do this, we view the circuit C as layered with d number of layers, where d is the depth of C. The first layer corresponds to the input gates and the d^{th} layer corresponds to the output gates. The i^{th} layer (except the first layer) has its input wires coming from the $(i-1)^{th}$ layer. Similarly, the output wires of i^{th} layer (except the d^{th} layer) are input to $(i+1)^{th}$ layer.

All the keys corresponding to i^{th} layer have size at most $\mathrm{O}(2^{d-i})$. That is, the keys at the 0^{th} layer have size at most $\mathrm{O}(2^{d})$.

Size of random masks. There is one random mask r_w assigned for every wire w and thus, the total size of all the random masks is upper bounded by the size of the circuit, $|C|$.

Representing Encoding as Polynomials. Assign polynomial $\mathbf{Q}[j, a_1, a_2]$ associated with j^{th} bit of $\widetilde{G_i}[a_1, a_2]$, variable $\mathbf{Y}_1[j, w, a_1, a_2]$ associated with j^{th} element of $K_w^{a_1 \oplus r_w, a_2}$ and variable $\mathbf{Y}_2[w]$ associated with bit r_w. Assign polynomial $\mathbf{T}[w_i, b]$ associated with input wire w_i corresponding to the random wire label $b \oplus r_{w_i}$.

The polynomial $\mathbf{Q}[j, a_1, a_2]$ over $\mathbb{F}_2$ is computed as follows: We consider the case when $a_1 = 1$ and $a_2 = 0$. We also consider G_i to be a NAND gate. The other three cases follow symmetrically.

1. When j is <u>not</u> the least significant bit of $K_w^{a_1 \oplus r_w, a_2}$:

$$\begin{aligned}\mathbf{Q}[j,1,0] \quad = \quad & (1-\mathbf{Y}_2[w_A]) \cdot \mathbf{Y}_1[j,w_A,1,0] + \mathbf{Y}_2[w_A] \cdot \mathbf{Y}_1[j,w_A,0,0] \\ & +(1-\mathbf{Y}_2[w_B]) \cdot \mathbf{Y}_1[j,w_B,0,1] + \mathbf{Y}_2[w_B] \cdot \mathbf{Y}_1[j,w_B,1,1] \\ & +(1-\mathbf{Y}_2[w_A]) \cdot (1-\mathbf{Y}_2[w_B]) \cdot \mathbf{Y}_1[j,w_A,1,0] \\ & +(1-\mathbf{Y}_2[w_A]) \cdot (\mathbf{Y}_2[w_B]) \cdot \mathbf{Y}_1[j,w_A,1,1] \\ & +(\mathbf{Y}_2[w_A]) \cdot (1-\mathbf{Y}_2[w_B]) \cdot \mathbf{Y}_1[j,w_A,0,0] \\ & +(\mathbf{Y}_2[w_A]) \cdot (\mathbf{Y}_2[w_B]) \cdot \mathbf{Y}_1[j,w_A,0,1]\end{aligned}$$

2. When j is the least significant bit of $K_w^{a_1 \oplus r_w, a_2}$ (this corresponds to the random mask):

$$\begin{aligned}\mathbf{Q}[j,1,0] \quad = \quad & (1-\mathbf{Y}_2[w_A]) \cdot \mathbf{Y}_1[j,w_A,1,0] + \mathbf{Y}_2[w_A] \cdot \mathbf{Y}_1[j,w_A,0,0] \\ & +(1-\mathbf{Y}_2[w_B]) \cdot \mathbf{Y}_1[j,w_B,0,1] + \mathbf{Y}_2[w_B] \cdot \mathbf{Y}_1[j,w_B,1,1] \\ & +(1-\mathbf{Y}_2[w_A]) \cdot (1-\mathbf{Y}_2[w_B]) \cdot (1-\mathbf{Y}_2[w_C]) \\ & +(1-\mathbf{Y}_2[w_A]) \cdot (\mathbf{Y}_2[w_B]) \cdot (1-\mathbf{Y}_2[w_C]) \\ & +(\mathbf{Y}_2[w_A]) \cdot (1-\mathbf{Y}_2[w_B]) \cdot (\mathbf{Y}_2[w_C]) \\ & +(\mathbf{Y}_2[w_A]) \cdot (\mathbf{Y}_2[w_B]) \cdot (1-\mathbf{Y}_2[w_C])\end{aligned}$$

Finally, we can set the polynomial $\mathbf{T}[w_i, a, b] = \mathbf{Y}_2[w_i] + b$, where b is a bit.

4.5 Instantiations: (T, Φ)-RP with Sub-linear Randomness

We consider two instantiations of the general approach. In the first instantiation, we first instantiate a (T, Φ)-randomizing polynomials over $\mathbb{F}_2$ and then apply the polynomial conversion procedure (Section B) to obtain (T, Φ)-randomizing polynomials over $\mathbb{F}_\mathbf{p}$. The resulting randomizing polynomial is such that the encoding of a circuit and an input is a binary string. In the other instantiation, we show how to directly instantiate a (T, Φ)-randomizing polynomials over $\mathbb{F}_\mathbf{p}$. In this instantiation, the encoding of a circuit and an input is a vector over $\mathbb{F}_\mathbf{p}$. We then plug in the appropriate PRGs for both the instantiations to get sub-linear randomness complexity.

4.5.1 Via Randomizing Polynomials over $\mathbb{F}_2$

We employ the following steps:

1. Step I: Start with (T, Φ)-RP over $\mathbb{F}_2$. In particular, we start with the construction in Section 4.4.
2. Step II: We apply the polynomial conversion procedure to Step I from Section B to obtain a (T, Φ)-RP over $\mathbb{F}_\mathbf{p}$.
3. Step III: We then instantiate the general approach (Section 4.3) using Step II and an appropriate algebraic PRG to achieve the sub-linear randomness complexity.

<u>Step I</u>: Consider the RP construction in the previous section. We now consider the wire keys over $\mathbb{F}_2$. That is, every entry in the garbled table can now be treated as XOR of wire keys. More specifically, the set of variables $\{\mathbf{Y}_1[j, w, a_1, a_2]\}$ and $\{\mathbf{Y}_2[w]\}$ take binary values.

STEP II: Applying the naive the conversion ConvPoly described in Section B to polynomials of the form $\mathbf{Q}[j,1,0]$, where $\mathbf{Y}_1[j,w,a_1,a_2]$ are assigned boolean values (Case 1), we get the resulting polynomials over field $\mathbb{F}_\mathbf{p}$. Further as described in the previous section, it can be ensured that these polynomials are (T,Φ)-respecting, for a suitably defined T and Φ.

STEP III: In this case, $\mathbf{Y}_1[\cdot,\cdot,\cdot,\cdot]$ can be substituted with a polynomial computing a boolean PRG, denoted by PRG_1, that converts strings of size, say ℓ, into strings of at least length $(2+\varepsilon)\cdot\ell$, where $\varepsilon > 0$ is a constant. Further, we substitute $\mathbf{Y}_2[\cdot]$ with a polynomial computing a boolean PRG, denoted by PRG_2, that converts strings of length, say ℓ, into strings of length $(1+\varepsilon)\cdot\ell$, where $\varepsilon > 0$ is a constant.

Theorem 5. *Set* $\mathsf{PRG}_1 = \mathsf{2.49[2]PRG}$ *and* $\mathsf{PRG}_2 = \mathsf{1.49PRG}$. *Assuming that 2.49PRG*[2]*Assumption and 2.49PRG*[**p**]*Assumption holds, there exists a* (T,Φ)*-randomizing polynomials scheme with sub-linear randomness complexity.*

4.5.2 Via Randomizing Polynomials over $\mathbb{F}_\mathbf{p}$

We employ the following steps:

1. STEP I: Instantiate (T,Φ)-RP over $\mathbb{F}_\mathbf{p}$.
2. STEP II: Plug in the appropriate PRG in Step I to obtain (T,Φ)-RP over $\mathbb{F}_\mathbf{p}$ with sub-linear randomness complexity.

STEP I: Consider the RP construction in the previous section. The wire keys are elements over $\mathbb{F}_\mathbf{p}$. The random masks are binary values. More specifically, the set of variables $\{\mathbf{Y}_1[j,w,a_1,a_2]\}$ take values over $\mathbb{F}_\mathbf{p}$ and $\{\mathbf{Y}_2[w]\}$ take binary values.

Theorem 6. *The scheme* (E,D) *described above is a* (T,Φ)*-randomizing polynomials for a class of circuits* $\mathcal{C}$*, where* $\mathsf{E}(C,x;r) \in \mathbb{Z}_p^{\mathrm{poly}(\lambda)}$ *and* $\mathsf{E}(\cdot,\cdot;\cdot)$ *can be represented by polynomials of degree 3.*

As before, we analyze the degree when the variables in the polynomials $\mathbf{Q}[\cdot,\cdot,\cdot]$ and $\mathbf{T}[\cdot,\cdot,\cdot]$ are substituted with polynomials. In particular, $\mathbf{Y}_1[\cdot,\cdot,\cdot,\cdot]$, $\mathbf{Y}_2[\cdot]$ be substituted by polynomials of degree $\mathbf{D}_{\mathbf{Y}_1}$ and $\mathbf{D}_{\mathbf{Y}_2}$ respectively.

Theorem 7. *The above randomizing polynomials scheme (where the wire keys are vectors over* $\mathbb{F}_\mathbf{p}$*) is such that the polynomials* $\mathbf{Q}[\cdot,\cdot,\cdot]$ *have degree* $\max\{2\mathbf{D}_{\mathbf{Y}_2}+\mathbf{D}_{\mathbf{Y}_1},\ 3\mathbf{D}_{\mathbf{Y}_2}\}$ *over* $\mathbb{F}_\mathbf{p}$.

STEP II: In this case, $\mathbf{Y}_2[\cdot]$ can be substituted with a polynomial computing a PRG over $\mathbb{Z}_p$, denoted by PRG_1, that converts vectors in $\mathbb{Z}_p$ of length ℓ into vectors of length $(2+\varepsilon)\cdot\ell$, where $\varepsilon > 0$ is a constant. As before, we substitute $\mathbf{Y}_2[\cdot]$ with a polynomial computing a boolean PRG, denoted by PRG_2, that converts strings of length, say ℓ, into strings of length $(1+\varepsilon)\cdot\ell$, where $\varepsilon > 0$ is a constant. In particular, we assign $\mathsf{PRG}_1 = \mathsf{2.49}[\mathbf{p}]\mathsf{PRG}$ and $\mathsf{PRG}_2 = \mathsf{1.49PRG}$. We note that the degree of PRG_1 over $\mathbb{F}_\mathbf{p}$ is 3. If we apply polynomial conversion from Section B on PRG_2, which is computed over $\mathbb{F}_2$, we end up with a polynomial of degree 5 over $\mathbb{F}_\mathbf{p}$.

Theorem 8. *Set* $\mathsf{PRG}_1 = \mathsf{2.49}[\mathbf{p}]\mathsf{PRG}$ *and* $\mathsf{PRG}_2 = \mathsf{1.49PRG}$. *Assuming 2.49PRG*[**p**]*Assumption and 1.49PRGAssumption, there exists a* (T,Φ)*-randomizing polynomials scheme with sub-linear randomness complexity. Furthermore, it is associated with* B*-composable linear decoder with* $B = \{0\}$ *(see Section 4.4). From Theorem 6 and 7, the degree of this scheme is 15.*

It is possible to weaken the PRG assumptions by considering PRGs of smaller stretch. See Remark 7 for more details.

Getting the Degree Below 15. The main reason why degree 15 appears is because, for every gate, we need to compute a function of three random masks corresponding to its input and output wires. Intuitively, random masks are used to emulate the role of permutation in Yao's garbled circuits – it is necessary that we hide which of the entries in the garbled table is decrypted. We consider different strategies of emulating the role of permutation that reduces the degree further. We sketch the strategies for degree 13 and degree 7 below. We describe the strategy for degree 5 in detail.

Degree 13: Consider the polynomials $\{\mathbf{Q}\}$ defined in Section 4.5. The degree of polynomials $\mathbf{Q}[j, \cdot, \cdot]$ is 13, when j is not the least significant bit. However, the degree of $\mathbf{Q}[j, \cdot, \cdot]$ is 15, when j is the least significant bit of the associated wire key. In this polynomial, we use two field elements to represent $\mathbf{Y}_2[w_C]$ and $1 - \mathbf{Y}_2[w_C]$ respectively. These two field elements are generated using an algebraic PRG (of degree 2, with more than quadratic stretch). We then separately give out these two elements in the clear along with a mapping from these two field elements to random boolean elements. The random boolean elements are generated using a boolean PRG. During decryption, you recover the random masks which are field elements and then using the mapping, recover the corresponding boolean elements. This is done for every gate in the circuit. This reduces the degree of $\mathbf{Q}[j, \cdot, \cdot]$ to be 12 when j is the least significant bit of the associated wire key. Computing the mapping part can be implemented by a degree 5 polynomial. Thus, the maximum degree of all polynomials is at most 13.

Degree 5: We elaborate on the intuition presented in the technical overview to reduce the degree below 15. We consider a specific class of circuits $\mathcal{C}$. We just focus on constructing randomizing polynomials for $\mathcal{C}$.

We define $\mathcal{C}$ as follows: every circuit $C \in \mathcal{C}$ of output length $\mathbf{N}$ is of the form $C = (C_1, \dots, C_{\mathbf{N}})$, where:

- C_i outputs the i^{th} output bit of C,
- Depth of C_i is $c \cdot \log(\lambda)$, where c is a constant independent of C,
- $|C_i| = \mathrm{poly}(\lambda)$ for a fixed polynomial poly and,
- C_i for every $i \in [\mathbf{N}]$ have the same topology. That is, every C_i can be written as $\widehat{C}[V_i]$, where $\hat{C}$ is a circuit and V_i are the constants on a subset of its wires.

We show in Appendix A that it suffices to build sublinear FE for $\mathcal{C}$ to obtain iO. We now focus on obtaining randomizing polynomials for $\mathcal{C}$.

We start with the randomizing polynomials scheme of [AIK06] that we described earlier. Recall that it involved generating a garbled table for every gate in the circuit C. Moreover, the randomness to generate this garbled table is derived from an algebraic and a boolean PRG. We will see that we can emulate the role of algebraic PRG using just a boolean PRG. We make the following useful changes: let $C = (C_1, \dots, C_{\mathbf{N}})$ such that C_i outputs the i^{th} output bit of C. Let $w_1^i, \dots, w_{\mathsf{nw}}^i$ be the set of wires in C_i and $G_1^i, \dots, G_{\mathsf{ng}}^i$ be the set of gates in C_i.

- We invoke nw number of instantiations of boolean PRGs $\mathsf{bPRG}_1^w, \dots, \mathsf{bPRG}_{\mathsf{nw}}^w$ and $\mathsf{bPRG}_1^r, \dots, \mathsf{bPRG}_{\mathsf{nw}}^r$. All these PRGs have the same structure (i.e., same predicates are used) and have degree 5 over arbitrary field (with stretch 1.49). Pseudorandom generator bPRG_j^w is used to generate wire keys for wires $w_j^1, \dots, w_j^{\mathbf{N}}$. Since we are using a boolean PRG, we generate bits of key and then combine it into a field element using powers of two.

  Recall that earlier we were using an algebraic PRG of stretch strictly greater than quadratic. This is because the size of wire keys was proportional to exponential in depth, which could potentially be linear in the size of the circuit. However, since we are considering the specific circuit class $\mathcal{C}$, the depth of every circuit is $c \log(\lambda)$. And thus the size of the wire keys is independent of the security parameter. This is turn allows us to use just a PRG of stretch 1.49. Finally, bPRG_j^r is used to generate random masks for the wires $w_j^1, \dots, w_j^{\mathbf{N}}$.

- We now consider the [AIK06] randomizing polynomials associated with circuit C. As before, we substitute the variables associated with wire keys and random masks with the polynomials associated with the appropriate PRGs. The formal variables in the PRG polynomials are associated with the seed.

- The result of the above process is the encoding of C consisting of polynomials $p_1, \dots, p_N$ with variables associated with the seeds of PRGs. Note that the degree of these polynomials is still 15.

- We then observe that there are polynomials $q_1, \dots, q_T$ in seed variables such that $p_1, \dots, p_N$ can be rewritten in terms of $q_1, \dots, q_T$ and moreover, the degree of p_i in the new variables $\{q_i\}$ is 5. The advantage of doing this is that the polynomials $\{q_i\}$ can be evaluated during the encryption phase. The only thing we need to be wary of is the fact that T could be as big as $|C|$. If this is the case then the encryption complexity would be at least linear in $|C|$, which violate the sublinearity of the FE scheme. We show how to carefully pick $q_1, \dots, q_T$ such that T is sub-linear in $|C|$ and the above properties hold.

We now describe the polynomials $\{q_i\}_{i \in [T]}$. First, we set up some notation. We use the formal variables $\mathbf{Y}_w[j,k]$ to represent the k^{th} seed element corresponding to bPRG^w_j. We use the formal variables $\mathbf{Y}_r[j,k]$ to represent the k^{th} seed element corresponding to bPRG^r_j. For every seed index $k \in [n]$, $j_1, j_2 \in [\mathsf{nw}]$, such that j_1^{th} and j_2^{th} wires are input wires of a gate,

$$\mathsf{Z}_1\left[\vec{j} = (j_1, j_2), k, \vec{S}\right] = \prod_{\vec{S}=(d_1,d_2,d_3)\in\{0,1\}^3} \mathbf{Y}_w[j_1,k]^{d_1} \cdot \mathbf{Y}_r[j_1,k]^{d_2} \cdot \mathbf{Y}_r[j_2,k]^{d_3}$$

We also compute the following polynomials for every $k \in [n]$, $j_1, j_2, j_3 \in [\mathsf{nw}]$ such that there exists a gate G with j_1^{th} and j_2^{th} wires being input wires of G, j_3^{th} wire being output wire of G:

$$\mathsf{Z}_2\left[\vec{j} = (j_1, j_2, j_3), k, \vec{S}\right] = \prod_{\vec{S}=(d_1,d_2,d_3)\in\{0,1\}^3} \mathbf{Y}_r[j_1,k]^{d_1} \cdot \mathbf{Y}_r[j_2,k]^{d_2} \cdot \mathbf{Y}_r[j_3,k]^{d_3}$$

The polynomials $\{\mathsf{Z}_1[\cdot,\cdot,\cdot], \mathsf{Z}_2[\cdot,\cdot,\cdot]\}$ form the set of polynomials $\{q_i\}_{i\in[T]}$. We have the following two claims.

Claim 2. *We have T to be sublinear in $|C|$.*

Proof. By construction, the total number of all Z_1 and Z_2 polynomials is $\mathrm{O}(k^3 \cdot |\mathsf{nw}|)$. □

Claim 3. *Consider the polynomials $\{\mathbf{Q}\}$ defined in Section 4.5. The polynomial $\mathbf{Q}[j,\cdot,\cdot]$ has degree 5 when expressed in terms of polynomials $q_1, \dots, q_T$.*

Proof. There are two cases to consider: Let $\{\mathbf{Q}[\mathbf{j},\cdot,\cdot]\}$ be the set of polynomials associated with a gate G. Let $w^i_{j_1}$ and $w^i_{j_2}$ be the input wires of G for some i. Let $w^i_{j_3}$ be the output wire of G.

- $\mathbf{Q}[\mathbf{j},\cdot,\cdot]$, when index $\mathbf{j}$ does not correspond to the least significant bit: it comprises of terms which are multi-linear in variables $\{\mathbf{Y}_w[j_1,k_c], \mathbf{Y}_w[j_1,k_c], \mathbf{Y}_r[j_2,k_c]\}_{c\in[5]}$. Thus, every term can be re-written in terms of polynomials $\mathsf{Z}_1[\vec{j} = (j_1,j_2), k_1, \cdot], \dots, \mathsf{Z}_1[\vec{j} = (j_1,j_2), k_5, \cdot]$ and the degree of every such term is at most 5.

- $\mathbf{Q}[\mathbf{j},\cdot,\cdot]$, when index $\mathbf{j}$ corresponds to the least significant bit: it comprises of terms which is multi-linear in variables $\{\mathbf{Y}_r[j_1,k_c], \mathbf{Y}_r[j_2,k_c], \mathbf{Y}_r[j_3,k_c]\}_{c\in[5]}$. Thus, every term can be re-written in terms of polynomials $\mathsf{Z}_1[\vec{j} = (j_1,j_2,j_3), k_1, \cdot], \dots, \mathsf{Z}_1[\vec{j} = (j_1,j_2,j_3), k_5, \cdot]$ and the degree of every such term is at most 5.

□

We thus have the following theorem.

Theorem 9. *Assuming* 1.49PRGAssumption, *there exists a* (T, Φ)*-randomizing polynomials scheme with sub-linear randomness complexity. Furthermore, it is associated with* B*-composable linear decoder with* $B = \{0\}$ *(see Section 4.4). The degree of this scheme is 5.*

It is possible to weaken the PRG assumptions by considering PRGs of smaller stretch. See Remark 7 for more details.

The sampling of the randomness in this randomizing polynomials scheme is done by first sampling the seeds of all the PRGs and then pre-computing the polynomials $q_1, \ldots, q_T$ on the sampled seeds. The output is a pre-computed randomness that will be used in the computation of the randomizing polynomial.

Degree 7: The previous two randomizing polynomials were essentially a modification of the degree-15 randomizing polynomials construction. We consider a different approach to achieve degree-7 randomizing polynomials. As we will see later, we cannot just hope to just plug in this randomizing polynomials construction in the construction of sub-linear FE from PAFE for 7-linear maps. Instead, we make "non-black box" use of this construction to build sub-linear FE from 7-linear maps. We elaborate on this more, when we deal with the construction of sub-linear FE.

We now sketch the approach of obtaining degree-7 randomizing polynomials. Recall that we had remarked that the main bottleneck to reduce the degree in all these constructions is the fact that we have to deal with hiding which one of the entries in the garbled table needs to be decrypted. As we saw earlier, one way to do this, to have masks associated with wire keys that signal which one of the entries needs to be decrypted. Another way to do this is to emulate the functionality of inner product functional encryption. The rough idea is the following:

- For every wire w_i, there is an instantiation of inner product functional encryption associated with it. We denote the master secret key of this instantiation to be MSK_i.
- Compute garbled table T of gate G according to the Yao's garbling scheme. The only difference is that, instead of the output wire key, you encrypt an inner product functional encryption key associated with the output wire. More specifically, do the following. Let the output wire of G be w_k. Instead of encrypting $K_{w_k}^b$ associated with bit b, you encrypt a inner product functional key $sk_{k,b}$ w.r.t instantiation associated with w_k. We will see soon, the functionality associated with this key. Finally, the table T is encrypted twice w.r.t instantiations associated with wires w_i and w_j, where w_i and w_j are input wires of G. That is, $T' = \mathsf{IPFE.Enc}(\mathsf{MSK}_i, \mathsf{IPFE.Enc}(\mathsf{MSK}_j, T))$. The garbled table associated with G is set to be T'.
- The wire key corresponding to input wire w_i and bit b is an inner product functional key $sk_{k,b}$ w.r.t instantiation associated with w_i.

The evaluation proceeds as follows: Consider a gate G with input wires w_i, w_j and output wire w_k. Let T' be associated with G. First, decrypt T' using sk_{i,b_0} to obtain T^* – T^* essentially consists of two entries of T associated with bit b_0 encrypted under MSK_j. Note that the inner product FE scheme essentially hides which of the two entries of T remains after the decryption operation. In the next, T^* is decrypted using sk_{j,b_1} and the result is exactly one of the entries of T that corresponds to bits b_0 and b_1. Again, the inner product FE scheme hides which is the entry of T that remains. Inner product functional encryption is used just as a template for the above construction.

Implementation Idea: We now focus on implementing the above high level idea. Without loss of generality, for every gate, we can call one of its input wires to be first input wire and the other input wire to be second input wire. We use binary vectors $\beta_1^1, \beta_1^{1'}, \beta_2^1, \beta_2^{1'}$ of dimension poly(λ)

such that $\langle \beta_{\mathbf{i}}^1, \beta_{\mathbf{j}}^{1'} \rangle = 1$ if $\mathbf{i} = \mathbf{j}$, else the inner product outputs 0. Similarly we use vectors $\beta_1^2, \beta_1^{2'}, \beta_2^2, \beta_2^{2'}$ such that $\langle \beta_{\mathbf{i}}^2, \beta_{\mathbf{j}}^{2'} \rangle = 1$ if $\mathbf{i} = \mathbf{j}$, else the inner product outputs 0. A garbled table T_G associated with gate G corresponding to input wires w_i (first input wire), w_j (second input wire) and output wire w_k, is computed as follows:

$$T_G = \beta_1^1 \otimes \beta_1^2 \mathsf{CT}_{00} + \beta_1^1 \otimes \beta_2^2 \mathsf{CT}_{01} + \beta_2^1 \otimes \beta_1^2 \mathsf{CT}_{10} + \beta_2^1 \otimes \beta_2^2 \mathsf{CT}_{11}$$

We set $\mathsf{CT}_{b_0 b_1}$ as follows:

$$\mathsf{CT}_{b_0 b_1} = K_{w_i}^{b_0} + K_{w_j}^{b_1} + K_{w_k}^{G(b_0,b_1)} + r_{w_k} + \beta_v^{u'},$$

r_{w_k} is a random binary vector associated with wire w_k, $v = G(b_0, b_1) + 1$ and $u = 1$ if w_k is the first input wire to some gate, otherwise if w_k is the second input wire to some gate then $u = 2$. T_G is encoded at the topmost level of multilinear maps. Additionally, you also encode the following two elements at the topmost level.

$$E_1 = \langle r_{w_i}, \beta_1^1 \otimes \beta_1^2 + \beta_1^1 \otimes \beta_2^2 + \beta_2^1 \otimes \beta_1^2 + \beta_2^1 \otimes \beta_2^2 \rangle; \quad E_2 = \langle r_{w_j}, \beta_1^2 + \beta_2^2 \rangle$$

The input wire key corresponding to wire w and bit b is $K_w^b + r_w + \beta_{b+1}^{u'}$, where $u = 1$ if w is the first input wire of a gate, otherwise $u = 2$. Note that these wire keys are given in the plaintext form.

The decryption proceeds as follows: Consider a gate G with first input wire w_i, second input wire w_j and output wire w_k. Suppose we have obtained the wire keys $K_{w_i}^{b_0} + r_{w_i} + \beta_{b_0+1}^{1'}$ and $K_{w_j}^{b_1} + r_{w_j} + \beta_{b_1+1}^{2'}$. We first decrypt the encoding of T using $K_{w_i}^{b_0} + r_{w_i} + \beta_{b_0+1}^{1'}$ to obtain an encoding of $\langle r_{w_i}, \beta_1^1 \otimes \beta_1^2 + \cdots + \beta_2^1 \otimes \beta_2^2 \rangle + \beta_1^2 \mathsf{CT}_{b_0 0} + \beta_2^2 \mathsf{CT}_{b_0 1}$. The $\langle r_{w_i}, * \rangle$ is canceled out using encoding of E_1. In the next step, this is further decrypted using $K_{w_j}^{b_1} + r_{w_j} + \beta_{b_1+1}^{2'}$ to obtain $\langle r_{w_j}, \beta_1^2 + \beta_2^2 \rangle + \mathsf{CT}_{b_0 b_1}$. The $\langle r_{w_j}, * \rangle$ is canceled out using encoding of E_2 to obtain $\mathsf{CT}_{b_0 b_1}$, which is nothing but the wire key of w_k. Note that $\mathsf{CT}_{b_0 b_1}$ is a boolean vector and can be recovered from zero testing.

The above scheme as described is not necessarily secure since the adversary can manipulate the encodings in an illegal manner. However, the above scheme can be combined with the PAFE construction (Section 7) in a "non-black box" manner to obtain sub-linear secret key FE along the same lines as sketched in Section 5. In the construction of sublinear FE, the computation of the randomizing polynomials happens in the first slot. The second slot will be used to ensure that the encoding part is correctly computed. The resulting encodings at the top level will correspond to the randomizing polynomials scheme.

Comment on the Degree: Note that r_{w_i} has to be computed randomly and furthermore, it has to be boolean. Thus we need a boolean PRG and this contributes to degree 5. Furthermore, this is multiplied by β_*^1 and β_*^2 (which are sampled at random during the setup phase and the same is used throughout), which additionally contributes degree 2. This is the same case even for the wire keys K_w^b part. Thus, the total degree is 7.

4.5.3 Ensuring Homogeneity

We now show how to achieve a *homogenous* (T, Φ)-randomizing polynomials scheme $\mathsf{RP}' = (\mathsf{E}', \mathsf{D}')$ over $\mathbb{F}_{\mathbf{p}}$ for a circuit class $\mathcal{C}$ starting from any (T, Φ)-randomizing polynomials scheme $\mathsf{RP} = (\mathsf{E}, \mathsf{D})$ (not necessarily homogenous) over $\mathbb{F}_{\mathbf{p}}$ for a circuit class $\mathcal{C}$.

The transformation is as follows: on input a circuit C, instance x and randomness r;

- E' first executes $\mathsf{E}(C, \cdot; \cdot)$ to determine the polynomials $p_1, \ldots, p_N$. It then "homogenizes" these polynomials by introducing dummy variables as follows. Let $\mathbf{D} = \max_{i \in [N]} \deg[p_i]$.

Pick dummy variables $\vec{\mathbf{z}} = \mathbf{z}_1, \ldots, \mathbf{z}_{\mathbf{D}}$. For every $i \in [N]$ and for every monomial t in p_i, it determines if the degree of t is exactly $\mathbf{D}$ or not. If the degree of t is strictly than t then it replaces t with the monomial t', where $t' = t \cdot \prod_{j \in [\mathbf{D}-\deg[t]]} \mathbf{z}_j$. That is, t' is the product of t and the first $\mathbf{D} - \deg[t]$ dummy variables $\mathbf{z}_1, \ldots, \mathbf{z}_{\mathbf{D}}$. After this operation is performed, we call the new set of polynomials $p'_1, \ldots, p'_N$. We observe that these polynomials are homogenous and of degree $\mathbf{D}$.

- E' then evaluates $p'_i(x, \vec{\mathbf{z}} = \vec{1}; r)$ for every $i \in [N]$. That is, it evaluates p'_i by setting all dummy variables to 1. The result of evaluations of all the polynomials p'_i is the randomized encoding of (C, x).

The decoding algorithm D' is identical to D.

Remark 8. *The degree of polynomials p'_i, in the above transformation, is exactly the same as the degree of polynomial p_i.*

5 iO from Constant Degree PAFE

We show how to obtain iO from projective arithmetic FE *over* $\mathbb{F}_{\mathbf{p}}$. This transformation requires that the multi-key FE satisfies multiplicative overhead property in the encryption complexity. We achieve this in two steps:

1. Step I: Build (sub-exponentially secure) single key sub-linear FE for boolean circuits starting from (sub-exponentially secure) projective arithmetic FE satisfying multiplicative overhead property in the encryption complexity. We additionally also use a sub-exponentially secure homogenous (T, Φ)-randomizing polynomials with sublinear randomness complexity.
2. Step II: Obtain iO from single key sub-linear FE for boolean circuits. This is in turn obtained in the following steps:
   (a) Obtain (sub-exponentially secure) XiO from (sub-exponentially secure) sub-linear secret key FE. This is obtained along the same lines as the construction of XiO from compact secret key FE. We sketch this construction in Section A for completeness.
   (b) Obtain (sub-exponentially secure) compact *public key* FE from (sub-exponentially secure) XiO and sub-exponentially secure learning with errors. This was shown in Lin et al. [LPST16].
   (c) Obtain iO from sub-exponentially secure compact public key FE. This was shown by two works [AJ15, BV15].

Since Section A when combined with [LPST16] and [AJ15, BV15] show how to achieve Step II, we focus on Step I.

Sub-linear Secret Key FE from Arithmetic FE. We construct a single-key sub-linear secret key FE for boolean circuits starting from projective arithmetic FE scheme for degree $\mathbf{D}$. Consider the following tool.

Homogenous $(T = (V, E), \Phi)$-randomizing polynomials scheme with ε-sub-linear randomness complexity: Denote this by $\mathsf{RP} = (\mathsf{E}, \mathsf{D})$. We use denote the degree of RP to be $\mathbf{D}$ – this is same as the degree of polynomials supported by PAFE. Associated with this scheme is a B-composable linear decoder. In particular, we employ a scheme satisfying $B = \{0\}$.

Without loss of generality, we assume that the polynomials as part of the encoding algorithm are made homogenous as described in Section 4.5.3. This means that the polynomials also additionally take as input the dummy variables $\mathbf{z}$.

Let $\mathsf{PAFE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{ProjectDec})$ be a projective arithmetic functional encryption scheme satisfying B-correctness. Specifically, we consider the construction of PAFE from T-structured mmaps described in Section 7.

$\underline{\mathsf{Setup}(1^\lambda)}$: On input security parameter λ, it executes the setup of the underlying projective arithmetic FE scheme to obtain $\mathsf{PAFE.MSK} \leftarrow \mathsf{PAFE.Setup}(1^\lambda)$. It outputs secret key $\mathsf{MSK} = \mathsf{PAFE.MSK}$.

$\underline{\mathsf{KeyGen}(\mathsf{MSK}, C)}$: It takes as input master secret key MSK and circuit C. Denote $\mathsf{E}(C, \cdot\ ; \cdot) = (p_1(\cdot), \ldots, p_N(\cdot))$, where p_i are $(T, \overrightarrow{\phi_i})$-respecting polynomials. Compute keys for every polynomial p_i with respect to the arithmetic FE scheme. That is, $sk_{p_i} \leftarrow \mathsf{PAFE.KeyGen}(\mathsf{PAFE.MSK}, p_i)$. It outputs the functional key $sk_C = (sk_{p_1}, \ldots, sk_{p_N})$.

$\underline{\mathsf{Enc}(\mathsf{MSK}, x)}$: It takes as input the master secret key MSK and input x. It samples a string R uniformly at random of length ℓ_R from an appropriate domain[14]. Here, ℓ_R is the length of randomness used in algorithm E to encode a circuit of size $|C|$ and input length ℓ_x. It picks an all 1s vector $\mathbf{z}$ of length $\mathbf{D}$ - this corresponds to the dummy variables that are necessary to enforce homogeneity. It then computes $\mathsf{CT} \leftarrow \mathsf{PAFE.Enc}(\mathsf{MSK}, (x, R, \mathbf{z}))$. It outputs CT.

$\underline{\mathsf{ProjectDec}(sk_C, \mathsf{CT})}$: It takes as input functional key sk_C and ciphertext CT, it executes the decryption algorithm of the arithmetic FE scheme as follows: it computes $\iota_i \leftarrow \mathsf{PAFE.ProjectDec}(sk_{p_i}, \mathsf{CT})$. It then executes the B-composable linear decoder corresponding to RP homomorphically on the partial decrypted values $\{\iota_i\}$. Specifically,

- The number of steps executed by the composable linear decoder is T.
- In the i^{th} step, the decryptor executes the linear function $\mathfrak{f}_i$ (chosen by the B-composable linear decoder) homomorphically on the partial decrypted values $\{\iota_i\}$. More formally, it computes $v_i \leftarrow \mathsf{Recover}(\mathfrak{f}, \{\iota_i\})$ (see Remark 3 in Section 3.1).
- Let v_T be the value recovered after T steps.

Output v_T.

Encryption Complexity. From the multiplicative overhead in encryption complexity of the projective arithmetic FE scheme, we have the following calculation.

$$\begin{aligned} |\mathsf{Enc}(\mathsf{MSK}, x)| &= (|x| + |R| + |\mathbf{z}|) \cdot \mathrm{poly}(\lambda, \log(\mathbf{p})) \\ &\leq |C|^{\varepsilon} \cdot \mathrm{poly}(|x|, \lambda) \ (\because\ \varepsilon\text{-sublinear randomness complexity of } \mathsf{RP}) \end{aligned}$$

Thus, the encryption complexity is ε-sublinear in the size of circuit.

Correctness. Consider a functional key defined w.r.t to sublinear FE scheme associated with boolean circuit C. Consider a ciphertext of message x.

The functional key of C in turn consists of many projective arithmetic FE keys associated with polynomials p_i, for every $i \in [N]$. Here, polynomials p_i are such that $(p_1, \ldots, p_N) \leftarrow \mathsf{E}(C, \cdot\ ; \cdot)$. Furthermore, the ciphertext of x consists of encryption w.r.t projective arithmetic FE of (x, R), where R is a random string of appropriate length.

From the B-composable linear decoding property, we have that the B-composable linear decoder of the randomizing polynomials scheme with oracle access to the values $(p_1(x; R), \ldots, p_N(x; R))$

[14]The domain is dictated by the description of the randomizing polynomials scheme. For instance, in Theorem 8, the randomness is divided into two parts: one part is composed of elements from $\mathbb{F}_\mathbf{p}$ and acts as a seed to algebraic PRG. The other part is composed of bits and acts as a seed to boolean PRG.

produces linear functions $\mathfrak{f}_1, \ldots, \mathfrak{f}_N$ adaptively such that $\mathfrak{f}_i(p_1(x;R), \ldots, p_N(x;R))$ produces the value v_i. Furthermore, (from the correctness of randomizing polynomials scheme), v_T is indeed the correct output of $C(x)$.

Now, let the decryption of functional key of p_i on ciphertext of (x, R) yield the partial decrypted value ι_i. The decryptor of the sublinear FE scheme first executes $\mathsf{Recover}(\mathfrak{f}_1, \{\iota_i\})$ and let the result be v_1'. By the correctness of arithmetic FE scheme, we can argue that v_1' is indeed v_1. Proceeding along these lines, we can show that the output of $\mathsf{Recover}(\mathfrak{f}_T, \{\iota_i\})$ is v_T. As remarked earlier, v_T is nothing but $C(x)$.

5.1 Security

Theorem 10. *The scheme* FE *is a secure 1-key* ε*-sub-linear secret key functional encryption scheme for* $\mathcal{C}$ *assuming the security of* (T, Φ)*-randomizing polynomials scheme* RP *of degree* $\mathbf{D}$ *and projective arithmetic FE scheme* PAFE *for degree* $\mathbf{D}$ *polynomials.*

Proof. We present the hybrids below. In all the hybrids below, the adversary (selectively) requests for challenge message pairs $(x_1^0, x_1^1), \ldots, (x_\mu^0, x_\mu^1)$ and circuit C. In response, it receives the ciphertexts $(\mathsf{CT}_1, \ldots, \mathsf{CT}_\mu)$ and key sk_C from the challenger.

In the first hybrid, the challenge bit b is picked at random and the b^{th} entry in every message pair query is encrypted by the challenger. In the final hybrid, the challenger's message to the adversary is independent of the challenge bit.

$\underline{\mathsf{Hyb}_1}$: This corresponds to the real experiment when the challenge bit b is picked at random. The challenger first samples $\mathsf{MSK} \leftarrow \mathsf{Setup}(1^\lambda)$. It then computes the following:

- $\mathsf{CT}_i \leftarrow \mathsf{Enc}(\mathsf{MSK}, x_i^b)$
- $sk_C \leftarrow \mathsf{KeyGen}(\mathsf{MSK}, C)$

$\underline{\mathsf{Hyb}_{2,\mathbf{i},\mathbf{j}} \text{ for } \mathbf{i} \in [\mu], \mathbf{j} \in [N+1]}$: The challenger samples bit b at random. It computes the following:

- If $i < \mathbf{i}$, it computes $\mathsf{CT}_i \leftarrow \mathsf{Enc}(\mathsf{MSK}, x_i^0)$. If $i \geq \mathbf{i}$, it computes $\mathsf{CT}_i \leftarrow \mathsf{Enc}(\mathsf{MSK}, x_i^b)$.
- The functional key sk_C is computed by first determining the polynomials $p_1, \ldots, p_N$ such that $\mathsf{E}(C, \cdot; \cdot) = (p_1(\cdot; \cdot), \ldots, p_N(\cdot; \cdot))$. It computes $\theta_j = p_j(x_\mathbf{i}^b)$ for $j \in [N]$. Consider the following cases.
  - If $j < \mathbf{j}$, it computes $sk_{p_j} \leftarrow \mathsf{PAFE.sfKG}(\mathsf{MSK}, p_j, \theta_j)$. Here, $\mathsf{PAFE.sfKG}$ is the semi-functional key generation algorithm of PAFE.
  - It $j \geq \mathbf{j}$, it computes $sk_{p_j} \leftarrow \mathsf{PAFE.KeyGen}(\mathsf{MSK}, p_j)$.

  Set $sk_C = (sk_{p_1}, \ldots, sk_{p_N})$.

$\underline{\mathsf{Hyb}_{3,\mathbf{i}} \text{ for } \mathbf{i} \in [\mu]}$: The challenger samples bit b at random. It computes the following:

- If $i < \mathbf{i}$, it computes $\mathsf{CT}_i \leftarrow \mathsf{Enc}(\mathsf{MSK}, x_i^0)$. Else if $i > \mathbf{i}$, it computes $\mathsf{CT}_i \leftarrow \mathsf{Enc}(\mathsf{MSK}, x_i^b)$. If $i = \mathbf{i}$ then it computes $\mathsf{CT}_i \leftarrow \mathsf{PAFE.sfEnc}(\mathsf{MSK}, 1^{\ell_{\mathsf{inp}}})$. Here, $\mathsf{PAFE.sfEnc}$ is the semi-functional encryption algorithm of PAFE.
- The functional key sk_C is computed by first determining the polynomials $p_1, \ldots, p_N$ such that $\mathsf{E}(C, \cdot; \cdot) = (p_1(\cdot; \cdot), \ldots, p_N(\cdot; \cdot))$. It computes $\theta_j = p_j(x_\mathbf{i}^b)$ for $j \in [N]$. It finally computes $sk_{p_j} \leftarrow \mathsf{PAFE.sfKG}(\mathsf{MSK}, p_j, \theta_j)$ for every $j \in [N]$.

Set $sk_C = (sk_{p_1}, \ldots, sk_{p_N})$.

$\underline{\mathsf{Hyb}_{4,\mathbf{i}} \text{ for } \mathbf{i} \in [\mu]}$: The challenger samples bit b at random. It computes the following:

- If $i < \mathbf{i}$, it computes $\mathsf{CT}_i \leftarrow \mathsf{Enc}(\mathsf{MSK}, x_i^0)$. Else if $i > \mathbf{i}$, it computes $\mathsf{CT}_i \leftarrow \mathsf{Enc}(\mathsf{MSK}, x_i^b)$. If $i = \mathbf{i}$ then it computes $\mathsf{CT}_i \leftarrow \mathsf{PAFE.sfEnc}(\mathsf{MSK}, 1^{\ell_{\mathsf{inp}}})$.
- The functional key sk_C is computed by first determining the polynomials $p_1, \ldots, p_N$ such that $\mathsf{E}(C, \cdot; \cdot) = (p_1(\cdot; \cdot), \ldots, p_N(\cdot; \cdot))$. It computes $\theta_j = p_j(x_{\mathbf{i}}^0)$ for $j \in [N]$. It finally computes $sk_{p_j} \leftarrow \mathsf{PAFE.sfKG}(\mathsf{MSK}, p_j, \theta_j)$ for every $j \in [N]$.

Set $sk_C = (sk_{p_1}, \ldots, sk_{p_N})$.

$\underline{\mathsf{Hyb}_{5,\mathbf{i}} \text{ for } \mathbf{i} \in [\mu]}$: The challenger samples bit b at random. It computes the following:

- If $i \leq \mathbf{i}$, it computes $\mathsf{CT}_i \leftarrow \mathsf{Enc}(\mathsf{MSK}, x_i^0)$. Else if $i > \mathbf{i}$, it computes $\mathsf{CT}_i \leftarrow \mathsf{Enc}(\mathsf{MSK}, x_i^b)$.
- The functional key sk_C is computed by first determining the polynomials $p_1, \ldots, p_N$ such that $\mathsf{E}(C, \cdot; \cdot) = (p_1(\cdot; \cdot), \ldots, p_N(\cdot; \cdot))$. It computes $\theta_j = p_j(x_{\mathbf{i}}^0)$ for $j \in [N]$. It finally computes $sk_{p_j} \leftarrow \mathsf{PAFE.sfKG}(\mathsf{MSK}, p_j, \theta_j)$ for every $j \in [N]$.

Set $sk_C = (sk_{p_1}, \ldots, sk_{p_N})$.

$\underline{\mathsf{Hyb}_{6,\mathbf{i},\mathbf{j}} \text{ for } \mathbf{i} \in [\mu], \mathbf{j} \in [N+1]}$: The challenger samples bit b at random. It computes the following:

- If $i \leq \mathbf{i}$, it computes $\mathsf{CT}_i \leftarrow \mathsf{Enc}(\mathsf{MSK}, x_i^0)$. If $i > \mathbf{i}$, it computes $\mathsf{CT}_i \leftarrow \mathsf{Enc}(\mathsf{MSK}, x_i^b)$.
- The functional key sk_C is computed by first determining the polynomials $p_1, \ldots, p_N$ such that $\mathsf{E}(C, \cdot; \cdot) = (p_1(\cdot; \cdot), \ldots, p_N(\cdot; \cdot))$. It computes $\theta_j = p_j(x_{\mathbf{i}}^0)$ for $j \in [N]$. Consider the following cases.
  - It $j < \mathbf{j}$, it computes $sk_{p_j} \leftarrow \mathsf{PAFE.KeyGen}(\mathsf{MSK}, p_j)$.
  - If $j \geq \mathbf{j}$, it computes $sk_{p_j} \leftarrow \mathsf{PAFE.sfKG}(\mathsf{MSK}, p_j, \theta_j)$.

  Set $sk_C = (sk_{p_1}, \ldots, sk_{p_N})$.

$\underline{\mathsf{Hyb}_7}$: This corresponds to the real experiment when the challenge bit 0 is used. The challenger first samples $\mathsf{MSK} \leftarrow \mathsf{Setup}(1^\lambda)$. It then computes the following:

- $\mathsf{CT}_i \leftarrow \mathsf{Enc}(\mathsf{MSK}, x_i^0)$
- $sk_C \leftarrow \mathsf{KeyGen}(\mathsf{MSK}, C)$

Progression of Hybrids: We describe the progression of hybrids in Figure 3.

$$\begin{tikzcd}
\mathsf{Hyb}_1 \arrow[r, "\text{Claim 4}"] & \mathsf{Hyb}_{2,1,1} \arrow[r, "\text{Claim 5}"] & \cdots \arrow[r, "\text{Claim 5}"] & \mathsf{Hyb}_{2,1,N+1} \arrow[dll, "\text{Claim 6}"'] & \\
 & \mathsf{Hyb}_{3,1} \arrow[r, "\text{Claim 7}"] & \mathsf{Hyb}_{4,1} \arrow[r, "\text{Claim 8}"] & \mathsf{Hyb}_{5,1} \arrow[dll, "\text{Claim 9}"'] & \\
 & \mathsf{Hyb}_{6,1,1} \arrow[r, "\text{Claim 10}"] & \cdots \arrow[r, "\text{Claim 10}"] & \mathsf{Hyb}_{6,1,N+1} \arrow[dll, "\text{Claim 11}"'] & \\
 & \mathsf{Hyb}_{2,2,1} \arrow[r, "\text{Claim 5}"] & \cdots \arrow[r, "\text{Claim 10}"] & \mathsf{Hyb}_{6,\mu,N+1} \arrow[r, "\text{Claim 12}"] & \mathsf{Hyb}_7
\end{tikzcd}$$

Figure 3: We describe the progression of hybrids and also indicate the claims proving the indistinguishability of hybrids.

Indistinguishability of Hybrids. We prove the indistinguishability of the above hybrids.

Claim 4. *The hybrids* Hyb_1 *and* $\mathsf{Hyb}_{2,1,1}$ *are identical.*

The above claim follows from the fact that in $\mathsf{Hyb}_{2,1,1}$, the real encryption algorithm is used to encrypt the b^{th} message in every message pair query and real key generation algorithm is used to compute the functional keys.

Claim 5. *The hybrids* $\mathsf{Hyb}_{2,\mathbf{i},\mathbf{j}}$ *and* $\mathsf{Hyb}_{2,\mathbf{i},\mathbf{j}+1}$, *for every* $\mathbf{i} \in [\mu], j \in [N-1]$, *are computationally indistinguishable assuming that* PAFE *satisfies indistinguishability of semi-functional keys property (Definition 6).*

Proof. We can directly invoke the indistinguishability of semi-functional keys property[15] here: The reduction sends the message queries $\mathbf{x}_i$, where $\mathbf{x} = x_i^0$ if $i < \mathbf{i}$ and $\mathbf{x} = x_i^b$ else if $i \geq \mathbf{i}$. The reduction sends three types of function queries,

- For all $j < \mathbf{j}$, it sends (p_j, θ_j).
- For all $j > \mathbf{j}$, it sends p_j.
- It also sends $(p_{\mathbf{j}}, \theta)$

For queries in bullet 1, the challenger (of PAFE scheme) uses the semi-functional key generation algorithm. For queries in bullet 2, the challenger uses the honest key generation algorithm. For the query in bullet 3, it either uses semi-functional key generation or honest key generation. The ciphertexts for the message queries are generated as in the description of the scheme.

If there is a distinguisher that distinguishes the hybrids with non-negligible probability in the statement of the claim then the reduction can break the indistinguishability of semi-functional keys security with non-negligible probability. □

Claim 6. *The hybrids* $\mathsf{Hyb}_{2,\mathbf{i},N}$ *and* $\mathsf{Hyb}_{3,\mathbf{i}}$ *are computationally indistinguishble assuming that PAFE satisfies the indistinguishability of semi-functional ciphertexts property (Definition 7).*

Proof. We can directly invoke the indistinguishability of semi-functional ciphertexts property[16] here. The reduction sends the message queries $\{\mathbf{x}_i\}_{i \neq \mathbf{i}}$, where $\mathbf{x}_i = x_i^0$ if $i < \mathbf{i}$ and $\mathbf{x}_i = x_i^b$ if $i > \mathbf{i}$. The challenge message is set to be $x_{\mathbf{i}}^b$. The reduction sends the function queries (p_j, θ_j) for $j \in N$ to the challenger. The challenger answers the message queries (except the challenge message) using the (real) encryption algorithm. The challenger answers the challenge message $x_{\mathbf{i}}^b$ by either using the semi-functional encryption algorithm or the (real) encryption algorithm. The function queries are answered using the semi-functional key generation algorithm.

If the distinguisher distinguishes the hybrids with non-negligible probability in the statement of the claim then the reduction can break the indistinguishability of semi-functional ciphertexts property with non-negligible probability.

□

Claim 7. *The hybrids* $\mathsf{Hyb}_{3,\mathbf{i}}$ *and* $\mathsf{Hyb}_{4,\mathbf{i}}$, *for every* $\mathbf{i} \in [\mu]$, *are computationally indistinguishable assuming the security of* RP.

Proof. Suppose $\mathsf{Hyb}_{3,\mathbf{i}}$ and $\mathsf{Hyb}_{4,\mathbf{i}}$ is distinguishable and denote the PPT distinguisher by $\mathcal{A}$. We use $\mathcal{A}$ to construct a reduction $\mathcal{R}$. The reduction $\mathcal{R}$ gets (x_i^0, x_i^1), for every $i \in [\mu]$, and circuit C from $\mathcal{A}$. It picks bit b at random and forwards $(x_{\mathbf{i}}^b, x_{\mathbf{i}}^0)$ and C to the challenger of RP. Note that since $\mathcal{A}$ is a valid adversary, it holds that $C(x_{\mathbf{i}}^0) = C(x_{\mathbf{i}}^1)$. In response it

[15] As we will see later, in the instantiation of PAFE, this property will be proven based on Assumption #1 (Section 7.2.1).

[16] As we will see later, in the instantiation of PAFE, this property will be proven based on Assumption #2 (Section 7.2.1).

receives the randomized encoding, denoted by $\Theta = (\theta_1, \ldots, \theta_N)$. It then uses Θ to generate the semi-functional keys $sk_{p_1}, \ldots, sk_{p_N}$. It sets $sk_C = (sk_{p_1}, \ldots, sk_{p_N})$. It generates $\mathsf{CT}_{\mathbf{i}}$ using the semi-functional encryption algorithm. The rest of the ciphertexts are generated using the real encryption algorithm.

If $\mathcal{A}$ distinguishes $\mathsf{Hyb}_{3,\mathbf{i}}$ and $\mathsf{Hyb}_{4,\mathbf{i}}$ with noticeable probability then $\mathcal{R}$ breaks the security if RP with noticeable probability. $\square$

Claim 8. *The hybrids* $\mathsf{Hyb}_{4,\mathbf{i}}$ *and* $\mathsf{Hyb}_{5,\mathbf{i}}$*, for every* $\mathbf{i} \in [\mu]$*, are computationally indistinguishable assuming that* PAFE *satisfies the indistinguishability of semi-functional ciphertexts property (Definition 7).*

Proof. This is similar to proof of Claim 6. $\square$

Claim 9. *The hybrids* $\mathsf{Hyb}_{5,\mathbf{i}}$ *and* $\mathsf{Hyb}_{6,\mathbf{i},1}$*, for every* $\mathbf{i} \in [\mu]$*, are identical*

Claim 10. *The hybrids* $\mathsf{Hyb}_{6,\mathbf{i},\mathbf{j}}$ *and* $\mathsf{Hyb}_{6,\mathbf{i},\mathbf{j}+1}$*, for every* $\mathbf{i} \in [\mu], \mathbf{j} \in [N-1]$*, are computationally indistinguishable assuming that* PAFE *satisfies indistinguishability of semi-functional keys property (Definition 6).*

Proof. This is similar to the proof of Claim 5. $\square$

Claim 11. *The hybrids* $\mathsf{Hyb}_{6,\mathbf{i},N}$ *and* $\mathsf{Hyb}_{2,\mathbf{i}+1,1}$ *are identical.*

Claim 12. *The hybrids* $\mathsf{Hyb}_{6,\mu,N}$ *and* Hyb_7 *are identical.*

$\square$

Instantiation. We consider $(T, \vec{\phi})$-randomizing polynomials of degree 5 for NC^1 (Theorem 8). This can be based on assumptions 1.49PRGAssumption (Section 4.4). Further, we consider projective arithmetic FE for degree 5 polynomials (construction in Section 7). Putting together both these tools in the above construction we get,

Theorem 11. *Assuming projective arithmetic FE for degree 5 polynomials and 1.49PRGAssumption holds, there exists a 1-key sublinear secret key FE scheme for* NC^1*.*

Note that, as demonstrated in Section A, the existence of sub-exponentially secure sub-linear FE for NC^1 (and additional assumptions) implies indistinguishability obfuscation.

6 Slotted Encodings

We define the notion of slotted encodings: this concept can be thought of as abstraction of composite order multilinear maps. It allows for jointly encoding a vector of elements. Given the encodings of two vectors, using the addition and multiplication operations it is possible to either homomorphically add the vectors component-wise or multiply them component-wise.

To define this primitive, we first define the notion of structured asymmetric multilinear maps in Section 6.1. We show in Section 6.2 how to instantiate this form of structured asymmetric multilinear maps using current known instantiations of multilinear maps. Once we have armed ourselves with the definition of structured multilinear maps, we define the notion of slotted encodings (a special type of structured multilinear maps) in Section 6.3. In Section 6.5, then show how to realize slotted encodings using structured asymmetric multilinear maps for the constant degree[17] case.

[17]As we see later, this corresponds to the scenario where the structured multilinear maps is associated with constant number of bilinear maps.

6.1 Structured (Asymmetric) Multilinear Maps

We define the notion of structured asymmetric multilinear maps. It is associated with a binary tree T. Every node is associated with a group structure and additionally, every non leaf node is associated with a *noisy* bilinear map. Every element in this group structure has multiple noisy representations as in the case of recent multilinear map candidates [GGH13a, CLT13, GGH15].

Suppose nodes u and v are children of node w in tree T. And let the respective associated groups be $\mathbf{G}_u, \mathbf{G}_v$ and $\mathbf{G}_w$ respectively. Let e_{uv} be the bilinear map associated with node w. Then $e_{uv} : \mathbf{G}_u \times \mathbf{G}_v \to \mathbf{G}_w$.

Before we define structured multilinear maps we first put forward some notation about trees and also define some structural properties that will be useful later.

Notation About Trees: Consider a tree $T = (V, E)$, where V denotes the set of vertices and E denotes the set of edges. We are only interested in binary trees (every node has only two children) in this work.

1. We define the function $\mathbf{lc} : [V] \to \{0, 1\}$ such that $\mathbf{lc}(u) = 0$ if u is the left child of its parent, else $\mathbf{lc}(u) = 1$ if u is the right child of its parent.
2. We define $\mathbf{par} : [V] \to [V]$ such that $\mathbf{par}(u) = v$ if v is the parent of u.
3. $\mathbf{rt}(T) = w$ if the root of T is w.

We now define what we call *left ancestral path*, denoted by **LPth**. At a high level, left ancestral path of a node u is defined to consist of its ancestors $v_1, \ldots, v_{k+1}$ such that all v_is, except v_{k+1}, are left children of their parents.

Definition 17 (Left Ancestral Path)**.** *We define* $\mathbf{LPth} : [V] \to [V]^*$ *as follows:* $\mathbf{LPth}(u) = (v_1, \ldots, v_{k+1})$ *if the following conditions are satisfied:*

- *All nodes* $u, v_1, \ldots, v_k$ *(v_{k+1} is not included) are left children. That is,* $\mathbf{lc}(u) = 0 \wedge \mathbf{lc}(v_1) = 0 \wedge \cdots \wedge \mathbf{lc}(v_k) = 0$
- *Node* v_1 *is a parent of* u *and node* v_i *is a parent of node* v_{i-1} *for* $i > 1$*. That is,* $v_1 = \mathbf{par}(u) \wedge \cdots \wedge v_k = \mathbf{par}(v_{k-1}) \wedge v_{k+1} = \mathbf{par}(v_k)$
- *Either* v_{k+1} *is a right child of some node or it is the left child of the root. That is, either* $\mathbf{lc}(v_{k+1}) = 1$ *or* $(\mathbf{rt}(T) = \mathbf{par}(v_{k+1})) \wedge \mathbf{lc}(v_{k+1}) = 0$.

We refer the reader to Figure 4 for a simple example.

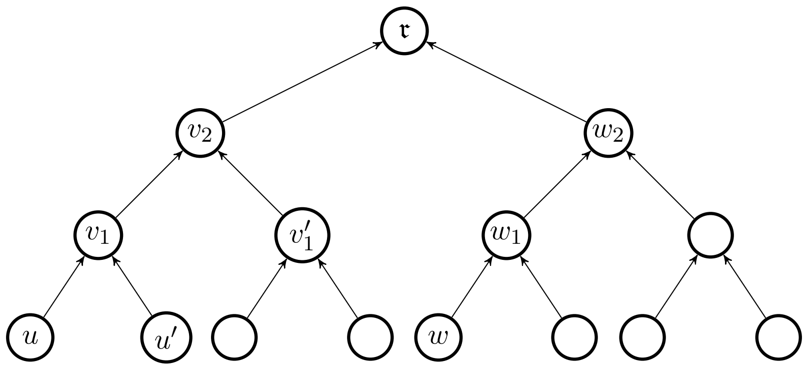

Figure 4: We have $\mathbf{LPth}(u) = (v_1, v_2)$; $\mathbf{LPth}(v_1) = (v_2)$; $\mathbf{LPth}(v_2) = \perp$; $\mathbf{LPth}(\mathfrak{r}) = \perp$; $\mathbf{LPth}(v_1') = \perp$; $\mathbf{LPth}(w) = (w_1, w_2)$.

Some observations about left ancestral path.

1. Consider a node u. We have $\mathbf{LPth}(u) = (v_1, \ldots, v_{k+1})$. Recall that v_1 is the parent of u. If $v_1 \neq \perp$ then $\mathbf{LPth}(v_1) = (v_2, \ldots, v_{k+1})$.
2. If u is a child of the root. We have $\mathbf{LPth}(u) = \perp$ irrespective of whether it is a left child or a right child.

Definition of Structured Multilinear Maps. A structured multilinear maps is defined by the tuple $\mathsf{SMMap} = (T = (V, E),\ \{\mathbf{G}_u\}_{u \in V})$ and associated with ring R, where:

- $T = (V, E)$ is a tree.
- $\mathbf{G}_u$ is a group structure associated with node $u \in V$. The order of the group is N.

The encoding of elements and operations performed on them are specified by the following algorithms:

- **Secret Key Generation,** $\mathsf{Gen}(1^\lambda)$: It outputs secret key $\mathbf{sk}$ and zero test parameters ztpp.
- **Encoding,** $\mathsf{Encode}(\mathbf{sk}, a, u \in V)$: In addition to secret key $\mathbf{sk}$, it takes as input $a \in R$ and a node $u \in V$. It outputs an encoding $[a]_{\mathbf{u}}$.
- **Add,** $[a]_{\mathbf{u}} + [b]_{\mathbf{u}} = [a + b]_{\mathbf{u}}$. Note that only elements corresponding to the same node in the tree can be added.
- **Multiply,** $[a]_{\mathbf{u}} \circ [b]_{\mathbf{v}} = [a \cdot b]_{\mathbf{w}}$. Here, w is the parent of u and v, i.e., $w = \mathbf{par}(u)$ and $w = \mathbf{par}(v)$.
- **Zero Test,** $\mathsf{ZeroTest}(\mathsf{ztpp}, [a]_{\mathfrak{r}})$: On input zero test parameters ztpp and an encoding $[a]_{\mathfrak{r}}$ at level $\mathfrak{r}$, where $\mathfrak{r} = \mathbf{rt}(T)$, output 0 if and only if $a = 0$.

We define degree of structured multilinear maps.

Definition 18 (Degree of SMMAP)**.** *Consider a structured multilinear maps scheme given by* $\mathsf{SMMap} = (T = (V, E),\ \{\mathbf{G}_u\}_{u \in V})$. *The* **degree** *of* SMMap *is defined recursively as follows. We assign degree to every node in the tree as follows:*

- *Degree of every leaf node u is 1.*
- *Consider a non leaf node w. Let u and v be its children. The degree of w is the sum of degree of u and degree of v.*

The degree of SMMap *is defined to be the degree of the root node.*

Remark 9. *If we restrict ourselves to only binary trees (which is the case in our work) and if d is the depth of the binary tree T then the degree of* SMMap, *associated with* $(T, \{\mathbf{G}_u\}_{u \in V})$ *is* 2^d.

Useful Notation: We employ the following notation that will be helpful later. Suppose $[v_1]_{\mathbf{i}}, \ldots, [v_m]_{\mathbf{i}}$ be a vector of encodings and let $\mathbf{v} = (v_1, \ldots, v_m) \in \mathbb{Z}_N^m$. Then, $[\mathbf{v}]_{\mathbf{i}}^m$ denotes $([v_1]_{\mathbf{i}}, \ldots, [v_m]_{\mathbf{i}})$. If the dimension of the vector is clear, we just drop m from the subscript and write $[\mathbf{v}]_{\mathbf{i}}$.

6.2 Instantiations of Structured Multilinear Maps

We can instantiate structured multilinear maps using the 'asymmetric' version of existing multilinear map candidates [GGH13a, CLT13]. For example, in asymmetric GGH, every encoding is associated with set S. Two encodings associated with the same set can be added. If there are two encodings associated with sets S_1 and S_2 respectively, then they can be paired if and only if $S_1 \cap S_2 = \emptyset$. The encoding at the final level is associated with the universe set, that is the union of all the sets.

To construct a structure multilinear map associated with $(T = (V, E), \phi)$, we can start with a universal set $U = \{1, \ldots, |V'|\}$, where $V' \subseteq V$ is the set of leaves in T. That is, there are as many elements as the number of leaves in V. We then design a bijection $\psi : U \to [V']$. An encoding is encoded at a leaf node u under the set $S_u = \{\psi^{-1}(u)\}$. For a non leaf node w, the encoding is performed under the set $S_w = S_u \cup S_v$, where u and v are the children of w.

6.3 Definition

A L-slotted encoding SEnc is a type of structured multilinear maps $\mathsf{SMMap} = (T = (V, E), \{\mathbf{G}_u\}_{u \in V})$ associated with ring R and is additionally parameterized by L. It consists of the following algorithms:

- **Secret Key Generation,** $\mathsf{Gen}(1^\lambda)$: It outputs secret key $\mathbf{sk}$ and zero test parameters ztpp.
- **Encoding,** $\mathsf{Encode}(\mathbf{sk}, a_1, \ldots, a_L, u \in V)$: In addition to secret key $\mathbf{sk}$, it takes as input $a_1, \ldots, a_L \in R$ and a node $u \in V$. If u is not the root node, it outputs an encoding $[a_1 | \cdots | a_L]_{\mathbf{u}}$. If u is indeed the root node, it outputs an encoding $\left[\sum_{i=1}^{L} a_i\right]_{\mathbf{u}}$.
- **Add,** $[a_1 | \cdots | a_L]_{\mathbf{u}} + [b_1 | \cdots | b_L]_{\mathbf{u}} = [a_1 + b_1 | \cdots | a_L + b_L]_{\mathbf{u}}$. Note that only elements corresponding to the same node in the tree can be added. Further, the elements in the vector are added component-wise. That is, if two encodings of $(a_1, \ldots, a_L)$ and $(b_1, \ldots, b_L)$ at node u are added then the result is an encoding of $(a_1 + b_1, \ldots, a_L + b_L)$ at the same node.
- **Multiply:** Suppose $w = \mathbf{par}(u)$ and $w = \mathbf{par}(v)$.

$$[a_1 | \cdots | a_L]_{\mathbf{u}} \circ [b_1 | \cdots | b_L]_{\mathbf{v}} = \begin{cases} [a_1 b_1 | \cdots | a_L b_L]_{\mathbf{w}} & \text{if } \mathbf{rt}(T) \neq w \\ \left[\sum\limits_{i=1}^{L} a_i b_i\right]_{\mathbf{w}} & \text{otherwise} \end{cases}$$

  The elements in the vectors are multiplied component-wise. That is, if you multiply two encodings of $(a_1, \ldots, a_L)$ at node u and $(b_1, \ldots, b_L)$ at node v then the result is an encoding of $(a_1 \cdot b_1, \ldots, a_L \cdot b_L)$ at node w.
- **Zero Test,** $\mathsf{ZeroTest}(\mathsf{ztpp}, [a]_{\mathfrak{r}})$: On input zero test parameters ztpp and an encoding $[a]_{\mathfrak{r}}$ at level $\mathfrak{r}$, where $\mathfrak{r} = \mathbf{rt}(T)$, output 0 if and only if $a = 0$.

Remark 10. *The degree of slotted encodings can be defined along the same lines as the degree of structured multilinear maps.*

6.4 Evaluation of Polynomials on Slotted Encodings

We consider the homomorphic evaluation of (T, ϕ)-respecting polynomials on slotted encodings. We first define evaluation of (T, ϕ)-respecting monomials on slotted encodings and then using this notion define evaluation of $(T, \overrightarrow{\phi})$-respecting polynomials on slotted encodings.

$\mathsf{HomEval}\left(t, \mathsf{SMMap}, \{\mathsf{E}_{1,u}\}_{u \in V}, \ldots, \{\mathsf{E}_{n,u}\}_{u \in V}\right)$: The input to this algorithm is (T, ϕ)-respecting monomial $t \in \mathbb{F}_{\mathbf{p}}[y_1, \ldots, y_n]$, slotted encoding scheme $\mathsf{SMMap} = (T = (V, E), \{\mathbf{G}_u\}_{u \in V})$ and slotted encodings $\mathsf{E}_{i,u}$, for every $i \in [n]$ and every $u \in V$, encoded under $\mathbf{G}_u$.

The evaluation proceeds recursively as follows: for every non leaf node $u \in V$, set $\widetilde{\mathsf{E}_u} = \mathsf{E}_{\phi(u),u}$. Consider the case when u is a non-leaf node and let v and w be the children of u. Compute encoding associated with node u as $\widetilde{\mathsf{E}_u} = \widetilde{\mathsf{E}_v} \circ \widetilde{\mathsf{E}_w}$. Let $\mathbf{rt}$ be the root of T. Output the encoding $\widetilde{\mathsf{E}_{\mathbf{rt}}}$ associated with $\mathbf{rt}$.

$\mathsf{HomEval}\,(p, \mathsf{SMMap}, \{\mathsf{E}_{1,u}\}_{u\in V}, \ldots, \{\mathsf{E}_{n,u}\}_{u\in V})$: The input to this algorithm is $(T, \overrightarrow{\phi})$-respecting polynomial $p \in \mathbb{F}_{\mathbf{p}}[y_1, \ldots, y_n]$, slotted encoding scheme $\mathsf{SMMap} = (T = (V, E),\ \{\mathbf{G}_u\}_{u\in V})$ and slotted encodings $\mathsf{E}_{i,u}$, for every $i \in [n]$ and every $u \in V$, encoded under $\mathbf{G}_u$.

Let $p = \sum_{i=1}^{n} c_i t_i$, for $c_i \in \mathbb{F}_{\mathbf{p}}$ and t_i is a (T, ϕ_i)-respecting monomial for every $i \in [n]$. The evaluation proceeds as follows: for every $i \in [n]$, execute $\widetilde{\mathsf{E}_{\mathbf{rt}}}^{(i)} \leftarrow \mathsf{HomEval}(t_i, \mathsf{SMMap}, \{\mathsf{E}_{1,u}\}_{u\in V}, \ldots, \{\mathsf{E}_{n,u}\}_{u\in V})$. Compute $\mathsf{E}_{\mathbf{rt}} = \sum\limits_{i=1}^{n} c_i \widetilde{\mathsf{E}_{\mathbf{rt}}}^{(i)}$. Output the encoding $\mathsf{E}_{\mathbf{rt}}$.

Remark 11. *Based on the current implementation of multilinear maps, given an encoding of an element $a \in \mathbb{F}_{\mathbf{p}}$, we don't know how to securely obtain encoding of $c \cdot a$ for some scalar $c \in \mathbb{F}_{\mathbf{p}}$ of our choice. But instead, we can still obtain encoding of $c \cdot a$, when c is small (for instance, polynomial in security parameter). This can achieved by adding encoding of a, c number of times.*

6.5 Implementation: Constant Degree Case

We show how to implement L-slotted encoding schemes associated with $(T = (V, E), \{\mathbf{G}_u\}_{u\in V})$ and ring R starting from any structured multilinear maps scheme associated with $(T = (V, E), \{\mathbf{G}_u\}_{u\in V})$ and ring R.

We begin by introducing some notation that will be helpful for presenting the transformation.

Notation. Considers vectors $\mu_1, \ldots, \mu_n \in \mathbb{Z}_p^m$. We denote the j^{th} element in μ_i to be μ_i^j. We denote by $\mu^* = \bigotimes_{i=1}^n \mu_i$ a vector in $\mathbb{Z}_p^{m^n}$ with the elements in μ^* indexed by $[m]^n$ such that the $(j_1, \ldots, j_n)^{th}$ element in μ^* is $\mu_1^{j_1} \cdots \mu_n^{j_n}$. That is, μ^* is the tensor product of $\{\mu_i\}_{i\in[n]}$.

We can define inner product operation for vectors of different dimension.

Definition 19. *Consider vectors of the form $\mu_1 \cdots \mu_n = \bigotimes_{i=1}^n \mu_i \in \mathbb{Z}_p^{m^n}$ and $\nu \in \mathbb{Z}_p^m$. We define $\langle \bigotimes_{i=1}^n \mu_i, \nu\rangle$ to be the vector $\bigotimes_{i=2}^n \mu_i \cdot \langle \mu_1, \nu\rangle \in \mathbb{Z}_p^{m^{n-1}}$.*

Remark 12. *According to the above definition, the notation $\langle \mu_1 \cdot \mu_2, \nu\rangle$ means $\mu_2 \cdot \langle \mu_1, \nu\rangle$ and the notation $\langle \mu_2 \cdot \mu_1, \nu\rangle$ means $\mu_1 \cdot \langle \mu_2, \nu\rangle$. Thus, $\langle \mu_1 \cdot \mu_2, \nu\rangle \neq \langle \mu_2 \cdot \mu_1, \nu\rangle$.*

We consider the following useful algorithm.

Dual Algorithm: We recall the dual algorithm Dual as defined in [BJK15]. It is a randomized algorithm that takes as input modulus N, dimension n[18] and outputs $(\overrightarrow{\mu} = (\mu_1, \ldots, \mu_n),\ \overrightarrow{\eta} = (\nu_1, \ldots, \nu_n))$ such that: (a) $\mu_i, \nu_i \in \mathbb{Z}_N^n$ (b) $\overrightarrow{\mu}$ is sampled at random conditioned on $\mu_1, \ldots, \mu_n$ being linearly independent, (c) $\langle \mu_i, \nu_j\rangle = 0$ if $j \neq i$ and, (d) $\langle \mu_i, \nu_i\rangle = 1$. Such vector spaces have been considered in the literature on bilinear maps [OT08, OT09, BJK15].

Implementation. Consider a structured multilinear maps scheme $\mathsf{SMMap} = (T = (V, E), \{\mathbf{G}_u\}_{u\in V})$. We construct a L-slotted encoding scheme as follows.

$\underline{\mathsf{Gen}(1^\lambda)}$: Let n be the number of the nodes in the tree. Compute $(\overrightarrow{\mu_i}, \overrightarrow{\nu_i}) \leftarrow \mathsf{Dual}(N, n \cdot L)$ for every $i \in [L]$. Here, $\overrightarrow{\mu_i} = \{\mu_u^i\}_{u\in V, i\in[L]}$ and $\overrightarrow{\nu_i} = \{\nu_u^i\}_{u\in V, i\in[L]}$ are such that: (a) $\mu_u^i, \nu_u^i \in \mathbb{Z}_N^{nL}$, (b) $\langle \mu_u^i, \nu_u^i\rangle = 1$ for every $u \in V$, (c) $\langle \mu_u^i, \nu_v^j\rangle = 0$ for every $u, v \in V$ and either $i \neq j$ or $u \neq v$.

For every $u \in V$ and $i \in [L]$, we assign β_u^i as follows:

- If u is the left child, i.e., $\mathbf{lc}(u) = 0$, set $\beta_u^i = \mu_u^i$.

[18]The dimension here indicates both the number of vectors as well as the number of components in the vectors.

- If u is the right child, then let v be the sibling of u. That is, $\mathbf{par}(u) = \mathbf{par}(v)$ and $\mathbf{lc}(v) = 0$. Assign $\beta_v^i = \nu_u^i$.
  [Note: if u and v are siblings, we have $\langle \beta_u^i, \beta_v^i \rangle = 1$.]

It also generates the setup of structured multilinear maps; $(\mathsf{SMMap.sk}, \mathsf{SMMap.ztpp}) \leftarrow \mathsf{SMMap.Gen}(1^\lambda)$. Set the secret key $\mathbf{sk} = (\mathsf{SMMap.MSK}, \{\beta_u^i\}_{u \in V, i \in [L]})$ and zero test parameters to be $\mathsf{ztpp} = \mathsf{SMMap.ztpp}$.

$\underline{\mathsf{Encode}(\mathbf{sk}, a_1, \dots, a_L, u \in V)}$: Let $\mathbf{LPth}(u) = (u_1, \dots, u_{k_u+1})$. Let $k = k_u$. Let $u_0 = u$. We set,

- If u is not the root,

$$[a_1|\cdots|a_L]_{\mathbf{u}} = \left[\sum_{i=1}^{L} \left(a_i \bigotimes_{j=0}^{k+1} \beta_{u_j}^i \right)\right]_{\mathbf{u}}^{(nL)^{k+2}}$$

- If u is the root,

$$[a_1|\cdots|a_L]_{\mathbf{u}} = \left[\sum_{i=1}^{L} a_i\right]_{\mathbf{u}}$$

That is, $[a_1|\cdots|a_L]_{\mathbf{u}}$ is generated as an output of $\mathsf{Encode}\left(\mathsf{SMMap.MSK}, \sum_{i=1}^{L} \left(a_i \bigotimes_{j=0}^{k+1} \beta_{u_j}^i\right), u\right)$.

<u>Addition</u>: Let $u \in V$ and let $\mathbf{LPth}(u) = (u_1, \dots, u_{k+1})$. Let $u_0 = u$.

$$\begin{aligned}
[a_1|\cdots|a_L]_{\mathbf{u}}^{(nL)^{k+1}} + [b_1|\cdots|b_L]_{\mathbf{u}}^{(nL)^{k+1}} &= \left[\sum_{i=1}^{L} \left(a_i \bigotimes_{j=0}^{k+1} \beta_{u_j}^i \right)\right]_{\mathbf{u}}^{(nL)^{k+2}} + \left[\sum_{i=1}^{L} \left(b_i \bigotimes_{j=0}^{k+1} \beta_{u_j}^i \right)\right]_{\mathbf{u}}^{(nL)^{k+2}} \\
&= \left[\sum_{i=1}^{L} \left((a_i + b_i) \bigotimes_{j=0}^{k+1} \beta_{u_j}^i \right)\right]_{\mathbf{u}}^{(nL)^{k+2}} \\
&= [a_1 + b_1|\cdots|a_L + b_L]_{\mathbf{u}}
\end{aligned}$$

<u>Multiplication</u>: Let $u, v, w \in V$ such that (i) $w = \mathbf{par}(u)$, (ii) $w = \mathbf{par}(v)$, (iii) $\mathbf{lc}(u) = 0$, (iv) $\mathbf{LPth}(u) = (u_1, \dots, u_{k+1})$, (v) $\mathbf{LPth}(v) = (v_1, \dots, v_{k+1})$. Let $u_0 = u$ and $v_0 = v$.

Note that by construction, $\langle \beta_u^i, \beta_v^j \rangle = 1$ if $i = j$, else $\langle \beta_u^i, \beta_v^j \rangle = 0$ if $i \neq j$.

Case $\mathbf{rt}(T) \neq w$: In this case, $v_1 = \perp$.

$$\begin{aligned}
[a_1|\cdots|a_L]_{\mathbf{u}} \circ [b_1|\cdots|b_L]_{\mathbf{v}} &= \left[\sum_{i=1}^{L} \left(a_i \bigotimes_{j=0}^{k+1} \beta_{u_j}^i \right)\right]_{\mathbf{u}}^{(nL)^{k+2}} \circ \left[\sum_{i=1}^{L} \left(b_i \beta_{v_0}^i \right)\right]_{\mathbf{v}}^{(nL)} \\
&= \left[\sum_{i=1}^{L} \left(\left(a_i b_i \bigotimes_{j=1}^{k+1} \beta_{u_j}^i \right) \cdot \langle \beta_{u_0}^i, \beta_{v_0}^i \rangle \right)\right]_{\mathbf{w}}^{(nL)^{k+1}} \\
&= [a_1 b_1|\cdots|a_L b_L]_{\mathbf{w}} \quad (\because \mathbf{LPth}(w) = (u_2, \dots, u_{k+1}))
\end{aligned}$$

Case $\mathbf{rt}(T) = w$: In this case, $u_1 = \perp$ and $v_1 = \perp$.

$$\begin{aligned}
[a_1|\cdots|a_L]_{\mathbf{u}} \circ [b_1|\cdots|b_L]_{\mathbf{v}} &= \left[\sum_{i=1}^{L}\left(a_i\beta_u^i\right)\right]_{\mathbf{u}}^{(nL)} \circ \left[\sum_{i=1}^{L}\left(b_i\beta_v^i\right)\right]_{\mathbf{v}}^{(nL)} \\
&= \left[\sum_{i=1}^{L} a_i b_i\right]_{\mathbf{w}}
\end{aligned}$$

Zero Testing, $\mathsf{ZeroTest}(\mathsf{ztpp}, [a]_{\mathfrak{r}})$: On input zero test parameters ztpp and encoding $[a]_{\mathfrak{r}}$, it executes the zero testing algorithm of SMMap scheme; $\mathsf{SMMap.ZeroTest}(\mathsf{SMMap.ztpp}, a)$. The result of $\mathsf{SMMap.ZeroTest}$ is output.

7 Projective Arithmetic FE from Slotted Encodings

We show how to construct projective arithmetic FE starting from the notion of slotted encodings defined in Section 6.3.

7.1 Construction

Consider a L-slotted encoding scheme SEnc, defined with respect to structured multilinear maps $\mathsf{SMMap} = (T = (V, E),\ \{\mathbf{G}_u\}_{u \in V})$ and is parameterized by L. We construct a multi-key secret key projective arithmetic functional encryption scheme PAFE for a function class $\mathcal{C} = \{\mathcal{C}_\lambda\}_{\lambda \in \mathbb{N}}$ as follows. Here, $\mathcal{C}_\lambda$ consists of functions with input length λ and output length poly(λ).

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

- It executes the secret key generation algorithm of the slotted encoding scheme to obtain $\mathbf{sk} \leftarrow \mathsf{Gen}(1^\lambda)$.
- Sample values $\alpha_{i,u} \in \mathbb{F}_{\mathbf{p}}$ for every $i \in [\ell_{\mathsf{inp}}], u \in V$ at random. We define ℓ_{inp} later. Denote $\overrightarrow{\alpha} = (\alpha_{i,u})_{i \in [\ell_{\mathsf{inp}}], u \in V}$.
- Sample a random value $S \in \mathbb{F}_{\mathbf{p}}$.

It outputs $\mathsf{MSK} = (\mathbf{sk}, \overrightarrow{\alpha}, S)$.

$\mathsf{KeyGen}(\mathsf{MSK}, p)$: It takes as input master secret key MSK and a polynomial $p \in \mathbb{F}_{\mathbf{p}}[y_1, \ldots, y_{\ell_{\mathsf{inp}}}]$ associated with an arithmetic circuit C. Let T a tree and $\vec{\phi} = (\phi_1, \ldots, \phi_K)$ with $\phi_i : [V] \rightarrow [\ell_{\mathsf{inp}}]$ be such that:

- $p = \sum_{j=1}^{K} c_i t_i$, where $c_i \in \mathbb{F}_{\mathbf{p}}$.
- t_i is a (T, ϕ_i)-respecting monomial in ℓ_{inp} variables.
- Thus, p is $(T, \vec{\phi})$-respecting.

Let δ_i be obtained by first assigning $\alpha_{\phi_i(u),u}$ to every leaf node u and then evaluating T[19]. That is, δ_i is the value obtained at the root of T. Assign $\Delta = \sum_{i=1}^{K} c_i \cdot \delta_i$.

Let $\mathbf{rt}$ be the root of T and let $\mathfrak{u}$ be its left child and $\mathfrak{v}$ be its right child. Compute $\mathsf{E}^C = \mathsf{Encode}\left(\mathbf{sk}, (0, \Delta \cdot S, p(0;0)), \mathfrak{u}\right)$ for every $i \in [n]$. Output $sk_C = (C, \mathsf{E}^C)$.

$\mathsf{Enc}(\mathsf{MSK}, x)$: It takes as input master secret key MSK and input $x \in \{0,1\}^{\ell_x}$. Let $\mathsf{inp} = x$ and $\ell_{\mathsf{inp}} = |x|$.

[19]Note that every non leaf node is treated as a multiplication gate.

It also samples an element $\gamma \in \mathbb{F}_{\mathbf{p}}$ at random. For every $i \in [\ell_{\mathsf{inp}}], u \in V$ and u is a leaf node, encode the tuple $(\mathsf{inp}_i, \gamma \cdot \alpha_{i,u}, 0)$ with inp_i denoting the i^{th} bit of inp, as follows: $\mathsf{E}^{\mathsf{inp}}_{i,u} = \mathsf{Encode}\left(\mathsf{MSK}, (\mathsf{inp}_i, \gamma \cdot \alpha_{i,u}, 0), u\right)$. Also encode γ^D under group $\mathbf{G}_{\mathfrak{v}}$, where $\mathfrak{v}$ is the right child of $\mathbf{rt}$: $\mathsf{E}_\gamma = \mathsf{Encode}(\mathsf{MSK}, (0, \gamma^D \cdot S^{-1}, 0), \mathfrak{v})$. Recall that D is the degree of homogeneity of RP.

Output the ciphertext $\mathsf{CT} = \left((\mathsf{E}_{i,u})_{i \in [\mathsf{inp}], u \in V}, \mathsf{E}_\gamma\right)$.

$\underline{\mathsf{ProjectDec}(sk_C, \mathsf{CT})}$: It takes as input functional key sk_C and ciphertext CT. It parses sk_C as (C, E^C) and CT as $\left((\mathsf{E}_{i,u})_{i \in [\mathsf{inp}], u \in V}, \mathsf{E}_\gamma\right)$. It executes the following:

- Compute $\mathsf{out}_1 = \mathsf{HomEval}(p, \mathsf{SMMap}, (\mathsf{E}_{i,u})_{i \in [\mathsf{inp}], u \in V})$.
- Compute $\mathsf{out}_2 = \mathsf{E}^C \circ \mathsf{E}_\gamma$.

Output the partial decrypted value $\iota = \mathsf{out}_1 - \mathsf{out}_2$.

$\underline{\mathsf{Recover}(c_1, \iota_1, \ldots, c_{\ell_f}, \iota_{\ell_f})}$: On input co-efficients $c_i \in \mathbb{F}_{\mathbf{p}}$, partial decrypted values ι_i, it first computes:

$$temp = c_1 \iota_1 + \cdots + c_{\ell_f} \iota_{\ell_f}$$

The addition carried out above corresponds to the addition associated with the slotted encodings scheme. Now, perform $\mathsf{ZeroTest}(\mathsf{ztpp}, temp)$ and output the result. Note that the output is either in $\{0, \ldots, B\}$ or its $\perp$.

(B, B')-Correctness. From the correctness of $\mathsf{HomEval}$ and slotted encodings, it follows that out_1 is an encoding of $(p(x), \gamma^D \cdot p(\{\alpha_{i,u}\}), 0)$. Further, out_2 is an encoding of $(0, \gamma^D \cdot p(\{\alpha_{i,u}\}), 0)$. Thus, the partial decrypted value $\mathsf{out}_1 - \mathsf{out}_2$ is an encoding of $(p(x), 0, 0)$.

With this observation, we remark that for many polynomials $p_1, \ldots, p_N$, the decryption of functional key of p_i on encryption of x yields as partial decrypted values, encodings of $(p_i(x), 0, 0)$. Thus, sum of all encodings of $(c_i \cdot p_i(x), 0, 0)$, where $c_i \in B'$ and $B' = \{0, \ldots, \mathrm{poly}(\lambda)\}$, yields a successful zero test query if and only if $\sum_{i=1}^{N} c_i p_i(x) = 0$.

We remark that if ztpp just contains parameters to test whether a top level encoding is zero or not, then the above construction only supports $B = \{0\}$. If it additionally contains encoding of 1, then we can set $B = \mathrm{poly}(\lambda)$.

Encryption Complexity: Multiplicative Overhead. We calculate the encryption complexity as follows.

$$|\mathsf{Enc}(\mathsf{MSK}, x)| \quad = \quad |x| \cdot (\text{Number of groups in } \mathsf{SMMap}) \cdot \mathrm{poly}(\lambda)$$

Thus, the above scheme satisfies the multiplicative overhead property.

7.2 Proof of Security

Semi-Functional Algorithms: We describe the semi-functional encryption and the key generation algorithms. We start with the semi-functional key generation algorithm.

$\underline{\mathsf{sfKG}(\mathsf{MSK}, p, \theta)}$: Parse MSK as $(\mathbf{sk}, \overrightarrow{\alpha})$. In addition, it takes as input a (T, ϕ)-respecting polynomial p and value θ to be hardwired in the third slot. Let $p = \sum_{j=1}^{K} c_j t_j$, where t_j is a (T, ϕ_j)-respecting monomial in ℓ_{inp} variables. Let δ_j be obtained by first assigning $\alpha_{\phi_j(u),u}$ to every leaf node u and then evaluating T. That is, δ_j is the value obtained at the root of T. Assign $\Delta = \sum_{j=1}^{K} c_{i,j} \cdot \delta_j$.

Let $\mathbf{rt}$ be the root of T and let $\mathfrak{u}$ be its left child and $\mathfrak{v}$ be its right child. Compute $\mathsf{E}^p = \mathsf{Encode}\left(\mathbf{sk}, (0, \Delta, p(0;0) - \theta), \mathfrak{u}\right)$ for every $i \in [n]$. Output $sk_C = (p, \mathsf{E}^p)$.

We now describe the semi-functional encryption algorithm.

<u>$\mathsf{sfEnc}(\mathsf{MSK}, 1^{\ell_{\mathsf{inp}}})$</u>: Parse MSK as $(\mathbf{sk}, \overrightarrow{\alpha})$. It samples an element $\gamma \in \mathbb{F}_{\mathbf{p}}$ at random.

For every $i \in [\ell_{\mathsf{inp}}], u \in V$ and u is a leaf node, encode the tuple $(0, \gamma \cdot \alpha_{i,u}, 0)$ as follows: $\mathsf{E}^{\mathsf{inp}}_{i,u} = \mathsf{Encode}\left(\mathsf{MSK}, (0, \gamma \cdot \alpha_{i,u}, 0), u\right)$. Also encode γ^D under group $\mathbf{G}_{\mathfrak{v}}$, where $\mathfrak{v}$ is the right child of $\mathbf{rt}$: $\mathsf{E}_\gamma = \mathsf{Encode}(\mathsf{MSK}, (0, \gamma^D, 1), \mathfrak{v})$. Recall that D is the degree of homogeneity of RP.

Output the ciphertext $\mathsf{CT} = \left((\mathsf{E}_{i,u})_{i \in [\mathsf{inp}], u \in V}, \mathsf{E}_\gamma\right)$.

We now prove the indistinguishability of semi-functional ciphertexts and indistinguishability of functional keys properties. Before that we state the assumptions on the slotted encodings upon which we prove the security of our scheme.

7.2.1 Assumptions

We define the following two assumptions.

<u>Assumption #1</u>: For all (i) inputs $\mathbf{x} = (x_1, \ldots, x_\mu) \in \{0,1\}^{\mu \cdot \ell_x}$, (ii) polynomials $p \in \mathbb{F}_{\mathbf{p}}[y_1, \ldots, y_n], \mathbf{q} = (q_1, \ldots, q_N) \in \mathbb{F}_{\mathbf{p}}[y_1, \ldots, y_n]^N$ be (T, ϕ)-respecting polynomials, (iii) subset $I \subseteq [n]$ and finally, (iv) values $\theta \in \mathbb{F}_{\mathbf{p}}, \Theta = (\theta_i)_{i \in I} \in \mathbb{F}_{\mathbf{p}}^{|I|}$ and for every sufficiently large $\lambda \in \mathbb{N}$, the following holds:

$$\{\ \mathsf{KeyGen}(\mathsf{MSK}, p, \theta),\ aux[\mathbf{x}, \mathbf{q}, I, \Theta]\ \} \ \cong_c\ \{\ \mathsf{sfKG}(\mathsf{MSK}, p),\ aux[\mathbf{x}, \mathbf{q}, I, \Theta]\}$$

- $\mathsf{MSK} \leftarrow \mathsf{Setup}(1^\lambda)$
- $aux[\mathbf{x}, \mathbf{q}, I, \Theta] = (\mathsf{CT}_1, \ldots, \mathsf{CT}_\mu, sk_1, \ldots, sk_N)$ *consists of two components:*
  1. *For every $i \in [n]$, compute $\mathsf{CT}_i \leftarrow \mathsf{Enc}(\mathsf{MSK}, x_i)$.*
  2. *For every $i \in [N]$ and $i \in I$, compute $sk_i \leftarrow \mathsf{sfKG}(\mathsf{MSK}, q_i, \theta_i)$. Else if $i \notin I$, compute $sk_i \leftarrow \mathsf{KeyGen}(\mathsf{MSK}, q_i)$.*

<u>Assumption #2</u>: For all (i) inputs $x^* \in \{0,1\}^{\ell_x}, \mathbf{x} = (x_1, \ldots, x_\mu) \in \{0,1\}^{\mu \cdot \ell_x}$, (ii) polynomials $\mathbf{q} = (q_1, \ldots, q_N) \in \mathbb{F}_{\mathbf{p}}[y_1, \ldots, y_n]^N$ be (T, ϕ)-respecting polynomials and finally, (iii) values $\Theta = (\theta_i)_{i \in [N]}$ and for every sufficiently large $\lambda \in \mathbb{N}$, the following holds:

$$\left\{\ \mathsf{sfEnc}(\mathsf{MSK}, 1^{\ell_{\mathsf{inp}}}),\ aux[\mathbf{x}, \mathbf{q}, \Theta]\ \right\} \ \cong_c\ \{\ \mathsf{Enc}(\mathsf{MSK}, x^*),\ aux[\mathbf{x}, \mathbf{q}, \Theta]\}$$

- $\mathsf{MSK} \leftarrow \mathsf{Setup}(1^\lambda)$
- $aux[\mathbf{x}, \mathbf{q}, \Theta] = (\mathsf{CT}_1, \ldots, \mathsf{CT}_\mu, sk_1, \ldots, sk_N)$ *is computed in the following way:*
  1. *For every $i \in [n]$, compute $\mathsf{CT}_i \leftarrow \mathsf{Enc}(\mathsf{MSK}, x_i)$.*
  2. *For every $i \in [N]$ $\theta_i = q_i(x^*)$.*
  3. *For every $i \in [N]$, compute $sk_i \leftarrow \mathsf{sfKG}(\mathsf{MSK}, q_i, \theta_i)$.*

The following two theorems directly follow from the above two assumptions.

Theorem 12. *The scheme* PAFE *satisfies indistinguishability of semi-functional keys under Assumption #1.*

Theorem 13. *The scheme* PAFE *satisfies indistinguishability of semi-functional ciphertexts under Assumption #2.*

From the above two theorems, we have the following theorem.

Theorem 14. *The* PAFE *satisfies semi-functional security under Assumptions #1 and #2.*

8 Justifying Assumptions in Ideal MMap Model

We justify the security of both the assumptions in the ideal multilinear map model. We first recall the definition of the ideal multilinear map model.

8.1 Ideal Multilinear Map Model

We describe the ideal multilinear model [BR14, BGK+14] tailored to the asymmetric setting.

The model is parameterized by structured multilinear maps $\mathsf{SMMap} = (T = (V, E), \{\mathbf{G}_u\}_{u \in V})$, where $\mathbf{G}_u$ is a group of order $\mathbf{p}$. The adversary in this model has access to an oracle $\mathcal{M}$. Initially, the adversary is handed out *handles* (sampled uniformly at random) instead of being handed out actual encodings. A handle is an element in a ring R of order $\mathbf{p}$. The oracle $\mathcal{M}$ maintains a list L consisting of tuples $(e, \mathbf{Y}[e], u)$, where e is the handle issued, $\mathbf{Y}[e]$ is the formal expression associated with e and e is associated with $\mathbf{G}_u$.

The adversary is allowed to submit the following types of queries to the oracle:

- *Addition/ Subtraction*: The adversary submits (e_1, u_1) and (e_2, u_2) along with the operation '+'(or '-') to the oracle. If there is no tuple associated with either e_1 or e_2, the oracle sends $\perp$ back to the adversary. Otherwise, it locates the tuples (e_1, p_{e_1}, u_1) and (e_2, p_{e_2}, u_2). If $u_1 \neq u_2$, it sends $\perp$ to adversary. Else, it creates a new handle e' (sampled uniformly at random from R) and appends $(e', p_{e_1} + p_{e_2}, u_1)$ (or $(e', p_{e_1} - p_{e_2}, u_1)$) to the list. The oracle sends e' to the adversary.
- *Multiplication*: The adversary submits (e_1, u_1) and (e_2, u_2) to the oracle. If there is no tuple associated with either e_1 or e_2, the oracle sends $\perp$ back to the adversary. Otherwise, it locates the tuples (e_1, p_{e_1}, u_1) and (e_2, p_{e_2}, u_2). If u_1 and u_2 are not siblings in the tree T, it sends $\perp$ to the adversary. Otherwise, let w be the parent of u_1 and u_2. It creates a new handle e' (sampled uniformly at random from R) and appends $(e', p_{e_1} * p_{e_2}, u_1)$ to the list.
- *Zero Test*: The adversary submits element e to the oracle. The oracle tries to locate the tuple (e, p_e, u). If $u \neq \mathfrak{r}$, where $\mathfrak{r}$ is the root of T, then the oracle outputs $\perp$. Otherwise, it checks if $p_e = 0$ (as a formal polynomial) and if so, it sends 0 to the adversary, else it sends 1.

Remark 13. *The above model can be strengthened to also allow for multiplication queries, where the adversary can submit (e_1, u_1) and (e_2, u_2) such that u_1 and u_2 are not necessarily siblings. In fact the instantiations of structured multilinear maps (see Section 6.1) does allow for such types of operations. We do not describe the security analysis in the extended model in this draft because as we will see later, the analysis for the current model is already quite involved. However, we note that our analysis for the above model can be extended to the general setting.*

SIMPLIFICATION OF NOTATION. An equivalent version of the above model is the setting where the adversary submits polynomials instead of handles. For instance, for an addition query, it submits two tuples (p_1, u_1) and (p_2, u_2) where p_1 and p_2 are polynomials (in variables associated to initial handles). As before, the oracle maintains a list comprising of tuples of the form (p, v). Similarly, instead of handing out handles in the beginning, the oracle hands out polynomials.

Useful Lemma. We define the following lemma which has been used in the literature to argue security in generic bilinear maps and multilinear maps.

Lemma 1 (Schwartz-Zippel-DeMillo-Lipton for Rational Polynomials)**.** *Consider two rational polynomials $h_1 = \frac{p_1}{q_1}$ and $h_2 = \frac{p_2}{q_2}$, where $p_1, q_1, p_2, q_2 \in \mathbb{F}_{\mathbf{p}}[y_1, \ldots, y_n]$. Suppose the maximum degree of p_1, q_1, p_2, q_2 is at most $\mathbf{D}$. If $\mathbf{p} = 2^\lambda$ and $\mathbf{D} = \mathrm{poly}(\lambda)$ then,*

$$\mathsf{Pr}_{y_1,\ldots,y_n \xleftarrow{\$} \mathbb{F}_{\mathbf{p}}}[h_1(y_1, \ldots, y_n) = h_2(y_1, \ldots, y_n) \ : \ h_1 \neq h_2] \leq \mathsf{negl}(\lambda)$$

Proof Sketch. We can rewrite $h_1 \neq h_2$ to be equivalent to $p_1 \cdot q_2 \neq p_2 \cdot q_1$. That is, $p^* = p_1 \cdot q_2 - p_2 \cdot q_1 \neq 0$. We bound the probability that the evaluation of p^* on a random point is 0. This holds from Schwartz-Zippel-DeMillo-Lipton for (standard) polynomials. Using this, we argue that h_1 and h_2 agree with each other only with negligible probability. The only case we need to be careful is when q_1 and q_2 are zero polynomials in which case $p^* = 0$ but then h_1 and h_2 won't agree with each other. Thus, we need to bound the probability that q_1 and q_2 are zero polynomials. Again from Schwartz-Zippel-DeMillo-Lipton for (standard) polynomials, q_1 and q_2 are not zero polynomials with overwhelming probability. $\square$

8.2 Slotted Encodings: Proof in Ideal MMap Model

We first state a property associated with slotted encodings and then we prove our construction of slotted encodings (Section 6.5) satisfies this property in the ideal multilinear map model.

At a high level, this property says that the adversary when given slotted encodings generated as in Section 6.5 can only do "component-wise" operations on the elements. That is, given encodings of $(a_1, \ldots, a_L)$ and $(b_1, \ldots, b_L)$, the adversary can only obtain $(a_1 + b_1, \ldots, a_L + b_L)$ if they both are encoded under the same node or obtain $(a_1 \cdot b_1, \ldots, a_L \cdot b_L)$ if they both are encoded under sibling nodes.

Slot Preservation Property: *Consider the L-slotted encoding scheme* SEnc *defined with respect to structured multilinear maps* $\mathsf{SMMap} = (T = (V, E),\ \{\mathbf{G}_u\}_{u \in V})$ *(Section 6). Consider an input distribution $\mathcal{D}$ defined over the space $\mathbb{F}_{\mathbf{p}}^K$, where $K = \mathrm{poly}(\lambda)$. For any sufficiently large security parameter $\lambda \in \mathbb{N}$, for any PPT adversary $\mathcal{A}$, we define the following experiment:*

$\underline{\mathsf{Expt}^{\mathcal{A}}(1^\lambda)}$:

1. *Adversary first chooses the groups under which the K elements need to be encoded;* $(u_1, \ldots, u_K) \leftarrow \mathcal{A}(1^\lambda)$
2. $\{(\mathbf{z}_1^i, \ldots, \mathbf{z}_K^i)\}_{i \in [L]} \xleftarrow{\$} \mathcal{D}(1^\lambda)$
3. *Generate the secret key* $\mathbf{sk} \leftarrow \mathsf{Gen}(1^\lambda)$
4. *Generate the encodings* $\mathsf{E}_j = \left[\mathbf{z}_j^1 | \cdots | \mathbf{z}_j^K\right]_{\mathbf{u_j}} \leftarrow \mathsf{Encode}(\mathbf{sk}, \mathbf{z}_j^1, \ldots, \mathbf{z}_j^K, u_j)$*. The adversary is given* $\{\mathsf{E}_j\}_{j \in [K]}$.
5. *Adversary outputs* E^* *encoded at the final level.*
6. *Output 1 if* $\mathsf{ZeroTest}(\mathsf{E}^*) = 0$.

Let $\mathfrak{r}$ be the root of the tree T. We have,

$$\mathsf{Pr}\left[\exists C:\ 0 \leftarrow \mathsf{ZeroTest}(\mathsf{Encode}(\mathbf{sk}, C(\{\mathbf{z}_j^1\})), \ldots, C(\{\mathbf{z}_j^K\})), \mathfrak{r}) \mid 1 \xleftarrow{\$} \mathsf{Expt}^{\mathcal{A}}(1^\lambda)\right]$$
$$\geq 1 - \mathsf{negl}(\lambda),$$

where negl *is a negligible function.*

Remark 14. *If $\mathcal{A}$ is an ideal multilinear map adversary then we say that the slotted encoding scheme* SEnc *satisfies slot preservation property in the ideal multilinear map model.*

8.2.1 Slot Preservation Property in Ideal MMap Model

We show the following,

Theorem 15. *L-slotted encodings implemented in Section 6.2 from prime order multilinear maps satisfies the slot preservation property in the ideal mmap model.*

We associate with the vector β_u^i (described in Section 6.2) the variables $\mathbf{Y}[\beta_u^i, j]$ for the j^{th} element of β_u^i, with $u \in V$ and $i \in [L]$. Depending on whether u is a left child or not, $\mathbf{Y}[\beta_u^i, j]$ is either a variable or a polynomial:

- If u is a left child, i.e., $\mathbf{lc}(u) = 0$ then $\mathbf{Y}[\beta_u^i, j]$ is a variable.
- If u is a right child then $\mathbf{Y}[\beta_u^i, j]$ is a polynomial. Let v be the sibling of u. Then, $\mathbf{Y}[\beta_u^i, j]$ is a polynomial in the variables $\{\mathbf{Y}[\beta_v^{i'}, j']\}_{i' \in [L], j' \in [n \cdot L]}$ defined by the following conditions:
  - If $i' = i$, $\sum_{j'=1}^{n \cdot L} \mathbf{Y}[\beta_u^i, j'] \cdot \mathbf{Y}[\beta_v^i, j'] = 1$
  - If $i' \neq i$, $\sum_{j'=1}^{n \cdot L} \mathbf{Y}[\beta_u^i, j'] \cdot \mathbf{Y}[\beta_v^i, j'] = 0$

  Note that for the case when u is a right child, $\mathbf{Y}[\beta_u^i, j]$ could potentially be a rational polynomial. And the only variables it depends on are the ones associated with its sibling.

We associate with $\mathbf{z}_c^i$ (as in the description of slot preservation property) the variables $\mathbf{Y}[\mathbf{z}_c^i]$ for every $c \in [K]$ and $i \in [L]$.

We introduce the following notation:

Definition 20 (Zero Test Circuits/ Polynomials)**.** *Consider a circuit C (resp., polynomial p) computed by the ideal mmap adversary that leads to a successful zero test query. We call C (resp., p) a zero test circuit (resp., zero test polynomial).*

We also recall a standard definition:

Definition 21 (Sub-circuit/ Sub-polynomial)**.** *A circuit C' is said to be a sub-circuit of another circuit C if C' is a sub-graph of C and if C' is not empty. In particular, if C and C' are written as polynomials p and p' respectively, then the co-efficient of p' in the computation of p is non zero. Here, p' will be called sub-polynomial of p.*

Definition 22 (Polynomial computed at a Node)**.** *A polynomial p is said to be computed at node u by the adversary if (p, u) is a valid tuple in the list maintained by the oracle.*

We prove the following lemma that proves the security of slotted preservation property in the ideal mmap model.

Theorem 16. *Consider a node $u \in V$. Consider a polynomial p_u created by the adversary at node u. Then one of the following conditions will hold:*

1. *Let p^* be a zero-test polynomial such that p_u is a sub-polynomial of p^*. Then,*
   - *All the sub-polynomials of p^* computed at node u are of the following form: Let $u_0 = u$, for every $\vec{j} = (j_0, \ldots, j_{k+1})$, $j_0, \ldots, j_{k+1} \in [n \cdot L]$,*

     $$p_u^{\vec{j}} = \sum_{i=1}^{L} C_u(\{\mathbf{Y}[\mathbf{z}_c^i]\}) \cdot \prod_{l=0}^{k+1} \mathbf{Y}[\beta_{u_\ell}^i, j_l]$$

     for $\mathbf{LPth}(u) = (u_1, \ldots, u_{k+1})$ and $u_0 = u$.
   - *There exists $\vec{j^*} \in [n \cdot L]^{k+2}$ such that $p_u = p_u^{\vec{j^*}}$.*
2. *p_u is not of the above form then p_u is not a sub-polynomial of any zero test polynomial.*

Proof. Consider a zero test polynomial p^* computed by adversary. We prove this by induction.

Base Case: Initially, the adversary is only given handles associated with polynomials of the form $\mathbf{Y}[\mathbf{z}_c^i] \cdot \prod_{l=0}^{k+1} \mathbf{Y}[\beta_{u_\ell}^i, j_l]$ associated with leaf node node u.

Induction Hypothesis: Consider a non leaf node w and let u and v be its left child and right child respectively. Let us assume the following:

- All the sub-polynomials computed at u is of the following form: Let $u_0 = u$, for every $\vec{j} = (j_0, \ldots, j_{k+1})$, $j_0, \ldots, j_{k+1} \in [n \cdot L]$,

$$p_u^{\vec{j}} = \sum_{i=1}^{L} C_u(\{\mathbf{Y}[\mathbf{z}_c^i]\}) \cdot \prod_{l=0}^{k+1} \mathbf{Y}[\beta_{u_0}^i, j_l]$$

for $\mathbf{LPth}(u) = (u_1, \ldots, u_k)$.

- All the sub-polynomials computed at v is of the following form: Let $v_0 = v$, for every $\vec{j} = (j_0, \ldots, j_{k+1})$, $j_0, \ldots, j_{k+1} \in [n \cdot L]$,

$$p_v^{\vec{j}} = \sum_{i=1}^{L} C_v(\{\mathbf{Y}[\mathbf{z}_c^i]\}) \cdot \prod_{l=0}^{k+1} \mathbf{Y}[\beta_{v_0}^i, j_l]$$

for $\mathbf{LPth}(v) = (v_1, \ldots, v_k)$.

Remark 15. *Note that when v is the right child then $\mathbf{LPth}(v) = \perp$. That is, $\mathbf{Y}[\beta_{v_l}^i, j_l] = 1$ for $l \in \{1, \ldots, k\}$.*

We claim the following:

Lemma 2. *The only sub-polynomials of p^* computed at node w are of the following form: for every $\vec{j_i} = (i, j_1, \ldots, j_{k+1})$, where $i, j_1, \ldots, j_{k+1} \in [n \cdot L]$,*

$$p_w^{\vec{j}} = \sum_{i=1}^{n \cdot L} p_u^{\vec{j_i}} \cdot p_v^{\vec{j_i}}$$

To prove the above lemma, we first prove the following useful claim.

Claim 13. *Consider a polynomial q in variables $\mathbf{y}_1, \ldots, \mathbf{y}_m$. Consider $I \subseteq [m]$. Let $\mathbf{y}_{\overline{I}} = \{\mathbf{y}_i\}_{i \notin I}$ and $\mathbf{y}_U = \{\mathbf{y}_1, \ldots, \mathbf{y}_m\}$. Suppose the following holds:*

- *q can be written as $q = q_1(q_2(\mathbf{y}_U), \mathbf{y}_{\overline{I}})$. That is, $q_2(\mathbf{y}_U)$ is a polynomial in variables in the set $\mathbf{y}_U$ and q_1 is a polynomial in $q_2(\mathbf{y}_U)$ and variables in $\mathbf{y}_{\overline{I}}$.*
- *The co-efficient of $q_2(\mathbf{y}_U)$ is non-zero in the polynomial q_1. That is, q_2 is the sub-polynomial of q_1.*
- *The polynomial q_2 is non-zero in the variables in the set $\mathbf{y}_I$. That is, at least one term in the expansion of q_2 contains at least one variable in the set $\mathbf{y}_I$.*

Then, the polynomial q is non-zero polynomial.

Proof. Let t be a term in q_2 that contains one of the variables in $\mathbf{y}_I$. Since term cannot be canceled by any of the variables in $\mathbf{y}_{\overline{I}}$, q_1 will contain a term of the form $t \cdot t'$, where t' is a term in variables in $\mathbf{y}_{\overline{I}}$ – here we are using the fact that the co-efficient of q_2 in q_1 is non zero. Thus, q_1 is a non-zero polynomial. □

Proof of Lemma 2. We first observe that every sub-polynomial $p_w^{\vec{j}}$ computed according to the theorem statement, does not contain any of the variables $\mathbf{Y}[\beta_{u_0}^i, j_l]$ or $\mathbf{Y}[\beta_{v_0}^i, j_l]$, for any $i \in [L], l \in [n \cdot L]$. Suppose $\mathbf{p}_w^{\vec{j}}$ is a polynomial computed at node w and is not of the form as described in the theorem statement. Then there are two possibilities:

- *$\mathbf{p}_w^{\vec{j}}$ does not contain variables $\mathbf{Y}[\beta_{u_0}^i, j_l]$ or $\mathbf{Y}[\beta_{v_0}^i, j_l]$, for any $i \in [L], l \in [n \cdot L]$*: This means that there are two distinct polynomials in variables $\{\mathbf{Y}[\beta_{v_0}^i, j_l]\}$ of small degree such that they both are zero polynomials. But by Schwartz-Zippel-DeMillo-Lipton theorem, this is not possible.

- $\mathbf{p}_w^{\vec{j}}$ *contains variables* $\mathbf{Y}[\beta^i_{u_0}, j_l]$ *or* $\mathbf{Y}[\beta^i_{v_0}, j_l]$*, for some* $i \in [L], l \in [n \cdot L]$: We invoke Claim 13: the variables $\{\mathbf{Y}[\beta^i_{u_0}, j_l]\}$, $\{\mathbf{Y}[\beta^i_{v_0}, j_l]\}$ are in the set I and furthermore, $\mathbf{p}_w^{\vec{j}}$ is a sub-polynomial of p^*. From Claim 13, that p^* is a non zero polynomial. This contradicts the hypothesis that p^* is a zero test polynomial.

□

□

8.3 Security of Assumptions in Ideal Multilinear Map Model

We now use the slot preservation property proved in the previous section to justify the security of assumptions stated in Section 7 in the ideal multilinear map model. Before that, we prove a general theorem about the scheme that will be useful to state. We begin by setting up some notation.

Consider an adversary $\mathcal{A}$ in the ideal multilinear map model. Let $\mathcal{A}$ have access to the oracle $\mathcal{M}$. The adversary initially receives the handles to the encodings corresponding to the ciphertexts and the functional keys. That is, it receives handles to the slotted encodings $\left\{ \left[\mathbf{z}^1_k|\mathbf{z}^2_k|\mathbf{z}^3_k\right]_{\mathbf{u}} \right\}_{j,u}$. Here, $\left[\mathbf{z}^1_k|\mathbf{z}^2_k|\mathbf{z}^3_k\right]_{\mathbf{u}}$ is one of the following forms:

- $\left[x_i^j|\gamma_i \cdot \alpha_{i,u}|0\right]_{\mathbf{u}}$. Here, x_i^j is the j^{th} bit of input x_i. The value γ_i is the randomness associated with the i^{th} ciphertext. In the indistinguishability of semi-functional ciphertexts game, x_i^j will be set to 0.
- $\left[0|\gamma_i^D \cdot S^{-1}|b_i\right]_{\mathbf{u}}$. The value b_i is set to 0, if this encoding corresponds to honestly generated ciphertexts, otherwise $b_i = 1$.
- $[0|\Delta \cdot S|b'_i]_{\mathbf{u}}$. The value b'_i is set to 0, if this encoding to honestly generated functional key, otherwise if it is associated with the semi-functional key we have $b'_i = v$ where v is the hardwired value corresponding to the semi-functional key.

For each of the above elements, the oracle uses this formal variable $\mathbf{Y}[\cdot]$ to associate with that particular element. For example, the element γ_i is associated with the variable $\mathbf{Y}[\gamma_i]$.

It then executes the addition/ subtraction, multiplication queries to the oracle. In the end, it submits a handle to the zero test query.

Let us consider the case when the zero test query was successful, i.e., the oracle returns 1. From the *slot preservation assumption*, we have that there exists a circuit C such that the encoding $\left[C(\{\mathbf{z}^1_j\})|\cdots|C(\{\mathbf{z}^K_j\})\right]_{\mathbf{r}}$, encoded at $\mathbf{G}_{\mathbf{r}}$, is actually an encoding of zero. We now make some structural observations about the polynomial $\mathbf{Q}_C$ corresponding to the arithmetic circuit C.

Theorem 17. *With probability negligibly close to 1, the polynomial* $\mathbf{Q}_C$ *is of the form: here,* $\ell_{\mathbf{x}}, \ell_{\mathbf{f}} \in \mathbb{N}$*. Let* $I_x \subseteq [\ell_{\mathbf{x}}]$ *and* $I_f \subseteq [\ell_{\mathbf{f}}]$*.*

$$\mathbf{Q}_C = \sum_{i \in I_x} \sum_{j \in I_f} (\mathbf{Y}[\gamma_i])^{\mathbf{D}} \cdot \left(\ c_{i,j} \cdot p_j(\{\mathbf{Y}[\alpha_{k,u}]\}) + c'_{i,j} \cdot \mathbf{Y}[\Delta_j] \ \right),$$

where $c_{i,j}, c'_{i,j} \in \mathbb{F}_{\mathbf{p}}$ *and* $c'_{i,j} = -c_{i,j}$*.*

Proof. We first observe that in the expansion of $\mathbf{Q}_C$, if the term $\mathbf{Y}[\Delta_j] \cdot \mathbf{Y}[S]$ is multiplied with a term t, for some $j \in [\ell_{\mathbf{f}}]$, then t has to be of the form $\mathbf{Y}[\gamma_i]^{\mathbf{D}} \cdot \mathbf{Y}[S]^{-1}$ for some $i \in [\ell_{\mathbf{x}}]$. Suppose not: there exists $j \in [\ell_{\mathbf{f}}]$ such that there is a term $\mathbf{Y}[\Delta_j] \cdot \mathbf{Y}[S] \cdot t$ in the expansion of $\mathbf{Q}_C$ such that $t \neq \mathbf{Y}[\gamma_i]^{\mathbf{D}} \cdot \mathbf{Y}[S]^{-1}$ for any $i \in [\ell_{\mathbf{x}}]$. But note that $\mathbf{Y}[S]^{-1}$ always appears together with

$\mathbf{Y}[\gamma_i]^{\mathbf{D}}$ and thus, t does not contain the variable $\mathbf{Y}[S]^{-1}$. Thus, the term $\mathbf{Y}[\Delta_j] \cdot \mathbf{Y}[S] \cdot t$ has degree at least 1 in the variable $\mathbf{Y}[S]$. Thus, $\mathbf{Q}_C$ can be viewed as a polynomial in $\mathbf{Y}[S]$. Since we assign a random field element to $\mathbf{Y}[S]$ in the real experiment, the probability that $\mathbf{Q}_C$ evaluates to 0 is negligible. This follows from Schwartz-Zippel-DeMillo-Lipton lemma for rational polynomials (Lemma 1). For the same reason, if the term $\mathbf{Y}[\gamma_i]^{\mathbf{D}} \cdot \mathbf{Y}[S]^{-1}$, for some $i \in [\ell_{\mathbf{x}}]$ is multiplied with a term t in the expansion of $\mathbf{Q}_C$ then t has to be of the form $\mathbf{Y}[\Delta_j] \cdot \mathbf{Y}[S]$ for some $j \in [\ell_{\mathbf{f}}]$.

We then consider terms involving variables $\mathbf{Y}[\gamma_i]$ and $\mathbf{Y}[\alpha_{k,u}]$. We can rewrite polynomial $\mathbf{Q}_C$ such that it is in turn the sum of the following types of polynomials:

- The first type of polynomials is of the form $c \cdot \mathbf{Y}[\gamma_i]^{d_1} \cdot \mathbf{Y}[\gamma_{i'}]^{d_2} \cdot$ (polynomial in $\mathbf{Y}[\alpha_{k,u}]$), where $c \in \mathbb{F}_{\mathbf{p}}$. Also, either $d_1 + d_2 < \mathbf{D}$ or $d_1 + d_2 > \mathbf{D}$.
- The second type is the same as the first type except $d_1 + d_2 = \mathbf{D}$ and both $d_1 \neq 0$ and $d_2 \neq 0$.
- The third type is of the form $c \cdot \mathbf{Y}[\gamma_i]^D \cdot$ (polynomial in $\mathbf{Y}[\alpha_{k,u}]$), where $c \in \mathbb{F}_{\mathbf{p}}$.

We consider the above type of polynomials one by one.

- The first type of polynomials cannot exist, i.e., the co-efficient c has to be zero. This is because if $d_1 + d_2 < \mathbf{D}$ then this cannot be the encoding at the final level. Similarly, $d_1 + d_2 > \mathbf{D}$ is not possible since the multilinear map model only allows for evaluations upto degree $\mathbf{D}$.
- The second type of polynomials also cannot exist, i.e., their co-efficients has to be zero. Suppose not, we can express $\mathbf{Q}_C$ as:

$$\begin{aligned}\mathbf{Q}_C &= c \cdot \mathbf{Y}[\gamma_i]^{d_1^*} \cdot \mathbf{Y}[\gamma_{i'}]^{d_2^*} \cdot (\text{polynomial in } \mathbf{Y}[\alpha_{k,u}]) \\ &\quad + \sum_{d_1 \neq d_1^* \text{ or } d_2 \neq d_2^*} \cdot \mathbf{Y}[\gamma_i]^{d_1^*} \cdot \mathbf{Y}[\gamma_{i'}]^{d_2^*} \cdot (\text{polynomial in } \alpha_{k,u}) \\ &\quad + \text{polynomial in} \mathbf{Y}[\gamma_i], \mathbf{Y}[S], \mathbf{Y}[\Delta]\end{aligned}$$

  We can view $\mathbf{Q}_C$ as a polynomial in variables $\{\mathbf{Y}[\gamma_i]\}$. Since $\mathbf{Y}[\gamma_i]$ is substituted with random values in the real experiment, by Schwartz-Zippel lemma, this polynomial would evaluate to zero only with negligible probability. This contradicts the hypothesis that $\mathbf{Q}_C$ is a zero-test polynomial.

This means that only the third type of polynomials survive in the expansion of $\mathbf{Q}_C$. Now we can rewrite $\mathbf{Q}_C$ as:

$$\mathbf{Q}_C = \sum_{i \in I_x} \mathbf{Y}[\gamma_i]^D \cdot (\text{polynomial in } \alpha_{k,u}) + \sum_{i \in I_x} \sum_{j \in I_f} c'_{i,j} \cdot \mathbf{Y}[\Delta_j] \cdot \mathbf{Y}[\gamma_i]^D$$

We now zoom in on polynomials of the form $\mathbf{Y}[\gamma_i]^D \cdot$ (polynomial in $\alpha_{k,u}$). We claim that for every $i \in I_x$, this polynomial should be of the form $\mathbf{Y}[\gamma_i]^D \cdot (\sum_{j \in I_f} p_j(\{\mathbf{Y}[\alpha_{k,u}]\}))$. Suppose not: that is there exists a polynomial h such that $\mathbf{Y}[\gamma_i]^D \cdot h(\{\mathbf{Y}[\alpha_{k,u}]\})$ is a sub-polynomial in $\mathbf{Q}_C$ and $h \neq p_j$ for every $j \in [\ell_{\mathbf{x}}]$. It is easy to see that this term can never get canceled with any of the other terms. This proves that every $i \in I_x$, this polynomial should be of the form $\mathbf{Y}[\gamma_i]^D \cdot (\sum_{j \in I_f} p_j(\{\mathbf{Y}[\alpha_{k,u}]\}))$.

Summarising all the observations we have,

$$\mathbf{Q}_C = \sum_{i \in I_x} \sum_{j \in I_f} (\mathbf{Y}[\gamma_i])^{\mathbf{D}} \cdot \left(\ c_{i,j} \cdot p_j(\{\mathbf{Y}[\alpha_{k,u}]\}) + c'_{i,j} \cdot \mathbf{Y}[\Delta_j]\ \right)$$

Now that we have ascertained the structure of $\mathbf{Q}_C$, we now use this to justify both assumptions #1 and #2.

Justifying Assumption #1. We consider two cases:

- In the first case, the third slot in the functional key corresponds to 0 (honestly generated functional key). In this case, evaluating the polynomial $\mathbf{Q}_C$ on all the elements in the third slot that are part of the encodings issued in the security game yield a value 0.
- In the second case, the third slot in the functional key corresponds to θ (semi-functional key). However, handles corresponding to encodings $\left[0|\gamma_i^D \cdot S^{-1}|0\right]_{\mathbf{u}}$ and $[0|\Delta \cdot S|v]_{\mathbf{u}}$ are paired with each other in the computation of $\mathbf{Q}_C$ for every i^{th} ciphertext. This is because the term $\mathbf{Y}[\gamma_i]^D \cdot \mathbf{Y}[\Delta]$ is present in $\mathbf{Q}_C$.

Justifying Assumption #2. We consider two cases:

- In the first case, the third slot in the (honestly generated) functional key corresponds to θ . However, the third slot is not activated. From the structure of $\mathbf{Q}_C$, the value obtained by computing $\mathbf{Q}_C$ in the first slot is θ. Similarly, the computation of $\mathbf{Q}_C$ on the third slot yields the value 0.
- In the second case, the third slot in the functional key corresponds to θ (semi-functional key). Since the third slot in the ciphertext is activated, evaluating the polynomial $\mathbf{Q}_C$ on all the elements in the third slot that are part of the encodings issued in the security game yield the value θ. This is again by observing the above structure of $\mathbf{Q}_C$. Furthermore, in the first slot, the evaluation of $\mathbf{Q}_C$ yields value 0.

□

References

[AAB$^+$15] Shashank Agrawal, Shweta Agrawal, Saikrishna Badrinarayanan, Abishek Kumarasubramanian, Manoj Prabhakaran, and Amit Sahai. On the practical security of inner product functional encryption. In *PKC*, pages 777–798, 2015.

[AB15] Benny Applebaum and Zvika Brakerski. Obfuscating circuits via composite-order graded encoding. In *TCC*, pages 528–556, 2015.

[ABCP15] Michel Abdalla, Florian Bourse, Angelo De Caro, and David Pointcheval. Simple functional encryption schemes for inner products. In *PKC*, pages 733–751, 2015.

[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.

[ABSV15] Prabhanjan Ananth, Zvika Brakerski, Gil Segev, and Vinod Vaikuntanathan. From selective to adaptive functional encryption. In *CRYPTO*, 2015.

[AIK06] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. Computationally private randomizing polynomials and their applications. *Computational Complexity*, 15(2):115–162, 2006.

[AJ15] Prabhanjan Ananth and Abhishek Jain. Indistinguishability obfuscation from compact functional encryption. In *Advances in Cryptology–CRYPTO 2015*, pages 308–326. Springer, 2015.

[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.

[AL16] Benny Applebaum and Shachar Lovett. Algebraic attacks against random local functions and their countermeasures. In *STOC*, pages 1087–1100, 2016.

[BGG$^+$14] Dan Boneh, Craig Gentry, Sergey Gorbunov, Shai Halevi, Valeria Nikolaenko, Gil Segev, Vinod Vaikuntanathan, and Dhinakaran Vinayagamurthy. Fully key-homomorphic encryption, arithmetic circuit ABE and compact garbled circuits. In *EUROCRYPT*, pages 533–556, 2014.

[BGI$^+$01] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. In *CRYPTO*, pages 1–18, 2001.

[BGJ$^+$16] Nir Bitansky, Shafi Goldwasser, Abhishek Jain, Omer Paneth, Vinod Vaikuntanathan, and Brent Waters. Time-lock puzzles from randomized encodings. In *ITCS*, 2016.

[BGK$^+$14] Boaz Barak, Sanjam Garg, Yael Tauman Kalai, Omer Paneth, and Amit Sahai. Protecting obfuscation against algebraic attacks. In *EUROCRYPT*, pages 221–238, 2014.

[BHR12] Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Adaptively secure garbling with applications to one-time programs and secure outsourcing. In *ASIACRYPT*, pages 134–153, 2012.

[BJK15] Allison Bishop, Abhishek Jain, and Lucas Kowalczyk. Function-hiding inner product encryption. In *ASIACRYPT*, pages 470–491, 2015.

[BLP16] Nir Bitansky, Huijia Lin, and Omer Paneth. On removing graded encodings from functional encryption. Cryptology ePrint Archive, Report 2016/962, 2016. `http://eprint.iacr.org/2016/962`.

[BNPW16] Nir Bitansky, Ryo Nishimaki, Alain Passelègue, and Daniel Wichs. From cryptomania to obfustopia through secret-key functional encryption. Cryptology ePrint Archive, Report 2016/558, 2016. `http://eprint.iacr.org/2016/558`.

[BNS13] Dan Boneh, Valeria Nikolaenko, and Gil Segev. Attribute-based encryption for arithmetic circuits. *IACR Cryptology ePrint Archive*, 2013:669, 2013.

[BP15] Nir Bitansky and Omer Paneth. Zaps and non-interactive witness indistinguishability from indistinguishability obfuscation. In *TCC*, 2015.

[BPR15] Nir Bitansky, Omer Paneth, and Alon Rosen. On the cryptographic hardness of finding a nash equilibrium. In *FOCS*, 2015.

[BR14] Zvika Brakerski and Guy N. Rothblum. Virtual black-box obfuscation for all circuits via generic graded encoding. In *TCC*, pages 1–25, 2014.

[BS15] Zvika Brakerski and Gil Segev. Function-private functional encryption in the private-key setting. In *TCC*, 2015.

[BSW07] John Bethencourt, Amit Sahai, and Brent Waters. Ciphertext-policy attribute-based encryption. In *IEEE (S&P 2007)*, pages 321–334, 2007.

[BV15] Nir Bitansky and Vinod Vaikuntanathan. Indistinguishability obfuscation from functional encryption. In *FOCS*. IEEE, 2015.

[CGH$^+$15] Jean-Sébastien Coron, Craig Gentry, Shai Halevi, Tancrède Lepoint, Hemanta K. Maji, Eric Miles, Mariana Raykova, Amit Sahai, and Mehdi Tibouchi. Zeroizing without low-level zeroes: New MMAP attacks and their limitations. In *CRYPTO*, pages 247–266, 2015.

[CHL$^+$15] Jung Hee Cheon, Kyoohyung Han, Changmin Lee, Hansol Ryu, and Damien Stehlé. Cryptanalysis of the multilinear map over the integers. In *EUROCRYPT*, pages 3–12, 2015.

[CHN$^+$16] Aloni Cohen, Justin Holmgren, Ryo Nishimaki, Vinod Vaikuntanathan, and Daniel Wichs. Watermarking cryptographic capabilities. In *STOC*, 2016.

[CIJ$^+$13] Angelo De Caro, Vincenzo Iovino, Abhishek Jain, Adam O'Neill, Omer Paneth, and Giuseppe Persiano. On the achievability of simulation-based security for functional encryption. In *CRYPTO*, pages 519–535, 2013.

[CLT13] Jean-Sébastien Coron, Tancrede Lepoint, and Mehdi Tibouchi. Practical multilinear maps over the integers. In *CRYPTO*, pages 476–493. Springer, 2013.

[CS02] Ronald Cramer and Victor Shoup. Universal hash proofs and a paradigm for adaptive chosen ciphertext secure public-key encryption. In *EUROCRYPT*, pages 45–64, 2002.

[DDM16] Pratish Datta, Ratna Dutta, and Sourav Mukhopadhyay. Functional encryption for inner product with full function privacy. In *PKC*, pages 164–195, 2016.

[GGG$^+$14] Shafi Goldwasser, S. Dov Gordon, Vipul Goyal, Abhishek Jain, Jonathan Katz, Feng-Hao Liu, Amit Sahai, Elaine Shi, and Hong-Sheng Zhou. Multi-input functional encryption. In *EUROCRYPT*, 2014.

[GGH13a] Sanjam Garg, Craig Gentry, and Shai Halevi. Candidate multilinear maps from ideal lattices. In *EUROCRYPT*, 2013.

[GGH$^+$13b] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *FOCS*, pages 40–49, 2013.

[GGH15] Craig Gentry, Sergey Gorbunov, and Shai Halevi. Graph-induced multilinear maps from lattices. In *TCC*, pages 498–527. Springer, 2015.

[GGHR14] Sanjam Garg, Craig Gentry, Shai Halevi, and Mariana Raykova. Two-round secure MPC from indistinguishability obfuscation. In *TCC*, 2014.

[GGHZ14] Sanjam Garg, Craig Gentry, Shai Halevi, and Mark Zhandry. Fully secure attribute based encryption from multilinear maps. *IACR Cryptology ePrint Archive*, 2014:622, 2014.

[GLSW15] Craig Gentry, Allison Bishop Lewko, Amit Sahai, and Brent Waters. Indistinguishability obfuscation from the multilinear subgroup elimination assumption. In *FOCS*, pages 151–170, 2015.

[GMM$^+$16] Sanjam Garg, Eric Miles, Pratyay Mukherjee, Amit Sahai, Akshayaram Srinivasan, and Mark Zhandry. Secure obfuscation in a weak multilinear map model. Cryptology ePrint Archive, Report 2016/817, 2016. http://eprint.iacr.org/2016/817.

[GMR89] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM J. Comput.*, 18(1):186–208, 1989.

[Gol11] Oded Goldreich. Candidate one-way functions based on expander graphs. In *Studies in Complexity and Cryptography. Miscellanea on the Interplay between Randomness and Computation*, pages 76–87. Springer, 2011.

[GPSW06] Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent Waters. Attribute-based encryption for fine-grained access control of encrypted data. In *ACM CCS*, 2006.

[GS08] Jens Groth and Amit Sahai. Efficient non-interactive proof systems for bilinear groups. In *EUROCRYPT*, pages 415–432, 2008.

[HSW14] Susan Hohenberger, Amit Sahai, and Brent Waters. Replacing a random oracle: Full domain hash from indistinguishability obfuscation. In *EUROCRYPT*, 2014.

[IK00] Yuval Ishai and Eyal Kushilevitz. Randomizing polynomials: A new representation with applications to round-efficient secure computation. In *FOCS*, pages 294–304, 2000.

[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*, pages 146–162, 2008.

[Lin16] Huijia Lin. Indistinguishability obfuscation from constant-degree graded encoding schemes. In *EUROCRYPT*, pages 28–57, 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 *EUROCRYPT*, pages 62–91, 2010.

[LPST16] Huijia Lin, Rafael Pass, Karn Seth, and Sidharth Telang. Indistinguishability obfuscation with non-trivial efficiency. In *PKC*, pages 447–462, 2016.

[LV16] Huijia Lin and Vinod Vaikuntanathan. Indistinguishability obfuscation from ddh-like assumptions on constant-degree graded encodings. In *FOCS*, 2016.

[MSZ16] Eric Miles, Amit Sahai, and Mark Zhandry. Annihilation attacks for multilinear maps: Cryptanalysis of indistinguishability obfuscation over GGH13. In *CRYPTO*, pages 629–658, 2016.

[OT08] Tatsuaki Okamoto and Katsuyuki Takashima. Homomorphic encryption and signatures from vector decomposition. In *International Conference on Pairing-Based Cryptography*, pages 57–74. Springer, 2008.

[OT09] Tatsuaki Okamoto and Katsuyuki Takashima. Hierarchical predicate encryption for inner-products. In *EUROCRYPT*, pages 214–231. Springer, 2009.

[OW14] Ryan O'Donnell and David Witmer. Goldreich's PRG: evidence for near-optimal polynomial stretch. In *CCC*, pages 1–12, 2014.

[SW05] Amit Sahai and Brent Waters. Fuzzy identity-based encryption. In *EUROCRYPT*, 2005.

[SW08] Amit Sahai and Brent Waters. Slides on functional encryption, powerpoint presentation. 2008.

[SW14] Amit Sahai and Brent Waters. How to use indistinguishability obfuscation: deniable encryption, and more. In *STOC*, pages 475–484, 2014.

[Wat09] Brent Waters. Dual system encryption: Realizing fully secure IBE and HIBE under simple assumptions. In *Advances in Cryptology – CRYPTO '09*, pages 619–636, 2009.

[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 *EUROCRYPT*, pages 439–467, 2015.

A Implication of Sub-Linear FE to XiO

We show how to obtain XiO (Definition 5) starting from 1-key ε-sub-linear FE (Definition 4). We denote a 1-key sub-linear FE by 1SLSKFE.

Suppose there exists a 1SLSKFE with the following structure:

- The size of a functional key of f has size $|f| \cdot \text{poly}(\lambda, \ell)$, where ℓ is the input length of f.
- The running time of encryption algorithm, on input message of length ℓ, is: $|f|^{1-\varepsilon}\text{poly}(\lambda, \ell)$, where ε is a constant associated with the description of the scheme.

We will call such a 1SLSKFE scheme as poly-overhead ε-1SLSKFE. Our construction of 1SLSKFEsatisfies the above properties.

Theorem 18. *Assuming the existence of poly-overhead ε-1SLSKFE for constant $\varepsilon < 1$, there exists a ε'-XiO scheme, where $\varepsilon' < 1$ is a constant.*

Proof Sketch. The construction of XiO follows along the same lines as the transformation from 1-key compact secret key FE to XiO, due to Bitansky et al. [BNPW16].

We describe at a high level how the construction works: an (XiO) obfuscation of $C : \{0,1\}^n \to \{0,1\}$ consists of a 1SLSKFE key of function G and ciphertexts $\mathsf{CT}_1, \ldots, \mathsf{CT}_L$ with $L = 2^{(1-c)n} - 1$, where c is a constant that we will fix soon. Here, G takes as input (C, y) and outputs $C(0||y), \ldots, C(2^{cn} - 1||y)$. The ciphertext CT_i is an encryption of (C, i), for $0 \leq i \leq L$. This completes the description of the obfuscation algorithm. The evaluation of the obfuscated circuit is straightforward.

The size of obfuscated circuit is $2^{cn} \cdot \mathrm{poly}(n, \lambda) + 2^{(1-c)n} \cdot (2^{cn\varepsilon} \cdot \mathrm{poly}(n, \lambda))$. Fix $c = \frac{1}{2}$. We have the size of obfuscated circuit to be $\leq 2^{\frac{n}{2}(1+\varepsilon)} \cdot \mathrm{poly}(n, \lambda)$. Since ε is a constant < 1, we have $(1 + \varepsilon) < 2$ and hence the above quantity is at most $2^{n\varepsilon'} \cdot \mathrm{poly}(n, \lambda)$ for some constant $\varepsilon' < 1$. □

We can also show that poly-overhead 1SLSKFE for NC^1 implies 1SLSKFE for all P.

Theorem 19. *Poly-overhead ε-1SLSKFE for NC^1 implies poly-overhead ε-1SLSKFE for all P, where $\varepsilon < 1$ is a constant.*

Proof Sketch. This transformation follows along the same lines as in prior bootstrapping theorems [ABSV15]: the functional key of f w.r.t 1SLSKFE for P scheme is essentially the functional key of G_τ w.r.t 1SLSKFE for NC^1 where G_τ takes as input $(x, K, \perp)$ and outputs a randomized encoding of $f(x)$ w.r.t randomness derived from $\mathsf{PRF}(K, \tau)$.

We now argue about the efficiency of the resulting scheme.

- The size of the functional key of G_τ is $|G_\tau| \cdot \mathrm{poly}(\lambda, \ell)$, where ℓ is the input length of f. This follows from the poly-overhead property of the underlying FE scheme. Furthermore, by using a suitable randomized encodings scheme such as Yao's garbled circuits, we have $|G_\tau| = |f| \cdot \mathrm{poly}(\lambda)$. This proves that the resulting scheme for P satisfies the poly-overhead property.
- Now, lets analyze the size of the ciphertext in the resulting scheme. The size of an encryption of message x of length ℓ is $|G_\tau|^\varepsilon \cdot \mathrm{poly}(\lambda, \ell)$. From the previous bullet, $|G_\tau| = |f| \cdot \mathrm{poly}(\lambda)$. Thus, the size of the ciphertext is $|f|^\varepsilon \cdot \mathrm{poly}(\lambda, \ell)$.

□

Combining Theorems 18 and 19, we have the following corollary.

Corollary 1. *Assuming poly-overhead ε-1SLSKFE for NC^1, there exists a ε'-XiO, where $\varepsilon, \varepsilon' < 1$ are constants.*

When the above corollary is combined with [LPST16] and [AJ15, BV15], we get

Corollary 2. *Assuming poly-overhead ε-1SLSKFE for NC^1 that is sub-exponentially secure and sub-exponentially secure learning with errors, there exists an indistinguishability obfuscation scheme.*

Remarks about Theorem 19. In the proof of Theorem 19, we prove that ε-1SLSKFE for $\mathcal{C} \in NC^1$ implies ε-1SLSKFE for arbitrary class of polynomial-sized circuits. Here, $\mathcal{C}$ consists of circuits that compute the encode algorithm of a randomized encoding scheme. More specifically, we can use Yao's garbling schemes.

We claim that every circuit $C \in \mathcal{C}$ of output length $\mathbf{N}$ is of the form $C = (C_1, \ldots, C_{\mathbf{N}})$, where:

- C_i outputs the i^{th} output bit of C,
- Depth of C_i is $c \cdot \log(\lambda)$, where c is a constant independent of C,
- $|C_i| = \text{poly}(\lambda)$ for a fixed polynomial poly and,
- C_i for every $i \in [\mathbf{N}]$ have the same topology. That is, every C_i can be written as $\widehat{C}[V_i]$, where $\hat{C}$ is a circuit and V_i are the constants on a subset of its wires.

To see this, we can view C_i as computing the garbled table associated with the i^{th} gate. The circuit computing a single garbled table has size fixed polynomial in security parameter and also its depth is $c \cdot \log(\lambda)$, where c is independent of the circuit being garbled. Every circuit C_i can be written as $\widehat{C}[V_i]$ with the constants V_i corresponding to the description of i^{th} gate.

B Conversion of Polynomials: From $\mathbb{F}_2$ to $\mathbb{F}_\mathbf{p}$

We recall the conversion of polynomials from $\mathbb{F}_2$ to polynomials over $\mathbb{F}_\mathbf{p}$. The same conversion when applied to (T, Φ)-respecting polynomials over $\mathbb{F}_2$ yields (T, Φ)-respecting polynomials over $\mathbb{F}_\mathbf{p}$ – that is, the associated tree T and set of maps Φ remains unchanged.

Transformation $\mathsf{ConvPoly}$. Consider a polynomial $p_1 \in \mathbb{F}_2[y_1, \ldots, y_n]$ and let the function computed by the polynomial be $F : \{0,1\}^n \to \{0,1\}$ i.e., $p(x) = F(x)$. We construct another polynomial $p_2 \in \mathbb{F}_\mathbf{p}[y_1, \ldots, y_n]$ associated with the same function F i.e., $p_2(x) = F(x)$. We denote $p_2 = \mathsf{ConvPoly}(p_1, 2, \mathbf{p})$.

The transformation inductively proceeds as follows:

Base Case: If $p_1 = y_i$ for some $i \in [n]$ then $p_2 = y_i$.

Recursion Step: There are two cases to consider here:

- If p_1 is of the form $p_1' + p_1''$. Let $p_2' \in \mathbb{F}_\mathbf{p}[y_1, \ldots, y_n]$ (resp., p_2'') be obtained by transforming $p_1' \in \mathbb{F}_2[y_1, \ldots, y_n]$ (resp., p_1''). Then, we construct $p_2 = (1 - p_2')p_2'' + (1 - p_2'')p_2'$.
  Explanation: Let $p_1'(y) = 0$ and $p_1''(y) = 0$. This means that $p_1(y) = 0$. Then by hypothesis, $p_2'(y) = 0$ and $p_2''(y) = 0$. Now, by definition, $p_2(y) = 0$. The same argument can be applied to the case when $p_1'(y) = 1$ and $p_1''(y) = 1$. The remaining case is when $p_1'(y) \neq p_1''(y)$. In this case, $p_1(y) = 1$. Then by hypothesis, $p_2'(y) \neq p_2''(y)$. By definition, this means that $p_2(y) = 1$.

- If p_1 is of the form $p_1' p_1''$. Let $p_2' \in \mathbb{F}_\mathbf{p}[y_1, \ldots, y_n]$ (resp., p_2'') be obtained by transforming $p_1' \in \mathbb{F}_2[y_1, \ldots, y_n]$ (resp., p_1''). Then we construct $p_2 = p_2' p_2''$.
  Explanation: $p_2(y) = 1$ if and only if $p_2'(y) = 1 \wedge p_2''(y) = 1$ which in turn is true if and only if $p_1'(y) = 1$ and $p_1''(y) = 1$.

Note that the degree of the resulting polynomial p_2 is exponential in the number of addition and multiplication operations (over $\mathbb{F}_2$) required to compute p_1.

Theorem 20. *Consider a polynomial $p_1 \in \mathbb{F}_2[y_1, \ldots, y_n]$. Let the number of addition and multiplication operations, over $\mathbb{F}_2$, required to compute p_1 be $\mathbf{D}$. Let $p_2 = \mathsf{ConvPoly}(p_1, 2, \mathbf{p})$. We have* $\deg[\mathbf{p}](p_2) = \mathrm{O}(2^{\mathbf{D}})$.