Functional Encryption for Bounded Collusions, Revisited

Shweta Agrawal* Alon Rosen†

Abstract

We provide a new construction of functional encryption (FE) for circuits in the bounded collusion model. In this model, security of the scheme is guaranteed as long as the number of colluding adversaries can be a-priori bounded by some polynomial q. Our construction supports *arithmetic* circuits as against Boolean circuits, which have been the focus of all prior work. The ciphertext of our scheme is sublinear in the circuit size for the circuit class NC_1 when based on Ring LWE and any constant depth when based on standard LWE. This gives the first constructions of arithmetic reusable garbled circuits.

Additionally, our construction achieves several desirable features:

- Our construction for reusable garbled circuits for NC_1 achieves the optimal "full" simulation based security.
- When generalised to handle Q queries for any fixed polynomial Q, our ciphertext size grows additively with Q^2. Such query dependence on ciphertext size has only been achieved in a weaker security game otherwise [Agr17].
- The ciphertext of our scheme can be divided into a succinct data dependent component and a non-succinct data independent component. This makes it well suited for optimization in an *online-offline model* that allows a majority of the computation to be performed in an offline phase, before the data becomes available.

Security of our scheme may be based on the Learning With Errors assumption (LWE) or its ring variant (Ring-LWE). To achieve our result, we provide new public key and ciphertext evaluation algorithms in the context of functional encryption. These algorithms are general, and may find application elsewhere.

*IIT Madras, India. Email: shweta@iitm.ac.in.

†Efi Arazi School of Computer Science, IDC Herzliya, Israel. Email: alon.rosen@idc.ac.il.

1 Introduction

Functional encryption (FE) [SW05, SW] generalizes public key encryption to allow fine grained access control on encrypted data. In functional encryption, a secret key SK_g corresponds to a function g, and a ciphertext $\mathsf{CT}_{\mathbf{x}}$ corresponds to some input $\mathbf{x}$ from the domain of g. Given SK_g and $\mathsf{CT}_{\mathbf{x}}$, functionality posits that the user may run the decryption procedure to learn the value $g(\mathbf{x})$, while security guarantees that nothing about $\mathbf{x}$ beyond $g(\mathbf{x})$ can be learned.

Recent years have witnessed significant progress towards constructing functional encryption for advanced functionalities [BF01, Coc01, BW06, BW07, GPV08, CHKP10, ABB10, GPSW06, BSW07, KSW08, LOS+10, AFV11, Wat12, GVW13, GGH+13c, GGH+13b, GVW15]. However, for the most general notion of functional encryption – one that allows the evaluation of arbitrary efficient functions and is secure against general adversaries, the only known constructions rely on indistinguishability obfuscation (iO) [GGH+13b] or on the existence of multilinear maps [GGHZ14]. For full-fledged functional encryption, reliance on such strong primitives is not a co-incidence, since functional encryption has been shown to imply indistinguishability obfuscation [AJ15, BV15, AJS15].

Unfortunately, all known candidate multi-linear map constructions [GGH13a, CLT13, GGH15] as well as some candidates of indistinguishability obfuscation have recently been broken [CHL+15, CGH+15, HJ16, CJL, CFL+16, MSZ16]. To support general functionalities and base hardness on standard assumptions, a prudent approach is to consider principled relaxations of the security definition, as studied in [GVW12, GKP+13, GVW15].

The notion of *bounded collusion functional encryption*, inspired from the domain of secure multiparty computation (MPC), was introduced by Gorbunov, Vaikuntanathan and Wee [GVW12]. This notion assumes that the number of colluding adversaries against a scheme can be upper bounded by some polynomial q, which is known at the time of system design. It is important to note that q-bounded security does not impose any restriction on the functionality of FE – in particular, it does not disallow the system from issuing an arbitrary number of keys. It only posits, à la MPC, that security is guaranteed as long as any collusion of attackers obtains at most q keys. Note that multiple independent collusions of size at most q are supported.

The notion of q-bounded FE is appealing – proving security under the assumption that not too many parties are dishonest is widely accepted as reasonable in protocol design. Even in the context of FE, for the special case of Identity Based Encryption (IBE), bounded collusion security has been considered in a number of works [DKXY02, CHH+07, GLW12].

Structure versus Generality. Gorbunov et al. [GVW12] showed that q-bounded FE can be constructed generically from *any* public key encryption (PKE) scheme by leveraging ideas from multiparty computation. Considering that most constructions of FE for general functionalities rely on the existence of sophisticated objects such as multilinear maps or indistinguishability obfuscation, basing a meaningful relaxation of FE on an assumption as generic and mild as PKE is both surprising, and aesthetically appealing. However, this generality comes at the cost of efficiency and useful structural properties. The ciphertext of the scheme is large and grows multiplicatively as $O(q^4)$ to support collusions of size q. Additionally, the entire ciphertext is data dependent, making the scheme unsuitable for several natural applications of FE, as discussed below.

1.1 Our Results

In this work, we provide a new construction of bounded key functional encryption. Our construction makes use of the recently developed Functional Encryption for Linear Functions [ABCP15, ALS16], denoted by LinFE, and combines this with techniques developed in the context of Fully Homomorphic Encryption (FHE)[1] [BV11b, BV11a]. Since LinFE and FHE can be based on LWE/Ring LWE, our construction inherits the same hardness assumption. Our construction offers several advantages:

1. Our construction supports *arithmetic* circuits as against Boolean circuits.
2. The ciphertext of our scheme is succinct for circuits in NC_1 under Ring LWE and any constant depth under standard LWE. This gives the first construction of arithmetic reusable garbled circuits[2].
3. Our construction achieves the optimal "full" simulation based security.
4. When generalised to handle Q queries for any fixed polynomial Q, our ciphertext size grows additively with Q^2 as against [GVW12], where it grows multiplicatively with Q^4.
5. The ciphertext of our scheme can be divided into a succinct data dependent component and a non-succinct data independent component. This makes it well suited for optimization in an *online-offline model* that allows a majority of the computation to be performed in an offline phase, before the data becomes available. The structure of our ciphertext makes it very suitable for *distributed data applications*, as discussed in Appendix B.

1.2 Related Work

The first functional encryption scheme for circuits was provided by Gorbunov, Vaikuntanathan and Wee [GVW12]. Surprisingly, the security of this construction may be based only on the existence of public key encryption. However, the ciphertext size of this construction is large and does not enjoy the online-offline property described above. Specifically, the online component of our scheme significantly outperforms the online component of [GVW12], and depends only on the message size, whereas that of [GVW12] additionally depends on the circuit size and the number of queries. Additionally, our overall ciphertext size grows additively with Q^2 for a collusion bound Q, whereas that of [GVW12] grows multiplicatively with Q^4.

More recently, Agrawal et al. [ALS16] provided a construction for bounded collusion FE. However, their ciphertext size grows as $O(Q^6)$ and does not enjoy the online-offline or decomposability properties that our construction does.

Concurrent and Subsequent Work. Subsequent to our work, Agrawal [Agr17] also constructed Q collusion Functional Encryption with ciphertext size growing additively with $O(Q^2)$. However, this construction only achieves semi-adaptive rather than full security in a weak security game where the attacker must announce all Q queries "in one shot". Additionally, it supports Boolean rather

[1] We emphasise that we do not rely on FHE in a black box way, but rather adapt techniques developed in this domain to our setting.

[2] We note that even single use arithmetic garbled circuits have only been constructed recently [AIK11].

than arithmetic circuits and makes black box use of "heavy machinery" such fully homomorphic encryption and attribute based encryption.

In another recent work, Canetti and Chen [CC17] provide a new construction for single key FE for NC_1 achieving full security. However, to generalise this construction to support Q queries, one must rely on the [GVW12] compiler, which incurs a multiplicative blowup of $O(Q^4)$ in the ciphertext size. For more details about related work, please see Appendix A.

1.3 Techniques

In this section, we describe our techniques. We begin by outlining the approach taken by previous work. [GVW12] begin with a single key FE scheme for circuits [SS10] and generalize this to a q query scheme for NC_1 circuits. This is the most sophisticated part of the construction, and leverages techniques from multiparty computation. Then, the q query FE for NC_1 is bootstrapped to q query FE for all circuits by replacing the circuit in the key by a tuple of low degree polynomials admitted by computational randomized encodings [AIK06].

Recently, Agrawal et al. [ALS16] observe that a different construction for bounded collusion FE can be obtained by replacing [SS10] and its generalisation to q query FE for NC_1, with an FE that computes inner products modulo some prime p. Such a scheme, which we denote by LinFE, was constructed by [ALS16] and computes the following functionality: the encryptor provides a ciphertext $\mathsf{CT}_{\mathbf{x}}$ for some vector $\mathbf{x} \in F_p^\ell$, the key generator provides a key $\mathsf{SK}_{\mathbf{v}}$ for some vector $\mathbf{v} \in F_p^\ell$, and the decryptor, given $\mathsf{CT}_{\mathbf{x}}$ and $\mathsf{SK}_{\mathbf{v}}$ can compute $\langle \mathbf{x}, \mathbf{v} \rangle \mod p$[3]. Since the bootstrapping theorem in [GVW12] only requires FE for degree 3 polynomials, and FE for linear functions trivially implies FE for bounded degree polynomials simply by linearizing the message terms $\mathbf{x}$ and encrypting each monomial $x_i x_j x_k$ separately, LinFE may be used to compute degree 3 polynomials.

Thus, in [ALS16], the challenge of supporting multiplication is "brute-forced" by merely having the encryptor encrypt each monomial separately so that the FE must only support linear functions in order to achieve bounded degree polynomials. This brute force approach has several disadvantages: the ciphertext is not decomposable as the influence of bit x_i cannot be contained to CT_i, is not online-succinct as the entire ciphertext is data dependent, and its size grows as $O(q^6)$. See Appendix A for more details.

Our Approach. In this work, we observe that viewing functional encryption through the lens of fully homomorphic encryption (FHE) enables a more sophisticated application of the Linear FE scheme LinFE, resulting in a bounded collusion FE scheme for circuits that is decomposable, online-succinct as well as achieves ciphertext growth of $O(q^2)$.

In what follows, we focus on FE for quadratic polynomials for ease of exposition. Additionally, here and in the rest of the paper, we present our construction from Ring-LWE rather than standard LWE, for notational convenience and clarity. Our construction can be ported to the standard LWE setting, by performing standard transformations such as replacing ring products by vector tensor products. Details are provided in Appendix C.

Consider the ring LWE based symmetric key FHE scheme of [BV11b]. Recall that the ciphertext of this scheme, as in [Reg09], is structured as (u, c) where $c = u \cdot s + 2 \cdot \mu + x$. Here, s is the symmetric

[3]We note that the FE scheme by Abdalla et al. [ABCP15] also supports linear functions but only over $\mathbb{Z}$, while [ALS16] requires an FE scheme that supports $\mathbb{Z}_p$. Also note the difference from Inner Product *orthogonality testing* schemes [KSW08, AFV11] which test whether $\langle \mathbf{x}, \mathbf{v} \rangle = 0 \mod p$ or not.

key chosen randomly over an appropriate ring R, u is an element chosen by the encryptor randomly over R, x is a message bit and μ is an error term chosen by the encryptor from an appropriate distribution over R. Given secret key s, the decryptor may compute $c - u \cdot s \mod 2$ to recover the bit x.

The main observation in [BV11b] was that if:

$$c_i = u_i \cdot s + 2 \cdot \mu_i + x_i$$
$$c_j = u_j \cdot s + 2 \cdot \mu_j + x_j$$

then the decryption equation can be written as

$$x_i x_j \approx c_i c_j + (u_i u_j)s^2 - (u_j c_i)s - (u_i c_j)s$$

Thus, the 3 tuple $(c_i c_j,\ u_i c_j + u_j c_i,\ u_i u_j)$ is a legitimate level 2 FHE ciphertext, decryptable by the secret key s. [BV11b] observed that it is sufficient to add one ciphertext element per level of the circuit to propagate the computation.

In the context of FE, things are significantly more complex even for quadratic polynomials, since we must return a key that allows the decryptor to learn $x_i x_j$ and nothing else. Hence, providing s to the decryptor is disastrous for FE security. Here we use our first trick: observe that in the above equation, the parenthesis can be shifted so that:

$$x_i x_j \approx c_i c_j + u_i u_j(s^2) - u_j(c_i s) - u_i(c_j s)$$

Now, if we use the Linear FE scheme to encrypt the terms in parenthesis, then we can have the decryptor recover the term $u_i u_j(s^2) - u_j(c_i s) - u_i(c_j s)$. More formally, let $|\mathbf{x}| = w$. Now if,

$$\mathsf{CT} = \mathsf{LinFE.Enc}(s^2, c_1 s, \ldots, c_w s)$$
$$\mathsf{SK}_{ij} = \mathsf{LinFE.KeyGen}(u_i u_j, -0-, u_i, -0-, u_j, -0-)$$

then, $\mathsf{LinFE.Dec}(\mathsf{SK}_{ij}, \mathsf{CT})$ should yield the above term by correctness. Since $c_1, \ldots, c_w$ may be provided directly in the ciphertext, the decryptor may itself compute the term $c_i c_j$. Now, LinFE decryption yields $u_i u_j(s^2) - u_j(c_i s) - u_i(c_j s)$, so the decryptor may recover (approximately) $x_i x_j$ as desired [4].

A bit more abstractly, we observe that a quadratic plaintext $x_i x_j$ can be represented as a quadratic polynomial which is quadratic in *public* terms c_i, c_j, and only *linear* in secret terms $c_i s$. In particular, since the number of secret terms $c_i s$ which must be encrypted is only linear in w, we appear to avoid the quadratic blowup caused by linearization.

This intuition, while appealing, is very misleading. To begin, note that if we permit the decryptor to learn the term $u_i u_j s^2 - u_j c_i s - u_i c_j s$ exactly, then he can recover exact quadratic equations in the secret s, completely breaking the security of the scheme. To handle this, we resort to our second trick: add noise *artificially* to the decryption equation. This takes care of the above attack, but to handle Q queries, we need Q fresh noise terms to be encrypted in the ciphertext. This step introduces the dependence of the ciphertext size on Q. Providing a proof of security requires crossing several additional hurdles. The details of the proof are provided in Section 4.

[4] As in FHE, approximate recovery is enough since the noise can be modded out.

New Public Key and Ciphertext Evaluation Algorithms. We develop new public key and ciphertext evaluation algorithms that enable an evaluator to propagate computation down a circuit using only the public encodings/ciphertext components and public key. Our algorithm allows computing on ciphertexts obliviously of encoded values: note that the only other known construction [BGG+14, GVW15] crucially requires the evaluator at least one of the encoded values x_1, x_2 in the clear in order to compute an encoding of $x_1 \cdot x_2$.

To achieve this, we observe that the quadratic scheme discussed above enables the decryptor to compute encodings of degree two polynomials that have the *same* structure as the message encodings. Hence, addition or multiplication may be performed on level 2 encodings exactly as above to propagate the computation.

In more detail, recall that the level 1 encodings $\mathbf{c}$ of message $\mathbf{x}$ along with level 2 encodings of message $\mathbf{c} \cdot s$ were sufficient to compute encodings of degree two polynomials in $\mathbf{x}$. Hence, at any level k, given an encoding $\mathbf{c}^{k-1}$ of message $\mathbf{y}$ where $\mathbf{y}$ is the output of the circuit at level $k-1$, as well as encodings of $\mathbf{c}^{k-1} \cdot s$, we would be in a position to compute encodings $\mathbf{c}^k$ of level k output of the circuit using the quadratic scheme described in Section 4.

This intuition is complicated by the fact that the encryptor may not provide $\mathbf{c}^{k-1}$ directly while maintaining the succinctness of the ciphertext, but rather may provide succinct *advice*, a set of encodings $\mathcal{C}^{k-1}$ which enables the decryptor to compute $\mathbf{c}^{k-1}$ on the fly. The question then, is whether there exists advice linear in the size of the set $\mathcal{C}^{k-1}$ which suffices to compute the encodings of $(\mathbf{c}^{k-1} \cdot s)$ required to compute $\mathbf{c}^k$ on the fly. We answer this affirmatively. Please see Section 5 for more details.

Organization of the paper. We provide preliminaries in Appendix 2 and recap some properties of linear functional encryption in Section 3. Our bounded collusion functional encryption scheme for quadratic polynomials is described in Section 4. To generalize our method beyond quadratic polynomials, we describe our public key and ciphertext evaluation procedures in Section 5. The succinct single key FE using these procedures is constructed in Section 6. The final scheme for bounded collusion FE for all circuits is described in Appendix 7. We discuss parameters of our construction in Appendix E.

2 Preliminaries

In this section, we define the notation and preliminaries we require for our constructions.

Notation. For any integer $q \geq 2$, we let $\mathbb{Z}_q$ denote the ring of integers modulo q and we represent $\mathbb{Z}_q$ as integers in $(-q/2, q/2]$. We let $\mathbb{Z}_q^{n \times m}$ denote the set of $n \times m$ matrices with entries in $\mathbb{Z}_q$. We use bold capital letters (e.g. $\mathbf{A}$) to denote matrices, bold lowercase letters (e.g. $\mathbf{x}$) to denote vectors that are components of our encryption scheme, and arrows (e.g. $\vec{v}$) to denote vectors that represent attributes or predicates. The notation $\mathbf{A}^\intercal$ denotes the transpose of the matrix $\mathbf{A}$. When we say a matrix defined over $\mathbb{Z}_q$ has *full rank*, we mean that it has full rank modulo each prime factor of q.

If $\mathbf{A}_1$ is an $n \times m$ matrix and $\mathbf{A}_2$ is an $n \times m'$ matrix, then $[\mathbf{A}_1 \| \mathbf{A}_2]$ denotes the $n \times (m + m')$ matrix formed by concatenating $\mathbf{A}_1$ and $\mathbf{A}_2$. If $\mathbf{x}_1$ is a length m vector and $\mathbf{x}_2$ is a length m' vector, then we let $[\mathbf{x}_1 | \mathbf{x}_2]$ denote the length $(m + m')$ vector formed by concatenating $\mathbf{x}_1$ and $\mathbf{x}_2$. However, when doing matrix-vector multiplication we always view vectors as column vectors.

We say a function $f(n)$ is *negligible* if it is $O(n^{-c})$ for all $c > 0$, and we use $\mathrm{negl}(n)$ to denote a negligible function of n. We say $f(n)$ is *polynomial* if it is $O(n^c)$ for some $c > 0$, and we use $\mathrm{poly}(n)$ to denote a polynomial function of n. We say an event occurs with *overwhelming probability* if its probability is $1 - \mathrm{negl}(n)$. The function $\lg x$ is the base 2 logarithm of x. The notation $\lfloor x \rceil$ denotes the nearest integer to x, rounding towards 0 for half-integers.

2.1 Functional Encryption

Let $\mathcal{X} = \{\mathcal{X}_\lambda\}_{\lambda\in\mathbb{N}}$ and $\mathcal{Y} = \{\mathcal{Y}_\lambda\}_{\lambda\in\mathbb{N}}$ denote ensembles where each $\mathcal{X}_\lambda$ and $\mathcal{Y}_\lambda$ is a finite set. Let $\mathcal{G} = \{\mathcal{G}_\lambda\}_{\lambda\in\mathbb{N}}$ denote an ensemble where each $\mathcal{G}_\lambda$ is a finite collection of circuits, and each circuit $g \in \mathcal{G}_\lambda$ takes as input a string $x \in \mathcal{X}_\lambda$ and outputs $g(x) \in \mathcal{Y}_\lambda$.

A functional encryption scheme $\mathcal{F}$ for $\mathcal{G}$ consists of four algorithms $\mathcal{F} = (\mathsf{FE.Setup}, \mathsf{FE.Keygen}, \mathsf{FE.Encrypt}, \mathsf{FE.Decrypt})$ defined as follows.

- $\mathsf{FE.Setup}(1^\lambda)$ is a p.p.t. algorithm takes as input the unary representation of the security parameter and outputs the master public and secret keys $(\mathsf{PK}, \mathsf{MSK})$.
- $\mathsf{FE.Keygen}(\mathsf{MSK}, g)$ is a p.p.t. algorithm that takes as input the master secret key MSK and a circuit $g \in \mathcal{G}_\lambda$ and outputs a corresponding secret key SK_g.
- $\mathsf{FE.Encrypt}(\mathsf{PK}, x)$ is a p.p.t. algorithm that takes as input the master public key PK and an input message $x \in \mathcal{X}_\lambda$ and outputs a ciphertext CT.
- $\mathsf{FE.Decrypt}(\mathsf{SK}_g, \mathsf{CT}_x)$ is a deterministic algorithm that takes as input the secret key SK_g and a ciphertext CT_x and outputs $g(x)$.

Definition 2.1 (Correctness)**.** A functional encryption scheme $\mathcal{F}$ is correct if for all $g \in \mathcal{G}_\lambda$ and all $x \in \mathcal{X}_\lambda$,

$$\Pr\left[\begin{array}{l}(\mathsf{PK}, \mathsf{MSK}) \leftarrow \mathsf{FE.Setup}(1^\lambda);\\ \mathsf{FE.Decrypt}\Big(\mathsf{FE.Keygen}(\mathsf{MSK}, g), \mathsf{FE.Encrypt}(\mathsf{PK}, x)\Big) \neq g(x)\end{array}\right] = \mathrm{negl}(\lambda)$$

where the probability is taken over the coins of $\mathsf{FE.Setup}$, $\mathsf{FE.Keygen}$, and $\mathsf{FE.Encrypt}$.

2.2 Simulation Based Security for Single Key FE

In this section, we define simulation based security for single key FE, as in [GKP+13, Defn 4.1].

Definition 2.2 ($\mathsf{FULL\text{-}SIM}$ Security)**.** Let $\mathcal{F}$ be a functional encryption scheme for a Boolean circuit family $\mathcal{C}$. For every stateful p.p.t. adversary Adv and a stateful p.p.t. simulator Sim, consider the following two experiments:

$\mathsf{Exp}^{\mathsf{real}}_{\mathcal{F},\mathsf{Adv}}(1^\lambda)$:	$\mathsf{Exp}^{\mathsf{ideal}}_{\mathcal{F},\mathrm{Sim}}(1^\lambda)$:
1: $(\mathsf{PK},\mathsf{MSK}) \leftarrow \mathsf{FE.Setup}(1^\lambda)$	1: $(\mathsf{PK},\mathsf{MSK}) \leftarrow \mathsf{FE.Setup}(1^\lambda)$
2: $C \leftarrow \mathsf{Adv}(1^\lambda, \mathsf{PK})$	2: $C \leftarrow \mathsf{Adv}(1^\lambda, \mathsf{PK})$
3: $\mathsf{SK}_C \leftarrow \mathsf{FE.Keygen}(\mathsf{MSK}, C)$	3: $\mathsf{SK}_C \leftarrow \mathsf{FE.Keygen}(\mathsf{MSK}, C)$
4: $\mathbf{x} \leftarrow \mathsf{Adv}(\mathsf{SK}_C)$	4: $\mathbf{x} \leftarrow \mathsf{Adv}(\mathsf{SK}_C)$
5: $\mathsf{CT}_\mathbf{x} \leftarrow \mathsf{FE.Encrypt}(\mathsf{PK}, \mathbf{x})$	5: $\mathsf{CT}_\mathbf{x} \leftarrow \mathrm{Sim}(1^\lambda, 1^{\lvert\mathbf{x}\rvert}, \mathsf{PK}, C, \mathsf{SK}_C, C(\mathbf{x}))$
6: $\alpha \leftarrow \mathsf{Adv}(\mathsf{CT}_\mathbf{x})$	6: $\alpha \leftarrow \mathsf{Adv}(\mathsf{CT}_\mathbf{x})$
7: Output $(\mathbf{x}, \alpha)$	7: Output $(\mathbf{x}, \alpha)$

The functional encryption scheme $\mathcal{F}$ is then said to be FULL-SIM-secure if there is an admissible stateful p.p.t. simulator Sim such that for every stateful p.p.t. adversary Adv, the following two distributions are computationally indistinguishable.

$$\left\{\mathsf{Exp}^{\mathsf{real}}_{\mathcal{F},\mathsf{Adv}}(1^\lambda)\right\}_{\lambda\in\mathbb{N}} \overset{c}{\approx} \left\{\mathsf{Exp}^{\mathsf{ideal}}_{\mathcal{F},\mathrm{Sim}}(1^\lambda)\right\}_{\lambda\in\mathbb{N}}$$

In the bounded collusion variant of the above definition, the adversary is permitted an a-priori fixed Q queries in Step 2, and Q is input to the FE.Setup algorithm. We note that [GVW12] also discusses a stronger variant of the above game, where the attacker can make key queries after seeing the challenge ciphertext, but it was shown by [BSW11] that such a definition is impossible to achieve for even a single key query while permitting an unbounded number of ciphertexts. Therefore, as in [GKP+13], we do not consider this definition here.

2.3 Lattice Preliminaries

An *m-dimensional lattice* Λ is a full-rank discrete subgroup of $\mathbb{R}^m$. A *basis* of Λ is a linearly independent set of vectors whose span is Λ.

Gaussian distributions. Let L be a discrete subset of $\mathbb{Z}^n$. For any vector $\mathbf{c} \in \mathbb{R}^n$ and any positive parameter $\sigma \in \mathbb{R}_{>0}$, let $\rho_{\sigma,\mathbf{c}}(\mathbf{x}) := \mathsf{Exp}\left(-\pi\|\mathbf{x}-\mathbf{c}\|^2/\sigma^2\right)$ be the Gaussian function on $\mathbb{R}^n$ with center $\mathbf{c}$ and parameter σ. Let $\rho_{\sigma,\mathbf{c}}(L) := \sum_{\mathbf{x}\in L}\rho_{\sigma,\mathbf{c}}(\mathbf{x})$ be the discrete integral of $\rho_{\sigma,\mathbf{c}}$ over L, and let $\mathcal{D}_{L,\sigma,\mathbf{c}}$ be the discrete Gaussian distribution over L with center $\mathbf{c}$ and parameter σ. Specifically, for all $\mathbf{y} \in L$, we have $\mathcal{D}_{L,\sigma,\mathbf{c}}(\mathbf{y}) = \frac{\rho_{\sigma,\mathbf{c}}(\mathbf{y})}{\rho_{\sigma,\mathbf{c}}(L)}$. For notational convenience, $\rho_{\sigma,\mathbf{0}}$ and $\mathcal{D}_{L,\sigma,\mathbf{0}}$ are abbreviated as ρ_σ and $\mathcal{D}_{L,\sigma}$, respectively.

The following lemma gives a bound on the length of vectors sampled from a discrete Gaussian.

Lemma 2.3 ([MR07, Lemma 4.4]). *Let Λ be an n-dimensional lattice, let $\mathbf{T}$ be a basis for Λ, and suppose $\sigma \geq \|\mathbf{T}\|_{\mathsf{GS}} \cdot \omega(\sqrt{\log n})$. Then for any $\mathbf{c} \in \mathbb{R}^n$ we have*

$$\Pr\left[\|\mathbf{x}-\mathbf{c}\| > \sigma\sqrt{n} : \mathbf{x} \overset{\mathrm{R}}{\leftarrow} \mathcal{D}_{\Lambda,\sigma,\mathbf{c}}\right] \;\leq\; \mathrm{negl}(n)$$

Lemma 2.4 (Flooding Lemma). *[GKPV10] Let $n \in \mathbb{N}$. For any real $\sigma = \omega(\sqrt{\log n})$, and any $\mathbf{c} \in \mathbb{Z}^n$,*

$$\mathsf{SD}(\mathcal{D}_{\mathbb{Z}^n,\sigma},\ \mathcal{D}_{\mathbb{Z}^n,\sigma,\mathbf{c}}) \leq \|\mathbf{c}\|/\sigma$$

2.4 Hardness Assumptions

Our constructions can be based on the hardness of LWE or Ring LWE, defined below.

Learning With Errors. The *Learning with Errors* problem, or LWE, is the problem of determining a secret vector over $\mathbb{F}_q$ given a polynomial number of "noisy" inner products. The decision variant is to distinguish such samples from random. More formally, we define the (average-case) problem as follows:

Definition 2.5 ([Reg09])**.** Let $n \geq 1$ and $q \geq 2$ be integers, and let χ be a probability distribution on $\mathbb{Z}_q$. For $\mathbf{r} \in \mathbb{Z}_q^n$, let $A_{\mathbf{r},\chi}$ be the probability distribution on $\mathbb{Z}_q^n \times \mathbb{Z}_q$ obtained by choosing a vector $\mathbf{a} \in \mathbb{Z}_q^n$ uniformly at random, choosing $e \in \mathbb{Z}_q$ according to χ, and outputting $(\mathbf{a}, \langle \mathbf{a}, \mathbf{r} \rangle + e)$.

The decision $\mathsf{LWE}_{q,n,\chi}$ problem is: for uniformly random $\mathbf{r} \in \mathbb{Z}_q^n$, given a $\mathrm{poly}(n)$ number of samples that are either (all) from $A_{\mathbf{r},\chi}$ or (all) uniformly random in $\mathbb{Z}_q^n \times \mathbb{Z}_q$, output 0 if the former holds and 1 if the latter holds.

We say the decision-$\mathsf{LWE}_{q,n,\chi}$ problem is *infeasible* if for all polynomial-time algorithms $\mathcal{A}$, the probability that $\mathcal{A}$ solves the decision-LWE problem (over $\mathbf{r}$ and $\mathcal{A}$'s random coins) is negligibly close to $1/2$ as a function of n.

Ring Learning with Errors. Let $R = \mathbb{Z}[x]/(\phi)$ where $\phi = x^n + 1$ and n is a power of 2. Let $R_q \triangleq R/qR$ where q is a large prime satisfying $q = 1 \mod 2n$. The ring learning with errors assumption, denoted by RLWE, [LPR10] is analogous to the standard LWE assumption introduced by Regev [Reg09]. Let χ be a probability distribution on R_q. For $s \in R_q$, let $A_{s,\chi}$ be the probability distribution on $R_q \times R_q$ obtained by choosing an element $a \in R_q$ uniformly at random, choosing $e \leftarrow \chi$ and outputting $(a, a \cdot s + e)$.

Definition 2.6 (Ring Learning With Errors - $\mathsf{RLWE}_{\phi,q,\chi}$)**.** The decision R-$\mathsf{LWE}_{\phi,q,\chi}$ problem is: for $s \leftarrow R_q$, given a $\mathrm{poly}(n)$ number of samples that are either (all) from $A_{s,\chi}$ or (all) uniformly random in $R_q \times R_q$, output 0 if the former holds and 1 if the latter holds.

Theorem 2.7 ([LPR10])**.** *Let $r \geq \omega(\sqrt{\log n})$ be a real number and let R, q be as above. Then, there is a randomized reduction from $2^{\omega(\log n)} \cdot (q/r)$ approximate* RSVP *to* $\mathsf{RLWE}_{\phi,q,\chi}$ *where χ is the discrete Gaussian distribution with parameter r. The reduction runs in time* $\mathrm{poly}(n, q)$.

3 Linear Functional Encryption

Our construction will make use of the linear functional encryption scheme, denoted by LinFE, constructed by [ABCP15, ALS16].In this section, we describe some structural properties of LinFE that our construction requires. This section may be skipped on first reading and referred to as required.

Recall the functionality of LinFE: the encryptor provides a ciphertext $\mathsf{CT}_{\mathbf{x}}$ for some vector $\mathbf{x} \in R_p^k$, the key generator provides a key $\mathsf{SK}_{\mathbf{v}}$ for some vector $\mathbf{v} \in R_p^k$, and the decryptor, given $\mathsf{CT}_{\mathbf{x}}$ and $\mathsf{SK}_{\mathbf{v}}$ can compute $\langle \mathbf{x}, \mathbf{v} \rangle$. The construction of LinFE [ABCP15, ALS16] has several useful structural properties which our construction will rely on. These are described below.

1. **Setup:** Say that the message space and function space of LinFE are of dimension k, denoted by R_p^k. Then, the public key in LinFE may be interpreted as $\mathsf{PK} = (\mathsf{PK}_1, \ldots, \mathsf{PK}_k, \mathsf{PK}_{\mathsf{indpt}})$.

2. **Encryption:** Given a vector $\mathbf{x} \in R_p^k$, $\mathsf{LinFE.Enc}$ computes $\mathsf{CT}_i = \mathcal{E}(\mathsf{PK}_i, x_i)$ for $i \in [k]$ and $\mathsf{CT}_{\mathsf{indpt}} \leftarrow \mathcal{E}(\mathsf{PK}_{\mathsf{indpt}})$. Here, $\mathcal{E}$ is an encoding algorithm and all components CT_i are constructed using some common randomness. Here the data dependent components of the ciphertext are $(\mathsf{CT}_1, \ldots, \mathsf{CT}_k)$ and the data independent component is $\mathsf{CT}_{\mathsf{indpt}}$. Where PK_i is evident from context, we will denote CT_i simply by $\mathcal{E}(x_i)$.

3. **Key Generation:** Given a function vector $\mathbf{v}$, the algorithm $\mathsf{LinFE.KeyGen}$ is structured as follows:

   - Compute $\mathsf{PK}_{\mathbf{v}} = \mathsf{Eval}_{\mathsf{PK}}(\mathsf{PK}_1, \ldots, \mathsf{PK}_k, \mathbf{v})$. Here, $\mathsf{Eval}_{\mathsf{PK}}$ takes as input the public key components $\mathsf{PK}_1, \ldots, \mathsf{PK}_k$ as well as a function vector $\mathbf{v} \in R_p^k$ and outputs a "functional" public key component $\mathsf{PK}_{\mathbf{v}}$.
   - Let $\mathbf{k}_{\mathbf{v}} \leftarrow \mathsf{GenKey}(\mathsf{PK}_{\mathbf{v}}, \mathsf{PK}_{\mathsf{indpt}})$. Here, GenKey takes as input a functional public key component $\mathsf{PK}_{\mathbf{v}}$ and the data independent component $\mathsf{PK}_{\mathsf{indpt}}$ and outputs a functional secret key $\mathbf{k}_{\mathbf{v}}$.

4. **Decryption.** Given the public key $\mathsf{PK} = (\mathsf{PK}_1, \ldots, \mathsf{PK}_k, \mathsf{PK}_{\mathsf{indpt}})$, a ciphertext $\mathsf{CT}_{\mathbf{x}} = (\mathsf{CT}_1, \ldots, \mathsf{CT}_k, \mathsf{CT}_{\mathsf{indpt}})$ and a functional key $\mathsf{SK}_{\mathbf{v}} = (\mathbf{v}, \mathbf{k}_{\mathbf{v}})$, $\mathsf{LinFE.Dec}$ is structured as:

   - Let $\mathsf{CT}_{\langle \mathbf{x};\mathbf{v} \rangle} = \mathsf{Eval}_{\mathsf{CT}}(\mathsf{CT}_1, \ldots, \mathsf{CT}_k, \mathbf{v})$. Here, $\mathsf{Eval}_{\mathsf{CT}}$ takes as input the ciphertext components $\mathsf{CT}_1, \ldots, \mathsf{CT}_k$ as well as a function vector $\mathbf{v} \in R_p^k$ and outputs a "functional" ciphertext $\mathsf{CT}_{\langle \mathbf{x};\mathbf{v} \rangle}$.
   - Output $d = \mathsf{Decode}(\mathbf{k}_{\mathbf{v}}, \mathsf{CT}_{\langle \mathbf{x};\mathbf{v} \rangle}, \mathsf{CT}_{\mathsf{indpt}})$. Here, Decode is an algorithm that takes the functional secret key $\mathbf{k}_{\mathbf{v}}$, the functional ciphertext $\mathsf{CT}_{\langle \mathbf{x};\mathbf{v} \rangle}$ and the data independent ciphertext $\mathsf{CT}_{\mathsf{indpt}}$ and outputs the value $d = \langle \mathbf{x}; \mathbf{v} \rangle$.

We note that the above structure is also enjoyed by other lattice based constructions of functional encryption [BGG+14, GVW15].

LinFE Algorithms. Additionally, the LinFE construction [ABCP15, ALS16] implements the above operations as below. We do not provide all the details but only those which our constructions will use; we refer the reader to [ABCP15, ALS16] for details.

1. It holds that $\mathsf{PK}_{\mathsf{indpt}} = \mathbf{w}$ and $\mathsf{GenKey}(\mathsf{PK}_{\mathbf{v}}, \mathsf{PK}_{\mathsf{indpt}})$ returns a short vector $\mathbf{k}$ such that $\langle \mathbf{w}, \mathbf{k} \rangle = \mathsf{PK}_{\mathbf{v}}$.

2. The ciphertext component $\mathsf{CT}_{\mathsf{indpt}}$ contains randomness encoding of the form $\mathbf{d} = \mathbf{w} \cdot s + p \cdot \eta$ for some LWE secret s and noise η.

3. The functional ciphertext $\mathsf{CT}_{\langle \mathbf{x};\mathbf{v} \rangle}$ has the structure

$$\mathsf{CT}_{\langle \mathbf{x};\mathbf{v} \rangle} = \mathsf{PK}_{\mathbf{v}} \cdot s + p \cdot \eta_{\mathbf{v}} + \langle \mathbf{x}; \mathbf{v} \rangle$$

   where s is an LWE secret (the same as in the previous step) and $\eta_{\mathbf{v}}$ is some noise.

4. The Decode algorithm computes $\mathbf{k}^\intercal \mathbf{d} - \mathsf{CT}_{\langle \mathbf{x};\mathbf{v} \rangle} \bmod p$ and outputs it. Correctness follows exactly as in the dual Regev public key encryption scheme [GPV08]. That is, since $\langle \mathbf{w}, \mathbf{k} \rangle =$

$\mathsf{PK}_{\mathbf{v}}$, we have:

$$\begin{aligned}\mathbf{k}^{\mathsf{T}}\mathbf{d} - \mathsf{CT}_{\langle \mathbf{x};\mathbf{v}\rangle} &= \mathsf{PK}_{\mathbf{v}} \cdot s + p \cdot \mathbf{k}^{\mathsf{T}}\eta - \mathsf{PK}_{\mathbf{v}} \cdot s + p \cdot \eta_{\mathbf{v}} + \langle \mathbf{x}; \mathbf{v}\rangle \\ &= \langle \mathbf{x}; \mathbf{v}\rangle \bmod p\end{aligned}$$

Additional Ciphertext Structure. Additionally, the ciphertext of [ABCP15, ALS16] has the following additional properties:

1. **Malleability:** The ciphertext components of LinFE are malleable so that if $\mathsf{CT}_i = \mathcal{E}(x_i)$ then $\mathsf{CT}_i + x_j = \mathcal{E}(x_i + x_j)$ (as long as $x_i + x_j$ belong to the message space of $\mathcal{E}$).

2. **Additive homomorphism of public key and ciphertext components:** The public key and ciphertext components of LinFE enjoy additive homomorphism in the following sense. If $\mathsf{CT}_i = \mathcal{E}(x_i)$ and $\mathsf{CT}_j = \mathcal{E}(x_j)$ then $v_i\ \mathsf{CT}_i + v_j\ \mathsf{CT}_j = \mathcal{E}(v_i x_i + v_j x_j)$, as long as v_i, v_j belong to the valid function space R_p of LinFE. Similarly, if PK_i is the public key for CT_i and PK_j is the public key for CT_j then $v_i\ \mathsf{PK}_i + v_j\ \mathsf{PK}_j$ is the public key for $v_i\ \mathsf{CT}_i + v_j\ \mathsf{CT}_j$.

3. **Succinctness:** The ciphertext of LinFE is succinct, i.e. the size of $\mathsf{CT}(\mathbf{x})$ is $O(\mathrm{poly}(\lambda), |\mathbf{x}|)$ where λ is the security parameter.

4. $\mathsf{FULL\text{-}SIM}$ **Security:** It was shown by [ALS16] that LinFE satisfies adaptive indistinguishability, i.e. $\mathsf{AD\text{-}IND}$ security (please see Appendix 2 for the definition). As noted in [ALS16], LinFE satisfies $\mathsf{FULL\text{-}SIM}$ security since it is a "pre-image sampleable" function, due to the equivalence shown by [O'N10]. We refer the reader to [O'N10, ALS16] for details.

5. **Decomposability:** Decomposability means that an FE scheme supporting messages of length k have the following property: the public key and ciphertext may be decomposed into k components such that each component corresponds to a single element of the message. That is, the public key in LinFE may be interpreted as $\mathsf{PK} = (\mathsf{PK}_1, \ldots, \mathsf{PK}_k, \mathsf{PK}_{\mathsf{indpt}})$ and the ciphertext may be interpreted as $\mathsf{CT}_{\mathbf{x}} = (\mathsf{CT}_1, \ldots, \mathsf{CT}_k, \mathsf{CT}_{\mathsf{indpt}})$. Here the data dependent components of the public key and the ciphertext are $(\mathsf{PK}_1, \ldots, \mathsf{PK}_k)$ and $(\mathsf{CT}_1, \ldots, \mathsf{CT}_k)$ and the data independent components are $(\mathsf{PK}_{\mathsf{indpt}}, \mathsf{CT}_{\mathsf{indpt}})$. Additionally, we may write:

$$\mathsf{CT}_i = \mathcal{E}\ (\mathsf{PK}_i, x_i)\ \forall i \in [k] \quad \text{and} \quad \mathsf{CT}_{\mathsf{indpt}} = \mathcal{E}\ (\mathsf{PK}_{\mathsf{indpt}})$$

   Here, $\mathcal{E}$ is a randomised encoding algorithm. All components of the encryption are tied together using some common randomness.

These properties follow in a straightforward way from the LWE based LinFE schemes of [ABCP15, ALS16] since the message encodings in the ciphertext of these schemes have the structure of FHE ciphertexts [BV11a, BV11b]. The properties above are easy to verify, please see [ABCP15, ALS16] for details.

4 Bounded Query Functional Encryption for Quadratic forms

As a warm-up, we present our bounded key FE for the special case of quadratic functions, which we denote by QuadFE. Our construction will make use of the linear functional encryption scheme, denoted by LinFE, constructed by [ABCP15, ALS16], described in Section 3.

Our construction makes use of two prime moduli $p_0 < p_1$ where p_0 serves as the message space for QuadFE, and p_1 serves as the message space for LinFE. Let $L = |1 \le j \le i \le w|$. Below, let the distribution $\mathcal{D}_0$ be a discrete Gaussian with width σ_0, and $\mathcal{D}_1$ to be a discrete Gaussian with width σ_1 where $\frac{L \cdot p_0 \cdot \sigma_0^2}{\sigma_1} = \mathrm{negl}(\lambda)$. For more details about parameters, please see Appendix E.

For ease of exposition, our key generation algorithm receives the index of the requested key as input[5]. This restriction can be removed using standard tricks, see Appendix D for details. Additionally, we present our construction using Ring-LWE. This is both for efficiency and ease of exposition, the transformation to standard LWE follows standard machinery. We refer the reader to Appendix C for details.

$\mathsf{FE.Setup}(1^\lambda, 1^w, 1^Q)$: On input a security parameter λ, a parameter w denoting the length of message vectors and a parameter Q denoting the number of keys supported, do:

1. Invoke $\mathsf{LinFE.Setup}(1^\lambda, 1^{w+1+Q})$ to obtain $\mathsf{LinFE.PK}$ and $\mathsf{LinFE.MSK}$.
2. Sample $\mathbf{u} \leftarrow R_{p_1}^w$.
3. Output $\mathsf{PK} = (\mathsf{LinFE.PK}, \mathbf{u})$, $\mathsf{MSK} = (\mathsf{LinFE.MSK})$.

$\mathsf{FE.Enc}(\mathsf{PK}, \mathbf{x})$: On input public parameters PK, and message vector $\mathbf{x} \in R_{p_0}^w$ do:

1. Sample $s_1 \leftarrow R_{p_1}$ and $\boldsymbol{\mu} \leftarrow \mathcal{D}_0^w$, and compute an encoding of the message as:
$$\mathbf{c} = \mathbf{u} \cdot s_1 + p_0 \cdot \boldsymbol{\mu} + \mathbf{x} \in R_{p_1}^w.$$
2. For $i \in [Q]$, sample $\eta_i \leftarrow \mathcal{D}_1$ and let $\boldsymbol{\eta} = (\eta_1, \ldots, \eta_Q)$.
3. Let $\mathbf{b} = \mathsf{LinFE.Enc}\ (s_1^2, c_1 s_1, \ldots, c_w s_1, p_0 \cdot \boldsymbol{\eta})$.
4. Output $\mathsf{CT} = (\mathbf{c}, \mathbf{b})$

$\mathsf{FE.KeyGen}(\mathsf{PK}, \mathsf{MSK}, k, \mathbf{g})$: On input the public parameters PK, the master secret key MSK, a counter $k \in [Q]$ denoting the index of the requested function key and a function $\mathbf{g} = \sum_{1 \le j \le i \le w} g_{ij} x_i x_j$, represented as a coefficient vector $(g_{ij}) \in \mathbb{Z}_{p_0}^L$ do:

1. Let $\mathbf{e}_k$ denote the binary unit vector with a 1 in the k^{th} position and 0 elsewhere. Compute
$$\mathbf{u}_\mathbf{g} = \Big(\sum_{1 \le j \le i \le w} g_{ij} \left(u_i u_j, 0....0, -u_i, 0...0, -u_j, 0...0\right) \Big) \in R_{p_1}^{w+1}.$$
2. Compute $\mathsf{SK}_\mathbf{g} = \mathsf{LinFE.KeyGen}\big(\mathsf{LinFE.PK}, \mathsf{LinFE.MSK}, (\mathbf{u}_\mathbf{g}, \mathbf{e}_k)\big)$ and output it.

$\mathsf{FE.Dec}(\mathsf{PK}, \mathsf{SK}_\mathbf{g}, \mathsf{CT}_\mathbf{x})$: On input the public parameters PK, a secret key $\mathsf{SK}_\mathbf{g}$ for polynomial $\sum_{1 \le j \le i \le w} g_{ij} x_i x_j$, and a ciphertext $\mathsf{CT}_\mathbf{x} = (\mathbf{c}, \mathbf{b})$, compute
$$\sum_{1 \le j \le i \le w} g_{ij} c_i c_j + \mathsf{LinFE.Dec}(\mathbf{b}, \mathsf{SK}_\mathbf{g}) \bmod p_1 \bmod p_0$$
and output it.

[5] Alternately, one may consider a stateful algorithm which keeps track of the number of keys requested.

4.1 Correctness

We establish correctness of the above scheme.

Let $1 \leq j \leq i \leq w$. Let us assume $\mathbf{g}$ is the k^{th} key constructed by KeyGen, where $k \in [Q]$. By definition

$$x_i + p_0 \cdot \mu_i = c_i - u_i s_1 \mod p_1$$
$$x_j + p_0 \cdot \mu_j = c_j - u_j s_1 \mod p_1$$

Letting $\mu_{ij} = x_i\mu_j + x_j\mu_i + p_0\mu_i\mu_j$, we have

$$x_i x_j + p_0 \cdot \mu_{ij} = c_i c_j - c_i u_j s_1 - c_j u_i s_1 + u_i u_j s_1^2 \mod p_1 \tag{4.1}$$

By correctness of the linear scheme LinFE, we have that

$$\mathsf{LinFE.Dec}(\mathbf{b}, \mathsf{SK}_{\mathbf{g}}) = \sum_{1 \leq j \leq i \leq w} g_{ij}\big(- c_i u_j s_1 - c_j u_i s_1 + u_i u_j s_1^2\big) + p_0 \cdot \eta_k \tag{4.2}$$

Hence,
$$\begin{aligned}\sum_{1 \leq j \leq i \leq w} g_{ij} c_i c_j + \mathsf{LinFE.Dec}(\mathbf{b}, \mathsf{SK}_{\mathbf{g}}) &= \sum_{1 \leq j \leq i \leq w} g_{ij}\Big(c_i c_j - c_i u_j s_1 - c_j u_i s_1 + u_i u_j s_1^2\Big) + p_0 \cdot \eta_k \\ &= \sum_{1 \leq j \leq i \leq w} g_{ij}\big(x_i x_j + p_0 \cdot \mu_{ij}\big) + p_0 \cdot \eta_k \\ &= \sum_{1 \leq j \leq i \leq w} g_{ij}\, x_i x_j \bmod p_1 \bmod p_0 \quad \text{as desired.}\end{aligned}$$

4.2 Security

Theorem 4.1. *The construction in Section 4 achieves full simulation based security as per definition 2.2.*

Proof. We describe our simulator.

Simulator $\mathrm{Sim}\big(\ 1^\lambda, 1^{|\mathbf{x}|}, \mathsf{PK}, \{\mathbf{g}_k, \mathsf{SK}_{\mathbf{g}_k}, \mathbf{g}_k(\mathbf{x})\}_{k\in[Q]}\ \big)$**.** The simulator given input the security parameter, length of message $\mathbf{x}$, the functions $\mathbf{g}_1, \ldots, \mathbf{g}_Q$, the secret keys $\mathsf{SK}_{\mathbf{g}_1}, \ldots, \mathsf{SK}_{\mathbf{g}_Q}$ and the values $\mathbf{g}_1(\mathbf{x}), \ldots, \mathbf{g}_Q(\mathbf{x})$ does the following:

1. It picks the ciphertext $\mathbf{c} \leftarrow R_{p_1}^w$ randomly.
2. It parses $\mathbf{g}_k = \sum_{1 \leq j \leq i \leq w} g_{ij}^k x_i x_j$ for some $g_{ij}^k \in R_{p_0}$. For $k \in [Q]$, it samples $\eta_k \leftarrow \mathcal{D}_1$ and computes $d_k = \sum_{1 \leq j \leq i \leq w} g_{ij}^k (x_i x_j - c_i c_j) + p_0 \cdot \eta_k$.
3. It invokes the Q key LinFE simulator with input $\mathbf{d} = (d_1, \ldots, d_Q)$. It sets as $\mathbf{b}$ the output received by the LinFE simulator.
4. It outputs $\mathsf{CT}_{\mathbf{x}} = (\mathbf{c}, \mathbf{b})$.

We will prove that the output of the simulator is indistinguishable from the real world via a sequence of hybrids.

The Hybrids. Our Hybrids are described below.

Hybrid 0. This is the real world.

Hybrid 1. In this hybrid, the only thing that is different is that $\mathbf{b}$ is computed using the LinFE simulator as $\mathbf{b} = \mathsf{LinFE.Sim}\big(1^\lambda, 1^{w+1+Q}, \{\mathbf{g}_k, \mathsf{SK}_{\mathbf{g}_k}, d_k\}_{k\in[Q]}\big)$ where

$$d_k = \sum_{1\le j\le i\le w} g^k_{ij}\big(x_i x_j - c_i c_j\big) + p_0 \cdot \Big(\sum_{1\le j\le i\le w} g^k_{ij}\mu_{ij} + \eta_k\Big) \;\; \forall \;\; k \in [Q]$$

Above, μ_{ij} is as defined in Equation 4.1.

Hybrid 2. In this hybrid, let $d_k = \sum\limits_{1\le j\le i\le w} g^k_{ij}\big(x_i x_j - c_i c_j\big) + p_0 \cdot \eta_k$ for $k \in [Q]$.

Hybrid 3. In this hybrid, sample $\mathbf{c}$ at random. This is the simulated world.

Indistinguishability of Hybrids. Below we establish that consecutive hybrids are indistinguishable.

Claim 4.2. *Hybrid 0 is indistinguishable from Hybrid 1 assuming that* LinFE *is secure.*

Proof. Recall that for $j \le i \le w$, we have:

$$x_i x_j + p_0 \cdot \mu_{ij} = c_i c_j - c_i u_j s_1 - c_j u_i s_1 + u_i u_j s_1^2$$

Hence, it holds that

$$\sum_{j\le i\le w} g^k_{ij}\big(x_i x_j + p_0 \cdot \mu_{ij}\big) = \sum_{j\le i\le w} g^k_{ij}\Big(c_i c_j + u_i u_j s_1^2 - u_j c_i s_1 - u_i c_j s_1\Big)$$

$$\sum_{j\le i\le w} g^k_{ij}\big(x_i x_j - c_i c_j\big) + p_0 \cdot \Big(\sum_{j\le i\le w} g^k_{ij}\mu_{ij} + \eta_k\Big) = \sum_{j\le i\le w} g^k_{ij}\big(u_i u_j s_1^2 - u_j c_i s_1 - u_i c_j s_1\big) + p_0 \cdot \eta_k$$

In Hybrid 0, we have by Equation 4.2 that the output of LinFE decryption is:

$$\sum_{1\le j\le i\le w} g_{ij}\big(-c_i u_j s_1 - c_j u_i s_1 + u_i u_j s_1^2\big) + p_0 \cdot \eta_k$$
$$= \sum_{j\le i\le w} g^k_{ij}\big(x_i x_j - c_i c_j\big) + p_0 \cdot \Big(\sum_{j\le i\le w} g^k_{ij}\mu_{ij} + \eta_k\Big)$$

In Hybrid 1, the LinFE simulator is invoked with the above value, hence by security of LinFE, Hybrids 0 and 1 are indistinguishable. □

Claim 4.3. *Hybrid 1 and Hybrid 2 are statistically indistinguishable.*

Proof. This follows by our choice of parameters since for $k \in [Q]$, we have

$$\mathsf{SD}\big(\sum_{1\le j\le i\le w} g_{ij}^k \mu_{ij} + \eta_k, \eta_k\big) = \mathrm{negl}(\lambda)$$

□

Hybrid 2 and Hybrid 3 are indistinguishable assuming the hardness of LWE. In more detail, we show:

Claim 4.4. *Assume Regev public key encryption is semantically secure. Then, Hybrid 2 is indistinguishable from Hybrid 3.*

Proof. Recall that by semantic security of Regev's (dual) public key encryption, we have that the ciphertext $\mathbf{c} = \mathbf{u} \cdot s_1 + p_0 \cdot \boldsymbol{\mu} + \mathbf{x}$ is indistinguishable from random, where $\mathbf{u}$ is part of the public key and $\boldsymbol{\mu} \leftarrow \mathcal{D}_0$ is suitably chosen noise. We refer the reader to [GPV08] for more details.

Given an adversary $\mathcal{B}$ who distinguishes between Hybrid 2 and Hybrid 3, we build an adversary $\mathcal{A}$ who breaks the semantic security of Regev public key encryption. The adversary $\mathcal{A}$ receives $\mathsf{PK} = \mathbf{u}$ upon which, it simulates the view of $\mathcal{B}$ as follows:

- Run $\mathsf{LinFE.Setup}$ to obtain $\mathsf{LinFE.PK}$ and $\mathsf{LinFE.MSK}$. Return $\mathsf{PK} = (\mathsf{LinFE.PK}, \mathbf{u})$ to $\mathcal{B}$.
- When $\mathcal{B}$ requests a key $\mathbf{g}_k$ for $k \in [Q]$, construct it honestly as in Hybrid 0.
- When $\mathcal{B}$ outputs challenge $\mathbf{x}$, $\mathcal{A}$ outputs the same.
- $\mathcal{A}$ receives $\mathbf{c}$ where $\mathbf{c} = \mathbf{u} \cdot s_1 + p_0 \cdot \boldsymbol{\mu} + \mathbf{x}$ or random.
- $\mathcal{A}$ samples $\eta_1, \ldots, \eta_Q$ as in Hybrid 2 and computes $d_k = \sum_{1\le j\le i\le w} g_{ij}^k (x_i x_j - c_i c_j) + p_0 \cdot \eta_k$. It invokes $\mathsf{LinFE.Sim}\big(1^\lambda, 1^{w+1+Q}, \{\mathbf{g}_k, \mathsf{SK}_{\mathbf{g}_k}, d_k\}_{k\in[Q]}\big)$ and receives LinFE ciphertext $\mathbf{b}$. It returns $(\mathbf{c}, \mathbf{b})$ to $\mathcal{B}$.
- $\mathcal{B}$ may request more keys (bounded above by Q) which are handled as before. Finally, when $\mathcal{B}$ outputs a guess bit b, $\mathcal{A}$ outputs the same.

Clearly, if $b = 0$, then $\mathcal{B}$ sees the distribution of Hybrid 2, whereas if $b = 1$, it sees the distribution of Hybrid 3. Hence the claim follows. □

□

Basing the Construction on Standard LWE. The above construction may equivalently be based on standard rather than ring LWE, please see Appendix C for details.

5 Public Key and Ciphertext Evaluation Algorithms

In this section, provide the tools to extend our construction for quadratic polynomials to circuits in NC_1. Throughout this section, we assume circular security of LWE. This is for ease of exposition as well as efficiency. This assumption can be removed by choosing new randomness s_i for each level i as in levelled fully homomorphic encryption. Since the intuition was discussed in Section 1, we proceed with the technical overview and construction.

Notation. To begin, it will be helpful to set up some notation. We will consider circuits of depth d, consisting of alternate addition and multiplication layers. Each layer of the circuit is associated with a modulus p_k for level k. For an addition layer at level k, the modulus p_k will be the same as the previous modulus p_{k-1}; for a multiplication layer at level k, we require $p_k > p_{k-1}$. This results in a tower of moduli $p_0 < p_1 = p_2 < p_3 = \ldots < p_d$.

As in LinFE [ABCP15, ALS16](please refer to Section 3), we define encoding functions $\mathcal{E}^k$ for $k \in [d]$ such that $\mathcal{E}^k : R_{p_{k-1}} \to R_{p_k}$. At level k, the encryptor will provide L^k encodings $\mathcal{C}^k$ for some $L^k = O(2^k)$. For $i \in [L^k]$ we define

$$\mathcal{E}^k(y_i) = u_i^k \cdot s + p_{k-1} \cdot \eta_i^k + y_i.$$

Here $u_i^k \in R_{p_k}$, $\eta_i^k \leftarrow \chi_k$ and $y_i \in R_{p_{k-1}}$. We will refer to $\mathcal{E}^k(y_i)$ as the Regev encoding of y_i. At level k, the decryptor will be able to compute a Regev encoding of $f^k(\mathbf{x})$ where f^k is the circuit f restricted to level k.

It will be convenient for us to denote encodings of $f^k(\mathbf{x})$ by c^k, i.e. $c^k = \mathcal{E}^k\big(f^k(\mathbf{x})\big)$. We emphasize that c^k are computed on the fly by the decryptor whereas $\mathcal{C}^k$ are a set of level k encodings provided by the encryptor to enable the decryptor to compute c^k. We will denote the public key or label of an encoding $\mathcal{E}^k(\cdot)$ by $\mathsf{PK}(\mathcal{E}^k(\cdot))$. In our construction, we will compose encodings, so that encodings at a level k are messages to encodings at level $k+1$. We refer to such encodings as *nested encodings*. We will need the notions of nesting level and nested message degree, defined as follows.

Definition 5.1 (Nesting level and Nested Message Degree.)**.** Given a composition of successive encodings, i.e. an encoding of the form $\mathcal{E}^k\big(\mathcal{E}^{k-1}\big(\ldots(\mathcal{E}^{\ell+1}(\mathcal{E}^{\ell}(y)\cdot s)\cdot s)\ldots\cdot s\big)\cdot s\big)$, we will denote as *nesting level* the value $k-\ell$, the *nested message* of the encoding as y, and the *nested message degree* of the encoding as the degree of the innermost polynomial y.

We prove the following theorem.

Theorem 5.2. *There exists a set of encodings $\mathcal{C}^i$ for $i \in [d]$, such that:*

1. **Encodings have size sublinear in circuit.** $\forall i \in [d]\ |\mathcal{C}^i| = O(2^i)$.
2. **Efficient public key and ciphertext evaluation algorithms.** *There exist efficient algorithms $\mathsf{Eval}_{\mathsf{PK}}$ and $\mathsf{Eval}_{\mathsf{CT}}$ so that for any circuit f of depth d, if $\mathsf{PK}_f = \mathsf{Eval}_{\mathsf{PK}}(\mathsf{PK}, f)$ and $\mathsf{CT}_{(f(\mathbf{x}))} = \mathsf{Eval}_{\mathsf{CT}}(\bigcup_{i\in[d]} \mathcal{C}^i, f)$, then $\mathsf{CT}_{(f(\mathbf{x}))}$ is a "Regev encoding" of $f(\mathbf{x})$ under public key PK_f. Specifically, for some LWE secret s, we have:*

$$\mathsf{CT}_{(f(\mathbf{x}))} = \mathsf{PK}_f \cdot s + p_{d-1} \cdot \eta_f^{d-1} + \mu_{f(\mathbf{x})} + f(\mathbf{x}) \tag{5.1}$$

where $p_{d-1} \cdot \eta_f^{d-1}$ is RLWE noise and $\mu_{f(\mathbf{x})} + f(\mathbf{x})$ is the desired message $f(\mathbf{x})$ plus some noise $\mu_{f(\mathbf{x})}$[6]. Here, $\mu_{f(\mathbf{x})} = p_{d-2} \cdot \eta_f^{d-2} + \ldots p_0 \cdot \eta_f^0$ for some noise terms $\eta_f^{d-2}, \ldots, \eta_f^0$.

[6]Here $\mu_{f(\mathbf{x})}$ is clubbed with the message $f(\mathbf{x})$ rather than the RLWE noise $p_{d-1} \cdot \eta_f^{d-1}$ since $\mu_{f(\mathbf{x})} + f(\mathbf{x})$ is what will be recovered after decryption of $\mathsf{CT}_{f(\mathbf{x})}$, whereas $p_{d-1} \cdot \eta_f^{d-1}$ will be removed by the decryption procedure. This is merely a matter of notation.

3. **Ciphertext and public key structure.** *The structure of the functional ciphertext is as:*

$$\mathsf{CT}_{f(\mathbf{x})} = \mathsf{Poly}_f(\mathcal{C}^1, \ldots, \mathcal{C}^{d-1}) + \langle \mathsf{Lin}_f, \mathcal{C}^d \rangle \tag{5.2}$$

where $\mathsf{Poly}_f(\mathcal{C}^1, \ldots, \mathcal{C}^{d-1}) \in R_{p_{d-1}}$ *is a high degree polynomial value obtained by computing a public* f*-dependent function on level* $k \leq d-1$ *encodings* $\{\mathcal{C}^k\}_{k\in[d-1]}$ *and* $\mathsf{Lin}_f \in R_{p_d}^{L_d}$ *is an* f*-dependent linear function. We also have*

$$f(\mathbf{x}) + \mu_{f(\mathbf{x})} = \mathsf{Poly}_f(\mathcal{C}^1, \ldots, \mathcal{C}^{d-1}) + \langle \mathsf{Lin}_f, \mathcal{M}^d \rangle \tag{5.3}$$

where $\mathcal{M}^d$ *are the messages encoded in* $\mathcal{C}^d$ *and* $\mu_{f(\mathbf{x})}$ *is functional noise. The public key for the functional ciphertext is structured as:*

$$\mathsf{PK}\big(\mathsf{CT}_{f(\mathbf{x})}\big) = \Big\langle \mathsf{Lin}_f,\ \big(\mathsf{PK}(\mathcal{C}_1^d), \ldots, \mathsf{PK}(\mathcal{C}_{L_d}^d)\big) \Big\rangle \tag{5.4}$$

The Encodings. We define $\mathcal{C}^k$ recursively as follows:

1. $\mathcal{C}^1 \triangleq \{\mathcal{E}^1(x_i), \mathcal{E}^1(s)\}$
2. If k is a multiplication layer, $\mathcal{C}^k = \{\mathcal{E}^k(\mathcal{C}^{k-1}), \mathcal{E}^k(\mathcal{C}^{k-1} \cdot s), \mathcal{E}^k(s^2)\}$. If k is an addition layer, let $\mathcal{C}^k = \mathcal{C}^{k-1}$.

We prove that:

Lemma 5.3. *Assume that* k *is a multiplication layer. Given* $\mathcal{C}^k$ *for any* $2 < k < d$*,*

1. *Level* k *encodings* $\mathcal{E}^k(c^{k-1} \cdot s)$ *and* $\mathcal{E}^k(c^{k-1})$ *may be expressed as quadratic polynomials in level* $k-1$ *encodings and the level* k *advice encodings* $\mathcal{C}^k$*. In particular, the polynomials are linear in terms* $\mathcal{C}^k$ *and quadratic in level* $k-1$ *encodings* $\mathcal{E}^{k-1}(y_i)\mathcal{E}^{k-1}(y_j)$*. The messages* y_i, y_j *of the form* c_ℓ^{k-3} *or* $c_\ell^{k-3} \cdot s$ *for some level* $k-3$ *ciphertext* c_ℓ^{k-3}*.*

   Since the exact value of the coefficients is not important, we express this as:

$$\mathcal{E}^k(c^{k-1} \cdot s), \mathcal{E}^k(c^{k-1}) = \mathsf{LinComb}\big(\ \mathcal{C}^k, \mathcal{E}^{k-1}(y_i)\mathcal{E}^{k-1}(y_j)\big)\ \ \forall\ i, j \tag{5.5}$$

2. *We can compute* c^k *and* c^{k+1} *as a linear combination of quadratic terms in level* $k-1$ *encodings and linear in level* k *encodings* $\mathcal{C}^k$*. In particular,*

$$\begin{aligned} c^k = \mathsf{CT}(f^k(\mathbf{x}) + \mu^k_{f(\mathbf{x})}) &= \langle \mathsf{Lin}_{f^k}, \mathcal{C}^k \rangle + \mathsf{LinComb}\big(\mathsf{Quad}(\mathcal{E}^{k-1}(y_i)\ \mathcal{E}^{k-1}(y_j))\big) \\ &= \langle \mathsf{Lin}_{f^k}, \mathcal{C}^k \rangle + \mathsf{Poly}_{f^k}\big(\mathcal{C}^1, \ldots, \mathcal{C}^{k-1}\big) \end{aligned}$$

Proof by induction.

Base Case. While the quadratic scheme described in Section 4 suffices as a base case, we work out an extended base case for level 4 circuits, since this captures the more general case.

We claim that $\mathcal{C}^4$ defined according to the above rules, permits the evaluator to compute :

1. $\mathcal{E}^4(c^3 \cdot s)$ and $\mathcal{E}^4(c^3)$ by taking linear combinations of elements in $\mathcal{C}^4$ and adding to this a quadratic term of the form $\mathcal{E}^3(y_i)\mathcal{E}^3(y_j)$ where $\mathcal{E}^3(y_i)\mathcal{E}^3(y_j) \in \mathcal{C}^3 = \mathcal{C}^2$. We note that since $k-1$ is an addition layer, $\mathcal{C}^3 = \mathcal{C}^2$.

2. Encodings of level 4 functions of $\mathbf{x}$, namely c^4.

Note that our level 2 ciphertext may be written as:

$$\begin{aligned} c^2_{i,j} = \mathcal{E}^2(x_i x_j + p_0 \cdot \mu_{ij}) &= \mathcal{E}^2\big(c^1_i c^1_j + \ u^1_i u^1_j(s^2) - u^1_j(c^1_i s) - u^1_i(c^1_j s)\ \big) \\ &= \mathcal{E}^2(x_i x_j + p_0 \cdot \mu_{ij}) = c^1_i c^1_j + \mathcal{E}^2\big(\ u^1_i u^1_j(s^2) - u^1_j(c^1_i s) - u^1_i(c^1_j s)\ \big) \\ &= c^1_i c^1_j + u^1_i u^1_j\ \mathcal{E}^2(s^2) - u^1_j\ \mathcal{E}^2(c^1_i s) - u^1_i\ \mathcal{E}^2(c^1_j s) \quad \in R_{p_2} \end{aligned} \tag{5.6}$$

In the above, the first equality follows by ciphertext malleability: here, $c^1_i c^1_j \in R_{p_1}$ is a message added to the encoding $\mathcal{E}^2(u^1_i u^1_j(s^2) - u^1_j(c^1_i s) - u^1_i(c^1_j s))$. The second equality follows by additive homomorphism of the encodings. Please see Section 3 for details. Additionally, the public key and the noise of the resultant encoding may be computed as:

$$\begin{aligned} u^2_\ell \triangleq \mathsf{PK}\left(\mathcal{E}^2(x_i x_j + p_0 \cdot \mu_{ij})\right) &= u^1_i u^1_j\ \mathsf{PK}\left(\mathcal{E}^2(s^2)\right) - u^1_j\ \mathsf{PK}\left(\mathcal{E}^2(c^1_i s)\right) - u^1_i\ \mathsf{PK}\left(\mathcal{E}^2(c^1_j s)\right) \\ \mathsf{Nse}^2_\ell \triangleq \mathsf{Nse}\left(\mathcal{E}^2(x_i x_j + p_0 \cdot \mu_{ij})\right) &= u^1_i u^1_j\ \mathsf{Nse}\left(\mathcal{E}^2(s^2)\right) - u^1_j\ \mathsf{Nse}\left(\mathcal{E}^2(c^1_i s)\right) - u^1_i\ \mathsf{Nse}\left(\mathcal{E}^2(c^1_j s)\right) \end{aligned}$$

Above, $\mathsf{Nse}(\mathcal{E}^2(\cdot))$ refers to the noise level in the relevant encoding. Note that even though u^1_i are chosen uniformly in R_{p_1}, they do not blow up the noise in the above equation since the above noise is relative to the larger ring R_{p_2}. This noise growth can be controlled further by using the bit decomposition trick [BV11a, BGV12] – we do not do this here for ease of exposition.

The Quadratic Method. Thus, we may compute a level 2 encoding as:

$$\mathcal{E}^2(x_i x_j + p_0 \cdot \mu_{ij}) = \mathcal{E}^1(x_i)\mathcal{E}^1(x_j) + u^1_i u^1_j\ \mathcal{E}^2(s^2) - u^1_j\ \mathcal{E}^2(\mathcal{E}^1(x_i) \cdot s) - u^1_i\ \mathcal{E}^2(\mathcal{E}^1(x_j) \cdot s) \tag{5.7}$$

We refer to this computation as the "quadratic method".

The key point is that our level 2 ciphertext has the exact same structure as a level 1 encoding, namely it is a Regev encoding using some secret s, some label and noise as computed in equations 5.7. Thus, letting $y_\ell = x_i x_j$, we may write

$$\mathcal{E}^2(y_\ell) = u^2_\ell \cdot s + \mathsf{Nse}^2_\ell + y_\ell \quad \in R_{p_2} \tag{5.8}$$

Addition (Level 3). To add two encoded messages $y_\ell = x_i x_j + p_0 \cdot \mu_{ij}$ and $y_{\ell'} = x_{i'} x_{j'} + p_0 \cdot \mu_{i'j'}$, it is easy to see that adding their encodings suffices. The resultant public key and noise is just the summation of the individual public keys and noise terms. Thus, if the ℓ^{th} wire is the sum of the i^{th} and j^{th} wires, we have:

$$c^3_\ell = c^2_i + c^2_j \tag{5.9}$$

and

$$\mathsf{PK}(c^3_\ell) = \mathsf{PK}(c^2_i) + \mathsf{PK}(c^2_j) \tag{5.10}$$

Multiplication (Level 4). The nontrivial case is that of multiplication. We next compute an encoding for the product of $y_\ell = x_i x_j + x_m x_t + p_0 \cdot \mu_\ell^4$ and $y_{\ell'} = x_{i'} x_{j'} + x_{m'} x_{t'} + p_0 \cdot \mu_{\ell'}^4$ where $\mu_\ell^4, \mu_{\ell'}^4$ are level 4 noise terms computed as $\mu_\ell^4 = \mu_{ij} + \mu_{mt}$ (analogously for $\mu_{\ell'}^4$). Let c_ℓ^3 and $c_{\ell'}^3$ denote the encodings of y_ℓ and $y_{\ell'}$ computed using the first three levels of evaluation. As before, we have by the above quadratic method:

$$\begin{aligned} c_t^4 &= \mathcal{E}^4(y_\ell y_{\ell'}) = c_\ell^3 c_{\ell'}^3 + \mathcal{E}^4\big(u_\ell^3 u_{\ell'}^3(s^2) - u_{\ell'}^3(c_\ell^3 s) - u_\ell^3(c_{\ell'}^3 s)\big) \quad \in R_{p_4} \\ &= c_\ell^3 c_{\ell'}^3 + u_\ell^3 u_{\ell'}^3\ \mathcal{E}^4(s^2) - u_{\ell'}^3\ \mathcal{E}^4(c_\ell^3 s) - u_\ell^3\ \mathcal{E}^4(c_{\ell'}^3 s) \end{aligned} \tag{5.11}$$

By correctness of first three levels of evaluation as described above, the decryptor can compute the encoding of y_ℓ, namely c_ℓ^3 correctly, hence the quadratic term $c_\ell^3 c_{\ell'}^3$ may be computed. It remains to compute the terms $\mathcal{E}^4(c_\ell^3 s)$. Note that the encryptor may not provide the encodings $\mathcal{E}^4(c_\ell^3 s)$ directly and preserve succinctness because $c_\ell^3 = \mathcal{E}^2(x_i\ x_j + p_0 \cdot \mu_{ij}) + \mathcal{E}^2(x_m\ x_t + p_0 \cdot \mu_{mt})$ and $\mathcal{E}^2(x_i\ x_j + p_0 \cdot \mu_{ij})$ contains the quadratic term $c_i^1 c_j^1$ as shown by Equation 5.6.

Consider the term $\mathcal{E}^4(c_\ell^3 s)$. In fact, we will only be able to compute a noisy version of this encoding, i.e. $\mathcal{E}^4(c_\ell^3 s + p_1 \cdot \mu_\ell^2)$ for some $p_1 \cdot \mu_\ell^2$.

$$\begin{aligned} \mathcal{E}^4(c_\ell^3 s) &= \mathcal{E}^4\big((\mathcal{E}^2(x_i\ x_j + p_0 \cdot \mu_{ij}) + \mathcal{E}^2(x_m\ x_t + p_0 \cdot \mu_{mt})) \cdot s\big) \\ &= \mathcal{E}^4\Big(\big(c_i^1 c_j^1 + u_i^1 u_j^1\ \mathcal{E}^2(s^2) - u_j^1\ \mathcal{E}^2(c_i^1 s) - u_i^1\ \mathcal{E}^2(c_j^1 s)\ \big)\ \cdot s\Big) \\ &+ \mathcal{E}^4\Big(\big(c_m^1 c_t^1 + u_m^1 u_t^1\ \mathcal{E}^2(s^2) - u_t^1\ \mathcal{E}^2(c_m^1 s) - u_m^1\ \mathcal{E}^2(c_t^1 s)\ \big)\ \cdot s\Big) \\ &= \mathcal{E}^4(c_i^1 c_j^1 s)\ + \mathcal{E}^4\big(u_i^1 u_j^1\ \mathcal{E}^2(s^2)\ s\big)\ - \mathcal{E}^4\big(u_j^1\ \mathcal{E}^2(c_i^1 s)\ s\big) - \mathcal{E}^4\big(u_i^1\ \mathcal{E}^2(c_j^1 s)\ s\big) \\ &+ \mathcal{E}^4(c_m^1 c_t^1 s)\ + \mathcal{E}^4\big(u_m^1 u_t^1\ \mathcal{E}^2(s^2)\ s\big) - \mathcal{E}^4\big(u_t^1\ \mathcal{E}^2(c_m^1 s) s\big) - \mathcal{E}^4\big(u_m^1\ \mathcal{E}^2(c_t^1 s)\ s\big) \\ &= \mathcal{E}^4(c_i^1 c_j^1 s)\ + u_i^1 u_j^1\ \mathcal{E}^4\big(\mathcal{E}^2(s^2)\ s\big)\ - u_j^1\ \mathcal{E}^4\big(\mathcal{E}^2(c_i^1 s)\ s\big) - u_i^1\ \mathcal{E}^4\big(\mathcal{E}^2(c_j^1 s)\ s\big) \\ &+ \mathcal{E}^4(c_m^1 c_t^1 s)\ + u_m^1 u_t^1\ \mathcal{E}^4\big(\mathcal{E}^2(s^2)\ s\big) - u_t^1\ \mathcal{E}^4\big(\mathcal{E}^2(c_m^1 s) s\big) - u_m^1\ \mathcal{E}^4\big(\mathcal{E}^2(c_t^1 s)\ s\big) \end{aligned} \tag{5.12}$$

Thus, to compute $\mathcal{E}^4(c_\ell^3 s)$ by additive homomorphism, it suffices to compute the encodings $\mathcal{E}^4(c_i^1 c_j^1 s)$, $\mathcal{E}^4\big(\mathcal{E}^2(s^2)\ s\big)$ and $\mathcal{E}^4\big(\mathcal{E}^2(c_j^1 s)\ s\big)$ for all i, j. Note that by definition of $\mathcal{C}^4$, we have that for $m \in [w]$,

$$\Big\{\mathcal{E}^4\big(\mathcal{E}^2(s^2)\ s\big),\quad \mathcal{E}^4\big(\mathcal{E}^2(c_m^1 s) s\big)\Big\} \subseteq \mathcal{C}^4 \tag{5.13}$$

Note that since level 3 is an addition layer, $\mathcal{E}^3 = \mathcal{E}^2$.

The only terms above not accounted for are $\mathcal{E}^4(c_i^1 c_j^1 s)$ and $\mathcal{E}^4\big(c_m^1 c_t^1 s\big)$. Consider the former. To compute this, we view $c_i^1 c_j^1 s$ as a quadratic term in c_i^1 and $c_j^1 \cdot s$ and re-apply the quadratic method given in Equation 5.7. This will enable us to compute a noisy version of $\mathcal{E}^4(c_i^1 c_j^1 s)$, namely $\mathcal{E}^4(c_i^1 c_j^1 s + p_1 \cdot \mu_{ij}^2)$ for some noise μ_{ij}^2.

Applying the Quadratic Method (Equation 5.7): Given $\mathcal{E}^2(c_i^1)$, $\mathcal{E}^2(c_j^1 \cdot s)$ along with $\mathcal{E}^4\big(\mathcal{E}^2(c_i^1)\ s\big)$ and $\mathcal{E}^4\big(\mathcal{E}^2(c_j^1 \cdot s)\ s\big)$ we may compute $\mathcal{E}^4(c_i^1 c_j^1 s + p_1 \cdot \mu_{ij}^2)$ as described above. In more detail, we let

$$d_i \triangleq \mathcal{E}^2(c_i^1)\ ,\quad h_j \triangleq \mathcal{E}^2(c_j^1 \cdot s)\ \in R_{p_2}\ \text{ and }\ \hat{d}_i \triangleq \mathcal{E}^4\big(\mathcal{E}^2(c_i^1)\ s\big)\ ,\quad \hat{h}_j \triangleq \mathcal{E}^4\big(\mathcal{E}^2(c_j^1 \cdot s)\ s\big)\ \in R_{p_4}$$

Then, we have:

$$\mathcal{E}^4(c_i^1 c_j^1 s + p_1 \cdot \mu_{ij}^2) = d_i h_j + \mathsf{PK}\big(\mathcal{E}^2(c_i^1)\big)\ \mathsf{PK}\big(\mathcal{E}^2(c_j^1 \cdot s)\big)\ \mathcal{E}^4(s^2) - \mathsf{PK}\big(\mathcal{E}^2(c_i^1)\big)\ \hat{h}_j - \mathsf{PK}\big(\mathcal{E}^2(c_j^1 \cdot s)\big)\ \hat{d}_i \quad \in R_{p_4} \tag{5.14}$$

where $\mu_{ij}^2 = c_i^1 \cdot \mathsf{Nse}(\mathcal{E}^2(c_j^1 \cdot s)) + c_j^2 \cdot \mathsf{Nse}(\mathcal{E}^2(c_i^1)) + p_1 \cdot \mathsf{Nse}(\mathcal{E}^2(c_j^1 \cdot s)) \cdot \mathsf{Nse}(\mathcal{E}^2(c_i^1))$.

Also, the public key for $\mathcal{E}^4(c_i^1 c_j^1 s + p_1 \cdot \mu_{ij}^2)$ may be computed as:

$$\mathsf{PK}\big(\mathcal{E}^4(c_i^1 c_j^1 s + p_1 \cdot \mu_{ij}^2)\big) = \mathsf{PK}\big(\mathcal{E}^2(c_i^1)\big)\ \mathsf{PK}\big(\mathcal{E}^2(c_j^1 \cdot s)\big)\ \mathsf{PK}\big(\mathcal{E}^4(s^2)\big) - \mathsf{PK}\big(\mathcal{E}^2(c_i^1)\big)\ \mathsf{PK}(\hat{h}_j) - \mathsf{PK}\big(\mathcal{E}^2(c_j^1 \cdot s)\big)\ \mathsf{PK}(\hat{d}_i) \tag{5.15}$$

Thus we have, $\mathcal{E}^4(c_\ell^3 s + p_1 \cdot \mu_\ell^2)$ is a Regev encoding with public key

$$\begin{aligned}
\mathsf{PK}\big(\mathcal{E}^4(c_\ell^3 s + p_1 \cdot \mu_\ell^2)\big) = \mathsf{PK}\Big(&\mathcal{E}^4(c_i^1 c_j^1 s + p_1 \cdot \mu_{ij}^2)\ + u_i^1 u_j^1\ \mathcal{E}^4\big(\mathcal{E}^2(s^2)\ s\big)\ - u_j^1\ \mathcal{E}^4\big(\mathcal{E}^2(c_i^1 s)\ s\big) - u_i^1\ \mathcal{E}^4\big(\mathcal{E}^2(c_j^1 s)\ s\big) \\
&+ \mathcal{E}^4\big((c_m^1 c_t^1 s + p_1 \cdot \mu_{mt}^2)\ + u_m^1 u_t^1\ \mathcal{E}^4\big(\mathcal{E}^2(s^2)\ s\big) - u_t^1\ \mathcal{E}^4\big(\mathcal{E}^2(c_m^1 s) s\big) - u_m^1\ \mathcal{E}^4\big(\mathcal{E}^2(c_t^1 s)\ s\big)\Big) \\
= \mathsf{PK}\big(&\mathcal{E}^4(c_i^1 c_j^1 s + p_1 \cdot \mu_{ij}^2)\big)\ + u_i^1 u_j^1\ \mathsf{PK}\big(\mathcal{E}^4\big(\mathcal{E}^2(s^2)\ s\big)\big)\ - u_j^1\ \mathsf{PK}\big(\mathcal{E}^4\big(\mathcal{E}^2(c_i^1 s)\ s\big)\big) \\
&- u_i^1\ \mathsf{PK}\big(\mathcal{E}^4\big(\mathcal{E}^2(c_j^1 s)\ s\big)\big) + \mathsf{PK}\big(\mathcal{E}^4\big((c_m^1 c_t^1 s + p_1 \cdot \mu_{mt}^2)\big)\ + u_m^1 u_t^1\ \mathsf{PK}\big(\mathcal{E}^4\big(\mathcal{E}^2(s^2)\ s\big)\big) \\
&- u_t^1\ \mathsf{PK}\big(\mathcal{E}^4\big(\mathcal{E}^2(c_m^1 s) s\big)\big) - u_m^1\ \mathsf{PK}\big(\mathcal{E}^4\big(\mathcal{E}^2(c_t^1 s)\ s\big)\big)
\end{aligned} \tag{5.16}$$

Above $\mathsf{PK}\big(\mathcal{E}^4(c_i^1 c_j^1 s + p_1 \cdot \mu_{ij}^2)\big)$ may be computed by Equation 5.15 and the remaining public keys are provided in $\mathcal{C}^4$ as described in Equation 5.13. Also, we have $\mu_\ell^2 = \mu_{ij}^2 + \mu_{mt}^2$.

By equations 5.12, 5.13 and 5.14, we may compute $\mathcal{E}^4(c_\ell^2 s + p_1 \cdot \mu_\ell^2)$ for any ℓ.

Note that,

$$\begin{aligned}
\mathcal{E}^4(c_\ell^2 s + p_1 \cdot \mu_\ell^2) &= \mathsf{LinComb}\Big(\mathcal{E}^2(c_i^1) \cdot \mathcal{E}^2(c_j^1 \cdot s), \mathcal{E}^4\big(\mathcal{E}^2(c_i^1)\ s\big),\ \mathcal{E}^4\big(\mathcal{E}^2(c_j^1 \cdot s)\ s\big)\Big) \\
&= \langle \mathsf{Lin}_{f^4},\ \mathcal{C}^4 \rangle + \mathsf{Quad}\big(\mathcal{E}^2(c_i^1) \cdot \mathcal{E}^2(c_j^1 \cdot s)\ \big)
\end{aligned}$$

for some linear function Lin_{f^4}.

5.1 Ciphertext and Public Key Structure.

By Equation 5.11, we then get that

$$\begin{aligned}
c_t^4 &= c_\ell^3\ c_{\ell'}^3 + u_\ell^3\ u_{\ell'}^3 \mathcal{E}^4(s^2) - u_\ell^3 \Big(\langle \mathsf{Lin'}_{f^4},\ \mathcal{C}^4 \rangle + \mathsf{Quad'}\big(\mathcal{E}^2(c_i^1) \cdot \mathcal{E}^2(c_j^1 \cdot s)\ \big)\Big) \\
&\quad - u_{\ell'}^3 \Big(\langle \mathsf{Lin''}_{f^4},\ \mathcal{C}^4 \rangle + \mathsf{Quad''}\big(\mathcal{E}^2(c_i^1) \cdot \mathcal{E}^2(c_j^1 \cdot s)\ \big)\Big) \\
&= \langle \mathsf{Lin'''}_{f^4},\ \mathcal{C}^4 \rangle + \mathsf{Poly}_{f^4}(\mathcal{C}^1, \mathcal{C}^2, \mathcal{C}^3)
\end{aligned}$$

for some linear functions $\mathsf{Lin'}_{f^4}, \mathsf{Lin''}_{f^4}, \mathsf{Lin'''}_{f^4}$ and quadratic functions $\mathsf{Quad'}$, $\mathsf{Quad''}$ and polynomial Poly_{f^4}.

Thus, we have computed $\mathcal{E}^4(c_\ell^3 s + p_1 \cdot \mu_\ell^2)$ and hence, c^4 by Equation 5.11. The final public key for c^4 is given by:

$$\mathsf{PK}(c^4) = u_\ell^3 u_{\ell'}^3\ \mathsf{PK}(\mathcal{E}^4(s^2)) - u_{\ell'}^3\ \mathsf{PK}(\mathcal{E}^4(c_\ell^3 s)) - u_\ell^3\ \mathsf{PK}(\mathcal{E}^4(c_{\ell'}^3 s)) \tag{5.17}$$

$\mathcal{E}^4(c^3)$ and $\mathcal{E}^4(c_i^1 c_j^1)$ are computed analogously. Thus, we have established correctness of the base case.

Note. In the base case, we see that each time the quadratic method is applied to compute an encoding of a product of two messages, we end up with an encoding of the desired product plus noise.

Induction Step. Assume that the claim is true for level $k-1$. Then we establish that it is true for level k.

By the I.H, we have that:

1. We can compute $\mathcal{E}^{k-1}(c^{k-2} \cdot s)$ and $\mathcal{E}^{k-1}(c^{k-2})$ by taking linear combinations of elements in $\mathcal{C}^{k-1}$ and quadratic terms of the form $\mathcal{E}^{k-2}(y_i)\mathcal{E}^{k-2}(y_j)$ for some y_i, y_j of the form c_i^{k-4}, $c_j^{k-4}\ s$.

2. We can compute c^{k-1}.

Computing $\mathcal{E}^k(c^{k-1} \cdot s)$. We claim that:

Claim 5.4. *The term $\mathcal{E}^k(c_\ell^{k-1} s)$ (hence c^k) can be computed as a linear combination of elements in $\mathcal{C}^k$ and quadratic terms of the form $\mathcal{E}^{k-1}(\cdot) \cdot \mathcal{E}^{k-1}(\cdot)$.*

Proof. The term $\mathcal{E}^k(c^{k-1} \cdot s)$ may be written as:

$$\begin{aligned}\mathcal{E}^k(c^{k-1} \cdot s) &= \mathcal{E}^k\Big((c_i^{k-2}\ c_j^{k-2} - u_i^{k-2}\mathcal{E}^{k-1}(c_j^{k-2} \cdot s) - u_j^{k-2}\mathcal{E}^{k-1}(c_i^{k-2} \cdot s) + u_i^{k-2}u_j^{k-2}\mathcal{E}^{k-1}(s^2)\ ) \cdot s\Big) \\ &= \mathcal{E}^k(c_i^{k-2}\ c_j^{k-2}\ s) - u_i^{k-2}\mathcal{E}^k\big(\mathcal{E}^{k-1}(c_j^{k-2} \cdot s) \cdot s\big) \\ &\quad - u_j^{k-2}\mathcal{E}^k\big(\mathcal{E}^{k-1}(c_i^{k-2} \cdot s) \cdot s\big) + u_i^{k-2}u_j^{k-2}\mathcal{E}^k\big(\mathcal{E}^{k-1}(s^2) \cdot s\big) \qquad (5.18)\end{aligned}$$

Consider $\mathcal{E}^k\big(\mathcal{E}^{k-1}(s^2) \cdot s\big)$. Since $\mathcal{E}^{k-1}(s^2) \in \mathcal{C}^{k-1}$ and $\mathcal{E}^k\big(\mathcal{C}^{k-1} \cdot s\big)$ is contained in $\mathcal{C}^k$, we have that $\mathcal{E}^k\big(\mathcal{E}^{k-1}(s^2) \cdot s\big) \in \mathcal{C}^k$.

Consider the term $\mathcal{E}^k(c_i^{k-2}\ c_j^{k-2}\ s)$. We may compute $\mathcal{E}^k(c_i^{k-2}\ c_j^{k-2}\ s)$ using the quadratic method as:

$$\begin{aligned}\mathcal{E}^k(c_i^{k-2}\ c_j^{k-2}\ s) &= \Big(\mathcal{E}^{k-1}(c_i^{k-2}) \cdot \mathcal{E}^{k-1}(c_j^{k-2} \cdot s)\Big)\ + \mathsf{PK}\big(\mathcal{E}^{k-1}(c_i^{k-2})\big)\mathsf{PK}\big(\mathcal{E}^{k-1}(c_j^{k-2} \cdot s)\big)\ \mathcal{E}^k(s^2) \\ &\quad - \mathsf{PK}\big(\mathcal{E}^{k-1}(c_i^{k-2})\big)\Big(\mathcal{E}^k\big(\mathcal{E}^{k-1}(c_j^{k-2} \cdot s) \cdot s\big)\Big) - \mathsf{PK}\big(\mathcal{E}^{k-1}(c_j^{k-2} \cdot s)\big)\Big(\mathcal{E}^k\big(\mathcal{E}^{k-1}(c_i^{k-2}) \cdot s\big)\Big) \qquad (5.19)\end{aligned}$$

Thus, to compute $\mathcal{E}^k(c^{k-1} \cdot s)$, it suffices to compute the term $\mathcal{E}^k(c_i^{k-2}\ c_j^{k-2}\ s)$ since the additional terms such as $\mathcal{E}^k\big(\mathcal{E}^{k-1}(c_i^{k-2} \cdot s) \cdot s\big)$ that appear in Equation 5.18 also appear in Equation 5.19 and will be computed in the process of computing $\mathcal{E}^k(c_i^{k-2}\ c_j^{k-2}\ s)$.

Note. We observe that in Equation 5.19, by "factoring out" the quadratic term $\mathcal{E}^{k-1}(c_i^{k-2}) \cdot \mathcal{E}^{k-1}(c_j^{k-2} \cdot s)$, we reduce the computation of $\mathcal{E}^k(c_i^{k-2}\ c_j^{k-2}\ s)$ to $\mathcal{E}^k\big(\mathcal{E}^{k-1}(c_j^{k-2} \cdot s) \cdot s\big)$ where the latter value has half the degree of the former at the cost of adding one more level of nesting and a new multiplication by s. By recursively applying Equation 5.19, we will obtain d quadratic encodings

in level $k-1$ and a linear term in level k advice encodings $\mathcal{C}^k$.

Proceeding, we see that to compute $\mathcal{E}^k(c_i^{k-2}\ c_j^{k-2}\ s)$, we are required to compute the following terms:

1. $\mathcal{E}^{k-1}(c_i^{k-2})$ and $\mathcal{E}^{k-1}(c_j^{k-2}\cdot s)$. These can be computed by the induction hypothesis using linear combinations of elements in $\mathcal{C}^{k-1}$ and quadratic terms of the form $\mathcal{E}^{k-2}(y_i)\mathcal{E}^{k-2}(y_j)$ for some y_i, y_j. Since the precise linear coefficients are not important, we shall denote:
$$\mathcal{E}^{k-1}(c_j^{k-2}\cdot s) = \mathsf{LinComb}\big(\mathcal{C}^{k-1}, \mathcal{E}^{k-2}(\cdot)\mathcal{E}^{k-2}(\cdot)\big) \tag{5.20}$$

2. $\mathcal{E}^k\big(\mathcal{E}^{k-1}(c_i^{k-2})\cdot s\big)$ and $\mathcal{E}^k\big(\mathcal{E}^{k-1}(c_j^{k-2}\cdot s)\cdot s\big)$: Consider the latter term (the former can be computed analogously).

   By the induction hypothesis,
$$\begin{aligned}\mathcal{E}^k\big(\mathcal{E}^{k-1}(c_j^{k-2}\cdot s)\cdot s\big) &= \mathcal{E}^k\Big(\mathsf{LinComb}\big(\mathcal{C}^{k-1}, \mathcal{E}^{k-2}(\cdot)\mathcal{E}^{k-2}(\cdot)\big)\cdot s\Big)\\ &= \mathcal{E}^k\Big(\mathsf{LinComb}\big(\mathcal{C}^{k-1}\cdot s\big)\Big) + \mathcal{E}^k\Big(\mathsf{LinComb}\big(\mathcal{E}^{k-2}(y_a)\mathcal{E}^{k-2}(y_b)\cdot s\big)\Big)\\ &= \mathsf{LinComb}\Big(\mathcal{E}^k\big(\mathcal{C}^{k-1}\cdot s\big)\Big) + \mathsf{LinComb}\Big(\mathcal{E}^k\big(\mathcal{E}^{k-2}(y_a)\mathcal{E}^{k-2}(y_b)\cdot s\big)\Big)\end{aligned} \tag{5.21}$$

   Again, we note that the terms $\mathcal{E}^k\big(\mathcal{C}^{k-1}\cdot s\big) \in \mathcal{C}^k$ by definition hence it remains to construct $\mathcal{E}^k\Big(\big(\mathcal{E}^{k-2}(y_i)\mathcal{E}^{k-2}(y_j)\big)\cdot s\Big)$.

   Thus, we have reduced the computation of $\mathcal{E}^k(c_i^{k-2}\ c_j^{k-2}\ s)$ to the computation of terms of the form $\mathcal{E}^k\big(\mathcal{E}^{k-2}(y_a)\mathcal{E}^{k-2}(y_b)\cdot s\big)$ for some y_a, y_b, by Equation 5.21[7]. Hence, expanding $c_i^{k-2}\ c_j^{k-2}\ s$ yields a degree 4 term in c^{k-3} whereas expanding the latter only yields a quadratic term in c^{k-3}.

   To proceed, again, we will consider $z_i = \mathcal{E}^{k-2}(y_i)$ and $z_j = \mathcal{E}^{k-2}(y_j)\cdot s$ as messages and apply the quadratic method to compute an encoding of their product. In more detail,
$$\begin{aligned}&\mathcal{E}^k\Big(\big(\mathcal{E}^{k-2}(y_i)\mathcal{E}^{k-2}(y_j)\big)\cdot s\Big)\\ &= \mathsf{LinComb}\Big(\mathcal{E}^{k-1}(\mathcal{E}^{k-2}(y_i))\cdot\mathcal{E}^{k-1}(\mathcal{E}^{k-2}(y_j)\cdot s),\ \mathcal{E}^k\big(\mathcal{E}^{k-1}(\mathcal{E}^{k-2}(y_i))\cdot s\big),\ \mathcal{E}^k\big(\mathcal{E}^{k-1}(\mathcal{E}^{k-2}(y_j)\cdot s)\cdot s\big)\Big)\end{aligned} \tag{5.22}$$

   Thus, we are required to compute:

   (a) $\mathcal{E}^{k-1}(\mathcal{E}^{k-2}(y_i))$, $\mathcal{E}^{k-1}(\mathcal{E}^{k-2}(y_j)\cdot s)$: These can be computed via the induction hypothesis.

[7] We note that while c_i^{k-2} is also a degree $k-2$ encoding, the difference between c_i^{k-2} and $\mathcal{E}^{k-2}(y_a)$ is that the former contains a quadratic term $c_{i'}^{k-3}c_{j'}^{k-3}$ for some i', j' while in the latter, y_a is of the form $c^{k-3}s$, which has half the nested message degree as the former.

(b) $\mathcal{E}^k\Big(\mathcal{E}^{k-1}\big(\mathcal{E}^{k-2}(y_i)\big)\cdot s\Big)$ and $\mathcal{E}^k\big(\mathcal{E}^{k-1}(\mathcal{E}^{k-2}(y_j)\cdot s)\cdot s\big)$: Consider the latter term (the former may be computed analogously). Note that

$$\mathcal{E}^{k-2}(y_j) = \mathsf{LinComb}\big(\mathcal{C}^{k-2}, \mathcal{E}^{k-3}(\cdot)\mathcal{E}^{k-3}(\cdot)\big)$$

$$\text{Hence, } \mathcal{E}^k\big(\mathcal{E}^{k-1}(\mathcal{E}^{k-2}(y_j)\cdot s)\cdot s\big) = \mathcal{E}^k\Big(\mathcal{E}^{k-1}(\mathsf{LinComb}\big(\mathcal{C}^{k-2}, \mathcal{E}^{k-3}(\cdot)\mathcal{E}^{k-3}(\cdot)\big)\cdot s)\cdot s\Big)$$

Again, $\mathcal{E}^k(\mathcal{E}^{k-1}(\mathcal{C}^{k-2}\cdot s)\cdot s)\in\mathcal{C}^k$ so we are left with the computation of

$$\begin{aligned}\mathcal{E}^k\Big(\mathcal{E}^{k-1}(\mathcal{E}^{k-3}(\cdot)\mathcal{E}^{k-3}(\cdot)\cdot s)\cdot s\Big) &= \mathcal{E}^k\Big(\mathsf{LinComb}\Big(\mathcal{E}^{k-2}\big(\mathcal{E}^{k-3}(\cdot)\cdot s\big)\cdot\mathcal{E}^{k-2}(\mathcal{E}^{k-3}(\cdot)),\\ &\qquad \mathcal{E}^{k-1}\big(\mathcal{E}^{k-2}\big(\mathcal{E}^{k-3}(\cdot)\cdot s\big)\cdot s\big)\Big)\Big)\\ &= \mathsf{LinComb}\Big(\mathcal{E}^{k-1}\big(\mathcal{E}^{k-2}\big(\mathcal{E}^{k-3}(\cdot)\cdot s\big)\big)\cdot\mathcal{E}^{k-1}\big(\mathcal{E}^{k-2}\big(\mathcal{E}^{k-3}(\cdot)\cdot s\big)\cdot s\big),\\ &\qquad \mathcal{E}^k\Big(\mathcal{E}^{k-1}\big(\mathcal{E}^{k-2}\big(\mathcal{E}^{k-3}(\cdot)\cdot s\big)\cdot s\big)\cdot s\Big)\cdot s\Big)\end{aligned}$$

Proceeding recursively, we see that $\mathcal{E}^k\big(\mathcal{E}^{k-1}(\mathcal{E}^{k-2}(y_j)\cdot s)\cdot s\big)$ contains a linear combination of quadratic terms of the form $\mathcal{E}^{k-1}(\cdot)\mathcal{E}^{k-1}(\cdot)$ for each level and nested encodings of the form $\mathcal{E}^k\Big(\mathcal{E}^{k-1}\big(\mathcal{E}^{k-2}(\cdots)\cdot s\big)\cdot s\Big)$. At the last level, we obtain nested encodings which are contained in $\mathcal{C}^k$ by construction. Hence we may compute $\mathcal{E}^k\big(\mathcal{E}^{k-1}(\mathcal{E}^{k-2}(y_j)\cdot s)\cdot s\big)$ as a linear combination of quadratic terms of the form $\mathcal{E}^{k-1}(\cdot)\mathcal{E}^{k-1}(\cdot)$ and linear terms in $\mathcal{C}^k$. Note that the public key $\mathsf{PK}(\mathcal{E}^k(c^{k-1}\cdot s))$ can be computed as a linear combination of the public keys $\mathsf{PK}(\mathcal{C}^k)$, as in Equation 5.16.

$$\mathsf{PK}(\mathcal{E}^k(c^{k-1}\cdot s)) = \mathsf{LinComb}(\mathsf{PK}(\mathcal{C}^k)) \tag{5.23}$$

Note that for the public key computation, the higher degree encoding computations are not relevant as these form the message of the final level k encoding.

□

Computing level k ciphertext. Next, we have that:

$$\begin{aligned}c_t^k &= c_\ell^{k-1}c_{\ell'}^{k-1} + \mathcal{E}^k\big(u_\ell^{k-1}u_{\ell'}^{k-1}(s^2) - u_{\ell'}^{k-1}(c_\ell^{k-1}s) - u_\ell^{k-1}(c_{\ell'}^{k-1}s)\big)\\ &= c_\ell^{k-1}c_{\ell'}^{k-1} + u_\ell^{k-1}u_{\ell'}^{k-1}\ \mathcal{E}^k(s^2) - u_{\ell'}^{k-1}\ \mathcal{E}^k(c_\ell^{k-1}s) - u_\ell^{k-1}\ \mathcal{E}^k(c_{\ell'}^{k-1}s)\end{aligned} \tag{5.24}$$

Similarly,

$$\mathsf{PK}(c_t^k) = u_\ell^{k-1}u_{\ell'}^{k-1}\ \mathsf{PK}(\mathcal{E}^k(s^2)) - u_{\ell'}^{k-1}\ \mathsf{PK}\big(\mathcal{E}^k(c_\ell^{k-1}s)\big) - u_\ell^{k-1}\ \mathsf{PK}\big(\mathcal{E}^k(c_{\ell'}^{k-1}s)\big) \tag{5.25}$$

Public Key and Ciphertext Structure. From the above, we claim:

Claim 5.5. *The public key for c_t^k (for any t) is a publicly computable linear combination of public keys of level k encodings $\mathsf{PK}(\mathcal{E}^k(s^2))$ and $\mathsf{PK}\big(\mathcal{E}^k(c_\ell^{k-1}s)\big)$ for all ℓ*

Regarding the ciphertext, since we computed $\mathcal{E}^k(c_\ell^{k-1}s)$ from $\mathcal{C}^k$ above, and c^{k-1} may be computed via the induction hypothesis, we may compute c^k as desired. Moreover, since $\mathcal{E}^k(c_\ell^{k-1}s)$ is linear in level k encodings and has quadratic terms in level $k-1$ encodings, we get by unrolling the recursion that $\mathcal{E}^k(c_\ell^{k-1}s)$ and hence level k ciphertext c^k is linear in level k encodings and polynomial in lower level encodings $\mathcal{C}^1, \ldots, \mathcal{C}^{k-1}$. Hence, we have that:

$$\begin{aligned} c^k = \mathsf{CT}(f^k(\mathbf{x}) + \mu^k_{f(\mathbf{x})}) &= \langle \mathsf{Lin}_{f^k}, \mathcal{C}^k \rangle + \mathsf{LinComb}\big(\mathsf{Quad}(\mathcal{E}^{k-1}(y_i)\ \mathcal{E}^{k-1}(y_j))\big) \\ &= \langle \mathsf{Lin}_{f^k}, \mathcal{C}^k \rangle + \mathsf{Poly}_{f^k}\big(\mathcal{C}^1, \ldots, \mathcal{C}^{k-1}\big) \end{aligned}$$

The Public Key and Ciphertext Evaluation Algorithms. Our evaluation algorithms $\mathsf{Eval}_{\mathsf{PK}}$ and $\mathsf{Eval}_{\mathsf{CT}}$ are defined recursively, so that to compute the functional public key and functional ciphertext at level k, the algorithms require the same for level $k-1$.

$\mathbf{Eval}^k_{\mathsf{PK}}(\bigcup_{i\in[k]} \mathsf{PK}(\mathcal{C}^i), \ell)$: To compute the label for the ℓ^{th} wire in the level k circuit, do:

1. If the ℓ^{th} wire at level k is the addition of the i^{th} and j^{th} wire at level $k-1$, then do the following:
   - If $k = 3$ (base case), then compute $\mathsf{PK}(c_\ell^3) = \mathsf{PK}(c_i^2) + \mathsf{PK}(c_j^2)$ as in Equation 5.10.
   - Let $\mathsf{PK}_i^{k-1} = \mathsf{Eval}^{k-1}_{\mathsf{PK}}(\bigcup_{j\in[k-1]} \mathsf{PK}(\mathcal{C}^j), i)$ and $\mathsf{PK}_j^{k-1} = \mathsf{Eval}^{k-1}_{\mathsf{PK}}(\bigcup_{i\in[k-1]} \mathsf{PK}(\mathcal{C}^i), j)$,
   - Let $\mathsf{PK}_\ell^k = \mathsf{PK}_i^{k-1} + \mathsf{PK}_j^{k-1}$
2. If the ℓ^{th} wire at level k is the multiplication of the i^{th} and j^{th} wire at level $k-1$, then do the following:
   - If $k = 4$ (base case), then compute PK_ℓ^k as described in Equation 5.17.
   - Let $u_i^{k-1} = \mathsf{Eval}^{k-1}_{\mathsf{PK}}(\bigcup_{j\in[k-1]} \mathsf{PK}(\mathcal{C}^j), i)$ and $u_j^{k-1} = \mathsf{Eval}^{k-1}_{\mathsf{PK}}(\bigcup_{i\in[k-1]} \mathsf{PK}(\mathcal{C}^i), j)$,
   - Let $\mathsf{PK}(c_\ell^k) = u_i^{k-1}u_j^{k-1}\ \mathsf{PK}(\mathcal{E}^k(s^2)) - u_j^{k-1}\ \mathsf{PK}\big(\mathcal{E}^k(c_i^{k-1}s)\big) - u_i^{k-1}\ \mathsf{PK}\big(\mathcal{E}^k(c_j^{k-1}s)\big)$ as in Equation 5.25. Here $\mathsf{PK}(\mathcal{E}^k(s^2))$, $\mathsf{PK}\big(\mathcal{E}^k(c_i^{k-1}s)\big)$ and $\mathsf{PK}\big(\mathcal{E}^k(c_j^{k-1}s)\big)$ are computed using $\mathcal{C}^k$ as described in Equation 5.16, 5.23.

$\mathbf{Eval}^k_{\mathsf{CT}}(\bigcup_{i\in[k]} \mathcal{C}^i, \ell)$ To compute the encoding for the ℓ^{th} wire in the level k circuit, do:

1. If the ℓ^{th} wire at level k is the addition of the i^{th} and j^{th} wire at level $k-1$, then do the following:
   - If $k = 3$ (base case), then compute $c_\ell^3 = c_i^2 + c_j^2$ as in Equation 5.9.
   - Let $\mathsf{CT}_i^{k-1} = \mathsf{Eval}^{k-1}_{\mathsf{CT}}(\bigcup_{j\in[k-1]} \mathcal{C}^j, i)$ and $\mathsf{CT}_j^{k-1} = \mathsf{Eval}^{k-1}_{\mathsf{CT}}(\bigcup_{i\in[k-1]} \mathcal{C}^i, j)$,
   - Let $\mathsf{CT}_\ell^k = \mathsf{CT}_i^{k-1} + \mathsf{CT}_j^{k-1}$
2. If the ℓ^{th} wire at level k is the multiplication of the i^{th} and j^{th} wire at level $k-1$, then do the following:

- If $k = 4$ (base case) then compute c_ℓ^4 (for any ℓ) using Equations 5.11 and 5.12.
- Let $c_i^{k-1} = \mathsf{Eval}_{\mathsf{CT}}^{k-1}(\underset{j\in[k-1]}{\cup} \mathcal{C}^j, i)$ and $c_j^{k-1} = \mathsf{Eval}_{\mathsf{CT}}^{k-1}(\underset{i\in[k-1]}{\cup} \mathcal{C}^i, j)$,
- Let $c_\ell^k = c_i^{k-1}c_j^{k-1} + u_i^{k-1}u_j^{k-1}\ \mathcal{E}^k(s^2) - u_j^{k-1}\ \mathcal{E}^k(c_i^{k-1}s) - u_i^{k-1}\ \mathcal{E}^k(c_j^{k-1}s)$ as in Equation 5.24. Here, the terms $\mathcal{E}^k(s^2)$, $\mathcal{E}^k(c_i^{k-1}s)$ and $\mathcal{E}^k(c_j^{k-1}s)$ are computed using $\mathcal{C}^k$ as described in claim 5.4

Remark. We note that the above $\mathsf{Eval}_{\mathsf{CT}}$ and $\mathsf{Eval}_{\mathsf{PK}}$ algorithms are generalisations of the corresponding algorithms in the LinFE construction (please see Section 3 for details) and produce ciphertext and public key with *exactly* the same structure as the evaluated LinFE ciphertext and public key. Indeed, the GenKey and Decode algorithms defined in Section 3 work correctly if invoked with outputs of $\mathsf{Eval}_{\mathsf{CT}}$ and $\mathsf{Eval}_{\mathsf{PK}}$ described above.

6 Succinct Functional Encryption for NC_1.

In this section, we extend the construction for quadratic functional encryption provided in Section 4 to circuits of depth $O(\log n)$. The construction generalises directly the QuadFE scheme using the public key and ciphertext evaluation algorithms from the previous section. We make non black box use of the LinFE scheme, please see Section 3 for definitions of algorithms GenKey and Decode.

We proceed to describe the construction.

$\mathsf{Poly.Setup}(1^\lambda, 1^w, 1^d)$: Upon input the security parameter λ, the message dimension w, and the circuit depth d, do:

1. For $k \in [d]$, let $L_k = |\mathcal{C}^k|$ where $\mathcal{C}^k$ is as defined in Theorem 5.2. For $k \in [d-1]$, $i \in [L_k]$, choose uniformly random $u_{i,k} \in R_{p_k}$. Denote $\mathbf{u}_k = (u_{i,k}) \in R_{p_k}^{L_k}$.
2. Invoke $\mathsf{LinFE.Setup}(1^\lambda, 1^{L_d}, p_d)$ to obtain $\mathsf{PK} = \mathsf{LinFE.PK}$ and $\mathsf{MSK} = \mathsf{LinFE.MSK}$. Parse $\mathsf{LinFE.PK} = (\mathbf{w}, \mathbf{u}_d)$.
3. Output $\mathsf{PK} = (\mathbf{w}, \mathbf{u}_1, \ldots, \mathbf{u}_d)$ and $\mathsf{MSK} = \mathsf{LinFE.MSK}$.

$\mathsf{Poly.KeyGen}(\mathsf{MSK}, f))$: Upon input the master secret key MSK and a circuit f of depth d, do:

1. Let $\mathsf{PK}_f = \mathsf{Eval}_{\mathsf{PK}}(\mathsf{PK}, f)$.
2. Invoke $\mathsf{GenKey}(\mathsf{MSK}, \mathsf{PK}_f)$ as described in Section 3 to obtain $\mathbf{k}_f$ and output it. In more detail, the algorithm chooses a short vector $\mathbf{k}_f$ such that $\langle \mathbf{w}, \mathbf{k}_f \rangle = \mathsf{PK}_f \mod p_d$ and outputs it.

$\mathsf{Poly.Enc}(\mathbf{x}, \mathsf{PK})$: Upon input the public key and the input $\mathbf{x}$, do:

1. Compute the encodings $\mathcal{C}^k$ for $k \in [d]$ as defined in Theorem 5.2 using the LinFE encoding function $\mathcal{E}$ described in Section 3. Denote by s the LWE secret used for these encodings.
2. Let $\mathbf{d} = \mathbf{w} \cdot s + \eta$ for some noise η. Note that the LWE secret s is the same as that used in computing encodings $\mathcal{C}^k$ for $k \in [d]$.

3. Output $\mathsf{CT}_{\mathbf{x}} = (\{\mathcal{C}^k\}_{k\in[d]}, \mathbf{d})$.

Note that an equivalent way to perform the first two steps above is to compute encodings $\mathcal{C}^k$ for $k \in [d-1]$ as defined in Theorem 5.2 and use $\mathsf{LinFE.Enc}(\mathsf{PK}, \mathcal{C}^{d-1}, \mathcal{C}^{d-1} \cdot s)$ to obtain the randomness encoding $\mathbf{d}$ and the level d encodings $\mathcal{C}^d$.

$\mathsf{Poly.Dec}(\mathsf{PK}, \mathsf{CT}_{\mathbf{x}}, \mathsf{SK}_f)$: Upon input a ciphertext $\mathsf{CT}_{\mathbf{x}}$ for vector $\mathbf{x}$, and a secret key $\mathsf{SK}_f = \mathbf{k}_f$ for circuit f, do:

1. Compute $\mathsf{CT}_{f(\mathbf{x})} = \mathsf{Eval}_{\mathsf{CT}}(\{\mathcal{C}^k\}_{k\in[d]}, f)$.
2. Compute $\mathsf{Decode}(\mathbf{k}_f, \mathsf{CT}_{f(\mathbf{x})}) = \langle \mathbf{k}, \mathbf{d}\rangle - \mathsf{CT}_{f(\mathbf{x})} \mod p_d \mod p_{d-1} \ldots \mod p_0$ and output it.

Correctness follows from correctness of $\mathsf{Eval}_{\mathsf{PK}}$ and $\mathsf{Eval}_{\mathsf{CT}}$. In more detail, we have that $\langle \mathbf{w}, \mathbf{k}_f\rangle = \mathsf{PK}_f \mod p_d$ by construction. Hence,

$$\langle \mathbf{k}_f, \mathbf{d}\rangle = \mathsf{PK}_f \cdot s + p_{k-1} \cdot \eta'_{k-1} \mod p_d$$

Additionally, by correctness of $\mathsf{Eval}_{\mathsf{PK}}$ and $\mathsf{Eval}_{\mathsf{CT}}$ algorithms, we have that

$$\mathsf{CT}_{f(\mathbf{x})} = \mathsf{PK}_f \cdot s + p_{k-1} \cdot \eta_{k-1} + \ldots + p_0 \cdot \eta_0 + f(\mathbf{x})$$

Hence,

$$\langle \mathbf{k}_f, \mathbf{d}\rangle - \mathsf{CT}_{f(\mathbf{x})} \mod p_d \mod p_{d-1} \ldots \mod p_0 = f(\mathbf{x})$$

as desired.

Analysis of Ciphertext Structure. Note that the ciphertext comprises message dependent encodings $\mathcal{C}^k$ for $k \in [d]$ and a randomness encoding $\mathbf{d}$. Since each message dependent encoding depends only on a single bit of the message, the ciphertext is decomposable (in the sense defined in Section 3). We note that this makes our ciphertext enjoy local-updates: if a single bit of the message changes, then only $O(d)$ encodings need updating, not the entire ciphertext.

Also, since the LinFE ciphertext is succinct as described in Section 3, the data dependent component of our ciphertext is also succinct.

Ring versus Standard LWE. The above construction may also be based on standard LWE as against Ring LWE, but at the cost of a larger ciphertext. Basing the construction on standard LWE incurs a factor n blowup in the ciphertext size at each level, and this restricts the depth of the circuit being computed to (any) constant. Please see Appendix C for details.

The proof of security is analogous to that of the quadratic scheme. We prove security for the case of a single key request, the case of bounded keys can be handled exactly as in Section 4.2.

Theorem 6.1. *The construction in Section 6 achieves full simulation based security as per definition 2.2.*

Proof. We describe our simulator.

Simulator $\mathsf{Sim}(1^\lambda, 1^{|\mathbf{x}|}, \mathsf{PK}, f, \mathsf{SK}_f, f(\mathbf{x}))$**.** The simulator given input the security parameter, length of message $\mathbf{x}$, the circuit f, the secret key SK_f and the value $f(\mathbf{x})$ does the following:

1. It computes $\mathsf{PK}_f = \mathsf{Eval}_{\mathsf{PK}}(\mathsf{PK}, f)$. Note that by claim 5.5, PK_f is a f-dependent linear combination of the level d public key components $\mathbf{u}_d$. Denote this linear function as $\mathbf{g}_f$.
2. It samples all encodings upto level $d-1$ randomly, i.e. $\mathcal{C}^k \leftarrow R_{p_k}^{L_k}$. Treating level d encodings $\mathcal{C}^d$ as formal variables, compute $\mathsf{CT}_{f(\mathbf{x})} = \mathsf{Eval}_{\mathsf{CT}}\big(\ \{\mathcal{C}^k\}_{k\in[d]}, f\ \big)$. Note that by the structure of $\mathsf{Eval}_{\mathsf{CT}}$ (please see Section 5), we have that $\mathsf{CT}_{f(\mathbf{x})}$ may be expressed as $z_{\mathrm{poly}} + z_{\mathsf{lin}}$ where z_{poly} is the high degree polynomial value obtained by evaluating on level $k < d-1$ encoding values and $z_{\mathsf{lin}} = \langle \mathbf{g}_f, \mathcal{C}^d \rangle$ is the linear function $\mathbf{g}_f$ evaluated on level d encoding variables $\mathcal{C}^d$.
3. It samples $\eta \leftarrow \mathcal{D}_d$ and computes $d' = f(\mathbf{x}) - z_{\mathrm{poly}} + \eta$ and invokes the single key LinFE simulator with input d'.
4. It sets as $\mathbf{b}$ the output received by the LinFE simulator.
5. It outputs $\mathsf{CT}_{\mathbf{x}} = (\{\mathcal{C}^k\}_{k\in[d]}, \mathbf{b})$.

We will prove that the output of the simulator is indistinguishable from the real world via a sequence of hybrids.

The Hybrids. Our Hybrids are described below.

Hybrid 0. This is the real world.

Hybrid 1. In this hybrid, the only thing that is different is that $\mathbf{b}$ is computed using the LinFE simulator In more detail,

- It computes $\mathsf{CT}_{f(\mathbf{x})} = \mathsf{Eval}_{\mathsf{CT}}\big(\ \{\mathcal{C}^k\}_{k\in[d]}, f\ \big)$.
- It parses
$$\mathsf{CT}_{f(\mathbf{x})} = \mathsf{PK}_f \cdot s + p_{k-1} \cdot \eta_{k-1} + p_{k-2} \cdot \eta_{k-2} + \ldots + p_0 \cdot \eta_0 + f(\mathbf{x})$$
for some noise terms $\eta_{k-1}, \ldots, \eta_0$. Here $p_{k-1} \cdot \eta_{k-1}$ is treated as the noise in the encoding and $p_{k-2} \cdot \eta_{k-2} + \ldots + p_0 \cdot \eta_0 + f(\mathbf{x})$ as the message.
- It alternately parses $\mathsf{CT}_{f(\mathbf{x})} = z_{\mathrm{poly}} + z_{\mathsf{lin}}$ where z_{poly} is the high degree polynomial obtained by evaluating on level $k < d-1$ encodings and $z_{\mathsf{lin}} = \langle \mathbf{g}_f, \mathcal{C}^d \rangle$ is the linear function $\mathbf{g}_f$ evaluated on level d encodings $\mathcal{C}^d$.
- It computes $d' = p_{k-2} \cdot \eta_{k-2} + \ldots + p_0 \cdot \eta_0 + f(\mathbf{x}) - z_{\mathrm{poly}}$.
- It samples η^* such that
$$\mathsf{SD}\big(\eta^*, p_{k-2} \cdot \eta_{k-2} + \ldots + p_0 \cdot \eta_0\big) \leq \mathrm{negl}(\lambda) \tag{6.1}$$
and invokes the single key LinFE simulator with input $d' + \eta^*$.

Hybrid 2. In this hybrid, let $d' = f(\mathbf{x}) - z_{\text{poly}}$ and invoke the LinFE simulator with $d' + \eta^*$.

Hybrid 3. In this hybrid, sample $\mathcal{C}^k$ for $k \in [d-1]$ at random. This is the simulated world.

Indistinguishability of Hybrids proceeds exactly as in Section 4. The only difference is indistinguishability of Hybrids 0 and 1. In Hybrid 0, LinFE decryption yields the value $p_{k-2} \cdot \eta_{k-2} + \ldots + p_0 \cdot \eta_0 + f(\mathbf{x}) + \eta^* - z_{\text{poly}}$ which is what is used to invoke the LinFE simulator in Hybrid 1. Hence, by security of LinFE, we have that Hybrids 0 and 1 are indistinguishable. Hybrids 1 and 2 are statistically indistinguishable by Equation 6.1. Hybrids 2 and 3 are indistinguishable due to semantic security of Regev encodings.

□

7 Putting it together : Bounded Collusion FE for all circuits

In this section, we describe how to put together the pieces from the previous sections to build a bounded collusion FE scheme for all circuits in P, denoted by BddFE. The approach follows the (by now) standard bootstrapping method of using low depth randomized encodings to represent any polynomial sized circuit. This approach was first suggested by Gorbunov et al. [GVW12], who show that q query FE for degree three polynomials can be bootstrapped to q query FE for all circuits.

At a high level, their approach can be summarized as follows. Let $\mathcal{C}$ be a family of polynomial sized circuits. Let $C \in \mathcal{C}$ and let $\mathbf{x}$ be some input. Let $\tilde{C}(\mathbf{x}, R)$ be a randomized encoding of C that is computable by a constant depth circuit with respect to inputs x and R. Then consider a new family of circuits $\mathcal{G}$ defined by:

$$G_{C,\Delta}(\mathbf{x}, R_1, \ldots, R_S) = \tilde{C}\Big(\mathbf{x}; \bigoplus_{a \in \Delta} R_a\Big)$$

Note that $G_{C,\Delta}(\cdot,\cdot)$ is computable by a degree three polynomial, one for each output bit. Given an FE scheme for $\mathcal{G}$, one may construct a scheme for $\mathcal{C}$ by having the decryptor first recover the output of $G_{C,\Delta}(\mathbf{x}, R_1, \ldots, R_S)$ and then applying the decoder for the randomized encoding to recover $C(\mathbf{x})$. Since our construction from Section 6 is capable of evaluating degree 3 polynomials, it suffices for bootstrapping, to yield q-query FE for all circuits. We will denote this scheme by PolyFE.

As in [GVW12, ALS16], let (S, v, m) be parameters to the construction. Let Δ_i for $i \in [q]$ be a uniformly random subset of $[S]$ of size v. To support q queries, the key generator identifies the set $\Delta_i \subseteq [S]$ with query i. If $v = O(\lambda)$ and $S = O(\lambda \cdot q^2)$ then the sets Δ_i are cover free with high probability as shown by [GVW12]. Let $L \triangleq (\ell^3 + S \cdot m)$.

$\mathsf{BddFE.Setup}(1^\lambda, 1^\ell)$: Upon input the security parameter λ and the message space $\{0,1\}^\ell$, invoke $(\mathsf{mpk}, \mathsf{msk}) = \mathsf{PolyFE.Setup}(1^\lambda, 1^L)$ and output it.

$\mathsf{BddFE.KeyGen}(\mathsf{msk}, C)$): Upon input the master secret key and a circuit C, do:

1. Choose a uniformly random subset $\Delta \subseteq [S]$ of size v.
2. Express $C(\mathbf{x})$ by $G_{C,\Delta}(\mathbf{x}, R_1, \ldots, R_S)$, which in turn can be expressed as a sequence of degree 3 polynomials $P_1, \ldots, P_k$, where $k \in \mathsf{poly}(\lambda)$.

3. Set $\mathsf{BddFE.SK}_C = \{\mathsf{SK}_i = \mathsf{PolyFE.KeyGen}(\mathsf{PolyFE.msk}, P_i)\}_{i\in[k]}$ and output it.

$\mathsf{BddFE.Enc}(\mathbf{x}, \mathsf{mpk})$: Upon input the public key and the input $\mathbf{x}$, do:

1. Choose $R_1, \ldots, R_S \leftarrow \{0,1\}^m$, where m is the size of the random input in the randomized encoding.
2. Set $\mathsf{CT}_{\mathbf{x}} = \mathsf{PolyFE.Enc}(\mathsf{PolyFE.mpk}, \mathbf{x}, R_1, \ldots, R_s)$ and output it.

$\mathsf{BddFE.Dec}(\mathsf{mpk}, \mathsf{CT}_{\mathbf{x}}, \mathsf{SK}_C)$: Upon input a ciphertext $\mathsf{CT}_{\mathbf{x}}$ for vector $\mathbf{x}$, and a secret key SK_C for circuit C, do the following:

1. Compute $G_{C,\Delta}(\mathbf{x}, R_1, \ldots, R_S) = \mathsf{PolyFE.Dec}(\mathsf{CT}_{\mathbf{x}}, \mathsf{SK}_C)$.
2. Run the Decoder for the randomized encoding to recover $C(\mathbf{x})$ from $G_{C,\Delta}(\mathbf{x}, R_1, \ldots, R_S)$.

Correctness follows immediately from the correctness of PolyFE and the correctness of randomized encodings.

Note that the above construction can be based on standard rather than ring LWE, since it only requires PolyFE for depth 3 circuits to apply the bootstrapping technique. As discussed in Appendix C, PolyFE can be constructed using standard LWE for constant depth circuits.

7.1 Security

we argue that the construction is secure. The proof follows easily from the security of randomized encodings and of the PolyFE scheme.

Let us assume that the randomized encodings are secure. Then, given an attacker $\mathcal{A}$ who breaks the security of the BddFE, we construct an attacker $\mathcal{B}$ who breaks the security of PolyFE as follows. $\mathcal{B}$ does the following:

1. $\mathcal{B}$ obtains the public key PK from the PolyFE challenger and returns this to $\mathcal{A}$.
2. $\mathcal{A}$ outputs challenges $\mathbf{x}_0, \mathbf{x}_1$. $\mathcal{B}$ chooses the randomness $\mathbf{R} = R_1, \ldots, R_S$ and returns $(\mathbf{x}_0, \mathbf{R})$ and $(\mathbf{x}_1, \mathbf{R})$ as the challenge messages to the PolyFE challenger.
3. The PolyFE challenger returns the challenge CT, which $\mathcal{B}$ returns to $\mathcal{A}$.
4. $\mathcal{B}$ picks $\Delta_1, \ldots, \Delta_q \subseteq S$ randomly, of size v each. When $\mathcal{A}$ makes a query for a circuit C_i, where $i \in [Q]$, $\mathcal{B}$ converts it to a tuple of degree 3 polynomials $G_{C_i,\Delta_i}(\cdot)$ and sends these to the challenger. It forwards the challenger's response to $\mathcal{A}$.
5. When $\mathcal{A}$ outputs a bit, $\mathcal{B}$ does the same.

Note that by security of randomized encodings, $\mathcal{B}$ is an admissible adversary. If $\mathcal{B}$ is admissible, then the advantage of $\mathcal{B}$ is exactly the advantage of $\mathcal{A}$.

Ciphertext Structure. We note that the BddFE ciphertext is a PolyFE ciphertext for the message vector $(\mathbf{x}, R_1, \ldots, R_S)$. Since the PolyFE ciphertext is decomposable, and has a succinct data dependent component, the same features are inherited by the BddFE ciphertext.

Distributed Data Applications. Note that since BddFE CT is decomposable and online succinct, k parties holding k pieces of data $x_1, \ldots, x_k$ can share a PRF seed to generate the common randomness required to tie together the components of the ciphertext. Given this seed, they can independently encode their individual inputs x_i to construct CT_i and upload these to a common server. We note that the data independent offline component $\mathsf{CT}_{\mathsf{indpt}}$ may be generated by a designated party with a high bandwidth upload link, or by the server itself.

Acknowledgements. We thank Damien Stehlé and Chris Peikert for helpful discussions.

REFERENCES AND SUPPLEMENTARY MATERIAL

References

[ABB10] Shweta Agrawal, Dan Boneh, and Xavier Boyen. Efficient lattice (H)IBE in the standard model. In *EUROCRYPT*, pages 553–572, 2010.

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

[AFV11] Shweta Agrawal, David Mandell Freeman, and Vinod Vaikuntanathan. Functional encryption for inner product predicates from learning with errors. In *Asiacrypt*, 2011.

[Agr17] Shweta Agrawal. Stronger security for reusable garbled circuits, general definitions and attacks. In *Crypto*, 2017.

[AGVW13] Shweta Agrawal, Sergey Gurbanov, Vinod Vaikuntanathan, and Hoeteck Wee. Functional encryption: New perspectives and lower bounds. In *Crypto*, 2013.

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

[AIK11] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. How to garble arithmetic circuits. In *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011*, pages 120–129, 2011.

[AIK14] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. How to garble arithmetic circuits. *SIAM J. Comput.*, 43(2):905–929, 2014.

[AIKW15] Benny Applebaum, Yuval Ishai, Eyal Kushilevitz, and Brent Waters. Encoding functions with constant online rate, or how to compress garbled circuit keys. *SIAM Journal on Computing*, 44(2):433–466, 2015.

[AJ15] Prabhanjan Ananth and Abhishek Jain. Indistinguishability obfuscation from compact functional encryption. In *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part I*, pages 308–326, 2015.

[AJS15] Prabhanjan Ananth, Abhishek Jain, and Amit Sahai. Indistinguishability obfuscation from functional encryption for simple functions. Eprint 2015/730, 2015.

[ALS16] Shweta Agrawal, Benoit Libert, and Damien Stehlé. Fully secure functional encryption for linear functions from standard assumptions, and applications. In *CRYPTO*, 2016. https://eprint.iacr.org/2015/608.

[BF01] Dan Boneh and Matthew K. Franklin. Identity-based encryption from the weil pairing. In *CRYPTO*, pages 213–229, 2001.

[BGG+14] Dan Boneh, Craig Gentry, Sergey Gorbunov, Shai Halevi, Valeria Nikolaenko, Gil Segev, Vinod Vaikuntanathan, and Dhinakaran Vinayagamurthy. Fully key-homomorphic encryption, arithmetic circuit abe and compact garbled circuits. In Phong Q. Nguyen and Elisabeth Oswald, editors, *Eurocrypt*, 2014.

[BGV12] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (Leveled) fully homomorphic encryption without bootstrapping. In *ITCS*, pages 309–325, 2012.

[BSW07] John Bethencourt, Amit Sahai, and Brent Waters. Ciphertext-policy attribute-based encryption. In *IEEE Symposium on Security and Privacy*, pages 321–334, 2007.

[BSW11] Dan Boneh, Amit Sahai, and Brent Waters. Functional encryption: Definitions and challenges. In *TCC*, pages 253–273, 2011.

[BV11a] Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) LWE. In *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011*, pages 97–106, 2011.

[BV11b] Zvika Brakerski and Vinod Vaikuntanathan. Fully homomorphic encryption from ring-lwe and security for key dependent messages. In *Advances in Cryptology - CRYPTO 2011 - 31st Annual Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2011. Proceedings*, pages 505–524, 2011.

[BV15] Nir Bitansky and Vinod Vaikuntanathan. Indistinguishability obfuscation from functional encryption. *IACR Cryptology ePrint Archive*, 2015:163, 2015.

[BW06] Xavier Boyen and Brent Waters. Anonymous hierarchical identity-based encryption (without random oracles). In *CRYPTO*, pages 290–307, 2006.

[BW07] Dan Boneh and Brent Waters. Conjunctive, subset, and range queries on encrypted data. In *TCC*, pages 535–554, 2007.

[CC17] Ran Canetti and Yilei Chen. Constraint-hiding constrained prfs for nc1 from lwe. In *Eurocrypt*, 2017.

[CFL+16] Jung Hee Cheon, Pierre-Alain Fouque, Changmin Lee, Brice Minaud, and Hansol Ryu. Cryptanalysis of the new clt multilinear map over the integers. In *Eurocrypt*, 2016.

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

[CHH+07] Ronald Cramer, Goichiro Hanaoka, Dennis Hofheinz, Hideki Imai, Eike Kiltz, Rafael Pass, Abhi Shelat, and Vinod Vaikuntanathan. Bounded cca2-secure encryption. In *Proceedings of the Advances in Crypotology 13th International Conference on Theory and Application of Cryptology and Information Security*, ASIACRYPT'07, 2007.

[CHKP10] David Cash, Dennis Hofheinz, Eike Kiltz, and Chris Peikert. Bonsai trees, or how to delegate a lattice basis. In *EUROCRYPT*, pages 523–552, 2010.

[CHL+15] J.-H. Cheon, K. Han, C. Lee, H. Ryu, and D. Stehlé. Cryptanalysis of the multilinear map over the integers. In *Proc. of EUROCRYPT*, volume 9056 of *LNCS*, pages 3–12. Springer, 2015.

[CJL] Jung Hee Cheon, Jinhyuck Jeong, and Changmin Lee. An algorithm for ntru problems and cryptanalysis of the ggh multilinear map without a low level encoding of zero. Eprint 2016/139.

[CLT13] Jean-Sébastien Coron, Tancrède Lepoint, and Mehdi Tibouchi. Practical multilinear maps over the integers. In *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part I*, pages 476–493, 2013.

[Coc01] Clifford Cocks. An identity based encryption scheme based on quadratic residues. In *IMA Int. Conf.*, pages 360–363, 2001.

[DKXY02] Yevgeniy Dodis, Jonathan Katz, Shouhuai Xu, and Moti Yung. Key-insulated public key cryptosystems. In *Proceedings of the International Conference on the Theory and Applications of Cryptographic Techniques: Advances in Cryptology*, EUROCRYPT '02, 2002.

[GGG+14] Shafi Goldwasser, S. Dov Gordon, Vipul Goyal, Abhishek Jain, Jonathan Katz, Feng-Hao Liu, Amit Sahai, Elaine Shi, and Hong-Sheng Zhou. Multi-input functional encryption. In *EUROCRYPT*, pages 578–602, 2014.

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

[GGH+13b] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *FOCS*, 2013. http://eprint.iacr.org/.

[GGH+13c] Sanjam Garg, Craig Gentry, Shai Halevi, Amit Sahai, and Brent Waters. Attribute-based encryption for circuits from multilinear maps. In *CRYPTO*, 2013.

[GGH15] Craig Gentry, Sergey Gorbunov, and Shai Halevi. Graph-induced multilinear maps from lattices. In *Theory of Cryptography - 12th Theory of Cryptography Conference, TCC 2015, Warsaw, Poland, March 23-25, 2015, Proceedings, Part II*, pages 498–527, 2015.

[GGHZ14] Sanjam Garg, Craig Gentry, Shai Halevi, and Mark Zhandry. Fully secure functional encryption without obfuscation. In *IACR Cryptology ePrint Archive*, volume 2014, page 666, 2014.

[GKP+13] Shafi Goldwasser, Yael Tauman Kalai, Raluca A. Popa, Vinod Vaikuntanathan, and Nickolai Zeldovich. Reusable garbled circuits and succinct functional encryption. In *STOC*, pages 555–564, 2013.

[GKPV10] Shafi Goldwasser, Yael Tauman Kalai, Chris Peikert, and Vinod Vaikuntanathan. Robustness of the learning with errors assumption. In *ITCS*, 2010.

[GLW12] Shafi Goldwasser, Allison Lewko, and David Wilson. Bounded-collusion ibe from key homomorphism. In *Theory of Cryptography*, Lecture Notes in Computer Science, 2012.

[GPSW06] Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent Waters. Attribute-based encryption for fine-grained access control of encrypted data. In *ACM Conference on Computer and Communications Security*, pages 89–98, 2006.

[GPV08] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In *STOC*, pages 197–206, 2008.

[GVW12] Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Functional encryption with bounded collusions from multiparty computation. In *CRYPTO*, 2012.

[GVW13] Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Attribute based encryption for circuits. In *STOC*, 2013.

[GVW15] Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Predicate encryption for circuits from lwe. In *CRYPTO*, 2015.

[HJ16] Y. Hu and H. Jia. Cryptanalysis of GGH map. In *Eurocrypt*, 2016.

[HKK+14] Yan Huang, Jonathan Katz, Vladimir Kolesnikov, Ranjit Kumaresan, and Alex J Malozemoff. Amortizing garbled circuits. In *International Cryptology Conference*, pages 458–475. Springer, 2014.

[KSW08] Jonathan Katz, Amit Sahai, and Brent Waters. Predicate encryption supporting disjunctions, polynomial equations, and inner products. In *EUROCRYPT*, pages 146–162, 2008.

[LOS+10] Allison B. Lewko, Tatsuaki Okamoto, Amit Sahai, Katsuyuki Takashima, and Brent Waters. Fully secure functional encryption: Attribute-based encryption and (hierarchical) inner product encryption. In *EUROCRYPT*, pages 62–91, 2010.

[LPR10] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. In *EUROCRYPT*, volume 6110, 2010.

[LR14] Yehuda Lindell and Ben Riva. Cut-and-choose yao-based secure computation in the online/offline and batch settings. In *International Cryptology Conference*, pages 476–494. Springer, 2014.

[MR07] Daniele Micciancio and Oded Regev. Worst-case to average-case reductions based on gaussian measures. *SIAM Journal on Computing (SICOMP)*, 37(1):267–302, 2007. extended abstract in FOCS 2004.

[MSZ16] Eric Miles, Amit Sahai, and Mark Zhandry. Annihilation attacks for multilinear maps: Cryptanalysis of indistinguishability obfuscation over ggh13. In *Crypto*, 2016.

[O'N10] Adam O'Neill. Definitional issues in functional encryption. Cryptology ePrint Archive, Report 2010/556, 2010. http://eprint.iacr.org/.

[Reg09] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. *J.ACM*, 56(6), 2009. extended abstract in STOC'05.

[SS10] 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*, CCS '10, 2010.

[SW] Amit Sahai and Brent Waters. Functional encryption:beyond public key cryptography. Power Point Presentation, 2008. http://userweb.cs.utexas.edu/bwaters/presentations/files/functional.ppt.

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

[Wat12] Brent Waters. Functional encryption for regular languages. In *Crypto*, 2012.

Appendix

A Previous Constructions for Bounded Collusion FE

In this section, we discuss previous constructions for Bounded Collusion FE.

A.1 The GVW12 Construction

In this section we sketch the GVW scheme and discuss why it is unsuitable for the applications discussed in Section 1. The scheme can be summarized as follows.

- The first ingredient they need is a single key FE scheme for all circuits. A construction for this was provided by Sahai and Seyalioglu in [SS10].
- Next, the single FE scheme is generalized to a q query scheme for NC_1 circuits. This gerenalization is fairly complex, we provide an outline here. At a high level, they run N copies of the single key scheme, where $N = O(q^4)$. The encryptor encrypts the views of the BGW MPC protocol for N parties, computing some functionality related to C. They rely on the fact that BGW is non-interactive when used to compute bounded degree functions. To generate a secret key, KeyGen chooses a random subset of the single query FE keys, where the parameters are set so that the subsets have small pairwise intersections. This subset of keys enables the decryptor to recover sufficiently many shares of $C(x)$ which allows him to recover $C(x)$. [GVW12] argue that an attacker with q keys only learns a share x_i when two subsets of keys intersect, but since the subsets were chosen to have small pairwise intersections, this does not occur often enough to recover enough shares of x. Finally, by the security of secret sharing, x remains hidden.

- As the last step they "bootstrap" the q query FE for NC_1 to q query FE for all circuits using computational randomized encodings [AIK06]. They must additionally use cover free sets to ensure that fresh randomness is used for each randomized encoding.

Thus, to encrypt a message x, the encryptor must secret share it into $N = O(q^4)$ shares, and encrypt each one with the one query FE. Since they use Shamir secret sharing with polynomial of degree t and $t = O(q^2)$, note that at most $O(q^2)$ shares can be generated offline, since $t+1$ points will determine the polynomial. Hence $O(q^4)$ shares must be generated in the online phase. This results in an online encryption time that degrades as $O(q^4)$.

A.2 The ALS16 construction

[ALS16] provide a conceptually simpler way to build q-query Functional Encryption for all circuits. Their construction replaces steps 1 and 2 described above with a inner product modulo p FE scheme, and then uses step 3 as in [GVW12]. Thus, the construction of single key FE in step 1 by Sahai and Seyalioglu, and the nontrivial "MPC in the head" of step 2 can both be replaced by the simple abstraction of an inner product FE scheme. For step 3, observe that the bootstrapping theorem of [GVW12] provides a method to bootstrap an FE for NC_1 that handles q queries to an FE for all polynomial-size circuits that is also secure against q queries. The bootstrapping relies on the result of Applebaum *et al.* [AIK06, Theorem 4.11] which states that every polynomial time computable function f admits a perfectly correct computational randomized encoding of degree 3.

In more detail, let $\mathcal{C}$ be a family of polynomial-size circuits. Let $C \in \mathcal{C}$ and let x be some input. Let $\widetilde{C}(x, R)$ be a randomized encoding of C that is computable by a constant depth circuit with respect to inputs x and R. Then consider a new family of circuits $\mathcal{G}$ defined by:

$$G_{C,\Delta}(x, R_1, \ldots, R_S) = \left\{ \widetilde{C}\Big(x; \bigoplus_{a\in\Delta} R_a\Big) : \ C \in \mathcal{C}, \ \Delta \subseteq [S] \right\},$$

for some sufficiently large S (quadratic in the number of queries q). As observed in [GVW12], circuit $G_{C,\Delta}(\cdot,\cdot)$ is computable by a constant degree polynomial (one for each output bit). Given an FE scheme for $\mathcal{G}$, one may construct a scheme for $\mathcal{C}$ by having the decryptor first recover the output of $G_{C,\Delta}(x, R_1, \ldots, R_S)$ and then applying the decoder for the randomized encoding to recover $C(x)$.

However, to support q queries the decryptor must compute q randomized encodings, each of which needs fresh randomness. This is handled by hardcoding S random elements in the ciphertext and using random subsets $\Delta \subseteq [S]$ (which are cover-free with overwhelming probability) to compute fresh randomness $\bigoplus_{a\in\Delta} R_a$ for every query. [ALS16] observe that bootstrapping only requires support for the particular circuit class $\mathcal{G}$ described above. This circuit class, being computable by degree 3 polynomials, may be supported by a linear FE scheme, via linearization of the degree 3 polynomials.

Putting it together, the encryptor encrypts all degree 3 monomials in the inputs $R_1, \ldots, R_S$ and $x_1, \ldots, x_\ell$. Note that this ciphertext is polynomial in size. Now, for a given circuit C, the keygen algorithm samples some $\Delta \subseteq [S]$ and computes the symbolic degree 3 polynomials which must be released to the decryptor. It then provides the linear FE keys to compute the same. By correctness and security of Linear FE as well as the randomizing polynomial construction, the decryptor learns $C(x)$ and nothing else.

Note that in this construction the challenge of supporting multiplication is sidestepped by merely having the encryptor encrypt each monomial $x_i x_j$ separately so that the FE need only support

addition. This "brute force" approach incurs several disadvantages. For instance, decomposability is lost – even though the ciphertext can be decomposed into $|\mathbf{x}|^2$ components, any input bit x_1 (say) must feature in $|\mathbf{x}|$ ciphertext components $x_1x_2, \ldots, x_1x_w$, where $w = |\mathbf{x}|$. This makes the scheme inapplicable for all applications involving distributed data, where a centre or a sensor device knows a bit x_i but is oblivious to the other bits. Additionally, the scheme is not online-offline, since all the ciphertext components depend on the data, hence the entire encryption operation must be performed after the data becomes available. For applications where a centre or sensor must transmit data-dependent ciphertext after the data is observed, this incurs a significant cost in terms of bandwidth. Indeed, the work performed by the sensor device in computing the data dependent ciphertext becomes proportional to the size of the function being computed on the data, which may be infeasible for weak devices.

Another approach to obtain bounded collusion FE is to compile the single key FE of Goldwasser et al [GKP+13] with the compiler of [GVW12] to support Q queries. Again, this approach yields succinct CTs but the CT grows as $O(q^4)$ rather than $O(q^2)$ as in our scheme.

B Distributed Data Applications.

Our construction is very suitable for applications where data belonging to a single party is distributed across multiple locations. As an example, consider a large healthcare organization with $k > 1$ branches across multiple cities, which desires to store data from all centres in a central repository and compute functions on this. Ideally, we would like each centre to upload encryptions of its data to the server independently of other centres. Our construction provides a natural solution – all k centres may share a PRF seed, using which centre i can construct ciphertext component CT_i that depends only on its data $\mathbf{x}_i$ and upload this. The size of the ciphertext component CT_i is $\mathrm{poly}(\lambda, |\mathbf{x}_i|)$, where λ is the security parameter, hence the upload operation consumes optimal bandwidth. The ciphertext components $\mathsf{CT}_1, \ldots, \mathsf{CT}_k$ can then be combined to produce a ciphertext $\mathsf{CT}_{\mathbf{x}}$ which may be decrypted by a function key SK_g to obtain $g(\mathbf{x})$. See Section 7 for more details. Note that decomposability does not imply the stronger notion of multi-input functional encryption [GGG+14], since in the former case, only "matching" ciphertext components (i.e. those that share randomness) can be combined, whereas in the latter, any combination of ciphertexts constructed by the multiple parties are valid input to the function key.

As another example consider the following scenario described by Applebaum et al. [AIK14]: a computationally weak device is sent to the field in order to perform some expensive computation C on some sensitive data $\mathbf{x}$. The computation is too expensive for the device to perform it on its own, but since the data is sensitive, the device cannot send the data back in the clear. As noted by [AIK14], garbled circuits provide the only feasible non-interactive solution to this question: the weak device can compute the garbled circuit for circuit C in an offline phase before going to the field, and keep the garbled keys $\{K_i^0, K_i^1\}_{i \in [|\mathbf{x}|]}$ with itself. Once it learns the input $\mathbf{x}$, it performs the lightweight operation of selecting the appropriate keys $K_i^{x_i}$ and sends these back. Note that this application crucially relies on succinctness of the "online" or data-dependent component of the encoding $K_i^{x_i}$, so that transmission cost is minimized. Indeed, reducing the online complexity of randomized encodings, for both single and multiple executions of the circuit, is an active area of research – see [LR14, HKK+14, AIKW15] and references therein. Also, note that *decomposability* of garbled circuits enables supporting multiple sensor devices: since the encoding $K_i^{x_i}$ depends only on

bit x_i and none other, device i may be given keys (K_i^0, K_i^1) before it is sent to the field, and the function may be computed on data collected by multiple devices.

Since functional encryption may be viewed as a generalization of randomized encodings, many applications of randomized encodings benefit immediately from the additional power offered by FE. For instance, by replacing the decomposable, online-succinct garbled circuit in the above example by a decomposable, online-succinct functional encryption scheme, we may: 1) choose the function to be computed on the data *after* the devices are sent to the field, 2) a given function may be computed on an unbounded number of ciphertexts, and 3) we may compute multiple (i.e. q) functions on a given set of data. It is evident that these capabilities are not enjoyed by the garbled circuit protocol described above. Also note that since computing CT_i is the online part of the operation and is efficient, it is within the capabilities of the weak device.

See Section 7 for more details about how our scheme achieves these properties and Appendix A for a discussion on why previous schemes could not.

C Construction from Standard LWE

In this section, we briefly discuss how the scheme presented in Section 4 can be modified to rely on standard LWE rather than ring LWE. The modification is straightforward: instead of requiring LinFE to compute a noisy linear function over a polynomial ring, we now compute a noisy linear function over a field $\mathbb{Z}_q$.

In more detail, the message carrier (for a vector $\mathbf{x} \in \mathbb{Z}_{p_0}^w$ in the encrypt algorithm is now a vector:

$$\mathbf{c} = \mathbf{U}^\mathsf{T}\mathbf{s}^1 + p \cdot \boldsymbol{\mu} + \mathbf{x} \in \mathbb{Z}_q^w.$$

Here, $\mathbf{U} = (\mathbf{u}_i) \in \mathbb{Z}_q^{n \times w}$, $\mathbf{s}^1 \leftarrow \mathbb{Z}_q^n$ and $\boldsymbol{\mu} \in \mathbb{Z}^w$ is appropriately sampled noise.

Let us examine what we require for correctness.

Let $1 \le j \le i \le w$. By definition

$$x_i + p \cdot \mu_i = c_i - \langle \mathbf{u}_i; \mathbf{s}^1 \rangle,$$
$$x_j + p \cdot \mu_j = c_j - \langle \mathbf{u}_j; \mathbf{s}^1 \rangle.$$

Letting $\mu_{ij} = x_i\mu_j + x_j\mu_i + \mu_i\mu_j$, we have

$$x_i x_j + p \cdot \mu_{ij} = c_i c_j - c_i\langle \mathbf{u}_j; \mathbf{s}^1 \rangle - c_j\langle \mathbf{u}_i; \mathbf{s}^1 \rangle + \langle \mathbf{u}_i; \mathbf{s}^1 \rangle\langle \mathbf{u}_j; \mathbf{s}^1 \rangle,$$

which equals

$$c_i c_j - \sum_{k\in[n]}\sum_{\tau\in[T]} u_{jk\tau}(2^\tau c_i s_k^1) - \sum_{k\in[n]}\sum_{\tau\in[T]} u_{ik\tau}(2^\tau c_j s_k^1) + \sum_{k,\ell\in[n]}\sum_{\tau\in[T]} (u_{ik}u_{j\ell})_\tau (2^\tau s_k^1 s_\ell^1). \tag{C.1}$$

Clearly

$$-\sum_{k\in[n]}\sum_{\tau\in[T]} u_{jk\tau}(2^\tau c_i s_k^1) - \sum_{k\in[n]}\sum_{\tau\in[T]} u_{ik\tau}(2^\tau c_j s_k^1) + \sum_{k,\ell\in[n]}\sum_{\tau\in[T]} (u_{ik}u_{j\ell})_\tau (2^\tau s_k^1 s_\ell^1). \tag{C.2}$$

is a linear equation which can be computed by the linear inner product scheme LinFE.

In more detail, LinFE encrypt algorithm encrypts messages $\{2^\tau c_i s_k^1 \in \mathbb{Z}_q\}$ and $\{2^\tau s_k^1 s_\ell^1 \in \mathbb{Z}_q\}$ for $k, \ell \in [n], \tau \in [\log q], i \in [w]$ while the key generator provides a key corresponding to vector

$$\left(\mathbf{0},\ (u_{ik}u_{j\ell})_\tau\ , \mathbf{0}, -u_{ik\tau},\ \ \mathbf{0},\ -u_{jk\tau}\ \mathbf{0}\right)$$

so that the equation C.2 may be computed. The remaining details are easily translated from the ring to the standard setting by carefully replacing ring products by field inner products.

Note that the c_i is multiplied by n elements $s_1^1, \ldots, s_n^1$ to encode at level 2, this causes a factor n blowup at each level. The "dimension reduction" trick developed in the context of fully homomorphic encryption [BV11a, BGV12] is not immediately applicable in our context, since it is not the keys that are growing in size, but the messages that are being encrypted at each level. While there may be more sophisticated techniques to perform dimension reduction in our setting, we do not explore this here and leave it to future work.

Hence, from standard LWE, the ciphertext grows by a factor n at each level and this forces the construction in Section 6 to be restricted to d levels for any constant d.

D Making KeyGen stateless for QuadFE

We note that the keygen algorithm can be made stateless by using standard tricks, as in [GVW12, Section 6]. To see this, let us examine how noise is added in the above system at present. The encryptor provides encryptions of Q noise terms $\boldsymbol{\eta}$ and for $i \in [Q]$, the key generator appends the i^{th} key vector $\mathbf{g}_i$ with a Q sized unit vector $\mathbf{e}_i$. As a result, decryption of the i^{th} key with the ciphertext results in an additional term $\boldsymbol{\eta}^\intercal \mathbf{e}_i = \eta_i$, which gets added to the decryption value $\mathbf{g}_i^{\mathbf{z}}$. Thus, to ensure that a fresh noise term η_i is added for each decryption equation, the key generator must keep track of how many keys it has issued in the past (or receive this information as input).

Cover free sets offer a natural tool to deal with this issue, and this technique was used towards a similar end in [GVW12]. The idea is to choose $Q' > Q$ and have the encryptor provide encryptions of Q' noise values in place of Q. The key generator is modified to be stateless and randomly pick a Q' sized binary vector of weight v in place of $\mathbf{e}_i$. Now the decryption computes a random subset sum of Q' noise terms for every key, which for appropriate setting of Q' and v guarantees that each decryption equation has at least one fresh noise term. It is shown in [GVW12] that $Q' = Q^2$ and $v = \lambda$ suffices.

The careful reader might notice that the summands in this case are discrete Gaussians and a random subset of discrete Gaussians does not yield the original distribution. However, fortunately this is not an issue, since the cover free property implies that every decryption equation has at least one freshly sampled discrete Gaussian that is not used in any other equation. This suffices for security.

E Parameters

In this section, we discuss the parameters for our constructions. We denote the magnitude of noise used in the level i encodings by B_i. We require $B_i \leq O(p_i/4)$ at every level for correct decryption. We have that the message space for level 1 encodings $\mathcal{E}^1$ is R_{p_0} and encoding space is R_{p_1}. Then message space for $\mathcal{E}^2$ is $O(p_0^2 + B_1^2) = O(B_1^2)$ since the noise at level 1 is a multiple of p_0. Then, p_2 must be chosen as $O(B_1^2)$. At the next multiplication level, i.e. level 4, we have the message space as $O(p_2^2 + B_2^2) = O(B_1^4)$. In general, for d levels, it suffices to set $p_d = O(B^{2^d})$.

We also require all the distinct moduli to be relatively prime, hence we choose all the moduli to be prime numbers of the aforementioned size.

We must also choose the size of the noise that is added for flooding. As described in Section 4, for quadratic polynomials we require $\frac{L \cdot p_0 \cdot \sigma_0^2}{\sigma_1} = \mathrm{negl}(\lambda)$ where σ_1 is the standard deviation for the noise η_i for $i \in [Q]$ encoded in the ciphertext. For depth d (Section 6), we require $\frac{L_d \cdot \prod_{i \in [0,d]} p_i \cdot B^{2^d}}{\sigma} = \mathrm{negl}(\lambda)$ where σ is the standard deviation of the noise η encoded in the ciphertext.

Since $L_d = \mathrm{poly}(\lambda)$ by definition, we require

$$\sigma \geq O(\mathrm{poly}(\lambda) B^{2^{d+1}}$$

We may set $p_0 = n$ with initial noise level as $B_1 = \mathrm{poly}(n)$ and any $B_i, p_i = O(B_1^{2^i})$. Also, the number of encodings provided at level d is $L_d = O(2^d)$, so in general we may let $d = O(\log n)$, thus supporting the circuit class NC_1.

We note that by the definition of efficiency of reusable garbled circuits [GKP+13], it suffices to have ciphertext size that is sublinear in circuit size, which is achieved by our construction.