Indistinguishability Obfuscation from Simple-to-State Hard Problems: New Assumptions, New Techniques, and Simplification

Romain Gay* Aayush Jain† Huijia Lin‡ Amit Sahai§

Abstract

In this work, we study the question of what set of simple-to-state assumptions suffice for constructing functional encryption and indistinguishability obfuscation ($i\mathcal{O}$), supporting all functions describable by polynomial-size circuits. Our work improves over the state-of-the-art work of Jain, Lin, Matt, and Sahai (Eurocrypt 2019) in multiple dimensions.

NEW ASSUMPTION: Previous to our work, all constructions of $i\mathcal{O}$ from simple assumptions required novel pseudorandomness generators involving LWE samples and constant-degree polynomials over the integers, evaluated on the error of the LWE samples. In contrast, Boolean pseudorandom generators (PRGs) computable by constant-degree polynomials have been extensively studied since the work of Goldreich (2000).[1] We show how to replace the novel pseudorandom objects over the integers used in previous works, with appropriate Boolean pseudorandom generators with sufficient stretch, when combined with LWE with binary error over suitable parameters. Both binary error LWE and constant degree Goldreich PRGs have been a subject of extensive cryptanalysis since much before our work and thus we back the plausibility of our assumption with security against algorithms studied in context of cryptanalysis of these objects.

NEW TECHNIQUES: we introduce a number of new techniques:

- We show how to build partially-hiding *public-key* functional encryption, supporting degree-2 functions in the secret part of the message, and arithmetic NC^1 functions over the public part of the message, assuming only standard assumptions over asymmetric pairing groups.

*IBM Research Zurich. Email: romain.rgay@gmail.com.

†UCLA, Center for Encrypted Functionalities, and NTT Research. Email: aayushjain@cs.ucla.edu.

‡UW. Email: rachel@cs.washington.edu.

§UCLA, Center for Encrypted Functionalities. Email: sahai@cs.ucla.edu.

[1] Goldreich and follow-up works study Boolean pseudorandom generators with constant-locality, which can be computed by constant-degree polynomials.

- We construct single-ciphertext and single-secret-key functional encryption for all circuits with long outputs, which has the features of *linear* key generation and compact ciphertext, assuming only the LWE assumption.

SIMPLIFICATION: Unlike prior works, our new techniques furthermore let us construct *public-key* functional encryption for polynomial-sized circuits directly (without invoking any bootstrapping theorem, nor transformation from secret-key to public key FE), and based only on the *polynomial hardness* of underlying assumptions. The functional encryption scheme satisfies a strong notion of efficiency where the size of the ciphertext is independent of the size of the circuit to be computed, and grows only sublinearly in the output size of the circuit and polynomially in the input size and the depth of the circuit. Finally, assuming that the underlying assumptions are subexponentially hard, we can bootstrap this construction to achieve $i\mathcal{O}$.

Contents

1 Introduction

This paper studies the notion of indistinguishability obfuscation ($i\mathcal{O}$) for general programs computable in polynomial time [BGI+01, GKR08, GGH+13], and develops several new techniques to strengthen the foundations of $i\mathcal{O}$. The key security property for $i\mathcal{O}$ requires that for any two equivalent programs P_0 and P_1 modeled as circuits of the same size, where "equivalent" means that $\mathsf{P}_0(x) = \mathsf{P}_1(x)$ for all inputs x, we have that $i\mathcal{O}(\mathsf{P}_0)$ is computationally indistinguishable to $i\mathcal{O}(\mathsf{P}_1)$. Furthermore, the obfuscator $i\mathcal{O}$ should run in probabilistically polynomial time.

This notion of obfuscation was coined by [BGI+01] in 2001. However, until 2013, there was not even a single candidate construction known. This changed with the breakthrough work of [GGH+13]. Soon after, the floodgates opened and a flurry of over 100 papers were published reporting applications of $i\mathcal{O}$ (e.g. [SW14, BFM14, GGG+14, HSW13, KLW15, BPR15] [CHN+16, GPS16, HJK+16]). Not only did $i\mathcal{O}$ enable the first constructions of numerous important cryptographic primitives, $i\mathcal{O}$ also *expanded* the scope of cryptography, allowing us to mathematically approach problems that were previously considered the domain of software engineering. A simple example along these lines is the notion of *crippleware* [GGH+13]: Alice, a software developer, has developed a program P using powerful secrets, and wishes to sell her work. Before requiring payment, Alice is willing to share with Bob a weakened (or "crippled") version of her software. Now, Alice could spend weeks developing this crippled version $\widetilde{\mathsf{P}}$ of her software, being careful not to use her secrets in doing so; or she could simply disable certain inputs to cripple it yielding an equivalent P', but this would run the risk of Bob hacking her software to re-enable those disabled features. $i\mathcal{O}$ brings this problem of software engineering into the realm of mathematical analysis. With $i\mathcal{O}$, Alice could avoid weeks of effort by simply giving to Bob $i\mathcal{O}(\mathsf{P}')$, and because this is indistinguishable from $i\mathcal{O}(\widetilde{\mathsf{P}})$, Alice is assured that Bob can learn no secrets.

Not only has $i\mathcal{O}$ been instrumental in realizing new cryptographic applications, it has helped us advance our understanding of long-standing theoretical questions. One such recent example is that of the first cryptographic evidence of the average-case hardness of the complexity class PPAD (which contains of the problem of finding Nash equilibrium). In particular, [BPR15] constructed hard instances for the End Of the Line (EOL) problem assuming subexponentially secure $i\mathcal{O}$ and one-way functions.

Our Contributions. In this work, we show how to simplify, both technically and conceptually, the task of constructing secure $i\mathcal{O}$ schemes. Notably, the ideas we develop in this work helped pave the way for the recent first construction of $i\mathcal{O}$ from well-studied assumptions [JLS20], resolving the central open question in the area of $i\mathcal{O}$.

We now discuss the contributions of our paper in detail.

What hardness assumptions suffice for constructing $i\mathcal{O}$? Given its importance, a crucial question is to identify what hardness assumptions, in particular, simple ones, suffice for constructing $i\mathcal{O}$. While it is hard to concretely measure simplicity in assumptions, important features include i) having succinct description, ii) being falsifiable and instance

independent (e.g., independent of the circuit being obfuscated), and iii) consisting of only a constant number of assumptions, as opposed to families of an exponential number of assumptions. However, research on this question has followed a tortuous path over the past several years, and so far, despite of a lot of progress, before our work, no known $i\mathcal{O}$ constructions [GGH$^+$13, BGK$^+$14, BR14, AGIS14, BMSZ16, GMM$^+$16, CVW18, Lin16, AS17, GLSW14, PST14, LV16, Lin17, LT17, GJK18, BIJ$^+$20, Agr19, AP20, AJL$^+$19, JLMS19, BDGM20] were based on assumptions that have all above features.

Our new assumption. In this work, building upon assumptions introduced in [AJL$^+$19, JLMS19], we introduce a new simple-to-state assumption, that satisfies all the features enumerated above. We show how to provably achieve $i\mathcal{O}$ based only on our new assumption combined with standard assumptions, namely subexponentially secure Learning With Errors (LWE) problem [Reg05], and subexponentially secure SXDH and bilateral DLIN assumptions over bilinear maps [Jou00, BF01]. Let us now describe, informally, our new assumption. In this introductory description, we will omit discussion of parameter choices; however, they are crucial (even for standard assumptions), and we discuss them in detail in our technical sections. We start by describing the ingredients that will go into the assumption.

Constant-degree[2] Boolean PRGs generalize constant-locality Boolean PRGs, as for Boolean functions, locality upper bounds the degree. The latter is tightly connected to the fundamental topic of Constraint Satisfaction Problems (CSPs) in complexity theory, and were first proposed for cryptographic use by Goldreich [Gol00] 20 years ago. The complexity theory and cryptography communities have jointly developed a rich body of literature on the cryptanalysis and theory of constant-locality Boolean PRGs [Gol00, MST03, ABR12, BQ12, App12, OW14, AL16, CDM$^+$18]. Our new assumption first postulates that there exists a constant d-degree Boolean PRG, $G : \{0,1\}^n \to \{0,1\}^m$ with sufficient stretch $m \geq n^{\lceil \frac{d}{2} \rceil \cdot (0.5+\epsilon)+\rho}$ for some constants $\epsilon, \rho > 0$, whose output $\boldsymbol{r} = G(\boldsymbol{x})$ should satisfy the standard notion of pseudorandomness. Furthermore, our assumption postulates that the pseudorandomness holds even when its Boolean input $\boldsymbol{x} \in \{0,1\}^n$ is embedded in LWE samples as noises, and the samples are made public. The latter is known as *Learning With Binary Errors (LWBE)*, which has been studied over the last decade [MP13, AG11, CTA19, CSA20]. Our new assumption, combining Boolean PRGs and LWBE, is as follows:

The G**-**LWEleak**-security assumption (informal).** For a prime modulus p that is subexponentially large in the security parameter.

$$\left(\{\boldsymbol{a}_i, \langle \boldsymbol{a}_i, \boldsymbol{s}\rangle + e_i \bmod p\}_{i\in[n]}, G, G(\boldsymbol{e})\right) \; //\boldsymbol{e} = (e_1, \ldots, e_n) \leftarrow \{0,1\}^n,\ \boldsymbol{a}_i, \boldsymbol{s} \leftarrow \mathbb{Z}_p^{n^{0.5+\epsilon}}$$

$$\approx \quad \left(\{\boldsymbol{a}_i, \langle \boldsymbol{a}_i, \boldsymbol{s}\rangle + e_i \bmod p\}_{i\in[n]}, G, \boldsymbol{r}\right) \; //\boldsymbol{r} \leftarrow \{0,1\}^m$$

As is evident here, this assumption is quite succinct, is falsifiable and instance-independent, does not involve an exponential family of assumptions, and does not use multilinear

[2]throughout this work, unless specified, by degree of boolean PRGs, we mean the degree of the polynomial computing the PRG over the reals.

maps. Furthermore, the ingredients that make up the assumption – Constant-degree Boolean PRGs and LWBE – have a long history of study within cryptography and complexity theory. As we discuss in detail in Section A.4, this assumption avoids attacks by all known cryptanalytic techniques. We note that the parameter n of LWBE samples is chosen to be sub-quadratic in the length $|\boldsymbol{s}|$ of the secret. This is needed in order to avoid Arora-Ge attacks on LWBE [AG11], and also avoid all known algebraic attacks [CTA19]. Indeed, the parameter choices we make are not possible using the previous work of [JLMS19], and the parameters used in [JLMS19] would render LWBE insecure.

Comparison of our assumption with the subsequent follow-up work of [JLS20]. Our shift to considering Boolean PRGs in the context of the approach of [JLMS19] provided a conceptual starting point for the subsequent work of [JLS20], which finally achieved $i\mathcal{O}$ from four well-founded assumptions: LPN over $\mathbb{F}_p$, LWE, Boolean PRGs in NC^0, and SXDH. Indeed, the work of [JLS20] essentially succeeds in "separating" the two ingredients in our assumption above — that is, basing $i\mathcal{O}$ on LWBE and the security of Goldreich's PRG with appropriate parameters separately, through a novel leveraging of the LPN over $\mathbb{F}_p$ assumption. Indeed, their work goes further and actually eliminates the need for the LWBE assumption entirely, and also eliminates the parameter requirements that we needed for Goldreich's PRG.

Complexity and clarity in $i\mathcal{O}$ constructions. Another motivation for our work is to address the complexity of existing $i\mathcal{O}$ constructions. Current constructions of $i\mathcal{O}$ are rather complex in the sense they often rely on many intermediate steps, each of which incur a complexity blow up, both in the sense of computational complexity and in the sense of difficulty of understanding. Ideally, for the sake of simplicity, $i\mathcal{O}$ schemes would minimize the number of such transformations, and instead aim at a more direct construction. In our case, we solely rely on the generic transformation of [AJ15, BV15], which shows that $i\mathcal{O}$ can be build from Functional Encryption [SW05], a primitive that was originally formulated by [BSW11, O'N10]. Roughly speaking, FE is a public-key or secret-key encryption scheme where users can generate restricted decryption keys, called functional keys, where each such key is associated with a particular function f. Such a key allows the decryptor to learn from an encryption of a plaintext m, the value $f(m)$, and nothing beyond that.

Previous constructions fell short in directly constructing a full-fledged FE needed for the implication of $i\mathcal{O}$ [AJ15, BV15]. For example, the work of [JLMS19] first obtain a "weak" FE that: i) is *secret-key*, ii) only generates function keys associated with function computable *only by* NC_0 *circuits*, iii) only ensures *weak security*, and iv) is based on subexponential hardness assumptions. Then, generic transformations are applied to "lift" the function class supported and the security level, which inevitably makes the final FE and $i\mathcal{O}$ schemes quite complex.

This state of affairs motivates simplifying $i\mathcal{O}$ constructions, for efficiency and simplicity itself, but also for making a technically deep topic more broadly accessible to the community. That is also one of the goals of this paper.

1.1 Our Results

Our main result is a simpler and more direct $i\mathcal{O}$ construction from the following assumptions.

Theorem 1.1. *There is a construction of $i\mathcal{O}$ for obfuscating all polynomial-sized circuits based on the following assumptions:*

- *There exists a constant-degree d Boolean PRG $G : \{0,1\}^n \to \{0,1\}^m$ with sufficient stretch $m \geq n^{\lceil \frac{d}{2} \rceil \cdot (0.5+\epsilon)+\rho}$ for some constant $\epsilon, \rho > 0$, and satisfies subexponential* G-LWEleak*-security,*
- *the subexponential LWBE assumption, and*
- *the subexponential bilateral DLIN and SXDH assumption over asymmetric pairing groups.*

Our techniques and additional results. Our construction of FE and $i\mathcal{O}$ are enabled by our new assumption and a number of new techniques designed to enable basing the security of $i\mathcal{O}$ on simple-to-state assumptions. We briefly summarize them here, but we elaborate on how they are used in the $i\mathcal{O}$ construction in the technical overview section immediately following this introduction.

Single-Ciphertext Functional Encryption with Linear Key Generation. We construct, assuming only LWE, a single-ciphertext secret-key functional encryption scheme able to give functional keys associated with any polynomial-sized circuit with depth bounded by λ, whose key generation and decryption algorithms have certain *simple structures*: i) The key generation algorithm computes a *linear* function on the master secret key and randomness, and ii) the decryption algorithm, given a ciphertext ct, a functional secret key sk_f associated with a function f and the description of f itself, first performs some deterministic computation on the ciphertext to get an intermediate ciphertext ct_f, followed by simply subtracting the sk_f from it, and then rounds to obtain the outcome. This object is previously known as special homomorphic encryption in the literature [AR17a, Agr19, LM18]. However, prior constructions only handles functional keys associated with NC_0 circuits (for those based on LWE) or NC^1 circuits (for those based on ring LWE). In this work, we view it through the FE lens, and construct it from LWE for all functions computable by polynomial-size circuits with any depth bounded by the security parameter λ. (Theorem 7.2). Constructing such single-ciphertext (or single-key) FE (that do not have compact ciphertexts) from standard assumptions is a meaningful goal on its own. In the literature, there are constructions of single-ciphertext FE from the minimal assumption of public-key encryption [SS10a, GVW12a], and several applications (e.g., [ABSV15]). However, they do not have the type of simple structures (e.g., linear key generation algorithm) our construction enjoys, and consequently cannot be used in our $i\mathcal{O}$ construction. These simple structural properties may also find uses in other applications.

Partially-Hiding Functional Encryption for NC^1 Public Computation and Degree-2 Private Computation. Partially-hiding Functional Encryption (PHFE) schemes involve functional secret keys, each of which is associated with some 2-ary function f, and decryption of a ciphertext encrypting $(\boldsymbol{x}, \boldsymbol{y})$ with such a key reveals $f(\boldsymbol{x}, \boldsymbol{y})$, $\boldsymbol{x}$, f, and nothing more about

$\boldsymbol{y}$. Since only the input $\boldsymbol{y}$ is hidden, such an FE scheme is called partially-hiding FE. The notion was originally introduced by [GVW12b] where it was used to bootstrap FE schemes. A similar notion of partially-hiding predicate encryption was proposed and constructed by [GVW15]. PHFE beyond the case of predicate encryption was first constructed by [AJS18] for functions f that compute degree-2 polynomials on the input $\boldsymbol{y}$ and degree-1 polynomials in $\boldsymbol{x}$, under the name of 3-restricted FE, in the secret-key setting. In this work, we construct a PHFE scheme from standard assumptions over bilinear pairing groups, that is *public-key* and supports functions f that have degree 2 in the private input $\boldsymbol{y}$, while performs an arithmetic NC^1 computation on the public input $\boldsymbol{x}$ (Theorem 8.1). More precisely, $f(\boldsymbol{x}, \boldsymbol{y}) = \langle g(\boldsymbol{x}), q(\boldsymbol{y}) \rangle$ where g is computable by an arithmetic log-depth circuit and q is a degree-2 polynomial. The previous best constructions of partially-hiding FE were secret-key, and could only handle NC_0 computation on the public input [JLMS19].

This contribution is interesting in its own right, as a step forward towards broadening the class of functions supported by FE schemes from standard assumptions. In particular, it can be used to combine rich access-control and perform selective computation on the encrypted data. In that context, the public input $\boldsymbol{x}$ represents some attributes, while the private input $\boldsymbol{y}$ is the plaintext. Functional secret keys reveal the evaluation of a degree-2 polynomial on the private input if some policy access, represented by an NC^1 arithmetic circuit evaluates to true on the attributes. This is the key-policy variant of a class of FE with rich access-control introduced in [ACGU20]. In the latter, the authors build an FE scheme where ciphertexts encrypt a Boolean formula (the public input) and a vector (the private input). Functional secret keys are associated with attributes and a vector of weights, and decryption yields the weighted sum of the plaintexts if the formula embedded in the ciphertext evaluates to true on the attributes embedded in the functional secret key. Their construction, as ours, rely on standard pairing assumptions, but only permits computation of *degree-1* polynomials on the private input. They also give a lattice-based construction, which is limited to identity-based access structures.

Independently of our work, [Wee20a] builds a public-key FE which support the same class of functions as our scheme (namely, NC^1 computation on the public input, and degree-2 on the private input), from standard assumptions in pairing groups. His construction has the notable advantage of have ciphertext size independent of the length of the public input.

2 Technical Overview

Below, we will use several different encryption schemes, and adopt the following notation to refer to ciphertexts and keys of different schemes. For a scheme x (e.g., a homomorphic encryption scheme HE, or a functional encryption scheme FE), we denote by $\mathsf{xct}, \mathsf{xsk}$ a ciphertext, or secret key of the scheme x. At times, we write $\mathsf{xct}(m)$, $\mathsf{xsk}(f)$ to make it explicit what is the encrypted message m and the associated function f; and write $\mathsf{xct}(k, m)$, $\mathsf{xsk}(k, f)$ to make explicit what is the key k they are generated from. We omit these details when they do not matter or are clear from the context.

2.1 Overview of Our FE Construction

Basic template of FE construction in prior works. We start with reviewing the basic template of FE construction in recent works [Agr19, AJL+19, JLMS19]. FE allows one to generate so-called functional secret key $\mathsf{fesk}(f)$ associated with a function f that decrypts an encryption of a plaintext $\boldsymbol{x}$, $\mathsf{fect}(\boldsymbol{x})$ to $f(\boldsymbol{x})$. Security ensures that beyond the evaluation of the function f on $\boldsymbol{x}$, nothing is revealed about $\boldsymbol{x}$. For constructing $i\mathcal{O}$, it suffices to have an FE scheme whose security is guaranteed against adversaries seeing only *a single functional secret key*, for a function with long output $f : \{0,1\}^n \to \{0,1\}^m$ and where the ciphertexts are *sublinearly-compact* in the sense that its size depends sublinearly in the output length m.

Towards this, the basic idea is encrypting the message using a Homomorphic Encryption scheme HE, which produces the ciphertext $\mathsf{hect}(\boldsymbol{s}, \boldsymbol{x})$, where $\boldsymbol{s}$ is the secret key of HE. It is possible to publicly evaluate homomorphically any function f directly on the ciphertext to obtain an so-called output ciphertext $\mathsf{hect}(\boldsymbol{s}, f(\boldsymbol{x})) \leftarrow \mathsf{HEEval}(\mathsf{hect}, f)$, that encrypts the output $f(\boldsymbol{x})$. Then, we use another *much simpler* FE scheme to decrypt $\mathsf{hect}(\boldsymbol{s}, f(\boldsymbol{x}))$ so as to reveal $f(\boldsymbol{x})$ and nothing more. Using this paradigm, the computation of the function f is delegated to HE, while the FE only computes the decryption of HE. This is motivated by the fact that HE for arbitrary functions can be built from standard assumptions, while existing FE schemes is either not compact, in the sense that the ciphertext grows with the output size of the functions [SS10b, GKP+13], or are limited to basic functions — namely, degree-2 polynomials at most, [BCFG17, Gay20] for the public-key setting, [Lin17, AS17] for the private-key setting[3]Furthermore, known HE schemes have very simple decryption — for most of them, it is simply computing an inner product, then rounding. That is, decryption computes $\langle \mathsf{hect}_f, \boldsymbol{s} \rangle = p/2 \cdot f(\boldsymbol{x}) + \boldsymbol{e}_f \pmod p$ for some modulus p, where $\boldsymbol{s}$ is the secret key of HE, and $\boldsymbol{e}_f$ is a small, polynomially bounded error (for simplicity, in this overview, we assume w.l.o.g that $f(\boldsymbol{x}) \in \{0,1\}$). While there are FE schemes that support computing inner products [ABDP15, ALS16], sublinearly compact FE that also computes the rounding are currently our of reach. Omitting this rounding would reveal $f(\boldsymbol{x})$, but also $\boldsymbol{e}_f$, which hurts the security of HE. Instead, we will essentially realize an approximate version of the rounding — thereby hiding the noise $\boldsymbol{e}_f$.

A natural approach to hide the noises $\boldsymbol{e}_f$ is to use larger, smudging noises. Since $\boldsymbol{e}_f$ depends on the randomness used by HEEnc, and the function f, the smudging noises must be fresh for every ciphertext. Hard-wiring the smudging noise in the ciphertext, as done in [AR17b], leads to non-succinct ciphertext, whose size grows linearly with the output size of the functions. Instead, we generate the smudging noises from a short seed, using a PRG. The latter must be simple enough to be captured by state of the art FE schemes.

Previous constructions use a weak pseudo-random generator, referred to as a noise generator NG, to generate many smudging noises $\boldsymbol{r} = \mathsf{NG}(\mathsf{sd})$ for hiding $\boldsymbol{e}_f$. To see how it works, suppose hypothetically that there is a noise generator computable by degree-2 polynomials. Then we can use $\mathsf{2FE}$, an FE scheme that support the generation of func-

[3]As mentioned in the introduction, partially hiding functional encryption allows to further strengthen the function class supported, by essentially adding computation on a public input, however computation on the private input is still limited to degree 2.

tional key for degree-2 polynomials, to compute $p/2 \cdot f(\boldsymbol{x}) + \boldsymbol{e}_f + \mathsf{NG}(\mathsf{sd})$, which reveals only $f(\boldsymbol{x})$ as desired. This gives a basic template of FE construction summarized below.

Basic Template of FE Construction (Intuition only, does not work)

$$\begin{array}{lcll} \mathsf{fesk}(f) \text{ contains} & : & & \mathsf{2fsk}(g) \\ \mathsf{fect}(\boldsymbol{x}) \text{ contains} & : & \mathsf{hect}(\boldsymbol{s}, \boldsymbol{x}), & \mathsf{2fct}(\boldsymbol{s}||\mathsf{sd}) \end{array}$$

The basic idea is using HE *with a one-time secret key* $\boldsymbol{s}$ *to perform the computation and using a simple FE for degree-2 polynomials,* 2FE*, to decrypt the output ciphertext and add a smudging noise generated via a noise generator* NG*. That is, we would like* $g(\boldsymbol{s}||\mathsf{sd}) = (p/2 \cdot f(\boldsymbol{x}) + \boldsymbol{e}_f + \mathsf{NG}(\mathsf{sd}))$*. However, there are many challenges to making this basic idea work.*

Unfortunately, to make the above basic idea work, we need to overcome a series of challenges. Below, we give an overview of the challenges, how we solve them using new tools, new techniques, and new assumptions, and how our solutions compare with previous solutions. In later subsections 2.2,2.3,2.4, we give more detail on our solutions.

Challenge 1: No Candidate Degree-2 Noise Generator. Several constraints are placed on the structure of the noise generators NG which renders their instantiation difficult.

- MINIMAL DEGREE. To use degree-2 FE to compute NG, the generator is restricted to have only degree 2 in the secret seed sd.
- SMALL (POLY-SIZED) OUTPUTS. Existing degree-2 FE are implemented using pairing groups: They perform the degree-2 computation in the exponent of the groups, and obtain the output in the exponent of the target group. This means the output $p/2 \cdot f(\boldsymbol{x}) + \boldsymbol{e}_f + \mathsf{NG}(\mathsf{sd})$ resides in the exponent, and the only way to extract $f(\boldsymbol{x}) \in \{0, 1\}$ is via brute force discrete logarithm to extract the whole $p/2 \cdot f(\boldsymbol{x}) + \boldsymbol{e}_f + \mathsf{NG}(\mathsf{sd})$. This in particular restricts NG to have polynomially bounded outputs.

Previous works [AJL+19, JLMS19] used new assumptions that combine LWE with constant-degree polynomials over the integers (see discussion in the introduction) to instantiate the noise generator. The resulting generator do not have exactly degree 2, but "close" to degree 2 in following sense:

Degree "2.5" Noise Generator: $\mathsf{NG}(\mathsf{pubsd}, \mathsf{privsd})$ is a polynomial in a public seed pubsd and a private seed privsd both of length n', and has polynomial stretch. The seeds are jointly sampled $(\mathsf{pubsd}, \mathsf{privsd}) \leftarrow \mathcal{D}_{\mathsf{sd}}$ from some distribution and pubsd is made publc. Degree 2.5 means that NG has constant degree in pubsd and degree 2 in privsd.

Previous degree-2.5 noise generators produce small integer outputs, and can only satisfy certain weak pseudo-randomness property (as opposed to standard pseudorandomness). To get a flavor, consider the fact that the outputs of previous candidates are exactly the outputs of some constant-degree polynomials computed over the integers. Individual

output elements are not uniformly distributed in any range, and two output elements that depend on the same seed element are noticably correlated. Hence, they are not pseudorandom or even pseudo-independent. In this work, our new assumption combines Learning With Binary Errors (LWBE) and constant-degree *Boolean* PRGs, and gives new degree-2.5 noise generators with *Boolean outputs* as follows:

- $\mathsf{pubsd} = \{\boldsymbol{c}_i = (\boldsymbol{a}_i, \boldsymbol{a}_i\boldsymbol{s} + e_i)\}_{i\in[n]}$: LWBE samples where $\boldsymbol{s}, \boldsymbol{a}_i \leftarrow \mathbb{Z}_p^{n^{0.5+\epsilon}},\ e_i \leftarrow \{0,1\}$.
- $\mathsf{privsd} = \otimes(\boldsymbol{s}||-1)^{\lceil \frac{d}{2} \rceil}$: tensoring $(\boldsymbol{s}||-1)$ for $\lceil \frac{d}{2} \rceil$ times.
- $\mathsf{PRG}(\mathsf{pubsd}, \mathsf{privsd}) = G(\cdots||e_i = \langle \boldsymbol{c}_i, (\boldsymbol{s}||-1)\rangle||\cdots) = G(\boldsymbol{e})$, where G is a constant degree Boolean PRG.

When the PRG G has sufficient stretch $m \geq n^{\lceil \frac{d}{2} \rceil \cdot (0.5+\epsilon)+\rho}$ for some constant $\epsilon, \rho > 0$, our new generator has polynomial stretch $m = |\mathsf{pubsd}||\mathsf{privsd}|^{1+\epsilon'}$ for some ϵ' depending on ϵ, ρ. Constant-degree Boolean PRGs are qualitatively different from constant-degree polynomials over the integers and have been extensively studied. Furthermore, our new assumption implies that the outputs of our generator are *pseudo-random* – in other words, we obtain a *degree-2.5 Boolean* PRG.

Not surprisingly, the stronger security property of degree-2.5 PRG lets us significantly simplify the construction and security proof.

Challenge 2: How to Evaluate Degree 2.5 Polynomials? To evaluate our degree-2.5 Boolean PRG, we need an FE scheme that is more powerful than 2FE. The notion of Partially-Hiding Functional Encryption PHFE, originally introduced by [GVW15] in the form of Partially Hiding Predicate Encryption (PHPE), fits exactly this task. As mentioned in introduction, PHFE strengthens the functionality of FE by allowing the ciphertext $\mathsf{phfct}(\boldsymbol{x}, \boldsymbol{y})$ to encode a public input $\boldsymbol{x}$, in addition to the usual private input $\boldsymbol{y}$. Decryption by a functional key $\mathsf{phfsk}(f)$ reveals $\boldsymbol{x}$ and $f(\boldsymbol{x}, \boldsymbol{y})$ and nothing else. The works of [AJL+19, JLMS19] constructed *private-key* PHFE for computing degree-2.5 polynomials (i.e., constant degree in $\boldsymbol{x}$ and degree 2 in $\boldsymbol{y}$) from pairing groups. (Like 2FE, the output is still computed in the exponent of the target group.) This suffices for evaluating degree-2.5 noise generator or PRG in the FE construction outlined above. The only drawback is that since PHFE is private-key, the resulting FE is also private-key.

In this work, we give a new construction of PHFE from pairing groups that is 1) public-key and 2) supports arithmetic NC^1 computation on the public input — more specifically, $f(\boldsymbol{x}, \boldsymbol{y}) = \langle g(\boldsymbol{x}), q(\boldsymbol{y})\rangle$ where g is computable by an arithmetic log-depth circuit and q is a degree-2 polynomial.

Theorem 2.1 (Public-key $(\mathsf{NC}^1, \text{deg-2})$-PHFE, Informal)**.** *There is a construction of a public-key PHFE for arithmetic* NC^1 *public computation and degree-2 private computation from standard assumptions over asymmetric pairing groups.*

This new construction allows us to obtain public key FE directly. Furthermore, our construction supports the most expressive class of functions among all known FE schemes from standard assumptions; we believe this is of independent interests.

Challenge 3: How to Ensure Integrity? Now that we have replaced 2FE with PHFE to compute degree-2.5 polynomials, the last question is how to ensure that PHFE decrypts only the right evaluated ciphertext hect_f (instead of any other ciphertext)? The function g we would like to compute via PHFE is $g(\boldsymbol{s}, \mathsf{pubsd}, \mathsf{privsd}) = \langle \mathsf{hect}_f, \boldsymbol{s} \rangle + \mathsf{NG}(\mathsf{pubsd}, \mathsf{privsd})$. The difficulty is that hect_f is unknown at key-generation time or at encryption time (as it depends on both f and $\mathsf{hect}(\boldsymbol{s}, \boldsymbol{x})$), and is too complex for PHFE to compute (as the homomorphic evaluation has high polynomial depth). To overcome this, we replace homormophic encryption with a *single-ciphertext* secret-key FE for polynomial size circuits with depth λ with *linear key generation*, denoted as 1LGFE, which has the following special structure.

Single Ciphertext FE with Linear Key Generation

$\mathsf{PPGen}(1^\lambda)$	:	generate public parameters pp
$\mathsf{Setup}(1^\lambda, \mathsf{pp})$	:	generate master secret key $\boldsymbol{s} \in \mathbb{Z}_p^\lambda$
$\mathsf{Enc}(\mathsf{pp}, \boldsymbol{s})$	:	generates a ciphertext 1LGFE.ct
$\mathsf{KeyGen}(\mathsf{pp}, \boldsymbol{s}, f)$	:	$\mathsf{pp}_f \leftarrow \mathsf{EvalPP}(\mathsf{pp}, f)$, $\boldsymbol{r} \leftarrow ([0, B-1] \cap \mathbb{Z})^m$, output f and secret key, $\mathsf{1LGFE.sk}(f) = \langle \mathsf{pp}_f, \boldsymbol{s} \rangle - \boldsymbol{r}$
$\mathsf{Dec}(\mathsf{1LGFE.ct}, (f, \mathsf{1LGFE.sk}))$	:	$\mathsf{1LGFE.ct}_f \leftarrow \mathsf{EvalCT}(\mathsf{1LGFE.ct}, f)$ output $\frac{q}{2}\boldsymbol{y} + \boldsymbol{e}_f + \boldsymbol{r} \leftarrow \mathsf{1LGFE.ct} - \mathsf{1LGFE.sk}$, $\lvert\boldsymbol{e}_f\rvert_\infty \leq B'$

The single-ciphertext FE has i) a key generation algorithm that is linear in the master secret key $\boldsymbol{s}$ and randomness $\boldsymbol{r}$, and ii) decryption first performs some computation on the ciphertext 1LGFE.ct *to obtain an intermediate ciphertext* $\mathsf{1LGFE.ct}_f$*, and then simply subtracts the secret key from* $\mathsf{1LGFE.ct}_f$*, and obtains the output $\boldsymbol{y}$ perturbed by a polynomially-bounded noise.*

We replace the ciphertext $\mathsf{hect}(\boldsymbol{s}, \boldsymbol{x})$ now with a ciphertext $\mathsf{1LGFE.ct}(\boldsymbol{s}, \boldsymbol{x})$ of 1LGFE. By the correctness and security of 1LGFE, revealing $\mathsf{1LGFE.sk}(f)$ only reveals the output $f(\boldsymbol{x})$. Hence, it suffices to use PHFE to compute the secret key. Thanks to the special structure of the key generation algorithm, this can be done in degree 2.5, using pseudoradnomness $\boldsymbol{r}$ expanded out via our degree-2.5 PRG. More concretely, PHFE computes the following degree-2.5 function g.

$$g(\boldsymbol{s}||\mathsf{pubsd}||\mathsf{privsd}) = \langle \mathsf{pp}_f, \boldsymbol{s} \rangle + \boldsymbol{r} = \mathsf{1LGFE.sk}(f), \qquad // \ g \text{ has degree 2.5}$$

$$\text{where } r_j = \sum_{k=0}^{\log B - 1} 2^k \mathsf{PRG}_{(j-1)\log B + k}(\mathsf{pubsd}, \mathsf{privsd}) \ .$$

One more technical caveat is that known pairing-based PHFE schemes actually compute the secret key 1LGFE.sk in the exponent of a target group element, which we denote by $[\mathsf{1LGFE.sk}]_T$, where for any exponent $a \in \mathbb{Z}_p$, $[a]_T = g_T^a$ for a generator g_T. Thanks to the special structure of the decryption algorithm of 1LGFE — namely, it is linear in 1LGFE.sk — these group elements are sufficient for decryption. A decryptor can first compute

$\mathsf{1LGFE.ct}_f$ from $\mathsf{1LGFE.ct}(\boldsymbol{s}, \boldsymbol{x})$ and f in the clear, then perform the decryption by subtracting $[\mathsf{1LGFE.ct}_f - \mathsf{1LGFE.sk}]_T$ in the exponent. This gives $[p/2 \cdot f(\boldsymbol{x}) + \boldsymbol{e}_f + \boldsymbol{r}]_T$, whose exponent $p/2 \cdot f(\boldsymbol{x}) + \boldsymbol{e}_f + \boldsymbol{r}$ can be extracted by enumrating all possible $\boldsymbol{e}_f + \boldsymbol{r}$, which are of polynomial size, and $f(\boldsymbol{x}) \in \{0, 1\}$.

Our single-ciphertext FE with linear key generation is essentially the same notion as that of Special Homomorphic Encryption (SHE) used in [Agr19, LM18]. SHE are homomorphic encryption with a special decryption equation $\mathsf{hect}_f - \langle \mathsf{pp}_f, \boldsymbol{s} \rangle = p/2 \cdot f(\boldsymbol{x}) + \boldsymbol{e}_f$ where pp_f (as in 1LGFE) can be computed efficiently from public parameters pp and f. We think it is more accurate to view this object as a functional encryption scheme, since what the special decryption equation gives is exactly a functional key $\langle \mathsf{pp}_f, \boldsymbol{s} \rangle + \boldsymbol{r}$ where $\boldsymbol{r}$ are smudging noises for hiding $\boldsymbol{e}_f$ to guarantee that only $p/2 \cdot f(\boldsymbol{x})$ is revealed.

Viewing this through the lens of FE brought us a significant benefit. Previous works constructed SHE by modifying the Brakerski-Vankuntanathan FHE scheme [BV11], but are limited to supporting NC^1 computations based on RLWE [AR17b], and NC_0 based on LWE [AR17b, LM18]. Instead, the FE lens led us to search for ideas in the predicate encryption literature. We show how to construct 1LGFE for polynomial sized circuits with depth bounded by λ from LWE by modifying the predicate encryption scheme of [GVW15]. This new construction allowed us to construct FE for polynomial sized circuits with depth bounded by λ directly without invoking any bootstrapping theorem from weaker function classes. More generally, we our 1LGFE can handle polynomial-sized circuits of any depth d and the ciphertext size grows polynomially with the input length n and the depth d.

Theorem 2.2 (1LGFE from LWE, informal)**.** *There is a construction of a single-key single-ciphertext secret-key FE for polynomial size circuits with linear key generation as described above, from LWE. The ciphertext size is* $\mathsf{poly}(\lambda, n, d)$ *where λ is the security parameter, n the input length, and d the depth of the circuit (independent of the circuit size* size *and the output length ℓ). The construction is parameterized with a positive constant $\epsilon > 0$ and satisfies single-key single-ciphertext ϵ-simulation security as defined in Definition 4.3.*

Above, the notion of ϵ-simulation security is a weakening of the standard simulation security of FE, which we discuss below.

Putting Pieces Together In summary, putting all the pieces together, our construction of FE for polynomial size circuits with depth λ is depicted below. Comparing with previous constructions, it enjoys several features: 1) it is public key, 2) it can be based on the polynomial-hardness of underlying assumptions, 3) it has simpler proofs (e.g., no bootstrapping theorem).

Our FE Construction

$\mathsf{fesk}(f)$ contains	:		$\mathsf{phfsk}(g)$
$\mathsf{fect}(\boldsymbol{x})$ contains	:	$\mathsf{1LGFE.ct}(\boldsymbol{s}, \boldsymbol{x})$	$\mathsf{phfct}(\boldsymbol{s}\|\|\mathsf{pubsd}\|\|\mathsf{privsd})$
$\mathsf{FEDec}(\mathsf{fect}, (f, \mathsf{fesk}))$	:	$[\mathsf{1LGFE.sk}]_T \leftarrow \mathsf{PHFEDec}(\mathsf{phfct}, \mathsf{phfsk})$	
		$\mathsf{1LGFE.ct}_f \leftarrow \mathsf{EvalCT}(\mathsf{1LGFE.ct}, f)$	
		$[\boldsymbol{y} + \boldsymbol{e}_f + \boldsymbol{r}]_T = \mathsf{1LGFE.ct}_f - [\mathsf{1LGFE.sk}]_T$	
		extract $\boldsymbol{y} + \boldsymbol{e}_f + \boldsymbol{r}$ and round to recover $\boldsymbol{y}$	

The basic idea is using PHFE *to compute a* 1LGFE *secret key* $\mathsf{1LGFE.sk}(f)$ *in the exponent of the target group, and then decrypting the ciphertext* $\mathsf{1LGFE.ct}(\boldsymbol{s}, \boldsymbol{x})$ *to reveal* $f(\boldsymbol{x})$ *only.*

ϵ-Simulation Security. The only aspect of our construction that we have not discussed explicitly is the security guarantee achieved by our 1LGFE scheme. In particular, it does not achieve the standard notion of indistinguishability or simulation security of FE; instead, it achieves a weaker notion called ϵ-simulation security, where ϵ is a constant in $(0, 1)$. The weaker security guarantee stems from the fact that the pseudorandom smudging error $\boldsymbol{r}$ used in the secret key $\mathsf{1LGFE.sk}(f)$ is only of polynomial magnitude, and therefore reveals nontrivial information about the error $\boldsymbol{e}_f$ in the output ciphertext $\mathsf{1LGFE.ct}_f$. Fortunately, we can still show that $\boldsymbol{r}$ hides $\boldsymbol{e}_f$ except for some $1/\operatorname{poly}$ probability, which allows us to prove that the public key, secret key, and ciphertext can be simulated using only the output of the computation with ϵ probability, and with probability $1 - \epsilon$, all bets are off and the encrypted input may be revealed — hence the name ϵ-simulation security. This weak security guarantee is then inherited by the FE construction described above.

Therefore, we need to amplify security. But amplifying ϵ-simulation security to full security turns out to be easy. We are able to achieve this amplification in a simple and direct construction (see Section 5) that avoids any need to use hard-core measures or any other such sophisticated and/or delicate amplification technology.

2.2 Instantiating Our Assumption

To instantiate our assumption, we need to *choose a degree d PRG with a stretch more than* $n^{\lceil \frac{d}{2} \rceil \cdot (0.5+\delta)+\rho}$. The good news is that there is a rich body of literature on both ingredients of our assumption that existed way before our work to guide the choice. Binary LWE was first considered by [AG11] and then by [MP13, ACF+15, BGPW16, CTA19]. Goldreich PRGs have been studied even before that. There are many prior works spanning areas in computer science devoted to cryptanalysis of these objects from lattice reduction algorithms and symmetric-key cryptanalysis, to algebraic algorithm tools such as the Gröbner basis algorithm and attacks arising from the Constraint Satisfaction Problem and Semi-Definite Programming literature. Guided by them, we list three candidates below. In Section A.4, we survey many of these attack algorithms, and we compute approximate running times of the attacks arising out of these algorithms on our candidates. For the parameters we choose, all those attacks are subexponential time.

A Goldreich's PRG G is defined by a predicate $P : \{0,1\}^{\ell'} \to \{0,1\}$, where ℓ' is the locality of the PRG, and a bipartiate input-output dependency graph Λ, which specifies for every output index $j \in [m]$, the subset $\Lambda(j) \subset [n]$ of input indexes of size ℓ' it depends on – the j'th output bit is simply set to $G(j) = P(\Lambda(j))$. Hence the degree of the PRG G is identical to the degree of the predicate P. Usually, the input-output dependency graph Λ is chosen at random, and the non-trivial part lies in choosing the predicate P.

Instantiation 1. The first instantiation is that of the predicate XORMAJ, which is a poplular PRG predicate [AL16, CDM+18].

$$\mathsf{XORMAJ}_{\ell,\ell}(x_1 \ldots, x_{2\ell}) = \oplus_{i\in[\ell]} x_i \oplus \mathsf{MAJ}(x_{\ell+1}, \ldots, x_{2\ell}).$$

The predicate above has a degree of $2 \cdot \ell$; thus, our construction require expansion $m > n^{\frac{\ell}{2}+\ell\delta+\rho}$. The predicate is $\ell + 1$ wise independent and thus it provably resists subexponential time SoS refutation attacks when $m(n) \leq n^{\frac{\ell+1}{2}-c}$ for $c > 0$ [KMOW17]. All other known attacks that we consider and even the algebraic attacks when instantiated in our combined assumption require subexponential time. We refer the reader to Section A.4 for a detailed discussion.

Instantiation 2. An slightly unsatisfactory aspect of the XORMAJ predicate is that the lower bound on the stretch of the PRG instantiated by XORMAJ for it to be useful in our FE construction is $> n^{\frac{\ell}{2}+\delta'}$, whereas the upper bound on the stretch to withstand existing attacks is very close $\leq n^{\frac{\ell+1}{2}-c}$, leaving only a tiny margin to work with. This motivates us to we consdier predicates with degree lower than the locality. One such predicate was analyzed in [LV17] for stretch upto $n^{1.25-c}$ for $c > 0$:

$$\mathsf{TSPA}(x_1, x_2, x_3, x_4, x_5) = x_1 \oplus x_2 \oplus x_3 \oplus ((x_2 \oplus x_4) \wedge (x_3 \oplus x_5)).$$

What is nice about this predicate is that, it has locality 5 but only degree 3; thus, our construction only require expansion $m > n^{\lceil \frac{3}{2} \rceil (0.5+\epsilon)+\rho} = n^{1+2\epsilon+\rho}$. In [LV17], it was proven that the PRG istantiated with TSPA resists subexponential time $\mathbb{F}_2$ linear and SoS attacks. We present analysis against other attacks in Section A.4, all taking subexponential time.

Instantiation 3. We present a degree reduction transformation that takes as input a non-linear predicate $g : \{0,1\}^k \to \{0,1\}$ and constructs a predicate P.

$$\mathsf{P}_g(x_1 \ldots, x_{2k+1}) = \oplus_{i\in[k+1]} x_i \oplus g(x_{k+2} \oplus x_2, \ldots, x_{2k+1} \oplus x_{k+1}).$$

We show in Section A.3, that the predicate above has a locality of $2k+1$ but a degree equal to $k + 1$; thus, our construction requires expansion $m > n^{\lceil \frac{k+1}{2} \rceil (0.5+\epsilon)+\rho}$. The predicate is also $k + 1$ wise independent. We show that all known attacks run in subexponential time even when the stretch is bounded by $m \leq n^{\frac{k+1}{2}-\delta}$ for some $\delta > 0$. Thanks to the gap between the locality and degree, we now have a very large margin between the lower

and upper bounds on the stretch. Hence, our work motivates the interesting question of studying such predicates.

Please refer to Table 1 for a summary of attacks on all these predicates as well as the combined assumption.

2.3 Single Ciphertext Functional Encryption with Linear Key Generation

We describe our construction of a single-ciphertext (secret-key) FE scheme for all polynomial-sized circuits with depth bounded by λ, that have the simple structure outlined in Section 2, denoted as ϵ-1LGFE, from LWE. In particular, the key generation and decryption algorithms have the following form, where $\boldsymbol{s}$ is the master secret key and pp is the public parameters.

$$
\begin{array}{lcl}
\mathsf{KeyGen}(\mathsf{pp}, \boldsymbol{s}, f) & : & \mathsf{pp}_f \leftarrow \mathsf{EvalPP}(\mathsf{pp}, f)\ ,\ \boldsymbol{r} \leftarrow ([0, B-1] \cap \mathbb{Z})^m, \\
 & & \text{output } f \text{ and secret key } \epsilon\text{-}\mathsf{1LGFE.sk}(f) = \langle \mathsf{pp}_f, \boldsymbol{s} \rangle - \boldsymbol{r} \\
\mathsf{Dec}(\epsilon\text{-}\mathsf{1LGFE.ct}, (f, \epsilon\text{-}\mathsf{1LGFE.sk})) & : & \epsilon\text{-}\mathsf{1LGFE.ct}_f \leftarrow \mathsf{EvalCT}(\epsilon\text{-}\mathsf{1LGFE.ct}, f) \\
 & & \text{output } \frac{q}{2}\boldsymbol{y} + \boldsymbol{e}_f + \boldsymbol{r} \leftarrow \epsilon\text{-}\mathsf{1LGFE.ct} - \epsilon\text{-}\mathsf{1LGFE.sk},\ |\boldsymbol{e}_f|_\infty \leq B'
\end{array}
$$

Importantly, decryption recovers a perturbed output where the error $\boldsymbol{e}_f + \boldsymbol{r}$ is polynomially bounded. As mentioned before, this object is essentially the same as the notion of Special Homomorphic Encryption (SHE) in the literature [AR17b, LM18]. Previous SHE schemes are constructed by modifying existing homomorphic encryption schemes of [BV11, BGV12]. These constructions are recursive and quite complex, and the overhead due to recursion prevents them from supporting computations beyond NC^1. In this work, viewing through the FE lens, we search the literature of predicate encryption, and show how to modify the predicate encryption scheme of [GVW15] (GVW) to obtain single-ciphertext FE with the desired structure. The GVW predicate encryption provide us with a single-ciphertext encryption scheme with the following properties:

- The public parameter generation algorithm PPGen samples a collection of random LWE matrices $\boldsymbol{A}_i, \boldsymbol{B}_j \leftarrow \mathbb{Z}_p^{n \times m}$, and sets the public parameters to $\mathsf{pp} = (\{\boldsymbol{A}_i\}, \{\boldsymbol{B}_j\})$.

- The setup algorithm Setup samples a master secret key constaining an LWE secret $\boldsymbol{s} \leftarrow \chi^n$ drawn from the noise distribution χ.

- The encryption algorithm to encrypt $\boldsymbol{x}$, generates a ciphertext $\mathsf{hect}(\boldsymbol{x})$ containing two sets of LWE samples of form $\boldsymbol{c}_i = \boldsymbol{s}^T\boldsymbol{A}_i + \widehat{\boldsymbol{x}}_i\boldsymbol{G} + \boldsymbol{e}_i$ and $\boldsymbol{d}_j = \boldsymbol{s}^T\boldsymbol{B}_j + \widehat{k}_j\boldsymbol{G} + \boldsymbol{e}'_j$, where $\boldsymbol{G} \in \mathbb{Z}_p^{n \times m}$ is the gadget matrix, vk is a freshly sampled secret key of a homomorphic encryption scheme, and $\boldsymbol{e}_i, \boldsymbol{e}'_j \leftarrow \chi^m$ are LWE noises. Furthermore, $\widehat{\boldsymbol{x}}_i$ is the i'th bit of a homomorphic encryption ciphertext of $\boldsymbol{x}$ under key $\boldsymbol{k}$.

- The predicate encryption scheme of [GVW15] provides two homomorphic procedures: The EvalCT procedure homomorphically evaluate f on $\{\boldsymbol{c}_i, \boldsymbol{A}_i\}$ and $\{\boldsymbol{d}_j, \boldsymbol{B}_j\}$ to obtain $\boldsymbol{c}_f$, and the EvalPP seperately homormorphically evaluates on $\{\boldsymbol{A}_i\}$ and $\{\boldsymbol{B}_i\}$ to obtain $\boldsymbol{A}_f$.

- The homomorphic evaluation outcomes $\boldsymbol{c}_f, \boldsymbol{A}_f$, has the property that the first coordinate $\boldsymbol{c}_{f,1}$ of $\boldsymbol{c}_f$ and the first column $\boldsymbol{A}_{f,1}$ of $\boldsymbol{A}_f$ satisfy the special decryption equation.

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

The above described encryption scheme almost gives the FE scheme we want except for the issue that it has super-polynomially large decryption error e_f. Thus, we turn to reducing the norm of the decryption error, by applying the rounding (or modulus switch) technique in the HE literature [BGV12]. Namely, to reduce the error norm by a factor of p/q for a $q < p$, we multiply $\boldsymbol{c}_{f,1}$ and $\boldsymbol{A}_{f,1}$ with q/p over the reals and then round to the nearest integer component wise. The rounding results satisfy the following equation

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

where the rounding error error is bounded by $|\mathsf{hesk}|_1 + O(1)$, which is polynomially bounded as the secret is sampled from the LWE noise distribution instead of uniformly.

We are now ready to instantiate the FE scheme we want. It uses the same public parameter generation, setup, and encryption algorithm. Now to generate a functional key for f, it first computes $A_f \leftarrow \mathsf{EvalPP}(\{\boldsymbol{A}_i\}, \{\boldsymbol{B}_j\})$ and sets $\mathsf{pp}_f = \lfloor \frac{q}{p} \boldsymbol{A}_{f,1} \rceil$, and then outputs a functional key $\epsilon\text{-}\mathsf{1LGFE.sk} = \langle \mathsf{pp}_f \boldsymbol{s} \rangle - \boldsymbol{r}$ where $\boldsymbol{r}$ is a random vector of smudging noises of sufficiently large but still polynomially bounded magnitude. The decryption algorithm decrypts a ciphertext $\epsilon\text{-}\mathsf{1LGFE.ct} = (\{\boldsymbol{c}_i\}, \{\boldsymbol{d}_j\})$ using a functional key $\epsilon\text{-}\mathsf{1LGFE.sk}$ as follows: It first computes $\boldsymbol{c}_f \leftarrow \mathsf{EvalPP}(\{\boldsymbol{A}_i, \boldsymbol{c}_i\}, \{\boldsymbol{B}_j, \boldsymbol{d}_j\})$, and sets $\epsilon\text{-}\mathsf{1LGFE.ct}_f = \lfloor \frac{q}{p} \boldsymbol{c}_{f,1} \rceil$, it then subtracts $\epsilon\text{-}\mathsf{1LGFE.sk}$ from it, yielding $f(\boldsymbol{x})\lfloor q/2 \rceil + \lfloor \frac{q}{p} e_f \rceil + \mathsf{error} + \boldsymbol{r}$ as desired.

2.4 Overview: Our $(\mathsf{arith}\text{-}\mathsf{NC}^1, \mathbf{deg}\text{-}2)$ Partially Hiding Functional Encryption

We construct 1-key PHFE with fully compact ciphertext of size linear in the input length n, for functions $F(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z})$ of the following form, from standard assumptions on asymmetric pairings. F maps three vectors $\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z} \in \mathbb{Z}_p^n$ to a (potentially longer) output vector in $\mathbb{Z}_p^m$ (our construction can handle any (polynomial) unbounded m), where each output element is computed by a function $f = F_k$ for $k \in [m]$ as the following matrix product:

$$f(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}) = f^0 f^1(\boldsymbol{x}) f^2(\boldsymbol{x}) \cdots f^\ell(\boldsymbol{x}) f^{\ell+1}(\boldsymbol{y} \otimes \boldsymbol{z}), \tag{1}$$

where $f^0 \in \mathbb{Z}_p^{1 \times w}$, for all $i \in [\ell]$, f^i takes as input a vector $\boldsymbol{x} \in \mathbb{Z}^n$ and outputs a matrix $f^i(\boldsymbol{x}) \in \mathbb{Z}_p^{w \times w}$, the function $f^{\ell+1}$ takes as input the vector $\boldsymbol{y} \otimes \boldsymbol{z} \in \mathbb{Z}^{n^2}$ and outputs a vector $f^{\ell+1}(\boldsymbol{y} \otimes \boldsymbol{z})$. Here, w denotes the width of the branching program, ℓ its length. The function f^i are affine, for all $i \in [\ell + 1]$. Such functions f can express computations such as $L(g(\boldsymbol{x}), \boldsymbol{y} \otimes \boldsymbol{z})$, where g is a Boolean circuit in NC^1, and L is a bilinear function, with degree one in $\boldsymbol{y} \otimes \boldsymbol{z}$.

Computing degree-2 polynomials on the private inputs.

Roughly speaking, we encrypt the private inputs $\boldsymbol{y}$ and $\boldsymbol{z}$ using encryption schemes with homomorphic properties that lets users manipulate the ciphertexts to obtain a new ciphertext, which encrypts the value $f^{\ell+1}(\boldsymbol{y} \otimes \boldsymbol{z})$, under a public key $\mathsf{pk}_{f^{\ell+1}}$ that depends on the function $f^{\ell+1}$. This manipulation can be performed publicly for arbitrary linear function $f^{\ell+1}$. At this point, providing the secret key associated to $\mathsf{pk}_{f^{\ell+1}}$ would reveal the value $f^{\ell+1}(\boldsymbol{y} \otimes \boldsymbol{z})$, and nothing else about the private inputs $\boldsymbol{y}$, $\boldsymbol{z}$. Otherwise stated, this would constitute a valid functional encryption scheme for degree-2 polynomials.

We implement this paradigm using cyclic groups $\mathbf{G}_1$, $\mathbf{G}_2$, $\mathbf{G}_T$ equipped with a pairing $e : \mathbf{G}_1 \times \mathbf{G}_2 \to \mathbf{G}_T$, and respectively generated by g_1, g_2, and $e(g_1, g_2)$. For any exponent $a \in \mathbb{Z}_p$, we denote by $[a]_T = e(g_1, g_2)^a \in \mathbf{G}_T$. To encrypt $\boldsymbol{y}$ and $\boldsymbol{z}$, we make generic use of a function-hiding inner product FE: the encryption of $\boldsymbol{y}$ comprises $\mathsf{IPFE.Enc}\begin{pmatrix} g_1^{y_i} \\ g_1^{r \cdot \alpha_i} \end{pmatrix}$ for all coordinates of $\boldsymbol{y}$, where $g_1^{\alpha_i}$ is a random group elements from $\mathbf{G}_1$ that is part of the public key, $r \leftarrow_{\mathrm{R}} \mathbb{Z}_p$ is some fresh randomness, sampled at encryption time, and $\mathsf{IPFE.Enc}$ is the encryption algorithm of IPFE. The encryption of $\boldsymbol{z}$ comprises $\mathsf{IPFE.KeyGen}\begin{pmatrix} g_2^{z_j} \\ g_2^{\beta_j} \end{pmatrix}$ for all coordinate of $\boldsymbol{z}$, where $g_2^{\beta_j}$ is a random group elements from $\mathbf{G}_2$ that is part of the public key, and $\mathsf{IPFE.KeyGen}$ is the key generation algorithm of IPFE. Correctness of IPFE yields the products $[y_i z_j + r\alpha_i\beta_j]_T$ for all $i, j \in [n]$. Because IPFE is secure and function-hiding, these products are the only information revealed on the private inputs $\boldsymbol{y}$ and $\boldsymbol{z}$. It is possible to compute for any linear function $f^{\ell+1}$ the elements: $[f^{\ell+1}(\boldsymbol{y}\otimes\boldsymbol{z})+rf^{\ell+1}(\alpha\otimes\beta)]_T$, which can be seen as an encryption of the value $f^{\ell+1}(\boldsymbol{y} \otimes \boldsymbol{z})$ under the public key $\mathsf{pk}_{f^{\ell+1}} = [f^{\ell+1}(\alpha \otimes \beta)]_T$. Because the parameters of the scheme IPFE are generated freshly during the encryption, even if IPFE is private-key —this is necessary for all function-hiding FE— the PHFE is public-key.

Computing branching programs on the public input.

We want to additionally force a specific computation on the public input $\boldsymbol{x} \in \mathbb{Z}^n$ before decryption. To do so, we produce re-encryption tokens, each of which computes one step of the matrix branching program directly on the ciphertext. That is, the token associated with the i-th product transform an encryption of $f^{i+1}(\boldsymbol{x}) \cdots f^{\ell}(\boldsymbol{x}) f^{\ell+1}(\boldsymbol{y} \otimes \boldsymbol{z})$ under $\mathsf{pk}_{f^{i+1} \cdots f^{\ell+1}}$ into an encryption $f^{i}(\boldsymbol{x}) \cdots f^{\ell}(\boldsymbol{x}) f^{\ell+1}(\boldsymbol{y} \otimes \boldsymbol{z})$ under $\mathsf{pk}_{f^{i} \cdots f^{\ell+1}}$, which we denote by ct_i. Finally, we release the secret key associated with the public key $\mathsf{pk}_{f^{0} \cdots f^{\ell+1}}$. To recover a meaningful information on the encrypted data, decryption is forced to perform the computation that precisely corresponds to the function $f^1 \cdots f^{\ell+1}$ encoded in the secret key.

The challenge is to realize these re-encryptions without blowing up the size of the ciphertext exponentially with the length ℓ. Concretely, the public keys will be of the form $\mathsf{pk}_{f^{i} \cdots f^{\ell+1}} = [f^{i}(\boldsymbol{u}_i) \cdots f^{\ell}(\boldsymbol{u}_\ell) f^{\ell+1}(\alpha \otimes \beta)]_T$, where the vectors $\boldsymbol{u}_i \leftarrow_{\mathrm{R}} \mathbb{Z}_p^n$ are part of the master secret key. These keys encode the last $\ell - i$ steps of the computation. Crucially, these keys do not grow with the length of the branching program, only its width. So is the case of the corresponding re-encryptions: we can handle polynomially large length efficiently.

The i-th re-encryption token is of the form: $[r(f^i(\boldsymbol{u}_i) - f^i(\boldsymbol{x}))f^{i+1}(\boldsymbol{u}_{i+1}) \cdots f^{\ell+1}(\alpha \otimes \beta)]_T$, which allows the decryption to transition from ct_{i-1} to ct_i. Ultimately, the final ciphertext $\mathsf{ct}_\ell = [f^0 f^1(\boldsymbol{x}) \cdots f^{\ell+1}(\boldsymbol{y} \otimes \boldsymbol{z}) + r f^0 f^1(\boldsymbol{u}_1) \cdots f^{\ell+1}(\alpha \otimes \beta)]_T$, is obtained. To decrypt it, we simply need a mechanism to recover the mask $[r f^0 f^1(\boldsymbol{u}_1) \cdots f^{\ell+1}(\alpha \otimes \beta)]_T$. Providing $[r]_1$ on the encryption side, and $[f^0 f^1(\boldsymbol{u}_1) \cdots f^{\ell+1}(\alpha \otimes \beta)]_2$ as the functional secret key would already give a scheme secure in the generic-group model (and idealized model that captures attacks that do not rely on the algebraic structure of the underlying group). To obtain security from standard assumptions, we encrypt $[r]_1$ using an inner-product FE. The functional key is the inner product FE key associated with the value $[f^0 f^1(\boldsymbol{u}_1) \cdots f^{\ell+1}(\alpha \otimes \beta)]_2$. This way, decrypting the inner-product FE yields the mask to decrypt the PHFE. Note that the function is described as $[f^0 f^1(\boldsymbol{u}_1) \cdots f^{\ell+1}(\alpha \otimes \beta)]_2$ in $\mathbf{G}_2$, and not in $\mathbb{Z}$; revealing the value in $\mathbb{Z}$ would be detrimental for the security of the PHFE.

Remains to find a way to generate these re-encryption tokens. To do so, we provide an encoding of the public input $\boldsymbol{x}$ as part of the PHFE ciphertext — note that we choose the word encoding rather than encryption, since the input $\boldsymbol{x}$ must not be hidden. This encoding is used with the functional secret key to produce the tokens. We leverage the simple structure of each computational step of the branching program. Namely, we use the fact that all the functions f^i are affine. Thus, we can use an inner-product FE encryption to generate the tokens. The encoding of $\boldsymbol{x}$ is an inner-product FE encryption of $[r, r\boldsymbol{x}]_1$, and the keys are associated with the appropriate functions depending on the f^i and the vectors $[\boldsymbol{u}_i]_2$, $[\alpha]_2$, $[\beta]_2$. The challenging part is to prove security even when the values $[\boldsymbol{u}_i]_2$, $[\alpha]_2$, $[\beta]_2$ are revealed. Indeed, such is the case when using a vanilla inner-product FE, as opposed to function-hiding FE, where these values would be hidden, but which would intrinsically be private-key.

Putting things together.

Each PHFE ciphertext contains $\mathsf{IPFE.Enc}\begin{pmatrix} g_1^{y_i} \\ g_1^{r \cdot \alpha_i} \end{pmatrix}$ and $\mathsf{IPFE.KeyGen}\begin{pmatrix} g_2^{z_j} \\ g_2^{\beta_j} \end{pmatrix}$ for all $i, j \in [n]$, from which can be computed the encryption of $f^{\ell+1}(\boldsymbol{y} \otimes \boldsymbol{z})$ under an associated public key $\mathsf{pk}_{f^{\ell+1}}$, for all linear functions $f^{\ell+1}$. The scheme IPFE is function-hiding, and is generated freshly by the encryption. The PHFE ciphertext also contains another inner-product FE encryption of the values $[r, r \cdot \boldsymbol{x}]_1$. These are used with functional secret keys associated with f^i, $[\boldsymbol{u}_i]_2$, $[\alpha]_2$ and $[\beta]_2$, to generate tokens. The latter transform the encryption of $f^{\ell+1}(\boldsymbol{y} \otimes \boldsymbol{z})$ into and encryption of $f(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z})$ under a public key that encodes the matrix branching program. This transformation is performed step by step. At last, the mask of the form $[r f^0 f^1(\boldsymbol{u}_1) \cdots f^\ell(\boldsymbol{u}_\ell) f^{\ell+1}(\alpha \otimes \beta)]_T$ is recovered exactly as the tokens, using the inner-product FE encryption of $[r]_1$ with a functional key associated with $[f^0 f^1(\boldsymbol{u}_1) \cdots f^{\ell+1}(\alpha \otimes \beta)]_2$.

3 Preliminaries

In this section, we describe preliminaries that are useful for rest of the paper. We denote the security parameter by λ. For any distribution $\mathcal{X}$, we denote by $x \leftarrow \mathcal{X}$ (or $x \leftarrow_{\mathrm{R}} \mathcal{X}$) the

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

By $\approx_c$ we denote the standard polynomial time computational indistinguishability. We say that two ensembles $\mathcal{X} = \{\mathcal{X}_\lambda\}_{\lambda\in\mathbb{N}}$ and $\mathcal{Y} = \{\mathcal{Y}_\lambda\}_{\lambda\in\mathbb{N}}$ are $(s(\lambda), \epsilon(\lambda))-$ indistinguishable if for every adversary $\mathcal{A}$ (modeled as a circuit) of size bounded by $s(\lambda)$ it holds that: $\left|\Pr_{x\leftarrow\mathcal{X}_\lambda}[\mathcal{A}(1^\lambda, x) = 1] - \Pr_{y\leftarrow\mathcal{Y}_\lambda}[\mathcal{A}(1^\lambda, y) = 1]\right| \leq \epsilon(\lambda)$ for every sufficiently large $\lambda \in \mathbb{N}$.

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

Throughout, when we refer to polynomials in security parameter, we mean constant degree polynomials that take positive value on non negative inputs. We denote by $\mathsf{poly}(\lambda)$ an arbitrary polynomial in security parameter satisfying the above requirements of non-negativity.

Pairing groups.

Throughout the paper, we use a sequence of asymmetric prime-order pairing groups:

$$\mathcal{G} = \{(p_\lambda,\, \mathbf{G}_{\lambda,1}, \mathbf{G}_{\lambda,2}, \mathbf{G}_{\lambda,T},\, P_{\lambda,1}, P_{\lambda,2}, P_{\lambda,T},\, e_\lambda)\}_{\lambda\in\mathbb{N}},$$

where for all $s \in \{1, 2, T\}$, $(\mathbf{G}_{\lambda,s}, +)$ is an cyclic group (for which we use additive notation) of order $p_\lambda = 2^{\lambda^{\Theta(1)}}$. $\mathbf{G}_{\lambda,1}$ and $\mathbf{G}_{\lambda,2}$ are generated by $P_{\lambda,1}$ and $P_{\lambda,2}$ respectively, and $e : \mathbf{G}_{\lambda,1} \times \mathbf{G}_{\lambda,2} \to \mathbf{G}_T$ is a non-degenerate bilinear map, that is, satisfying $e_\lambda(aP_{\lambda,1}, bP_{\lambda,2}) = abP_T$ for all integers $a, b \in \mathbb{Z}_p$, where $P_T = e(P_{\lambda,1}, P_{\lambda,2})$ is a generator of $\mathbf{G}_{\lambda,T}$.

Remark 3.1. We require the group operations as well as the pairing operation to be efficiently computable. The rest of the paper will refer to this sequence of bilinear pairing groups, and the corresponding sequence of prime orders of the groups $\{p_\lambda\}_{\lambda\in\mathbb{N}}$.

When clear from context, we omit the subscript λ. We also use implicit representation of group elements. That is, for $s \in \{1, 2, T\}$ and $a \in \mathbb{Z}_p$, define $[a]_s = aP_s \in \mathbf{G}_s$ as the implicit representation of a in G_s. More generally, for a matrix $\mathbf{A} = (a_{ij}) \in \mathbb{Z}_p^{n\times m}$ we define $[\mathbf{A}]_s$ as the implicit representation of $\mathbf{A}$ in $\mathbf{G}_s$:

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

Given $[a]_1$ and $[b]_2$, one can efficiently compute $[a \cdot b]_T$ using the pairing e. For matrices $\mathbf{A}$ and $\mathbf{B}$ of matching dimensions, define $e([\mathbf{A}]_1, [\mathbf{B}]_2) := [\mathbf{AB}]_T$. For any matrix $\mathbf{A}, \mathbf{B} \in \mathbb{Z}_p^{n\times m}$, any group $s \in \{1, 2, T\}$, we denote by $[\mathbf{A}]_s + [\mathbf{B}]_s = [\mathbf{A} + \mathbf{B}]_s$.

For any prime p, we define the following distribution. The DDH_p distribution over $\mathbb{Z}_p^2$: Sample $a \leftarrow_{\text{R}} \mathbb{Z}_p$, output $\boldsymbol{a} := \binom{1}{a}$. The DLIN_p distribution over $\mathbb{Z}_p^{3\times 2}$: $a, b \leftarrow_{\text{R}} \mathbb{Z}_p$, outputs $\mathbf{A} := \begin{pmatrix} a & 0 \\ 0 & b \\ 1 & 1 \end{pmatrix}$.

Definition 3.1 (DDH assumption)**.** *For any adversary $\mathcal{A}$, any sequence of asymmetric prime-order pairing groups $\mathcal{G}$, any $s \in \{1, 2, T\}$ and any security parameter λ, let*

$$\mathsf{adv}^{\mathsf{DDH}}_{\mathcal{G},s,\mathcal{A}}(\lambda) := |\Pr[1 \leftarrow \mathcal{A}(1^\lambda, [\boldsymbol{a}]_s, [\boldsymbol{a}r]_s)] - \Pr[1 \leftarrow \mathcal{A}(1^\lambda, [\boldsymbol{a}]_s, [\boldsymbol{u}]_s)]|,$$

where the probabilities are taken over $\boldsymbol{a} \leftarrow_R \mathsf{DDH}_{p_\lambda}$, $r \leftarrow_R \mathbb{Z}_{p_\lambda}$, $\boldsymbol{u} \leftarrow_R \mathbb{Z}^2_{p_\lambda}$, and the random coins of $\mathcal{A}$. Note that the adversary $\mathcal{A}$ also gets a description of the groups $\mathbf{G}_{\lambda,1}$, $\mathbf{G}_{\lambda,2}$, $\mathbf{G}_{\lambda,3}$, and the bilinear map e_λ.

We say DDH holds for a group $\mathcal{G}$ in $\mathbf{G}_s$ for $s \in \{1, 2, T\}$ if for all PPT adversaries $\mathcal{A}$, there exists a negligible function negl *such that* $\mathsf{adv}^{\mathsf{DDH}}_{\mathbf{G},s,\mathcal{A}}(\lambda) < \mathsf{negl}(\lambda)$.

Definition 3.2 (SXDH assumption)**.** *For any sequence of asymmetric prime-order pairing groups $\mathcal{G}$, we says the SXDH assumption holds for $\mathcal{G}$ if DDH holds for $\mathbf{G}_{\lambda,1}$ in $\mathbf{G}_s$ for both $s = 1$ and $s = 2$.*

Definition 3.3 (Bilateral DLIN assumption)**.** *For any adversary $\mathcal{A}$, any sequence of asymmetric prime-order pairing groups $\mathcal{G}$, any security parameter λ, let*

$$\mathsf{adv}^{\mathsf{DLIN}}_{\mathcal{G},\mathcal{A}}(\lambda) := \left|\Pr\left[1 \leftarrow \mathcal{A}\left(1^\lambda, \{[\mathbf{A}]_s, [\mathbf{A}\boldsymbol{r}]_s\}_{s\in[1,2]}\right)\right] - \Pr\left[1 \leftarrow \mathcal{A}\left(1^\lambda, \{[\mathbf{A}]_s, [\boldsymbol{u}]_s\}_{s\in[1,2]}\right)\right]\right|,$$

where the probabilities are taken over $\mathbf{A} \leftarrow_R \mathsf{DLIN}_{p_\lambda}$, $\boldsymbol{r} \leftarrow_R \mathbb{Z}^2_{p_\lambda}$, $\boldsymbol{u} \leftarrow_R \mathbb{Z}^3_{p_\lambda}$, and the random coins of $\mathcal{A}$. Note that the adversary $\mathcal{A}$ also gets a description of the groups $\mathbf{G}_{\lambda,1}$, $\mathbf{G}_{\lambda,2}$, $\mathbf{G}_{\lambda,3}$, and the bilinear map e_λ. We say bilateral DLIN holds in $\mathcal{G}$ if for all PPT adversaries $\mathcal{A}$, there exists a negligible function negl *such that* $\mathsf{adv}^{\mathsf{DLIN}}_{\mathcal{G},\mathcal{A}}(\lambda) < \mathsf{negl}(\lambda)$.

4 Functional Encryption Definitions

We denote by $\mathcal{F} = \cup_{n,d,\ell,\mathsf{size}\in\mathsf{poly}} \left(\{\mathcal{F}_{\lambda,n(\lambda),d(\lambda),\ell(\lambda),\mathsf{size}(\lambda)}\}_{\lambda\in\mathbb{N}}\right)$ an abstract function class, which is parameterised by $\lambda \in \mathbb{N}$ and four polynomials $n(\lambda), d(\lambda), \ell(\lambda), \mathsf{size}(\lambda))$. We call prmtr the tuple $(n, d, \ell, \mathsf{size})$. In this abstract class, every function $f \in \mathcal{F}_{\lambda,\mathsf{prmtr}}$ takes an input from $\mathcal{X}_{\lambda,\mathsf{prmtr}} \times \mathcal{Y}_{\lambda,\mathsf{prmtr}}$ and outputs in $\mathcal{Z}_{\lambda,\mathsf{prmtr}}$. We will specify what this exactly denotes in the exact constructions. Two specific instantiations of those classes are described below:

- The function class $\mathcal{F}^{\mathsf{CIRC}}_{\lambda,\mathsf{prmtr}}$: Here $\mathcal{Y}_{\lambda,\mathsf{prmtr}}$ consists of $\{0,1\}^n$, $\mathcal{X}_{\lambda,\mathsf{prmtr}}$ is empty, $\mathcal{Z}_{\lambda,\mathsf{prmtr}} = \{0,1\}^\ell$. This family consists of Boolean circuits of depth d and size size.

- The function class $\mathcal{F}^{\mathsf{PHFE}}_{\lambda,\mathsf{prmtr}}$: Here $\mathcal{X}_{\lambda,\mathsf{prmtr}} = \mathcal{Y}_{\lambda,\mathsf{prmtr}} = \mathbb{Z}^{n(\lambda)}_{p_\lambda}$ where p_λ is the prime order for the group $\mathcal{G}_\lambda$ (see Remark 3.1), and $\mathcal{Z} = \mathbf{G}_{\lambda,T}$, which denotes the target group of $\mathcal{G}_\lambda$. The class consists of arithmetic circuits NC^1 circuits on the input that belongs to $\mathcal{X}_{\lambda,\mathsf{prmtr}}$, and degree-2 polynomials on the input that belongs to $\mathcal{Y}_{\lambda,\mathsf{prmtr}}$. We describe the exact class later when we need it.

Here we provide the relevant definition regarding functional encryption (FE) and partially-hiding FE (PHFE) along with several notions of efficiency and security properties. FE corresponds to the particular case where the public part of the message (referred to as $\mathcal{X}_{\lambda,\mathsf{prmtr}}$ below) is empty.

Definition 4.1. *(Syntax of a PHFE Scheme.) A partially-hiding functional encryption scheme,* PHFE*, for a functionality* $\{\mathcal{F}_{\lambda,\mathsf{prmtr}} : \mathcal{X}_{\lambda,\mathsf{prmtr}} \times \mathcal{Y}_{\lambda,\mathsf{prmtr}} \to \mathcal{Z}_{\lambda,\mathsf{prmtr}}\}_{\lambda,\mathsf{prmtr}}$*, consists of the following PPT algorithms:*

- $\mathsf{PPGen}(1^\lambda, \mathsf{prmtr})$: *Given as input the security parameter* 1^λ *and additional parameters* $\mathsf{prmtr} = (n, d, \ell, \mathsf{size})$*, it outputs a string* pp*. We assume that* pp *is implicitly given as input to all the algorithms below.*
- $\mathsf{Setup}(\mathsf{pp})$*: Given as input* pp*, it outputs a public key* pk *and a master secret key* msk*.*
- $\mathsf{Enc}(\mathsf{pk}, (x, y))$*: Given as input the public key* pk *and a message* (x, y) *with public part* $x \in \mathcal{X}_{\lambda,\mathsf{prmtr}}$ *and private part* $y \in \mathcal{Y}_{\lambda,\mathsf{prmtr}}$*, outputs the ciphertext* ct *along with the input* x*.*
- $\mathsf{KeyGen}(\mathsf{msk}, f)$*: Given as input the master secret key* msk *and a function* $f \in \mathcal{F}_{\lambda,\mathsf{prmtr}}$*, it outputs a functional decryption key* sk_f*.*
- $\mathsf{Dec}(\mathsf{sk}_f, (x, \mathsf{ct}))$*: Given a functional decryption key* sk_f *and a ciphertext* (x, ct)*, it deterministically outputs a value* z *in* $\mathcal{Z}_{\lambda,\mathsf{prmtr}}$*, or* $\perp$ *if it fails.*

Remark 4.1. (On Secret Key Schemes.) An FE scheme is said to be secret-key is pk is empty, and the encryption algorithm takes as additional input the master secret key msk.

Remark 4.2. (On FE vs PHFE.) The syntax of FE is identical to PHFE described above except that for all $\lambda \in \mathbb{N}$, the set $\mathcal{X}_{\lambda,\mathsf{prmtr}} = \emptyset$, that is, all the input remains private.

Definition 4.2. *(Correctness.) A Partially hiding FE scheme* PHFE *for the functionality* $\mathcal{F} = \{\mathcal{F}_{\lambda,\mathsf{prmtr}}\}_{\lambda,\mathsf{prmtr}}$ *is correct if for security parameter* $\lambda \in \mathbb{N}$ *and every polynomials* $n, d, \ell, \mathsf{size}$ *there exists a negligible function* $\mathsf{negl}(\lambda)$ *such that for all messages* $(x, y) \in \mathcal{X}_{\lambda,\mathsf{prmtr}} \times \mathcal{Y}_{\lambda,\mathsf{prmtr}}$ *and all functions* $f \in \mathcal{F}$*, we have:*

$$\Pr\left[\begin{array}{c} \mathsf{pp} \leftarrow \mathsf{PPGen}(1^\lambda, \mathsf{prmtr}) \\ (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Setup}(\mathsf{pp}) \\ (x, \mathsf{ct}) \leftarrow \mathsf{Enc}(\mathsf{pk}, (x, y)) \\ \mathsf{sk}_f \leftarrow \mathsf{KeyGen}(\mathsf{sk}, f) \\ \mathsf{Dec}(\mathsf{sk}_f, x, \mathsf{ct})) \neq f(x, y) \end{array}\right] \leq \mathsf{negl}(\lambda).$$

Now we give the security notions for PHFE and FE.

4.1 Security Definition

We discuss two security notions. First, for any constant $\epsilon \in (0, 1]$, we present the notion of ϵ-simulation security below:

Definition 4.3 (ϵ-simulation security)**.** *For all* $\epsilon \in (0, 1]$*, we say a PHFE scheme for the functionality* $\mathcal{F} = \{\mathcal{F}_{\lambda,\mathsf{prmtr}}\}_{\lambda,\mathsf{prmtr}}$ *denoted by* PHFE *is* ϵ*-simulation secure if there exists a (possibly stateful) PPT simulator* $\mathcal{S} = (\widetilde{\mathsf{Setup}}, \widetilde{\mathsf{Enc}}, \widetilde{\mathsf{KeyGen}})$ *such that for all stateful PPT adversaries*

$\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, *there exists a negligible function* negl *such that for all security parameters* $\lambda \in \mathbb{N}$, *all polynomials* $\mathsf{prmtr} = (n, d, \ell, \mathsf{size})$, *we have:*

$$\mathsf{adv}^{\mathsf{SIM}}_{\mathsf{PHFE},\mathcal{A}}(1^\lambda, \mathsf{prmtr}) := |\Pr[1 \leftarrow \mathsf{Real}^{\mathsf{PHFE}}_{\mathcal{A}}(1^\lambda, \mathsf{prmtr})] - \Pr[1 \leftarrow \mathsf{Ideal}^{\mathsf{PHFE}}_{\mathcal{A},\mathcal{S}}(1^\lambda, \mathsf{prmtr})]| < \mathsf{negl}(\lambda),$$

where the experiments $\mathsf{Real}^{\mathsf{PHFE}}_{\mathcal{A}}(1^\lambda)$ *and* $\mathsf{Ideal}^{\mathsf{PHFE}}_{\mathcal{A},\mathcal{S}}(1^\lambda)$ *are defined below. The differences between these two experiments are highlighted in red.*

$\underline{\mathsf{Real}^{\mathsf{PHFE}}_{\mathcal{A}}(1^\lambda, \mathsf{prmtr})}$*:*
$(x^*, y^*) \in \mathcal{X}_{\lambda,\mathsf{prmtr}} \times \mathcal{Y}_{\lambda,\mathsf{prmtr}}, (f_j \in \mathcal{F}_{\lambda,\mathsf{prmtr}})_{j\in[Q_{\mathsf{sk}}]} \leftarrow \mathcal{A}_1(1^\lambda)$
$\mathsf{pp} \leftarrow \mathsf{PPGen}(1^\lambda, \mathsf{prmtr})$
$(\mathsf{pk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(\mathsf{pp})$
$(x^*, \mathsf{ct}^*) \leftarrow \mathsf{Enc}(\mathsf{pk}, (x^*, y^*))$
$\forall j \in [Q_{\mathsf{sk}}]$: $\mathsf{sk}_{f_j} \leftarrow \mathsf{KeyGen}(\mathsf{msk}, f_j)$
$\alpha \leftarrow \mathcal{A}_2(\mathsf{pp}, \mathsf{pk}, (\mathsf{sk}_{f_j})_{j\in Q_{\mathsf{sk}}}, x^*, \mathsf{ct}^*)$
Output α.

$\underline{\mathsf{Ideal}^{\mathsf{PHFE}}_{\mathcal{A},\mathcal{S}}(1^\lambda, \mathsf{prmtr})}$*:*
$(x^*, y^*) \in \mathcal{X}_{\lambda,\mathsf{prmtr}} \times \mathcal{Y}_{\lambda,\mathsf{prmtr}}, (f_j \in \mathcal{F}_{\lambda,\mathsf{prmtr}})_{j\in[Q_{\mathsf{sk}}]} \leftarrow \mathcal{A}_1(1^\lambda)$
$\mathsf{pp} \leftarrow \mathsf{PPGen}(1^\lambda, \mathsf{prmtr})$
$(\widetilde{\mathsf{pk}}, \mathsf{td}) \leftarrow \widetilde{\mathsf{Setup}}(\mathsf{pp}), \omega \leftarrow \mathsf{Sample}(x^*, y^*, (f_j)_{j\in[Q_{\mathsf{sk}}]})$
$(x^*, \widetilde{\mathsf{ct}^*}) \leftarrow \widetilde{\mathsf{Enc}}(\mathsf{td}, \omega)$
$\forall j \in [Q_{\mathsf{sk}}] : \widetilde{\mathsf{sk}_{f_j}} \leftarrow \widetilde{\mathsf{KeyGen}}(\mathsf{td}, f_j, \omega)$
$\alpha \leftarrow \mathcal{A}_2\left(\mathsf{pp}, \widetilde{\mathsf{pk}}, (\widetilde{\mathsf{sk}_{f_j}})_{j\in Q_{\mathsf{sk}}}, x^*, \widetilde{\mathsf{ct}^*}\right)$
Output α.

The algorithm Sample, *given as input the tuple* $\left(x^*, (f_j, f_j(x^*, y^*))_{j\in[Q_{\mathsf{sk}}]}\right)$, *flips a biased coin. If the outcome is tails (which happens with probability* ϵ *over the coin flip), then it outputs* $\omega = \left(x^*, (f_j, f_j(x^*, y^*))_{j\in[Q_{\mathsf{sk}}]}\right)$. *If the outcome is heads (which happens with probability* $1 - \epsilon$ *over the coin flip), then it outputs* $\omega = \left(x^*, y^*(f_j)_{j\in[Q_{\mathsf{sk}}]}\right)$.

Remark 4.3 (Standard simulation security)**.** If $\epsilon = 1$, the algorithm Sample always outputs $\omega = (x^*, (f_j, f_j(x^*, y^*))_{j\in[Q_{\mathsf{sk}}]})$, which corresponds to the standard simulation security definition.

Remark 4.4 (Secret-Key schemes)**.** This definition can be easily adapted to a secret-key scheme simply by having the encryption algorithm get the additional input msk.

Remark 4.5 (Subexponential security)**.** If $\epsilon = 1$, and the negl above is $2^{-\lambda^{\Omega(1)}}$, then the scheme is said to satisfy subexponential security.

Remark 4.6 (Number of functional decryption keys)**.** We say a a scheme is many-key secure if security holds for any polynomial Q_{sk}, and one-key secure if $Q_{\mathsf{sk}} = 1$. When we do not specify it explicitly, we mean one-key security.

We also give an indistinguishability-based security definition.

Definition 4.4 (IND security)**.** *We say an FE scheme* FE *for functionality* $\mathcal{F} = \{\mathcal{F}_{\lambda,\mathsf{prmtr}}\}_{\lambda\in\mathbb{N}}$ *is IND secure if for all stateful PPT adversaries* $\mathcal{A}$*, all polynomial parameters* $\mathsf{prmtr} = (n, d, \ell, \mathsf{size})$ *there exists a negligible function* negl *such that , we have:*

$$\mathsf{adv}^{\mathsf{IND}}_{\mathsf{FE},\mathcal{A}}(\lambda) := 2 \cdot |1/2 - \Pr[1 \leftarrow \mathsf{IND}^{\mathsf{FE}}_{\mathcal{A}}(1^\lambda, \mathsf{prmtr})]| < \mathsf{negl}(\lambda),$$

where the experiment $\mathsf{IND}^{\mathsf{FE}}_{\mathcal{A}}(1^\lambda, \mathsf{prmtr})$ *is defined below.*

$\underline{\mathsf{IND}^{\mathsf{FE}}_{\mathcal{A}}(1^\lambda, \mathsf{prmtr})}$*:*
$\{x^i_0, x^i_1\}_{i\in[Q_{\mathsf{ct}}]}, \{f^j\}_{j\in[Q_{\mathsf{sk}}]} \leftarrow \mathcal{A}(1^\lambda)$
$\mathsf{pp} \leftarrow \mathsf{PPGen}(1^\lambda, \mathsf{prmtr})$
Where $\forall i \in [Q]$*:* $x^i_0, x^i_1 \in \mathcal{Y}_{\lambda,\mathsf{prmtr}}$*,* $\forall j \in [Q_{\mathsf{sk}}]$*:* $f^j \in \mathcal{F}_{\lambda,\mathsf{prmtr}}$
$(\mathsf{pk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(\mathsf{pp})$*,* $b \leftarrow_R \{0, 1\}$
$\forall i \in [Q_{\mathsf{ct}}] : \mathsf{ct}_i \leftarrow \mathsf{Enc}(\mathsf{pk}, x^i_b)$*,* $\forall j \in [Q_{\mathsf{sk}}] : \mathsf{sk}_j \leftarrow \mathsf{KeyGen}(\mathsf{msk}, f^j)$
$b' \leftarrow \mathcal{A}(\{\mathsf{ct}_i\}_{i\in[Q_{\mathsf{ct}}]}, \{\mathsf{sk}_j\}_{j\in[Q_{\mathsf{sk}}]}, \mathsf{pk})$
Return 1 *if* $b = b'$ *and* $\forall\ i \in [Q_{\mathsf{ct}}],\ j \in [Q_{\mathsf{sk}}], f^j(x^i_0) = f^j(x^i_1)$*,* 0 *otherwise.*

As for simulation security, we say that FE *satisfies subexponential security if* $\mathsf{negl}(\lambda) = 2^{-\lambda^{\Omega(1)}}$.

We also define secret-key function hiding FE as follows.

Definition 4.5 (Function Hiding Indistinguishability security)**.** *We say a secret-key FE scheme* FE *for functionality* $\mathcal{F} = \{\mathcal{F}_{\lambda,\mathsf{prmtr}}\}_{\lambda,\mathsf{prmtr}}$ *is IND-FH secure if for all stateful PPT adversaries* $\mathcal{A}$*, and all polynomial paramters* $\mathsf{prmtr} = (n, d, \ell, \mathsf{size})$ *there exists a negligible function* negl *such that, we have:*

$$\mathsf{adv}^{\mathsf{IND-FH}}_{\mathsf{FE},\mathcal{A}}(\lambda) := 2 \cdot |1/2 - \Pr[1 \leftarrow \mathsf{IND} - \mathsf{FH}^{\mathsf{FE}}_{\mathcal{A}}(1^\lambda, \mathsf{prmtr})]| < \mathsf{negl}(\lambda),$$

where the experiment $\mathsf{IND} - \mathsf{FH}^{\mathsf{FE}}_{\mathcal{A}}(1^\lambda, \mathsf{prmtr})$ *is defined below.*

$\underline{\mathsf{IND} - \mathsf{FH}^{\mathsf{FE}}_{\mathcal{A}}(1^\lambda, \mathsf{prmtr})}$*:*
$\{x^i_0, x^i_1\}_{i\in[Q_{\mathsf{ct}}]}, \{f^j_0, f^j_1\}_{j\in[Q_{\mathsf{sk}}]} \leftarrow \mathcal{A}_1(1^\lambda)$
$\mathsf{pp} \leftarrow \mathsf{PPGen}(1^\lambda, \mathsf{prmtr})$
$\mathsf{msk} \leftarrow \mathsf{Setup}(\mathsf{pp})$*,* $b \leftarrow_R \{0, 1\}$
$\forall i \in [Q_{\mathsf{ct}}] : \mathsf{ct}_i \leftarrow \mathsf{Enc}(\mathsf{msk}, x^i_b)$*,* $\forall j \in [Q_{\mathsf{sk}}] : \mathsf{sk}_j \leftarrow \mathsf{KeyGen}(\mathsf{msk}, f^j_b)$
$b' \leftarrow \mathcal{A}(\{\mathsf{ct}_i\}_{i\in[Q_{\mathsf{ct}}]}, \{\mathsf{sk}_j\}_{j\in[Q_{\mathsf{sk}}]})$
Return 1 *if* $b = b'$ *and* $\forall\ i \in [Q_{\mathsf{ct}}],\ j \in [Q_{\mathsf{sk}}], f^j_0(x^i_0) = f^j_1(x^i_1)$*,* 0 *otherwise.*

Remark 4.7 (Subexponential Security)**.** In both the definitions above, we say that the schemes satisfy subexponential security if $\mathsf{negl} = 2^{-\lambda^{\Omega(1)}}$.

4.2 Efficiency Features

We now define various efficiency notions for PHFE (which are straightforward to adapt to FE).

Definition 4.6 (Linear efficiency)**.** *We say a PHFE for the functionality $\mathcal{F} = \{\mathcal{F}_{\lambda,\mathsf{prmtr}}\}_{\lambda,\mathsf{prmtr}}$ satisfies linear efficiency if there exists a polynomial* poly *such that for all security parameters $\lambda \in \mathbb{N}$ and all polynomial parameters $\mathsf{prmtr} = (n, d, \ell, \mathsf{size})$, all messages $(x, y) \in \mathcal{X}_{\lambda,\mathsf{prmtr}} \times \mathcal{Y}_{\lambda,\mathsf{prmtr}}$, all* pp *in the support of* $\mathsf{PPGen}(1^\lambda, \mathsf{prmtr})$*, all* $(\mathsf{pk}, \mathsf{msk})$ *in the support of* $\mathsf{Setup}(\mathsf{pp})$ *the size of the circuit computing* $\mathsf{Enc}(\mathsf{pk}, \cdot)$ *on the input (x, y) is at most $(|x| + |y|) \cdot \mathsf{poly}(\lambda)$, for some fixed polynomial* poly *where $|x|$ and $|y|$ denote the size of x and y, respectively.*

Now we define the notion of sublinearity for FE scheme for the functionality $\mathcal{F}$ (i.e. all polynomial circuits, defined in Section 3). It was shown in a series of works [AJ15, BV15, BNPW16] that such FE schemes for P/poly imply obfuscation (assuming subexponential security).

Definition 4.7 (Sublinearity)**.** *Let* FE *be an FE scheme for the functionality $\mathcal{F} = \{\mathcal{F}_{\lambda,\mathsf{prmtr}}\}_{\lambda,\mathsf{prmtr}}$. If there exists $\epsilon \in (0, 1)$ and a polynomial* poly *such that for all tuple of polynomials* $\mathsf{prmtr} = (n, d, \ell, \mathsf{size})$*, all $\lambda \in \mathbb{N}$, all* pp *in the support of* $\mathsf{PPGen}(1^\lambda, \mathsf{prmtr})$*, all* $(\mathsf{pk}, \mathsf{msk})$ *in the support of* $\mathsf{Setup}(\mathsf{pp})$*:*

- *if the size of the circuit* $\mathsf{Enc}(\mathsf{pk}, \cdot)$ *is at most* $\mathsf{size}^{1-\epsilon} \cdot \mathsf{poly}(n, \lambda)$ *then* FE *is said to be sublinearly efficient. It is said to be compact if $\epsilon = 1$.*
- *if for all $x \in \{0, 1\}^n$, all ciphertexts* ct *in the support of* $\mathsf{Enc}(\mathsf{pk}, x)$*, the size of* ct *is at most* $\mathsf{size}^{1-\epsilon} \cdot \mathsf{poly}(n, \lambda)$ *then* FE *is said to be sublinearly ciphertext-efficient.*
- *if for all $x \in \{0, 1\}^n$, all ciphertexts* ct *in the support of* $\mathsf{Enc}(\mathsf{pk}, x)$*, the size of* ct *is at most* $\ell^{1-\epsilon} \cdot \mathsf{poly}(n, \lambda)$ *then* FE *is said to be sublinearly output-efficient.*

Remark 4.8 (levelled linear efficiency, compactness, and sublinearity)**.** More generally, we say that the scheme satisfies levelled linear efficiency or levelled compactness, or levelled sublinearity if the multiplicative factor $\mathsf{poly}(n, \lambda)$ in Definition 4.6 or Definition 4.7 is replaced by $\mathsf{poly}(\lambda, n, d)$, i.e. the polynomial also depends on the depth bound d.

4.3 Structural Properties

Now we define some structural properties that are very specific to our construction. First we define the notion of special structure which captures the property of a function key can be generated just by applying a linear function of the master secret key over some field along with the fact that the decryption of a ciphertext is "almost linear" (specified below).

Definition 4.8. *(Special Structure*.) We say that a functional encryption scheme* FE *for $\mathcal{F}^{\mathsf{CIRC}} = \{\mathcal{F}^{\mathsf{CIRC}}_{\lambda,\mathsf{prmtr}}\}_{\lambda,\mathsf{prmtr}}$ satisfies special structure* if there exist polynomials h_1, h_2, h_3, h_4 such that the following holds. Recall $\mathcal{F}^{\mathsf{CIRC}}_{\lambda,\mathsf{prmtr}}$ for $\mathsf{prmtr} = (n, d, \ell, \mathsf{size})$ consists of all Boolean circuits with n bits of input, ℓ bits of output, depth d and size* size*.*

- *(PP Syntax.) The* pp *generated by the* $\mathsf{PPGen}(1^\lambda, \mathsf{prmtr})$ *algorithm contains a $h_1(\lambda)$-bit prime modulus p.*

- *(Linear secret key Structure.) The master secret key is a vector in $\boldsymbol{s} \in \mathbb{Z}_p^{h_2(\lambda)}$. For any function $f \in \mathcal{F}_{\lambda,\mathsf{prmtr}}$, let $f = \{f_i\}_{i\in[\ell]}$ denote the circuit computing i^{th} bit of f. The functional secret key is of the form $\mathsf{sk}_f = \{\mathsf{sk}_{f_i}\}_{i\in[\ell]}$ where each $\mathsf{sk}_{f_i} = \langle \mathsf{pp}_{f_i}, \boldsymbol{s}\rangle + e_i \mod p$ where $e_i \leftarrow_R \{0,\ldots,h_3(\lambda,n,\ell,d)\}$ and pp_{f_i} is some deterministic polynomial time computable function of pp and f_i.*

- *(Linear + Round Decryption with polynomial decryption error.) There exists a deterministic poly-time algorithm such that given an encryption ct of $m \in \{0,1\}^n$ and a function $f = (f_1,\ldots,f_\ell) \in \mathcal{F}_{\lambda,\mathsf{prmtr}}$, for every $i \in [\ell]$, computes ct_{f_i} such that $|\mathsf{ct}_{f_i} - \langle \mathsf{pp}_{f_i}, \boldsymbol{s}\rangle - f_i(m)\lceil \frac{p}{2}\rceil| \leq h_4(\lambda,d,\ell,\mathsf{size})$. Given the secret-key for a function $f = (f_1,\ldots,f_\ell)$, this can be used to recover $f(m) = (f_1(m),\ldots,f_\ell(m))$.*

5 ϵ-simulation Secure FE to Fully Secure FE

In this section, we show how to construct an IND secure FE scheme for all circuits from a ϵ-simulation secure FE for all circuits for any $\epsilon \in (0,1)$ (as per Definition 4.3), additionally assuming LWE. The IND secure FE (see Definition 4.4) that results from our transformation inherits the sublinear efficiency property of the underlying ϵ-simulation secure FE.

5.1 Homomorphic Secret Sharing

We first recall the notion of a Homomorphic-Secret Sharing (HSS) scheme for circuits that will be used as one of the tools for our construction. We recall that for $\lambda \in \mathbb{N}$, and polynomial parameters $\mathsf{prmtr} = (n,d,\ell,\mathsf{size})$, $\mathcal{F}^{\mathsf{CIRC}}_{\lambda,\mathsf{prmtr}}$ consists of all boolean circuits with n bits of inputs, ℓ bits of outputs, depth d and size size.

Definition 5.1 (Syntax of a HSS Scheme)**.** *An* HSS *scheme for circuits $\mathcal{F}^{\mathsf{CIRC}} = \{\mathcal{F}^{\mathsf{CIRC}}_{\lambda,\mathsf{prmtr}}\}_{\lambda,\mathsf{prmtr}}$ consists of the following PPT algorithms:*

- $\mathsf{Share}(1^\lambda, \mathsf{prmtr}, 1^t, x)$ *: Given as input the security parameter 1^λ, the tuple of polynomials $\mathsf{prmtr} = (n,\ell,d,\mathsf{size})$, a threshold 1^t, and an input $x \in \{0,1\}^n$, the sharing algorithm outputs t shares $(\mathsf{sh}_1,\ldots,\mathsf{sh}_t)$.*

- $\mathsf{Eval}(f,\mathsf{sh}_i)$*: Given as input a function $f \in \mathcal{F}^{\mathsf{CIRC}}_{\lambda.\mathsf{prmtr}}$ and a share sh_i, the evaluation algorithm deterministically outputs an evaluation $\widehat{\mathsf{sh}}_i$.*

- $\mathsf{Decode}(\{\widehat{\mathsf{sh}}_i\}_{i\in[t]})$*: Given as input evaluations evaluations $\widehat{\mathsf{sh}}_i$ for all $i \in [t]$, the decoding algorithm deterministically outputs $z \in \{0,1\}^\ell$ or $\perp$ if it fails.*

Correctness. An HSS scheme is said to be correct if for all $\lambda \in \mathbb{N}$, all polynomials $\mathsf{prmtr} = (n,d,\ell,\mathsf{size})$, $t \in \mathbb{N}$, all $f \in \mathcal{F}^{\mathsf{CIRC}}_{\lambda,\mathsf{prmtr}}$, all shares $\mathsf{sh}_1,\ldots,\mathsf{sh}_t$ in the support of $\mathsf{Share}(1^\lambda,\mathsf{prmtr},1^t,x)$, all evaluations $\widehat{\mathsf{sh}}_1,\ldots,\widehat{\mathsf{sh}}_t$ in the support of $\mathsf{Eval}(f,\mathsf{sh}_1),\ldots,\mathsf{Eval}(f,\mathsf{sh}_t)$ respectively, we have $\mathsf{Decode}(\{\widehat{\mathsf{sh}}_i\}_{i\in[t]}) = f(x)$.

Definition 5.2 (IND security)**.** *An HSS scheme for circuits* HSS *is said to be IND-secure if for all admissible stateful ppt adversaries* $\mathcal{A}$*, there exists a negligible function* negl*, such that for all tuple of polynomials* $\mathsf{prmtr} = (n, d, \mathsf{size}, \ell)$ *and* t*:*

$$\mathsf{adv}^{\mathsf{Ind}}_{\mathsf{HSS},\mathcal{A}}(1^\lambda) := |\Pr[1 \leftarrow \mathsf{Expt}^0_{\mathcal{A}}(1^\lambda)] - \Pr[1 \leftarrow \mathsf{Expt}^1_{\mathcal{A}}(1^\lambda)]| < \mathsf{negl}(\lambda),$$

where the experiments $\mathsf{Expt}^b_{\mathcal{A}}(1^\lambda)$ *for* $b \in \{0,1\}$ *are defined below. We say an adversary* $\mathcal{A}$ *is admissible if* S *is not equal to* $[t]$ *and* $f(x_0) = f(x_1)$*. Further, the scheme is subexponentially secure if* $\mathsf{negl}(\lambda) = 2^{-\lambda^{\Omega(1)}}$*.*

$\underline{\mathsf{Expt}^b_{\mathcal{A}}(1^\lambda)}$*:*
$(x_1, x_0, f) \in (\{0,1\}^n)^2 \times \mathcal{F}_{\lambda,\mathsf{prmtr}} \leftarrow \mathcal{A}(1^\lambda)$
$S \subsetneq [t] \leftarrow \mathcal{A}(1^\lambda)$
$(\{\mathsf{sh}_i\}_{i\in[t]}) \leftarrow \mathsf{Share}(1^\lambda, \mathsf{prmtr}, 1^t, x_b)$
$\forall i \in [t] : \widehat{\mathsf{sh}}_i \leftarrow \mathsf{Eval}(f, \mathsf{sh}_i)$
$\alpha \leftarrow \mathcal{A}(\{\mathsf{sh}_i\}_{i\in S}, \{\widehat{\mathsf{sh}}_i\}_{i\in[t]})$
Output α*.*

We now list the efficiency properties we require from the HSS scheme.

Efficiency Properties. There exist polynomials h_1, h_2, h_3, h_4, h_5 such that for all polynomials $n, d, \ell, \mathsf{size}, t$ and all security parameters λ, the following holds:

- The circuit computing the function $\mathsf{Share}(1^\lambda, \mathsf{prmtr}, 1^t, \cdot)$ is of size $h_1(n, d, t, \lambda)$. The length of each share is $n \cdot h_2(\lambda, d)$
- For any function $f \in \mathcal{F}^{\mathsf{CIRC}}_{\lambda,\mathsf{prmtr}}$, the circuit computing the function $\mathsf{Eval}(f, \cdot)$ is of size $\mathsf{size} \cdot h_3(\lambda, d)$ and of depth $d \cdot h_4(\lambda)$.
- The output length of $\mathsf{Eval}(f, \cdot)$ is $\ell \cdot h_5(\lambda, d)$

As shown in [MW16, DHRW16, BGG+18], if there exists a constant $\rho > 0$ such that the LWE assumption with modulus-to-noise ratio $2^{\dim^\rho}$ holds, then there exists an IND-secure HSS scheme for circuits.

5.2 Transformation

We now present our transformation.

Theorem 5.1. *Let* $\epsilon \in (0,1)$*,* FE *be a (single-key)* ϵ*-simulation secure FE for* $\mathcal{F}^{\mathsf{CIRC}}$*, and* HSS *be a IND-secure HSS for* $\mathcal{F}^{\mathsf{CIRC}}$*. Then,* $\mathsf{FE}_{\mathsf{amp}}$ *defined below satisfies (single-key) indistinguishability security. Further, if* FE *and* HSS *are subexponentially secure, then so is* FE*.*

Let HSS be an HSS for circuits and FE be an FE for circuits. Below is our construction $\mathsf{FE}_{\mathsf{amp}}$ from HSS and FE.

- $\mathsf{PPGen}(1^\lambda, \mathsf{prmtr})$: It takes as input the security parameter 1^λ and the tuple of polynomials $\mathsf{prmtr} = (n, \ell, d, \mathsf{size})$. Let h_1, h_2, h_3, h_4, h_5 be the polynomials associated with HSS (see the efficiency properties above). The algorithm sets the polynomials $t(\lambda) = \lambda$, $n'(\lambda) = n \cdot h_2(\lambda, d)$, $\mathsf{size}' = \mathsf{size} \cdot h_3(\lambda, d)$, $d' = d \cdot h_4(\lambda)$, $\ell' = \ell \cdot h_5(\lambda, d)$ and $\mathsf{prmtr}' = (n', \ell', \mathsf{size}', d')$. Run $\mathsf{FE.pp} \leftarrow_{\mathrm{R}} \mathsf{FE.PPGen}(1^\lambda, \mathsf{prmtr}')$. Output $\mathsf{pp} = \mathsf{FE.pp}$.

- $\mathsf{Setup}(\mathsf{pp})$: For all $i \in [t]$, run $(\mathsf{FE.pk}_i, \mathsf{FE.msk}_i) \leftarrow_{\mathrm{R}} \mathsf{FE.Setup}(\mathsf{pp})$. Output $\mathsf{pk} = (\mathsf{FE.pk}_1, \ldots, \mathsf{FE.pk}_t)$ and $\mathsf{msk} = (\mathsf{FE.msk}_1, \ldots, \mathsf{FE.msk}_t)$.

- $\mathsf{Enc}(\mathsf{pk}, m)$: Run $\mathsf{HSS.Share}(1^\lambda, \mathsf{prmtr}, 1^t, m) \to \{\mathsf{sh}_i\}_{i \in [t]}$. Parse $\mathsf{pk} = (\mathsf{FE.pk}_1, \ldots, \mathsf{FE.pk}_t)$. Compute $\mathsf{ct}[i] = \mathsf{FE.Enc}(\mathsf{FE.pk}_i, \mathsf{sh}_i)$ for all $i \in [t]$. Output $\mathsf{ct} = (\mathsf{ct}[1], \ldots, \mathsf{ct}[t])$.

- $\mathsf{KeyGen}(\mathsf{msk}, f)$: Parse $\mathsf{msk} = (\mathsf{FE.msk}_1, \ldots, \mathsf{FE.msk}_t)$. Let F be the function $\mathsf{HSS.Eval}(f, \cdot)$. For all $i \in [t]$, compute $\mathsf{sk}_F[i] \leftarrow \mathsf{FE.KeyGen}(\mathsf{FE.msk}_i, F)$. Output $\mathsf{sk}_f = (\mathsf{sk}_F[1], \ldots, \mathsf{sk}_F[t])$.

- $\mathsf{Dec}(\mathsf{sk}, \mathsf{ct})$: Parse $\mathsf{sk}_f = (\mathsf{sk}_F[1], \ldots, \mathsf{sk}_F[t])$ and $\mathsf{ct} = (\mathsf{ct}[1], \ldots, \mathsf{ct}[t])$. Compute $\widehat{\mathsf{sh}}_i = \mathsf{FE.Dec}(\mathsf{sk}_F[i], \mathsf{ct}[i])$ for all $i \in [t]$. Output $z = \mathsf{HSS.Decode}(\widehat{\mathsf{sh}}_1, \ldots, \widehat{\mathsf{sh}}_t)$.

Correctness. The correctness of the scheme follows immediately from the correctness of FE and HSS.

Sublinear-Efficiency. We show that the transformation preserves (levelled) sublinear (output) effiency. Namely, we have:

- If FE satisfies (levelled) ciphertext sublinearity, then, the length of the ciphertext of $\mathsf{FE}_{\mathsf{amp}}$ is $O(t \cdot \mathsf{size}'^{1-\rho}\, \mathsf{poly}(n', d', \lambda))$ for some $\rho > 0$. Observe that $t = \lambda$, $\mathsf{size}' = \mathsf{size} \cdot h_3(\lambda, d)$, and $d' = d \cdot h_4(\lambda)$ and $n' = n \cdot h_2(\lambda, d)$. Thus, the size of the ciphertext is $O(\mathsf{size}^{1-\rho}\, \mathsf{poly}'(n, d, \lambda))$. Thus, the resulting scheme is also (levelled) ciphertext sublinear.

- If FE is (levelled) sublinearly efficient, then the following happens. The size of the circuit computing the encryption is $O(h_1(n, d, t, \lambda) + t \cdot \mathsf{size}'^{1-\rho}\, \mathsf{poly}(\lambda, d', n'))$. The first part comes from the size of the circuit computing the HSS share, and the second part comes from the size of the circuit computing FE encryptions. Thus, this is $O(\mathsf{size}^{1-\rho}\, \mathsf{poly}'(n, d, \lambda))$ and $\mathsf{FE}_{\mathsf{amp}}$ is (levelled) sublinearly efficient.

- If FE satisfies (levelled) output-sublinearity, then the following happens. The size of the circuit computing the encryption is $O(h_1(n, d, t, \lambda) + t \cdot \ell'^{1-\rho}\, \mathsf{poly}(\lambda, d', n'))$. The first part comes from the size of the circuit computing the HSS share, and the second part comes from the size of the circuit computing FE encryptions. Thus, this is $O(\ell^{1-\rho}\, \mathsf{poly}'(n, d, \lambda))$ and $\mathsf{FE}_{\mathsf{amp}}$ is (levelled) output-sublinearly efficient.

Security Proof. We complete the proof of Theorem 5.1, namely we prove the IND security of $\mathsf{FE}_{\mathsf{amp}}$ from the ϵ-simulation security of FE and the IND security of HSS. Note that if the underlying FE and HSS are subexponentially secure, then so is $\mathsf{FE}_{\mathsf{amp}}$. We first list the hybrids and then argue indistinguishability between them inline. The first hybrid the

challenge bit is $b \leftarrow_{\mathrm{R}} \{0,1\}$, whereas the last hybrid is independent of b.

$\mathbf{Hybrid}_0$:

- $\mathcal{A}$ outputs (m_0, m_1), and a function f.
- Sample $b \leftarrow_{\mathrm{R}} \{0,1\}$. Compute $(\mathsf{sh}_1, \ldots, \mathsf{sh}_t) \leftarrow_{\mathrm{R}} \mathsf{Share}(1^\lambda, \mathsf{prmtr}, 1^t, m_b)$.
- Run $\mathsf{pp} \leftarrow_{\mathrm{R}} \mathsf{FE.PPGen}(1^\lambda, \mathsf{prmtr}')$ and sample $(\mathsf{FE.pk}_i, \mathsf{FE.msk}_i) \leftarrow_{\mathrm{R}} \mathsf{Setup}(\mathsf{pp})$ for $i \in [t]$. Set $\mathsf{pk} = (\mathsf{FE.pk}_1, \ldots, \mathsf{FE.pk}_t)$.
- Compute $\mathsf{ct}[i] \leftarrow \mathsf{FE.Enc}(\mathsf{pk}_i, \mathsf{sh}_i)$ for $i \in [t]$. Let $\mathsf{ct} = (\mathsf{ct}[1], \ldots, \mathsf{ct}[t])$.
- Compute $\mathsf{sk}_F[i] \leftarrow \mathsf{FE.KeyGen}(\mathsf{FE.msk}_i, F)$ where $F = \mathsf{HSS.Eval}(f, \cdot)$ for $i \in [t]$. Let $\mathsf{sk}_f = (\mathsf{sk}_F[1], \ldots, \mathsf{sk}_F[t])$.
- Output $(\mathsf{pp}, \mathsf{pk}, \mathsf{ct}, \mathsf{sk}_f)$

$\mathbf{Hybrid}_1$:

- $\mathcal{A}$ outputs (m_0, m_1), and a function f.
- Sample $b \leftarrow_{\mathrm{R}} \{0,1\}$. Compute $(\mathsf{sh}_1, \ldots, \mathsf{sh}_t) \leftarrow_{\mathrm{R}} \mathsf{Share}(1^\lambda, \mathsf{prmtr}, 1^t, m_b)$.
- Run $\mathsf{pp} \leftarrow_{\mathrm{R}} \mathsf{FE.PPGen}(1^\lambda, \mathsf{prmtr}')$ and run $(\mathsf{FE.}\widetilde{\mathsf{pk}}_i, \mathsf{FE.td}_i) \leftarrow_{\mathrm{R}} \widetilde{\mathsf{Setup}}(\mathsf{pp})$ for $i \in [t]$. Set $\mathsf{pk} = (\mathsf{FE.}\widetilde{\mathsf{pk}}_1, \ldots, \mathsf{FE.}\widetilde{\mathsf{pk}}_t)$.
- Let $\gamma_i \leftarrow \mathsf{Sample}(\mathsf{sh}_i, F)$.
- Compute $\mathsf{ct}[i] \leftarrow \widetilde{\mathsf{FE.Enc}}(\mathsf{FE.td}_i, \gamma_i)$ and $\mathsf{sk}_F[i] \leftarrow \widetilde{\mathsf{FE.KeyGen}}(\mathsf{FE.td}_i, F, \gamma_i)$ for $i \in [t]$.
- Let $\mathsf{ct} = (\mathsf{ct}[1], \ldots, \mathsf{ct}[t])$ and $\mathsf{sk}_f = (\mathsf{sk}_F[1], \ldots, \mathsf{sk}_F[t])$.
- Output $(\mathsf{pp}, \mathsf{pk}, \mathsf{ct}, \mathsf{sk}_f)$

The hybrids above are computationally indistinguishable due to ϵ-simulation security of FE. The only difference between these hybrids is how $\mathsf{ct}[i], \mathsf{sk}_F[i]$ is generated for all $i \in [t]$. In $\mathbf{Hybrid}_0$ is generated using honest Enc and KeyGen algorithms and in $\mathbf{Hybrid}_1$ they are generated using the simulation algorithms.

$\mathbf{Hybrid}_2$:

- $\mathcal{A}$ outputs (m_0, m_1), and a function f.
- Sample $b \leftarrow_{\mathrm{R}} \{0,1\}$. Compute $(\mathsf{sh}_1, \ldots, \mathsf{sh}_t) \leftarrow_{\mathrm{R}} \mathsf{Share}(1^\lambda, \mathsf{prmtr}, 1^t, m_b)$.
- Run $\mathsf{pp} \leftarrow_{\mathrm{R}} \mathsf{FE.PPGen}(1^\lambda, \mathsf{prmtr}')$ and run $(\mathsf{FE.}\widetilde{\mathsf{pk}}_i, \mathsf{FE.td}_i) \leftarrow_{\mathrm{R}} \widetilde{\mathsf{Setup}}(\mathsf{pp})$ for $i \in [t]$. Set $\mathsf{pk} = (\mathsf{FE.}\widetilde{\mathsf{pk}}_1, \ldots, \mathsf{FE.}\widetilde{\mathsf{pk}}_t)$.

- Sample $\boldsymbol{\alpha} \in \{0,1\}^t$. Each is independently sampled where $\alpha_i = 1$ with probability ϵ, and 0 otherwise. If $\alpha_i = 1$, set $\gamma_i = (F, F(\mathsf{sh}_i))$, otherwise $\gamma_i = (\mathsf{sh}_i, F)$.
- Compute $\mathsf{ct}[i] \leftarrow \widetilde{\mathsf{FE.Enc}}(\mathsf{FE.td}_i, \gamma_i)$ and $\mathsf{sk}_F[i] \leftarrow \widetilde{\mathsf{FE.KeyGen}}(\mathsf{FE.td}_i, F, \gamma_i)$ for $i \in [t]$.
- Let $\mathsf{ct} = (\mathsf{ct}[1], \ldots, \mathsf{ct}[t])$ and $\mathsf{sk}_f = (\mathsf{sk}_F[1], \ldots, \mathsf{sk}_F[t])$.
- Output $(\mathsf{pp}, \mathsf{pk}, \mathsf{ct}, \mathsf{sk}_f)$

$\mathbf{Hybrid}_1$ and $\mathbf{Hybrid}_2$ are identical as we just expanded how the Sample operates. We now describe $\mathbf{Hybrid}_3$, where we abort when $\alpha = 0^t$.

$\mathbf{Hybrid}_3$:

- $\mathcal{A}$ outputs (m_0, m_1), and a function f.
- Sample $b \leftarrow_{\mathrm{R}} \{0,1\}$. Compute $(\mathsf{sh}_1, \ldots, \mathsf{sh}_t) \leftarrow_{\mathrm{R}} \mathsf{Share}(1^\lambda, \mathsf{prmtr}, 1^t, m_b)$.
- Run $\mathsf{pp} \leftarrow_{\mathrm{R}} \mathsf{FE.PPGen}(1^\lambda, \mathsf{prmtr}')$ and run $(\mathsf{FE.}\widetilde{\mathsf{pk}}_i, \mathsf{FE.td}_i) \leftarrow_{\mathrm{R}} \widetilde{\mathsf{Setup}}(\mathsf{pp})$ for $i \in [t]$. Set $\mathsf{pk} = (\mathsf{FE.}\widetilde{\mathsf{pk}}_1, \ldots, \mathsf{FE.}\widetilde{\mathsf{pk}}_t)$.
- Sample $\boldsymbol{\alpha} \in \{0,1\}^t$. Each is independently sampled where $\alpha_i = 1$ with probability ϵ, and 0 otherwise. If $\alpha_i = 1$, set $\gamma_i = (F, F(\mathsf{sh}_i))$, otherwise $\gamma_i = (\mathsf{sh}_i, F)$. Abort if $\boldsymbol{\alpha} = 0^t$.
- Compute $\mathsf{ct}[i] \leftarrow \widetilde{\mathsf{FE.Enc}}(\mathsf{FE.td}_i, \gamma_i)$ and $\mathsf{sk}_F[i] \leftarrow \widetilde{\mathsf{FE.KeyGen}}(\mathsf{FE.td}_i, F, \gamma_i)$ for $i \in [t]$.
- Let $\mathsf{ct} = (\mathsf{ct}[1], \ldots, \mathsf{ct}[t])$ and $\mathsf{sk}_f = (\mathsf{sk}_F[1], \ldots, \mathsf{sk}_F[t])$.
- Output $(\mathsf{pp}, \mathsf{pk}, \mathsf{ct}, \mathsf{sk}_f)$

$\mathbf{Hybrid}_2$ and $\mathbf{Hybrid}_3$ are statistically indistinguishable with the statistical distance being $\Pr[\alpha = 0^t] = (1-\epsilon)^t$. If $\epsilon \in (0,1]$ is a constant then this is $2^{-\lambda^{\Omega(1)}}$. In the next hybrid, we share m_0 instead of m_b.

$\mathbf{Hybrid}_4$:

- $\mathcal{A}$ outputs (m_0, m_1), and a function f.
- Compute $(\mathsf{sh}_1, \ldots, \mathsf{sh}_t) \leftarrow_{\mathrm{R}} \mathsf{Share}(1^\lambda, \mathsf{prmtr}, 1^t, m_0)$.
- Run $\mathsf{pp} \leftarrow_{\mathrm{R}} \mathsf{FE.PPGen}(1^\lambda, \mathsf{prmtr}')$ and run $(\mathsf{FE.}\widetilde{\mathsf{pk}}_i, \mathsf{FE.td}_i) \leftarrow_{\mathrm{R}} \widetilde{\mathsf{Setup}}(\mathsf{pp})$ for $i \in [t]$. Set $\mathsf{pk} = (\mathsf{FE.}\widetilde{\mathsf{pk}}_1, \ldots, \mathsf{FE.}\widetilde{\mathsf{pk}}_t)$.
- Sample $\boldsymbol{\alpha} \in \{0,1\}^t$. Each is independently sampled where $\alpha_i = 1$ with probability ϵ, and 0 otherwise. If $\alpha_i = 1$, set $\gamma_i = (F, F(\mathsf{sh}_i))$, otherwise $\gamma_i = (\mathsf{sh}_i, F)$. Abort if $\boldsymbol{\alpha} = 0^t$.
- Compute $\mathsf{ct}[i] \leftarrow \widetilde{\mathsf{FE.Enc}}(\mathsf{FE.td}_i, \gamma_i)$ and $\mathsf{sk}_F[i] \leftarrow \widetilde{\mathsf{FE.KeyGen}}(\mathsf{FE.td}_i, F, \gamma_i)$ for $i \in [t]$.

- Let $\mathsf{ct} = (\mathsf{ct}[1], \ldots, \mathsf{ct}[t])$ and $\mathsf{sk}_f = (\mathsf{sk}_F[1], \ldots, \mathsf{sk}_F[t])$.
- Output $(\mathsf{pp}, \mathsf{pk}, \mathsf{ct}, \mathsf{sk}_f)$

The only difference between $\mathbf{Hybrid}_3$ and $\mathbf{Hybrid}_4$ is that we now share m_0 instead of m_b. We can formally show that if there is an attacker that distinguishes between these hybrids, then, it should break the security of HSS scheme. We can design the reduction as follows. Let $\mathcal{A}'$ be the adversary for distinguishing the hybrids.
Reduction $\mathcal{R}$

- $\mathcal{A}$ outputs (m_0, m_1), and a function f.
- Sample $\alpha \in \{0,1\}^t$. Each α_i for $i \in [t]$ is sampled independently to be 1 with probability ϵ and 0 with probability $1-\epsilon$. Abort if $\alpha = 0^t$, otherwise proceed. Let S be the set of indices where $\alpha_i = 0$. Note, $S \neq [t]$.
- Sample $b \leftarrow_{\mathrm{R}} \{0,1\}$. Send $f, S, (m_b, m_0)$ to the HSS challenger. The reduction gets back $\{\mathsf{sh}_i\}_{i \in S}$ and $\{F(\gamma_i)\}_{i \notin S}$. If $i \in S$, set $\gamma_i = (\mathsf{sh}_i, F)$ otherwise $\gamma_i = (F(\mathsf{sh}_i), F)$.
- Run $\mathsf{pp} \leftarrow_{\mathrm{R}} \mathsf{FE.PPGen}(1^\lambda, \mathsf{prmtr}')$ and sample $(\mathsf{FE.}\widetilde{\mathsf{pk}}_i, \mathsf{FE.td}_i) \leftarrow_{\mathrm{R}} \widetilde{\mathsf{Setup}}(\mathsf{pp})$ for $i \in [t]$. Set $\mathsf{pk} = (\mathsf{FE.}\widetilde{\mathsf{pk}}_1, \ldots, \mathsf{FE.}\widetilde{\mathsf{pk}}_t)$.
- Compute $\mathsf{ct}[i] \leftarrow \widetilde{\mathsf{FE.Enc}}(\mathsf{FE.td}_i, \gamma_i)$ and $\mathsf{sk}_F[i] \leftarrow \widetilde{\mathsf{FE.KeyGen}}(\mathsf{FE.td}_i, F, \gamma_i)$ for $i \in [t]$.
- Give $(\mathsf{pp}, \mathsf{pk}, \mathsf{ct}, \mathsf{sk}_f)$ to $\mathcal{A}'$. Output whatever it outputs.

Note that the view of $\mathcal{A}'$ is exactly as in $\mathbf{Hybrid}_3$ if m_b is shared, and it is exactly as $\mathbf{Hybrid}_4$ if m_0 is shared. If $\mathcal{A}'$ indeed succeeds in distinguishing these two hybrids, then our Reduction will succeed in the HSS game with the same advantage.

6 Definition of Structured-Seed PRG

Definition 6.1 (Syntax of Structured-Seed Pseudo-Random Generators (sPRG))**.** *Let τ be a positive constant. A structured-seed Boolean PRG,* sPRG*, with stretch τ that maps $(n \cdot \mathsf{poly}(\lambda))$-bit binary strings into $(m = n^\tau)$-bit strings, where* poly *is a fixed polynomial, is defined by the following PPT algorithms:*

- $\mathsf{PPGen}(1^\lambda, 1^n)$ *takes as input the security parameter λ, and an input length 1^n, which is a polynomial in λ. It outputs public parameters* pp*, which amongst other things contains an odd prime modulus $p(\lambda)$ which is* $\mathsf{poly}(\lambda)$ *bit prime for some polynomial independent of n.*
- $\mathsf{IdSamp}(\mathsf{pp})$ *samples a function index I.*
- $\mathsf{SdSamp}(I)$ *jointly samples two binary strings, a public seed and a private seed,* $\mathsf{sd} = (\mathsf{P}, \mathsf{S})$*. These are vectors over $\mathbb{Z}_p$. The combined dimension of these vectors is $n \cdot \mathsf{poly}(\lambda)$.*
- $\mathsf{Eval}(I, \mathsf{sd})$ *computes a string in $\{0,1\}^m$.*

Remark 6.1 (The modulus $p(\lambda)$)**.** The size of the modulus $p(\lambda)$ is some fixed polynomial in the security parameter λ independent of n.

Remark 6.2 (Polynomial Stretch.)**.** We say that an sPRG has polynomial stretch if $\tau > 1$ for some constant τ.

Remark 6.3 (Linear Efficiency.)**.** We say that an sPRG has linear-efficiency if the time to sample sd is $n \cdot \mathsf{poly}(\lambda)$.

Remark 6.4 (On $\mathsf{poly}(\lambda)$ multiplicative factor in the seed length.)**.** As opposed to a standard Boolean PRG definition where the length of the output is set to be n^τ where n is the seed length, we allow the length of the seed to increase multiplicatively by a fixed polynomial poly in a parameter λ. Looking ahead, one should view n as an arbitrary large polynomial in λ, and hence sPRG will be expanding in length.

Definition 6.2 (Security of sPRG)**.** *A structured-seed Boolean PRG,* sPRG*, satisfies*

Pseudorandomness: *Let $\lambda \in \mathbb{N}$ be the security parameter, let $n(\lambda)$ be a polynomial in λ. Then, following distributions are indistinguishable.*

$$(\mathsf{pp}, I, \mathsf{P},\ \mathsf{Eval}(I, \mathsf{sd}))$$
$$(\mathsf{pp}, I,\ \mathsf{P},\ \boldsymbol{r}\ )$$

where $\mathsf{pp} \leftarrow \mathsf{PPGen}(1^\lambda, 1^n),\ I \leftarrow \mathsf{IdSamp}(\mathsf{pp}),\ \mathsf{sd} \leftarrow \mathsf{SdSamp}(I),\ \boldsymbol{r} \leftarrow \{0,1\}^m$.

Definition 6.3 (Complexity and degree of sPRG)**.** *Let $D \in \mathbb{N}$, let $\lambda \in \mathbb{N}$ and $n = n(\lambda)$ be arbitrary positive polynomial in λ, and $p = p(\lambda)$ denote a prime modulus which is sampled during* PPGen*. Let $\mathbb{C}$ be a complexity class. A* sPRG *has complexity $\mathbb{C}$ in the public seed and degree D in private seed over $\mathbb{Z}_p$, denoted as,* $\mathsf{sPRG} \in (\mathbb{C}, \deg D)$*, if for every I in the support of* $\mathsf{IdSamp}(1^\lambda, 1^n)$*, there exists an algorithm* $\mathsf{Process}_I$ *in $\mathbb{C}$ and an $m(n)$-tuple of polynomials Q_I that can be efficiently generated from I, such that for all* sd *in the support of* $\mathsf{SdSamp}(I)$*, it holds that:*

$$\mathsf{Eval}(I, \mathsf{sd}) = Q_I(\mathsf{P}', \mathsf{S}) \text{ over } \mathbb{Z}_p\ ,\ \mathsf{P}' = \mathsf{Process}_I(\mathsf{P})\ ,$$

where Q_I has degree 1 in P *and degree D in* S*.*

We remark that the above definition generalizes the standard notion of families of PRGs in two aspects: 1) the seed consists of a public part and a private part, jointly sampled and arbitrarily correlated, and 2) the seed may not be uniform. Therefore, we obtain the standard notion as a special case.

Definition 6.4 (Pseudo-Random Generators, degree, and locality)**.** *A (uniform-seed) Boolean PRG (*PRG*) is an* sPRG *with a seed sampling algorithm* $\mathsf{SdSamp}(I)$ *that outputs a public seed* P *that is an empty string and a uniformly random private seed* $\mathsf{S} \leftarrow \{0,1\}^n$*, where the polynomial* poly *is fixed to be* 1*.*

Let $D, \mathsf{Loc} \in \mathbb{N}$. The PRG *has multilinear degree D if for every $n \in \mathbb{N}$ and I in the support of* $\mathsf{IdSamp}(1^n)$*, we have that* $\mathsf{Eval}(I, \mathsf{sd})$ *can be written as an $m(n)$-tuple of degree-D polynomials over $\mathbb{Z}$ in* S*. It has constant locality* Loc *if for every $n \in \mathbb{N}$ and I in the support of* $\mathsf{IdSamp}(1^n)$*, every output bit of* $\mathsf{Eval}(I, \mathsf{sd})$ *depends on at most* Loc *bits of* S*.*

6.1 Construction of sPRG and Our New Assumption

Our New Assumption. In this section, we describe our new assumption. Our new assumption is stronger than the one described next. The assumption is widely known in cryptography as the LWE with binary error assumption.

Definition 6.5 ($\mathsf{LWBE}_{\epsilon,\rho}$ Assumption)**.** *For any constants $\epsilon > 0$ and $\rho > 0$, we say that the assumption $\mathsf{LWBE}_{\epsilon,\rho}$ holds if for every odd prime modulus $p = O(2^{n^\rho})$ the following happens. We define two distributions below. The assumption requires that the following distributions are computationally indistinguishable:*

$\underline{\mathsf{PseudoR}_{\mathcal{A}}(1^n)}$:
$\boldsymbol{s} \leftarrow \mathbb{Z}_p^{n^{0.5+\epsilon}}$; $\boldsymbol{a}_i \leftarrow \mathbb{Z}_p^{n^{0.5+\epsilon}}$;
$e_i \leftarrow \{0,1\}\ \forall\ i \in [n]$;
Output $\left(\{\boldsymbol{a}_i, \langle \boldsymbol{a_i}, \boldsymbol{s}\rangle + e_i \mod p\}_{i\in[n]}\right)$

$\underline{\mathsf{Random}_{\mathcal{A}}(1^n)}$:
$\boldsymbol{s} \leftarrow \mathbb{Z}_p^{n^{0.5+\epsilon}}$; $\boldsymbol{a}_i \leftarrow \mathbb{Z}_p^{n^{0.5+\epsilon}}$;
$r_i \leftarrow \mathbb{Z}_p \forall\ i \in [n]$;
Output $\left(\{\boldsymbol{a}_i, r_i\}_{i\in[n]}\right)$

Formally, we say that $\mathsf{LWBE}_{\epsilon,p}$ holds if for any ppt distinguisher $\mathcal{A}$:

$$\mathsf{adv}_{\mathcal{A}}^{\mathsf{LWBE}_{\epsilon,p}}(1^n) := |\Pr[\mathcal{A}(z_1) = 1] - \Pr[\mathcal{A}(z_2) = 1]| < \mathsf{negl}(n),$$

where $z_1 \leftarrow \mathsf{PseudoR}_{\mathcal{A}}(1^n)$ and $z_2 \leftarrow \mathsf{Random}_{\mathcal{A}}(1^n)$.

We discuss the state of this assumption in Section A.4. Next, we describe our main new assumption which can be seen as an assumption arising from the interplay between the two assumptions described above (the assumption of LWBE and that of a pseudorandom generator with large enough stretch). We discuss the plausibility of this assumption too in Section A.4.

Definition 6.6 ($\mathsf{G\text{-}LWEleak}_{d,\epsilon,\rho}$ Security)**.** *For any constant integer $D > 0$, constants $\epsilon > 0$ and $\rho \in (0, 0.5)$, we say that a degree D pseudorandom generator G of stretch at least $m(n) \geq n^{\lceil \frac{D}{2} \rceil \cdot (0.5+\epsilon)+\rho}$ satisfies $\mathsf{G\text{-}LWEleak}_{d,\epsilon,\rho}$-security if for any odd prime modulus $p = O(2^{n^\rho})$, the following two distributions are computationally indistinguishable:*

$\underline{\mathsf{PseudoR}^G_{\mathcal{A}}(1^n)}$:
$\boldsymbol{s} \leftarrow \mathbb{Z}_p^{n^{0.5+\epsilon}}$; $\boldsymbol{a}_i \leftarrow \mathbb{Z}_p^{n^{0.5+\epsilon}}$;
$e_i \leftarrow \{0,1\}\ \forall\ i \in [n]$;
Output $\left(\{\boldsymbol{a}_i, \langle \boldsymbol{a_i}, \boldsymbol{s}\rangle + e_i \mod p\}_{i\in[n]}, \mathsf{G}(\boldsymbol{e})\right)$

$\underline{\mathsf{Random}^G_{\mathcal{A}}(1^n)}$:
$\boldsymbol{s} \leftarrow \mathbb{Z}_p^{n^{0.5+\epsilon}}$; $\boldsymbol{a}_i \leftarrow \mathbb{Z}_p^{n^{0.5+\epsilon}}$;
$e_i \leftarrow \{0,1\}\ \forall\ i \in [n]$;
$r \leftarrow \{0,1\}^m$;
Output $\left(\{\boldsymbol{a}_i, \langle \boldsymbol{a_i}, \boldsymbol{s}\rangle + e_i \mod p\}_{i\in[n]}, r\right)$

Formally, we say that G satisfies $\mathsf{LWEleak}_{d,\epsilon,p}$ if for any ppt distinguisher $\mathcal{A}$:

$$\mathsf{adv}^{\mathsf{LWEleak}_{d,\epsilon,\rho}}_{\mathsf{G},\mathcal{A}}(1^n) := |\Pr[\mathcal{A}(z_1) = 1] - \Pr[\mathcal{A}(z_2) = 1]| < \mathsf{negl}(n),$$

where $z_1 \leftarrow \mathsf{PseudoR}^{\mathsf{G}}_{\mathcal{A}}(1^n)$ and $z_2 \leftarrow \mathsf{Random}^{G}_{\mathcal{A}}(1^n)$.

Further, we say the assumption is subexponetially secure if $\mathsf{negl} = 2^{-n^{\Omega(1)}}$.

Construction of sPRG**.** We now describe our construction using the assumption above for parameters $D \in \mathbb{N}$, $\epsilon, \rho > 0$, and then argue properties. Consider the following setting of parameters:

- λ is the security parameter,
- $n(\lambda)$ is an arbitrary polynomial in λ which is the length parameter,
- $n' = n^{\frac{1}{(0.5+\epsilon)\cdot\lceil\frac{D}{2}\rceil}}$,
- $m(n') \geq n'^{\lceil\frac{D}{2}\rceil\cdot(0.5+\epsilon)+\rho} = n^{1+\gamma}$ where $\gamma = \frac{\rho}{(0.5+\epsilon)\lceil\frac{D}{2}\rceil}$. Let $\tau = 1 + \gamma > 1$. This will be the stretch of our sPRG.

$\mathsf{sPRG.PPGen}(1^\lambda, 1^n)$: Run Bilinear map generator on input λ to sample $\mathcal{G}_\lambda = (\mathbf{G}_1, \mathbf{G}_2, \mathbf{G}_T, p, P_1, P_2, e)$. Output $\mathsf{pp} = (\mathcal{G}_\lambda, p, 1^n)$. The modulus p is implicitly a part of $\mathcal{G}_\lambda$ and is a $\mathsf{poly}(\lambda)$ bit modulus.

$\mathsf{sPRG.IdSamp}(\mathsf{pp})$: Sample $\boldsymbol{a}_i \leftarrow \mathbb{Z}_p^{n'^{0.5+\epsilon}}$ for $i \in [n']$. Output $I = (\{\boldsymbol{a}_i\}_{i\in[n']}, \mathsf{G})$.

$\mathsf{sPRG.SdSamp}(I)$: Sample $\boldsymbol{s} \leftarrow_{\mathrm{R}} \mathbb{Z}_p^{n'^{0.5+\epsilon}}$. For every $i \in [n']$, sample $e_i \leftarrow \{0,1\}$. Set $b_i = \langle \boldsymbol{a}_i, \boldsymbol{s}\rangle + e_i \mod p$ for $i \in [n']$. Set $\mathsf{P} = \boldsymbol{b}$ where $\boldsymbol{b} = (b_1, \ldots, b_{n'})$. Compute $\mathsf{S} = (1, \boldsymbol{s})^{\otimes\lceil\frac{D}{2}\rceil}$. Output $\mathsf{sd} = (\mathsf{P}, \mathsf{S})$.

$\mathsf{sPRG.Eval}(I, \mathsf{sd})$: Output $\boldsymbol{\sigma} = G(b_1 - \langle \boldsymbol{a}_1, \boldsymbol{s}\rangle, \ldots, b_{n'} - \langle \boldsymbol{a}_{n'}, \boldsymbol{s}\rangle) \in \{0,1\}^{m(n')}$. Note that every bit σ_i is computable by a polynomial map $Q_I(\mathsf{P}, \mathsf{S}) \to \mathbb{Z}_p^{m(n')}$ which is a degree D map in P and degree 2 map in S over $\mathbb{Z}_p$.

We now argue properties associated with it.

Stretch. Note that the length of the output is n^τ, where as the length of the seed is $O(n' \log_2 p + n'^{(0.5+\epsilon)\lceil\frac{D}{2}\rceil} \log_2 p) = O(n\,\mathsf{poly}(\lambda))$, because p is a $\mathsf{poly}(\lambda)$ bit prime for some polynomial independent of n. Thus the sPRG has a stretch of τ.

Linear Efficiency. We now compute the run time to sample sd. It computes n' LWE samples, each of dimension $n'^{0.5+\epsilon}$. This can be done in time $n'^{1.5+\epsilon}\,\mathsf{poly}(\lambda) = n^{\frac{1}{\lceil\frac{D}{2}\rceil}} \cdot n^{\frac{1}{(0.5+\epsilon)\lceil\frac{D}{2}\rceil}}\,\mathsf{poly}(\lambda) = O(n)\,\mathsf{poly}(\lambda)$ if $D \geq 5$. It also computes S, which also takes linear time as: $O(n'^{(0.5+\epsilon)\lceil\frac{D}{2}\rceil}\,\mathsf{poly}(\lambda)) = O(n\,\mathsf{poly}(\lambda))$. Thus, it satisfies linear efficiency if $D \geq 5$.

Pseudorandomness. The adversary gets to see $\mathsf{P} = \boldsymbol{b}$ along with $\boldsymbol{\sigma} = G(\boldsymbol{e})$. The Pseudorandomness property follows immediately from the assumption $\mathsf{LWEleak}_{d,\epsilon,\rho}$.

Theorem 6.1. *Let $\epsilon, \rho > 0$ and $D \in \mathbb{N}$. If the assumption $\mathsf{LWEleak}_{D,\epsilon,\rho}$ holds for some PRG G, then there exists $\tau > 1$ such that the construction sPRG above is a secure structured seed PRG with stretch τ, with complexity $(\mathsf{arith\text{-}NC}^0, \deg\ 2)$. If $D \geq 5$, the sPRG also satisfies linear efficiency.*

7 Single Ciphertext Functional Encryption with Linear KeyGen from LWE

In this section, for any constant $\epsilon \in (0, 1)$, we construct a variant of secret key functional encryption satisfying the following specifications. We denote this primitive by ϵ-1LGFE.

- (Function Class.) The function class for ϵ-1LGFE is $\mathcal{F}^{\mathsf{CIRC}} = \{\mathcal{F}^{\mathsf{CIRC}}_{\lambda,\mathsf{prmtr}}\}_{\lambda,\mathsf{prmtr}}$ which consists of all polynomial sized Boolean circuits that takes n bit inputs, produces ℓ-bit outputs, has depth bounded by d, and size bounded by size. Here all parameters $n, \ell, d, \mathsf{size}$ are polynomially related to the security parameter, and can be arbitrary polynomials.
- (Security.) Satisfy 1-key single ciphertext ϵ-simulation security as in Definition 4.4. Single ciphertext specifies that the number of ciphertexts is 1 and 1-key specifies that the number of secret keys is also 1 (i.e., $Q_{\mathsf{sk}} = 1$ in Definition 4.4). Note however that the functions supported directly have many output bits.
- (Efficiency.) Satisfy *levelled* compactness as in Definition 4.7 and Remark 4.8. In particular, ciphertext size is $\mathsf{poly}(\lambda, n, d)$, independent of the function size size and output length ℓ.
- Also admits Special Structure* defined in Definition 4.8.

We will construct such a scheme relying on the GVW predicate encryption scheme [GVW15]. Below we recall some preliminaries from there and then we construct ϵ-1LGFE.

7.1 GVW Preliminaries

Predicate Encryption. Now we recall the definition of predicate encryption scheme. A predicate encryption is a functional encryption scheme as described in Section 4, with the following differences.

- Encryptor encrypts messages of the form (attr, m) where $\mathsf{attr} \in \{0,1\}^n$ and m is a bit.
- The functions supported has special form C_P, where P is a *predicate* in the class $\mathcal{F}^{\mathsf{CIRC}}_{\lambda,n,d,1,\mathsf{size}}$ which on input (attr, m) outputs m if $P(\mathsf{attr}) = 1$ and 0 otherwise.

- (Selective) indistinguishability security of PE states that given the public key PK, the adversary cannot distinguish encryption of (attr_0, m_0) from encryption of (attr_1, m_1), even if it can ask for any number of functional keys corresponding to predicates $P_1, ..., P_\eta$, as long as for all $i \in [\eta]$, $P_i(\mathsf{attr}_0) = P_i(\mathsf{attr}_1) = 0$. In the security experiments, the challenges attributes attr_0 and attr_1 are chosen selectively at the beginning.

For a complete definition, please refer to [GVW15]. For our construction, we require some special properties from the predicate encryption scheme, such as efficiency, circuit homomorphism, etc. All these properties are satisfied by the construction of [GVW15], and we recall them next. The text below is a very succinct summary of the [GVW15] scheme, and will assume familiarity with some lattice preliminaries described in Section B.

Properties of GVW Predicate Encryption Scheme. We first describe various algorithms and associated properties of the GVW predicate encryption scheme PE.

Setup. The setup algorithm takes as input the security parameter λ, and outputs a public key PK and a secret key SK. Namely, $\mathsf{Setup}(1^\lambda, \mathsf{prmtr} = (n, \ell, d, \mathsf{size})) \to (\mathsf{PK}, \mathsf{SK})$, where PK contains the following parameters:

- A polynomial-length modulus p_1. p_1 is of magnitude $2^{\mathsf{poly}(\lambda,d)}$ and is set below.
- LWE matrix dimensions $\dim_1 = \mathsf{poly}_2(\lambda, d)$ and $\dim_2 = \mathsf{poly}_3(\lambda, d)$ which are some fixed polynomials in the security parameter.
- Uniform matrices $\mathbf{B}_1, ..., \mathbf{B}_{n'}, \mathbf{A}, \mathbf{D}$ where $n' = n \cdot \mathsf{poly}(\lambda, d)$ for some polynomial poly determined below. Each matrix is in $\mathbb{F}_{p_1}^{\dim_1 \times \dim_2}$. This is also the space of the gadget matrix $\mathbf{G}$.

Encryption. The encryption algorithm takes as input public key PK, attribute $\mathsf{attr} \in \{0,1\}^n$ and a message $m \in \{0,1\}$ and outputs a two-part ciphertext, $\mathsf{Enc}(\mathsf{PK}, \mathsf{attr}, m) \to (\mathsf{ct}_1, \mathsf{ct}_2)$, which we describe in more detail below.

- The encryption algorithm first samples a secret vector $\boldsymbol{s}$ from $\chi^{\dim_1 \times 1}$. Here χ is an LWE error distribution that is bounded by a fixed polynomial $\mathsf{poly}_\chi(\lambda, d)$.
- Then, it encodes attr into $\widehat{\mathsf{attr}} = (\widehat{\mathsf{attr}}_p, \widehat{\mathsf{attr}}_s) \in \mathbb{F}_{p_1}^{n'}$ of length n', where $\widehat{\mathsf{attr}}_p$ can and will be released in public and $\widehat{\mathsf{attr}}_p$ must be kept secret. In the [GVW15] scheme, $\widehat{\mathsf{attr}}_s$ is an FHE secret key and $\widehat{\mathsf{attr}}_p$ is an FHE encryption of attr under that secret key. Below, we will only use the fact that the size of $|\widehat{\mathsf{attr}}_p| + |\widehat{\mathsf{attr}}_s| = n' = \mathsf{poly}(\lambda, n, d) \log p_1$, and will not rely on other details of $(\widehat{\mathsf{attr}}_p, \widehat{\mathsf{attr}}_s)$.
- Now ct_1 is constructed using algorithm $\mathsf{Enc}_1(\mathsf{PK}, \boldsymbol{s}, \mathsf{attr})$ which does:
  - Compute $\boldsymbol{b}_i = \boldsymbol{s}^T(\mathbf{B}_i + \widehat{\mathsf{attr}}_i \mathbf{G}) + \boldsymbol{e}_i$ for $i \in [n']$. Here $\boldsymbol{e}_i \leftarrow \chi^{1 \times \dim_2}$.

- Output $\mathsf{ct}_1 = (\boldsymbol{b}_1, \ldots, \boldsymbol{b}_{n'}, \widehat{\mathsf{attr}_p})$

- Now ct_2 is constructed as follows.
  - Compute $\boldsymbol{a} = \boldsymbol{s}^T\mathbf{A} + \boldsymbol{e}_1$. Here $\boldsymbol{e}_1 \leftarrow \chi^{1\times \dim_2}$.
  - Compute $\boldsymbol{d} = \boldsymbol{s}^T\mathbf{D} + \boldsymbol{e}_2 + m\lfloor p_1/2\rfloor[1, 0, ..., 0]$. Here $\boldsymbol{e}_2 \leftarrow \chi^{1\times \dim_2}$.
  - Output $\mathsf{ct}_2 = (\boldsymbol{a}, \boldsymbol{d})$.

Without loss of security we can assume $\boldsymbol{s}[1] = 1$ (first component of vector $\boldsymbol{s}$). This ensures that $\boldsymbol{v} = \boldsymbol{s}^T\mathbf{G}$ satisfies $\boldsymbol{v}[1] = 1$.

Note: Looking ahead, in our construction of ϵ-1LGFE, we will only use the Enc_1 algorithm instead of the encryption algorithm, thereby not computing ct_2 at all. This does not hamper security as we are just giving less information.

Evaluation. There are two algorithms: EvalPK operates only on the public key PK and EvalCT operates on both PK and ct_1.

- $\mathsf{EvalPK}(C, \mathbf{B}_1, ..., \mathbf{B}_{n'}) \to \mathbf{B}_C$: EvalPK on input $\mathbf{B}_1, \ldots, \mathbf{B}_{n'}$ contained in PK, and a predicate $C \in \mathcal{F}_{n,d,1,\mathsf{size}}$, deterministically outputs $\mathbf{B}_C \in \mathbb{F}_{p_1}^{\dim_1 \times \dim_2}$.

- $\mathsf{EvalCT}(\mathsf{PK}, C, \mathsf{ct}_1) \to b_C$: EvalCT on input ct_1 containing $\widehat{\mathsf{attr}_p}$, $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_{n'}$ and a predicate C, output a scalor $b_C \in \mathbb{F}_{p_1}$, that satisfies the following important identity: Let $\mathbf{B}_C[\star, 1]$ denotes the first column of $\mathbf{B}$;

$$\begin{aligned} b_C &= \boldsymbol{s}^T\left(\mathbf{B}_C[\star, 1] + (C(\mathsf{attr})\lfloor p_1/2\rfloor + e'_C)\mathbf{G}\right) + \boldsymbol{e}_C[1] \\ &= \boldsymbol{s}^T\mathbf{B}_C[\star, 1] + C(\mathsf{attr})\lfloor p_1/2\rfloor + e_C \end{aligned} \tag{2}$$

The second identity above follows from the fact that $(\boldsymbol{s}^T\mathbf{G})[1] = 1$ and setting $e_C = e'_C + \boldsymbol{e}_C[1]$. It satisfies that $|e_C|/p_1 < 2^{-\lambda^c}$ for some constant $c > 0$. (For readers familiar with the [GVW15] PE scheme, e'_C is an error resulted from the homomorphic evaluation of the FHE scheme, which is polynomially bounded, and $\boldsymbol{e}_C[1]$ is the error resulted from the homomorphic evaluation of the ABE scheme, which is exponentially large. These details are not important for our ϵ-1LGFE construction, which only relies on Equation (2).)

Remark 7.1. The algorithms described above are already close enough to imply a construction of ϵ-1LGFE where the encryption is simply Enc_1 producing ct_1 as above, the master secret key is $\boldsymbol{s}$, and the function key for any function C could just be computed as $\mathsf{sk}_C = \langle \boldsymbol{s}, \mathbf{B}[\star, 1]\rangle + e$, where e is chosen freshly from some bounded smudging distribution. This leads to the evaluation-decryption equation:

$$\mathsf{EvalCT}(\mathsf{PK}, C, \mathsf{ct}_1) - \mathsf{sk}_C = C(\mathsf{attr}) \cdot \lceil p_1/2\rceil + e_C - e$$

The only problem is that e_C is not polynomially bounded, and thus this does not fit in the requirements for an ϵ-1LGFE scheme, which requires both e and $e_C - e$ to be polynomially bounded. To fix this issue, we introduce the following algorithm, which rounds $b_C \in \mathbb{F}_{p_1}$ to another modulus p so that the rounded version of the error e'_C is polynomially bounded.

Rounding-Evaluation. We now describe a procedure of rounding evaluation, RoundEval, which can be done publicly. $\mathsf{RoundEval}(\mathsf{PK}, C, \mathsf{ct}_1, p)$, takes as input $\mathsf{PK}, \mathsf{ct}_1 = (\widehat{\mathsf{attr}}_p, \boldsymbol{b}_1, \ldots, \boldsymbol{b}_{n'})$, a predicate C, and a modulus $p < p_1$, and does the following:

1. Run $\mathsf{EvalCT}(\mathsf{PK}, C, \mathsf{ct}_1) \to b_C$.
2. Compute $b'_C = \lceil p/p_1 \cdot b_C \rfloor$. Namely multiply b_C with p/p_1 over the reals and then take the nearest integer. Output $b'_C \in \mathbb{F}_p$.

Now we observe the structure of b'_C, relying on the lemma proven about rounding in [BGV12] (see lemma 1 of the paper).

Theorem 7.1. *For any primes p and $p_1 > p$, given that*

- $b_C = \boldsymbol{s}^T \mathbf{B}_C[\star, 1] + C(\mathsf{attr})\lfloor p_1/2 \rfloor + e_C$ *where $\boldsymbol{s}$ is sampled from the distribution $\chi^{\dim_1}$, and*
- χ *is a polynomially bounded distribution, bounded by,* $\mathsf{poly}_\chi(\lambda, d)$,

$$b'_C = \boldsymbol{s}^T \cdot \boldsymbol{d}_C + C(\mathsf{attr})\lfloor p/2 \rfloor + e'_C + e_{\mathrm{rnd}} \quad (3)$$
$$\text{where } \boldsymbol{d}_C = \mathsf{Round}_{p_1 \to p}(\mathbf{B}_C[\star, 1]),\ e'_C = \mathsf{Round}_{p_1 \to p}(e_C)$$

where $\mathsf{Round}_{p_1 \to p}(v) = \lceil p \cdot v/p_1 \rfloor$ *and e_{rnd} is the rounding error satisfying* $|e_{\mathrm{rnd}}| < \dim_1 \cdot \mathsf{poly}_\chi(\lambda, d) + 5$.

Below, we will refer to b'_C as the output ciphertext from the RoundEval procedure, and $\mathsf{error} = (e'_C + e_{\mathrm{rnd}})$ the noise or error in this output ciphertext.

7.2 Setting Parameters

Now we set the parameters that will be used in our constructions of ϵ-1LGFE. All these parameters can be realized using standard LWE assumption with subexponential approximation factors.

- $\dim_1$ and $\dim_2$ are chosen as in the [GVW15] predicate encryption scheme.
- The error distribution χ is a truncated discrete Gaussian distribution bounded by $\mathsf{poly}_\chi(\lambda, d)$.
- The prime p is exactly the order of a prime-order bilinear pairing group, which is sub-exponentially large in the security parameter. (More precisely, $\{p_\lambda\}$ is exactly the order of a sequence of prime-order bilinear pairing groups.)
- The prime modulus p_1 is so that part of the error e'_C as shown in Equation (3) in the output ciphertext from the RoundEval procedure is bounded by λ.

By the properties of the PE scheme of [GVW15], the error resulted by evaluating a predicate grows exponentially with the security parameter λ and the depth d of the predicate.

$$|e_C| \leq (\dim_1 + \dim_2)^{\mathsf{poly}_{\mathsf{PE}}(\lambda,d)} \cdot \mathsf{poly}_\chi(\lambda, d)$$

where $\mathsf{poly}_{\mathsf{PE}}$ is a fixed polynomial decided by the PE scheme.

Thus p_1 must satisfy:

$$|e'_C| \leq \left\lceil \frac{p \cdot (\dim_1 + \dim_2)^{\mathsf{poly}_{\mathsf{PE}}(\lambda,d)} \cdot \mathsf{poly}_\chi(\lambda, d)}{p_1} \right\rceil \leq \lambda$$

Set p_1 to any prime satisfying the above inequality, of magnitude:

$$p_1 = \Theta\left(p(\dim_1 + \dim_2)^{\mathsf{poly}_{\mathsf{PE}}(\lambda,d)} \cdot \mathsf{poly}_\chi(\lambda, d)\right) \ ,$$

The bit length of the prime p_1 is $\log p + \mathsf{poly}(\lambda, d) = \mathsf{poly}'(\lambda, d)$.

By the above way of setting parameters, we have that the error in the output ciphertexts by $\mathsf{RoundEval}$ is bounded by a fixed polynomial in λ.

Claim 7.1. *Under the above way of setting parameters, there is a universal polynomial* $\mathsf{Bnd}(\cdot)$*, such that, for every* $\lambda \in \mathbb{N}$*, predicate* $C \in \mathcal{F}_{n,d,1,\mathsf{size}}$*, every input* $x \in \{0,1\}^n$*, every* PK *in the support of* $\mathsf{Setup}(1^\lambda, \mathsf{prmtr} = (n, \ell, d, \mathsf{size}))$*, ciphertext* ct_1 *in the support of* Enc_1*, and every output ciphertext* b'_C *in the support of* $\mathsf{RoundEval}(\mathsf{PK}, C, \mathsf{ct}_1, p)$*, the error* error *in* b'_C *satisfies* $|\mathsf{error}| \leq \mathsf{Bnd}(\lambda, d)$*.*

Proof. The claim follows from the fact that $\mathsf{error} = e'_C + e_{\mathrm{rnd}}$, where the former is bounded by λ due to how the modulus p_1 is set, and the latter is bounded by $\dim_1 \cdot \mathsf{poly}_\chi(\lambda, d) + 5$ by Theorem 7.1. Therefore, we set simply $\mathsf{Bnd} = \lambda + \dim_1 \cdot \mathsf{poly}_\chi(\lambda, d) + 5$. □

Efficiency: We particularly remark about the ciphertext size $|\mathsf{ct}_1|$ of the PE scheme, which will dictate the ciphertext size of our ϵ-1LGFE below. The $\mathsf{ct}_1 = (\boldsymbol{b}_1, \cdots, \boldsymbol{b}_{n'}, \widehat{\mathsf{attr}}_p)$, where each $\boldsymbol{b}_i \in \mathbb{Z}_{p_1}^{1\times \dim_2}$. By our parameter setting above p_1 has $\mathsf{poly}(\lambda, d)$ bits. Therefore $n' = |\widehat{\mathsf{attr}}_p| + |\widehat{\mathsf{attr}}_s| = \mathsf{poly}(\lambda, n) \log p_1 = \mathsf{poly}(\lambda, n, d)$. Since further $\dim_2 = \mathsf{poly}(\lambda, d)$, the total size of ct_1 is bounded by $\mathsf{poly}(\lambda, n, d)$.

7.3 Construction of ϵ-1LGFE

We adapt the predicate encryption of [GVW15] with additional rounding to our the construction of ϵ-1LGFE, which is described in Figure 7.3. To avoid confusion, every algorithm X of the PE scheme will be denoted as $\mathsf{PE.X}$.

Observe that correctness and syntactic properties are immediate due to the properties of the predicate encryption scheme. For completeness, we sketch these below.

ϵ-1LGFE.PPGen$(1^\lambda, \mathsf{prmtr} = (n, d, \ell, \mathsf{size}))$:

- Set p to be the prime order of a fixed bilinear pairing group for security parameter λ.
- Run PE.Setup$(1^\lambda, \mathsf{prmtr}) \to (\mathsf{PK}, \mathsf{SK})$, where $\mathsf{PK} = ((\mathbf{B}_1, ..., \mathbf{B}_{n'}), p_1)$. Set $\mathsf{pp} = (p, \mathsf{prmtr}, \mathsf{PK})$ (and discard SK.)

ϵ-1LGFE.Setup(pp) : Sample $\boldsymbol{s} \leftarrow \chi^{\dim_1 \times 1}$. Set $\mathsf{msk} = \boldsymbol{s}$.

ϵ-1LGFE.Enc$(\boldsymbol{s}, m)$: Run PE.Enc$_1(\mathsf{PK}, \boldsymbol{s}, m) \to \mathsf{ct}$. Output ct.

ϵ-1LGFE.KeyGen$(\boldsymbol{s}, C)$: C has ℓ-bit outputs. For every $j \in [\ell]$,

- set C_j to be the depth d circuit that computes the j'th output bit,
- compute PE.EvalPK$(\mathsf{PK}, C_j) \to \mathbf{B}_{C_j}$, and $\boldsymbol{d}_{C_j} = \mathsf{Round}_{p_1 \to p}(\mathbf{B}_{C_j}[\star, 1])$, the rounded version of the first column of $\mathbf{B}_{C_j}$,
- compute $\mathsf{sk}_{C_j} = \langle \boldsymbol{d}_{C_j}, \boldsymbol{s} \rangle + e_j \bmod p$ where $e_j \leftarrow [-\mathsf{Bnd}', \mathsf{Bnd}'] \cap \mathbb{Z}$ and $\mathsf{Bnd}' = \mathsf{Bnd}'(\lambda)$ is a fixed polynomial set to: $\mathsf{Bnd}'(\lambda) = \lceil \frac{\mathsf{Bnd} \cdot \ell}{1-\epsilon} \rceil$.

Output secret key $\mathsf{sk}_C = \{\mathsf{sk}_{C_j}\}_{j \in [\ell]}$

ϵ-1LGFE.Dec$(\mathsf{sk}_C, \mathsf{ct})$: For every $j \in [\ell]$, compute PE.RoundEval$(\mathsf{PK}, C_j, \mathsf{ct}, p) \to b'_{C_j}$, and if $z_j = (b'_{C_j} - \mathsf{sk}_{C_j}) \bmod p \in [-p/4, p/4]$, set $y_j = 0$, and otherwise set $y_j = 1$. Output y.

Figure 1: Construction of ϵ-1LGFE.

Correctness: If the setup, encryption, and the key generation are done honestly, then from the properties of the PE scheme, the following happens. Let ct denote a ciphertext encrypting $m \in \{0,1\}^n$, and let $\mathsf{sk}_C = \{\mathsf{sk}_{C_j}\}$ be a function secret key for a circuit $C \in \mathcal{F}^{\mathsf{CIRC}}_{\lambda,\mathsf{prmtr}}$. Then, for every $j \in [\ell]$,

$$b'_{C_j} = \mathsf{PE.RoundEval}(\mathsf{PK}, C_j, \mathsf{ct}, p) = \boldsymbol{s}^T \cdot \boldsymbol{d}_{C_j} + C_j(m)\lfloor p/2 \rfloor + \mathsf{error}_j$$
$$\mathsf{sk}_C = \boldsymbol{s}^T \cdot \boldsymbol{d}_{C_j} + e_j, \qquad \text{where } \boldsymbol{d}_{C_j} = \mathsf{Round}_{p_1 \to p}(\mathbf{B}_{C_j}[\star, 1]) \ .$$

Moreover, e_j is bounded by a polynomial Bnd', and by Claim 7.1, error_j is bounded by another polynomial Bnd. Decryption computes

$$z_j = b'_{C_j} - \mathsf{sk}_{C_j} \bmod p = C_j(m)\lceil p/2 \rceil + \mathsf{error}'_j, \quad \mathsf{error}'_j = e_j - \mathsf{error}_j \ .$$

Because p is subexponentially large and error'_j is polynomially bounded, if $C_j(m) = 0$ then $z \in [-p/4, p/4]$ and otherwise not. Therefore, the decryption algorithm recovers every bit of the output correctly.

Special Structure*. It is easy to verify that our ϵ-1LGFE has the special structure*: We have four polynomials q_1, $\dim_1$, Bnd', $\mathsf{Bnd} + \mathsf{Bnd}'$, such that, all required properties are satisfied.

- (PP Syntax.) The pp contains modulus p, which is a $q_1 = \mathsf{poly}(\lambda)$-bit prime.
- (Linear secret key Structure.) The master secret key is a vector in $\boldsymbol{s} \in \mathbb{Z}_{p_1}^{\dim_1}$. For any function $C \in \mathcal{F}^{\mathsf{CIRC}}_{\lambda,n,\ell,d,\mathsf{size}}$, the functional secret key $\mathsf{sk}_C = \{\mathsf{sk}_{C_j}\}$ and each sk_{C_j} satisfies the special form of $\mathsf{sk}_{C_j} = \langle \boldsymbol{d}_{C_j}, \boldsymbol{s} \rangle + e_j \bmod p$ where $e_j \leftarrow_{\mathrm{R}} [-\mathsf{Bnd}', \mathsf{Bnd}'] \cap \mathbb{Z}$. Finally, d_{C_j} is deterministically and efficiently computed from the PK contained in pp and f.
- (Linear + Round Decryption with polynomial decryption error.) The procedure $\mathsf{PE.RoundEval}(\mathsf{PK}, C_j, \mathsf{ct}, p)$ computes output ciphertext b'_{C_j}, such that, $|b'_{C_j} - \langle \boldsymbol{d}_{C_j}, \boldsymbol{s} \rangle - C_j(m)\lfloor p/2 \rfloor\rangle| \leq \mathsf{Bnd}' + \mathsf{Bnd}$.

Remark 7.2. Note that above we consider e_j to be sampled from $[-\mathsf{Bnd}', \mathsf{Bnd}']$ but equivalently we could have considered it being sampled from $[0, 2\mathsf{Bnd}']$ (which was the requirement in Definition 4.8). This is because the keys can be transformed from one into other by adding a fixed constant Bnd'.

Leveled Compactness: Observe that the ciphertext ct for a message m is exactly the ct_1 generated by $\mathsf{PE.Enc}_1(\mathsf{PK}, \boldsymbol{s}, m)$, which as analyzed in Section 7.1 is $\mathsf{poly}(\lambda, n, d)$.

7.4 Single-Ciphertext ϵ-simulation security

For any fixed constant $\epsilon \in (0,1)$, we now prove that the construction in Figure 7.3 is 1-key single-ciphertext ϵ-simulation secure.

Theorem 7.2. *Assuming LWE assumption holds for the parameters described in Section 7.2, the construction ϵ-1LGFE in Figure 7.3 satisfies 1-key single-ciphertext ϵ-simulation security.*

$\underline{\mathsf{Real}_{\mathcal{A}}^{\epsilon\text{-1LGFE}}(1^\lambda)}$:

1. *Challenge:* $(m,\ C) \leftarrow \mathcal{A}_1(1^\lambda, \mathsf{prmtr} = (n, d, \ell, \mathsf{size}))$ (where n, d, ℓ, and size are polynomials, $m \in \{0,1\}^n$, and $C \in \mathcal{F}^{\mathsf{CIRC}}_{\lambda,n,\ell,d,\mathsf{size}}$). Below $n = n(\lambda)$, $\ell = \ell(\lambda)$, $d = d(\lambda)$, $\mathsf{size} = \mathsf{size}(\lambda)$, and $\mathsf{prmtr} = (n, \ell, d, \mathsf{size})$.

2. *Generate* pp*:* $\mathsf{pp} \leftarrow \epsilon\text{-1LGFE.PPGen}(1^\lambda, \mathsf{prmtr})$, where $\mathsf{pp} = (p, \mathsf{prmtr}, \mathsf{PK})$ and $\mathsf{PK} = ((\mathbf{B}_1, ..., \mathbf{B}_{n'}), p_1)$.

3. *Generate master secret key:* $\boldsymbol{s} \leftarrow \epsilon\text{-1LGFE.Setup}(\mathsf{pp})$, where $\boldsymbol{s} \leftarrow \chi^{\dim_1 \times 1}$.

4. *Generate the challenge ciphertext:* $\mathsf{ct} \leftarrow \epsilon\text{-1LGFE.Enc}(\boldsymbol{s}, m)$, where ct is generated using $\mathsf{PE.Enc}_1(\mathsf{PK}, \boldsymbol{s}, m)$.

5. *Generate the challenge secret key:* $\mathsf{sk}_C \leftarrow \epsilon\text{-1LGFE.KeyGen}(\boldsymbol{s}, C)$, where $\mathsf{sk}_C = \{\mathsf{sk}_{C_j}\}_{j\in[\ell]}$, and $\mathsf{sk}_{C_j} = \langle \boldsymbol{d}_{C_j}, \boldsymbol{s}\rangle + e_j \bmod p$, for $\mathbf{B}_{C_j} \leftarrow \mathsf{PE.EvalPK}(\mathsf{PK}, C_j)$, $\boldsymbol{d}_{C_j} = \mathsf{Round}_{p_1\to p}(\mathbf{B}_{C_j}[\star, 1])$, and $e_j \leftarrow [-\mathsf{Bnd}', \mathsf{Bnd}'] \cap \mathbb{Z}$.

Output $\beta \leftarrow \mathcal{A}_2(\{\mathsf{pp}, \mathsf{ct}, \mathsf{sk}_C\})$.

Figure 2: The real-world ϵ-simulation security game of ϵ-1LGFE.

Proof. By definition 4.3, to prove single ciphertext ϵ-simulation security, we need to show a PPT simulator $\mathcal{S}$, such that for all stateful PPT adversaries $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, there is a negligible function negl, such that, for any security parameters $\lambda \in \mathbb{N}$,

$$\mathsf{adv}^{\mathsf{SIM}}_{\epsilon\text{-1LGFE},\mathcal{A}}(1^\lambda) := |\Pr[1 \leftarrow \mathsf{Real}^{\epsilon\text{-1LGFE}}_{\mathcal{A}}(1^\lambda)] - \Pr[1 \leftarrow \mathsf{Ideal}^{\epsilon\text{-1LGFE}}_{\mathcal{A},\mathcal{S}}(1^\lambda)]| < \mathsf{negl}(\lambda),$$

where the Real experiment is described in Figure 7.4, and the ideal experiment Ideal defined in figure 7.4, which also defines out simulator $\mathcal{S}$. We show that the view of $\mathcal{A}$ in the real and ideal experiments are indistinguishable (and hence has the same probability of outputting 1) through two intermediate hybrids:

Hybrid H_1**:** This hybrid is identical to the real world experiment except for how sk_C is generated: by construction of ϵ-1LGFE, it holds that

$$b'_{C_j} = \mathsf{PE.RoundEval}(\mathsf{PK}, C_j, \mathsf{ct}, p) = \boldsymbol{s}^T \cdot \boldsymbol{d}_{C_j} + y_j \lfloor p/2 \rfloor + \mathsf{error}_j$$
$$\mathsf{sk}_C = \boldsymbol{s}^T \cdot \boldsymbol{d}_{C_j} + e_j$$
where $\boldsymbol{d}_{C_j} = \mathsf{Round}_{p_1\to p}(\mathbf{B}_{C_j}[\star, 1])$ and by Claim 7.1 $|\mathsf{error}_j| < \mathsf{Bnd}$.

Therefore, H_1 replace Step 5 in the real world by the following equivalent way of generating sk_C:

5′. For every j, compute $b'_{C_j} = \mathsf{PE.RoundEval}(\mathsf{PK}, C_j, \mathsf{ct}, p)$, then figure out the noise error_j in b'_{C_j} using the master secret key $\boldsymbol{s}$, and set

$$\mathsf{sk}_{C_j} = b'_{C_j} - y_j \lfloor p/2 \rfloor + \mathsf{error}'_j\ ,\ \text{where } \mathsf{error}'_j = (e_j - \mathsf{error}_j),\ e_j \leftarrow [-\mathsf{Bnd}', \mathsf{Bnd}'] \cap \mathbb{Z}\ .$$

$\underline{\mathsf{Ideal}^{\epsilon\text{-1LGFE}}_{\mathcal{A},\mathcal{S}}(1^\lambda)}$:

1. *Challenge:* $(m,\ C) \leftarrow \mathcal{A}_1(1^\lambda, \mathsf{prmtr} = (n, d, \ell, \mathsf{size}))$ (where n, d, ℓ, and size are polynomials, $m \in \{0,1\}^n$, and $C \in \mathcal{F}^{\mathsf{CIRC}}_{\lambda,n,\ell,d,\mathsf{size}}$). Below $n = n(\lambda)$, $\ell = \ell(\lambda)$, $d = d(\lambda)$, $\mathsf{size} = \mathsf{size}(\lambda)$, and $\mathsf{prmtr} = (n, \ell, d, \mathsf{size})$.

2. *Generate* pp: $\mathsf{pp} \leftarrow \epsilon\text{-1LGFE.PPGen}(1^\lambda, \mathsf{prmtr})$, where $\mathsf{pp} = (p, \mathsf{prmtr}, \mathsf{PK})$ and $\mathsf{PK} = ((\mathbf{B}_1, ..., \mathbf{B}_{n'}), p_1)$.

3. *Sampling Bernouli Variables:* Sample Bernouli random variable v as follows. For each $j \in [\ell]$, sample $v_j \leftarrow \mathrm{Bernouli}(\sigma)$ for $\sigma = \frac{2(\mathsf{Bnd}'-\mathsf{Bnd})+1}{2\mathsf{Bnd}'+1}$; v_j indicates whether the good event good_j occurs:

   $$\text{Event } \mathsf{good}_j \ :\ e_j - \mathsf{error}_j \in \mathsf{GoodRng},\ \text{where } \mathsf{GoodRng} = [-\mathsf{Bnd}' + \mathsf{Bnd}, \mathsf{Bnd}' - \mathsf{Bnd}]\ .$$

   Then set $v := \min_{j\in[\ell]}(v_j)$, indicating whether good_j occurs for all j.

   By Claim 7.2, the probability σ of good_j occurring is at least $\sigma \geq 1 - \frac{(1-\epsilon)}{\ell}$. Thus the probability that v is 1 is at least $\Pr[v = 1] \geq 1 - \ell(1-\sigma) \geq \epsilon$.

   Next, the simulator $\mathcal{S} = (\widetilde{\mathsf{Enc}}, \widetilde{\mathsf{KeyGen}})$ simulates the ciphertext and secret keys: If $v = 1$, $\mathcal{S}$ receives from the oracle $w = (C, C(m))$, and otherwise $w = (C, m)$. Sample $\boldsymbol{s} \leftarrow \chi^{\dim_1 \times 1}$. Let $\mathsf{td} = \boldsymbol{s}$.

4. *Simulate the challenge ciphertext* $\widetilde{\mathsf{Enc}}(\mathsf{td}, w)$*:* If) v = 1, simulate the ciphertext $\mathsf{ct} \leftarrow \mathsf{PE.Enc}_1(\mathsf{PK}, \boldsymbol{s}, 0)$, and If) v = 0, generate the ciphertext of m honestly $\mathsf{ct} \leftarrow \mathsf{PE.Enc}_1(\mathsf{PK}, \boldsymbol{s}, m)$.

5. *Simulate the challenge secret key* $\widetilde{\mathsf{KeyGen}}(\mathsf{td}, w, C)$*:* $\mathsf{sk}_C = \{\mathsf{sk}_{C_j}\}$, where for every $j \in [\ell]$,

   $$b'_{C_j} = \mathsf{PE.RoundEval}(\mathsf{PK}, C_j, \mathsf{ct}, p)\ , \quad \mathsf{sk}_{C_j} = b'_{C_j} - y_j\lceil p/2 \rceil + \mathsf{error}'_j\ ,$$

   and the noise error'_j is sampled as below:

   - If $v_j = 1$, $\mathsf{error}'_j \leftarrow \mathsf{GoodRng}$;
   - if $v_j = 0$ (which implies $v = 0$), first figure out the error in b'_{C_j},

     $$b'_{C_j} = \langle \boldsymbol{d}_{C_j}, \boldsymbol{s} \rangle + y_j\lceil p/2 \rceil + \mathsf{error}_j, \quad |\mathsf{error}_j| < \mathsf{Bnd}\ ,$$

     set $\mathsf{error}'_j = e_j - \mathsf{error}_j$, by rejection sampling $e_j \leftarrow [-\mathsf{Bnd}', \mathsf{Bnd}'] \cap \mathbb{Z}$ conditioned on $\mathsf{error}'_j \notin \mathsf{GoodRng}$.

Output $\beta \leftarrow \mathcal{A}_2(\{\mathsf{pp}, \mathsf{ct}, \mathsf{sk}_C\})$.

Figure 3: The ideal-world ϵ-simulation security game of ϵ-1LGFE, and the definition of simulator $\mathcal{S}$. The red steps are steps that are different from the real execution.

By construction of ϵ-1LGFE, H_1 and the real world generate identically distributed secret keys. Therefore the probabilities that $\mathcal{A}$ outputs 1 are identical in these two worlds.

Next, we observe two properties of the noise $\mathsf{error}'_j = e_j - \mathsf{error}_j$ that will be instrumental later. Denote by good_j the event that $\mathsf{error}'_j \in \mathsf{GoodRng} = [-\mathsf{Bnd}' + \mathsf{Bnd}, \mathsf{Bnd}' - \mathsf{Bnd}]$. We show that the probability that good_j occurs is independent of the value of error_j, and the distribution of error_j conditioned on good_j is uniform over GoodRng.

Claim 7.2. *Let* $\mathsf{Bnd}' > \mathsf{Bnd}$ *be positive integers and* $\mathsf{GoodRng} = [-\mathsf{Bnd}' + \mathsf{Bnd}, \mathsf{Bnd}' - \mathsf{Bnd}]$. *For every* $\mathsf{error} \in [-\mathsf{Bnd}, \mathsf{Bnd}]$,

$$\Pr[\mathsf{error}' = e - \mathsf{error} \in \mathsf{GoodRng}] = \frac{2(\mathsf{Bnd}' - \mathsf{Bnd}) + 1}{2\mathsf{Bnd}' + 1} = \sigma \geq 1 - \frac{1-\epsilon}{\ell}$$

where $e \leftarrow [-\mathsf{Bnd}', \mathsf{Bnd}'] \cap \mathbb{Z}$, *and the distribution of* error' *conditioned on* $\mathsf{error}' \in \mathsf{GoodRng}$ *is uniform over* GoodRng.

Proof. The probability of $\mathsf{error}' \in \mathsf{GoodRng}$ is the same as the probability of $e \in [-\mathsf{Bnd}' + \mathsf{Bnd} + \mathsf{error}, \mathsf{Bnd}' - \mathsf{Bnd} - \mathsf{error}] \subseteq [-\mathsf{Bnd}', \mathsf{Bnd}'] \cap \mathbb{Z}$. Since e is uniformly distributed in $[-\mathsf{Bnd}', \mathsf{Bnd}'] \cap \mathbb{Z}$, this probability is exactly σ.

Furthermore, for any value $y \in \mathsf{GoodRng}$, $\Pr[\mathsf{error}' = y | \mathsf{error}' \in \mathsf{GoodRng}]$ is the same as $\Pr[e = y + \mathsf{error} | e \in [-\mathsf{Bnd}' + \mathsf{Bnd} + \mathsf{error}, \mathsf{Bnd}' - \mathsf{Bnd} - \mathsf{error}]] = 1/(2(\mathsf{Bnd}' - \mathsf{Bnd}) + 1)$. Therefore, the conditional distribution is uniform.

Finally, the lower bound on the probability follows immediately from the way that Bnd' is set in the construction; see Figure 7.3. □

Second, observe that given error_j, we can efficiently sample error'_j conditioned on good_j not occurring using rejection sampling: Keep sampling $e_j \leftarrow [-\mathsf{Bnd}', \mathsf{Bnd}'] \cap \mathbb{Z}$ till $\mathsf{error}'_j = e_j - \mathsf{error}_j \notin \mathsf{GoodRng}$. This rejection sampling is efficient since the probability of good_i not occurring is $\frac{2\mathsf{Bnd}}{2\mathsf{Bnd}'+1}$ which is polynomial. Given these properties, we are now ready to present the next hybrid.

Hybrid H_2: This hybrid is identical to H_1, except that when generating sk_C, the noise error'_j is sampled in two cases:

5″. For every j, compute $b'_{C_j} = \mathsf{PE.RoundEval}(\mathsf{PK}, C_j, \mathsf{ct}, p)$, and set

$$\mathsf{sk}_{C_j} = b'_{C_j} - y_j \lfloor p/2 \rfloor + \mathsf{error}'_j \,.$$

Here, error'_j is sampled as follows: Sample indicator random variable $v_j \leftarrow \mathrm{Bernouli}(\sigma)$.

- If $v_j = 1$, $\mathsf{error}'_j \leftarrow \mathsf{GoodRng}$;
- if $v_j = 0$, figure out the noise error_j in b'_{C_j} using the master secret key $\boldsymbol{s}$, and set $\mathsf{error}'_j = e_j - \mathsf{error}_j$, by rejection sampling $e_j \leftarrow [-\mathsf{Bnd}', \mathsf{Bnd}'] \cap \mathbb{Z}$ conditioned on $\mathsf{error}'_j \notin \mathsf{GoodRng}$.

It follows from our observations above that the H_1 and H_2 generate the same dis-

tribution of errors error_j's and hence the probability that $\mathcal{A}$ outputs 1 is identical in these two hybrids. Furthermore, also note that H_2 is efficient since rejection sampling is efficient.

The simulation world: The simulation world Ideal described in Figure 7.4 is identical to H_2 except that *i)* it samples all indicator variables v_j in Step 3, and set $v = \min_j(v_j)$ to be the indicator variable for the event that all v_j's are 1. *ii)* the main difference is that when $v = 1$, we switch the challenge ciphertext ct from encrypting m to encrypting 0. The first change does not change the distribution of the simulated secret key and ciphertext. The second change produces an indistinguishable ciphertext since when $v = 1$, every $v_j = 1$ and error'_j is sampled uniformly from $\mathsf{GoodRng}$ without using the master secret key $\boldsymbol{s}$. Therefore, changing the encrypted value is indistinguishable.

Finally, we observe that the simulator is well-formed: Whenever $v = 1$, it simulates using only $y = C(m)$, and when $v = 0$, it simulates using C, m. Furthermore, the probability that $v = 1$ is at least $1 - \ell(1 - \sigma) \geq \epsilon$ as required.

□

Remark 7.3 (On Subexponential Security)**.** Above, we proved polynomial security of ϵ-1LGFE, that is the real and ideal worlds are indistinguishable to polynomial-sized adversaries with negligible distinguishing advantage. If relying on the subexponential indistinguishability of LWE, we can prove that real and ideal worlds are subexponentially indistinguishable. This is because the views of $\mathcal{A}$ in the real world, hybrid H_1, and H_2 are identically distributed; and the views in H_2 and ideal worlds are subexponentially indistinguishable by subexponential LWE.

8 Our $(\mathsf{arith}\text{-}\mathsf{NC}^1, \deg 2)$-PHFE from Pairings

In Fig.8.2 we present a Partially-Hiding FE (PHFE) for the functionality $\mathcal{F}^{\mathsf{PHFE}} = \mathbb{U}_{n,\ell,w\in\mathsf{poly}}\{\mathcal{F}^{\mathsf{PHFE}}_{\lambda,n(\lambda),\ell(\lambda),w(\lambda}$ where for all $\lambda \in \mathbb{N}$, $\mathcal{F}^{\mathsf{PHFE}}_{\lambda,n(\lambda),\ell(\lambda),w(\lambda)}$ denotes the set of all functions defined relative a sequence of pairing group $\mathcal{G}_\lambda$ of the form $\mathcal{G}_\lambda = (\mathbf{G}_1, \mathbf{G}_2, \mathbf{G}_T, p, P_1, P_2, e)$ (see the definition of pairing group in Section 3) as follows. Each function in $\mathcal{F}^{\mathsf{PHFE}}_{\lambda,\mathsf{prmtr}}$ is represented by a tuple $(f^0, \ldots, f^{\ell+1})$ such that for all inputs $(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}) \in (\mathbb{Z}_p^n)^3$, it outputs $\left[f^0 \prod_{i=1}^{\ell} f^i(\boldsymbol{x}) f^{\ell+1}(\boldsymbol{y} \otimes \boldsymbol{z})\right]_T \in \mathbf{G}_T$, where $f^0 \in \mathbb{Z}_p^{1\times w}$, $f^i(\boldsymbol{x}) \in \mathbb{Z}_p^{w\times w}$ for all $i \in [d]$, $f^{\ell+1}(\boldsymbol{y} \otimes \boldsymbol{z}) \in \mathbb{Z}_p^w$, and all functions f^i for $i > 0$ are linear maps. The computation is performed in the exponent of a generator of the cyclic group $\mathbf{G}_T$, of order p. This model of computation captures functions f of the form: $f(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}) = w(g(\boldsymbol{x}), \boldsymbol{y} \otimes \boldsymbol{z})$, where w is a multilinear degree two polynomial (with degree one in $\boldsymbol{y} \otimes \boldsymbol{z}$) and g is a matrix branching program of width w and length ℓ over $\mathbb{Z}_p$. By Barrington's theorem, for sufficiently large $\ell, w, \log(p) = \mathsf{poly}(\lambda)$, it also contains the case when g is a Boolean NC^1 circuit ($\boldsymbol{x}$ being restricted to be a binary vector in this case). Note that to realize Boolean NC^1 circuits, we need each function f^i to be affine, which can be ensured by setting, say, $x_1 = 1$.

We give a modular construction of PHFE for the functionality $\mathcal{F}^{\mathsf{PHFE}}_{\mathsf{prmtr}}$ in Section 8.2 that builds upon inner-product FE, defined in Section 8.1. Our construction is linearly

efficient as per Definition 4.6. That is, the ciphertext is computed by a circuit of size at most$n \cdot \mathsf{poly}(\lambda)$ for a fixed polynomial, where λ denotes the security parameter and n is dimension of the vectors being encrypted. As such, our PHFE can be used to build general purpose FE in Section 9. Finally, we build the concrete inner-product FE scheme that underlies our PHFE in Section 8.3. The security of all of our constructions rely on standard assumptions in pairing groups.

8.1 Ingredients: Inner-Product FE

We define the functionality $\mathcal{F}^{\mathsf{IPFE}} = \bigcup_{\dim\in\mathsf{poly}}\{\mathcal{F}^{\mathsf{IPFE}}_{\dim,\lambda}\}_\lambda$ where for all dimensions $\dim \in \mathsf{poly}(\lambda)$, all $\lambda \in \mathbb{N}$, $\mathcal{F}^{\mathsf{IPFE}}_{\dim,\lambda}$ is the set of function $f : \mathbf{G}_1^{\dim} \to \mathbf{G}_T$, described by a vector $[\boldsymbol{y}]_2 \in \mathbf{G}_2^{\dim}$, such that given as input $[\boldsymbol{x}]_1 \in \mathbf{G}_1^{\dim}$, outputs $[\boldsymbol{x}^\top\boldsymbol{y}]_T \in \mathbf{G}_T$. We define the functionality $\mathcal{F}^{\mathsf{ipfe}'}$ similarly except the inputs $\boldsymbol{x}$ are in $\mathbb{Z}^{\dim}$ instead of $\mathbf{G}_1^{\dim}$. To build an FE for $\mathcal{F}^{\mathsf{PHFE}}$, we rely on a private-key IND-function-hiding FE $\widehat{\mathsf{IPFE}}$ for the functionality $\mathcal{F}^{\mathsf{IPFE}}$ and an FE $\overline{\mathsf{IPFE}}$ for the functionality $\mathcal{F}^{\mathsf{IPFE}'}$. We only require that the scheme $\overline{\mathsf{IPFE}}$ satisfies a simulation security that is slightly weaker than defined Definition 4.3, in the sense that the simulator generates the functional secret keys for a function $[\boldsymbol{y}]_2$ only knowing the output $[\boldsymbol{x}^\top\boldsymbol{y}]_2$ in $\mathbf{G}_2$ or $[\boldsymbol{x}^\top\boldsymbol{y}]_1$ in $\mathbf{G}_1$, as opposed to $\mathbf{G}_T$, where $\boldsymbol{x}$ denotes the challenge (see Definition 8.1).

Definition 8.1 (Weak simulation security)**.** *Let* FE *be an FE scheme for the functionality* $\mathcal{F}^{\mathsf{IPFE}'}$ *defined above. We say that* FE *is weakly simulation secure if there exists a PPT simulator* $\mathcal{S} := (\widetilde{\mathsf{Setup}}, \widetilde{\mathsf{Enc}}, \widetilde{\mathsf{KeyGen}}_1, \widetilde{\mathsf{KeyGen}}_2)$ *such that for all* $\dim \in \mathsf{poly}$, $\lambda \in \mathbb{N}$, *all* $\mathsf{prmtr} = (w, \ell, n)$, *all* pp *in the support of* $\mathsf{PPGen}(1^\lambda, \mathsf{prmtr})$, *all* $(\widetilde{\mathsf{pk}}, \widetilde{\mathsf{msk}})$ *in the support of* $\widetilde{\mathsf{Setup}}(1^\lambda)$, *all* $\boldsymbol{y} \in \mathbb{Z}_p^{\dim}$, $v \in \mathbb{Z}_p$, *the following are identically distributed:*

$$\widetilde{\mathsf{KeyGen}}_1(\widetilde{\mathsf{msk}}, [\boldsymbol{y}]_1, [v]_1) \;\; \textit{and} \;\; \widetilde{\mathsf{KeyGen}}_2(\widetilde{\mathsf{msk}}, [\boldsymbol{y}]_2, [v]_2).$$

Moreover, for all stateful PPT adversaries $\mathcal{A}$, *there exists a negligible function* negl *such that for all* $\lambda \in \mathbb{N}$, *all* $\mathsf{prmtr} = (n, \ell, w)$, *we have:*

$$\mathsf{adv}^{\mathsf{weak\text{-}SIM}}_{\mathsf{FE},\mathcal{A}}(\lambda, \mathsf{prmtr}) := |\Pr[1 \leftarrow \mathsf{Real}^{\mathsf{FE}}_{\mathcal{A}}(1^\lambda, \mathsf{prmtr})] - \Pr[1 \leftarrow \mathsf{Ideal}^{\mathsf{FE}}_{\mathcal{A},\mathcal{S}}(1^\lambda, \mathsf{prmtr})]| = \mathsf{negl}(\lambda),$$

where the experiments are defined below.

$\mathsf{Real}^{\mathsf{FE}}_{\mathcal{A}}(1^\lambda, \mathsf{prmtr})$*:*
$[\boldsymbol{x}]_1 \leftarrow \mathcal{A}(1^\lambda)$
$\mathsf{pp} \leftarrow \mathsf{PPGen}(1^\lambda, \mathsf{prmtr})$
$(\mathsf{pk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(\mathsf{pp})$
$\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pk}, [\boldsymbol{x}]_1)$
$\alpha \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{KeyGen}}(\cdot)}(\mathsf{ct}, \mathsf{pk})$

$\mathsf{Ideal}^{\mathsf{FE}}_{\mathcal{A},\mathcal{S}}(1^\lambda, \mathsf{prmtr})$*:*
$[\boldsymbol{x}]_1 \leftarrow \mathcal{A}(1^\lambda)$
$\mathsf{pp} \leftarrow \mathsf{PPGen}(1^\lambda, \mathsf{prmtr})$
$(\widetilde{\mathsf{pk}}, \widetilde{\mathsf{msk}}) \leftarrow \widetilde{\mathsf{Setup}}(\mathsf{pp})$
$\mathsf{ct} \leftarrow \widetilde{\mathsf{Enc}}(\widetilde{\mathsf{msk}})$
$\alpha \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{KeyGen}}(\cdot)}(\mathsf{ct}, \widetilde{\mathsf{pk}})$

In the real experiment, the key generation oracle $\mathcal{O}_{\mathsf{KeyGen}}$, *when given as input* $[\boldsymbol{y}]_2 \in \mathbf{G}_2^{\dim}$, *returns* $\mathsf{KeyGen}(\mathsf{msk}, [\boldsymbol{y}]_2)$. *In the ideal experiment, the key generation oracle* $\mathcal{O}_{\mathsf{KeyGen}}$, *when given as input* $[\boldsymbol{y}]_2 \in \mathbf{G}_2^{\dim}$, *computes* $[\boldsymbol{x}^\top\boldsymbol{y}]_2$, *and returns* $\widetilde{\mathsf{KeyGen}}_2(\widetilde{\mathsf{msk}}, [\boldsymbol{y}]_2, [\boldsymbol{x}^\top\boldsymbol{y}]_2)$. *Note that this differs from Definition 4.3, where the algorithm* $\widetilde{\mathsf{KeyGen}}$ *gets as input* $[\boldsymbol{x}^\top\boldsymbol{y}]_T \in \mathbf{G}_T$, *not in* $\mathbf{G}_2$.

8.2 Modular Construction of the Partially-Hiding FE

In Fig.8.2 we present a modular construction of PHFE for the functionality $\mathcal{F}^{\mathsf{PHFE}}$, which relies on an IND-function-hiding FE for the functionality $\mathcal{F}^{\mathsf{IPFE}}$ and weakly simulation-secure FE for the functionality $\mathcal{F}^{\mathsf{IPFE}'}$. The simulation security of our PHFE relies on the security of the underlying building blocks and the SXDH assumption in $\mathcal{G}$.

Linear efficiency:

The ciphertexts ct_i and ct'_j for all $i, j \in [n]$ are encryptions and functional keys associated with vectors of *constant* dimension (in particular, independent of n). Moreover, the order of the group is also independent of n. Thus, by the sheer fact that the encryption and functional key algorithm are polynomial-time, there exists a polynomial p_1 such that the circuit that outputs all the ciphertexts ct_i and ct'_j is of size $n \cdot p_1(\lambda)$. By linear efficiency of $\overline{\mathsf{IPFE}}$, there exists a polynomial p_2 such that the circuit that outputs $\overline{\mathsf{ct}}$ is of size at most $n \cdot p_2(\lambda)$. Overall, the size of the encryption algorithm Enc is $n \cdot (p_1(\lambda) + p_2(\lambda))$, that is, the scheme has linear efficiency.

Correctness:

By correctness of $\widehat{\mathsf{IPFE}}$, for all $i, j \in [n]$, we have:

$$[\theta_{i,j}]_T = [y_i z_j + rs\boldsymbol{a}_i^\top \boldsymbol{b}_j]_T \text{ and } [\theta]_T = f(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}) + rsf(\boldsymbol{x}, \boldsymbol{a}, \boldsymbol{b}),$$

where $f(\boldsymbol{x}, \boldsymbol{a}, \boldsymbol{b}) = f^0 \prod_{i\in[\ell]} f^i(\boldsymbol{x}) f^{\ell+1}(\boldsymbol{a} \otimes \boldsymbol{b})$, with $\boldsymbol{a} \otimes \boldsymbol{b} = (\boldsymbol{a}_i^\top \boldsymbol{b}_j)_{i,j\in[n]} \in \mathbb{Z}^{n^2}$.

By correctness of $\overline{\mathsf{IPFE}}$, for all $t \in [\ell]$, we have:

$$[\boldsymbol{w}_t]_T = \left[rs\big(f^t(\boldsymbol{u}_t) - f^t(\boldsymbol{x})\big) \prod_{t<i\leq\ell} f^i(\boldsymbol{u}_i) f^{\ell+1}(\boldsymbol{a} \otimes \boldsymbol{b}) \right]_T .$$

Besides, we have:

$$[\theta_{\ell+1}]_T = \left[rsf^0 \prod_{i\in[\ell]} f^i(\boldsymbol{u}_i) f^{\ell+1}(\boldsymbol{a} \otimes \boldsymbol{b}) \right]_T .$$

Thus, the telescoping sum is of the form:

$$\left[\sum_{t\in[\ell]} f^0 \prod_{0<i<t} f^i(\boldsymbol{x}) \boldsymbol{w}_t \right]_T = [\theta_{\ell+1} - rsf(\boldsymbol{x}, \boldsymbol{a}, \boldsymbol{b})]_T .$$

Consequently, we have:

$$[\theta]_T + \left[\sum_{t\in[\ell]} f^0 \prod_{0<i<t} f^i(\boldsymbol{x}) \boldsymbol{w}_t \right]_T - [\theta_{\ell+1}]_T = [f(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z})]_T .$$

$\underline{\mathsf{PPGen}(1^\lambda, \mathsf{prmtr} = (n, w, \ell)):}$

Compute $\widehat{\mathsf{pp}} \leftarrow \widehat{\mathsf{PPGen}}(1^\lambda, \dim = 3)$, $\overline{\mathsf{pp}} \leftarrow \overline{\mathsf{PPGen}}(1^\lambda, \dim = n+1)$. These parameters are both defined relative to the same sequence of groups $\{\mathcal{G}_\lambda = (\mathbf{G}_1, \mathbf{G}_2, \mathbf{G}_T, p, P_1, P_2, e)\}_\lambda$. Set $\mathsf{pp} = (\widehat{\mathsf{pp}}, \overline{\mathsf{pp}})$.

$\underline{\mathsf{Setup}(\mathsf{pp}):}$

Compute $(\overline{\mathsf{pk}}, \overline{\mathsf{msk}}) \leftarrow \overline{\mathsf{Setup}}(1^\lambda, \dim = n+1)$. For all $i, j \in [n]$: $\boldsymbol{a}_i, \boldsymbol{b}_j \leftarrow_{\mathrm{R}} \mathsf{DDH}$, for all $k \in [\ell]$, $\boldsymbol{u}_k \leftarrow_{\mathrm{R}} \mathbb{Z}_p^n$. Return $\mathsf{pk} := \left(\overline{\mathsf{pk}}, \{[\boldsymbol{a}_i]_1, [\boldsymbol{b}_j]_2\}_{i,j\in[n]}\right)$ and $\mathsf{msk} := \left(\overline{\mathsf{msk}}, \{\boldsymbol{a}_i, \boldsymbol{b}_j, \boldsymbol{u}_k\}_{i,j\in[n],k\in[\ell]}\right)$.

$\underline{\mathsf{Enc}(\mathsf{pk}, \boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z} \in (\mathbb{Z}_p)^3):}$

$r, s \leftarrow_{\mathrm{R}} \mathbb{Z}_p$, $(\widehat{\mathsf{pk}}, \widehat{\mathsf{msk}}) \leftarrow \widehat{\mathsf{Setup}}(1^\lambda, \dim = 3)$, $\overline{\mathsf{ct}} \leftarrow \overline{\mathsf{Enc}}\left(\overline{\mathsf{pk}}, \begin{pmatrix} rs\boldsymbol{x} \\ rs \end{pmatrix}\right)$. For all $i, j \in [n]$: $\mathsf{ct}_i \leftarrow \widehat{\mathsf{Enc}}\left(\widehat{\mathsf{msk}}, \begin{bmatrix} y_i \\ \boldsymbol{a}_i r \end{bmatrix}_1\right)$, $\mathsf{ct}'_j \leftarrow \widehat{\mathsf{KeyGen}}\left(\widehat{\mathsf{msk}}, \begin{bmatrix} z_j \\ \boldsymbol{b}_j s \end{bmatrix}_2\right)$. Return $\left(\overline{\mathsf{ct}}, \{\mathsf{ct}_i, \mathsf{ct}'_j\}_{i,j\in[n]}\right)$.

$\underline{\mathsf{KeyGen}\big(\mathsf{msk}, (f^0, \ldots, f^{\ell+1})\big):}$

For all $t \in [\ell]$, we write $[\mathbf{M}_t]_2 \in \mathbf{G}_2^{(n+1)\times w}$, the linear function such that for all $\begin{bmatrix} \boldsymbol{v} \\ \alpha \end{bmatrix}_1 \in \mathbf{G}_1^{n+1}$, $\left[\mathbf{M}_t^\top \begin{pmatrix} \boldsymbol{v} \\ \alpha \end{pmatrix}\right]_T = \left[\left(\alpha \cdot f^t(\boldsymbol{u}_t) - f^t(\boldsymbol{v})\right) \prod_{t<i\le\ell} f^i(\boldsymbol{u}_i) f^{\ell+1}(\boldsymbol{a}\otimes\boldsymbol{b})\right]_T \in \mathbf{G}_T^w$, and $[\boldsymbol{m}_{\ell+1}]_2 \in \mathbf{G}_2^{n+1}$ the linear function such that for all $\begin{bmatrix} \boldsymbol{v} \\ \alpha \end{bmatrix}_1 \in \mathbf{G}_1^{n+1}$, $\left[\boldsymbol{m}_{\ell+1}^\top \begin{pmatrix} \boldsymbol{v} \\ \alpha \end{pmatrix}\right]_T = \left[\alpha \cdot f^0 \prod_{i\in[\ell]} f^i(\boldsymbol{u}_i) f^{\ell+1}(\boldsymbol{a}\otimes\boldsymbol{b})\right]_T \in \mathbf{G}_T$, where $\boldsymbol{a}\otimes\boldsymbol{b} = (\boldsymbol{a}_i^\top \boldsymbol{b}_j)_{i,j\in[n]} \in \mathbb{Z}^{n^2}$. For all $t \in [\ell]$, $\mathsf{sk}_t \leftarrow \overline{\mathsf{KeyGen}}(\overline{\mathsf{msk}}, [\mathbf{M}_t]_2)$, and $\mathsf{sk}_{\ell+1} \leftarrow \overline{\mathsf{KeyGen}}(\overline{\mathsf{msk}}, [\boldsymbol{m}_{\ell+1}]_2)$. Return $\{\mathsf{sk}_t\}_{t\in[\ell+1]}$.

$\underline{\mathsf{Dec}(\mathsf{ct}, \mathsf{sk}):}$

Parse $\mathsf{ct} = \left(\overline{\mathsf{ct}}, \{\mathsf{ct}_i, \mathsf{ct}'_j\}_{i,j\in[n]}\right)$ and $\mathsf{sk} = \{\mathsf{sk}_t\}_{t\in[\ell+1]}$. For all $i, j \in [n]$: $[\theta_{i,j}]_T \leftarrow \widehat{\mathsf{Dec}}(\mathsf{ct}_i, \mathsf{ct}'_j) \in \mathbf{G}_T$. $[\theta]_T = \left[f^0 \prod_{i\in[\ell]} f^i(\boldsymbol{x}) f^{\ell+1}(\theta_{i,j})_{i,j\in[n]}\right]_T \in \mathbf{G}_T$. For all $t \in [\ell]$, $[\boldsymbol{w}_t]_T \leftarrow \overline{\mathsf{Dec}}(\overline{\mathsf{ct}}, \mathsf{sk}_t) \in \mathbf{G}_T^w$, $[\theta_{\ell+1}]_T \leftarrow \overline{\mathsf{Dec}}(\overline{\mathsf{ct}}, \mathsf{sk}_{\ell+1}) \in \mathbf{G}_T$. Return $[\theta]_T + \left[\sum_{t\in[\ell]} f^0\big(\prod_{0<m<t} f^m(\boldsymbol{x})\big)\boldsymbol{w}_t\right]_T - [\theta_{\ell+1}]_T$.

Figure 4: This is PHFE, a simulation-secure FE scheme for the functionality $\mathcal{F}^{\mathsf{phfe}}$. Here, $\widehat{\mathsf{IPFE}} := (\widehat{\mathsf{Setup}}, \widehat{\mathsf{Enc}}, \widehat{\mathsf{KeyGen}}, \widehat{\mathsf{Dec}})$ is an IND-function-hiding FE for the functionality $\mathcal{F}^{\mathsf{IPFE}}$, and $\overline{\mathsf{IPFE}} := (\overline{\mathsf{Setup}}, \overline{\mathsf{Enc}}, \overline{\mathsf{KeyGen}}, \overline{\mathsf{Dec}})$ is a weakly simulation-secure FE for the functionality $\mathcal{F}^{\mathsf{IPFE}'}$.

Theorem 8.1 (Simulation security)**.** *The scheme presented in Fig.8.2 is simulation secure (as per Definition 4.3), provided the underlying* $\widehat{\mathsf{IPFE}}$ *is indistinguishability function-hiding secure (as defined in Definition 4.4), and* $\overline{\mathsf{IPFE}}$ *is simulation secure as per Definition 8.1, which is implied by the notion given in Definition 4.3. Namely, for any PPT adversary* $\mathcal{A}$*, there exist PPT adversaries* $\mathcal{B}_1$*,* $\mathcal{B}_2$*,* $\mathcal{B}_3$ *and* $\mathcal{B}_4$ *such that:*

$$\mathsf{adv}^{\mathsf{SIM}}_{\mathsf{PHFE},\mathcal{A}}(\lambda) \leq \mathsf{adv}^{\mathsf{weak\text{-}SIM}}_{\overline{\mathsf{IPFE}},\mathcal{B}_1}(\lambda) + (\ell+1)\cdot \mathsf{adv}^{\mathsf{DDH}}_{\mathbf{G}_2,\mathcal{B}_2}(\lambda) + 3\cdot \mathsf{adv}^{\mathsf{DDH}}_{\mathbf{G}_1,\mathcal{B}_3}(\lambda) + \mathsf{adv}^{\mathsf{IND\text{-}FH}}_{\widehat{\mathsf{IPFE}},\mathcal{B}_4}(\lambda) + \frac{2}{p}.$$

Proof. The proof proceeds using a series of hybrid games, described below. Let $\mathcal{A}$ be a PPT adversary against the simulation security of the scheme. For any game $\mathbf{Hybrid}_i$, we denote by $\mathsf{adv}_i := \Pr[1 \leftarrow \mathbf{Hybrid}_i(\mathcal{A})]$ the probability that $\mathbf{Hybrid}_i$ returns 1 when interacting with $\mathcal{A}$.

• $\underline{\mathbf{Hybrid}_0}$: is the real experiment as given in Definition 4.3.

• $\underline{\mathbf{Hybrid}_1}$: is the same as $\mathbf{Hybrid}_0$, except we replace the scheme $(\overline{\mathsf{Setup}}, \overline{\mathsf{Enc}}, \overline{\mathsf{KeyGen}})$ by its simulator $(\widetilde{\mathsf{Setup}}, \widetilde{\mathsf{Enc}}, \widetilde{\mathsf{KeyGen}}_2)$. That is, we sample $(\widetilde{\mathsf{pk}}, \widetilde{\mathsf{msk}}) \leftarrow \widetilde{\mathsf{Setup}}(\overline{\mathsf{pp}})$, instead of $(\overline{\mathsf{pk}}, \overline{\mathsf{msk}}) \leftarrow \overline{\mathsf{Setup}}(\overline{\mathsf{pp}})$.

The challenge ciphertext is generated using $\overline{\mathsf{ct}} \leftarrow \widetilde{\mathsf{Enc}}\left(\widetilde{\mathsf{msk}}\right)$ instead of $\overline{\mathsf{ct}} \leftarrow \overline{\mathsf{Enc}}\left(\overline{\mathsf{pk}}, \begin{pmatrix} rs\boldsymbol{x} \\ rs \end{pmatrix}\right)$.

The functional secret keys are generated using, for all $t \in [\ell]$:

$$\mathsf{sk}_t \leftarrow \widetilde{\mathsf{KeyGen}}_2\left(\widetilde{\mathsf{msk}}, [\mathbf{M}_t]_2, \left[rs\big(f^t(\boldsymbol{u}_t) - f^t(\boldsymbol{x})\big)\prod_{t<i\leq\ell} f^i(\boldsymbol{u}_i) f^{\ell+1}(\boldsymbol{a}\otimes\boldsymbol{b}),\right]_2\right)$$

and

$$\mathsf{sk}_{\ell+1} \leftarrow \widetilde{\mathsf{KeyGen}}_2\left(\widetilde{\mathsf{msk}}, [\boldsymbol{m}_{\ell+1}]_2, \left[rsf^0\prod_{i\in[\ell]} f^i(\boldsymbol{u}_i) f^{\ell+1}(\boldsymbol{a}\otimes\boldsymbol{b})\right]_2\right),$$

where $\boldsymbol{a}\otimes\boldsymbol{b} = (\boldsymbol{a}_i^\top \boldsymbol{b}_j)_{i,j\in[n]} \in \mathbb{Z}^{n^2}$.

This transition is justified by the simulation security of $\overline{\mathsf{IPFE}}$. Namely, there is a PPT adversary $\mathcal{B}_0$ such that:

$$|\mathsf{adv}_0 - \mathsf{adv}_1| \leq \mathsf{adv}^{\mathsf{weak\text{-}SIM}}_{\overline{\mathsf{IPFE}},\mathcal{B}_0}(\lambda).$$

• $\underline{\mathbf{Hybrid}_2}$: is the same as $\mathbf{Hybrid}_1$, except we replace the vectors $\{\boldsymbol{u}_k\}_{k\in[\ell]}$ by $\{\boldsymbol{u}_k + \boldsymbol{x}\}_{k\in[\ell]}$. These values are identically distributed, since the vectors $\boldsymbol{u}_k$ are sampled uniformly over $\mathbb{Z}_p^n$, independently of the challenge $\boldsymbol{x}$, which is chosen beforehand. Consequently, the functional secret keys are now generated using, for all $t \in [\ell]$:

$$\mathsf{sk}_t \leftarrow \widetilde{\mathsf{KeyGen}}_2\left(\widetilde{\mathsf{msk}}, [\mathbf{M}_t]_2, \left[rsf^t(\boldsymbol{u}_t)\prod_{t<i\leq\ell} f^i(\boldsymbol{u}_i+\boldsymbol{x}) f^{\ell+1}(\boldsymbol{a}\otimes\boldsymbol{b}),\right]_2\right)$$

and

$$\mathsf{sk}_{\ell+1} \leftarrow \widetilde{\mathsf{KeyGen}}_2\left(\widetilde{\mathsf{msk}}, [\boldsymbol{m}_{\ell+1}]_2, \left[rsf^0\prod_{i\in[\ell]} f^i(\boldsymbol{u}_i+\boldsymbol{x}) f^{\ell+1}(\boldsymbol{a}\otimes\boldsymbol{b})\right]_2\right).$$

Here, we use the fact that the functions f^i for all $i > 0$ are linear. We have:

$$\mathsf{adv}_1 = \mathsf{adv}_2.$$

• $\underline{\mathbf{Hybrid}_3}$: is the same as $\mathbf{Hybrid}_2$, except we replace the vectors $[s\boldsymbol{u}_k]_2$ by fresh $[\boldsymbol{s}_k]_2 \leftarrow_{\mathrm{R}} \mathbf{G}_2^n$ for all $k \in [\ell]$, using the DDH assumption in $\mathbf{G}_2$. Consequently, the functional secret keys are now generated using, for all $t \in [\ell]$:

$$\mathsf{sk}_t \leftarrow \widetilde{\mathsf{KeyGen}}_2\left(\widetilde{\mathsf{msk}}, [\mathbf{M}_t]_2, \left[rf^t(\boldsymbol{s}_t)\prod_{t<i\leq\ell} f^i(\boldsymbol{u}_i+\boldsymbol{x})f^{\ell+1}(\boldsymbol{a}\otimes\boldsymbol{b}),\right]_2\right)$$

and

$$\mathsf{sk}_{\ell+1} \leftarrow \widetilde{\mathsf{KeyGen}}_2\left(\widetilde{\mathsf{msk}}, [\boldsymbol{m}_{\ell+1}]_2, [v]_2\right),$$

where

$$[v]_2 = \left[rsf(\boldsymbol{x},\boldsymbol{a},\boldsymbol{b}) + r\sum_{i\in[\ell]}\big(\prod_{j<i} f^j(\boldsymbol{x})\big)f^i(\boldsymbol{s}_i)\big(\prod_{j>i} f^j(\boldsymbol{u}_j+\boldsymbol{x})\big)f^{\ell+1}(\boldsymbol{a}\otimes\boldsymbol{b})\right]_2.$$

We proceed via a hybrid argument, switching the vector $[s\boldsymbol{u}_k]_2$ to uniformly random $[\boldsymbol{s}_k]_2 \leftarrow_{\mathrm{R}} \mathbb{Z}_p^n$ one index $k \in [\ell]$ at a time. That is, we define $\mathbf{Hybrid}_{2.\rho}$ for all $\rho \in [0, d]$ as $\mathbf{Hybrid}_2$, except the first ρ-th functional keys are computed as in $\mathbf{Hybrid}_3$. For all $\rho \in [\ell]$, we show there exists a PPT adversary $\mathcal{B}_{2.\rho}$ such that $|\mathsf{adv}_{2.\rho-1} - \mathsf{adv}_{2.\rho}| \leq \mathsf{adv}^{\mathsf{DDH}}_{\mathbf{G}_2,\mathcal{B}_{2.\rho}}(\lambda)$.

The adversary $\mathcal{B}_{2.\rho}$ takes as input a tuple $([s]_2, [\boldsymbol{u}_\rho]_2, [\boldsymbol{s}_\rho]_2)$ where the value $[\boldsymbol{s}_\rho]_2$ is either of the form $[s\boldsymbol{u}_\rho]_2$ (case 1), or uniformly random over $\mathbf{G}_2^n$ (case 2). The adversary $\mathcal{B}_{2.\rho}$ samples $r \leftarrow_{\mathrm{R}} \mathbb{Z}_p$, $\boldsymbol{a}_i, \boldsymbol{b}_j \leftarrow_{\mathrm{R}} \mathsf{DDH}$ for all $i, j \in [n]$, $\boldsymbol{u}_m \leftarrow_{\mathrm{R}} \mathbb{Z}_p^n$ for all $m \neq \rho$, $\boldsymbol{s}_t \leftarrow_{\mathrm{R}} \mathbb{Z}_p^n$ for all $t < \rho$, $(\widetilde{\mathsf{msk}}, \widetilde{\mathsf{pk}}) \leftarrow \widetilde{\mathsf{Setup}}(\overline{\mathsf{pp}})$, upon which it can simulate the view of the adversary $\mathcal{A}$. In case 1, $\mathcal{B}_{2.\rho}$ simulates $\mathbf{Hybrid}_{2.\rho-1}$ to $\mathcal{A}$, whereas it simulates $\mathbf{Hybrid}_{2.\rho}$ in case 2.

Putting everything together, we have the existence of a PPT adversary $\mathcal{B}_2$ such that:

$$|\mathsf{adv}_2 - \mathsf{adv}_3| \leq \ell \cdot \mathsf{adv}^{\mathsf{DDH}}_{\mathbf{G}_2,\mathcal{B}_2}(\lambda).$$

• $\underline{\mathbf{Hybrid}_4}$: is the same as $\mathbf{Hybrid}_3$, except that we replace the values $[\boldsymbol{b}_j s]_2$ used for generating functional secret keys by fresh $[\boldsymbol{w}_j]_2 \leftarrow_{\mathrm{R}} \mathbf{G}_2^2$ for all $j \in [n]$, using the DDH assumption in $\mathbf{G}_2$.

Consequently, the challenge ciphertext now contains:

$$\mathsf{ct}'_j \leftarrow \widehat{\mathsf{KeyGen}}\left(\widehat{\mathsf{msk}}, \begin{bmatrix} z_j \\ -\boldsymbol{w}_j \end{bmatrix}_2\right).$$

Moreover, the functional secret keys are now generated using:

$$\mathsf{sk}_{\ell+1} \leftarrow \widetilde{\mathsf{KeyGen}}_2\left(\widetilde{\mathsf{msk}}, [\boldsymbol{m}_{\ell+1}]_2, [v]_2\right),$$

where

$$[v]_2 = \left[rf(\boldsymbol{x}, \boldsymbol{a}, \boldsymbol{w}) + r \sum_{i \in [\ell]} \Big(\prod_{j<i} f^j(\boldsymbol{x}) \Big) f^i(\boldsymbol{s}_i) \big(\prod_{j>i} f^j(\boldsymbol{u}_j + \boldsymbol{x}) \big) f^{\ell+1}(\boldsymbol{a} \otimes \boldsymbol{b}) \right]_2 .$$

We show there exists a PPT adversary $\mathcal{B}_3$ such that:

$$|\mathsf{adv}_3 - \mathsf{adv}_4| \leq \mathsf{adv}^{\mathsf{DDH}}_{\mathbf{G}_2, \mathcal{B}_3}(\lambda).$$

The adversary $\mathcal{B}_1$ takes as input a tuple $([s]_2, \{[\boldsymbol{b}_j]_2, [\boldsymbol{w}_j]_2\}_{j \in [n]})$ where the values $[\boldsymbol{w}_j]_2$ are either of the form $[\boldsymbol{b}_j s]_2$ (case 1), or uniformly random over $\mathbf{G}_2^2$ (case 2). The adversary $\mathcal{B}_3$ samples $r \leftarrow_{\mathrm{R}} \mathbb{Z}_p$, $(\widetilde{\mathsf{msk}}, \widetilde{\mathsf{pk}}) \leftarrow \widetilde{\mathsf{Setup}}(\overline{\mathsf{pp}})$, $\boldsymbol{a}_i \leftarrow_{\mathrm{R}} \mathsf{DDH}$ for all $i \in [n]$, $\boldsymbol{u}_k, \boldsymbol{s}_k \leftarrow_{\mathrm{R}} \mathbb{Z}_p^n$ for all $k \in [\ell]$, upon which it can simulate the view of the adversary $\mathcal{A}$ straightforwardly. In case 1, it simulates $\mathbf{Hybrid}_3$ to $\mathcal{A}$, whereas it simulates $\mathbf{Hybrid}_4$ in case 2.

• $\underline{\mathbf{Hybrid}_5}$: is the same as $\mathbf{Hybrid}_4$, except we use the key generation algorithm $\widetilde{\mathsf{KeyGen}}_1$, which takes inputs from $\mathbf{G}_1$ instead of $\widetilde{\mathsf{KeyGen}}_2$, which takes inputs from $\mathbf{G}_2$. Namely, the secret keys are now generated using, for all $t \in [\ell]$:

$$\mathsf{sk}_t \leftarrow \widetilde{\mathsf{KeyGen}}_1 \left(\widetilde{\mathsf{msk}}, [\mathbf{M}_t]_1 , \left[rf^t(\boldsymbol{s}_t) \prod_{t<i\leq\ell} f^i(\boldsymbol{u}_i + \boldsymbol{x}) f^{\ell+1}(\boldsymbol{a} \otimes \boldsymbol{b}), \right]_1 \right)$$

and

$$\mathsf{sk}_{\ell+1} \leftarrow \widetilde{\mathsf{KeyGen}}_1 \left(\widetilde{\mathsf{msk}}, [\boldsymbol{m}_{\ell+1}]_1, [v]_1 \right),$$

where

$$[v]_1 = \left[rf(\boldsymbol{x}, \boldsymbol{a}, \boldsymbol{w}) + r \sum_{i \in [\ell]} \Big(\prod_{j<i} f^j(\boldsymbol{x}) \Big) f^i(\boldsymbol{s}_i) \Big(\prod_{j>i} f^j(\boldsymbol{u}_j + \boldsymbol{x}) \Big) f^{\ell+1}(\boldsymbol{a} \otimes \boldsymbol{b}) \right]_1 .$$

By definition of the weak simulation security (cf Definition 8.1), the output of $\widetilde{\mathsf{KeyGen}}_1$ and $\widetilde{\mathsf{KeyGen}}_2$ are identically distributed, thus:

$$\mathsf{adv}_4 = \mathsf{adv}_5.$$

• $\mathbf{Hybrid}_6$: is the same as $\mathbf{Hybrid}_5$, except that we replace the values $[\boldsymbol{a}_i r]_1$ by fresh $[\boldsymbol{v}_i]_1 \leftarrow_{\mathrm{R}} \mathbf{G}_1^2$ for all $i \in [n]$, using the DDH assumption in $\mathbf{G}_1$. Consequently, the challenge ciphertext now contains:

$$\mathsf{ct}_i \leftarrow \widehat{\mathsf{KeyGen}} \left(\widehat{\mathsf{msk}}, \begin{bmatrix} y_i \\ \boldsymbol{v}_i \end{bmatrix}_1 \right).$$

Moreover, the secret keys are now generated using, for all $t \in [\ell]$:

$$\mathsf{sk}_t \leftarrow \widetilde{\mathsf{KeyGen}}_1\left(\widetilde{\mathsf{msk}}, [\mathbf{M}_t]_1, \left[f^t(\boldsymbol{s}_t)\prod_{t<i\leq\ell} f^i(\boldsymbol{u}_i+\boldsymbol{x})f^{\ell+1}(\boldsymbol{v}\otimes\boldsymbol{b}),\right]_1\right)$$

and

$$\mathsf{sk}_{\ell+1} \leftarrow \widetilde{\mathsf{KeyGen}}_1\left(\widetilde{\mathsf{msk}}, [\boldsymbol{m}_{\ell+1}]_1, [v]_1\right),$$

where

$$[v]_1 = \left[f(\boldsymbol{x},\boldsymbol{v},\boldsymbol{w}) + \sum_{i\in[\ell]}\Big(\prod_{j<i} f^j(\boldsymbol{x})\Big)f^i(\boldsymbol{s}_i)\Big(\prod_{j>i} f^j(\boldsymbol{u}_j+\boldsymbol{x})\Big)f^{\ell+1}(\boldsymbol{v}\otimes\boldsymbol{b})\right]_1.$$

We show there exists a PPT adversary $\mathcal{B}_5$ such that:

$$|\mathsf{adv}_5 - \mathsf{adv}_6| \leq \mathsf{adv}^{\mathsf{DDH}}_{\mathbf{G}_1,\mathcal{B}_5}(\lambda).$$

The adversary $\mathcal{B}_5$ takes as input a tuple $\big([r]_1, \{[\boldsymbol{a}_i]_1, [\boldsymbol{v}_i]_1\}_{i\in[n]}\big)$ where the values $[\boldsymbol{v}_i]_1$ are either of the form $[\boldsymbol{a}_i r]_1$ (case 1), or uniformly random over $\mathbf{G}_1^2$ (case 2). The adversary $\mathcal{B}_5$ samples $(\widetilde{\mathsf{msk}}, \widetilde{\mathsf{pk}}) \leftarrow \widetilde{\mathsf{Setup}}(\overline{\mathsf{pp}})$, $\boldsymbol{b}_j \leftarrow_{\mathrm{R}} \mathsf{DDH}$, $\boldsymbol{w}_j \leftarrow_{\mathrm{R}} \mathbb{Z}_p^2$ for all $j \in [n]$, $\boldsymbol{u}_k, \boldsymbol{s}_k \leftarrow_{\mathrm{R}} \mathbb{Z}_p^n$ for all $k \in [\ell]$, upon which it can simulate the view of the adversary $\mathcal{A}$ straightforwardly. In case 1, it simulates $\mathbf{Hybrid}_5$ to $\mathcal{A}$, whereas it simulates $\mathbf{Hybrid}_6$ in case 2.

• $\underline{\mathbf{Hybrid}_7}$: is the same as $\mathbf{Hybrid}_6$, except we replace the values $\{\boldsymbol{v}_i\}_{i\in[n]}$ by $\{\boldsymbol{v}_i + y_i\boldsymbol{h}\}_{i\in[n]}$, where $\boldsymbol{h} \leftarrow_{\mathrm{R}} \mathbb{Z}_p^2$. These values are identically distributed, since the $\boldsymbol{v}_i$ are sampled uniformly over $\mathbb{Z}_p^2$, independently of the challenge $\{y_i\}_{i\in[n]}$, which is chosen beforehand. Therefore, we have:

$$\mathsf{adv}_6 = \mathsf{adv}_7.$$

Consequently, the challenge ciphertext now contains:

$$\mathsf{ct}_i \leftarrow \widehat{\mathsf{KeyGen}}\left(\widehat{\mathsf{msk}}, \begin{bmatrix} y_i \\ \boldsymbol{v}_i + y_i\boldsymbol{h}\end{bmatrix}_1\right).$$

Moreover, the secret keys are now generated using for all $t \in [\ell]$:

$$\mathsf{sk}_t \leftarrow \widetilde{\mathsf{KeyGen}}_1\left(\widetilde{\mathsf{msk}}, [\mathbf{M}_t]_1, \left[f^t(\boldsymbol{s}_t)\prod_{t<i\leq\ell} f^i(\boldsymbol{u}_i+\boldsymbol{x})f^{\ell+1}((\boldsymbol{v}+\boldsymbol{y}\otimes\boldsymbol{h})\otimes\boldsymbol{b}),\right]_1\right),$$

and

$$\mathsf{sk}_{\ell+1} \leftarrow \widetilde{\mathsf{KeyGen}}_1\left(\widetilde{\mathsf{msk}}, [\boldsymbol{m}_{\ell+1}]_1, [v]_1\right),$$

where

$$[v]_1 = \left[f(\boldsymbol{x},\boldsymbol{v}+\boldsymbol{y}\otimes\boldsymbol{h},\boldsymbol{w}) + \sum_{i\in[\ell]}\Big(\prod_{j<i} f^j(\boldsymbol{x})\Big)f^i(\boldsymbol{s}_i)\Big(\prod_{j>i} f^j(\boldsymbol{u}_j+\boldsymbol{x})\Big)f^{\ell+1}(\boldsymbol{v}+\boldsymbol{y}\otimes\boldsymbol{h})\otimes\boldsymbol{b})\right]_1,$$

where $\boldsymbol{y} \otimes \boldsymbol{h} = (y_j \cdot \boldsymbol{h})_{j\in[n]} \in \mathbb{Z}^{2n}$, and $(\boldsymbol{v} + \boldsymbol{y} \otimes \boldsymbol{h}) \otimes \boldsymbol{b} = ((\boldsymbol{v}_i + y_i\boldsymbol{h})^\top \boldsymbol{b}_j)_{i,j\in[n]} \in \mathbb{Z}^{n^2}$.

• $\underline{\mathbf{Hybrid}_8}$: is the same as $\mathbf{Hybrid}_7$, except that we replace the values $[\boldsymbol{v}_i + y_i\boldsymbol{h}]_1$ by $[\boldsymbol{d}r_i + y_i\boldsymbol{h}]_1$ with $\boldsymbol{d} \leftarrow \mathsf{DDH}$ and $r_i \leftarrow_{\mathrm{R}} \mathbb{Z}_p$ for all $i \in [n]$, using the DDH assumption in $\mathbf{G}_1$. Consequently, the ciphertexts now contains:

$$\mathsf{ct}_i \leftarrow \widehat{\mathsf{KeyGen}}\left(\widehat{\mathsf{msk}}, \begin{bmatrix} y_i \\ \boldsymbol{d}r_i + y_i\boldsymbol{h} \end{bmatrix}_1\right).$$

Moreover, the secret keys are now generated using for all $t \in [\ell]$:

$$\mathsf{sk}_t \leftarrow \widetilde{\mathsf{KeyGen}}_1\left(\widetilde{\mathsf{msk}}, [\mathbf{M}_t]_1, \left[f^t(\boldsymbol{s}_t)\prod_{t<i\le\ell} f^i(\boldsymbol{u}_i + \boldsymbol{x})f^{\ell+1}((\boldsymbol{r}\otimes\boldsymbol{d} + \boldsymbol{y}\otimes\boldsymbol{h})\otimes\boldsymbol{b}),\right]_1\right),$$

and

$$\mathsf{sk}_{\ell+1} \leftarrow \widetilde{\mathsf{KeyGen}}_1\left(\widetilde{\mathsf{msk}}, [\boldsymbol{m}_{\ell+1}]_1, [v]_1\right),$$

where

$$[v]_1 = \Big[f(\boldsymbol{x}, \boldsymbol{r}\otimes\boldsymbol{d}+\boldsymbol{y}\otimes\boldsymbol{h}, \boldsymbol{w}) + \sum_{i\in[\ell]}\Big(\prod_{j<i} f^j(\boldsymbol{x})\Big)f^i(\boldsymbol{s}_i)\Big(\prod_{j>i} f^j(\boldsymbol{u}_j+\boldsymbol{x})\Big)f^{\ell+1}(\boldsymbol{r}\otimes\boldsymbol{d}+\boldsymbol{y}\otimes\boldsymbol{h})\otimes\boldsymbol{b})\Big]_1,$$

where $\boldsymbol{r} \otimes \boldsymbol{d} = (r_i \cdot \boldsymbol{d})_{i\in[n]} \in \mathbb{Z}^{2n}$, and $(\boldsymbol{r} \otimes \boldsymbol{d} + \boldsymbol{y} \otimes \boldsymbol{h}) \otimes \boldsymbol{b} = ((\boldsymbol{d}r_i + y_i\boldsymbol{v})^\top \boldsymbol{b}_j)_{i,j\in[n]} \in \mathbb{Z}^{n^2}$.

We show there exists a PPT adversary $\mathcal{B}_7$ such that:

$$|\mathsf{adv}_7 - \mathsf{adv}_8| \le \mathsf{adv}^{\mathsf{DDH}}_{\mathbf{G}_1,\mathcal{B}_7}(\lambda).$$

The adversary $\mathcal{B}_7$ takes as input a tuple $([\boldsymbol{d}]_1, \{[\boldsymbol{v}_i]_1\}_{i\in[n]})$ where the values $[\boldsymbol{v}_i]_1$ are either of the form $[\boldsymbol{d}r_i]_1$ (case 1), or uniformly random over $\mathbf{G}_1^2$ (case 2). The adversary $\mathcal{B}_7$ samples $\boldsymbol{h} \leftarrow_{\mathrm{R}} \mathbb{Z}_p^2$, $(\widetilde{\mathsf{msk}}, \widetilde{\mathsf{pk}}) \leftarrow \widetilde{\mathsf{Setup}}(\overline{\mathsf{pp}})$, $\boldsymbol{a}_i, \boldsymbol{b}_j \leftarrow_{\mathrm{R}} \mathsf{DDH}$, $\boldsymbol{w}_j \leftarrow_{\mathrm{R}} \mathbb{Z}_p^2$ for all $i, j \in [n]$, $\boldsymbol{u}_k, \boldsymbol{s}_k \leftarrow_{\mathrm{R}} \mathbb{Z}_p^n$ for all $k \in [\ell]$, upon which it can simulate the view of the adversary $\mathcal{A}$ straightforwardly. In case 1, it simulates $\mathbf{Hybrid}_8$ to $\mathcal{A}$, whereas it simulates $\mathbf{Hybrid}_7$ in case 2.

• $\underline{\mathbf{Hybrid}_9}$: is the same as $\mathbf{Hybrid}_8$, except 1) we change the distribution of $\boldsymbol{h}$ from uniformly random over $\mathbb{Z}_p^2$ to uniformly random over $\mathbb{Z}_p^2 \setminus \mathsf{Span}(\boldsymbol{d})$, which only induces a statistical change of $1/p$, given $\mathsf{Span}(\boldsymbol{d})$ is of size at most p; 2) we replace the values $\{\boldsymbol{w}_j\}_{j\in[n]}$ by $\{\boldsymbol{w}_j + z_i\boldsymbol{d}^\perp\}_{j\in[n]}$, where $\boldsymbol{d}^\perp \in \mathbb{Z}_p^2$ is such that $\boldsymbol{d}^\top\boldsymbol{d}^\perp = 0$ and $\boldsymbol{h}^\top\boldsymbol{d}^\perp = 1$ (note that such a vector exists as long as $\boldsymbol{h} \notin \mathsf{Span}(\boldsymbol{d})$). These values are identically distributed, since the $\boldsymbol{w}_j$ are sampled uniformly over $\mathbb{Z}_p^2$, independently of the challenge $\{z_j\}_{j\in[n]}$, which is chosen beforehand. Therefore, we have:

$$|\mathsf{adv}_8 - \mathsf{adv}_9| \le \frac{1}{p}.$$

Consequently, the ciphertexts now contains:

$$\mathsf{ct}'_j \leftarrow \widehat{\mathsf{KeyGen}}\left(\widehat{\mathsf{msk}}, \begin{bmatrix} z_j \\ -\boldsymbol{w}_j - z_j\boldsymbol{d}^\perp \end{bmatrix}_1\right).$$

Moreover, the secret keys are now generated using:

$$\mathsf{sk}_{\ell+1} \leftarrow \widetilde{\mathsf{KeyGen}}_1\left(\widetilde{\mathsf{msk}}, [\boldsymbol{m}_{\ell+1}]_1, [v]_1\right),$$

where

$$[v]_1 = \Big[f(\boldsymbol{x}, \boldsymbol{r}\otimes\boldsymbol{d} + \boldsymbol{y}\otimes\boldsymbol{h}, \boldsymbol{w}) + f(\boldsymbol{x},\boldsymbol{y},\boldsymbol{z}) + \sum_{i\in[\ell]}\Big(\prod_{j<i} f^j(\boldsymbol{x})\Big) f^i(\boldsymbol{s}_i)\Big(\prod_{j>i} f^j(\boldsymbol{u}_j+\boldsymbol{x})\Big) f^{\ell+1}\big((\boldsymbol{r}\otimes\boldsymbol{d}+\boldsymbol{y}\otimes\boldsymbol{h})\otimes\boldsymbol{b}\big)\Big]_1.$$

• $\underline{\mathbf{Hybrid}_{10}}$: is the same as $\mathbf{Hybrid}_9$, except the challenge ciphertext contains:

$$\mathsf{ct}_i \leftarrow \widehat{\mathsf{Enc}}\left(\widehat{\mathsf{msk}}, \begin{bmatrix}0\\ \boldsymbol{d}r_i + y_i\boldsymbol{h}\end{bmatrix}_1\right),\ \mathsf{ct}'_j \leftarrow \widehat{\mathsf{KeyGen}}\left(\widehat{\mathsf{msk}}, \begin{bmatrix}0\\ -\boldsymbol{w}_j\end{bmatrix}_1\right)$$

instead of

$$\mathsf{ct}_i \leftarrow \widehat{\mathsf{Enc}}\left(\widehat{\mathsf{msk}}, \begin{bmatrix}y_i\\ \boldsymbol{d}r_i + y_i\boldsymbol{h}\end{bmatrix}_1\right),\ \mathsf{ct}'_j \leftarrow \widehat{\mathsf{KeyGen}}\left(\widehat{\mathsf{msk}}, \begin{bmatrix}z_j\\ -\boldsymbol{w}_j - z_j\boldsymbol{d}^\perp\end{bmatrix}_1\right).$$

This transition is justified by the function-hiding IND security of $\widehat{\mathsf{IPFE}}$, which can be used since for all $i,j\in[n]$, we have $\binom{y_i}{\boldsymbol{d}r_i+y_i\boldsymbol{h}}^\top\binom{z_j}{-\boldsymbol{w}_j-z_j\boldsymbol{d}^\perp} = \binom{0}{\boldsymbol{d}r_i+y_i\boldsymbol{h}}^\top\binom{0}{-\boldsymbol{w}_j}$. The equality uses the fact that $\boldsymbol{d}^\top\boldsymbol{d}^\perp = 0$ and $\boldsymbol{h}^\top\boldsymbol{d}^\perp = 1$.

There exists a PPT adversary $\mathcal{B}_9$ such that:

$$|\mathsf{adv}_9 - \mathsf{adv}_{10}| \le \mathsf{adv}^{\mathsf{IND\text{-}FH}}_{\widehat{\mathsf{IPFE}},\mathcal{B}_9}(\lambda).$$

The adversary $\mathcal{B}_9$ first samples $\boldsymbol{d} \leftarrow_{\mathrm{R}} \mathsf{DDH}$, $\boldsymbol{h} \leftarrow_{\mathrm{R}} \mathbb{Z}_p^2 \setminus \mathsf{Span}(\boldsymbol{d})$, $\boldsymbol{d}^\perp \in \mathbb{Z}_p^2$ such that $\boldsymbol{d}^\top\boldsymbol{d}^\perp = 0$ and $\boldsymbol{h}^\top\boldsymbol{d}^\perp = 1$, $(\widetilde{\mathsf{msk}}, \widetilde{\mathsf{pk}}) \leftarrow \widetilde{\mathsf{Setup}}(\overline{\mathsf{pp}})$, $r_i \leftarrow_{\mathrm{R}} \mathbb{Z}_p$, $\boldsymbol{a}_i, \boldsymbol{b}_j \leftarrow_{\mathrm{R}} \mathsf{DDH}$, $\boldsymbol{w}_j \leftarrow_{\mathrm{R}} \mathbb{Z}_p^2$ for all $i,j\in[n]$, $\boldsymbol{u}_k, \boldsymbol{s}_k \leftarrow_{\mathrm{R}} \mathbb{Z}_p^n$ for all $k\in[\ell]$. It sends the challenge

$$\left\{\begin{bmatrix}y_i\\ \boldsymbol{d}r_i + y_i\boldsymbol{h}\end{bmatrix}_1, \begin{bmatrix}0\\ \boldsymbol{d}r_i + y_i\boldsymbol{h}\end{bmatrix}_1\right\}_{i\in[n]}, \left\{\begin{bmatrix}z_j\\ -\boldsymbol{w}_j - z_j\boldsymbol{d}^\perp\end{bmatrix}_1, \begin{bmatrix}0\\ -\boldsymbol{w}_j\end{bmatrix}_1\right\}_{j\in[n]}$$

to its own experiment, upon which it receives $\{\mathsf{ct}_i\}_{i\in[n]}$, encryptions of the left or right challenges; together with $\{\mathsf{ct}'_j\}_{j\in[n]}$, functional secret keys associated with the left or right challenges. In the left case, $\mathcal{B}_9$ simulates $\mathbf{Hybrid}_9$ to $\mathcal{A}$, whereas it simulates $\mathbf{Hybrid}_{10}$ in the right case.

• $\underline{\mathbf{Hybrid}_{11}}$: is the same as $\mathbf{Hybrid}_{10}$, except 1) we change the distribution of $\boldsymbol{h}$ from uniformly random over $\mathbb{Z}_p^2 \setminus \mathsf{Span}(\boldsymbol{d})$ to uniformly random over $\mathbb{Z}_p^2$; this introduces a statistical distance of $1/p$ since the size of $\mathsf{Span}(\boldsymbol{d})$ is at most p; 2) we replace the values $\{[\boldsymbol{d}r_i + y_i\boldsymbol{h}]_1\}_{i\in[n]}$ by $\{[\boldsymbol{v}_i + y_i\boldsymbol{h}]_1\}_{i\in[n]}$, where $\boldsymbol{v}_i \leftarrow_{\mathrm{R}} \mathbb{Z}_p^2$ for all $i\in[n]$, using the DDH assumption in $\mathbf{G}_1$. This transition is the reverse to the transition from $\mathbf{Hybrid}_5$ to $\mathbf{Hybrid}_6$.

Consequently, the challenge ciphertext now contains:

$$\mathsf{ct}_i \leftarrow \widehat{\mathsf{Enc}}\left(\widehat{\mathsf{msk}}, \begin{bmatrix} 0 \\ \boldsymbol{v}_i + y_i \boldsymbol{h} \end{bmatrix}_1\right),$$

and the secret keys are now generated using, for all $t \in [\ell]$:

$$\mathsf{sk}_t \leftarrow \widetilde{\mathsf{KeyGen}}_1\left(\widetilde{\mathsf{msk}}, [\mathbf{M}_t]_1, \left[f^t(\boldsymbol{s}_t) \prod_{t<i\leq\ell} f^i(\boldsymbol{u}_i + \boldsymbol{x}) f^{\ell+1}((\boldsymbol{v} + \boldsymbol{y} \otimes \boldsymbol{h}) \otimes \boldsymbol{b}),\right]_1\right),$$

and

$$\mathsf{sk}_{\ell+1} \leftarrow \widetilde{\mathsf{KeyGen}}_1\left(\widetilde{\mathsf{msk}}, [\boldsymbol{m}_{\ell+1}]_1, [v]_1\right),$$

where

$$[v]_1 = \Big[f(\boldsymbol{x}, \boldsymbol{v} + \boldsymbol{y} \otimes \boldsymbol{h}, \boldsymbol{w}) + f(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}) + \sum_{i\in[\ell]} \Big(\prod_{j<i} f^j(\boldsymbol{x})\Big) f^i(\boldsymbol{s}_i) \Big(\prod_{j>i} f^j(\boldsymbol{u}_j + \boldsymbol{x})\Big) f^{\ell+1}\big((\boldsymbol{v} + \boldsymbol{y} \otimes \boldsymbol{h}) \otimes \boldsymbol{b}\big)\Big]_1.$$

We show there exists a PPT adversary $\mathcal{B}_{10}$ such that:

$$|\mathsf{adv}_{10} - \mathsf{adv}_{11}| \leq \mathsf{adv}^{\mathsf{DDH}}_{\mathbf{G}_1, \mathcal{B}_{10}}(\lambda) + \frac{1}{p}.$$

The adversary $\mathcal{B}_{10}$ takes as input a tuple $([\boldsymbol{d}]_1, \{[\boldsymbol{v}_i]_1\}_{i\in[n]})$ where the vectors $[\boldsymbol{v}_i]_1$ are either of the form $[\boldsymbol{d}r_i]_1$ (case 1), or uniformly random over $\mathbf{G}_1^2$ (case 2). The adversary $\mathcal{B}_{10}$ samples $\boldsymbol{h} \leftarrow_{\mathrm{R}} \mathbb{Z}_p^2$, $(\widetilde{\mathsf{msk}}, \widetilde{\mathsf{pk}}) \leftarrow \widetilde{\mathsf{Setup}}(\overline{\mathsf{pp}})$, $\boldsymbol{a}_i, \boldsymbol{b}_j \leftarrow_{\mathrm{R}} \mathsf{DDH}$, $\boldsymbol{w}_j \leftarrow_{\mathrm{R}} \mathbb{Z}_p^2$ for all $i, j \in [n]$, $\boldsymbol{u}_k, \boldsymbol{s}_k \leftarrow_{\mathrm{R}} \mathbb{Z}_p^n$ and for all $k \in [\ell]$, upon which it can simulate the view of the adversary $\mathcal{A}$ straightforwardly. In case 1, it simulates $\mathbf{Hybrid}_{11}$ to $\mathcal{A}$, whereas it simulates $\mathbf{Hybrid}_{10}$ in case 2.

• $\underline{\mathbf{Hybrid}_{12}}$: is the same as $\mathbf{Hybrid}_{11}$, except we replace the values $\{\boldsymbol{v}_i + y_i \boldsymbol{h}\}_{i\in[n]}$ by $\{\boldsymbol{v}_i\}_{i\in[n]}$. These values are identically distributed, since the $\boldsymbol{v}_i$ are sampled uniformly over $\mathbb{Z}_p^2$, independently of the challenge $\{y_i\}_{i\in[n]}$, which is chosen beforehand. Therefore, we have:

$$\mathsf{adv}_{11} = \mathsf{adv}_{12}.$$

This transition is the reverse of the transition from $\mathbf{Hybrid}_6$ to $\mathbf{Hybrid}_7$. The secret keys are now generated using, for all $t \in [\ell]$:

$$\mathsf{sk}_t \leftarrow \widetilde{\mathsf{KeyGen}}_1\left(\widetilde{\mathsf{msk}}, [\mathbf{M}_t]_1, \left[f^t(\boldsymbol{s}_t) \prod_{t<i\leq\ell} f^i(\boldsymbol{u}_i + \boldsymbol{x}) f^{\ell+1}(\boldsymbol{v} \otimes \boldsymbol{b}),\right]_1\right),$$

and

$$\mathsf{sk}_{\ell+1} \leftarrow \widetilde{\mathsf{KeyGen}}_1\left(\widetilde{\mathsf{msk}}, [\boldsymbol{m}_{\ell+1}]_1, [v]_1\right),$$

where

$$[v]_1 = \left[f(\boldsymbol{x},\boldsymbol{v},\boldsymbol{w}) + f(\boldsymbol{x},\boldsymbol{y},\boldsymbol{z}) + \sum_{i\in[\ell]} \Big(\prod_{j<i} f^j(\boldsymbol{x})\Big) f^i(\boldsymbol{s}_i) \Big(\prod_{j>i} f^j(\boldsymbol{u}_j+\boldsymbol{x})\Big) f^{\ell+1}(\boldsymbol{v}\otimes\boldsymbol{b}) \right]_1 .$$

In $\mathbf{Hybrid}_{13}$, the challenge ciphertext $\left(\overline{\mathsf{ct}}, \{\mathsf{ct}_i, \mathsf{ct}'_j\}_{i,j\in[n]}\right)$ is as follows. $\overline{\mathsf{ct}} \leftarrow \widetilde{\mathsf{Enc}}\left(\widetilde{\mathsf{msk}}\right)$. For all $i,j\in[n]$: $\mathsf{ct}_i \leftarrow \widehat{\mathsf{Enc}}\left(\widehat{\mathsf{msk}}, \begin{bmatrix}0\\ \boldsymbol{v}_i\end{bmatrix}_1\right)$, $\mathsf{ct}'_j \leftarrow \widehat{\mathsf{KeyGen}}\left(\widehat{\mathsf{msk}}, \begin{bmatrix}0\\ -\boldsymbol{w}_j\end{bmatrix}_2\right)$, $\overline{\mathsf{ct}} \leftarrow \widetilde{\mathsf{Enc}}\left(\widetilde{\mathsf{msk}}\right)$.

This exactly corresponds to the experiment $\mathsf{Ideal}^{\mathsf{FE}}_{\mathcal{A},\mathcal{S}}(1^\lambda)$ for the simulator $\mathcal{S} = (\widetilde{\mathsf{Setup}}, \widetilde{\mathsf{Enc}}, \widetilde{\mathsf{KeyGen}})$ defined in Fig.8.2.

Summing up, we have PPT adversaries $\mathcal{B}_1$, $\mathcal{B}_2$, $\mathcal{B}_3$ and $\mathcal{B}_4$ such that:

$$\mathsf{adv}^{\mathsf{SIM}}_{\mathsf{PHFE},\mathcal{A}}(\lambda) \le \mathsf{adv}^{\mathsf{weak\text{-}SIM}}_{\overline{\mathsf{IPFE}},\mathcal{B}_1}(\lambda) + (\ell+1)\cdot \mathsf{adv}^{\mathsf{DDH}}_{\mathbf{G}_2,\mathcal{B}_2}(\lambda) + 3\cdot \mathsf{adv}^{\mathsf{DDH}}_{\mathbf{G}_1,\mathcal{B}_3}(\lambda) + \mathsf{adv}^{\mathsf{IND\text{-}FH}}_{\widehat{\mathsf{IPFE}},\mathcal{B}_4}(\lambda) + \frac{2}{p}.$$

□

$\underline{\widetilde{\mathsf{Setup}}(\mathsf{pp}):}$
$(\widetilde{\mathsf{pk}}, \widetilde{\mathsf{msk}}) \leftarrow \widetilde{\mathsf{Setup}}(\overline{\mathsf{pp}})$. For all $i,j\in[n]$: $\boldsymbol{a}_i, \boldsymbol{b}_j \leftarrow_{\mathrm{R}} \mathsf{DDH}$, $\boldsymbol{v}_i, \boldsymbol{w}_j \leftarrow_{\mathrm{R}} \mathbb{Z}_p^2$. For all $k\in[\ell]$: $\boldsymbol{u}_k \leftarrow_{\mathrm{R}} \mathbb{Z}_p^n$. $\widetilde{\mathsf{pk}} := \left(\widetilde{\mathsf{pk}}, \{[\boldsymbol{a}_i]_1, [\boldsymbol{b}_j]_2\}_{i,j\in[n]}\right)$, $\widetilde{\mathsf{msk}} := \left(\widetilde{\mathsf{msk}}, \{\boldsymbol{a}_i, \boldsymbol{b}_j, \boldsymbol{v}_i, \boldsymbol{u}_k\}_{i,j\in[n],k\in[\ell]}\right)$. Return $\widetilde{\mathsf{pk}}, \widetilde{\mathsf{msk}}$.

$\underline{\widetilde{\mathsf{Enc}}(\widetilde{\mathsf{msk}}):}$
$(\widehat{\mathsf{pk}}, \widehat{\mathsf{msk}}) \leftarrow \widehat{\mathsf{Setup}}(\widehat{\mathsf{pp}})$, $\overline{\mathsf{ct}} \leftarrow \widetilde{\mathsf{Enc}}\left(\widetilde{\mathsf{msk}}\right)$. For all $i,j\in[n]$: $\mathsf{ct}_i \leftarrow \widehat{\mathsf{Enc}}\left(\widehat{\mathsf{msk}}, \begin{bmatrix}0\\ \boldsymbol{v}_i\end{bmatrix}_1\right)$, $\mathsf{ct}'_j \leftarrow \widehat{\mathsf{KeyGen}}\left(\widehat{\mathsf{msk}}, \begin{bmatrix}0\\ -\boldsymbol{w}_j\end{bmatrix}_2\right)$. Return $\left(\overline{\mathsf{ct}}, \{\mathsf{ct}_i, \mathsf{ct}'_j\}_{i,j\in[n]}\right)$.

$\underline{\widetilde{\mathsf{KeyGen}}(\widetilde{\mathsf{msk}}, (f^0,\ldots,f^{\ell+1}), f(\boldsymbol{x},\boldsymbol{y},\boldsymbol{z}), \boldsymbol{x}):}$
For all $t\in[\ell]$, $\mathsf{sk}_t \leftarrow \widetilde{\mathsf{KeyGen}}_1\left(\widetilde{\mathsf{msk}}, [\mathbf{M}_t]_1, \left[f^t(\boldsymbol{s}_t)\prod_{t<i\le\ell} f^i(\boldsymbol{u}_i+\boldsymbol{x}) f^{\ell+1}(\boldsymbol{v}\otimes\boldsymbol{b}),\right]_1\right)$, $\mathsf{sk}_{\ell+1} \leftarrow \widetilde{\mathsf{KeyGen}}_1\left(\widetilde{\mathsf{msk}}, [\boldsymbol{m}_{\ell+1}]_1, [v]_1\right)$, where

$$[v]_1 = \left[f(\boldsymbol{x},\boldsymbol{v},\boldsymbol{w}) + f(\boldsymbol{x},\boldsymbol{y},\boldsymbol{z}) + \sum_{i\in[\ell]} \Big(\prod_{j<i} f^j(\boldsymbol{x})\Big) f^i(\boldsymbol{s}_i) \Big(\prod_{j>i} f^j(\boldsymbol{u}_j+\boldsymbol{x})\Big) f^{\ell+1}(\boldsymbol{v}\otimes\boldsymbol{b}) \right]_1 .$$

Return $\{\mathsf{sk}_t\}_{t\in[\ell+1]}$.

Figure 5: Simulator for the FE scheme depicted in Fig.8.2 for the functionality $\mathcal{F}^{\mathsf{PHFE}}$.

8.3 Constructing Inner-Product FE

Here, we build a public-key FE inner products, that is, the functionality $\mathcal{F}^{\mathsf{IPFE}'} = \mathbb{U}_{\dim\in\mathsf{poly}}\{\mathcal{F}_{\lambda,\dim(\lambda)}\}_\lambda$ defined with respect to the sequence of groups $\mathcal{G} = \{\mathcal{G}_\lambda\}_\lambda$ where for all $\dim \in \mathsf{poly}$, all $\lambda \in \mathbb{N}$, the set of function $\mathcal{F}_{\lambda,\dim(\lambda)}$ is the set of all functions $f : \mathbf{G}_1^{\dim} \to \mathbf{G}_T$, described by a vector $[\boldsymbol{y}]_2 \in \mathbf{G}_2^{\dim}$, such that given as input $\boldsymbol{x} \in \mathbb{Z}_p^{\dim}$, it outputs $[\boldsymbol{x}^\top\boldsymbol{y}]_T \in \mathbf{G}_T$. Our scheme is presented in Fig.8.3.

It builds upon the inner-product FE from [ALS16], that relies on the DDH assumption in pairing-free cyclic groups. We instead use a pairing group $\mathcal{G}_\lambda = (\mathbf{G}_1, \mathbf{G}_2, \mathbf{G}_T, p, P_1, P_2, e)$, where the ciphertexts will consist of group elements in $\mathbf{G}_1$, and the ALS functional secret key are embedded in $\mathbf{G}_2$, instead of $\mathbb{Z}_p$. Decryption now yields the inner product in $\mathbf{G}_T$.

This simple modification of ALS scheme already satisfies a simulation-security where the simulator needs to know the values $[\boldsymbol{x}^\top\boldsymbol{y}]_2 \in \mathbf{G}_2$ and $[\boldsymbol{y}]_2 \in \mathbf{G}_2^{\dim}$ in order to simulate the challenge ciphertext that encrypts $[\boldsymbol{x}]_1 \in \mathbf{G}_1^{\dim}$ and the functional secret key associated to $[\boldsymbol{y}]_2 \in \mathbf{G}_2^{\dim}$. This security property is inherited from the ALS scheme, which was proven simulation-secure in [Wee17] (see also [AGRW17, Appendix A]). Note that this is weaker than the standard simulation security notion, given in Definition 4.3, where the simulator gets the output of the function, which in this case, is $[\boldsymbol{x}^\top\boldsymbol{y}]_T \in \mathbf{G}_T$, not $[\boldsymbol{x}^\top\boldsymbol{y}]_2$.

For our purposes, we want it to be possible for the simulator to choose whether it simulates the adversary's view from the values $[\boldsymbol{x}^\top\boldsymbol{y}]_2$, $[\boldsymbol{y}]_2$ or $[\boldsymbol{x}^\top\boldsymbol{y}]_1$, $[\boldsymbol{y}]_1$. We achieve this by giving two copies of the encryption, one in $\mathbf{G}_1$, one $\mathbf{G}_2$, and splitting each functional secret key in two additive secret shares summing up to the actual key, one in $\mathbf{G}_2$ and one in $\mathbf{G}_1$. This simulation security relies on the fact that it is possible to produce both of these shares knowing the secret either in $\mathbf{G}_1$ or $\mathbf{G}_2$.

$\underline{\mathsf{Setup}(\mathsf{pp})}$:
Given $\mathsf{pp} = \dim \in \mathsf{poly}$, it computes $\boldsymbol{a} \leftarrow_{\mathrm{R}} \mathsf{DDH}$, $\mathbf{W} \leftarrow_{\mathrm{R}} \mathbb{Z}_p^{\dim\times 2}$, Return $\mathsf{pk} := \{[\boldsymbol{a}]_s, [\mathbf{W}\boldsymbol{a}]_s\}_{s\in[1,2]}$ and $\mathsf{msk} = \mathbf{W}$.

$\underline{\mathsf{Enc}(\mathsf{pk}, \boldsymbol{x} \in \mathbb{Z}^{\dim})}$:
$r \leftarrow_{\mathrm{R}} \mathbb{Z}_p$, $\boldsymbol{c} = \begin{pmatrix} \boldsymbol{a}r \\ \boldsymbol{x} + \mathbf{W}\boldsymbol{a}r \end{pmatrix}$. Return $([\boldsymbol{c}]_1, [\boldsymbol{c}]_2)$.

$\underline{\mathsf{KeyGen}(\mathsf{msk}, \boldsymbol{y} \in \mathbb{Z}^{\dim})}$:
$\boldsymbol{u} \leftarrow_{\mathrm{R}} \mathbb{Z}_p^{2+\dim}$, $\boldsymbol{k} = \begin{pmatrix} -\mathbf{W}^\top\boldsymbol{y} \\ \boldsymbol{y} \end{pmatrix}$. Return $([\boldsymbol{u}]_1, [\boldsymbol{k} - \boldsymbol{u}]_2)$.

$\underline{\mathsf{Dec}(\mathsf{ct}, \mathsf{sk})}$:
Parse $\mathsf{ct} = ([\boldsymbol{c}_1]_1, [\boldsymbol{c}_2]_2)$ and $\mathsf{sk} = ([\boldsymbol{k}_1]_1, [\boldsymbol{k}_2]_2)$. Return $[\boldsymbol{c}_1^\top\boldsymbol{k}_1 + \boldsymbol{c}_2^\top\boldsymbol{k}_2]_T$.

Figure 6: This is IPFE, an FE scheme for the functionality $\mathcal{F}^{\mathsf{IPFE}'}$, with weak-simulation security.

Linear efficiency.

Observe that the encryption time of any vector $\boldsymbol{x} \in \mathbb{Z}^{\dim}$ is proportional to $\dim$.

Correctness.

For any $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{Z}^{\dim}$:

$$[\boldsymbol{c}_1^\top \boldsymbol{k}_1 + \boldsymbol{c}_2^\top \boldsymbol{k}_2]_T = [\boldsymbol{c}^\top \boldsymbol{k}]_T = [\boldsymbol{x}^\top \boldsymbol{y}]_T.$$

Theorem 8.2 (Weak-simulation security)**.** *The scheme presented in Fig.8.3 is weakly-simulation secure (as per Definition 8.1) assuming the bilateral DLIN assumption. Namely, for any PPT adversary $\mathcal{A}$, there exists a PPT adversary $\mathcal{B}$ such that:*

$$\mathsf{adv}^{\mathsf{weak\text{-}SIM}}_{\mathsf{IPFE},\mathcal{A}}(\lambda) \leq \mathsf{adv}^{\mathsf{DLIN}}_{\mathcal{PG},\mathcal{B}}(\lambda) + \frac{1}{p}.$$

Proof. The proof proceeds using a series of hybrid games, described below. Let $\mathcal{A}$ be a PPT adversary against the weak simulation security of the scheme. For any game $\mathbf{Hybrid}_i$, we denote by $\mathsf{adv}_i := \Pr[1 \leftarrow \mathbf{Hybrid}_i(\mathcal{A})]$ the probability that $\mathbf{Hybrid}_i$ returns 1 when interacting with $\mathcal{A}$.

• $\underline{\mathbf{Hybrid}_0}$: is the real experiment as given in Definition 8.1.

• $\underline{\mathbf{Hybrid}_1}$: is the same as $\mathbf{Hybrid}_0$, except the challenge ciphertext is computed using $\boldsymbol{c} = \begin{pmatrix} \boldsymbol{u} \\ \boldsymbol{x} + \mathbf{W}\boldsymbol{u} \end{pmatrix}$ with $\boldsymbol{u} \leftarrow_{\mathrm{R}} \mathbb{Z}_p^2$ instead of $\boldsymbol{c} = \begin{pmatrix} \boldsymbol{a}r \\ \boldsymbol{x} + \mathbf{W}\boldsymbol{a}r \end{pmatrix}$ with $r \leftarrow_{\mathrm{R}} \mathbb{Z}_p$, using the bilateral DLIN assumption. We show there exists a PPT adversary $\mathcal{B}$ such that:

$$|\mathsf{adv}_0 - \mathsf{adv}_1| \leq \mathsf{adv}^{\mathsf{DLIN}}_{\mathcal{PG},\mathcal{B}}(\lambda).$$

The adversary $\mathcal{B}$ takes as input a tuple $([\mathbf{A}]_s, [\boldsymbol{z}]_s)_{s\in[1,2]}$, where the vectors $[\boldsymbol{z}]_s$ are of the form $[\mathbf{A}\boldsymbol{r}]_s$ with $\boldsymbol{r} \leftarrow_{\mathrm{R}} \mathbb{Z}_p^2$ (case 1) or uniformly random over $\mathbf{G}_s^2$ (case 2). The adversary $\mathcal{B}$ samples $\mathbf{W} \leftarrow_{\mathrm{R}} \mathbb{Z}_p^{\dim \times 3}$, upon which it can simulate the view of the adversary $\mathcal{A}$ straightforwardly. In case 1, it simulate $\mathbf{Hybrid}_0$, whereas it simulates $\mathbf{Hybrid}_1$ in case 2.

• $\underline{\mathbf{Hybrid}_2}$: is the same as $\mathbf{Hybrid}_1$, except the challenge ciphertext is computed using $\boldsymbol{u} \leftarrow_{\mathrm{R}} \mathbb{Z}_p^3 \setminus \mathsf{Span}(\mathbf{A})$ instead of $\boldsymbol{u} \leftarrow_{\mathrm{R}} \mathbb{Z}_p^3$. This only induces a statistical change of $1/p$ since the size of $\mathsf{Span}(\mathbf{A})$ is at most p^2. Thus:

$$|\mathsf{adv}_1 - \mathsf{adv}_2| \leq \frac{1}{p}.$$

• $\underline{\mathbf{Hybrid}_3}$: is the same as $\mathbf{Hybrid}_2$, except the challenge ciphertext is computed using:

$$\boldsymbol{c} = \begin{pmatrix} \boldsymbol{u} \\ \mathbf{W}\boldsymbol{u} \end{pmatrix},$$

where $\boldsymbol{u} \leftarrow_{\mathrm{R}} \mathbb{Z}_p^3 \setminus \mathsf{Span}(\mathbf{A})$. Besides, the functional keys are computed using:

$$\boldsymbol{k} = \begin{pmatrix} \boldsymbol{x}^\top \boldsymbol{y} - \mathbf{W}^\top \boldsymbol{y} \\ \boldsymbol{y} \end{pmatrix}.$$

We show that these two games are identically distributed, using the fact that for any $\boldsymbol{x} \in \mathbb{Z}^{\dim}$ and $\boldsymbol{a}^\perp \in \mathbb{Z}_p^3$, the following are identically distributed:

$$\mathbf{W} \;\text{ and }\; \mathbf{W} - \boldsymbol{x}(\boldsymbol{a}^\perp)^\top,$$

with $\mathbf{W} \leftarrow_{\mathrm{R}} \mathbb{Z}_p^{\dim \times 3}$. We use that fact with $\boldsymbol{x}$ the challenge chosen by the adversary, which is chosen beforehand, and therefore, independently of $\mathsf{msk} = \mathbf{W}$; and $\boldsymbol{a}^\perp \in \mathbb{Z}_p^3$ such that $\mathbf{A}^\top \boldsymbol{a}^\perp = \mathbf{0}$ and $\boldsymbol{u}^\top \boldsymbol{a}^\perp = 1$. Note that such a vector exists since $\boldsymbol{u} \notin \mathsf{Span}(\mathbf{A})$. The leftmost distribution corresponds to $\mathbf{Hybrid}_2$, whereas the rightmost distribution corresponds to $\mathbf{Hybrid}_3$. Thus:

$$\mathsf{adv}_2 = \mathsf{adv}_3.$$

It is clear hat $\mathbf{Hybrid}_3$ corresponds to $\mathsf{Ideal}^{\mathsf{IPFE}}_{\mathcal{A},\mathcal{S}}(1^\lambda)$ with the simulator $\mathcal{S}$ described in Fig.8.3. Consequently, we have:

$$\mathsf{adv}^{\mathsf{weak\text{-}SIM}}_{\mathsf{IPFE},\mathcal{A}}(\lambda) \leq \mathsf{adv}^{\mathsf{DLIN}}_{\mathcal{PG},\mathcal{B}}(\lambda) + \frac{1}{p}.$$

□

$\underline{\widetilde{\mathsf{Setup}}(\mathsf{pp})}$:
$\mathbf{A} \leftarrow_{\mathrm{R}} \mathsf{DLIN}$, $\mathbf{W} \leftarrow_{\mathrm{R}} \mathbb{Z}_p^{\dim \times 3}$, $\boldsymbol{u} \leftarrow_{\mathrm{R}} \mathbb{Z}_p^3 \setminus \mathsf{Span}(\mathbf{A})$, $\boldsymbol{a}^\perp \in \mathbb{Z}_p^3$ such that $\mathbf{A}^\top \boldsymbol{a}^\perp = \mathbf{0}$ and $\boldsymbol{u}^\top \boldsymbol{a}^\perp = 1$. Return $\widetilde{\mathsf{pk}} = \{[\mathbf{A}]_s, [\mathbf{WA}]_s\}_{s\in[1,2]}\}$ and $\widetilde{\mathsf{msk}} = (\mathbf{W}, \boldsymbol{u}, \boldsymbol{a}^\perp)$.

$\underline{\widetilde{\mathsf{Enc}}(\widetilde{\mathsf{msk}})}$:
$\boldsymbol{c} = \begin{pmatrix} \boldsymbol{u} \\ \mathbf{W}\boldsymbol{u} \end{pmatrix}$. Return $([\boldsymbol{c}]_1, [\boldsymbol{c}]_2)$.

$\underline{\text{For all } s \in [1,2],\ \widetilde{\mathsf{KeyGen}}_s(\widetilde{\mathsf{msk}}, [\boldsymbol{x}^\top \boldsymbol{y}]_s, [\boldsymbol{y}]_s)}$:
Return $\begin{bmatrix} \boldsymbol{x}^\top \boldsymbol{y} \cdot \boldsymbol{a}^\perp - \mathbf{W}^\top \boldsymbol{y} \\ \boldsymbol{y} \end{bmatrix}_s$.

Figure 7: Simulator for the FE scheme from Fig.8.3 for the functionality $\mathcal{F}^{\mathsf{IPFE}'}$.

9 Construction of ϵ-Simulation Secure FE

In this section, we construct a ϵ-simulation secure public-key functional encryption scheme FE for circuits $\mathcal{F}^{\mathsf{CIRC}} = \{\mathcal{F}^{\mathsf{CIRC}}_{\lambda,\mathsf{prmtr}}\}_{\lambda,\mathsf{prmtr}}$ for some $\epsilon \in (0,1)$. $\mathcal{F}^{\mathsf{CIRC}}_{\lambda,\mathsf{prmtr}}$ is the function class where for all λ and all polynomials $\mathsf{prmtr} = (n, d, \ell, \mathsf{size})$ it denotes the set of Boolean circuits with input length $n(\lambda)$, depth at most $d(\lambda)$, output length $\ell(\lambda)$, and size at most $\mathsf{size}(\lambda)$. It uses the following ingredients:

- ϵ-1LGFE: a secret-key FE scheme for the function class $\mathcal{F}^{\mathsf{CIRC}}$ defined above, satisfying the following properties:
  - (Security.) 1-key single ciphertext ϵ-simulation security as in Definition 4.4 for some constant $\epsilon \in (0,1)$ specified later. Note that although the scheme is for a single key, it however allows circuits with multiple output bits.
  - (Efficiency.) *levelled* compactness as in Definition 4.6. In particular, ciphertext size as well as the size of encryption circuit is $\mathsf{poly}(\lambda, n, d)$, independent of the function size size and output length ℓ.
  - (Structural property.) Special Structure* as per Definition 4.8. Recall, it says that:
    * (PP Syntax.) The pp generated by the $\mathsf{PPGen}(1^\lambda, \mathsf{prmtr})$ algorithm contains a $h_1(\lambda)$-bit prime modulus which is the modulus of the bilinear map $\mathcal{G}_\lambda$, p.
    * (Linear secret key Structure.) The master secret key is a vector in $\boldsymbol{s} \in \mathbb{Z}_p^{h_2(\lambda)}$. For any function $f \in \mathcal{F}_{\lambda,\mathsf{prmtr}}$, let $f = \{f_i\}_{i\in[\ell]}$ denote the circuit computing i^{th} bit of f. The functional secret key is of the form $\mathsf{sk}_f = \{\mathsf{sk}_{f_i}\}_{i\in[\ell]}$ where each $\mathsf{sk}_{f_i} = \langle \mathsf{pp}_{f_i}, \boldsymbol{s}\rangle + e_i \mod p$ where $e_i \leftarrow_{\mathrm{R}} \{0, \ldots, h_3(\lambda, n, \ell, d)\}$ and pp_{f_i} is some deterministic polynomial time computable function of pp and f_i. For our construction below we require that $h_3(\lambda, n, \ell, d) = 2^t - 1$ for some natural number $t = O(\log(n \cdot d \cdot \ell \cdot \mathsf{size}))$. We can always choose an a constant $\epsilon \in (0,1)$ for the construction in Section 7 such that there exists an ϵ-1LGFE scheme with this property, satisfying ϵ-simulation security. We use that value of ϵ.
    * (Linear + Round Decryption with polynomial decryption error.) There exists a deterministic poly-time algorithm such that given an encryption ct of $m \in \{0,1\}^n$ and a function $f = (f_1, \ldots, f_\ell) \in \mathcal{F}^{\mathsf{CIRC}}_{\lambda,\mathsf{prmtr}}$, for every $i \in [\ell]$, computes ct_{f_i} such that $|\mathsf{ct}_{f_i} - \langle \mathsf{pp}_{f_i}, \boldsymbol{s}\rangle - f_i(m)\lceil \frac{p}{2} \rceil| \leq h_4(\lambda, d, \ell, \mathsf{size})$. Given the secret-key for a function $f = (f_1, \ldots, f_\ell)$, this can be used to recover $f(m) = (f_1(m), \ldots, f_\ell(m))$.

  Such a scheme is constructed in Section 7.

- PHFE: a public-key PHFE for the class of functions $\mathcal{F}^{\mathsf{PHFE}}$ defined with respect to bilinear groups of order p (which is the same as the modulus of ϵ-1LGFE) and is in fact the order of group $\mathcal{G}_\lambda$. $\mathcal{F}^{\mathsf{PHFE}} = \{\mathcal{F}^{\mathsf{PHFE}}_{\lambda,n'}\}_{\lambda,n'}$ for every polynomial n' consists of all functions f that takes an input of the form $(\boldsymbol{x}, \boldsymbol{y}) \in \mathbb{Z}_p^{n'} \times \mathbb{Z}_p^{n'}$, and computes $f(\boldsymbol{x}, \boldsymbol{y}) = [\sum_{j,k} f_{j,k}(\boldsymbol{x}) \cdot y_j \cdot y_k]_T \in \mathbf{G}_T$ where $f_{j,k}$ is a constant degree polynomial over

$\boldsymbol{x}$ (i.e. an arithmetic circuit in NC^0), and $\mathbf{G}_T$ denotes the target group (see pairing groups in Definition 3). The scheme PHFE satisfies the following properties:

  - (Security.) 1-simulation security for unbounded key queries.
  - (Efficiency.) Linear run-time as per Definition 4.6.

  Such a scheme is constructed in Section 8 (in fact the scheme in Section 8 handles even a larger class of function, namely arithmetic NC^1 on the public part of the input). We set n' later.

- sPRG: a structured-seed PRG with stretch $\tau > 1$, linear efficiency as per Definition 6.1. This sPRG works with the modulus $p(\lambda)$ of the bilinear map $\mathcal{G}_\lambda$. The evaluation algorithm of sPRG computes an arithmetic NC^0 circuit on the public part of the seed, and a degree-2 polynomial on the secret part of the seed, that is, $\mathsf{sPRG} \in (\mathsf{arith\text{-}NC}^0, \deg\ 2)$. This sPRG is implementable by $\mathcal{F}^{\mathsf{PHFE}}$.

We now describe the construction.

Parameters: For sPRG, we set the length parameter to be $\ell^{\frac{1}{\tau}} \cdot \lambda$. Thus, $\ell_{\mathsf{sPRG}} = \ell^{\frac{1}{\tau}}\ \mathsf{poly}(\lambda)$ is the number of $\mathbb{Z}_p$ elements in the sPRG seed for some polynomial poly independent of the ℓ. Define $n' = h_2(\lambda, d) + \ell_{\mathsf{sPRG}}$. Let $t = \log_2(h_3(\lambda, n, \ell, d) + 1)$.

Construction: Please refer to the construction in Figure 9.

Correctness: Consider a message $m \in \{0,1\}^n$, a circuit $C \in \mathcal{F}_{\mathsf{prmtr}}$, an encryption of m, of the form $\mathsf{ct} = (\mathsf{ct}_1, \mathsf{ct}_2)$, and a functional decryption key $\mathsf{sk}_C = (\mathsf{sk}_{C_1}, \ldots, \mathsf{sk}_{C_\ell})$. Recall that decryption computes the following:

- It computes $[w_i]_T \leftarrow \mathsf{PHFE.Dec}(\mathsf{sk}_{C_i}, \mathsf{ct}_2)$. By correctness of PHFE, we have $[w_i]_T = [\langle \epsilon\text{-}\mathsf{1LGFE.pp}_{C_i}, \mathsf{msk}' \rangle + \sum_{j \in [t]} 2^{j-1} \cdot r_{(i-1)\cdot t + j}]_T$, where the bits $r_{(i-1)\cdot t+j}$ denote the output of sPRG.
- By the special structure* property of ϵ-$\mathsf{1LGFE}$, there exists a polynomial h_4, and a deterministic polynomial time algorithm that given ct_1 and C_i, outputs ct_i such that $\mathsf{ct}_i = \langle \epsilon\text{-}\mathsf{1LGFE.pp}_{C_i}, \mathsf{msk}' \rangle + e_i + C_i(m) \cdot \lceil p/2 \rceil$, where $|e_i| \leq h_4(\lambda, n, d, \ell)$.
- Then, the decryption computes $[z_i]_T = [\mathsf{ct}_{C_i} - w_i]_T$. By the first observation above, we have $w_i = \langle \epsilon\text{-}\mathsf{1LGFE.pp}_{C_i}, \mathsf{msk}' \rangle + e_i'$, where $|e_i'| \leq h_3(\lambda, n, d, \ell)$. Thus, if $C_i(m) = 0$, $|z_i| \leq h_3(\lambda, n, d, \ell) + h_4(\lambda, n, d, \ell)$, otherwise it is large in absolute value. Thus the decryption suceeds.

Sublinearity: We now bound the size of the ciphertext and the size of the encryption circuit. We do that in two cases. This is our theorem:

Theorem 9.1. *If* PHFE *satisfies linear efficiency and* ϵ-$\mathsf{1LGFE}$ *satisfies (levelled) output sublinearity, and* sPRG *has a stretch of* $\tau > 1$*, then the* FE *scheme above satisfies ciphertext sublinearity. In addition, if* sPRG *satisfies linear efficiency then* FE *satisfies output sublinearity.*

$\underline{\mathsf{FE.PPGen}(1^\lambda, \mathsf{prmtr}) :}$
Given 1^λ and the tuple of polynomials $\mathsf{prmtr} = (n, \mathsf{size}, d, \ell)$, it samples $\mathsf{PHFE.pp} \leftarrow \mathsf{PHFE.PPGen}(1^\lambda, 1^{n'})$, $\epsilon\text{-1LGFE.pp} \leftarrow \epsilon\text{-1LGFE.PPGen}(1^\lambda, \mathsf{prmtr})$ and $\mathsf{sPRG.pp} \leftarrow \mathsf{sPRG.PPGen}(1^\lambda, 1^{\ell^{\frac{1}{\tau}} \cdot \lambda})$, $I \leftarrow \mathsf{sPRG.IdSamp}(\mathsf{sPRG.pp})$. Let p denote the prime modulus of $\mathcal{G}_\lambda$. Output $\mathsf{pp} = (\mathsf{PHFE.pp}, \epsilon\text{-1LGFE.pp}, \mathsf{sPRG.pp}, I, p)$.

$\underline{\mathsf{FE.Setup(pp)}}$: Run $\mathsf{PHFE.Setup(PHFE.pp)} \rightarrow (\mathsf{PHFE.pk}, \mathsf{PHFE.msk})$. Set and output $\mathsf{FE.pk} = \mathsf{PHFE.pk}$ and $\mathsf{FE.msk} = \mathsf{PHFE.msk}$.

$\underline{\mathsf{FE.Enc}(\mathsf{FE.pk}, m \in \{0,1\}^n) :}$

- $\mathsf{msk}' \leftarrow \epsilon\text{-1LGFE.Setup}(\epsilon\text{-1LGFE.pp})$
- $\mathsf{ct}_1 \leftarrow \epsilon\text{-1LGFE.Enc}(\mathsf{msk}', m)$.
- $(\mathsf{P}, \mathsf{S}) \leftarrow \mathsf{SdSamp}(I)$.
- $\mathsf{ct}_2 \leftarrow \mathsf{PHFE.Enc}(\mathsf{PHFE.pk}, (\mathsf{P}, (\mathsf{S}, \mathsf{msk}')))$.

It returns $\mathsf{ct} = (\mathsf{ct}_1, \mathsf{ct}_2)$.

$\underline{\mathsf{FE.KeyGen}(\mathsf{FE.msk}, C)}$: Given as input a circuit $C \in \mathcal{F}_{\mathsf{prmtr}}$, denote $C = (C_1, \ldots, C_\ell)$ where each C_i is the circuit computing the i^{th} output bit of C. For every $i \in [\ell]$, do the following:

- let $\epsilon\text{-1LGFE.pp}_{C_i}$ be the vector computed deterministically from $\epsilon\text{-1LGFE.pp}$ and C_i such that $\mathsf{sk}_{C_i} \approx \langle \mathsf{msk}', \epsilon\text{-1LGFE.pp}_{C_i} \rangle$ (see the linear secret key structure in Definition 4.8).
- Compute $\mathsf{sk}_{C_i} \leftarrow \mathsf{PHFE.KeyGen}(\mathsf{PHFE.msk}, f_i)$ where f_i takes as input $(\mathsf{P}, (\mathsf{S}, \mathsf{msk}'))$ and outputs $\langle \mathsf{msk}', \epsilon\text{-1LGFE.pp}_{C_i} \rangle + \sum_{j \in [1,t]} 2^{j-1} \cdot r_{(i-1)\cdot t+j}$, where for all $\theta \in [m]$, r_θ denotes the θ'th bit output by $\mathsf{sPRG.Eval}(I, \mathsf{sd}) \in \{0,1\}^m$.

It returns $\mathsf{sk}_C = (\mathsf{sk}_{C_1}, \ldots \mathsf{sk}_{C_\ell})$.

$\underline{\mathsf{FE.Dec}(\mathsf{sk}_C, \mathsf{ct}) :}$ Parse $\mathsf{sk}_C = (\mathsf{sk}_{C_1}, ..., \mathsf{sk}_{C_\ell})$ and $\mathsf{ct} = (\mathsf{ct}_1, \mathsf{ct}_2)$. For every $i \in [\ell]$, do the following:

- By the special structure* of ϵ-1LGFE, compute $\mathsf{ct}_{C,i}$ using the ciphertext ct_1.
- Compute $[w_i]_T \leftarrow \mathsf{PHFE.Dec}(\mathsf{sk}_{C_i}, \mathsf{ct}_2)$.
- Compute $[z_i]_T = [\mathsf{ct}_{C_i} - w_i]_T$.
- Check if $|z_i| \leq h_3(\lambda, n, d, \ell) + h_4(\lambda, n, d, \ell)$ (by brute-force). If so set $y_i = 0$. Otherwise, set $y_i = 1$. Output $(y_1, ..., y_\ell)$.

Figure 8: Construction of Functional Encryption Scheme FE.

We first find out the size of an encryption. The ciphertext consists of two components: ct_1 and ct_2. The size of ct_1 is $\ell^{1-\gamma_1}\,\mathsf{poly}(\lambda, n, d)$ for some $\gamma_1 > 0$ and some polynomial poly due to (levelled) output sublinear efficiency of ϵ-1LGFE. The size of ct_2 is $n' \cdot \mathsf{poly}(\lambda)$ for some polynomial poly due to the linear efficiency of PHFE. Note that $n' = p_2(\lambda, d) + \ell^{\frac{1}{\tau}}\lambda$. Thus, overall the size of the ciphertext is $\ell^{\max\{1-\gamma_1, \frac{1}{\tau}\}} \cdot \mathsf{poly}(\lambda, n, d)$ for some polynomial poly.

Now, assume that sPRG in addition has linear efficiency. The size of the circuit computing ct is the size of the circuit computing ct_1 and ct_2. ct_1 can be computed by a circuit of size $\ell^{1-\gamma_1}\,\mathsf{poly}(\lambda, n, d)$ for some constant $\gamma_1 > 0$ and some polynomial poly due to output sublinearity of ϵ-1LGFE. The size of the circuit computing ct_2 is $n'\,\mathsf{poly}(\lambda)$ due to linear efficiency of PHFE and sPRG. Thus, combined, the size of the circuit computing ct is also $O(\ell^{\max\{1-\gamma_1, \frac{1}{\tau}\}}\,\mathsf{poly}(\lambda, n, d))$ for some polynomial poly.

Security: We now prove security. Let the parameters be set as described in the construction. Then, we prove the following:

Theorem 9.2. *If* ϵ-1LGFE *is (single-key)* ϵ*-simulation secure,* PHFE *is many-key* 1*-simulation secure,* sPRG *is a secure structured seed PRG and parameters are set as described above, then* FE *is (single-key)* ϵ*-simulation secure.*

We now list hybrids. The first hybrid corresponds to the real security game, whereas the last hybrid is our ideal security game. This hybrid implicitly defines the simulated algorithms $\widetilde{\mathsf{Setup}}$, $\widetilde{\mathsf{Enc}}$ and $\widetilde{\mathsf{KeyGen}}$. We argue indistinguishability between each one of them. The differences between hybrids are highlighted in red.

$\mathbf{Hybrid}_0$:

- Adversary outputs a circuit $C \in \mathcal{F}^{\mathsf{CIRC}}_{\lambda,\mathsf{prmtr}}$ along with a message $m \in \{0,1\}^n$.
- Run PPGen as in the scheme to compute pp.
- Run $\mathsf{PHFE.Setup}(\mathsf{PHFE.pp}) \to (\mathsf{PHFE.pk}, \mathsf{PHFE.msk})$. Set $\mathsf{FE.pk} = \mathsf{PHFE.pk}$ and $\mathsf{FE.msk} = \mathsf{PHFE.msk}$. Send $\mathsf{FE.pk}$ to the adversary.
- To encrypt the message m:
  - $\mathsf{msk}' \leftarrow \epsilon\text{-}\mathsf{1LGFE.Setup}(\epsilon\text{-}\mathsf{1LGFE.pp})$
  - $\mathsf{ct}_1 \leftarrow \epsilon\text{-}\mathsf{1LGFE.Enc}(\mathsf{msk}', m)$.
  - $(\mathsf{P}, \mathsf{S}) \leftarrow \mathsf{SdSamp}(I)$.
  - $\mathsf{ct}_2 \leftarrow \mathsf{PHFE.Enc}(\mathsf{PHFE.pk}, (\mathsf{P}, (\mathsf{S}, \mathsf{msk}')))$.
  - Give $(\mathsf{ct}_1, \mathsf{ct}_2)$ to the adversary.
- For generating the secret key for the circuit C, do the following steps. Denote $C = (C_1, \ldots, C_\ell)$ where each C_i is the circuit computing the i^{th} output bit of C. For every $i \in [\ell]$, do the following:

- Let ϵ-1LGFE.pp_{C_i} be the vector computed deterministically from ϵ-1LGFE.pp and C_i such that $\mathsf{sk}_{C_i} \approx \langle \mathsf{msk}', \epsilon\text{-1LGFE.pp}_{C_i} \rangle$ (see the linear secret key structure in Definition 4.8).
- Compute $\mathsf{sk}_{C_i} \leftarrow \mathsf{PHFE.KeyGen}(\mathsf{PHFE.msk}, f_i)$ where f_i takes as input $(\mathsf{P}, (\mathsf{S}, \mathsf{msk}'))$ and outputs $\langle \mathsf{msk}', \epsilon\text{-1LGFE.pp}_{C_i} \rangle + \sum_{j \in [1,t]} 2^{j-1} \cdot r_{(i-1)\cdot t+j}$, where for all $\theta \in [\ell]$, r_θ denotes the θ'th bit output by $\mathsf{sPRG.Eval}(I, \mathsf{sd}) \in \{0,1\}^\ell$.

- Give to the adversary $\mathsf{sk}_C = (\mathsf{sk}_{C_1}, \ldots \mathsf{sk}_{C_\ell})$.

In the next hybrid, we use the simulator for the PHFE scheme. These two hybrids are indistinguishable due to the security of the PHFE scheme.

Hybrid$_1$:

- Adversary outputs a circuit C along with a message m.
- Run PPGen as in the scheme to compute pp.
- Run $\mathsf{PHFE.}\widetilde{\mathsf{Setup}}(\mathsf{PHFE.pp}) \rightarrow (\mathsf{PHFE.pk}, \mathsf{PHFE.td})$. Set $\mathsf{FE.pk} = \mathsf{PHFE.pk}$ and $\mathsf{FE.msk} = \mathsf{PHFE.td}$.
- Compute $\mathsf{msk}' \leftarrow \epsilon\text{-1LGFE.Setup}(\epsilon\text{-1LGFE.pp})$
- Compute $(\mathsf{P}, \mathsf{S}) \leftarrow \mathsf{SdSamp}(I)$.
- Compute $\mathsf{ct}_1 \leftarrow \epsilon\text{-1LGFE.Enc}(\mathsf{msk}', m)$.
- For $i \in [\ell]$ let $v_i = \langle \mathsf{msk}', \epsilon\text{-1LGFE.pp}_{C_i} \rangle + \sum_{j \in [1,t]} 2^{j-1} \cdot r_{(i-1)\cdot t+j}$ where $\boldsymbol{r} = \mathsf{sPRG.Eval}(I, \mathsf{sd})$.
- Compute $\mathsf{ct}_2 \leftarrow \mathsf{PHFE.}\widetilde{\mathsf{Enc}}(\mathsf{td}, w)$ where $w = (\mathsf{P}, \{f_i\}_{i \in [\ell]}, \{v_i\}_{i \in [\ell]})$. Let $\mathsf{ct} = (\mathsf{ct}_1, \mathsf{ct}_2)$.
- For all $j \in [\ell]$, compute $\mathsf{sk}_{C_j} \leftarrow \mathsf{PHFE.}\widetilde{\mathsf{KeyGen}}(\mathsf{td}, f_j, w)$. Let $\mathsf{sk}_C = (\mathsf{sk}_{C_1}, \ldots, \mathsf{sk}_{C_\ell})$.
- Give to the adversary $\mathsf{FE.pk}$, $\mathsf{ct} = (\mathsf{ct}_1, \mathsf{ct}_2)$ and $\mathsf{sk}_C = (\mathsf{sk}_{C_1}, \ldots, \mathsf{sk}_{C_\ell})$.

In the next, hybrid, we replace $\boldsymbol{r}$ with a random string. These two hybrids are indistinguishable due to the security of sPRG. For this to happen we need $\ell \cdot t$ to be less than k^τ where k is length parameter of sPRG. Note that the length parameter of sPRG is set as $\ell^{\frac{1}{\tau}}\lambda$, thus $k^\tau = \lambda^\tau \cdot \ell$. Since $t = O(\log(\lambda \cdot n \cdot \ell \cdot d \cdot \mathsf{size}))$, this holds.

Hybrid$_2$:

- Adversary outputs a circuit C along with a message m.
- Run PPGen as in the scheme to compute pp.
- Run $\mathsf{PHFE.}\widetilde{\mathsf{Setup}}(\mathsf{PHFE.pp}) \rightarrow (\mathsf{PHFE.pk}, \mathsf{PHFE.td})$. Set $\mathsf{FE.pk} = \mathsf{PHFE.pk}$ and $\mathsf{FE.msk} = \mathsf{PHFE.td}$.

- Compute $\mathsf{msk}' \leftarrow \epsilon\text{-}\mathsf{1LGFE.Setup}(\epsilon\text{-}\mathsf{1LGFE.pp})$
- Compute $(\mathsf{P}, \mathsf{S}) \leftarrow \mathsf{SdSamp}(I)$.
- Compute $\mathsf{ct}_1 \leftarrow \epsilon\text{-}\mathsf{1LGFE.Enc}(\mathsf{msk}', m)$.
- For $i \in [\ell]$ let $v_i = \langle \mathsf{msk}', \epsilon\text{-}\mathsf{1LGFE.pp}_{C_i} \rangle + \sum_{j \in [1,t]} 2^{j-1} \cdot r_{(i-1)\cdot t+j}$ where $\boldsymbol{r} \leftarrow_{\mathrm{R}} \{0,1\}^{\ell\lambda^\tau}$.
- Compute $\mathsf{ct}_2 \leftarrow \mathsf{PHFE.}\widetilde{\mathsf{Enc}}(\mathsf{td}, w)$ where $w = (\mathsf{P}, \{f_i\}_{i\in[\ell]}, \{v_i\}_{i\in[\ell]})$. Let $\mathsf{ct} = (\mathsf{ct}_1, \mathsf{ct}_2)$.
- For all $j \in [\ell]$, compute $\mathsf{sk}_{C_j} \leftarrow \mathsf{PHFE.}\widetilde{\mathsf{KeyGen}}(\mathsf{td}, f_j, w)$. Let $\mathsf{sk}_C = (\mathsf{sk}_{C_1}, \ldots, \mathsf{sk}_{C_\ell})$.
- Give to the adversary $\mathsf{FE.pk}$, $\mathsf{ct} = (\mathsf{ct}_1, \mathsf{ct}_2)$ and $\mathsf{sk}_C = (\mathsf{sk}_{C_1}, \ldots, \mathsf{sk}_{C_\ell})$.

Finally in the last hybrid, we simulate $\mathsf{ct}_1, v_1, \ldots, v_\ell$ using ϵ-1LGFE.Sim. Note that this is possible since the values $v_1, \ldots, v_\ell$ exactly correspond to ϵ-1LGFE functional decryption keys (see the special structure* property in Definition 4.8) . These two hybrids are indistinguishable due to ϵ-simulation security of the ϵ-1LGFE scheme.

$\mathbf{Hybrid}_3$:

- Adversary outputs a circuit C along with a message m.
- Run PPGen as in the scheme to compute pp.
- Run $\mathsf{PHFE.}\widetilde{\mathsf{Setup}}(\mathsf{PHFE.pp}) \rightarrow (\mathsf{PHFE.pk}, \mathsf{PHFE.td})$. Set $\mathsf{FE.pk} = \mathsf{PHFE.pk}$ and $\mathsf{FE.msk} = \mathsf{PHFE.td}$.
- Compute $\mathsf{td} \leftarrow \epsilon\text{-}\mathsf{1LGFE.}\widetilde{\mathsf{Setup}}(\epsilon\text{-}\mathsf{1LGFE.pp})$
- Compute $(\mathsf{P}, \mathsf{S}) \leftarrow \mathsf{SdSamp}(I)$.
- Let $\gamma = \epsilon\text{-}\mathsf{1LGFE.Sample}(C, m)$.
- Compute $\mathsf{ct}_1 \leftarrow \epsilon\text{-}\mathsf{1LGFE.}\widetilde{\mathsf{Enc}}(\mathsf{td}, \gamma)$ and for $i \in [\ell]$, $v_i \leftarrow \epsilon\text{-}\mathsf{1LGFE.}\widetilde{\mathsf{KeyGen}}(\mathsf{td}, C_i, \gamma)$.
- Compute $\mathsf{ct}_2 \leftarrow \mathsf{PHFE.}\widetilde{\mathsf{Enc}}(\mathsf{td}, w)$ where $w = (\mathsf{P}, \{f_i\}_{i\in[\ell]}, \{v_i\}_{i\in[\ell]})$. Let $\mathsf{ct} = (\mathsf{ct}_1, \mathsf{ct}_2)$.
- For all $j \in [\ell]$, compute $\mathsf{sk}_{C_j} \leftarrow \mathsf{PHFE.}\widetilde{\mathsf{KeyGen}}(\mathsf{td}, f_j, w)$. Let $\mathsf{sk}_C = (\mathsf{sk}_{C_1}, \ldots, \mathsf{sk}_{C_\ell})$.
- Give to the adversary $\mathsf{FE.pk}$, $\mathsf{ct} = (\mathsf{ct}_1, \mathsf{ct}_2)$ and $\mathsf{sk}_C = (\mathsf{sk}_{C_1}, \ldots, \mathsf{sk}_{C_\ell})$.

In $\mathbf{Hybrid}_3$, only the value γ is needed to simulate the adversary's view, which is equal to $C, C(m)$ with probability ϵ and equal to C, m with remaining probability. Namely, the simulator for FE does the following. Given as input the parameters pp, it samples $\mathsf{FE.pk}$, ϵ-1LGFE.pk, using the simulator for PHFE and ϵ-1LGFE. Then it samples $(\mathsf{P}, \mathsf{S}) \leftarrow \mathsf{SdSamp}(I)$. It then simulates ct_1 and ϵ-1LGFE keys $\{v_i\}_{i\in[\ell]}$ using the simulator for ϵ-1LGFE. This uses γ as an input. Then, it generates PHFE ciphertext ct_2 and keys $\{\mathsf{sk}_{C_j}\}_{j\in[\ell]}$. This is done using simulator for PHFE scheme. This uses $\{v_i\}_{i\in[\ell]}$ as an input, alongside P and function descriptions f_j. This gives us the simulator for the scheme FE.

10 Summing Up: Construction of $i\mathcal{O}$

We obtain the following main result:

Theorem 10.1. *Assuming the following assumptions hold:*

- *SXDH and bilateral DLIN assumptions over bilinear maps.*
- *Learning with Error assumption.*
- *A structured seed PRG* sPRG *exists (can be instantiated using a pseudorandom generator PRG,* G *satisfying* $\mathsf{G} - \mathsf{LWEleak}_{D,\epsilon,\rho}$ *security for some constants* $D \geq 3$ *and constants* $\epsilon, \rho \in (0, 0.5)$*.)*

There exists a (levelled) sublinearly efficient public-key functional encryption scheme for all polynomial sized circuits. Further, the scheme is subexponentially secure if each of these assumptions are subexponentially secure.

Since subexponentially secure sublinearly-efficient public key encryption scheme implies $i\mathcal{O}$ [AJ15, BV15, KNT18], we obtain the following result:

Theorem 10.2. *Assuming the subexponential versions of the following assumptions hold:*

- *SXDH and bilateral DLIN assumptions over bilinear maps.*
- *Learning with Error assumption.*
- *A structured seed PRG,* sPRG *exists (can be instantiated using a pseudorandom generator* G *satisfying* $\mathsf{G}-\mathsf{LWEleak}_{D,\epsilon,\rho}$ *security for some constants* $D \geq 3$ *and constants* $\epsilon, \rho \in (0, 0.5)$*.)*

Then, there exists an $i\mathcal{O}$ *scheme for all circuits.*

We show the above by combining many theorems.

First, use Theorem 8.1 to construct a PHFE scheme from SXDH and Bilateral DLIN assumptions. Then, use Theorem 6.1 to construct a structured seed PRG from $\mathsf{G}-\mathsf{LWEleak}_{D,\epsilon,\rho}$ assumption. Then, for any constant $\epsilon' \in (0, 1)$ use LWE to construct an ϵ'-simulation secure, (levelled) compact, single ciphertext, secret-key FE scheme satisfying special structure* as described in Theorem 7.2.

Then, combine the above three ingredients to build a sublinearly efficient, public-key, single key ϵ''-simulation secure functional encryption scheme as described in Theorem 9.2 for some constant $\epsilon'' \in (0, 1)$.

Finally use Theorem 5.1, to construct a public-key sublinearly efficient FE with IND security. Observe that, if each of the assumptions are subexponentially secure, then the final result is also subexponentially secure.

We also observe that subsequent to the initial publication of our work, Wee [Wee20b] showed how to construct a PHFE scheme from only the Bilateral DLIN assumption. By using Wee's theorem in place of Theorem 8.1, we obtain:

Theorem 10.3. *Assuming the following assumptions hold:*

- *Bilateral DLIN assumption over bilinear maps (immediately implied by the DLIN assumption over symmetric bilinear groups).*
- *Learning with Error assumption.*
- *A structured seed PRG* sPRG *exists (can be instantiated using a pseudorandom generator PRG,* G *satisfying* $\mathsf{G} - \mathsf{LWEleak}_{D,\epsilon,\rho}$ *security for some constants* $D \geq 3$ *and constants* $\epsilon, \rho \in (0, 0.5)$*.)*

There exists a (levelled) sublinearly efficient public-key functional encryption scheme for all polynomial sized circuits. Further, the scheme is subexponentially secure if each of these assumptions are subexponentially secure.

and

Theorem 10.4. *Assuming the subexponential versions of the following assumptions hold:*

- *Bilateral DLIN assumption over bilinear maps (immediately implied by the DLIN assumption over symmetric bilinear groups).*
- *Learning with Error assumption.*
- *A structured seed PRG,* sPRG *exists (can be instantiated using a pseudorandom generator* G *satisfying* $\mathsf{G}-\mathsf{LWEleak}_{D,\epsilon,\rho}$ *security for some constants* $D \geq 3$ *and constants* $\epsilon, \rho \in (0, 0.5)$*.)*

Then, there exists an $i\mathcal{O}$ *scheme for all circuits.*

11 Acknowledgements

This work was partly carried out while Romain Gay was a postdoctoral researcher at UC Berkeley, then a postdoctoral research at Cornell Tech. Aayush Jain was partially supported by grants listed under Amit Sahai, a Google PhD fellowship and a DIMACS award. This work was partly carried out during a research visit conducted with support from DIMACS in association with its Special Focus on Cryptography.

Huijia Lin was supported by NSF grants CNS-1528178, CNS-1929901, CNS-1936825 (CAREER), the Defense Advanced Research Projects Agency (DARPA) and Army Research Office (ARO) under Contract No. W911NF-15-C-0236, and a subcontract No. 2017-002 through Galois.

Amit Sahai was supported in part from DARPA SAFEWARE and SIEVE awards, NTT Research, NSF Frontier Award 1413955, and NSF grant 1619348, BSF grant 2012378, a Xerox Faculty Research Award, a Google Faculty Research Award, an equipment grant from Intel, and an Okawa Foundation Research Grant. This material is based upon work supported by the Defense Advanced Research Projects Agency through Award HR00112020024 and the ARL under Contract W911NF-15-C- 0205. Amit Sahai is also grateful for the contributions of the LADWP to this effort.

The views expressed are those of the authors and do not reflect the official policy or position of the Department of Defense, DARPA, ARO, Simons, Intel, Okawa Foundation, ODNI, IARPA, DIMACS, BSF, Xerox, the National Science Foundation, NTT Research, Google, or the U.S. Government.

12 References

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

[ABR12] Benny Applebaum, Andrej Bogdanov, and Alon Rosen. A dichotomy for local small-bias generators. In Ronald Cramer, editor, *TCC 2012*, volume 7194 of *LNCS*, pages 600–617. Springer, Heidelberg, March 2012.

[ABSV15] Prabhanjan Ananth, Zvika Brakerski, Gil Segev, and Vinod Vaikuntanathan. From selective to adaptive security in functional encryption. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part II*, volume 9216 of *LNCS*, pages 657–677. Springer, Heidelberg, August 2015.

[ACF+15] Martin R. Albrecht, Carlos Cid, Jean-Charles Faugère, Robert Fitzpatrick, and Ludovic Perret. Algebraic algorithms for LWE problems. *ACM Commun. Comput. Algebra*, 49(2):62, 2015.

[ACGU20] Michel Abdalla, Dario Catalano, Romain Gay, and Bogdan Ursu. Inner-product functional encryption with fine-grained access control. Cryptology ePrint Archive, Report 2020/577, 2020. `https://eprint.iacr.org/2020/577`.

[AG11] Sanjeev Arora and Rong Ge. New algorithms for learning in presence of errors. In Luca Aceto, Monika Henzinger, and Jiri Sgall, editors, *ICALP 2011, Part I*, volume 6755 of *LNCS*, pages 403–415. Springer, Heidelberg, July 2011.

[AGIS14] Prabhanjan Ananth, Divya Gupta, Yuval Ishai, and Amit Sahai. Optimizing obfuscation: Avoiding Barrington's theorem. In *ACM CCS*, pages 646–658, 2014.

[Agr19] Shweta Agrawal. Indistinguishability obfuscation without multilinear maps: New methods for bootstrapping and instantiation. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part I*, volume 11476 of *LNCS*, pages 191–225. Springer, Heidelberg, May 2019.

[AGRW17] Michel Abdalla, Romain Gay, Mariana Raykova, and Hoeteck Wee. Multi-input inner-product functional encryption from pairings. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part I*, volume 10210 of *LNCS*, pages 601–626. Springer, Heidelberg, April / May 2017.

[AJ15] Prabhanjan Ananth and Abhishek Jain. Indistinguishability obfuscation from compact functional encryption. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part I*, volume 9215 of *LNCS*, pages 308–326. Springer, Heidelberg, August 2015.

[AJL+19] Prabhanjan Ananth, Aayush Jain, Huijia Lin, Christian Matt, and Amit Sahai. Indistinguishability obfuscation without multilinear maps: New paradigms via low degree weak pseudorandomness and security amplification. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part III*, volume 11694 of *LNCS*, pages 284–332. Springer, Heidelberg, August 2019.

[AJS18] Prabhanjan Ananth, Aayush Jain, and Amit Sahai. Indistinguishability obfuscation without multilinear maps: io from lwe, bilinear maps, and weak pseudorandomness. *IACR Cryptology ePrint Archive*, 2018:615, 2018.

[AL16] Benny Applebaum and Shachar Lovett. Algebraic attacks against random local functions and their countermeasures. In Daniel Wichs and Yishay Mansour, editors, *48th ACM STOC*, pages 1087–1100. ACM Press, June 2016.

[ALS16] Shweta Agrawal, Benoît Libert, and Damien Stehlé. Fully secure functional encryption for inner products, from standard assumptions. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part III*, volume 9816 of *LNCS*, pages 333–362. Springer, Heidelberg, August 2016.

[AP20] Shweta Agrawal and Alice Pellet-Mary. Indistinguishability obfuscation without maps: Attacks and fixes for noisy linear FE. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology - EUROCRYPT 2020 - 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10-14, 2020, Proceedings, Part I*, volume 12105 of *Lecture Notes in Computer Science*, pages 110–140. Springer, 2020.

[App12] Benny Applebaum. Pseudorandom generators with long stretch and low locality from random local one-way functions. In Howard J. Karloff and Toniann Pitassi, editors, *44th ACM STOC*, pages 805–816. ACM Press, May 2012.

[AR17a] Shweta Agrawal and Alon Rosen. Functional encryption for bounded collusions, revisited. In *TCC*, pages 173–205, 2017.

[AR17b] Shweta Agrawal and Alon Rosen. Functional encryption for bounded collusions, revisited. In Yael Kalai and Leonid Reyzin, editors, *TCC 2017, Part I*, volume 10677 of *LNCS*, pages 173–205. Springer, Heidelberg, November 2017.

[AS17] Prabhanjan Ananth and Amit Sahai. Projective arithmetic functional encryption and indistinguishability obfuscation from degree-5 multilinear maps. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part I*, volume 10210 of *LNCS*, pages 152–181. Springer, Heidelberg, April / May 2017.

[BCFG17] Carmen Elisabetta Zaira Baltico, Dario Catalano, Dario Fiore, and Romain Gay. Practical functional encryption for quadratic functions with applications to predicate encryption. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part I*, volume 10401 of *LNCS*, pages 67–98. Springer, Heidelberg, August 2017.

[BDGM20] Zvika Brakerski, Nico Dottling, Sanjam Garg, and Guilio Malavolta. Candidate io from homomorphic encryption schemes. In *EUROCRYPT*, 2020.

[BF01] Dan Boneh and Matthew K. Franklin. Identity-based encryption from the Weil pairing. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 213–229. Springer, Heidelberg, August 2001.

[BFM14] Christina Brzuska, Pooya Farshim, and Arno Mittelbach. Indistinguishability obfuscation and UCEs: The case of computationally unpredictable sources. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part I*, volume 8616 of *LNCS*, pages 188–205. Springer, Heidelberg, August 2014.

[BGG+18] Dan Boneh, Rosario Gennaro, Steven Goldfeder, Aayush Jain, Sam Kim, Peter M. R. Rasmussen, and Amit Sahai. Threshold cryptosystems from threshold fully homomorphic encryption. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part I*, volume 10991 of *Lecture Notes in Computer Science*, pages 565–596. Springer, 2018.

[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 Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 1–18. Springer, Heidelberg, August 2001.

[BGK+14] Boaz Barak, Sanjam Garg, Yael Tauman Kalai, Omer Paneth, and Amit Sahai. Protecting obfuscation against algebraic attacks. In Phong Q. Nguyen and Elisabeth Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 221–238. Springer, Heidelberg, May 2014.

[BGPW16] Johannes A. Buchmann, Florian Göpfert, Rachel Player, and Thomas Wunderer. On the hardness of LWE with binary error: Revisiting the hybrid lattice-reduction and meet-in-the-middle attack. In David Pointcheval, Abderrahmane Nitaj, and Tajjeeddine Rachidi, editors, *AFRICACRYPT 16*, volume 9646 of *LNCS*, pages 24–43. Springer, Heidelberg, April 2016.

[BGV12] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. In *Innovations in Theoretical Computer Science 2012, Cambridge, MA, USA, January 8-10, 2012*, pages 309–325, 2012.

[BIJ+20] James Bartusek, Yuval Ishai, Aayush Jain, Fermi Ma, Amit Sahai, and Mark Zhandry. Affine determinant programs: A framework for obfuscation and witness encryption. In Thomas Vidick, editor, *11th Innovations in Theoretical Computer Science Conference, ITCS 2020, January 12-14, 2020, Seattle, Washington, USA*, volume 151 of *LIPIcs*, pages 82:1–82:39. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

[BMSZ16] Saikrishna Badrinarayanan, Eric Miles, Amit Sahai, and Mark Zhandry. Post-zeroizing obfuscation: New mathematical tools, and the case of evasive circuits. In *Advances in Cryptology - EUROCRYPT*, pages 764–791, 2016.

[BNPW16] Nir Bitansky, Ryo Nishimaki, Alain Passelègue, and Daniel Wichs. From cryptomania to obfustopia through secret-key functional encryption. In Martin Hirt and Adam D. Smith, editors, *TCC 2016-B, Part II*, volume 9986 of *LNCS*, pages 391–418. Springer, Heidelberg, October / November 2016.

[BPR15] Nir Bitansky, Omer Paneth, and Alon Rosen. On the cryptographic hardness of finding a Nash equilibrium. In Venkatesan Guruswami, editor, *56th FOCS*, pages 1480–1498. IEEE Computer Society Press, October 2015.

[BQ12] Andrej Bogdanov and Youming Qiao. On the security of goldreich's one-way function. *Comput. Complex.*, 21(1):83–127, 2012.

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

[BSW11] Dan Boneh, Amit Sahai, and Brent Waters. Functional encryption: Definitions and challenges. In Yuval Ishai, editor, *TCC 2011*, volume 6597 of *LNCS*, pages 253–273. Springer, Heidelberg, March 2011.

[BV11] Zvika Brakerski and Vinod Vaikuntanathan. Fully homomorphic encryption from ring-LWE and security for key dependent messages. In Phillip Rogaway, editor, *CRYPTO 2011*, volume 6841 of *LNCS*, pages 505–524. Springer, Heidelberg, August 2011.

[BV15] Nir Bitansky and Vinod Vaikuntanathan. Indistinguishability obfuscation from functional encryption. In Venkatesan Guruswami, editor, *56th FOCS*, pages 171–190. IEEE Computer Society Press, October 2015.

[CDM+18] Geoffroy Couteau, Aurélien Dupin, Pierrick Méaux, Mélissa Rossi, and Yann Rotella. On the concrete security of Goldreich's pseudorandom generator. In Thomas Peyrin and Steven Galbraith, editors, *ASIACRYPT 2018, Part II*, volume 11273 of *LNCS*, pages 96–124. Springer, Heidelberg, December 2018.

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

[CSA20] Mehdi Tibouchi Chao Sun and Masayuki Abe. Revisiting the hardness of binary error lwe. Cryptology ePrint Archive, Report 2020/666, 2020. `https://eprint.iacr.org/2020/666`.

[CTA19] Sun Caho, Mehdi Tibouchi, and Masayuki Abe. Sample-time trade-off for the arorage attack on binary lwe. *Symposium on Cryptography and Information Theory*, 2019.

[CVW18] Yilei Chen, Vinod Vaikuntanathan, and Hoeteck Wee. GGH15 beyond permutation branching programs: Proofs, attacks, and candidates. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part II*, volume 10992 of *LNCS*, pages 577–607. Springer, Heidelberg, August 2018.

[DHRW16] Yevgeniy Dodis, Shai Halevi, Ron D. Rothblum, and Daniel Wichs. Spooky encryption and its applications. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part III*, volume 9816 of *LNCS*, pages 93–122. Springer, Heidelberg, August 2016.

[Gay20] Romain Gay. A new paradigm for public-key functional encryption for degree-2 polynomials. In *PKC 2020, Part I*, LNCS, pages 95–120. Springer, Heidelberg, 2020.

[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 Phong Q. Nguyen and Elisabeth Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 578–602. Springer, Heidelberg, May 2014.

[GGH+13] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *54th FOCS*, pages 40–49. IEEE Computer Society Press, October 2013.

[GJK18] Craig Gentry, Charanjit S. Jutla, and Daniel Kane. Obfuscation using tensor products. *Electronic Colloquium on Computational Complexity (ECCC)*, 25:149, 2018.

[GKP+13] Shafi Goldwasser, Yael Tauman Kalai, Raluca A. Popa, Vinod Vaikuntanathan, and Nickolai Zeldovich. Reusable garbled circuits and succinct functional encryption. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *45th ACM STOC*, pages 555–564. ACM Press, June 2013.

[GKR08] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. One-time programs. In David Wagner, editor, *CRYPTO 2008*, volume 5157 of *LNCS*, pages 39–56. Springer, Heidelberg, August 2008.

[GLSW14] Craig Gentry, Allison B. Lewko, Amit Sahai, and Brent Waters. Indistinguishability obfuscation from the multilinear subgroup elimination assumption. *IACR Cryptology ePrint Archive*, 2014:309, 2014.

[GMM+16] Sanjam Garg, Eric Miles, Pratyay Mukherjee, Amit Sahai, Akshayaram Srinivasan, and Mark Zhandry. Secure obfuscation in a weak multilinear map model. In Martin Hirt and Adam D. Smith, editors, *TCC 2016-B, Part II*, volume 9986 of *LNCS*, pages 241–268. Springer, Heidelberg, October / November 2016.

[Gol00] Oded Goldreich. Candidate one-way functions based on expander graphs. *Electronic Colloquium on Computational Complexity (ECCC)*, 7(90), 2000.

[GPS16] Sanjam Garg, Omkant Pandey, and Akshayaram Srinivasan. Revisiting the cryptographic hardness of finding a nash equilibrium. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part II*, volume 9815 of *LNCS*, pages 579–604. Springer, Heidelberg, August 2016.

[GVW12a] Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Functional encryption with bounded collusions via multi-party computation. In *Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings*, pages 162–179, 2012.

[GVW12b] Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Functional encryption with bounded collusions via multi-party computation. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 162–179. Springer, Heidelberg, August 2012.

[GVW15] Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Predicate encryption for circuits from LWE. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part II*, volume 9216 of *LNCS*, pages 503–523. Springer, Heidelberg, August 2015.

[HJK+16] Dennis Hofheinz, Tibor Jager, Dakshita Khurana, Amit Sahai, Brent Waters, and Mark Zhandry. How to generate and use universal samplers. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *ASIACRYPT 2016, Part II*, volume 10032 of *LNCS*, pages 715–744. Springer, Heidelberg, December 2016.

[HSW13] Susan Hohenberger, Amit Sahai, and Brent Waters. Full domain hash from (leveled) multilinear maps and identity-based aggregate signatures. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part I*, volume 8042 of *LNCS*, pages 494–512. Springer, Heidelberg, August 2013.

[JLMS19] Aayush Jain, Huijia Lin, Christian Matt, and Amit Sahai. How to leverage hardness of constant-degree expanding polynomials overa $\mathbb{R}$ to build $i\mathcal{O}$. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part I*, volume 11476 of *LNCS*, pages 251–281. Springer, Heidelberg, May 2019.

[JLS20] Aayush Jain, Huijia Lin, and Amit Sahai. Indistinguishability obfuscation from well-founded assumptions. Cryptology ePrint Archive, Report 2020/1003, 2020. `https://eprint.iacr.org/2020/1003`.

[Jou00] Antoine Joux. A one round protocol for tripartite diffie-hellman. In Wieb Bosma, editor, *Algorithmic Number Theory, 4th International Symposium, ANTS-IV, Leiden, The Netherlands, July 2-7, 2000, Proceedings*, volume 1838 of *Lecture Notes in Computer Science*, pages 385–394. Springer, 2000.

[KLW15] Venkata Koppula, Allison Bishop Lewko, and Brent Waters. Indistinguishability obfuscation for turing machines with unbounded memory. In *STOC*, 2015.

[KMOW17] Pravesh K. Kothari, Ryuhei Mori, Ryan O'Donnell, and David Witmer. Sum of squares lower bounds for refuting any CSP. In Hamed Hatami, Pierre McKenzie, and Valerie King, editors, *49th ACM STOC*, pages 132–145. ACM Press, June 2017.

[KNT18] Fuyuki Kitagawa, Ryo Nishimaki, and Keisuke Tanaka. Obfustopia built on secret-key functional encryption. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part II*, volume 10821 of *LNCS*, pages 603–648. Springer, Heidelberg, April / May 2018.

[Las01] Jean B. Lasserre. New positive semidefinite relaxations for nonconvex quadratic programs. In *Advances in convex analysis and global optimization (Pythagorion, 2000)*, volume 54 of *Nonconvex Optim. Appl.*, pages 319–331. Kluwer Acad. Publ., Dordrecht, 2001.

[Lin16] Huijia Lin. Indistinguishability obfuscation from constant-degree graded encoding schemes. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part I*, volume 9665 of *LNCS*, pages 28–57. Springer, Heidelberg, May 2016.

[Lin17] Huijia Lin. Indistinguishability obfuscation from SXDH on 5-linear maps and locality-5 PRGs. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part I*, volume 10401 of *LNCS*, pages 599–629. Springer, Heidelberg, August 2017.

[LM18] Huijia Lin and Christian Matt. Pseudo flawed-smudging generators and their application to indistinguishability obfuscation. *IACR Cryptology ePrint Archive*, 2018:646, 2018.

[LT17] Huijia Lin and Stefano Tessaro. Indistinguishability obfuscation from trilinear maps and block-wise local PRGs. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part I*, volume 10401 of *LNCS*, pages 630–660. Springer, Heidelberg, August 2017.

[LV16] Huijia Lin and Vinod Vaikuntanathan. Indistinguishability obfuscation from DDH-like assumptions on constant-degree graded encodings. In Irit Dinur, editor, *57th FOCS*, pages 11–20. IEEE Computer Society Press, October 2016.

[LV17] Alex Lombardi and Vinod Vaikuntanathan. Minimizing the complexity of goldreich's pseudorandom generator. *IACR Cryptology ePrint Archive*, 2017:277, 2017.

[MP12] Daniele Micciancio and Chris Peikert. Trapdoors for lattices: Simpler, tighter, faster, smaller. In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 700–718. Springer, Heidelberg, April 2012.

[MP13] Daniele Micciancio and Chris Peikert. Hardness of SIS and LWE with small parameters. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part I*, volume 8042 of *LNCS*, pages 21–39. Springer, Heidelberg, August 2013.

[MST03] Elchanan Mossel, Amir Shpilka, and Luca Trevisan. On e-biased generators in NC0. In *44th FOCS*, pages 136–145. IEEE Computer Society Press, October 2003.

[MW16] Pratyay Mukherjee and Daniel Wichs. Two round multiparty computation via multi-key FHE. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 735–763. Springer, Heidelberg, May 2016.

[Nes00] Yurii Nesterov. Squared functional systems and optimization problems. In *High performance optimization*, volume 33 of *Appl. Optim.*, pages 405–440. Kluwer Acad. Publ., Dordrecht, 2000.

[O'N10] Adam O'Neill. Definitional issues in functional encryption. *IACR Cryptology ePrint Archive*, 2010:556, 2010.

[OW14] Ryan O'Donnell and David Witmer. Goldreich's PRG: evidence for near-optimal polynomial stretch. In *IEEE 29th Conference on Computational Complexity, CCC 2014, Vancouver, BC, Canada, June 11-13, 2014*, pages 1–12. IEEE Computer Society, 2014.

[Par00] Pablo A Parrilo. *Structured semidefinite programs and semialgebraic geometry methods in robustness and optimization*. PhD thesis, Citeseer, 2000.

[PST14] Rafael Pass, Karn Seth, and Sidharth Telang. Indistinguishability obfuscation from semantically-secure multilinear encodings. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part I*, volume 8616 of *LNCS*, pages 500–517. Springer, Heidelberg, August 2014.

[Reg05] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In *STOC*, pages 84–93, 2005.

[Sch94] Claus-Peter Schnorr. Block reduced lattice bases and successive minima. *Comb. Probab. Comput.*, 3:507–522, 1994.

[Sho87] N. Z. Shor. Quadratic optimization problems. *Izv. Akad. Nauk SSSR Tekhn. Kibernet.*, (1):128–139, 222, 1987.

[SS10a] Amit Sahai and Hakan Seyalioglu. Worry-free encryption: functional encryption with public keys. In *Proceedings of the 17th ACM conference on Computer and communications security*, pages 463–472. ACM, 2010.

[SS10b] Amit Sahai and Hakan Seyalioglu. Worry-free encryption: functional encryption with public keys. In Ehab Al-Shaer, Angelos D. Keromytis, and Vitaly Shmatikov, editors, *ACM CCS 2010*, pages 463–472. ACM Press, October 2010.

[Ste] Damien Stehlé. Slides: The lwe problem from lattices to cryptography. *https://summerschool-croatia.cs.ru.nl/2015/Lattice-based%20crypto.pdf.*

[SW05] Amit Sahai and Brent R. Waters. Fuzzy identity-based encryption. In Ronald Cramer, editor, *EUROCRYPT 2005*, volume 3494 of *LNCS*, pages 457–473. Springer, Heidelberg, May 2005.

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

[Wee17] Hoeteck Wee. Attribute-hiding predicate encryption in bilinear groups, revisited. In Yael Kalai and Leonid Reyzin, editors, *TCC 2017, Part I*, volume 10677 of *LNCS*, pages 206–233. Springer, Heidelberg, November 2017.

[Wee20a] Hoeteck Wee. Functional encryption for quadratic functions from k-lin, revisited. In *TCC 2020, Part I*, LNCS, pages 210–228. Springer, Heidelberg, March 2020.

[Wee20b] Hoeteck Wee. Functional encryption for quadratic functions from k-lin, revisited. In Rafael Pass and Krzysztof Pietrzak, editors, *Theory of Cryptography - 18th International Conference, TCC 2020, Durham, NC, USA, November 16-19, 2020, Proceedings, Part I*, volume 12550 of *Lecture Notes in Computer Science*, pages 210–228. Springer, 2020.

A Cryptanalysis of Our Assumption

A.1 A Survey of the PRG Candidates

We consider Goldreich PRG candidates [Gol00]. We recall the definition of a hypergraph first.

Definition A.1. *We define an $(n, m, d)-$hypergraph H to be a hypergraph with n vertices and m hyperedges of cardinality d. Each hyperedge σ_i for $i \in [m]$ is of the form $\sigma_i = \{\sigma_{i,1}, \ldots, \sigma_{i,d}\}$ where each $\sigma_{i,j_1} \in [n]$ is distinct from $\sigma_{i,j_2} \in [n]$ for every $i \in [m]$ and $j_1 \neq j_2$. Also, we assume that each σ_i is an ordered set.*

We now define Goldreich PRG candidates.

Definition A.2. *Goldreich's candidate d-local PRG $\mathsf{G}_{H,\mathsf{P}}$ forms a family of local PRG candidates where $\mathsf{G}_{H,\mathsf{P}} : \{0,1\}^n \to \{0,1\}^m$ is parameterized by an $(n, m, d)-$hypergraph $H = (\sigma_1, \ldots, \sigma_m)$ and a boolean predicate $\mathsf{P} : \{0,1\}^d \to \{0,1\}$. The functionality is defined as follows: On input $\boldsymbol{x} \in \{0,1\}^n$, $\mathsf{G}_{H,\mathsf{P}}$ return m-bit strings: $(\mathsf{P}(x_{\sigma_{1,1}}, \ldots, x_{\sigma_{1,d}}), \ldots, \mathsf{P}(x_{\sigma_{m,1}}, \ldots, x_{\sigma_{m,d}}))$.*

Typically P is some predicate satisfying some nice properties, d is a constant integer greater than equal to 5, and H is a randomly chosen graph from some distribution. The security should hold with high probability over the choice of this graph.

Coming back to our assumption, intuitively, our assumption suggests that as long as other parameters are chosen appropriately, any Goldreich PRG predicate of constant degree d admitting a stretch of $\Omega(n^{\frac{1}{2}\cdot\lceil\frac{d}{2}\rceil+c})$ for any constant $c > 0$ can potentially form a nice choice to instantiate our assumption. Traditionally Goldreich's PRG has been a subject

of extensive study (For example, see [Gol00, MST03, ABR12, BQ12, App12, OW14, AL16, CDM+18].). The standard complexity measure for a Goldreich's PRG is locality of the predicate (and not the $\mathbb{Z}-$degree.). Locality of the predicate is the number of bits that the predicate takes as input. Since the predicate in a Goldreich PRG is a boolean function, the locality of the predicate forms an upper bound on the $\mathbb{Z}$-degree of the predicate. We now survey some known results below and we will remark about both locality and $\mathbb{Z}-$degree of the predicate. Analysis of the PRG predicates in literature has focused mainly, on the following broad classes of attacks:

- $\mathbb{F}_2$ linear bias distinguishing attacks.
- Attacks from optimization literature such as (e.g. SoS based SDP algorithms.).
- Algebraic attacks that include, e.g. Gröbner Basis Attacks.
- Guess and Determine Attacks.

It is known from the work of [MST03], that in order to construct a PRG with polynomial stretch the minimum locality needs to be 5. For such a locality, [OW14] proved an optimal stretch of $m(n) = n^{1.5-\epsilon}$ for the Goldreich PRG instantiated with the TSA predicate[4], for any constant $\epsilon > 0$, against subexponential SDP adversaries and $\mathbb{F}_2$ linear bias adversaries.

This understanding can be generalized.

SoS **Attacks.** In fact for attacks relying on Semi-Definite Programming (SDP), there is a very powerful infrastructure to prove systematic lower bounds. This is captured by the *sum-of-squares* (SoS) hierarchy [Sho87, Par00, Nes00, Las01]. It was proven in [KMOW17] that the Goldreich PRG with a stretch $m(n) = n^{1+(\frac{k}{2}-1)(1-\delta)}$ for some constant $\delta > 0$, when instantiated using a random hypergraph and a predicate P that is $k-$wise independent[5], will require an SoS program of level $O(n^\delta)$ for deriving refutations. This translates (very roughly) to an SDP that requires $2^{O(n^\delta)}$ time to solve. This shows that for the TSA predicate with stretch of $n^{1.5-c}$, the SDP approach will take at least $2^{O(n^{2c})}$ time perform refutations/inversion.

$\mathbb{F}_2$ **Linear Bias.** These attacks are distinguishing attacks. $\mathbb{F}_2$ linear bias security consists of proving the following. For outputs $y_1, \ldots, y_m$ of the PRG, it requires that for every non-empty set $S \subseteq [n]$, it holds that $\left| \mathbb{E}[\oplus_{i \in S} y_i] - 0.5 \right| \leq 2^{-n^\epsilon}$ for some constant $\epsilon > 0$. Usually this is a very hard property to prove in general. In fact, we only have sound analysis of very few predicates [MST03, ABR12, OW14, AL16]. The analysis in [AL16] is the first incident where a general degree d of the predicate is considered. Unfortunately, the analysis there can't be applied in our case because the parameters they achieve are not good enough for our setting. Unless a theorem already exists, we won't be discussing about these attacks for most of our candidates.

[4]Recall, $\mathsf{TSA}(x_1, \ldots, x_5) = x_1 \oplus x_2 \oplus x_3 \oplus \mathsf{AND}(x_4, x_5)$.

[5]A predicate is $k-$wise independent if for any set S of size at most $k-1$, $\mathbb{E}[P(x_1, \ldots, x_d) \oplus_{i \in S} x_i] = 0.5$.

Algebraic Attacks / Guess and Determine Attacks. Algebraic attacks consists of resolution style attacks where some equations are set up and then they are manipulated until a search or refutation is made. This class of attacks capture the Gröbner Basis Attacks. In order to avoid the algebraic attacks with the stretch $m(n) = n^s$, as outlined by [AL16], the predicate should have a rational degree [6] greater than s. The reason for that is that, if the rational degree is lower than s, then the following happens. Write $\mathsf{P} \cdot \mathsf{Q} = \mathsf{R}$ where Q and R are degree $e < s$ functions. Given samples $(y_1, \ldots, y_m)$ one can write $y_i \cdot \mathsf{Q}(\boldsymbol{x}_{S_i}) = \mathsf{R}(\boldsymbol{x}_{S_i})$ where S_i is the corresponding indices on which the predicate P was applied to obtain y_i. Note that these are m degree e equations. This system can be linearized if $s > e$. In [AL16], the authors also prove lower bounds for subexponential algorithms in this model but unfortunately they are too weak to be applied here. However, in a very interesting work [CDM+18], this attack was further improved where the authors considered rational degrees of predicates obtained by fixing some bits of the input called the bit-fixing algebraic immunity (hence the name *Guess and Determine*.). Thereby, under reasonable heuristic assumptions fine-tuned trade-offs of stretch vs running time were obtained. Refer to Proposition 5, 7 and 8 in [CDM+18] for details. The paper is also an excellent source on the concrete security of various candidates and a survey of state-of-the-art attacks. For our candidates, we estimate running times of these algorithms by relying on the theorems from this work. All known attacks for our candidates and required parameters require subexponential time. We discuss the state of some of the major known algorithms and how they fare against our candidates in Table 1.

We now discuss our candidates below and how each of the attacks discussed above fare for these candidates.

A.2 The $\mathsf{XORMAJ}_{\ell,\ell}$ Predicate

As suggested earlier, for a general degree, there is a gap between provable security against the classes of attacks discussed above and actual attacks known in practice. While for a general degree d, the best known analysis in [AL16] only constructs a PRG predicate that has a provable stretch [7] of $\Omega(n^{d/38})$. As pointed out it in Corollary 2, and Proposition 8 in [CDM+18], any Goldreich PRG instantiated with a predicate of the form (e.g. the $\mathsf{XOR}_\ell\mathsf{MAJ}_k$ predicates.)

$$\mathsf{P}(x_1 \ldots, x_{\ell+k}) = \oplus_{i\in[\ell]} x_i \oplus g(x_{\ell+1}, \ldots, x_{\ell+k}).$$

for a non-linear balanced predicate g of locality k, can be broken in polynomial time (under a heuristic assumption) if the stretch of the PRG is more than $\widetilde{O}(n^{\lceil \frac{k}{2} \rceil + 1})$. The predicate above if g is balanced, is $(\ell+1)-$wise independent. Thus, in light of these attacks and the SDP atttacks, to design a predicate of this form in general, one needs $\frac{\ell+1}{2} > \frac{1}{2} \cdot \lceil \frac{k+\ell}{2} \rceil$, because of the SDP condition, and $\lceil \frac{k}{2} \rceil + 1 > \frac{1}{2} \cdot \lceil \frac{k+\ell}{2} \rceil$ because of the attacks in [CDM+18]. This leaves us with a tight margin to develop predicates in this manner. One might choose

[6] Recall that the rational degree of P is the minimum degree e such that there exist degree e predicates Q and R such that $PQ = R$. Rational Degree is also known as algebraic immunity.

[7] Actually the result holds for locality.

$k = \ell$, where ℓ is odd. Then, in the first equation $\frac{\ell+1}{2} > \frac{\ell}{2}$ and in the second equation, $\frac{\ell+3}{2} > \frac{\ell}{2}$. Thus, for an odd $\ell \geq 3$ define:

$$\mathsf{XORMAJ}_{\ell,\ell}(x_1 \ldots, x_{2\ell}) = \oplus_{i\in[\ell]} x_i \oplus \mathsf{MAJ}(x_{\ell+1}, \ldots, x_{2\ell}).$$

This predicate above has been widely studied, and has been regarded as the gold standard PRG predicate owing to the fact that Majority has the optimal rational degree [AL16].

SoS **Attacks.** We consider a stretch of $n^{\frac{\ell+1}{2}-c}$ for some constant $c > 0$. Under such circumstances we can show an SoS lower bound relying on the result of [KMOW17] against subexponential sized SoS programs. The exact parameters are computed in Table 1.

Algebraic Attacks. Unfortunately, we can't use the theorems in [AL16] to argue provable security against such attacks, we show that these as well as the improved attacks in [CDM+18] approximately take subexponential time for our parameter setting. The exact parameters are computed in Table 1. In the table we rely on Proposition 5, 7 and 8 in [CDM+18] to populate the parameters.

A.3 Low-Degree High-Locality Predicates

As pointed out in the previous section, in general, we just have small room of parameters to build predicates with the stretch $n^{\frac{\ell}{4}+\epsilon}$ where ℓ is the locality in the way described above.

That points us to the following issue. Much of the research has been done in optimizing locality of the PRG predicates vs the stretch. However, in this work, we actually do not care much about the locality. For us, it is the degree of the predicate of the integers that is crucial. This allows us to design clever predicates that has much lower degree than the locality.

For example, consider the predicate proposed by Lombardi and Vaikunthanathan [LV17] that has a locality of 5, but a degree of just 3!

$$\mathsf{TSPA}(x_1, x_2, x_3, x_4, x_5) = x_1 \oplus x_2 \oplus x_3 \oplus ((x_2 \oplus x_4) \wedge (x_3 \oplus x_5)).$$

At first sight, it does not appear to have a degree of 3, but on careful examination we can indeed show this. We also extend this observation and design a family of predicates that have much lower $\mathbb{Z}$ degree than the locality. We now discuss the status of known attacks for this particular predicate.

- SoS **Attacks.** Since the predicate is $3-$wise independent, relying on the result of [KMOW17] it can be shown that for a stretch of $m(n) \leq n^{1.5-c}$ for any constant $c \in (0, 0.5)$, the predicate provably resists attacks via the *sum-of-squares* paradigm running in time $O(2^{n^{2c}})$.

- **Linear Bias Attacks.** In [LV17] it was proven that for a stretch of $n^{1.25-c}$ for any $c > 0$, the predicate provably resists linear bias distinguishing attacks relying on the dichotomy theorem of [ABR12]. Also, authors conjecture, that for this candidate by a tighter analysis even a stretch of $n^{1.5-c}$ should be possible against linear bias attacks.

- **Algebraic and [CDM+18] Style Attacks.** First observe that the rational degree of TSA and TSPA is the same because the variables are just related by an invertible linear transformation. Namely,

$$\mathsf{TSPA}(x_1, \ldots, x_5) = \mathsf{TSA}(x_1, x_2, x_3, x_4 \oplus x_2, x_3 \oplus x_5).$$

  Thus many of the ideas used to analyze TSA can be applied here. We work out the running time of the known attacks as a function of stretch in Table 1 for these attacks.

Next, we consider the following instantiation inspired by the TSPA predicate above. We suggest a general approach using which we construct a predicate of locality $2 \cdot k + 1$, and a $\mathbb{Z}-$degree of just $k + 1$ for any constant integer $k > 0$. The predicate additionally satisfies $(k + 1)-$wise independence. Further, the non-linear part will have an $\mathbb{F}_2$ degree of k. This allows us to enlarge the margin in parameters for constructing useful predicates as discussed above. Consider g, a non-linear boolean function of $\mathbb{F}_2$ degree k. Then, the predicate is simply:

$$\mathsf{P}_g(x_1 \ldots, x_{2k+1}) = \oplus_{i \in [k+1]} x_i \oplus g(x_{k+2} \oplus x_2, \ldots, x_{2k+1} \oplus x_{k+1}).$$

Put it simply, this can also be written in the template above:

$$\mathsf{P}_g(x_1 \ldots, x_{2k+1}) = x_1 \oplus g'(x_2, \ldots x_{2k+1}),$$

where,

$$g'(x_2 \ldots, x_{2k+1}) = x_2 \oplus \ldots \oplus x_{k+1} \oplus g(x_{k+2} \oplus x_2, \ldots, x_{k+1} \oplus x_{2k+1}).$$

Now we argue $(k+1)-$wise independence. The predicate above is $(k+1)-$wise independent.

The reason for that is that in Fourier notation[8]:

$$\widehat{\mathsf{P}_g}(X_1 \ldots, X_{2k+1}) = \Pi_{i \in [k+1]} X_i \cdot g(X_{k+2} \cdot X_2, \ldots, X_{2k+1} \cdot X_{k+1}).$$

[8]Recall that for any boolean function $f : \{0,1\}^n \to \{0,1\}$, $f(x_1, \ldots, x_n)$, the fourier expansion of f, denoted by $\widehat{f} : \{-1,+1\}^n \to \{-1,+1\}$, is related as:

$$\widehat{f}(X_1, \ldots, X_n) = \sum_{S \subseteq [n]} \widehat{f}_S \cdot \chi_S(X_1, \ldots, X_n).$$

Here $\widehat{f}(X_1, \ldots, X_n) = 1 - 2 \cdot f(x_1, \ldots, x_n)$ and each $X_i = 1 - 2 \cdot x_i$. For any set S, $\chi_S(X_1, \ldots, X_n) = \prod_{i \in S} X_i$.

Also observe that in the Fourier expansion:

$$\widehat{g}(Y_1 \ldots, Y_k) = \sum_{S \subseteq [k]} \widehat{g}_S \chi_S(Y_1, \ldots, Y_k).$$

We substitute $Y_i = X_{i+1} \cdot X_{k+i+1}$. Thus, we get:

$$\widehat{\mathsf{P}_g}(X_1 \ldots, X_{2k+1}) = \Pi_{i \in [k+1]} X_i \cdot \sum_{S \subset [k]} \widehat{g}_S \chi_S(X_2 \cdot X_{k+2}, \ldots, X_{2k+1} \cdot X_{k+1}).$$

Thus, the Fourier expansion of P_g is a homogeneous polynomial of degree $k+1$. Hence, the predicate is $(k+1)-$wise independent. From the above, it is also clear that $\mathbb{Z}$ degree of P_g is also $k+1$. Infact, TSPA is obtained as a special case of this compiler where g is just the AND function. For an odd $k \geq 3$, we consider $\mathsf{P}_{\mathsf{MAJ}_k}$ as one of our candidate. For this candidate, consider:

- SoS **Attacks.** Since the predicate is $(k+1)-$wise independent, relying on the result of [KMOW17] it can be shown that for a stretch of $m(n) \leq n^{\frac{k+1}{2}-c}$ for any constant $c > 0$), the predicate provably resists attacks via the *sum-of-squares* paradigm running in subexponential time.
- **Algebraic and [CDM+18] Style Attacks.** First observe that the rational degree of $\mathsf{P}_{\mathsf{MAJ}_k}$ and $\mathsf{XORMAJ}_{k+1,k}$ is same because the variables are just related by an invertible linear transformation. Thus many of the ideas used to analyze XORMAJ can be applied here. We work out the running time of the known attacks as a function of stretch in Table 1 for these attacks.

A.4 Justifying Security of the Combined Assumptions

We now discuss the plausibility of our assumptions along with the binary LWE leakage part. The first category of attacks we discuss consists of attacks targeting the binary LWE part alone. Since the standalone PRG security has been discussed above, we do not discuss it here. Then we discuss the third category of attacks that consists of algebraic attacks over $\mathbb{F}_p$ that utilize both the LWE samples and the PRG leakage on the error of the LWE samples.

A.4.1 Binary LWE Security

Binary LWE has been a subject of study in quite a few number of works [MP13, ACF+15, AG11, CTA19]. Let n denote the dimension of the secret. While the problem is provably hard, and backed by a security reduction from worst case lattice problems, when the the number of samples $m(n) = n(1 + \Omega(\frac{1}{\log_2 n}))$ [MP13], the problem is easy when $m(n) \geq \Omega(n^2)$, as shown by [AG11]. We work in the regime when the number of samples $m(n) = n^s$ for some $s \in (1, 2)$. Under this regime, there are two kinds of algorithms that are studied.

Gröbner Basis Attacks: Arora-Ge algorithm [AG11] is a special case of a whole family of algebraic algorithms that consider all degree D algebraic constraints implied by the given equations for some large enough D so that the ideal generated by the unique solution can be recovered. Depending on the constraints, the degree defines the running time of the algorithm. The running time of these algorithm typically roughly grows like $n^{O(D)}$. In [CTA19], it was proven that Gröbner basis algorithm require $2^{O(n^\epsilon)}$ time to run assuming that the number of samples are given by $m(n) = n^{2-\epsilon}$ for some $\epsilon > 0$. We will discuss this aspect again when we talk about the third category of attacks.

Lattice Attacks: The only attacks based on lattice reduction techniques that we are aware of apply to LWE more generally, and not just to binary-error LWE. The most relevant attack reduces the LWE instance to a BDD problem and then use the BKZ algorithm [Sch94] to solve it (see, e.g., [Ste] for details). With our setting of parameters, the time complexity of this attack would be $\Omega(2^{n^{0.5+\epsilon-\rho}})$. Because $\rho < 0.5$, this yields at best a subexponential attack.

A.4.2 Algebraic Attacks on the Combined Assumption

A natural approach to combine the information from both the PRG and LWE samples can be to form all equations that one can and then compute the Gröbner basis of the system generated by the equations. Recall a typical instance of our assumption contains:

- LWE samples $\{\boldsymbol{a}_i, b_i = \langle \boldsymbol{a}_i, \boldsymbol{s}\rangle + e_i \mod p\}$ for $i \in [n]$. Here, $\boldsymbol{s}$ has dimension $n^{0.5+\epsilon}$ for some $\epsilon > 0$.
- Degree-d PRG evaluations: $\boldsymbol{y} = \mathsf{G}(e_1, \ldots, e_n) = (\mathsf{G}_{n,1}(\boldsymbol{e}), \ldots, \mathsf{G}_{n,m(n)}(\boldsymbol{e}))$ where $m(n) = n^{\lceil \frac{d}{2} \rceil \cdot (0.5+\epsilon)+\rho}$.

This means, that one can form the following equations.

$$(b_i - \langle \boldsymbol{a}_i, \boldsymbol{s}\rangle)^2 = (b_i - \langle \boldsymbol{a}_i, \boldsymbol{s}\rangle) \; \forall i \in [n],$$

$$y_i = \mathsf{G}_{n,i}(b_1 - \langle \boldsymbol{a}_1, \boldsymbol{s}\rangle, \ldots, b_n - \langle \boldsymbol{a}_n, \boldsymbol{s}\rangle) \; \forall i \in [m].$$

Here, the first equation is result of booleanity of the errors e_i. We now consider an example of this case when $d = 3$, and G is the Goldreich PRG instantiated with the TSPA predicate. We set $\epsilon = 0.1$, $\rho = 0.04$ and $m = n^{1.24} = n^{\lceil \frac{3}{2} \rceil \cdot (0.5+\epsilon)+\rho}$. This enforces the dimension to be $n^{0.6} = n^{0.5+\epsilon}$. Thus we have $\ell = m + n$ equations. m of them are degree 3 equations and n of them are degree 2. Let us denote these equations as $\{q_i(\boldsymbol{s}) = 0\}_{i\in[\ell]}$. A quick and dirty way to approximately gauge the performance of Gröbner basis algorithm is to fix a degree D, and then collect all equations of the form:

$$h(\boldsymbol{s}) \cdot q_i(\boldsymbol{s}) = 0,$$

for all monomials h of degree upto $D - \deg(q_i)$. Finally, if degree D is large enough, and there exists a unique solution, there will exist a D at which point, we can perform gaussian elimination in $n^{0.6 \cdot O(D)}$ variables (variables corresponding to all monomials of degree less than or equal to D generated by $\boldsymbol{s}$) to recover the secret $\boldsymbol{s}$.

For this strategy to succeed we want that the number of monomials of degree less than or equal to D in $\boldsymbol{s}$ to be lesser than the number of equations formed. This happens when:

$$n \cdot \binom{n^{0.6} + D - 2}{D - 2} + n^{1.24} \cdot \binom{n^{0.6} + D - 3}{D - 3} \geq \binom{n^{0.6} + D}{D}.$$

Which means that $D \geq n^{0.1}$. We can also do a similar analysis for a general degree d, which will require:

$$n \cdot \binom{n^{0.5+\epsilon} + D - 2}{D - 2} + m(n) \cdot \binom{n^{0.5+\epsilon} + D - d}{D - d} \geq \binom{n^{0.5+\epsilon} + D}{D}.$$

Here, $m = n^{\lceil \frac{d}{2} \rceil \cdot (0.5+\epsilon)+\rho}$. This requires $D \geq O(\min(n^{\epsilon}, n^{\frac{1}{d} \cdot (\lfloor \frac{d}{2} \rfloor - \rho)}))$. In fact, the above approach is really simplified and ignores many subtle issue but gives a lower bound on the actual degree D that should be considered. For a brief discussion about this, please refer [CTA19]. We will use this calculation to denote running times for various predicates under the column GB in Table 1.

A.5 Summary: Our Assumptions

We start with a table of comparison of our three instantiations where we list four kinds of attacks. SoS represent the sum-of-squares attacks applicable only to the PRG part of the instance. BKZ represent the running time obtained by using BKZ algorithm only the binary LWE part of the instance. GB represent an approximation of the running time of the algebraic attacks over $\mathbb{F}_p$ on the combined assumption discussed in the previous section. Finally in the last column we compute the running time for attacks on the PRG predicates using Propositions 5, 7 and 8 in [CDM+18]. We make the following assumption:

Assumption A.1 (TSPA-LWEleak Assumption)**.** *The Goldreich pseudorandom generator construction instantiated with the* TSPA *predicate satisfies* TSPA-LWEleak$_{3,\epsilon,\rho}$ *security for some constants $\epsilon > 0$ and $\rho > 0$.*

Similarly, we make the following assumptions:

Assumption A.2 (XORMAJ$_{\ell,\ell}$-LWEleak Assumption)**.** *The Goldreich pseudorandom generator construction instantiated with the* XORMAJ$_{\ell,\ell}$ *predicate for an odd integer $\ell \geq 3$ satisfies* XORMAJ-LWEleak$_{2 \cdot \ell,\epsilon,\rho}$ *security for some constants $\epsilon > 0$ and $\rho > 0$.*

Assumption A.3 (P$_{\mathsf{MAJ}_k}$-LWEleak Assumption)**.** *The Goldreich pseudorandom generator construction instantiated with the* P$_{\mathsf{MAJ}_k}$ *predicate for an odd integer $k \geq 3$ satisfies* P$_{\mathsf{MAJ}_k}$-LWEleak$_{k+1,\epsilon,\rho}$ *security for some constants $\epsilon > 0$ and $\rho > 0$.*

P	d	m_1	m_2	SoS	BKZ	GB	CDMRR
TSPA	3	$n^{1.45}$	n^{1+c}	$n^{0.10}$	$n^{0.71}$	$n^{0.22}$	$n^{0.4}$
$\mathsf{XORMAJ}_{5,5}$	10	$n^{2.95}$	$n^{2.5+c}$	$n^{0.025}$	$n^{0.582}$	$n^{0.082}$	$n^{0.5125}$
$\mathsf{P}_{\mathsf{MAJ}_5}$	6	$n^{2.95}$	$n^{1.5+c}$	$n^{0.025}$	$n^{0.97}$	$n^{0.48}$	$n^{0.51}$

Table 1: Running time for various known inversion attacks. Above P is a predicate of degree d. m_1 denotes the considered stretch, m_2 is the minimum stretch required in order to construct obfuscation via our assumption. $c > 0$ is arbitrary constant. SoS denotes the attacks known via the Sum-of-Squares paradigm. BKZ denotes the running time of the attacks via the BKZ lattice reduction algorithm. GB denotes the algebraic attacks on the combined assumption based on the Gröbner Basis algorithm. The last column denotes the running time from the subexponential time algorithm in [CDM+18] (Propositions 5,7 and 8). The cells represent $\widetilde{O}(\log_2(\cdot))$ of the running times where we hide logarithmic factors. The value of ρ is chosen to be 0.01, and so the modulus is $p = O(2^{n^{0.01}})$. We set ϵ so that, $m_1 = n^{\lceil \frac{d}{2} \rceil \cdot (0.5+\epsilon)+\rho}$.

B Lattice Preliminaries

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

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

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

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

Let S^m denote the set of vectors in $\mathbb{R}^m$ whose length is 1. The norm of a matrix $\mathbf{R} \in \mathbb{R}^{m \times m}$ is defined to be $\mathsf{sup}_{\boldsymbol{x} \in S^m} ||\mathbf{R}\boldsymbol{x}||$. The LWE problem was introduced by Regev [Reg05], who showed that solving it *on average* is as hard as (quantumly) solving several standard lattice problems *in the worst case*.

Definition B.1 (LWE Assumption)**.** *For an integer $p = p(\mathsf{d}) \geq 2$, and an error distribution $\chi = \chi(\mathsf{d})$ over $\mathbb{Z}_p$, the* Learning With Errors assumption $\mathsf{LWE}_{\mathsf{d},m,p,\chi}$ *holds if it is hard to distinguish between the following pairs of distributions:*

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

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

Gadget matrix. The gadget matrix described below is proposed in [MP12].

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