(Leveled) Fully Homomorphic Encryption without Bootstrapping

Zvika Brakerski[*] Stanford University

Craig Gentry[†] IBM Research

Vinod Vaikuntanathan[‡] University of Toronto

Abstract

We present a novel approach to fully homomorphic encryption (FHE) that dramatically improves performance and bases security on weaker assumptions. A central conceptual contribution in our work is a new way of constructing leveled fully homomorphic encryption schemes (capable of evaluating arbitrary polynomial-size circuits), *without Gentry's bootstrapping procedure.*

Specifically, we offer a choice of FHE schemes based on the learning with error (LWE) or Ring LWE (RLWE) problems that have 2^λ security against known attacks. We construct:

- A leveled FHE scheme that can evaluate depth-L arithmetic circuits (composed of fan-in 2 gates) using $\tilde{O}(\lambda \cdot L^3)$ per-gate computation. That is, the computation is *quasi-linear* in the security parameter. Security is based on RLWE for an approximation factor exponential in L. This construction does not use the bootstrapping procedure.
- A leveled FHE scheme that can evaluate depth-L arithmetic circuits (composed of fan-in 2 gates) using $\tilde{O}(\lambda^2)$ per-gate computation, which is *independent of L*. Security is based on RLWE for *quasi-polynomial* factors. This construction uses bootstrapping *as an optimization.*

We obtain similar results for LWE, but with worse performance. All previous (leveled) FHE schemes required a per-gate computation of $\tilde{\Omega}(\lambda^{3.5})$, and all of them relied on sub-exponential hardness assumptions.

We introduce a number of further optimizations to our scheme based on the Ring LWE assumption. As an example, for circuits of large width – e.g., where a constant fraction of levels have width $\Omega(\lambda)$ – we can reduce the per-gate computation of the bootstrapped version to $\tilde{O}(\lambda)$, independent of L, by *batching the bootstrapping operation.*

At the core of our construction is a much more effective approach for managing the noise level of lattice-based ciphertexts as homomorphic operations are performed, using new techniques recently introduced by Brakerski and Vaikuntanathan (FOCS 2011).

[*]E-mail: `zvika@stanford.edu`. Supported by a Simons Postdoctoral Fellowship.

[†]E-mail: `cbgentry@us.ibm.com`. Sponsored by the Air Force Research Laboratory (AFRL). Disclaimer: This material is based on research sponsored by DARPA under agreement number FA8750-11-C-0096. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of DARPA or the U.S. Government. Approved for Public Release, Distribution Unlimited.

[‡]E-mail: `vinodv@cs.toronto.edu`. This material is based on research sponsored by DARPA under Agreement number FA8750-11-2-0225. All disclaimers as above apply.

1 Introduction

Fully homomorphic encryption (FHE) [RAD78, Gen09b] allows a computationally powerful worker to receive encrypted data and perform arbitrarily complex, dynamically chosen computations on that data while it remains encrypted, despite not having the secret decryption key. Until recently, all FHE schemes [Gen09b, DGHV10, SV10, GH11b, CMNT, BV11a] followed the same blueprint, namely, the one laid out in Gentry's original construction [Gen09b, Gen09a].

The first step in Gentry's blueprint is to construct a *somewhat homomorphic encryption (SWHE) scheme*, namely an encryption scheme capable of evaluating "low-degree" multivariate polynomials homomorphically. Starting with Gentry's original construction based on ideal lattices [Gen09b], there are by now a number of such schemes in the literature [DGHV10, SV10, GH11b, CMNT, BV11a, LNV11], all of which based on lattices (either directly or implicitly). The ciphertexts in all these schemes are "noisy", where the noise grows slightly during homomorphic addition and explosively during homomorphic multiplication, and hence, the limitation of low-degree polynomials.

To obtain FHE, Gentry provided a remarkable *bootstrapping theorem* which states that given a SWHE scheme that can evaluate its own decryption function (plus an additional operation), one can transform it into a "leveled"[1] FHE scheme. Bootstrapping "refreshes" a ciphertext by running the decryption function on it homomorphically using an encrypted secret key (given in the public key), resulting in a reduced noise.

Thus, to finish the construction, it is sufficient to design a SWHE scheme that is capable of homomorphically evaluating its own decryption circuit (plus some). Unfortunately, until very recently, natural SWHE schemes used to be incapable of evaluating their own decryption circuits without making significant modifications to the scheme. (We discuss recent exceptions [GH11a, BV11b] below.) Thus, the final step in Gentry's blueprint is to *squash the decryption circuit* of the SWHE scheme, namely transform the scheme into one with the same homomorphic capacity but a decryption circuit that is simple enough to allow bootstrapping. Gentry [Gen09b] showed how to do this by adding a "hint" – namely, a large set of numbers with a secret sparse subset that sums to the original secret key – to the public key. Of course, the hint can be seen as useful information about the secret key, and the security of the scheme in the presence of the hint relies on a new "sparse subset sum" assumption (which, roughly speaking, can be thought of as saying that the hint is useless to a computationally bounded adversary).

1.1 Efficiency of FHE

The efficiency of fully homomorphic encryption has been a (perhaps, *the*) big question following its invention. In this paper, we are concerned with the *per-gate computation overhead* of the FHE scheme, defined as the ratio between the time it takes to compute a circuit homomorphically on encrypted inputs to the time it takes to compute it on plaintext inputs.[2] Unfortunately, FHE schemes that follow Gentry's blueprint (some of which have actually been implemented [GH11b, CMNT]) have fairly poor performance: their per-gate computation overhead is $p(\lambda)$, a large polynomial in

[1]In a "leveled" FHE scheme, the parameters of the scheme may depend on the *depth* of the circuits that the scheme can evaluate (but not on their size). The schemes we construct in this work are all leveled FHE schemes. One can obtain a "pure" FHE scheme (with a constant-size public key) from these leveled FHE schemes by assuming "circular security", namely that it is "safe" to encrypt the leveled FHE secret key under its own public key. With this understanding, and when there is no cause for confusion, we will omit the term "leveled" throughout this work.

[2]Other measures of efficiency, such ciphertext/key size and encryption/decryption time, are also important. In fact, the schemes we present in this paper are very efficient in these aspects (as are the schemes in [GH11a, BV11b]).

the security parameter. In fact, as we argue below, this penalty in performance seems somewhat inherent for schemes that follow this blueprint.

First, the complexity of (known approaches to) bootstrapping is *inherently* at least the complexity of decryption *times* the bit-length of the individual ciphertexts that are used to encrypt the bits of the secret key. The reason is that bootstrapping involves evaluating the decryption circuit *homomorphically* – that is, in the decryption circuit, each secret-key bit is replaced by a (large) ciphertext that encrypts that bit – and both the complexity of decryption and the ciphertext lengths must each be $\Omega(\lambda)$.

Second, the undesirable properties of known SWHE schemes conspire to ensure that *the real cost of bootstrapping for FHE schemes that follow this blueprint is actually much worse than quadratic.* Known FHE schemes start with a SWHE scheme that can evaluate polynomials of degree D (multiplicative depth $\log D$) securely only if the underlying lattice problem is hard to 2^D-approximate. To achieve hardness against 2^λ time adversaries, the lattice must have dimension $\Omega(D \cdot \lambda)$. This is because we have lattice algorithms in n dimensions that compute $2^{n/\lambda}$-approximations of short vectors in time $2^{\widetilde{O}(\lambda)}$. Moreover, the coefficients of the vectors used in the scheme have bit length $\Omega(D)$ to allow the ciphertext noise room to expand to 2^D. Therefore, the size of "fresh" ciphertexts (e.g., those that encrypt the bits of the secret key) is $\tilde{\Omega}(D^2 \cdot \lambda)$. Since the SWHE scheme must be "bootstrappable" – i.e., capable of evaluating its own decryption function – D must exceed the degree of the decryption function. Typically, the degree of the decryption function is $\Omega(\lambda)$. Thus, overall, "fresh" ciphertexts have size $\tilde{\Omega}(\lambda^3)$. So, the real cost of bootstrapping – even if we optimistically assume that the "stale" ciphertext that needs to be refreshed can be decrypted in only $\Theta(\lambda)$-time – is $\tilde{\Omega}(\lambda^4)$.

The analysis above ignores a nice optimization by Stehlé and Steinfeld [SS10], which so far has not been useful in practice, that uses Chernoff bounds to asymptotically reduce the decryption degree down to $O(\sqrt{\lambda})$. With this optimization, the per-gate computation of FHE schemes that follow the blueprint is $\tilde{\Omega}(\lambda^3)$.[3]

1.2 Recent Deviations from Gentry's Blueprint, and the Hope for Better Efficiency

Recently, Gentry and Halevi [GH11a], and Brakerski and Vaikuntanathan [BV11b], independently found very different ways to construct FHE without using the squashing step, and thus without the sparse subset sum assumption. These schemes are the first major deviations from Gentry's blueprint for FHE. Surprisingly, Brakerski and Vaikuntanathan [BV11b] showed how to base security entirely on LWE (for sub-exponential approximation factors), avoiding reliance on ideal lattices.

From an efficiency perspective, however, these results are not a clear win over previous schemes. Both of the schemes still rely on the problematic aspects of Gentry's blueprint – namely, bootstrapping and an SWHE scheme with the undesirable properties discussed above. Thus, their per-gate computation is still more than $\tilde{\Omega}(\lambda^4)$. Nevertheless, the techniques introduced in these recent constructions are very interesting and useful to us. In particular, we use the tools and techniques introduced by Brakerski and Vaikuntanathan [BV11b] in an essential way to achieve remarkable efficiency gains.

[3] We note that bootstrapping lazily – i.e., applying the refresh procedure only at a $1/L$ fraction of the circuit levels for $L > 1$ – cannot reduce the per-gate computation further by more than a logarithmic factor for schemes that follow this blueprint, since these SWHE schemes can evaluate only log multiplicative depth before it becomes *absolutely necessary* to refresh – i.e., $L = O(\log \lambda)$.

An important, somewhat orthogonal question is the strength of assumptions underlying FHE schemes. All the schemes so far rely on the hardness of short vector problems on lattices with a *subexponential* approximation factor. Can we base FHE on the hardness of finding a polynomial approximation?

1.3 Our Results and Techniques

We leverage Brakerski and Vaikuntanathan's techniques [BV11b] to achieve asymptotically very efficient FHE schemes. Also, we base security on lattice problems with *quasi-polynomial* approximation factors. (All previous schemes relied on the hardness of problems with sub-exponential approximation factors.) In particular, we have the following theorem (informal):

- Assuming Ring LWE for an approximation factor exponential in L, we have a leveled FHE scheme that can evaluate L-level arithmetic circuits *without using bootstrapping*. The scheme has $\tilde{O}(\lambda \cdot L^3)$ per-gate computation (namely, *quasi-linear* in the security parameter).
- Alternatively, assuming Ring LWE is hard for *quasi-polynomial* factors, we have a leveled FHE scheme that uses bootstrapping *as an optimization*, where the per-gate computation (which includes the bootstrapping procedure) is $\tilde{O}(\lambda^2)$, *independent of* L.

We can alternatively base security on LWE, albeit with worse performance. We now sketch our main idea for boosting efficiency.

In the BV scheme [BV11b], like ours, a ciphertext vector $\mathbf{c} \in R^n$ (where R is a ring, and n is the "dimension" of the vector) that encrypts a message m satisfies the decryption formula $m = \big[[\langle \mathbf{c}, \mathbf{s} \rangle]_q\big]_2$, where $\mathbf{s} \in R^n$ is the secret key vector, q is an odd modulus, and $[\cdot]_q$ denotes reduction into the range $(-q/2, q/2)$. This is an abstract scheme that can be instantiated with either LWE or Ring LWE – in the LWE instantiation, R is the ring of integers mod q and n is a large dimension, whereas in the Ring LWE instantiation, R is the ring of polynomials over integers mod q and an irreducible $f(x)$, and the dimension $n = 2$.

We will call $[\langle \mathbf{c}, \mathbf{s} \rangle]_q$ the *noise* associated to ciphertext $\mathbf{c}$ under key $\mathbf{s}$. Decryption succeeds as long as the magnitude of the noise stays smaller than $q/2$. Homomorphic addition and multiplication increase the noise in the ciphertext. Addition of two ciphertexts with noise at most B results in a ciphertext with noise at most $2B$, whereas multiplication results in a noise as large as B^2. [4] We will describe a *noise-management technique* that keeps the noise in check by reducing it after homomorphic operations, without bootstrapping.

The key technical tool we use for noise management is the "modulus switching" technique developed by Brakerski and Vaikuntanathan [BV11b]. Jumping ahead, we note that while they use modulus switching in "one shot" to obtain a small ciphertext (to which they then apply Gentry's bootstrapping procedure), we will use it (iteratively, gradually) to keep the noise level essentially constant, while stingily sacrificing modulus size and gradually sacrificing the remaining homomorphic capacity of the scheme.

1.4 Modulus Switching

The essence of the modulus-switching technique is captured in the following lemma. In words, the lemma says that an evaluator, who does not know the secret key $\mathbf{s}$ but instead only knows a

[4]The noise after multiplication is in fact a bit larger than B^2 due to the additional noise from the BV "re-linearization" process. For the purposes of this exposition, it is best to ignore this minor detail.

bound on its length, can transform a ciphertext $\mathbf{c}$ modulo q into a different ciphertext modulo p while preserving correctness – namely, $[\langle \mathbf{c}', \mathbf{s} \rangle]_p = [\langle \mathbf{c}, \mathbf{s} \rangle]_q \bmod 2$. The transformation from $\mathbf{c}$ to $\mathbf{c}'$ involves simply scaling by (p/q) and rounding appropriately! Most interestingly, if $\mathbf{s}$ is short and p is sufficiently smaller than q, the "noise" in the ciphertext actually decreases – namely, $|[\langle \mathbf{c}', \mathbf{s} \rangle]_p| < |[\langle \mathbf{c}, \mathbf{s} \rangle]_q|$.

Lemma 1. *Let p and q be two odd moduli, and let $\mathbf{c}$ be an integer vector. Define $\mathbf{c}'$ to be the integer vector closest to $(p/q) \cdot \mathbf{c}$ such that $\mathbf{c}' = \mathbf{c} \bmod 2$. Then, for any $\mathbf{s}$ with $|[\langle \mathbf{c}, \mathbf{s} \rangle]_q| < q/2 - (q/p) \cdot \ell_1(\mathbf{s})$, we have*

$$\begin{aligned} [\langle \mathbf{c}', \mathbf{s} \rangle]_p &= [\langle \mathbf{c}, \mathbf{s} \rangle]_q \bmod 2 \quad \text{and} \\ |[\langle \mathbf{c}', \mathbf{s} \rangle]_p| &< (p/q) \cdot |[\langle \mathbf{c}, \mathbf{s} \rangle]_q| + \ell_1(\mathbf{s}) \end{aligned}$$

where $\ell_1(\mathbf{s})$ is the ℓ_1-norm of $\mathbf{s}$.

Proof. For some integer k, we have $[\langle \mathbf{c}, \mathbf{s} \rangle]_q = \langle \mathbf{c}, \mathbf{s} \rangle - kq$. For the same k, let $e_p = \langle \mathbf{c}', \mathbf{s} \rangle - kp \in \mathbb{Z}$. Since $\mathbf{c}' = \mathbf{c}$ and $p = q$ modulo 2, we have $e_p = [\langle \mathbf{c}, \mathbf{s} \rangle]_q \bmod 2$. Therefore, to prove the lemma, it suffices to prove that $e_p = [\langle \mathbf{c}', \mathbf{s} \rangle]_p$ and that it has small enough norm. We have $e_p = (p/q)[\langle \mathbf{c}, \mathbf{s} \rangle]_q + \langle \mathbf{c}' - (p/q)\mathbf{c}, \mathbf{s} \rangle$, and therefore $|e_p| \leq (p/q)[\langle \mathbf{c}, \mathbf{s} \rangle]_q + \ell_1(\mathbf{s}) < p/2$. The latter inequality implies $e_p = [\langle \mathbf{c}', \mathbf{s} \rangle]_p$. □

Amazingly, this trick permits the evaluator to reduce the magnitude of the noise without knowing the secret key, and without bootstrapping. In other words, modulus switching gives us a very powerful and lightweight way to *manage the noise* in FHE schemes! In [BV11b], the modulus switching technique is bundled into a "dimension reduction" procedure, and we believe it deserves a separate name and close scrutiny. It is also worth noting that our use of modulus switching does not require an "evaluation key", in contrast to [BV11b].

1.5 Our New Noise Management Technique

At first, it may look like modulus switching is not a very effective noise management tool. If p is smaller than q, then of course modulus switching may reduce the magnitude of the noise, but it reduces the modulus size by essentially the same amount. In short, the ratio of the noise to the "noise ceiling" (the modulus size) does not decrease at all. Isn't this ratio what dictates the remaining homomorphic capacity of the scheme, and how can potentially worsening (certainly not improving) this ratio do anything useful?

In fact, it's not just the ratio of the noise to the "noise ceiling" that's important. The *absolute magnitude of the noise* is also important, especially in multiplications. Suppose that $q \approx x^k$, and that you have two mod-q SWHE ciphertexts with noise of magnitude x. If you multiply them, the noise becomes x^2. After 4 levels of multiplication, the noise is x^{16}. If you do another multiplication at this point, you reduce the ratio of the noise ceiling (i.e. q) to the noise level by a huge factor of x^{16} – i.e., you reduce this gap very fast. Thus, the actual magnitude of the noise impacts how fast this gap is reduced. After only $\log k$ levels of multiplication, the noise level reaches the ceiling.

Now, consider the following alternative approach. Choose a *ladder of gradually decreasing moduli* $\{q_i \approx q/x^i\}$ for $i < k$. After you multiply the two mod-q ciphertexts, switch the ciphertext to the smaller modulus $q_1 = q/x$. As the lemma above shows, the noise level of the new ciphertext (now with respect to the modulus q_1) goes from x^2 back down to x. (Let's suppose for now that

$\ell_1(\mathbf{s})$ is small in comparison to x so that we can ignore it.) Now, when we multiply two ciphertexts (wrt modulus q_1) that have noise level x, the noise again becomes x^2, but then we switch to modulus q_2 to reduce the noise back to x. In short, each level of multiplication only reduces the ratio (noise ceiling)/(noise level) by a factor of x (not something like x^{16}). With this new approach, we can perform about k (not just $\log k$) levels of multiplication before we reach the noise ceiling. We have just increased (without bootstrapping) the number of multiplicative levels that we can evaluate by an exponential factor!

This exponential improvement is enough to achieve leveled FHE without bootstrapping. For *any* polynomial L, we can evaluate circuits of depth L. The performance of the scheme degrades with L – e.g., we need to set $q = q_0$ to have bit length proportional to L – but it degrades only polynomially with L.

Our main observation – the key to obtaining FHE without bootstrapping – is so simple that it is easy to miss and bears repeating: We get noise reduction *automatically* via modulus switching, and by carefully calibrating our ladder of moduli $\{q_i\}$, one modulus for each circuit level, to be decreasing *gradually*, we can keep the noise level very small and essentially constant from one level to the next while only gradually sacrificing the size of our modulus until the ladder is used up. With this approach, we can efficiently evaluate arbitrary polynomial-size arithmetic circuits without resorting to bootstrapping.

In terms of performance, this scheme trounces previous FHE schemes (at least asymptotically; the concrete performance remains to be seen). Instantiated with ring-LWE, it can evaluate L-level arithmetic circuits with per-gate computation $\tilde{O}(\lambda \cdot L^3)$ – i.e., computation *quasi-linear* in the security parameter. Since the ratio of the largest modulus (namely, $q \approx x^L$) to the noise (namely, x) is exponential in L, the scheme relies on the hardness of approximating short vectors to within an exponential in L factor.

1.6 Bootstrapping for Better Efficiency and Better Assumptions

In our FHE-without-bootstrapping scheme, the per-gate computation depends polynomially on the number of levels in the circuit that is being evaluated. While this approach is efficient (in the sense of "polynomial time") for polynomial-size circuits, the per-gate computation may become undesirably high for very deep circuits. So, we re-introduce bootstrapping *as an optimization*[5] that makes the per-gate computation independent of the circuit depth, and that (if one is willing to assume circular security) allows homomorphic operations to be performed indefinitely without needing to specify in advance a bound on the number of circuit levels. The main idea is that to compute arbitrary polynomial-depth circuits, it is enough to compute the decryption circuit of the scheme homomorphically. Since the decryption circuit has depth $\approx \log \lambda$, the largest modulus we need has only polylog(λ) bits, and therefore we can base security on the hardness of lattice problems with quasi-polynomial factors. Since the decryption circuit has size $\widetilde{O}(\lambda)$ for the RLWE-based instantiation, the per-gate computation becomes $\tilde{O}(\lambda^2)$ (independent of L). See Section 6 for details.

We then consider *batching* as an optimization. The idea behind batching is to pack multiple

[5] We are aware of the seeming irony of trumpeting "FHE without bootstrapping" and then proposing bootstrapping "as an optimization". But, first, FHE without bootstrapping is exciting theoretically, independent of performance. Second, whether bootstrapping actually improves performance depends crucially on the number of levels in the circuit one is evaluating. For example, for circuits of depth sub-polynomial in the security parameter, it will be more efficient asymptotically to forgo the bootstrapping optimization.

plaintexts into each ciphertext so that a function can be homomorphically evaluated on multiple inputs with approximately the same efficiency as homomorphically evaluating it on one input.

An especially interesting case is *batching the decryption function* so that multiple ciphertexts – e.g., all of the ciphertexts associated to gates at some level in the circuit – can be bootstrapped simultaneously very efficiently. For circuits of large width (say, width λ), batched bootstrapping reduces the per-gate computation in the RLWE-based instantiation to $\tilde{O}(\lambda)$, independent of L. We give the details in Section 6.

1.7 Related and Subsequent Work

Prior to Gentry's construction, there were already a few interesting homomorphic encryptions schemes that could be called "somewhat homomorphic", including Boneh-Goh-Nissim [BGN05] (that evaluates quadratic formulas using bilinear maps), Aguilar Melchor-Gaborit-Herranz [MGH10] (that evaluates constant degree polynomials using lattices) and Ishai-Paskin [IP07] (that evaluates branching programs).

Our work has inspired a number of follow-up works, building upon it and extending it in both theoretical and practical directions.

Our RLWE-based FHE scheme without bootstrapping requires only $\tilde{O}(\lambda \cdot L^3)$ per-gate computation where L is the depth of the circuit being evaluated, while the bootstrapped version has only $\tilde{O}(\lambda^2)$ per-gate computation. For circuits of width $\Omega(\lambda)$, we can use batching to reduce the per-gate computation of the bootstrapped version by another factor of λ. In a follow-up work, Gentry, Halevi and Smart [GHS12b] showed how to reduce the per-gate evaluation overhead to polylogarithmic, making clever use of sorting networks to avoid packing/unpacking after every batched operation. They further suggested [GHS12a] choices of parameters so as to optimize the implementation of RLWE-based schemes. However, the polylogarithmic factors in these constructions are still too large to offer improvement for any "reasonable" value of the security parameter. One future direction toward a truly practical scheme is to tighten up these polylogarithmic factors considerably.

Using the ideas from this work in an essential way, Gentry, Halevi and Smart [GHS12c] implemented a variant of our scheme and presented benchmarks for homomorphic evaluation of the AES function.

Brakerski [Bra12] showed how to achieve comparable performance to ours without using modulus switching, for arbitrary values of "initial" q. This is achieved by scaling the ciphertexts at the beginning of time rather than doing so after every operation. Gentry, Halevi, Peikert and Smart [GHPS12] extended our ideas from Section 4 and showed how to perform ideal reduction in the general case (and not just for the rings that we consider).

2 Preliminaries

2.1 Basic Notation

In our construction, we will use a ring R. In the concrete instantiations, we prefer to use either $R = \mathbb{Z}$ (the integers) or the polynomial ring $R = \mathbb{Z}[x]/(x^d + 1)$, where d is a power of 2.

We write elements of R in lowercase – e.g., $r \in R$. We write vectors in bold – e.g., $\mathbf{v} \in R^n$. The notation $\mathbf{v}[i]$ refers to the i-th coefficient of $\mathbf{v}$. We write the dot product of $\mathbf{u}, \mathbf{v} \in R^n$ as $\langle \mathbf{u}, \mathbf{v} \rangle = \sum_{i=1}^{n} \mathbf{u}[i] \cdot \mathbf{v}[i] \in R$. When R is a polynomial ring, $\|r\|$ for $r \in R$ refers to the Euclidean norm of r's coefficient vector. We say $\gamma_R = \max\{\|a \cdot b\|/\|a\|\|b\| : a, b \in R\}$ is the expansion factor of R. For $R = \mathbb{Z}[x]/(x^d + 1)$, the value of γ_R is at most $\sqrt{d}$ by Cauchy-Schwarz. (The canonical embedding [LPR10] provides a better, tighter way of handling the geometry of cyclotomic rings. We instead use the expansion factor, defined above, for its simplicity, and since it suffices for our asymtotic results.)

For integer q, we use R_q to denote R/qR. Sometimes we will use abuse notation and use R_2 to denote the set of R-elements with binary coefficients – e.g., when $R = \mathbb{Z}$, R_2 may denote $\{0, 1\}$, and when R is a polynomial ring, R_2 may denote those polynomials that have 0/1 coefficients. We use $R_{q,d}$ when we also want to specify the degree of the polynomial associated to R. When it is obvious that q is not a power of two, we will use $\lceil \log q \rceil$ to denote $1 + \lfloor \log q \rfloor$. For $a \in R$, we use the notation $[a]_q$ to refer to a mod q, with coefficients reduced into the range $(-q/2, q/2]$.

2.2 Homomorphic Encryption

We start with a generic definition of homomorphic encryption that captures both fully and non-fully homomorphic schemes. A homomorphic encryption scheme is a tuple (HE.KeyGen, HE.Enc, HE.Dec, HE.Eval) of probabilistic polynomial time algorithms. In this work, the message space of the scheme will always be some ring $R_{\mathcal{M}}$ and our computational model will be arithmetic circuits over this ring (i.e. addition and multiplication gates).

1. HE.$KeyGen$ takes the security parameter (and possibly other parameters of the scheme) and produces a secret key sk and a public key pk.

2. HE.Enc takes the public key pk a message m and produces a ciphertext c which is the encryption of m.

3. HE.Dec takes the secret key sk and a ciphertext c and produces a message m.

4. HE.Eval takes the public key pk, an arithmetic circuit f over $R_{\mathcal{M}}$, and ciphertexts $c_1, \ldots, c_\ell$, where ℓ is the number of inputs to f, and outputs a ciphertext c_f.

Definition 1. *We say that a homomorphic encryption* correctly evaluates *a circuit family $\mathcal{F}$ if for all $f \in \mathcal{F}$ and for all $m_1, \ldots, m_\ell \in R_{\mathcal{M}}$ it holds that if sk, pk were properly generated by* HE.KeyGen *with security parameter λ, and if $c_i = \mathsf{HE.Enc}_{pk}(m_i)$ for all i, and $c_f = \mathsf{HE.Eval}_{pk}(f, c_1, \ldots, c_\ell)$, then*

$$\Pr[\mathsf{HE.Dec}_{sk}(c_f) \neq f(m_1, \ldots, m_\ell)] = \mathsf{negl}(\lambda) \ ,$$

where the probability is taken over all the randomness in the experiment.

We say that the scheme compactly evaluates *the family if in addition the running time of the decryption circuit only depends on λ and not on its input.*

The security notion we use is standard semantic security (or equivalently security under chosen plaintext attack).

Definition 2. *A homomorphic scheme is secure if any polynomial time adversary that first gets a properly generated pk, then specifies $m_0, m_1 \in R_{\mathcal{M}}$ and finally gets $\mathsf{HE.Enc}_{pk}(m_b)$ for a random b, cannot guess the value of b with probability $> 1/2 + \mathsf{negl}(\lambda)$.*

Most of this paper will focus on the construction of a *leveled* fully homomorphic scheme, in the sense that the parameters of the scheme depend (polynomially) on the depth of the circuits that the scheme is capable of evaluating.

Definition 3 (Leveled FHE [Gen09a])**.** *We say that a family of homomorphic encryption schemes $\{\mathcal{E}^{(L)} : L \in \mathbb{Z}^+\}$ is leveled fully homomorphic if, for all $L \in \mathbb{Z}^+$, they all use the same decryption circuit, $\mathcal{E}^{(L)}$ compactly evaluates all circuits of depth at most L, and the computational complexity of $\mathcal{E}^{(L)}$'s algorithms is polynomial (the same polynomial for all L) in the security parameter, L, and (in the case of the evaluation algorithm) the size of the circuit.*

Gentry's bootstrapping theorem allows to transform schemes that enjoy some level of homomorphism (bootstrappable schemes) into leveled FHE schemes. Let us first define what a bootstrappable scheme is:

Definition 4. *We say that a homomorphic encryption scheme $\mathcal{E}$ is bootstrappable if $\mathcal{E}$ compactly evaluates all circuits of depth at most $(D+1)$, where D is the depth of $\mathcal{E}$'s decryption circuit,*[6] *and the computational complexity of $\mathcal{E}$'s algorithms is polynomial in the security parameter and (in the case of the evaluation algorithm) the size of the circuit.*

We can now state the bootstrapping theorem:

Theorem 1 (Bootstrapping [Gen09a])**.** *If there exists a bootstrappable encryption scheme $\mathcal{E}$, then there also exists a leveled FHE scheme $\{\mathcal{E}^{(L)}\}$ with related security.*

Letting S be the size of $\mathcal{E}$'s decryption circuit, the per-gate evaluation complexity of the leveled FHE is exactly the complexity of evaluating a $(2S+1)$-gate circuit using the bootstrappable scheme.

2.3 The Learning with Errors (LWE) Problem

The learning with errors (LWE) problem was introduced by Regev [Reg05]. It is defined as follows.

Definition 5 (LWE)**.** *For security parameter λ, let $n = n(\lambda)$ be an integer dimension, let $q = q(\lambda) \geq 2$ be an integer, and let $\chi = \chi(\lambda)$ be a distribution over $\mathbb{Z}$. The $\mathsf{LWE}_{n,q,\chi}$ problem is to distinguish the following two distributions: In the first distribution, one samples $(\mathbf{a}_i, b_i)$ uniformly from $\mathbb{Z}_q^{n+1}$. In the second distribution, one first draws $\mathbf{s} \leftarrow \mathbb{Z}_q^n$ uniformly and then samples $(\mathbf{a}_i, b_i) \in \mathbb{Z}_q^{n+1}$ by sampling $\mathbf{a}_i \leftarrow \mathbb{Z}_q^n$ uniformly, $e_i \leftarrow \chi$, and setting $b_i = \langle \mathbf{a}, \mathbf{s} \rangle + e_i$. The $\mathsf{LWE}_{n,q,\chi}$ assumption is that the $\mathsf{LWE}_{n,q,\chi}$ problem is infeasible.*

Regev [Reg05] proved that for certain moduli q and Gaussian error distributions χ, the $\mathsf{LWE}_{n,q,\chi}$ assumption is true as long as certain worst-case lattice problems are hard to solve using a quantum algorithm. We state this result using the terminology of B-bounded distributions, which is a distribution over the integers where the magnitude of a sample is bounded with high probability. A definition follows.

Definition 6 (B-bounded distributions)**.** *A distribution ensemble $\{\chi_n\}_{n \in \mathbb{N}}$, supported over the integers, is called B-bounded if*

$$\Pr_{e \leftarrow \chi_n}[|e| > B] = \mathsf{negl}(n) \ .$$

[6]The size and depth of the decryption circuit are measured when the secret key is treated as the input, and the decrypted ciphertext is treated as constant.

We can now state Regev's worst-case to average-case reduction for LWE.

Theorem 2 (Regev [Reg05])**.** *For any integer dimension n, prime integer $q = q(n)$, and $B = B(n) \geq 2n$, there is an efficiently samplable B-bounded distribution χ such that if there exists an efficient (possibly quantum) algorithm that solves* $\mathsf{LWE}_{n,q,\chi}$*, then there is an efficient quantum algorithm for solving $\tilde{O}(qn^{1.5}/B)$-approximate worst-case* SIVP *and* gapSVP*.*

Peikert [Pei09] de-quantized Regev's results to some extent – that is, he showed the $\mathsf{LWE}_{n,q,\chi}$ assumption is true as long as certain worst-case lattice problems are hard to solve using a *classical* algorithm. (See [Pei09] for a precise statement of these results.)

Applebaum et al. [ACPS09] showed that if LWE is hard for the above distribution of $\mathbf{s}$, then it is also hard when $\mathbf{s}$'s coefficients are sampled according to the noise distribution χ.

2.4 The Ring Learning with Errors (RLWE) Problem

The ring learning with errors (RLWE) problem was introduced by Lyubaskevsky, Peikert and Regev [LPR10]. We will use an simplified special-case version of the problem that is easier to work with [Reg10, BV11a].

Definition 7 (RLWE)**.** *For security parameter λ, let $f(x) = x^d + 1$ where $d = d(\lambda)$ is a power of 2. Let $q = q(\lambda) \geq 2$ be an integer. Let $R = \mathbb{Z}[x]/(f(x))$ and let $R_q = R/qR$. Let $\chi = \chi(\lambda)$ be a distribution over R. The* $\mathsf{RLWE}_{d,q,\chi}$ *problem is to distinguish the following two distributions: In the first distribution, one samples (a_i, b_i) uniformly from R_q^2. In the second distribution, one first draws $s \leftarrow R_q$ uniformly and then samples $(a_i, b_i) \in R_q^2$ by sampling $a_i \leftarrow \mathbb{R}_q$ uniformly, $e_i \leftarrow \chi$, and setting $b_i = a_i \cdot s + e_i$. The* $\mathsf{RLWE}_{d,q,\chi}$ *assumption is that the* $\mathsf{RLWE}_{d,q,\chi}$ *problem is infeasible.*

The RLWE problem is useful, because the well-established shortest vector problem (SVP) over ideal lattices can be reduced to it, specifically:

Theorem 3 (Lyubashevsky-Peikert-Regev [LPR10])**.** *For any d that is a power of 2, ring $R = \mathbb{Z}[x]/(x^d + 1)$, prime integer $q = q(d) = 1 \bmod d$, and $B = \omega(\sqrt{d \log d})$, there is an efficiently samplable distribution χ that outputs elements of R of length at most B with overwhelming probability, such that if there exists an efficient algorithm that solves* $\mathsf{RLWE}_{d,q,\chi}$*, then there is an efficient quantum algorithm for solving $d^{\omega(1)} \cdot (q/B)$-approximate worst-case* SVP *for ideal lattices over R.*

Typically, to use RLWE with a cryptosystem, one chooses the noise distribution χ according to a Gaussian distribution, where vectors sampled according to this distribution have length only poly(d) with overwhelming probability. This Gaussian distribution may need to be "ellipsoidal" for certain reductions to go through [LPR10]. It has been shown for RLWE that one can equivalently assume that s is alternatively sampled from the noise distribution χ [LPR10].

2.5 The General Learning with Errors (GLWE) Problem

The learning with errors (LWE) problem and the ring learning with errors (RLWE) problem are syntactically identical, aside from using different rings ($\mathbb{Z}$ versus a polynomial ring) and different vector dimensions over those rings ($n = \text{poly}(\lambda)$ for LWE, but n is constant – namely, 1 – in the RLWE case). To simplify our presentation, we define a "General Learning with Errors (GLWE)" Problem, and describe a single "GLWE-based" FHE scheme, rather than presenting essentially the same scheme twice, once for each of our two concrete instantiations.

Definition 8 (GLWE). *For security parameter λ, let $n = n(\lambda)$ be an integer dimension, let $f(x) = x^d + 1$ where $d = d(\lambda)$ is a power of 2, let $q = q(\lambda) \geq 2$ be a prime integer, let $R = \mathbb{Z}[x]/(f(x))$ and $R_q = R/qR$, and let $\chi = \chi(\lambda)$ be a distribution over R. The $\mathsf{GLWE}_{n,f,q,\chi}$ problem is to distinguish the following two distributions: In the first distribution, one samples $(\mathbf{a}_i, b_i)$ uniformly from R_q^{n+1}. In the second distribution, one first draws $\mathbf{s} \leftarrow R_q^n$ uniformly and then samples $(\mathbf{a}_i, b_i) \in R_q^{n+1}$ by sampling $\mathbf{a}_i \leftarrow R_q^n$ uniformly, $e_i \leftarrow \chi$, and setting $b_i = \langle \mathbf{a}_i, \mathbf{s} \rangle + e_i$. The $\mathsf{GLWE}_{n,f,q,\chi}$ assumption is that the $\mathsf{GLWE}_{n,f,q,\chi}$ problem is infeasible.*

LWE is simply GLWE instantiated with $d = 1$. RLWE is GLWE instantiated with $n = 1$. Interestingly, as far as we know, instances of GLWE between these extremes have not been explored. One would suspect that GLWE is hard for any (n, d) such that $n \cdot d = \Omega(\lambda \log(q/B))$, where B is a bound (with overwhelming probability) on the length of elements output by χ. For fixed $n \cdot d$, perhaps GLWE gradually becomes harder as n increases (if it is true that general lattice problems are harder than ideal lattice problems), whereas increasing d is probably often preferable for efficiency.

If q is much larger than B, the associated GLWE problem is believed to be easier (i.e., there is less security). Previous FHE schemes required q/B to be sub-exponential in n or d to give room for the noise to grow as homomorphic operations (especially multiplication) are performed. In our FHE scheme without bootstrapping, q/B will be exponential in the number of circuit levels to be evaluated. However, since the decryption circuit can be evaluated in logarithmic depth, the bootstrapped version of our scheme will only need q/B to be quasi-polynomial, and we thus base security on lattice problems for quasi-polynomial approximation factors.

By the GLWE assumption, the distribution $\{(\mathbf{a}_i, \langle \mathbf{a}_i, \mathbf{s} \rangle + t \cdot e_i)\}$ is computational indistinguishable from uniform for any t relatively prime to q. This fact will be convenient for encryption, where, for example, a message m may be encrypted as $(\mathbf{a}, \langle \mathbf{a}, \mathbf{s} \rangle + 2e + m)$, and this fact can be used to argue that the second component of this message is indistinguishable from random.

3 (Leveled) FHE without Bootstrapping: Our Construction

The plan of this section is to present our "leveled FHE without bootstrapping" construction in modular steps. First, we describe a plain GLWE-based encryption scheme with no homomorphic operations. Next, we augment the plain scheme with variants of the "relinearization" and "dimension reduction" techniques of [BV11b]. Finally, in Section 3.4, we lay out our full-fledged construction of FHE without bootstrapping.

3.1 The Basic Encryption Scheme

We begin by presenting a basic GLWE-based encryption scheme with no homomorphic operations.

Let λ be the security parameter, representing our goal of achieving 2^λ-security against known attacks. ($\lambda = 100$ is, for example, a reasonable value.) Let $R = R(\lambda)$ be a ring. Specifically, we will be interested in two instantiations:

- $R = \mathbb{Z}$, which will give us a scheme based on (standard) LWE, and
- $R = \mathbb{Z}[x]/f(x)$ where (e.g.) $f(x) = x^d + 1$ and $d = d(\lambda)$ is a power of 2, which will give us a scheme based on RLWE with the appropriate parameteres.

Let the "dimension" $n = n(\lambda)$, the degree $d = d(\lambda)$, an odd positive integer modulus $q = q(\lambda)$, and a "noise" distribution $\chi = \chi(\lambda)$ over R be parameters of the system which come from the GLWE assumption. Let $R_q := R/R$. For simplicity, assume for now that the plaintext space is $R_2 := R/2R$, though we will discuss the issue of supporting larger plaintext spaces in the sequel. In addition to the usual GLWE parameters, we will use an additional parameter $N = N(\lambda) = n \cdot \mathrm{polylog}(q)$ which we will discuss following the description of the scheme.

We go ahead and stipulate here – even though it only becomes important when we introduce homomorphic operations – that the noise distribution χ will be supported over a set $[-B, \ldots, B]$ where B is set to be as small as possible (while maintaining security).

The Basic GLWE-Based Encryption Scheme. The scheme $\mathsf{E} = (\mathsf{E.Setup}, \mathsf{E.SecretKeyGen}, \mathsf{E.PublicKeyGen}, \mathsf{E.Enc}, \mathsf{E.Dec})$ works as follows.

- $\mathsf{E.Setup}(1^\lambda, 1^\mu, b)$: Use the bit $b \in \{0, 1\}$ to determine whether we are setting parameters for a LWE-based scheme (where $d = 1$) or a RLWE-based scheme (where $n = 1$). Choose a μ-bit modulus q and choose the parameters $d = d(\lambda, \mu, b)$, $n = n(\lambda, \mu, b)$, $N = N(\lambda, \mu, b)$ and $\chi = \chi(\lambda, \mu, b)$ appropriately to ensure that the scheme is based on a GLWE instance that achieves 2^λ security against known attacks. Let $R = \mathbb{Z}[x]/(x^d + 1)$ and let $params = (q, d, n, N, \chi)$.

- $\mathsf{E.SecretKeyGen}(params)$: Sample $\mathbf{t} \leftarrow \chi^n$. Output

$$sk = \mathbf{s} \leftarrow (1, \mathbf{t}[1], \ldots, \mathbf{t}[n]) \in R_q^{n+1}$$

- $\mathsf{E.PublicKeyGen}(params, sk)$: Takes as input a secret key $sk = \mathbf{s} = (1, \mathbf{t})$ with $\mathbf{s}[0] = 1$ and $\mathbf{t} \in R_q^n$ and the parameters $params$. Generate a matrix $\mathbf{B} \leftarrow R_q^{N \times n}$ uniformly and a vector $\mathbf{e} \leftarrow \chi^N$ and set $\mathbf{b} := \mathbf{B}\mathbf{t} + 2\mathbf{e}$. Set $\mathbf{A}$ to be the $(n+1)$-column matrix consisting of $\mathbf{b}$ followed by the n columns of $-\mathbf{B}$. Set the public key $pk = \mathbf{A}$.

  Remark: Observe that $\mathbf{A} \cdot \mathbf{s} = 2\mathbf{e}$.

- $\mathsf{E.Enc}(params, pk, m)$: To encrypt a message $m \in R_2$, set $\mathbf{m} := (m, 0, \ldots, 0) \in R_q^{n+1}$, sample $\mathbf{r} \leftarrow R_2^N$ and output the ciphertext

$$\mathbf{c} := \mathbf{m} + \mathbf{A}^T \mathbf{r} \in R_q^{n+1}$$

- $\mathsf{E.Dec}(params, sk, \mathbf{c})$: Output $m := [[\langle \mathbf{c}, \mathbf{s} \rangle]_q]_2$.

Let us now describe, informally, why this scheme is correct and secure. Correctness is easy to see: decryption works correctly because

$$[[\langle \mathbf{c}, \mathbf{s} \rangle]_q]_2 = [[(\mathbf{m}^T + \mathbf{r}^T \mathbf{A}) \cdot \mathbf{s}]_q]_2 = [[m + 2\mathbf{r}^T \mathbf{e}]_q]_2 = [m + 2\mathbf{r}^T \mathbf{e}]_2 = m$$

where the third equality holds because $\mathbf{e}$ and $\mathbf{r}$ are so short that the value $m + 2\mathbf{r}^T \mathbf{e}$ is guaranteed not to wrap around modulo q.

It is straightforward to base security on special cases (depending on the parameters) of the GLWE assumption (and one can find such proofs of special cases in prior work). To sketch the main ideas, first note that if an attacker can distinguish the public key $\mathbf{A}$ from a uniformly random matrix over $R_q^{N\times(n+1)}$, then the attacker can be used to solve the GLWE problem (for specific parameters). Therefore, assuming the GLWE problem is hard, an attacker cannot efficiently distinguish. Second, if $\mathbf{A}$ was indeed chosen uniformly from $R_q^{N\times(n+1)}$, the encryption procedure generates ciphertexts that are statistically independent from m (by the leftover hash lemma), and therefore the attacker has negligible advantage in guessing m.

For the LWE case, it suffices to take $N > 2n\log q$ [Reg05]. For RLWE, it does not necessarily work just to take $N > 2n\log q = 2\log q$ due to subtle distributional issues – in particular, the problem is that R_q may have many zero divisors. Micciancio's regularity lemma [Mic07] assures us that if $\mathbf{A} \in R_q^{N\times(n+1)}$ and $\mathbf{r} \in R_2^N$ are uniform, then $\mathbf{A}^T\mathbf{r}$ has negligible statistical distance from uniform when $N = \log(q \cdot \lambda^{\omega(1)})$. Lyubashevsky et al. [LPR10] (full version of the paper) give a stronger result when all of the ring elements in the matrix $\mathbf{A}$ are in R_q^* (non-zero-divisors) – namely, the distribution is within $2^{-\Omega(d)}$ of uniform when the ring elements in the $\mathbf{r}$ are chosen from a discrete Gaussian distribution of width $dq^{1/N}$. (Using this result would necessitate some changes to the encryption scheme above.)

To achieve 2^λ security against known lattice attacks, one must have $n \cdot d = \Omega(\lambda \cdot \log(q/B))$ where B is a bound on the length of the noise. Since n or d depends logarithmically on q, and since the distribution χ (and hence B) depends sub-linearly on n or d, the distribution χ (and hence B) depends sub-logarithmically on q. This dependence is weak, and one should think of the noise distribution as being essentially independent of q.

An Alternate Encryption Scheme Based on RLWE. While we think our description of encryption above is useful in that it highlights the high-level similarity of LWE and RLWE, the distributional issues discussed above make it more desirable, in practice, to use a slightly different approach to encryption in the RLWE setting. In particular, Lyubashevsky et al. [LPR10] streamline public key generation and encryption in the RLWE setting as follows:

- $\mathsf{E.PublicKeyGen}(params, sk)$: As above, except $N = 1$.
- $\mathsf{E.Enc}(params, pk, m)$: To encrypt a message $m \in R_2$, set $\mathbf{m} := (m, 0) \in R_q^2$, sample $r \leftarrow \chi$ and $\mathbf{e} \leftarrow \chi^2$. Output the ciphertext

  $$\mathbf{c} := \mathbf{m} + 2 \cdot \mathbf{e} + \mathbf{A}^T \cdot r \in R_q^2$$

  (That is $\mathbf{c} \in R_q^2$ is the sum of $\mathbf{m}$, a small even vector, and r (a small ring element) times the single encryption of zero given in the public key (namely $\mathbf{A}^T$)).

The security of LPR encryption relies on RLWE: assuming RLWE, $\mathbf{A}^T$ is computationally indistinguishable from uniform in R_q^2. Then, invoking the RLWE assumption once again, the two ring elements $m+a_1\cdot r+e_1$ and $a_2\cdot r+e_2$ of the ciphertext generated during encryption are pseudorandom.

In what follows, some of our schemes will invoke the function $\mathsf{E.PublicKeyGen}(params, sk, N)$ with an integer parameter N. In that case, it invokes the first version of $\mathsf{E.PublicKeyGen}$ (not the alternate LPR version presented above) with the specified value of N.

3.2 Key Switching

We start by reminding the reader that in the basic GLWE-based encryption scheme above, the decryption equation for a ciphertext $\mathbf{c}$ that encrypts m under key $\mathbf{s}$ can be written as $m = [[L_{\mathbf{c}}(\mathbf{s})]_q]_2$

where $L_{\mathbf{c}}(\mathbf{x})$ is a ciphertext-dependent linear equation over the coefficients of $\mathbf{x}$ given by $L_{\mathbf{c}}(\mathbf{x}) = \langle \mathbf{c}, \mathbf{x} \rangle$.

Suppose now that we have two ciphertexts $\mathbf{c}_1$ and $\mathbf{c}_2$, encrypting m_1 and m_2 respectively under the same secret key $\mathbf{s}$. The way homomorphic multiplication is accomplished in [BV11b] is to consider the *quadratic* equation $Q_{\mathbf{c}_1,\mathbf{c}_2}(\mathbf{x}) := L_{\mathbf{c}_1}(\mathbf{x}) \cdot L_{\mathbf{c}_2}(\mathbf{x})$. Assuming the noises of the initial ciphertexts are small enough, we obtain $m_1 \cdot m_2 = [Q_{\mathbf{c}_1,\mathbf{c}_2}(\mathbf{s})]_q]_2$, as desired. If one wishes, one can view $Q_{\mathbf{c}_1,\mathbf{c}_2}(\mathbf{x})$ as a *linear* equation $L^{long}_{\mathbf{c}_1,\mathbf{c}_2}(\mathbf{x} \otimes \mathbf{x})$ over the coefficients of $\mathbf{x} \otimes \mathbf{x}$ – that is, the tensoring of $\mathbf{x}$ with itself – where $\mathbf{x} \otimes \mathbf{x}$'s dimension is roughly the square of $\mathbf{x}$'s. Using this interpretation, the ciphertext represented by the coefficients of the linear equation L^{long} is decryptable by the long secret key $\mathbf{s}_1 \otimes \mathbf{s}_1$ via the usual dot product. Of course, we cannot continue increasing the dimension like this indefinitely and preserve efficiency.

Thus, Brakerski and Vaikuntanathan convert the long ciphertext represented by the linear equation L^{long} and decryptable by the long tensored secret key $\mathbf{s}_1 \otimes \mathbf{s}_1$ into a shorter ciphertext $\mathbf{c}_2$ that is decryptable by a different secret key $\mathbf{s}_2$. (The secret keys need to be different to avoid a "circular security" issue). Encryptions of $\mathbf{s}_1 \otimes \mathbf{s}_1$ under $\mathbf{s}_2$ are provided in the public key as a "hint" to facilitate this conversion. They call this the *relinearization* procedure.

The starting point of our key switching procedure is to observe that Brakerski and Vaikuntanathan's relinearization procedure is actually quite a bit more general. It can be used to not only reduce the dimension of the ciphertext, but more generally, can be used to transform a ciphertext $\mathbf{c}_1$ that is decryptable under one secret key vector $\mathbf{s}_1$ to a different ciphertext $\mathbf{c}_2$ that encrypts the same message, but is now decryptable under a second secret key vector $\mathbf{s}_2$. The vectors $\mathbf{c}_2, \mathbf{s}_2$ may not necessarily be of lower degree or dimension than $\mathbf{c}_1, \mathbf{s}_1$. Because of this generality, we prefer to call this the key switching procedure, which we now describe in detail.

Two Useful Subroutines. The procedures will use some subroutines that, given two vectors $\mathbf{c}$ and $\mathbf{s}$, "expand" these vectors to get longer (higher-dimensional) vectors $\mathbf{c}'$ and $\mathbf{s}'$ such that $\langle \mathbf{c}', \mathbf{s}' \rangle = \langle \mathbf{c}, \mathbf{s} \rangle \bmod q$. We describe these subroutines first.

- $\mathsf{BitDecomp}(\mathbf{x} \in R_q^n, q)$ decomposes $\mathbf{x}$ into its bit representation. Namely, write $\mathbf{x} = \sum_{j=0}^{\lfloor \log q \rfloor} 2^j \cdot \mathbf{u}_j$ with all $\mathbf{u}_j \in R_2^n$. Output $(\mathbf{u}_0, \mathbf{u}_1, \ldots, \mathbf{u}_{\lfloor \log q \rfloor}) \in R_2^{n \cdot \lceil \log q \rceil}$.
- $\mathsf{Powersof2}(\mathbf{x} \in R_q^n, q)$ outputs $(\mathbf{x}, 2 \cdot \mathbf{x}, \ldots, 2^{\lfloor \log q \rfloor} \cdot \mathbf{x}) \in R_q^{n \cdot \lceil \log q \rceil}$.

Observe that:

Lemma 2. *For vectors $\mathbf{c}, \mathbf{s}$ of equal length, we have*

$$\langle \mathsf{BitDecomp}(\mathbf{c}, q), \mathsf{Powersof2}(\mathbf{s}, q) \rangle = \langle \mathbf{c}, \mathbf{s} \rangle \bmod q \ .$$

Proof. The proof follows by a simple calculation from the definitions of the procedures $\mathsf{BitDecomp}$ and $\mathsf{Powersof2}$ above. In particular,

$$\langle \mathsf{BitDecomp}(\mathbf{c}, q), \mathsf{Powersof2}(\mathbf{s}, q) \rangle = \sum_{j=0}^{\lfloor \log q \rfloor} \left\langle \mathbf{u}_j, 2^j \cdot \mathbf{s} \right\rangle = \sum_{j=0}^{\lfloor \log q \rfloor} \left\langle 2^j \cdot \mathbf{u}_j, \mathbf{s} \right\rangle = \left\langle \sum_{j=0}^{\lfloor \log q \rfloor} 2^j \cdot \mathbf{u}_j, \mathbf{s} \right\rangle = \langle \mathbf{c}, \mathbf{s} \rangle$$

□

We remark that this obviously generalizes to decompositions with respect to bases other than the powers of 2. Also, as an optimization, if one knows *a priori* that $\mathbf{x}$ has coefficients in $[0, B]$ for $B \ll q$, then $\mathsf{BitDecomp}$ can be optimized in the obvious way to output a shorter decomposition in $R_2^{n \cdot \lceil \log B \rceil}$.

The Key Switching Procedure. Key switching consists of two procedures: first, an algorithm $\mathsf{SwitchKeyGen}(\mathbf{s}_1, \mathbf{s}_2, n_1, n_2, q)$, which takes as input the two secret key vectors, the respective dimensions of these vectors, and the modulus q, and outputs some auxiliary information $\tau_{\mathbf{s}_1 \to \mathbf{s}_2}$ that enables the switching; and second, an algorithm $\mathsf{SwitchKey}(\tau_{\mathbf{s}_1 \to \mathbf{s}_2}, \mathbf{c}_1, n_1, n_2, q)$, that takes this auxiliary information and a ciphertext encrypted under $\mathbf{s}_1$ and outputs a new ciphertext $\mathbf{c}_2$ that encrypts the same message under the secret key $\mathbf{s}_2$. (Below, we often suppress the additional arguments n_1, n_2, q.)

$\mathsf{SwitchKeyGen}(\mathbf{s}_1 \in R_q^{n_1}, \mathbf{s}_2 \in R_q^{n_2})$:

1. Run $\mathbf{A} \leftarrow \mathsf{E.PublicKeyGen}(\mathbf{s}_2, N)$ for $N = n_1 \cdot \lceil \log q \rceil$.
2. Add $\mathsf{Powersof2}(\mathbf{s}_1) \in R_q^N$ to $\mathbf{A}$'s first column to get a matrix $\mathbf{B}$. Output $\tau_{\mathbf{s}_1 \to \mathbf{s}_2} = \mathbf{B}$.

$\mathsf{SwitchKey}(\tau_{\mathbf{s}_1 \to \mathbf{s}_2}, \mathbf{c}_1)$: Output $\mathbf{c}_2 = \mathsf{BitDecomp}(\mathbf{c}_1)^T \cdot \mathbf{B} \in R_q^{n_2}$.

Note that, in $\mathsf{SwitchKeyGen}$, the matrix $\mathbf{A}$ basically consists of encryptions of 0 under the key $\mathbf{s}_2$. Then, pieces of the key $\mathbf{s}_1$ are added to these encryptions of 0. Thus, in some sense, the matrix $\mathbf{B}$ consists of encryptions of pieces of $\mathbf{s}_1$ (in a certain format) under the key $\mathbf{s}_2$. We now establish that the key switching procedures are meaningful, in the sense that they preserve the correctness of decryption under the new key.

As before, the processes above are adapted to the plaintext space R_2, but are easy to generalize.

Lemma 3. **[Correctness]** *Let* $\mathbf{s}_1, \mathbf{s}_2, q, \mathbf{A}, \mathbf{B} = \tau_{\mathbf{s}_1 \to \mathbf{s}_2}$ *be as in* $\mathsf{SwitchKeyGen}(\mathbf{s}_1, \mathbf{s}_2)$*, and let* $\mathbf{A} \cdot \mathbf{s}_2 = 2\mathbf{e}_2 \in R_q^N$*. Let* $\mathbf{c}_1 \in R_q^{n_1}$ *and* $\mathbf{c}_2 \leftarrow \mathsf{SwitchKey}(\tau_{\mathbf{s}_1 \to \mathbf{s}_2}, \mathbf{c}_1)$*. Then,*

$$\langle \mathbf{c}_2, \mathbf{s}_2 \rangle = 2 \langle \mathsf{BitDecomp}(\mathbf{c}_1), \mathbf{e}_2 \rangle + \langle \mathbf{c}_1, \mathbf{s}_1 \rangle \bmod q$$

Proof.

$$\begin{aligned} \langle \mathbf{c}_2, \mathbf{s}_2 \rangle &= \mathsf{BitDecomp}(\mathbf{c}_1)^T \cdot \mathbf{B} \cdot \mathbf{s}_2 \\ &= \mathsf{BitDecomp}(\mathbf{c}_1)^T \cdot (2\mathbf{e}_2 + \mathsf{Powersof2}(\mathbf{s}_1)) \\ &= 2 \langle \mathsf{BitDecomp}(\mathbf{c}_1), \mathbf{e}_2 \rangle + \langle \mathsf{BitDecomp}(\mathbf{c}_1), \mathsf{Powersof2}(\mathbf{s}_1) \rangle \\ &= 2 \langle \mathsf{BitDecomp}(\mathbf{c}_1), \mathbf{e}_2 \rangle + \langle \mathbf{c}_1, \mathbf{s}_1 \rangle \end{aligned}$$

□

Note that the dot product of $\mathsf{BitDecomp}(\mathbf{c}_1)$ and $\mathbf{e}_2$ is small, since $\mathsf{BitDecomp}(\mathbf{c}_1)$ is in R_2^N. Overall, we have that $\mathbf{c}_2$ is a valid encryption of m under key $\mathbf{s}_2$, with noise that is larger by a small additive factor.

We will need in the sequel the following lemma which says that the key switching parameter generated by $\mathsf{SwitchKeyGen}(\mathbf{s}_1, \mathbf{s}_2)$ is computationally indistinguishable from random for any $\mathbf{s}_1 \in$

$R_q^{n_1}$ and a uniformly random $\mathbf{s}_2 \in R_q^{n_2}$. This is simply because of the properties of the basic scheme E which states that the matrix $\mathbf{A} \leftarrow \mathsf{E.PublicKeyGen}(\mathbf{s}_2, N)$ is computationally indistinguishable from uniform.

Lemma 4 (Security)**.** *For every $\mathbf{s}_1 \in R_q^{n_2}$ and a uniformly random $\mathbf{s}_2 \leftarrow R_q^{n_1}$, the following two distributions are computationally indistinguishable:*

$$\left\{(\mathbf{A}, \tau_{\mathbf{s}_1 \to \mathbf{s}_2}) : \mathbf{A} \leftarrow \mathsf{E.PublicKeyGen}(\mathbf{s}_2, N), \tau_{\mathbf{s}_1 \to \mathbf{s}_2} \leftarrow \mathsf{SwitchKeyGen}(\mathbf{s}_1, \mathbf{s}_2)\right\} \approx$$
$$\left\{(\mathbf{A}, \tau_{\mathbf{s}_1 \to \mathbf{s}_2}) : \ \mathbf{A} \leftarrow R_q^{N \times (n_2+1)}, \tau_{\mathbf{s}_1 \to \mathbf{s}_2} \leftarrow R_q^{N \times (n_2+1)}\right\}$$

is computationally indistinguishable from uniform, where the randomness is over the choice of $\mathbf{s}_2 \leftarrow R_q^{n_2}$, and the coins of $\mathsf{E.PublicKeyGen}$ *and* $\mathsf{SwitchKeyGen}$.

3.3 Modulus Switching

Suppose $\mathbf{c}$ is a valid encryption of m under $\mathbf{s}$ modulo q (i.e., $m = [[\langle \mathbf{c}, \mathbf{s} \rangle]_q]_2$), and that $\mathbf{s}$ is a *short* vector. Suppose also that $\mathbf{c}'$ is basically a simple scaling of $\mathbf{c}$ – in particular, $\mathbf{c}'$ is the R-vector closest to $(p/q) \cdot \mathbf{c}$ such that $\mathbf{c}' = \mathbf{c} \bmod 2$. Then, it turns out (subject to some qualifications) that $\mathbf{c}'$ is a valid encryption of m under $\mathbf{s}$ modulo p using the usual decryption equation – that is, $m = [[\langle \mathbf{c}', \mathbf{s} \rangle]_p]_2$! In other words, we can change the inner modulus in the decryption equation – e.g., to a smaller number – while preserving the correctness of decryption under the same secret key! The essence of this modulus switching idea, a variant of Brakerski and Vaikuntanathan's modulus reduction technique, is formally captured in Lemma 5 below.

Definition 9 (Scale)**.** *For integer vector $\mathbf{x}$ and integers $q > p > m$, we define $\mathbf{x}' \leftarrow \mathsf{Scale}(\mathbf{x}, q, p, r)$ to be the R-vector closest to $(p/q) \cdot \mathbf{x}$ that satisfies $\mathbf{x}' = \mathbf{x} \bmod r$.*

Definition 10 ($\ell_1^{(R)}$ norm)**.** *The (usual) norm $\ell_1(\mathbf{s})$ over the reals equals $\sum_i \|\mathbf{s}[i]\|$. We extend this to our ring R as follows: $\ell_1^{(R)}(\mathbf{s})$ for $\mathbf{s} \in R^n$ is defined as $\sum_i \|\mathbf{s}[i]\|$.*

Lemma 5. *Let d be the degree of the ring (e.g., $d = 1$ when $R = \mathbb{Z}$). Let $q > p > r$ be positive integers satisfying $q = p = 1 \bmod r$. Let $\mathbf{c} \in R^n$ and $\mathbf{c}' \leftarrow \mathsf{Scale}(\mathbf{c}, q, p, r)$. Then, for any $\mathbf{s} \in R^n$ with $\|[\langle \mathbf{c}, \mathbf{s} \rangle]_q\| < q/2 - (q/p) \cdot \gamma_R \cdot (r/2) \cdot \sqrt{d} \cdot \ell_1^{(R)}(\mathbf{s})$, we have*

$$\begin{aligned} [\langle \mathbf{c}', \mathbf{s} \rangle]_p &= [\langle \mathbf{c}, \mathbf{s} \rangle]_q \bmod r \quad \text{and} \\ \|[\langle \mathbf{c}', \mathbf{s} \rangle]_p\| &< (p/q) \cdot \|[\langle \mathbf{c}, \mathbf{s} \rangle]_q\| + \gamma_R \cdot (r/2) \cdot \sqrt{d} \cdot \ell_1^{(R)}(\mathbf{s}) \end{aligned}$$

Proof. (Lemma 5) We have

$$[\langle \mathbf{c}, \mathbf{s} \rangle]_q = \langle \mathbf{c}, \mathbf{s} \rangle - kq$$

for some $k \in R$. For the same k, let

$$e_p = \langle \mathbf{c}', \mathbf{s} \rangle - kp \in R$$

Note that $e_p = [\langle \mathbf{c}', \mathbf{s} \rangle]_p \bmod p$. We claim that $\|e_p\|$ is so small that $e_p = [\langle \mathbf{c}', \mathbf{s} \rangle]_p$. We have:

$$\begin{aligned}
\|e_p\| &= \| -kp + \langle (p/q) \cdot \mathbf{c}, \mathbf{s} \rangle + \big\langle \mathbf{c}' - (p/q) \cdot \mathbf{c}, \mathbf{s} \big\rangle \| \\
&\leq \| -kp + \langle (p/q) \cdot \mathbf{c}, \mathbf{s} \rangle \| + \| \big\langle \mathbf{c}' - (p/q) \cdot \mathbf{c}, \mathbf{s} \big\rangle \| \\
&\leq (p/q) \cdot \|[\langle \mathbf{c}, \mathbf{s} \rangle]_q\| + \gamma_R \cdot \sum_{j=1}^{n} \|\mathbf{c}'[j] - (p/q) \cdot \mathbf{c}[j]\| \cdot \|\mathbf{s}[j]\| \\
&\leq (p/q) \cdot \|[\langle \mathbf{c}, \mathbf{s} \rangle]_q\| + \gamma_R \cdot (r/2) \cdot \sqrt{d} \cdot \ell_1^{(R)}(\mathbf{s}) \\
&< p/2
\end{aligned}$$

Furthermore, modulo r, we have $[\langle \mathbf{c}', \mathbf{s} \rangle]_p = e_p = \langle \mathbf{c}', \mathbf{s} \rangle - kp = \langle \mathbf{c}, \mathbf{s} \rangle - kq = [\langle \mathbf{c}, \mathbf{s} \rangle]_q$. □

The lemma implies that an evaluator, who does not know the secret key but instead only knows a bound on its length, can potentially transform a ciphertext $\mathbf{c}$ that encrypts m under key $\mathbf{s}$ for modulus q – i.e., $m = [[\langle \mathbf{c}, \mathbf{s} \rangle]_q]_r$ – into a ciphertext $\mathbf{c}$ that encrypts m under the same key $\mathbf{s}$ for modulus p – i.e., $m = [[\langle \mathbf{c}, \mathbf{s} \rangle]_p]_r$. Specifically, the following corollary follows immediately from Lemma 5.

Corollary 1. *Let p and q be two odd moduli. Suppose $\mathbf{c}$ is an encryption of bit m under key $\mathbf{s}$ for modulus q – i.e., $m = [[\langle \mathbf{c}, \mathbf{s} \rangle]_q]_r$. Moreover, suppose that $\mathbf{s}$ is a fairly short key and the "noise" $e_q \leftarrow [\langle \mathbf{c}, \mathbf{s} \rangle]_q$ has small magnitude – precisely, assume that $\|e_q\| < q/2 - (q/p) \cdot (r/2) \cdot \sqrt{d} \cdot \gamma_R \cdot \ell_1^{(R)}(\mathbf{s})$. Then $\mathbf{c}' \leftarrow \mathsf{Scale}(\mathbf{c}, q, p, r)$ is an encryption of of bit m under key $\mathbf{s}$ for modulus p – i.e., $m = [[\langle \mathbf{c}, \mathbf{s} \rangle]_p]_r$. The noise $e_p = [\langle \mathbf{c}', \mathbf{s} \rangle]_p$ of the new ciphertext has magnitude at most $(p/q) \cdot \|[\langle \mathbf{c}, \mathbf{s} \rangle]_q\| + \gamma_R \cdot (r/2) \cdot \sqrt{d} \cdot \ell_1^{(R)}(\mathbf{s})$.*

Amazingly, assuming p is smaller than q and $\mathbf{s}$ has coefficients that are small in relation to q, this trick permits the evaluator to reduce the magnitude of the noise without knowing the secret key! (Of course, this is also what Gentry's bootstrapping transformation accomplishes, but in a much more complicated way.)

3.4 (Leveled) FHE Based on GLWE without Bootstrapping

We now present our scheme FHE. Given the machinery that we have described in the previous subsections, the scheme itself is remarkably simple.

In our scheme, we will use a parameter L indicating the number of levels of arithmetic circuit that we want our FHE scheme to be capable of evaluating. Note that this is an exponential improvement over prior schemes, that would typically use a parameter d indicating the *degree* of the polynomials to be evaluated.

(Note: the linear polynomial L^{long}, used below, is defined in Section 3.2.)

Our FHE Scheme without Bootstrapping.

- $\mathsf{FHE.Setup}(1^\lambda, 1^L, b)$: Takes as input the security parameter, a number of levels L, and a bit b. Use the bit $b \in \{0, 1\}$ to determine whether we are setting parameters for a LWE-based scheme (where $d = 1$) or a RLWE-based scheme (where $n = 1$). Let $\mu = \mu(\lambda, L, b) = \theta(\log \lambda + \log L)$ be a parameter that we will specify in detail later. For $j = L$ (input level of circuit) to 0 (output level), run $params_j \leftarrow \mathsf{E.Setup}(1^\lambda, 1^{(j+1)\cdot\mu}, b)$ to obtain a ladder of parameters, including a ladder of decreasing moduli from q_L ($(L+1)\cdot\mu$ bits) down to q_0 (μ bits). (The ring degree d_j, dimension n_j, and noise distribution χ_j do not necessarily need to vary (decrease) with the circuit level. In the procedure below, we allow n_j and χ_j to vary, but defer the case of decreasing d_j to Section 4.)

- $\mathsf{FHE.KeyGen}(\{params_j\})$: For $j = L$ down to 0, do the following:

  1. Run $\mathbf{s}_j \leftarrow \mathsf{E.SecretKeyGen}(params_j)$ and $\mathbf{A}_j \leftarrow \mathsf{E.PublicKeyGen}(params_j, \mathbf{s}_j)$.
  2. Set $\mathbf{s}'_j \leftarrow \mathbf{s}_j \otimes \mathbf{s}_j \in R_{q_j}^{\binom{n_j+1}{2}}$. That is, $\mathbf{s}'_j$ is a tensoring of $\mathbf{s}_j$ with itself whose coefficients are each the product of two coefficients of $\mathbf{s}_j$ in R_{q_j}.
  3. Run $\tau_{\mathbf{s}'_{j+1}\to\mathbf{s}_j} \leftarrow \mathsf{SwitchKeyGen}(\mathbf{s}'_{j+1}, \mathbf{s}_j)$. (Omit this step when $j = L$.)

  The secret key sk consists of the $\mathbf{s}_j$'s and the public key pk consists of the $\mathbf{A}_j$'s and $\tau_{\mathbf{s}'_{j+1}\to\mathbf{s}_j}$'s.[7]

- $\mathsf{FHE.Enc}(params, pk, m)$: Take a message in R_2. Run $\mathsf{E.Enc}(params_L, \mathbf{A}_L, m)$.

- $\mathsf{FHE.Dec}(params, sk, \mathbf{c})$: Suppose the ciphertext is under key $\mathbf{s}_j$. Run $\mathsf{E.Dec}(params_j, \mathbf{s}_j, \mathbf{c})$. (The ciphertext could be augmented with an index indicating which level it belongs to.)

- $\mathsf{FHE.Add}(pk, \mathbf{c}_1, \mathbf{c}_2)$: Takes two ciphertexts encrypted under the same $\mathbf{s}_j$. (If needed, use $\mathsf{FHE.Refresh}$ (below) to make it so.) Set $\mathbf{c}_3 \leftarrow \mathbf{c}_1 + \mathbf{c}_2 \bmod q_j$. Interpret $\mathbf{c}_3$ as a ciphertext under $\mathbf{s}'_j$ ($\mathbf{s}'_j$'s coefficients include all of $\mathbf{s}_j$'s since $\mathbf{s}'_j = \mathbf{s}_j \otimes \mathbf{s}_j$ and $\mathbf{s}_j$'s first coefficient is 1) and output:

$$\mathbf{c}_4 \leftarrow \mathsf{FHE.Refresh}(\mathbf{c}_3, \tau_{\mathbf{s}'_j\to\mathbf{s}_{j-1}}, q_j, q_{j-1})$$

- $\mathsf{FHE.Mult}(pk, \mathbf{c}_1, \mathbf{c}_2)$: Takes two ciphertexts encrypted under the same $\mathbf{s}_j$. (If needed, use $\mathsf{FHE.Refresh}$ (below) to make it so.) First, multiply: the new ciphertext, under the secret key $\mathbf{s}'_j = \mathbf{s}_j \otimes \mathbf{s}_j$, is the coefficient vector $\mathbf{c}_3$ of the linear equation $L^{long}_{\mathbf{c}_1,\mathbf{c}_2}(\mathbf{x}\otimes\mathbf{x})$. Then, output:

$$\mathbf{c}_4 \leftarrow \mathsf{FHE.Refresh}(\mathbf{c}_3, \tau_{\mathbf{s}'_j\to\mathbf{s}_{j-1}}, q_j, q_{j-1})$$

- $\mathsf{FHE.Refresh}(\mathbf{c}, \tau_{\mathbf{s}'_j\to\mathbf{s}_{j-1}}, q_j, q_{j-1})$: Takes a ciphertext encrypted under $\mathbf{s}'_j$, the auxiliary information $\tau_{\mathbf{s}'_j\to\mathbf{s}_{j-1}}$ to facilitate key switching, and the current and next moduli q_j and q_{j-1}. Do the following:

  1. Switch Keys: Set $\mathbf{c}_1 \leftarrow \mathsf{SwitchKey}(\tau_{\mathbf{s}'_j\to\mathbf{s}_{j-1}}, \mathbf{c}, q_j)$, a ciphertext under the key $\mathbf{s}_{j-1}$ for modulus q_j.
  2. Switch Moduli: Set $\mathbf{c}_2 \leftarrow \mathsf{Scale}(\mathbf{c}_1, q_j, q_{j-1}, 2)$, a ciphertext under the key $\mathbf{s}_{j-1}$ for modulus q_{j-1}.

Remark 1. *We mention the obvious fact that, since addition increases the noise much more slowly than multiplication, one does not necessarily need to refresh after additions, even high fan-in ones.*

[7] As observed by Gentry [Gen09b] in a similar context, it is possible to take all keys $\{\mathbf{s}_j\}$ to be the same (rather than drawing each one independently). This will make the sizes of the keys independent of L, but the security will need rely on a circular assumption.

The key step of our new FHE scheme is the Refresh procedure. If the modulus q_{j-1} is chosen to be smaller than q_j by a sufficient multiplicative factor, then Corollary 1 implies that the noise of the ciphertext output by Refresh is smaller than that of the input ciphertext – that is, the ciphertext will indeed be a "refreshed" encryption of the same value. We elaborate on this analysis in the next section.

One can reasonably argue that this scheme is not "FHE without bootstrapping" since $\tau_{\mathbf{s}'_j \to \mathbf{s}_{j-1}}$ can be viewed as an encrypted secret key, and the SwitchKey step can viewed as a homomorphic evaluation of the decryption function. We prefer not to view the SwitchKey step this way. While there is some high-level resemblance, the low-level details are very different, a difference that becomes tangible in the much better asymptotic performance. To the extent that it performs decryption, SwitchKey does so very efficiently using an efficient (not bit-wise) representation of the secret key that allows this step to be computed in quasi-linear time for the RLWE instantiation, below the quadratic lower bound for bootstrapping. Certainly SwitchKey does not use the usual ponderous approach of representing the decryption function as a boolean circuit to be traversed homomorphically. Another difference is that the SwitchKey step does not actually reduce the noise level (as bootstrapping does); rather, the noise is reduced by the Scale step.

4 Trading Off Degree for Dimension in GLWE

The definition of a leveled FHE scheme requires the complexity of the decryption circuit to be independent of L, the number of levels the leveled FHE scheme can evaluate. This required property is not superfluous. With this property, it is easy to argue that, assuming circular security, we can use bootstrapping to convert a leveled FHE into a "pure" FHE scheme whose parameters are independent of levels; without this property, such a conversion may still be possible, but the argument is less immediate.

In the LWE setting, the parameters n_0, q_0 and bound B_{χ_0} on the distribution χ_0 can all be chosen independently of L, the number of levels the leveled FHE scheme can evaluate. Consequently, the (final) ciphertext length and decryption complexity (at level 0) do not depend on L, and we therefore have a leveled FHE scheme based on LWE.

Unfortunately, in the RLWE setting, things are not so simple. For security reasons, the dimension d_j of the ring used at level j must grow linearly with the bit-length of q_j, which (as we will see) grows at least linearly in j. Therefore, if we use a fixed ring throughout, we cannot claim that the (final) ciphertext length and decryption complexity (at level 0) are independent of L. Until we address this issue, we do not have a leveled FHE scheme without bootstrapping based on RLWE. The question arises: is there a way to reduce the dimension of the ring as we progress through the circuit?

4.1 Techniques

Here, we show that there is an interplay between the dimension n of a GLWE problem and the degree d of the modulus polynomial. We show that an $GLWE_{n,x^d+1,q,\chi}$ ciphertext can be efficiently broken into two $GLWE_{2n,x^{d/2}+1,q,\chi}$ ciphertexts.

We slightly deviate from the notation in the body of the paper and denote use $\mathsf{GLWE}_{n,d,q,\chi}$ to denote $\mathsf{GLWE}_{n,x^d+1,q,\chi}$ (recall that d is always a power of 2). We further denote $R_{q,d} = \mathbb{Z}[x]/(x^d+1)$.

We begin by presenting a formal decomposition of elements from $R_{q,d}$ into elements of $R_{q,d/2}$. We show that each element $a = a(x) \in R_{q,d}$ can be represented using $a^{(\mathrm{even})}, a^{(\mathrm{odd})} \in R_{q,d/2}$. Recalling that an element in $R_{q,d}$ is a polynomial with d coefficients over $\mathbb{Z}_q$, the task seems very simple. We embed half of the coefficients of the polynomial a as coefficients of $a^{(\mathrm{even})}$ and the other half as coefficients of $a^{(\mathrm{odd})}$. However, in order to preserve an algebraic structure over the the new elements, it is critical that the coefficients are divided between them in a special way.

Specifically, we define $a^{(\mathrm{even})}, a^{(\mathrm{odd})}$ to be the elements of $R_{q,d/2}$ for which

$$a(x) = a^{(\mathrm{even})}(x^2) + x \cdot a^{(\mathrm{odd})}(x^2) \ .$$

In other words, $a^{(\mathrm{even})}$ assumes the even coefficients of a, and $a^{(\mathrm{odd})}$ the odd ones.

To see that the algebraic properties are preserved, suppose we have an equation $c(x) = a(x) \cdot b(x)$ over $R_{q,d}$, then it holds that

$$\begin{aligned}
&c^{(\mathrm{even})}(x^2) + xc^{(\mathrm{odd})}(x^2) \\
&\quad = (a^{(\mathrm{even})}(x^2) + xa^{(\mathrm{odd})}(x^2)) \cdot (b^{(\mathrm{even})}(x^2) + xb^{(\mathrm{odd})}(x^2)) \\
&\quad = a^{(\mathrm{even})}(x^2) \cdot b^{(\mathrm{even})}(x^2) + x^2 a^{(\mathrm{odd})}(x^2) \cdot b^{(\mathrm{odd})}(x^2) \\
&\qquad + x[a^{(\mathrm{even})}(x^2) \cdot b^{(\mathrm{odd})}(x^2) + a^{(\mathrm{odd})}(x^2) \cdot b^{(\mathrm{even})}(x^2)] \ .
\end{aligned}$$

Noting that the parity of a power of x cannot change by reducing modulo $x^d + 1$ (since d is a power of 2 and thus always even), it follows that we can separate odd and even powers in the expression above:

$$\begin{aligned}
c^{(\mathrm{even})}(x^2) &= a^{(\mathrm{even})}(x^2) \cdot b^{(\mathrm{even})}(x^2) + x^2 a^{(\mathrm{odd})}(x^2) \cdot b^{(\mathrm{odd})}(x^2) \\
c^{(\mathrm{odd})}(x^2) &= a^{(\mathrm{even})}(x^2) \cdot b^{(\mathrm{odd})}(x^2) + a^{(\mathrm{odd})}(x^2) \cdot b^{(\mathrm{even})}(x^2) \ .
\end{aligned}$$

The equations above are still over the ring $R_{q,d}$. In order to switch down to the ring $R_{q,d/2}$, we consider the following fact. In general, if $w(x^2) = u(x^2) \cdot v(x^2)$ over $R_{q,d}$ – i.e., modulo q and $x^d + 1$ – then it holds that $w(x) = u(x) \cdot v(x)$ over $R_{q,d/2}$ – i.e., modulo q and $x^{d/2} + 1$. This follows syntactically by replacing x^2 everywhere with x.

Therefore, we have that the following holds over $R_{q,d/2}$:

$$\begin{aligned}
c^{(\mathrm{even})}(x) &= a^{(\mathrm{even})}(x) \cdot b^{(\mathrm{even})}(x) + xa^{(\mathrm{odd})}(x) \cdot b^{(\mathrm{odd})}(x) \\
c^{(\mathrm{odd})}(x) &= a^{(\mathrm{even})}(x) \cdot b^{(\mathrm{odd})}(x) + a^{(\mathrm{odd})}(x) \cdot b^{(\mathrm{even})}(x) \ .
\end{aligned}$$

Suppose that we have a ciphertext vector $\mathbf{c} \in R_{q,d}^{n+1}$ that encrypts $m \in R_2$ under key $\mathbf{s} \in R_{q,d}^{n+1}$; that is, $m = [[\langle \mathbf{c}, \mathbf{s} \rangle]_q]_2$. Applying the facts above recursively, we conclude we can decompose $\mathbf{c}$ and $\mathbf{s}$ into vectors $\mathbf{c}^{(\mathrm{even})}, \mathbf{c}^{(\mathrm{odd})}, \mathbf{s}^{(\mathrm{even})}, \mathbf{s}^{(\mathrm{odd})} \in R_{q,d/2}^{n+1}$ such that

$$\begin{aligned}
m^{(\mathrm{even})} &= [[\left\langle \mathbf{c}^{(\mathrm{even})}, \mathbf{s}^{(\mathrm{even})} \right\rangle + x \left\langle \mathbf{c}^{(\mathrm{odd})}, \mathbf{s}^{(\mathrm{odd})} \right\rangle]_q]_2 \\
m^{(\mathrm{odd})} &= [[\left\langle \mathbf{c}^{(\mathrm{even})}, \mathbf{s}^{(\mathrm{odd})} \right\rangle + \left\langle \mathbf{c}^{(\mathrm{odd})}, \mathbf{s}^{(\mathrm{even})} \right\rangle]_q]_2 \ .
\end{aligned}$$

We therefore have two ciphertexts, one that encrypts $m^{(\mathrm{even})}$ under secret key $(\mathbf{s}^{(\mathrm{even})}, \mathbf{s}^{(\mathrm{odd})})$ and one that encrypts $m^{(\mathrm{odd})}$ under secret key $(\mathbf{s}^{(\mathrm{odd})}, \mathbf{s}^{(\mathrm{even})})$. In fact, by a simple re-ordering of the ring elements within the second ciphertext, the two ciphertexts are made to be under the same secret key $(\mathbf{s}^{(\mathrm{even})}, \mathbf{s}^{(\mathrm{odd})})$. Assuming the hardness of $\mathsf{GLWE}_{n,d/2,q,\chi}$, we can use key switching to reduce the dimension of these ciphertexts from $2n + 2$ to $n + 1$.

It should clear that, when d is a power of 2, our observations for moving from a $\mathsf{GLWE}_{n,d,q,\chi}$ ciphertext to two $\mathsf{GLWE}_{2n,d/2,q,\chi}$ ciphertexts can be generalized to move from a $\mathsf{GLWE}_{n,d,q,\chi}$ ciphertext to 2^k $\mathsf{GLWE}_{2^k n,d/2^k,q,\chi}$ ciphertexts for any 2^k up to d.

Observe that, if we only encode messages in the even coefficients from the beginning, and encode the associated $m^{(\text{odd})}$'s as zeroes, then the $m^{(\text{odd})}$'s are zeroes throughout the intermediates stages of the computation, and we obtain the final output message as the $m^{(\text{even})}$ of the final ciphertext. In particular, the second ciphertext can be ignored, and therefore does not affect decryption complexity.

With this technique, we achieve leveled FHE with no bootstrapping based on RLWE. The decryption complexity of the final level-0 ciphertext in $R_{q_0,d_0}^{n_0+1}$ is independent of the homomorphic capacity L of the leveled FHE scheme. Of course, for this magic to work, we need $\mathsf{GLWE}_{n_j,d_j,q_j,\chi_j}$ to be a hard problem for all j.

Generalizations. We suspect that our techniques can be extended to subfields of cyclotomics other than $x^{2^k}+1$, but we leave this for future work.

5 Correctness, Parameter Settings, Performance, and Security

Here, we will show how to set the parameters of the scheme so that the scheme is correct. Mostly, this involves analyzing each of the steps within FHE.Add and FHE.Mult – namely, the addition or multiplication itself, and then the SwitchKey and Scale steps that make up FHE.Refresh – to establish that the output of each step is a decryptable ciphertext with bounded noise. This analysis will lead to concrete suggestions for how to set the ladder of moduli and to asymptotic bounds on the performance of the scheme.

Let us begin by considering how much noise FHE.Enc introduces initially. Throughout, B_χ denotes a bound such that R-elements sampled from the the noise distribution χ have length at most B_χ with overwhelming probability.

5.1 The Initial Noise from FHE.Enc

Recall that FHE.Enc simply invokes E.Enc for suitable parameters ($params_L$) that depend on λ and L. In turn, the noise of ciphertexts output by E.Enc depends on the noise of the initial "ciphertext(s)" (the encryption(s) of 0) implicit in the matrix $\mathbf{A}$ output by E.PublicKeyGen, whose noise distribution is dictated by the distribution χ.

Lemma 6. *Let q, d, n, N be the parameters associated to* FHE.Enc. *Let γ_R be the expansion factor associated to R. (γ_R and d are both 1 in the LWE case $R = \mathbb{Z}$.) The length of the noise in ciphertexts output by* FHE.Enc *is at most $\sqrt{d} + 2 \cdot \gamma_R \cdot \sqrt{d} \cdot N \cdot B_\chi$.*

Proof. We have $\mathbf{A} \cdot \mathbf{s} = 2\mathbf{e}$ where $\mathbf{s} \leftarrow \mathsf{E.SecretKeyGen}$, $\mathbf{A} \leftarrow \mathsf{E.PublicKeyGen}(\mathbf{s}, N)$, and $\mathbf{e} \leftarrow \chi^N$. Recall that encryption works as follows: $\mathbf{c} \leftarrow \mathbf{m} + \mathbf{A}^T\mathbf{r} \bmod q$ where $\mathbf{r} \in R_2^N$. We have that the noise of this ciphertext is $[\langle \mathbf{c}, \mathbf{s} \rangle]_q = [m + 2\langle \mathbf{r}, \mathbf{e} \rangle]_q$. The magnitude of this element is at most

$$\sqrt{d} + 2 \cdot \gamma_R \cdot \sum_{j=1}^{N} \|\mathbf{r}[j]\| \cdot \|\mathbf{e}[j]\| \leq \sqrt{d} + 2 \cdot \gamma_R \cdot \sqrt{d} \cdot N \cdot B_\chi$$

□

One can easily obtain a similar small bound on the noise of ciphertexts output by LPR encryption in the RLWE setting: a small polynomial in the security parameter λ, L, and $\log q$.

The correctness of decryption for ciphertexts output by FHE.Enc, assuming the noise bound above is less than $q/2$, follows directly from the correctness of the basic encryption and decryption algorithms E.Enc and E.Dec.

5.2 Correctness and Performance of FHE.Add and FHE.Mult (before FHE.Refresh)

Consider FHE.Mult. One begins $\mathsf{FHE.Mult}(pk, \mathbf{c}_1, \mathbf{c}_2)$ with two ciphertexts under key $\mathbf{s}_j$ for modulus q_j that have noises $e_i = [L_{\mathbf{c}_i}(\mathbf{s}_j)]_{q_j}$, where $L_{\mathbf{c}_i}(\mathbf{x})$ is simply the dot product $\langle \mathbf{c}_i, \mathbf{x} \rangle$. To multiply together two ciphertexts, one multiplies together these two linear equations to obtain a quadratic equation $Q_{\mathbf{c}_1,\mathbf{c}_2}(\mathbf{x}) \leftarrow L_{\mathbf{c}_1}(\mathbf{x}) \cdot L_{\mathbf{c}_2}(\mathbf{x})$, and then interprets this quadratic equation as a linear equation $L^{long}_{\mathbf{c}_1,\mathbf{c}_2}(\mathbf{x} \otimes \mathbf{x}) = Q_{\mathbf{c}_1,\mathbf{c}_2}(\mathbf{x})$ over the tensored vector $\mathbf{x} \otimes \mathbf{x}$. The coefficients of this long linear equation compose the new ciphertext vector $\mathbf{c}_3$. Clearly, $[\langle \mathbf{c}_3, \mathbf{s}_j \otimes \mathbf{s}_j \rangle]_{q_j} = [L^{long}_{\mathbf{c}_1,\mathbf{c}_2}(\mathbf{s}_j \otimes \mathbf{s}_j)]_{q_j} = [e_1 \cdot e_2]_{q_j}$. Thus, if the noises of $\mathbf{c}_1$ and $\mathbf{c}_2$ have length at most B, then the noise of $\mathbf{c}_3$ has length at most $\gamma_R \cdot B^2$, where γ_R is the expansion factor of R. If this length is less than $q_j/2$, then decryption works correctly. In particular, if $m_i = [\langle \mathbf{c}_i, \mathbf{s}_j \rangle]_{q_j}]_2 = [e_i]_2$ for $i \in \{1, 2\}$, then over R_2 we have $[\langle \mathbf{c}_3, \mathbf{s}_j \otimes \mathbf{s}_j \rangle]_{q_j}]_2 = [[e_1 \cdot e_2]_{q_j}]_2 = [e_1 \cdot e_2]_2 = [e_1]_2 \cdot [e_2]_2 = m_1 \cdot m_2$. That is, correctness is preserved as long as this noise does not wrap modulo q_j.

The correctness of FHE.Add and FHE.Mult, before the FHE.Refresh step is performed, is formally captured in the following lemmas.

Lemma 7. *Let $\mathbf{c}_1$ and $\mathbf{c}_2$ be two ciphertexts under key $\mathbf{s}_j$ for modulus q_j, where $\|[\langle \mathbf{c}_i, \mathbf{s}_j \rangle]_{q_j}\| \leq B$ and $m_i = [[\langle \mathbf{c}_i, \mathbf{s}_j \rangle]_{q_j}]_2$. Let $\mathbf{s}'_j = \mathbf{s}_j \otimes \mathbf{s}_j$, where the "non-quadratic coefficients" of $\mathbf{s}'_j$ (namely, the '1' and the coefficients of $\mathbf{s}_j$) are placed first. Let $\mathbf{c}' = \mathbf{c}_1 + \mathbf{c}_2$, and pad $\mathbf{c}'$ with zeros to get a vector $\mathbf{c}_3$ such that $\langle \mathbf{c}_3, \mathbf{s}'_j \rangle = \langle \mathbf{c}', \mathbf{s}_j \rangle$. The noise $[\langle \mathbf{c}_3, \mathbf{s}'_j \rangle]_{q_j}$ has length at most $2B$. If $2B < q_j/2$, $\mathbf{c}_3$ is an encryption of $m_1 + m_2$ under key $\mathbf{s}'_j$ for modulus q_j – i.e., $m_1 \cdot m_2 = [[\langle \mathbf{c}_3, \mathbf{s}'_j \rangle]_{q_j}]_2$.*

Lemma 8. *Let $\mathbf{c}_1$ and $\mathbf{c}_2$ be two ciphertexts under key $\mathbf{s}_j$ for modulus q_j, where $\|[\langle \mathbf{c}_i, \mathbf{s}_j \rangle]_{q_j}\| \leq B$ and $m_i = [[\langle \mathbf{c}_i, \mathbf{s}_j \rangle]_{q_j}]_2$. Let the linear equation $L^{long}_{\mathbf{c}_1,\mathbf{c}_2}(\mathbf{x} \otimes \mathbf{x})$ be as defined above, let $\mathbf{c}_3$ be the coefficient vector of this linear equation, and let $\mathbf{s}'_j = \mathbf{s}_j \otimes \mathbf{s}_j$. The noise $[\langle \mathbf{c}_3, \mathbf{s}'_j \rangle]_{q_j}$ has length at most $\gamma_R \cdot B^2$. If $\gamma_R \cdot B^2 < q_j/2$, $\mathbf{c}_3$ is an encryption of $m_1 \cdot m_2$ under key $\mathbf{s}'_j$ for modulus q_j – i.e., $m_1 \cdot m_2 = [[\langle \mathbf{c}_3, \mathbf{s}'_j \rangle]_{q_j}]_2$.*

The computation needed to compute the tensored ciphertext c_3 is $\tilde{O}(d_j n_j^2 \log q_j)$. For the RLWE instantiation, since $n_j = 1$ and since (as we will see) d_j (resp. $\log q_j$) depend only quasi-linearly (resp. logarithmically) on the security parameter and linearly (resp. linearly) on L, the computation here is only quasi-linear in the security parameter. For the LWE instantiation, the computation is quasi-quadratic.

5.3 Correctness and Performance of FHE.Refresh

FHE.Refresh consists of two steps: Switch Keys and Switch Moduli. We address each of these steps in turn.

Correctness and Performance of the Switch-Key Step. In the Switch Keys step, we take as input a ciphertext $\mathbf{c}$ under key $\mathbf{s}'_j$ for modulus q_j and set $\mathbf{c}_1 \leftarrow \mathsf{SwitchKey}(\tau_{\mathbf{s}'_j \to \mathbf{s}_{j-1}}, \mathbf{c}, q_j)$, a ciphertext under the key $\mathbf{s}_{j-1}$ for modulus q_j. In Lemma 3, we proved the correctness of key switching and showed that the noise grows by the additive factor $2\,\langle \mathsf{BitDecomp}(\mathbf{c}, q_j), \mathbf{e}\rangle$, where $\mathsf{BitDecomp}(\mathbf{c}, q_j)$ is a (short) bit-vector and $\mathbf{e}$ is a (short and fresh) noise vector with elements sampled from χ. In particular, if the noise originally had length B, then after the Switch Keys step it has length at most $B+2\cdot\gamma_R\cdot B_\chi\cdot\sum_{i=1}^{w_j}\|\mathsf{BitDecomp}(\mathbf{c}, q_j)[i]\| \leq B+2\cdot\gamma_R\cdot B_\chi\cdot w_j\cdot\sqrt{d_j}$, where $w_j \leq \binom{n_j+1}{2}\cdot\lceil\log q_j\rceil$ is the dimension of $\mathsf{BitDecomp}(\mathbf{c}, q_j)$.

We capture the correctness of the Switch-Key step in the following lemma.

Lemma 9. *Let $\mathbf{c}$ be a ciphertext under the key $\mathbf{s}'_j = \mathbf{s}_j \otimes \mathbf{s}_j$ for modulus q_j such that $e_1 \leftarrow [\langle \mathbf{c}, \mathbf{s}'_j\rangle]_{q_j}$ has length at most B and $m = [e_1]_2$. Let $\mathbf{c}_1 \leftarrow \mathsf{SwitchKey}(\tau_{\mathbf{s}'_j\to\mathbf{s}_{j-1}}, \mathbf{c}, q_j)$, and let $e_2 = [\langle \mathbf{c}_1, \mathbf{s}_{j-1}\rangle]_{q_j}$. Then, e_2 (the new noise) has length at most $B + 2\cdot\gamma_R\cdot B_\chi\cdot\binom{n_j+1}{2}\cdot\lceil\log q_j\rceil\cdot\sqrt{d_j}$ and (assuming this noise length is less than $q_j/2$) we have $m = [e_2]_2$.*

The Switch-Key step involves multiplying the transpose of w_j-dimensional vector $\mathsf{BitDecomp}(\mathbf{c}, q_j)$ with a $w_j \times (n_j+1)$ matrix $\mathbf{B}$. This computation is $\tilde{O}(d_j n_j^3 \log^2 q_j)$. Still this is quasi-linear in the RLWE instantiation.

<u>Correctness and Performance of the Switch-Moduli Step</u>. The Switch Moduli step takes as input a ciphertext $\mathbf{c}_1$ under the secret bit-vector $\mathbf{s}_{j-1}$ for the modulus q_j, and outputs the ciphertext $\mathbf{c}_2 \leftarrow \mathsf{Scale}(\mathbf{c}_1, q_j, q_{j-1}, 2)$, which we claim to be a ciphertext under key $\mathbf{s}_{j-1}$ for modulus q_{j-1}. Note that $\mathbf{s}_{j-1}$ is a *short* secret key. By Corollary 1, and using the fact that $\ell_1(\mathbf{s}_{j-1}) \leq (n_{j-1}+1)\cdot B_\chi$, the following is true: if the noise of $\mathbf{c}_1$ has length at most $B < q_j/2-(q_j/q_{j-1})\cdot\sqrt{d_j}\cdot\gamma_R\cdot(n_{j-1}+1)\cdot B_\chi$, then correctness is preserved and the noise of $\mathbf{c}_2$ is bounded by $(q_{j-1}/q_j)\cdot B+\sqrt{d_j}\cdot\gamma_R\cdot(n_{j-1}+1)\cdot B_\chi$. Of course, the key feature of this step for our purposes is that switching moduli may *reduce* the length of the moduli when $q_{j-1} < q_j$.

We capture the correctness of the Switch-Moduli step in the following lemma.

Lemma 10. *Let $\mathbf{c}_1$ be a ciphertext under the key $\mathbf{s}_{j-1}$, sampled from $\chi^{n_{j-1}}$, such that $e_j \leftarrow [\langle \mathbf{c}_1, \mathbf{s}_{j-1}\rangle]_{q_j}$ has length at most B and $m = [e_j]_2$. Let $\mathbf{c}_2 \leftarrow \mathsf{Scale}(\mathbf{c}_1, q_j, q_{j-1}, 2)$, and let $e_{j-1} = [\langle \mathbf{c}_2, \mathbf{s}_{j-1}\rangle]_{q_{j-1}}$. Then, e_{j-1} (the new noise) has length at most $(q_{j-1}/q_j)\cdot B+\sqrt{d_j}\cdot\gamma_R\cdot(n_{j-1}+1)\cdot B_\chi$, and (assuming this noise length is less than $q_{j-1}/2$) we have $m = [e_{j-1}]_2$.*

The computation in the Switch-Moduli step is $\tilde{O}(d_j n_{j-1}\log q_j)$.

5.4 Putting the Pieces Together: Parameters, Correctness, Performance

So far we have established that the scheme is correct, *assuming* that the noise does not wrap modulo q_j or q_{j-1}. Now we need to show that we can set the parameters of the scheme to ensure that such wrapping never occurs.

Our strategy for setting the parameters is to pick a "universal" bound B on the noise length, and then prove, for all j, that a valid ciphertext under key $\mathbf{s}_j$ for modulus q_j has noise length at most B. This bound B is quite small: polynomial in λ and $\log q_L$, where q_L is the largest modulus in our ladder. It is clear that such a bound B holds for fresh ciphertexts output by $\mathsf{FHE.Enc}$. (Recall the discussion from Section 3.1 where we explained that we use a noise distribution χ that is essentially independent of the modulus.) The remainder of the proof is by induction – i.e., we will show that if the bound holds for two ciphertexts $\mathbf{c}_1, \mathbf{c}_2$ at level j, our lemmas above imply that the bound also holds for the ciphertext $\mathbf{c}' \leftarrow \mathsf{FHE.Mult}(pk, \mathbf{c}_1, \mathbf{c}_2)$ at level $j-1$. ($\mathsf{FHE.Mult}$ increases the noise strictly more in the worst-case than $\mathsf{FHE.Add}$ for any reasonable choice of parameters.)

Specifically, after the first step of FHE.Mult (without the Refresh step), the noise has length at most $\gamma_R \cdot B^2$. Then, we apply the SwitchKey function, which introduces an additive term $\eta_{\mathsf{SwitchKey},j}$. Finally, we apply the Scale function. The noise is now at most

$$(q_{j-1}/q_j) \cdot \left(\gamma_R \cdot B^2 + \eta_{\mathsf{SwitchKey},j}\right) + \eta_{\mathsf{Scale},j}$$

where $\eta_{\mathsf{Scale},j}$ is another additive term. Now we want to choose our parameters so that this bound is at most B.

Suppose we set our ladder of moduli and the bound B such that the following two properties hold:

- Property 1: $B \geq 2 \cdot (\eta_{\mathsf{Scale},j} + \eta_{\mathsf{SwitchKey},j})$ for all j.
- Property 2: $q_j/q_{j-1} \geq 2 \cdot B \cdot \gamma_R$ for all j.

Then we have

$$\begin{aligned}
&(q_{j-1}/q_j) \cdot \left(\gamma_R \cdot B^2 + \eta_{\mathsf{SwitchKey},j}\right) + \eta_{\mathsf{Scale},j} \\
&\quad < (q_{j-1}/q_j) \cdot \gamma_R \cdot B^2 + \eta_{\mathsf{Scale},j} + \eta_{\mathsf{SwitchKey},j} \\
&\quad \leq \frac{1}{2 \cdot B \cdot \gamma_R} \cdot \gamma_R \cdot B^2 + \frac{1}{2} \cdot B \\
&\quad \leq B
\end{aligned}$$

It only remains to set our ladder of moduli and B so that Properties 1 and 2 hold.

Unfortunately, there is some circularity in Properties 1 and 2: q_L depends on B, which depends on q_L, albeit only polylogarithmically. However, it is easy to see that this circularity is not fatal. As a non-optimized example to illustrate this, set $B = \lambda^a \cdot L^b$ for very large constants a and b, and set $q_j \approx 2^{(j+1)\cdot\omega(\log\lambda+\log L)}$. If a and b are large enough, B dominates $\eta_{\mathsf{Scale},L} + \eta_{\mathsf{SwitchKey},L}$, which is polynomial in λ and $\log q_L$, and hence polynomial in λ and L (Property 1 is satisfied). Since q_j/q_{j-1} is super-polynomial in both λ and L, it dominates $2 \cdot B \cdot \gamma_R$ (Property 2 is satisfied). In fact, it works fine to set q_j as a modulus having $(j+1)\cdot\mu$ bits for some $\mu = \theta(\log\lambda + \log L)$ with small hidden constant.

Overall, we have that q_L, the largest modulus used in the system, is $\theta(L \cdot (\log\lambda + \log L))$ bits, and $d_L \cdot n_L$ must be approximately that number times λ for 2^λ security.

Theorem 4. *For some $\mu = \theta(\log\lambda+\log L)$, FHE is a correct L-leveled FHE scheme – specifically, it correctly evaluates circuits of depth L with Add and Mult gates over R_2. The per-gate computation is $\tilde{O}(d_L \cdot n_L^3 \cdot \log^2 q_j) = \tilde{O}(d_L \cdot n_L^3 \cdot L^2)$. For the LWE case (where $d = 1$), the per-gate computation is $\tilde{O}(\lambda^3 \cdot L^5)$. For the RLWE case (where $n = 1$), the per-gate computation is $\tilde{O}(\lambda \cdot L^3)$.*

The bottom line is that we have a RLWE-based leveled FHE scheme with per-gate computation that is only *quasi-linear* in the security parameter, albeit with somewhat high dependence on the number of levels in the circuit.

Let us pause at this point to reconsider the performance of previous FHE schemes in comparison to our new scheme. Specifically, as we discussed in the Introduction, in previous SWHE schemes, the ciphertext size is at least $\tilde{O}(\lambda \cdot d^2)$, where d is the *degree* of the circuit being evaluated. One may view our new scheme as a very powerful SWHE scheme in which this dependence on *degree* has been replaced with a similar dependence on *depth*. (Recall the degree of a circuit may be exponential in its depth.) Since polynomial-size circuits have polynomial depth, which is certainly not true of *degree*, our scheme can efficiently evaluate arbitrary circuits without resorting to bootstrapping.

5.5 Security

The security of FHE follows by a standard hybrid argument from the security of E, the basic scheme described in Section 3.1.

Theorem 5 (security)**.** *Let $n = n(\lambda)$, $d = d(\lambda)$, $q = q(\lambda)$, $\chi = \chi(\lambda)$ and $L = L(\lambda)$ be functions of the security parameter. Let $N := n \cdot \mathrm{polylog}(q, \lambda)$. The scheme FHE is CPA secure under the GLWE assumption with parameters n, d, q and χ.*

In a high level, the idea is this: the view of a CPA adversary for our scheme is very similar to that for the basic scheme E, except that our adversary also gets to see the key switching parameters. However, the key switching parameters are made up of a sequence of outputs of the $\mathsf{SwitchKeyGen}$ algorithm which, by Lemma 4, are indistinguishable from uniform. A formal proof follows by a hybrid argument.

Proof. We proceed with the proof using a sequence of hybrids. Let $\mathcal{A}$ be an IND-CPA adversary for FHE. We consider a series of hybrids where $\mathsf{Adv}_H[\mathcal{A}]$ denotes the success probability of $\mathcal{A}$ in hybrid H.

- **Hybrid H_0:** This is identical to the IND-CPA game, where the adversary gets a properly distributed public key, generated by $\mathsf{FHE.KeyGen}$, and an encryption of either 0 or 1 computed using $\mathsf{FHE.Enc}$. Recall that the public key consists of:
  - a matrix $\mathbf{A}_L \leftarrow \mathsf{E.PublicKeyGen}(\mathbf{s}_L)$; and
  - pairs $(\mathbf{A}_\ell, \tau_{\mathbf{s}_{(\ell+1)} \to \mathbf{s}_\ell})$ for $\ell = L-1$ downto 0, where $\mathbf{A}_\ell \leftarrow \mathsf{E.PublicKeyGen}(\mathbf{s}_\ell)$ and $\tau_{\mathbf{s}_{(\ell+1)} \to \mathbf{s}_\ell} \leftarrow \mathsf{SwitchKeyGen}(\mathbf{s}_{\ell+1} \otimes \mathbf{s}_{\ell+1}, \mathbf{s}_\ell)$.

  For the sake of contradiction, assume that there is a polynomial function $p(\cdot)$ such that

  $$\mathsf{Adv}_{H_0}[\mathcal{A}] := \big|\Pr[\mathcal{A}(pk, \mathsf{FHE.Enc}(pk, \mu_0) = 1] - \Pr[\mathcal{A}(pk, \mathsf{FHE.Enc}(pk, \mu_1) = 1]\big| > 1/p(\lambda). \quad (1)$$

- **Hybrid H_ℓ, for $\ell \in [L]$:** The hybrids H_ℓ are identical to $H_{\ell-1}$ except that the public key $\mathbf{A}_{\ell-1}$ and the key switching parameter $\tau_{\mathbf{s}_\ell \to \mathbf{s}_{\ell-1}}$ are chosen to be uniformly random from the appropriate domains. We claim that

  $$|\mathsf{Adv}_{H_\ell}[\mathcal{A}] - \mathsf{Adv}_{H_{\ell-1}}[\mathcal{A}]| \le \mathsf{negl}(\lambda). \quad (2)$$

  Assume for contradiction that there is an adversary $\mathcal{A}$ that distinguishes between H_ℓ and $H_{\ell-1}$ with non-negligible advantage $1/q(\lambda)$ for some polynomial $q(\cdot)$. Then, we construct a distinguisher $\mathcal{B}$ that distinguishes between a correctly generated pair $(\mathbf{A}_\ell, \tau_{\mathbf{s}_{(\ell+1)} \to \mathbf{s}_\ell})$ and a uniformly random tuple of the same dimensions, for some $\mathbf{s}_{\ell-1}$. Such a distinguisher cannot exist, by Lemma 4.

  The distinguisher $\mathcal{B}$ picks vectors $\mathbf{s}_\ell, \ldots, \mathbf{s}_{L-1}$ uniformly at random and generates pairs $(\mathbf{A}_\ell, \tau_{\mathbf{s}_{(\ell+1)} \to \mathbf{s}_\ell})$ where

  $$\mathbf{A}_\ell \leftarrow \mathsf{E.PublicKeyGen}(\mathbf{s}_\ell) \text{ and } \tau_{\mathbf{s}_{(\ell+1)} \to \mathbf{s}_\ell} \leftarrow \mathsf{SwitchKeyGen}(\mathbf{s}_{\ell+1} \otimes \mathbf{s}_{\ell+1}, \mathbf{s}_\ell)$$

  and also generates $\mathbf{A}_L \leftarrow \mathsf{E.PublicKeyGen}(\mathbf{s}_L)$. $\mathcal{B}$ then sets $(\mathbf{A}_{\ell-1}, \tau_{\mathbf{s}_\ell \to \mathbf{s}_{\ell-1}})$ to be its own challenge input, and chooses the rest of the components of the public key at random from their respective domains. Finally, it encrypts a random bit b and feeds it to the distinguisher $\mathcal{A}$. If $\mathcal{A}$ returns a bit $b' = b$, then output 1, else output 0.

It is easy to see that the distribution generated by $\mathcal{B}$ is either $H_{\ell-1}$ or H_ℓ depending on whether its challenge input was correctly generated or random. This proves our claim.

Note that in hybrid H_L, all the pairs $(\mathbf{A}_{\ell-1}, \tau_{\mathbf{s}_\ell \to \mathbf{s}_{\ell-1}})$ are uniformly random for every $\ell \in [L]$.

- **Hybrid H_{L+1}:** Hybrid H_{L+1} is identical to H_L except that the matrix $\mathbf{A}_L$ is chosen uniformly at random rather than being generated as an output of $\mathsf{E.PublicKeyGen}(\mathbf{s}_L)$. It follows that

$$|\mathsf{Adv}_{H_{L+1}}[\mathcal{A}] - \mathsf{Adv}_{H_L}[\mathcal{A}]| \leq \mathsf{negl}(\lambda). \tag{3}$$

by an argument essentially identical to the one above.

Note that in H_{L+1}, all the elements of the public key are uniformly random and independent of the message. Thus, invoking the semantic security of the basic encryption scheme E, we get that

$$\mathsf{Adv}_{H_{L+1}}[\mathcal{A}] = \mathsf{Adv}_{\mathsf{E},cpa}[\mathcal{A}] = \mathsf{negl}(\lambda) \ . \tag{4}$$

Putting equations 2, 3 and 4 together, we get

$$\mathsf{Adv}_{H_0}[\mathcal{A}] \leq \mathsf{Adv}_{H_{L+1}}[\mathcal{A}] + \sum_{\ell=0}^{L} |\mathsf{Adv}_{H_\ell}[\mathcal{A}] - \mathsf{Adv}_{H_{\ell+1}}[\mathcal{A}]| = \mathsf{negl}(\lambda)$$

contradicting equation 1. The theorem follows. □

6 Optimizations: Reducing the Per-gate Computation Overhead

Despite the fact that the RLWE-based version of our new FHE scheme has per-gate computation only quasi-linear in the security parameter, it has a rather large dependence on the number of levels in the circuit (see Theorem 4). In this section, we show how to reduce the per-gate computation overhead to quasi-linear in the security parameter, *independent* of the number of levels in the circuit to be evaluated. To this end, we present several significant optimizations.

Our first optimization (Section 6.1) is *batching*. Batching allows us to evaluate a function f homomorphically in parallel on $\ell = \Omega(\lambda)$ blocks of encrypted data, paying only a *polylogarithmic* (in the security parameter λ) overhead over evaluating f on the unencrypted data. (The overhead is still polynomial in the depth L of the circuit computing f.) Thus, batching allows us to reduce the per-gate computation from quasi-linear in the security parameter to *polylogarithmic*, in an amortized sense. Batching works essentially by packing multiple plaintexts into each ciphertext.

Secondly, we bring *bootstrapping* back into the picture as an optimization rather than a necessity (Section 6.2). Bootstrapping allows us to achieve per-gate computation *quasi-quadratic* in the security parameter, *independent* of the number levels in the circuit being evaluated. Although the dependence on the security parameter has worsened from $\tilde{O}(\lambda)$ to $\tilde{O}(\lambda^2)$, the key win here is that the per-gate computation does not depend on the depth of the circuit any more! This optimization applies in a stand-alone setting where the function is evaluated on a single input.

Finally, we obtain our goal of near-linear per-gate computation overhead (in the security parameter λ) by showing that *batching the bootstrapping function* is a powerful combination (Section 6.3). With this optimization, circuits that are wide on average, namely ones whose average width is at least λ, can be evaluated homomorphically with only $\tilde{O}(\lambda)$ per-gate computation, independent of the number of levels.

6.1 Batching

Suppose we want to evaluate the same function f on ℓ blocks of encrypted data. (Or, similarly, suppose we want to evaluate the same encrypted function f on ℓ blocks of plaintext data.) Can we do this using less than ℓ times the computation needed to evaluate f on one block of data? Can we batch?

For example, consider a keyword search function that returns '1' if the keyword is present in the data and '0' if it is not. The keyword search function is mostly composed of a large number of equality tests that compare the target word w to all of the different subsequences of data; this is followed up by an OR of the equality test results. All of these equality tests involve running the same w-dependent function on different blocks of data. If we could batch these equality tests, it could significantly reduce the computation needed to perform keyword search homomorphically.

If we use bootstrapping as an optimization (see Section 6.2), then obviously we will be running the decryption function homomorphically on multiple blocks of data – namely, the multiple ciphertexts that need to be refreshed. Can we batch the bootstrapping function? If we could, then we might be able to drastically reduce the average per-gate cost of bootstrapping.

Smart and Vercauteren [SV11] were the first to rigorously analyze batching in the context of FHE. In particular, they observed that ideal-lattice-based (and RLWE-based) ciphertexts can have many plaintext slots, associated to the factorization of the plaintext space into algebraic ideals.

When we apply batching to our new RLWE-based FHE scheme, the results are pretty amazing. Evaluating f homomorphically on $\ell = \Omega(\lambda)$ blocks of encrypted data requires only *polylogarithmically* (in terms of the security parameter λ) more computation than evaluating f on the unencrypted data. (The overhead is still polynomial in the depth L of the circuit computing f.) As we will see later, for circuits whose levels have average width at least λ, *batching the bootstrapping function* (i.e., batching homomorphic evaluation of the decryption function) allows us to reduce the per-gate computation of our bootstrapped scheme from $\tilde{O}(\lambda^2)$ to $\tilde{O}(\lambda)$ (independent of L).

To make the exposition a bit simpler, in our RLWE-based instantiation where $R = \mathbb{Z}[x]/(x^d+1)$, we will not use R_2 as our plaintext space, but instead use a plaintext space R_p, prime $p = 1 \bmod 2d$, where we have the isomorphism $R_p \cong R_{\mathfrak{p}_1} \times \cdots \times R_{\mathfrak{p}_d}$ of many plaintext spaces (think Chinese remaindering), so that evaluating a function *once* over R_p implicitly evaluates the function *many* times in parallel over the respective smaller plaintext spaces. The $\mathfrak{p}_i$'s will be *ideals* in our ring $R = \mathbb{Z}[x]/(x^d+1)$. (One could still use R_2 as in [SV11], but the number theory there is a bit more involved.)

6.1.1 Some Number Theory

Let us take a very brief tour of algebraic number theory. Suppose p is a prime number satisfying $p = 1 \bmod 2d$, and let a be a primitive $2d$-th root of unity modulo p. Then, $x^d + 1$ factors completely into linear polynomials modulo p – in particular, $x^d + 1 = \prod_{i=1}^{d}(x - a_i) \bmod p$ where $a_i = a^{2i-1} \bmod p$. In some sense, the converse of the above statement is also true, and this is the essence of *reciprocity* – namely, in the ring $R = \mathbb{Z}[x]/(x^d + 1)$ the prime integer p is not actually prime, but rather it splits completely into prime ideals in R – i.e., $p = \prod_{i=1}^{d} \mathfrak{p}_i$. The ideal $\mathfrak{p}_i$ equals $(p, x - a_i)$ – namely, the set of all R-elements that can be expressed as $r_1 \cdot p + r_2 \cdot (x - a_i)$ for some $r_1, r_2 \in R$. Each ideal $\mathfrak{p}_i$ has norm p – that is, roughly speaking, a $1/p$ fraction of R-elements are in $\mathfrak{p}_i$, or, more formally, the p cosets $0 + \mathfrak{p}_i, \ldots, (p-1) + \mathfrak{p}_i$ partition R. These ideals are relatively prime, and so they behave like relatively prime integers. In particular, the Chinese Remainder

Theorem applies: $R_p \cong R_{\mathfrak{p}_1} \times \cdots \times R_{\mathfrak{p}_d}$.

Although the prime ideals $\{\mathfrak{p}_i\}$ are relatively prime, they are close siblings, and it is easy, in some sense, to switch from one to another. One fact that we will use (when we finally apply batching to bootstrapping) is that, for any i, j there is an automorphism $\sigma_{i \to j}$ over R that maps elements of $\mathfrak{p}_i$ to elements of $\mathfrak{p}_j$. Specifically, $\sigma_{i \to j}$ works by mapping an R-element $r = r(x) = r_{d-1}x^{d-1} + \cdots + r_1 x + r_0$ to $r(x^{e_{ij}}) = r_{d-1}x^{e_{ij}(d-1) \bmod 2d} + \cdots + r_1 x^{e_{ij}} + r_0$ where e_{ij} is some number relative prime to $2d$. Notice that this automorphism just permutes the coefficients of r and fixes the free coefficient. Notationally, we will use $\sigma_{i \to j}(\mathbf{v})$ to refer to the vector that results from applying $\sigma_{i \to j}$ coefficient-wise to $\mathbf{v}$.

6.1.2 How Batching Works

We will show that the following holds:

Theorem 6. *Let $p = 1 \bmod 2d$ be a prime of size polynomial in λ. The* RLWE*-based instantiation of* FHE *using the ring $R = \mathbb{Z}[x]/(x^d + 1)$ can be adapted to use the plaintext space $R_p = \otimes_{i=1}^{d} R_{\mathfrak{p}_i}$ while preserving correctness and the same asymptotic performance. For any boolean circuit f of depth L, the scheme can homomorphically evaluate f on ℓ sets of inputs with per-gate computation $\tilde{O}(\lambda \cdot L^3 / \min\{d, \ell\})$.*

When $\ell = \Omega(\lambda)$, and taking $d = \tilde{\Omega}(\lambda)$, the per-gate computation is only polylogarithmic in the security parameter (still cubic in L).

Proof. Deploying batching inside our scheme FHE is quite straightforward. First, we pick a prime $p = 1 \bmod 2d$ of size polynomial in the security parameter. (One should exist under the GRH.)

The next step is simply to recognize that our scheme FHE works just fine when we replace the original plaintext space R_2 with R_p. There is nothing especially magical about the number 2. In the basic scheme E described in Section 3.1, $\mathsf{E.PublicKeyGen}(params, sk)$ is modified in the obvious way so that $\mathbf{A} \cdot \mathbf{s} = p \cdot \mathbf{e}$ rather than $2 \cdot \mathbf{e}$. (This modification induces a similar modification in $\mathsf{SwitchKeyGen}$.) Decryption becomes $m = [[\langle \mathbf{c}, \mathbf{s} \rangle]_q]_p$. Homomorphic operations use mod-p gates rather than boolean gates, and it is easy (if desired) to emulate boolean gates with mod-p gates – e.g., we can compute $\text{XOR}(a, b)$ for $a, b \in \{0, 1\}^2$ using mod-p gates for any p as $a + b - 2ab$. For modulus switching, we use $\mathsf{Scale}(\mathbf{c}_1, q_j, q_{j-1}, p)$ rather than $\mathsf{Scale}(\mathbf{c}_1, q_j, q_{j-1}, 2)$. The larger rounding error from this new scaling procedure increases the noise slightly, but this additive noise is still polynomial in the security parameter and the number of levels, and thus is still consistent with our setting of parameters. In short, FHE can easily be adapted to work with a plaintext space R_p for p of polynomial size.

The final step is simply to recognize that, by the Chinese Remainder Theorem, evaluating an arithmetic circuit over R_p on input $\mathbf{x} \in R_p^n$ implicitly evaluates, for each i, the same arithmetic circuit over $R_{\mathfrak{p}_i}$ on input $\mathbf{x}$ projected down to $R_{\mathfrak{p}_i}^n$. The evaluations modulo the various prime ideals do not "mix" or interact with each other. □

6.2 Bootstrapping as an Optimization

Bootstrapping is no longer strictly necessary to achieve leveled FHE. However, in some settings, it may have some advantages:

- Performance: The per-gate computation is independent of the depth of the circuit being evaluated.
- Flexibility: Assuming circular security, a bootstrapped scheme can perform homomorphic evaluations indefinitely without needing to specify in advance, during Setup, a bound on the number of circuit levels.
- Memory: Bootstrapping permits short ciphertexts – e.g., encrypted using AES other space-efficient cryptosystem – to be de-compressed to longer ciphertexts that permit homomorphic operations. Bootstrapping thus allows us to save memory by storing data encrypted in the compressed form, while retaining the ability to perform homomorphic operations.

Here, we revisit bootstrapping, viewing it as an optimization rather than a necessity. We also reconsider the scheme FHE that we described in Section 3, viewing the scheme not as an end in itself, but rather as a very powerful SWHE whose performance degrades polynomially in the *depth* of the circuit being evaluated, as opposed to previous SWHE schemes whose performance degrades polynomially in the *degree*. In particular, we analyze how efficiently it can evaluate its decryption function, as needed to bootstrap. Not surprisingly, our faster SWHE scheme can also bootstrap faster. The decryption function has only logarithmic depth and can be evaluated homomorphically in time quasi-quadratic in the security parameter (for the RLWE instantiation), giving a bootstrapped scheme with quasi-quadratic per-gate computation overall.

To apply the Bootstrapping Theorem (Theorem 1), we need to bound the circuit complexity of the decryption of our scheme. We start by recalling that the decryption function is $m = [[\langle \mathbf{c}, \mathbf{s} \rangle]_q]_2$.[8] Suppose that we are given the "bits" (elements in R_2) of $\mathbf{s}$ as input, and we want to compute $[[\langle \mathbf{c}, \mathbf{s} \rangle]_q]_2$ using an arithmetic circuit that has Add and Mult gates over R_2. (When we bootstrap, of course we are given the bits of $\mathbf{s}$ in encrypted form.) Note that we will run the decryption function homomorphically on level-0 ciphertexts – i.e., when q is small, only polynomial in the security parameter. What is the complexity of this circuit? Most importantly for our purposes, what is its depth and size? The answer is that we can perform decryption with $\tilde{O}(\lambda)$ computation and $O(\log \lambda)$ depth. Thus, in the RLWE instantiation, we can evaluate the decryption function *homomorphically* using our new scheme with quasi-quadratic computation. (For the LWE instantiation, the bootstrapping computation is quasi-quadratic.)

Lemma 11. *Let $n = O(\lambda)$, $q = \mathrm{poly}(\lambda)$ and let $\mathbf{c} \in \mathbb{Z}_q^n$. Consider the function $D(\mathbf{s}) = [[\langle \mathbf{c}, \mathbf{s} \rangle]_q]_2$, for inputs $\mathbf{s} \in \mathbb{Z}_q^n$, where the elements of $\mathbf{s}$ are represented using standard binary representation. Then $D(\cdot)$ can be computed by a circuit of size $\tilde{O}(\lambda)$ and depth $O(\log \lambda)$.*

Proof. Obviously, each product $\mathbf{c}[i] \cdot \mathbf{s}[i]$ can be written as the sum of at most $\log q$ "shifts" of $\mathbf{s}[i]$. These horizontal shifts of $\mathbf{s}[i]$ use at most $2 \log q$ columns. Thus, $\langle \mathbf{c}, \mathbf{s} \rangle$ can be written as the sum of $n \cdot \log q$ numbers, where each number has $2 \log q$ digits. As discussed in [Gen09b], we can use the three-for-two trick, which takes as input three numbers in binary (of arbitrary length) and outputs (using constant depth) two binary numbers with the same sum. Thus, with $O(\log(n \cdot \log q)) = O(\log n + \log \log q)$ depth and $O(n \log^2 q)$ computation, we obtain two numbers with the desired

[8] In the previous section we argued towards using $m = [[\langle \mathbf{c}, \mathbf{s} \rangle]_q]_p$. However, as we explained there, we can emulate any binary circuit using a modulo-p circuit with only constant blowup. Therefore it is sufficient to consider the binary complexity of decryption.

sum, each having $O(\log n + \log q)$ bits. We can sum the final two numbers with $O(\log\log n + \log\log q)$ depth and $O(\log n + \log q)$ computation. So far, we have used depth $O(\log n + \log\log q)$ and $O(n \log^2 q)$ computation to compute $\langle \mathbf{c}, \mathbf{s} \rangle$. Reducing this value modulo q is an operation akin to division, for which there are circuits of size $\mathrm{polylog}(q)$ and depth $\log\log q$. Finally, reducing modulo 2 just involves dropping the most significant bits. Overall, since we are interested only in the case where $\log q = O(\log \lambda)$, we have that decryption requires $\tilde{O}(\lambda)$ computation and depth $O(\log \lambda)$. $\square$

The above lemma generalizes to GLWE as follows.

Lemma 12. *Let $q = \mathrm{poly}(\lambda)$, n, d s.t. d is a power of 2 and $n \cdot d = O(\lambda)$, $R = \mathbb{Z}[x]/(x^d + 1)$, and $\mathbf{c} \in R_q^n$. Consider the function $D(\mathbf{s}) = [[\langle \mathbf{c}, \mathbf{s} \rangle]_q]_2$, for inputs $\mathbf{s} \in R_q^n$, where the coefficients of $\mathbf{s}$ are represented using standard binary representation. Then $D(\cdot)$ can be computed by a circuit of size $\tilde{O}(\lambda)$ and depth $O(\log \lambda)$.*

Proof. The proof is practically identical to Lemma 11, except we use DFT to multiply elements in R instead of "standard" integer multiplication. Since all roots of $x^d + 1$ are $2d$th roots of unity, we can use FFT to achieve multiplication in logarithmic depth and quasi-linear size (in $d \log q$).

The procedure thus proceeds exactly as in Lemma 11 except the last operations of taking modulo q and then modulo 2 are preformed d times, but since these only require $\log(q)$ depth and $\mathrm{polylog}(q)$ size, the bound still holds. $\square$

We can now apply the Bootstrapping Theorem to our scheme from Section 3. The relevant values of n, d, q when applying Lemma 12 are those of the last level (level L) of the scheme, since these are the values that are actually being decrypted. While modulus reduction guarantees that the values of q_L, n_L are independent of L, it is normally the case that d_L depend on L (think e.g. about the case of $n = 1$). It may seem therefore that the decryption depth depends on L contrary to the compactness requirement. Using the techniques of Section 4, we can convert d_L back down to d_0 which is independent of L. (We remark, however, that even if we don't make the conversion, the dependence on L is small enough that our scheme is bootstrappable.)

Putting the pieces together, we get the following.

Theorem 7. *The scheme* FHE *with parameters n, d, L s.t. $n \cdot d = O(\lambda)$ and $L = O(\log \lambda)$ is bootstrappable. The per-gate computation is $\tilde{O}(\lambda^4)$ for the* LWE *case ($d = 1$) and $\tilde{O}(\lambda^2)$ for the* RLWE *case ($n = 1$).*

Proof. The theorem is a straightforward derivation from combining Theorem 1, Lemma 12 and Theorem 4. $\square$

6.3 Batching the Bootstrapping Operation

Suppose that we are evaluating a circuit homomorphically, that we are currently at a level in the circuit that has at least d gates (where d is the dimension of our ring), and that we want to bootstrap (refresh) all of the ciphertexts corresponding to the respective wires at that level. That is, we want to homomorphically evaluate the decryption function at least d times in parallel. This seems like an ideal place to apply batching.

However, there are some nontrivial problems. In Section 6.1, our focus was rather limited. For example, we did not consider whether homomorphic operations could continue after the batched computation. Indeed, at first glance, it would appear that homomorphic operations *cannot* continue, since, after batching, the encrypted data is partitioned into non-interacting relatively-prime plaintext slots, whereas the whole point of homomorphic encryption is that the encrypted data can interact (within a common plaintext slot). Similarly, we did not consider homomorphic operations *before* the batched computation. Somehow, we need the input to the batched computation to come pre-partitioned into the different plaintext slots.

What we need are Pack and Unpack functions that allow the batching procedure to interface with "normal" homomorphic operations. One may think of the Pack and Unpack functions as an on-ramp to and an exit-ramp from the "fast lane" of batching. Let us say that normal homomorphic operations will always use the plaintext slot $R_{\mathfrak{p}_1}$. Roughly, the Pack function should take a bunch of ciphertexts $\mathbf{c}_1, \ldots, \mathbf{c}_d$ that encrypt messages $m_1, \ldots, m_d \in \mathbb{Z}_p$ under key $\mathbf{s}_1$ for modulus q and plaintext slot $R_{\mathfrak{p}_1}$, and then *aggregate* them into a single ciphertext $\mathbf{c}$ under some possibly different key $\mathbf{s}_2$ for modulus q, so that correctness holds with respect to all of the different plaintext slots – i.e. $m_i = [[\langle \mathbf{c}, \mathbf{s}_2 \rangle]_q]_{\mathfrak{p}_i}$ for all i. The Pack function thus allows normal homomorphic operations to feed into the batch operation. The Unpack function should accept the output of a batched computation, namely a ciphertext $\mathbf{c}'$ such that $m_i = [[\langle \mathbf{c}', \mathbf{s}'_1 \rangle]_q]_{\mathfrak{p}_i}$ for all i, and then de-aggregate this ciphertext by outputting ciphertexts $\mathbf{c}'_1, \ldots, \mathbf{c}'_d$ under some possibly different common secret key $\mathbf{s}'_2$ such that $m_i = [[\langle \mathbf{c}'_i, \mathbf{s}'_2 \rangle]_q]_{\mathfrak{p}_1}$ for all i. Now that all of the ciphertexts are under a common key and plaintext slot, normal homomorphic operations can resume. With such Pack and Unpack functions, we could indeed batch the bootstrapping operation. For circuits of large width (say, at least d) we could reduce the per-gate bootstrapping computation by a factor of d, making it only quasi-linear in λ. Assuming the Pack and Unpack functions have complexity at most quasi-quadratic in d (per-gate this is only quasi-linear, since Pack and Unpack operate on d gates), the overall per-gate computation of a batched-bootstrapped scheme becomes only quasi-linear.

Here, we describe suitable Pack and Unpack functions. These functions will make heavy use of the automorphisms $\sigma_{i \to j}$ over R that map elements of $\mathfrak{p}_i$ to elements of $\mathfrak{p}_j$. (See Section 6.1.1.) We note that Smart and Vercauteren [SV11] used these automorphisms to construct something similar to our Pack function (though for unpacking they resorted to bootstrapping). We also note that Lyubashevsky, Peikert and Regev [LPR10] used these automorphisms to permute the ideal factors $\mathfrak{q}_i$ of the modulus q, which was an essential tool toward their proof of the pseudorandomness of RLWE.

Toward Pack and Unpack procedures, we begin with the observation that if m is encoded as a number in $\{0, \ldots, p-1\}$ and if $m = [[\langle \mathbf{c}, \mathbf{s} \rangle]_q]_{\mathfrak{p}_i}$, then $m = [[\langle \sigma_{i \to j}(\mathbf{c}), \sigma_{i \to j}(\mathbf{s}) \rangle]_q]_{\mathfrak{p}_j}$. That is, we can switch the plaintext slot but leave the decrypted message unchanged by applying the same automorphism to the ciphertext and the secret key. (These facts follow from the fact that $\sigma_{i \to j}$ is a homomorphism, that it maps elements of $\mathfrak{p}_i$ to elements of $\mathfrak{p}_j$, and that it fixes integers.) Of course, then we have a problem: the ciphertext is now under a different key, whereas we may want the ciphertext to be under the same key as other ciphertexts. To get the ciphertexts to be back under the same key, we simply use the SwitchKey algorithm to switch all of the ciphertexts to a new common key.

Some technical remarks before we describe Pack / Unpack more formally: We mention again that E.PublicKeyGen is modified in the obvious way so that $\mathbf{A} \cdot \mathbf{s} = p \cdot \mathbf{e}$ rather than $2 \cdot \mathbf{e}$, and that this modification induces a similar modification in SwitchKeyGen. Also, let $u \in R$ be a short element such that $u \in 1 + \mathfrak{p}_1$ and $u \in \mathfrak{p}_j$ for all $j \neq 1$. It is obvious that such a u with coefficients in $(-p/2, p/2]$ can be computed efficiently by first picking *any* element u' such that $u' \in 1 + \mathfrak{p}_1$ and

$u' \in \mathfrak{p}_j$ for all $j \neq 1$, and then reducing the coefficients of u' modulo p.

<u>$\mathsf{PackSetup}(\mathbf{s}_1, \mathbf{s}_2)$</u>: Takes as input two secret keys $\mathbf{s}_1, \mathbf{s}_2$. For all $i \in [1, d]$, it runs $\tau_{\sigma_{1\to i}(\mathbf{s}_1)\to\mathbf{s}_2} \leftarrow \mathsf{SwitchKeyGen}(\sigma_{1\to i}(\mathbf{s}_1), \mathbf{s}_2)$.

<u>$\mathsf{Pack}(\{\mathbf{c}_i\}_{i=1}^d, \{\tau_{\sigma_{1\to i}(\mathbf{s}_1)\to\mathbf{s}_2}\}_{i=1}^d)$</u>: Takes as input ciphertexts $\mathbf{c}_1, \ldots, \mathbf{c}_d$ such that $m_i = [[\langle \mathbf{c}_i, \mathbf{s}_1\rangle]_q]_{\mathfrak{p}_1}$ and $0 = [[\langle \mathbf{c}_i, \mathbf{s}_1\rangle]_q]_{\mathfrak{p}_j}$ for all $j \neq 1$, and also some auxiliary information output by $\mathsf{PackSetup}$. For all i, it does the following:

- Computes $\mathbf{c}_i^* \leftarrow \sigma_{1\to i}(\mathbf{c}_i)$. (Observe: We have $m_i = [[\langle \mathbf{c}_i^*, \sigma_{1\to i}(\mathbf{s}_1)\rangle]_q]_{\mathfrak{p}_i}$ while $0 = [[\langle \mathbf{c}_i^*, \sigma_{1\to i}(\mathbf{s}_1)\rangle]_q]_{\mathfrak{p}_j}$ for all $j \neq i$.)

- Runs $\mathbf{c}_i^\dagger \leftarrow \mathsf{SwitchKey}(\tau_{\sigma_{1\to i}(\mathbf{s}_1)\to\mathbf{s}_2}, \mathbf{c}_i^*)$ (Observe: Assuming the noise does not wrap, we have that $m_i = [[\langle \mathbf{c}_i^\dagger, \mathbf{s}_2\rangle]_q]_{\mathfrak{p}_i}$ and $0 = [[\langle \mathbf{c}_i^\dagger, \mathbf{s}_2\rangle]_q]_{\mathfrak{p}_j}$ for all $j \neq i$.)

Finally, it outputs $\mathbf{c} \leftarrow \sum_{i=1}^d \mathbf{c}_i^\dagger$. (Observe: Assuming the noise does not wrap, we have $m_i = [[\langle \mathbf{c}, \mathbf{s}_2\rangle]_q]_{\mathfrak{p}_i}$ for all i.)

<u>$\mathsf{UnpackSetup}(\mathbf{s}_1, \mathbf{s}_2)$</u>: Takes as input secret keys $\mathbf{s}_1, \mathbf{s}_2$. For all $i \in [1, d]$, it runs

$$\tau_{\sigma_{i\to 1}(\mathbf{s}_1)\to\mathbf{s}_2} \leftarrow \mathsf{SwitchKeyGen}(\sigma_{i\to 1}(\mathbf{s}_1), \mathbf{s}_2) \ .$$

<u>$\mathsf{Unpack}(\mathbf{c}, \{\tau_{\sigma_{i\to 1}(\mathbf{s}_1)\to\mathbf{s}_2}\}_{i=1}^d)$</u>: Takes as input a ciphertext $\mathbf{c}$ such that $m_i = [[\langle \mathbf{c}, \mathbf{s}_1\rangle]_q]_{\mathfrak{p}_i}$ for all i, and also some auxiliary information output by $\mathsf{UnpackSetup}$. For all i, it does the following:

- Computes $\mathbf{c}_i \leftarrow u \cdot \sigma_{i\to 1}(\mathbf{c})$. (Observe: Assuming the noise does not wrap, $m_i = [[\langle \mathbf{c}_i, \sigma_{i\to 1}(\mathbf{s}_1)\rangle]_q]_{\mathfrak{p}_1}$ and $0 = [[\langle \mathbf{c}_i, \sigma_{i\to 1}(\mathbf{s}_1)\rangle]_q]_{\mathfrak{p}_j}$ for all $j \neq 1$.)

- Outputs $\mathbf{c}_i^* \leftarrow \mathsf{SwitchKey}(\tau_{\sigma_{i\to 1}(\mathbf{s}_1)\to\mathbf{s}_2}, \mathbf{c}_i)$. (Observe: Assuming the noise does not wrap, $m_i = [[\langle \mathbf{c}_i^*, \mathbf{s}_2\rangle]_q]_{\mathfrak{p}_1}$ and $0 = [[\langle \mathbf{c}_i^*, \mathbf{s}_2\rangle]_q]_{\mathfrak{p}_j}$ for all $j \neq 1$.)

The properties of the resulting scheme are summarized in the following theorem, which is only stated for the RLWE setting (since batched bootstrapping only works in the ring setting).

Theorem 8. *The scheme* FHE *with parameters* n, d, L *s.t.* $n = 1$, $d = O(\lambda)$ *and* $L = O(\log \lambda)$ *with batched bootstrapping has per-gate computation* $\tilde{O}(\lambda)$ *when evaluating circuits of average width* $\Omega(\lambda)$.

Proof. Consider an evaluation of depth t circuit, and let $w_1, \ldots, w_t$ be the width of each of the levels of the circuit. Then $w = (1/t) \cdot \sum_{i\in[t]} w_i$ be the average width of the circuit and by the theorem statement $w = \Omega(\lambda)$. The total number of gates is (naturally) $t \cdot w$.

The evaluation of level i of the circuit involves a bootstrapping operation of the $2 \cdot w_i$ input wires into the gates, in addition to w_i gate evaluations.

We pack the $2 \cdot w_i$ standalone ciphertexts into $\lceil 2w_i/d \rceil \leq 2w_i/d + 1$ packed ciphertexts, where $d = \Omega(\lambda)$ is the parameter of the ring that also determines the number of ciphertexts that can be packed. The cost of this packing/unpacking is linear in the input length which is $2w_i \cdot \tilde{O}(\lambda)$.

For each packed ciphertext, we perform bootstrapping which involves a homomorphic evaluation of the decryption circuit. The total cost per packed ciphertext is thus $\tilde{O}(\lambda^2)$. The total complexity for all packed ciphertexts is therefore at most $(2w_i/d + 1) \cdot \tilde{O}(\lambda^2)$.

Finally the evaluation of the w_i actual gates has (total) complexity $w_i \cdot \tilde{O}(\lambda)$.

Summing all of the above, we get that the total complexity of evaluating level i of the circuit is

$$2w_i \cdot \tilde{O}(\lambda) + (2w_i/d + 1) \cdot \tilde{O}(\lambda^2) + w_i \cdot \tilde{O}(\lambda) = (w_i + \lambda + w_i\lambda/d) \cdot \tilde{O}(\lambda) \ .$$

Summing over all t levels, we get that the total complexity of evaluating the entire circuit is at most

$$(t \cdot w + t \cdot \lambda + t \cdot w\lambda/d) \cdot \tilde{O}(\lambda) \ ,$$

and the per-gate cost is obtained by dividing by $t \cdot w$, and recalling that $w, d = \Omega(\lambda)$:

$$(1 + \lambda/w + \lambda/d) \cdot \tilde{O}(\lambda) = \tilde{O}(\lambda) \ .$$

The theorem thus follows. □

Acknowledgments. We thank Carlos Aguilar Melchor, Boaz Barak, Shafi Goldwasser, Shai Halevi, Chris Peikert, Nigel Smart, and Jiang Zhang for helpful discussions and insights.

References

[ACPS09] Benny Applebaum, David Cash, Chris Peikert, and Amit Sahai. Fast cryptographic primitives and circular-secure encryption based on hard learning problems. In *CRYPTO*, volume 5677 of *Lecture Notes in Computer Science*, pages 595–618. Springer, 2009.

[BGN05] Dan Boneh, Eu-Jin Goh, and Kobbi Nissim. Evaluating 2-DNF formulas on ciphertexts. In *Proceedings of Theory of Cryptography Conference 2005*, volume 3378 of *LNCS*, pages 325–342, 2005.

[Bra12] Zvika Brakerski. Fully homomorphic encryption without modulus switching from classical gapsvp. In Safavi-Naini and Canetti [SNC12], pages 868–886.

[BV11a] Zvika Brakerski and Vinod Vaikuntanathan. Fully homomorphic encryption from ring-LWE and security for key dependent messages. In *CRYPTO*, volume 6841, page 501, 2011.

[BV11b] Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) LWE. In Ostrovsky [Ost11], pages 97–106. References are to full version: http://eprint.iacr.org/2011/344.

[CMNT] Jean-Sébastien Coron, Avradip Mandal, David Naccache, and Mehdi Tibouchi. Fully-homomorphic encryption over the integers with shorter public-keys. Manuscript, to appear in Crypto 2011.

[DGHV10] Marten van Dijk, Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. Fully homomorphic encryption over the integers. In *Advances in Cryptology - EUROCRYPT'10*, volume 6110 of *Lecture Notes in Computer Science*, pages 24–43. Springer, 2010. Full version available on-line from http://eprint.iacr.org/2009/616.

[Gen09a] Craig Gentry. *A fully homomorphic encryption scheme.* PhD thesis, Stanford University, 2009. crypto.stanford.edu/craig.

[Gen09b] Craig Gentry. Fully homomorphic encryption using ideal lattices. In Michael Mitzenmacher, editor, *STOC*, pages 169–178. ACM, 2009.

[GH11a] Craig Gentry and Shai Halevi. Fully homomorphic encryption without squashing using depth-3 arithmetic circuits. In Ostrovsky [Ost11], pages 107–109.

[GH11b] Craig Gentry and Shai Halevi. Implementing gentry's fully-homomorphic encryption scheme. In *EUROCRYPT*, volume 6632 of *Lecture Notes in Computer Science*, pages 129–148. Springer, 2011.

[GHPS12] Craig Gentry, Shai Halevi, Chris Peikert, and Nigel P. Smart. Ring switching in bgv-style homomorphic encryption. In Ivan Visconti and Roberto De Prisco, editors, *SCN*, volume 7485 of *Lecture Notes in Computer Science*, pages 19–37. Springer, 2012.

[GHS12a] Craig Gentry, Shai Halevi, and Nigel P. Smart. Better bootstrapping in fully homomorphic encryption. In Marc Fischlin, Johannes Buchmann, and Mark Manulis, editors, *Public Key Cryptography*, volume 7293 of *Lecture Notes in Computer Science*, pages 1–16. Springer, 2012.

[GHS12b] Craig Gentry, Shai Halevi, and Nigel P. Smart. Fully homomorphic encryption with polylog overhead. In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT*, volume 7237 of *Lecture Notes in Computer Science*, pages 465–482. Springer, 2012. See also `http://eprint.iacr.org/2011/566`.

[GHS12c] Craig Gentry, Shai Halevi, and Nigel P. Smart. Homomorphic evaluation of the aes circuit. In Safavi-Naini and Canetti [SNC12], pages 850–867.

[IP07] Yuval Ishai and Anat Paskin. Evaluating branching programs on encrypted data. In Salil P. Vadhan, editor, *TCC*, volume 4392 of *Lecture Notes in Computer Science*, pages 575–594. Springer, 2007.

[LNV11] Kristin Lauter, Michael Naehrig, and Vinod Vaikuntanathan. Can homomorphic encryption be practical? Manuscript at http://eprint.iacr.org/2011/405, 2011.

[LPR10] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. In *EUROCRYPT*, volume 6110 of *Lecture Notes in Computer Science*, pages 1–23, 2010.

[MGH10] Carlos Aguilar Melchor, Philippe Gaborit, and Javier Herranz. Additively homomorphic encryption with -operand multiplications. In Tal Rabin, editor, *CRYPTO*, volume 6223 of *Lecture Notes in Computer Science*, pages 138–154. Springer, 2010.

[Mic07] Daniele Micciancio. Generalized compact knapsacks, cyclic lattices, and efficient one-way functions. *Computational Complexity*, 16(4):365–411, December 2007. Preliminary version in FOCS 2002.

[Ost11] Rafail Ostrovsky, editor. *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011*. IEEE, 2011.

[Pei09] Chris Peikert. Public-key cryptosystems from the worst-case shortest vector problem: extended abstract. In *STOC*, pages 333–342. ACM, 2009.

[RAD78] Ron Rivest, Leonard Adleman, and Michael L. Dertouzos. On data banks and privacy homomorphisms. In *Foundations of Secure Computation*, pages 169–180, 1978.

[Reg05] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In Harold N. Gabow and Ronald Fagin, editors, *STOC*, pages 84–93. ACM, 2005.

[Reg10] Oded Regev. The learning with errors problem (invited survey). In *IEEE Conference on Computational Complexity*, pages 191–204. IEEE Computer Society, 2010.

[SNC12] Reihaneh Safavi-Naini and Ran Canetti, editors. *Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings*, volume 7417 of *Lecture Notes in Computer Science*. Springer, 2012.

[SS10] Damien Stehlé and Ron Steinfeld. Faster fully homomorphic encryption. In *ASIACRYPT*, volume 6477 of *Lecture Notes in Computer Science*, pages 377–394. Springer, 2010.

[SV10] Nigel P. Smart and Frederik Vercauteren. Fully homomorphic encryption with relatively small key and ciphertext sizes. In *Public Key Cryptography - PKC'10*, volume 6056 of *Lecture Notes in Computer Science*, pages 420–443. Springer, 2010.

[SV11] Nigel P. Smart and Frederik Vercauteren. Fully homomorphic simd operations. *IACR Cryptology ePrint Archive*, 2011:133, 2011.