

# Fully Homomorphic Encryption without Bootstrapping

Zvika Brakerski  
Weizmann Institute of Science

Craig Gentry\*  
IBM T.J. Watson Research Center

Vinod Vaikuntanathan†  
University of Toronto

## Abstract

We present a radically new 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  $2^\lambda$  security against known attacks. For RLWE, we have:

- A leveled FHE scheme that can evaluate  $L$ -level arithmetic circuits with  $\tilde{O}(\lambda \cdot L^3)$  per-gate computation – i.e., computation *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 uses bootstrapping *as an optimization*, where the per-gate computation (which includes the bootstrapping procedure) is  $\tilde{O}(\lambda^2)$ , *independent of  $L$* . Security is based on the hardness of RLWE for *quasi-polynomial* factors (as opposed to the sub-exponential factors needed in previous schemes).

We obtain similar results for LWE, but with worse performance. We introduce a number of further optimizations to our schemes. As an example, for circuits of large width – e.g., where a constant fraction of levels have width at least  $\lambda$  – we can reduce the per-gate computation of the bootstrapped version to  $\tilde{O}(\lambda)$ , independent of  $L$ , by *batching the bootstrapping operation*. Previous FHE schemes all required  $\tilde{\Omega}(\lambda^{3.5})$  computation per gate.

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 some new techniques recently introduced by Brakerski and Vaikuntanathan (FOCS 2011).

---

\*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.

†This material is based on research sponsored by DARPA under agreement number FA8750-11-2-0225. The same disclaimer as above applies.

# 1 Introduction

**Ancient History.** Fully homomorphic encryption (FHE) [19, 8] allows a 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 [8, 6, 20, 10, 5, 4] followed the same blueprint, namely the one laid out in Gentry’s original construction [8, 7].

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” polynomials homomorphically. Starting with Gentry’s original construction based on ideal lattices [8], there are by now a number of such schemes in the literature [6, 20, 10, 5, 4, 13], all of which are based on lattices (either directly or implicitly). The ciphertexts in all these schemes are “noisy”, with a noise that 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”<sup>1</sup> 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.

As if by a strange law of nature, SWHE schemes tend to be incapable of evaluating their own decryption circuits (plus some) without significant modifications. (We discuss recent exceptions [9, 3] below.) Thus, the final step 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 [8] showed how to do this by adding a “hint” – namely, a large set with a secret sparse subset that sums to the original secret key – to the public key and relying on a “sparse subset sum” assumption.

## 1.1 Efficiency of Fully Homomorphic Encryption

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 to the time it takes to compute it in the clear.<sup>2</sup> Unfortunately, FHE schemes that follow Gentry’s blueprint (some of which have actually been implemented [10, 5]) have fairly poor performance – their per-gate computation overhead is  $p(\lambda)$ , a large polynomial in the security parameter. In fact, we would like to argue that this penalty in performance is 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 in  $2^\lambda$  time. For this to be hard, the lattice must have dimension  $\Omega(D \cdot \lambda)$ .<sup>3</sup> Moreover, the coefficients of the vectors used in the

---

<sup>1</sup>In a “leveled” FHE scheme, the size of the public key is linear in the *depth* of the circuits that the scheme can evaluate. One can obtain a “pure” FHE scheme (with a constant-size public key) from a leveled FHE scheme by assuming “circular security” – namely, that it is safe to encrypt the leveled FHE secret key under its own public key. We will omit the term “leveled” in this work.

<sup>2</sup>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 [9, 3]).

<sup>3</sup>This is because we have lattice algorithms in  $n$  dimensions that compute  $2^{n/\lambda}$ -approximations of short vectors in time  $2^{\tilde{O}(\lambda)}$ .

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 [22], 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  $\tilde{\Omega}(\lambda^3)$ .<sup>4</sup>

**Recent Deviations from Gentry’s Blueprint, and the Hope for Better Efficiency.** Recently, Gentry and Halevi [9], and Brakerski and Vaikuntanathan [3], 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. Brakerski and Vaikuntanathan [3] manage 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  $\tilde{\Omega}(\lambda^4)$  (in fact, that is an optimistic evaluation of their performance). 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 [3] in an essential way to achieve remarkable efficiency gains.

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 polynomial hardness assumptions?

## 1.2 Our Results and Techniques

We leverage Brakerski and Vaikuntanathan’s techniques [3] to achieve asymptotically very efficient FHE schemes. Also, we base security on lattice problems with *quasi-polynomial* approximation factors. (Previous schemes all used sub-exponential 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 [3], 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  $m = \llbracket \langle \mathbf{c}, \mathbf{s} \rangle \rrbracket_q$ , where  $\mathbf{s} \in R^n$  is the secret key vector,  $q$  is an odd modulus, and  $\llbracket \cdot \rrbracket_q$  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  $n = 1$ .

<sup>4</sup>We note that bootstrapping lazily – i.e., applying the refresh procedure only at a  $1/k$  fraction of the circuit levels for  $k > 1$  – 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.,  $k = O(\log \lambda)$ .

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  $B^2$ .<sup>5</sup> 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 [3]. 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.

**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 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,  $|\langle \mathbf{c}', \mathbf{s} \rangle| < |\langle \mathbf{c}, \mathbf{s} \rangle|$ .

**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  $|\langle \mathbf{c}, \mathbf{s} \rangle| < q/2 - (q/p) \cdot \ell_1(\mathbf{s})$ , we have*

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

where  $\ell_1(\mathbf{s})$  is the  $\ell_1$ -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  $p = q \bmod 2$ , 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  $e_p = [\langle \mathbf{c}', \mathbf{s} \rangle]_p$ .  $\square$

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 [3], 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 [3].

**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

<sup>5</sup>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.

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  $k$ , we can evaluate circuits of depth  $k$ . The performance of the scheme degrades with  $k$  – e.g., we need to set  $q = q_0$  to have bit length proportional to  $k$  – but it degrades only polynomially with  $k$ .

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.

Performance-wise, this scheme trounces previous (bootstrapping-based) 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.

**Bootstrapping for Better Efficiency and Better Assumptions.** The per-gate computation of our FHE-without-bootstrapping scheme 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*<sup>6</sup> 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  $\tilde{O}(\lambda)$  bits, and therefore we can base security on the hardness of lattice problems with quasi-polynomial factors. Since the decryption circuit has size  $\tilde{O}(\lambda)$  for the RLWE-based instantiation, the per-gate computation becomes  $\tilde{O}(\lambda^2)$  (independent of  $L$ ). See Section 5 for details.

**Other Optimizations.** We also consider *batching* as an optimization. The idea behind batching is to pack multiple 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.

---

<sup>6</sup>We are aware of the seeming irony of trumpeting “FHE without bootstrapping” and then proposing bootstrapping “as an optimization”. 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, this “optimization” will not improve performance asymptotically.

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  $\lambda$ ), 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 5.

### 1.3 Other Related Work

We note that prior to Gentry’s construction, there were already a few interesting homomorphic encryptions schemes that could be called “somewhat homomorphic”, including Boneh-Goh-Nissim [2] (evaluates quadratic formulas using bilinear maps), (Aguilar Melchor)-Gaborit-Herranz [15] (evaluates constant degree polynomials using lattices) and Ishai-Paskin [12] (evaluates branching programs).

## 2 Preliminaries

**Basic Notation.** In our construction, we will use a ring  $R$ . In our 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  $\gamma_R$  is at most  $\sqrt{d}$  by Cauchy-Schwarz.

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. When it is obvious that  $q$  is not a power of two, we will use  $\lceil \log q \rceil$  to denote  $1 + \lceil \log q \rceil$ . For  $a \in R$ , we use the notation  $[a]_q$  to refer to  $a \bmod q$ , with coefficients reduced into the range  $(-q/2, q/2]$ .

**Leveled Fully Homomorphic Encryption.** 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 1** (Leveled Fully Homomorphic Encryption [7]). *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  $L$  (that use some specified complete set of gates), 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.*

### 2.1 The Learning with Errors (LWE) Problem

The learning with errors (LWE) problem was introduced by Regev [17]. It is defined as follows.

**Definition 2** (LWE). *For security parameter  $\lambda$ , 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  $\text{LWE}_{n,q,\chi}$  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  $b_i = \langle \mathbf{a}_i, \mathbf{s} \rangle + e_i$ . The  $\text{LWE}_{n,q,\chi}$  assumption is that the  $\text{LWE}_{n,q,\chi}$  problem is infeasible.*

Regev [17] proved that for certain moduli  $q$  and Gaussian error distributions  $\chi$ , the  $\text{LWE}_{n,q,\chi}$  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 3** (*B*-bounded distributions). A distribution ensemble  $\{\chi_n\}_{n \in \mathbb{N}}$ , supported over the integers, is called *B*-bounded if

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

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

**Theorem 1** (Regev [17]). For any integer dimension  $n$ , prime integer  $q = q(n)$ , and  $B = B(n) \geq 2n$ , there is an efficiently samplable *B*-bounded distribution  $\chi$  such that if there exists an efficient (possibly quantum) algorithm that solves  $\text{LWE}_{n,q,\chi}$ , then there is an efficient quantum algorithm for solving  $\tilde{O}(qn^{1.5}/B)$ -approximate worst-case SVP and gapSVP.

Peikert [16] de-quantized Regev’s results to some extent – that is, he showed the  $\text{LWE}_{n,q,\chi}$  assumption is true as long as certain worst-case lattice problems are hard to solve using a *classical* algorithm. (See [16] for a precise statement of these results.)

Applebaum et al. [1] showed that if LWE is hard for the above distribution of  $s$ , then it is also hard when  $s$ ’s coefficients are sampled according to the noise distribution  $\chi$ .

## 2.2 The Ring Learning with Errors (RLWE) Problem

The ring learning with errors (RLWE) problem was introduced by Lyubashevsky, Peikert and Regev [14]. We will use an simplified special-case version of the problem that is easier to work with [18, 4].

**Definition 4** (RLWE). For security parameter  $\lambda$ , 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  $\text{RLWE}_{d,q,\chi}$  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  $a_i \leftarrow R_q$  uniformly,  $e_i \leftarrow \chi$ , and setting  $b_i = a_i \cdot s + e_i$ . The  $\text{RLWE}_{d,q,\chi}$  assumption is that the  $\text{RLWE}_{d,q,\chi}$  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 2** (Lyubashevsky-Peikert-Regev [14]). For any  $d$  that is a power of 2, ring  $R = \mathbb{Z}[x]/(x^d + 1)$ , prime integer  $q = q(d) = 1 \pmod d$ , and  $B = \omega(\sqrt{d} \log d)$ , there is an efficiently samplable distribution  $\chi$  that outputs elements of  $R$  of length at most  $B$  with overwhelming probability, such that if there exists an efficient algorithm that solves  $\text{RLWE}_{d,q,\chi}$ , 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  $\chi$  according to a Gaussian distribution, where vectors sampled according to this distribution have length only  $\text{poly}(d)$  with overwhelming probability. This Gaussian distribution may need to be “ellipsoidal” for certain reductions to go through [14]. It has been shown for RLWE that one can equivalently assume that  $s$  is alternatively sampled from the noise distribution  $\chi$  [14].

## 2.3 The General Learning with Errors (GLWE) Problem

The learning with errors (LWE) problem and the ring learning with errors problem RLWE 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 5 (GLWE).** For security parameter  $\lambda$ , 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  $\text{GLWE}_{n,f,q,\chi}$  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  $\text{GLWE}_{n,f,q,\chi}$  assumption is that the  $\text{GLWE}_{n,f,q,\chi}$  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  $\chi$ . 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.

The GLWE assumption implies that 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 describe variants of the “relinearization” and “dimension reduction” techniques of [3]. Finally, in Section 3.4, we lay out our construction of FHE without bootstrapping.

#### 3.1 Basic Encryption Scheme

We begin by presenting a basic GLWE-based encryption scheme with no homomorphic operations. Let  $\lambda$  be the security parameter, representing  $2^\lambda$  security against known attacks. ( $\lambda = 100$  is a reasonable value.)

Let  $R = R(\lambda)$  be a ring. For example, one may use  $R = \mathbb{Z}$  if one wants a scheme based on (standard) LWE, or one may use  $R = \mathbb{Z}[x]/f(x)$  where (e.g.)  $f(x) = x^d + 1$  and  $d = d(\lambda)$  is a power of 2 if one wants a scheme based on RLWE. Let the “dimension”  $n = n(\lambda)$ , an odd modulus  $q = q(\lambda)$ , a “noise” distribution  $\chi = \chi(\lambda)$  over  $R$ , and an integer  $N = N(\lambda)$  be additional parameters of the system. These parameters come from the GLWE assumption, except for  $N$ , which is set to be larger than  $(2n + 1) \log q$ . Note that  $n = 1$  in the RLWE instantiation. For simplicity, assume for now that the plaintext space is  $R_2 = R/2R$ , though larger plaintext spaces are certainly possible.

We go ahead and stipulate here – even though it only becomes important when we introduce homomorphic operations – that the noise distribution  $\chi$  is set to be as small as possible. Specifically, to base security on LWE or GLWE, one must use (typically Gaussian) noise distributions with deviation *at least* some sub-linear function of  $d$  or  $n$ , and we will let  $\chi$  be a noise distribution that barely satisfies that requirement. To

achieve  $2^\lambda$  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  $\chi$  (and hence  $B$ ) depends sub-linearly on  $n$  or  $d$ , the distribution  $\chi$  (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  $q$ .

Here is a basic GLWE-based encryption scheme with no homomorphic operations:

### Basic GLWE-Based Encryption Scheme:

- **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  $\mu$ -bit modulus  $q$  and choose the other parameters ( $d = d(\lambda, \mu, b)$ ,  $n = n(\lambda, \mu, b)$ ,  $N = \lceil (2n + 1) \log q \rceil$ ,  $\chi = \chi(\lambda, \mu, b)$ ) appropriately to ensure that the scheme is based on a GLWE instance that achieves  $2^\lambda$  security against known attacks. Let  $R = \mathbb{Z}[x]/(x^d + 1)$  and let  $params = (q, d, n, N, \chi)$ .
- **E.SecretKeyGen**( $params$ ): Draw  $s' \leftarrow \chi^n$ . Set  $sk = s \leftarrow (1, s'[1], \dots, s'[n]) \in R_q^{n+1}$ .
- **E.PublicKeyGen**( $params, sk$ ): Takes as input a secret key  $sk = s = (1, s')$  with  $s[0] = 1$  and  $s' \in R_q^n$  and the  $params$ . Generate matrix  $A' \leftarrow R_q^{N \times n}$  uniformly and a vector  $e \leftarrow \chi^N$  and set  $b \leftarrow A's' + 2e$ . Set  $A$  to be the  $(n + 1)$ -column matrix consisting of  $b$  followed by the  $n$  columns of  $-A'$ . (Observe:  $A \cdot s = 2e$ .) Set the public key  $pk = A$ .
- **E.Enc**( $params, pk, m$ ): To encrypt a message  $m \in R_2$ , set  $\mathbf{m} \leftarrow (m, 0, \dots, 0) \in R_q^{n+1}$ , sample  $\mathbf{r} \leftarrow R_2^N$  and output the ciphertext  $\mathbf{c} \leftarrow \mathbf{m} + A^T \mathbf{r} \in R_q^{n+1}$ .
- **E.Dec**( $params, sk, \mathbf{c}$ ): Output  $m \leftarrow \llbracket \langle \mathbf{c}, \mathbf{s} \rangle \rrbracket_q$ .

Correctness is easy to see, and 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).

### 3.2 Key Switching (Dimension Reduction)

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  $m = \llbracket [L_{\mathbf{c}}(\mathbf{s})]_q \rrbracket_2$  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 [3] is to consider the *quadratic* equation  $Q_{\mathbf{c}_1, \mathbf{c}_2}(\mathbf{x}) \leftarrow L_{\mathbf{c}_1}(\mathbf{x}) \cdot L_{\mathbf{c}_2}(\mathbf{x})$ . Assuming the noises of the initial ciphertexts are small enough, we obtain  $m_1 \cdot m_2 = \llbracket [Q_{\mathbf{c}_1, \mathbf{c}_2}(\mathbf{s})]_q \rrbracket_2$ , as desired. If one wishes, one can view  $Q_{\mathbf{c}_1, \mathbf{c}_2}(\mathbf{x})$  as a *linear* equation  $L_{\mathbf{c}_1, \mathbf{c}_2}^{long}(\mathbf{x} \otimes \mathbf{x})$  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.

We observe that Brakerski and Vaikuntanathan’s relinearization / dimension reduction procedures are actually quite a bit more general. They 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  $\mathbf{c}_1, \mathbf{s}_1$ .

Below, we review the concrete details of Brakerski and Vaikuntanathan's key switching procedures. 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  $\langle \mathbf{c}', \mathbf{s}' \rangle = \langle \mathbf{c}, \mathbf{s} \rangle \pmod q$ . We describe these subroutines first.

- $\text{BitDecomp}(\mathbf{x} \in R_q^n, q)$  decomposes  $\mathbf{x}$  into its bit representation. Namely, write  $\mathbf{x} = \sum_{j=0}^{\lceil \log q \rceil} 2^j \cdot \mathbf{u}_j$ , where all of the vectors  $\mathbf{u}_j$  are in  $R_2^n$ , and output  $(\mathbf{u}_0, \mathbf{u}_1, \dots, \mathbf{u}_{\lceil \log q \rceil}) \in R_2^{n \cdot \lceil \log q \rceil}$ .
- $\text{Powersof2}(\mathbf{x} \in R_q^n, q)$  outputs the vector  $(\mathbf{x}, 2 \cdot \mathbf{x}, \dots, 2^{\lceil \log q \rceil} \cdot \mathbf{x}) \in R_q^{n \cdot \lceil \log q \rceil}$ .

If one knows *a priori* that  $\mathbf{x}$  has coefficients in  $[0, B]$  for  $B \ll q$ , then  $\text{BitDecomp}$  can be optimized in the obvious way to output a shorter decomposition in  $R_2^{n \cdot \lceil \log B \rceil}$ . Observe that:

**Lemma 2.** *For vectors  $\mathbf{c}, \mathbf{s}$  of equal length, we have  $\langle \text{BitDecomp}(\mathbf{c}, q), \text{Powersof2}(\mathbf{s}, q) \rangle = \langle \mathbf{c}, \mathbf{s} \rangle \pmod q$ .*

*Proof.*

$$\langle \text{BitDecomp}(\mathbf{c}, q), \text{Powersof2}(\mathbf{s}, q) \rangle = \sum_{j=0}^{\lceil \log q \rceil} \langle \mathbf{u}_j, 2^j \cdot \mathbf{s} \rangle = \sum_{j=0}^{\lceil \log q \rceil} \langle 2^j \cdot \mathbf{u}_j, \mathbf{s} \rangle = \left\langle \sum_{j=0}^{\lceil \log q \rceil} 2^j \cdot \mathbf{u}_j, \mathbf{s} \right\rangle = \langle \mathbf{c}, \mathbf{s} \rangle .$$

□

We remark that this obviously generalizes to decompositions wrt bases other than the powers of 2.

Now, key switching consists of two procedures: first, a procedure  $\text{SwitchKeyGen}(\mathbf{s}_1, \mathbf{s}_2, n_1, n_2, q)$ , which takes as input the two secret key vectors as input, the respective dimensions of these vectors, and the modulus  $q$ , and outputs some auxiliary information  $\tau_{\mathbf{s}_1 \rightarrow \mathbf{s}_2}$  that enables the switching; and second, a procedure  $\text{SwitchKey}(\tau_{\mathbf{s}_1 \rightarrow \mathbf{s}_2}, \mathbf{c}_1, n_1, n_2, q)$ , 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  $n_1, n_2, q$ .)

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

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

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

Note that, in  $\text{SwitchKeyGen}$ , 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.

**Lemma 3. [Correctness]** *Let  $\mathbf{s}_1, \mathbf{s}_2, q, n_1, n_2, \mathbf{A}, \mathbf{B} = \tau_{\mathbf{s}_1 \rightarrow \mathbf{s}_2}$  be as in  $\text{SwitchKeyGen}(\mathbf{s}_1, \mathbf{s}_2)$ , 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  $\mathbf{c}_2 \leftarrow \text{SwitchKey}(\tau_{\mathbf{s}_1 \rightarrow \mathbf{s}_2}, \mathbf{c}_1)$ . Then,*

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

*Proof.*

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

□

Note that the dot product of  $\text{BitDecomp}(\mathbf{c}_1)$  and  $\mathbf{e}_2$  is small, since  $\text{BitDecomp}(\mathbf{c}_1)$  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.

### 3.3 Modulus Switching

Suppose  $\mathbf{c}$  is a valid encryption of  $m$  under  $\mathbf{s}$  modulo  $q$  (i.e.,  $m = \llbracket \langle \mathbf{c}, \mathbf{s} \rangle \rrbracket_q$ ), 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,  $m = \llbracket \langle \mathbf{c}', \mathbf{s} \rangle \rrbracket_p$ ! 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 4 below.

**Definition 6** (Scale). *For integer vector  $\mathbf{x}$  and integers  $q > p > m$ , we define  $\mathbf{x}' \leftarrow \text{Scale}(\mathbf{x}, q, p, r)$  to be the  $R$ -vector closest to  $(p/q) \cdot \mathbf{x}$  that satisfies  $\mathbf{x}' = \mathbf{x} \bmod r$ .*

**Definition 7** ( $\ell_1^{(R)}$  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 4.** *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  $\mathbf{c}' \leftarrow \text{Scale}(\mathbf{c}, q, p, r)$ . Then, for any  $\mathbf{s} \in R^n$  with  $\|\llbracket \langle \mathbf{c}, \mathbf{s} \rangle \rrbracket_q\| < q/2 - (q/p) \cdot (r/2) \cdot \sqrt{d} \cdot \gamma(R) \cdot \ell_1^{(R)}(\mathbf{s})$ , we have*

$$\llbracket \langle \mathbf{c}', \mathbf{s} \rangle \rrbracket_p = \llbracket \langle \mathbf{c}, \mathbf{s} \rangle \rrbracket_q \bmod r \quad \text{and} \quad \|\llbracket \langle \mathbf{c}', \mathbf{s} \rangle \rrbracket_p\| < (p/q) \cdot \|\llbracket \langle \mathbf{c}, \mathbf{s} \rangle \rrbracket_q\| + (r/2) \cdot \sqrt{d} \cdot \gamma(R) \cdot \ell_1^{(R)}(\mathbf{s})$$

*Proof.* (Lemma 4) We have

$$\llbracket \langle \mathbf{c}, \mathbf{s} \rangle \rrbracket_q = \langle \mathbf{c}, \mathbf{s} \rangle - kq$$

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

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

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

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

Furthermore, modulo  $r$ , we have  $\llbracket \langle \mathbf{c}', \mathbf{s} \rangle \rrbracket_p = e_p = \langle \mathbf{c}', \mathbf{s} \rangle - kp = \langle \mathbf{c}, \mathbf{s} \rangle - kq = \llbracket \langle \mathbf{c}, \mathbf{s} \rangle \rrbracket_q$ . □

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 4.

**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  $\mathbf{c}' \leftarrow \text{Scale}(\mathbf{c}, q, p, r)$  is an encryption 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 FHE scheme. 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  $L^{\text{long}}$ , used below, is defined in Section 3.2.)

Our FHE Scheme without Bootstrapping:

- $\text{FHE.Setup}(1^\lambda, 1^L, b)$ : 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  $\text{params}_j \leftarrow \text{E.Setup}(1^\lambda, 1^{(j+1) \cdot \mu}, b)$  to obtain a ladder of decreasing moduli from  $q_L ((L + 1) \cdot \mu$  bits) down to  $q_0$  ( $\mu$  bits). For  $j = L - 1$  to 0, replace the value of  $d_j$  in  $\text{params}_j$  with  $d = d_L$  and the distribution  $\chi_j$  with  $\chi = \chi_L$ . (That is, the ring dimension and noise distribution do not depend on the circuit level, but the vector dimension  $n_j$  still might.)
- $\text{FHE.KeyGen}(\{\text{params}_j\})$ : For  $j = L$  down to 0, do the following:
  1. Run  $\mathbf{s}_j \leftarrow \text{E.SecretKeyGen}(\text{params}_j)$  and  $\mathbf{A}_j \leftarrow \text{E.PublicKeyGen}(\text{params}_j, \mathbf{s}_j)$ .
  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. Set  $\mathbf{s}''_j \leftarrow \text{BitDecomp}(\mathbf{s}'_j, q_j)$ .
  4. Run  $\tau_{\mathbf{s}''_{j+1} \rightarrow \mathbf{s}_j} \leftarrow \text{SwitchKeyGen}(\mathbf{s}''_j, \mathbf{s}_{j-1})$ . (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  $\tau_{\mathbf{s}''_{j+1} \rightarrow \mathbf{s}_j}$ ’s.

- $\text{FHE.Enc}(\text{params}, pk, m)$ : Take a message in  $R_2$ . Run  $\text{E.Enc}(\mathbf{A}_L, m)$ .
- $\text{FHE.Dec}(\text{params}, sk, \mathbf{c})$ : Suppose the ciphertext is under key  $\mathbf{s}_j$ . Run  $\text{E.Dec}(\mathbf{s}_j, \mathbf{c})$ . (The ciphertext could be augmented with an index indicating which level it belongs to.)

- $\text{FHE.Add}(pk, \mathbf{c}_1, \mathbf{c}_2)$ : Takes two ciphertexts encrypted under the same  $\mathbf{s}_j$ . (If they are not initially, use  $\text{FHE.Refresh}$  (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:

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

- $\text{FHE.Mult}(pk, \mathbf{c}_1, \mathbf{c}_2)$ : Takes two ciphertexts encrypted under the same  $\mathbf{s}_j$ . If they are not initially, use  $\text{FHE.Refresh}$  (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  $L_{\mathbf{c}_1, \mathbf{c}_2}^{long}(\mathbf{x} \otimes \mathbf{x})$ . Then, output:

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

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

1. Expand: Set  $\mathbf{c}_1 \leftarrow \text{Powersof2}(\mathbf{c}, q_j)$ . (Observe:  $\langle \mathbf{c}_1, \mathbf{s}''_j \rangle = \langle \mathbf{c}, \mathbf{s}'_j \rangle \bmod q_j$  by Lemma 2.)
2. Switch Moduli: Set  $\mathbf{c}_2 \leftarrow \text{Scale}(\mathbf{c}_1, q_j, q_{j-1}, 2)$ , a ciphertext under the key  $\mathbf{s}''_j$  for modulus  $q_{j-1}$ .
3. Switch Keys: Output  $\mathbf{c}_3 \leftarrow \text{SwitchKey}(\tau_{\mathbf{s}''_j \rightarrow \mathbf{s}_{j-1}}, \mathbf{c}_2, q_{j-1})$ , 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.

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  $\tau_{\mathbf{s}''_j \rightarrow \mathbf{s}_{j-1}}$  can be viewed as an encrypted secret key, and the SwitchKey step can be 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 Correctness, Setting the Parameters, 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  $\text{FHE.Add}$  and  $\text{FHE.Mult}$  – namely, the addition or multiplication itself, and then the Powersof2, Scale and SwitchKey steps that make up  $\text{FHE.Refresh}$  – 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  $\text{FHE.Enc}$  introduces initially.

## 4.1 The Initial Noise from FHE.Enc

Recall that FHE.Enc simply invokes E.Enc for suitable parameters ( $params_L$ ) that depend on  $\lambda$  and  $L$ . In turn, the noise of ciphertexts output by E.Enc depends on the noise of the initial “ciphertexts” (the encryptions of 0) implicit in the matrix  $A$  output by E.PublicKeyGen, whose noise distribution is dictated by the distribution  $\chi$ .

**Lemma 5.** *Let  $n_L$  and  $q_L$  be the parameters associated to FHE.Enc. Let  $d$  be the dimension of the ring  $R$ , and let  $\gamma_R$  be the expansion factor associated to  $R$ . (Both of these quantities are 1 when  $R = \mathbb{Z}$ .) Let  $B_\chi$  be a bound such that  $R$ -elements sampled from the noise distribution  $\chi$  have length at most  $B_\chi$  with overwhelming probability. The length of the noise in ciphertexts output by FHE.Enc is at most  $1 + 2 \cdot \gamma_R \cdot \sqrt{d} \cdot ((2n_L + 1) \log q_L) \cdot B_\chi$ .*

*Proof.* Recall that  $\mathbf{s} \leftarrow \text{E.SecretKeyGen}$  and  $\mathbf{A} \leftarrow \text{E.PublicKeyGen}(\mathbf{s}, N)$  for  $N = (2n_L + 1) \log q_L$ , where  $\mathbf{A} \cdot \mathbf{s} = 2\mathbf{e}$  for  $\mathbf{e} \leftarrow \chi$ . 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  $[\langle \mathbf{c}, \mathbf{s} \rangle]_q = [m + 2\langle \mathbf{r}, \mathbf{e} \rangle]_q$ , whose magnitude is at most  $1 + 2 \cdot \gamma_R \cdot \sum_{j=1}^N \|\mathbf{r}[j]\| \cdot \|\mathbf{e}[j]\| \leq 1 + 2 \cdot \gamma_R \cdot \sqrt{d} \cdot N \cdot B_\chi$ .  $\square$

Notice that we are using very loose (i.e., conservative) upper bounds for the noise. These bounds could be tightened up with a more careful analysis. 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.

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

Consider FHE.Mult. One begins 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  $L_{\mathbf{c}_1, \mathbf{c}_2}^{long}(\mathbf{x} \otimes \mathbf{x}) = Q_{\mathbf{c}_1, \mathbf{c}_2}(\mathbf{x})$  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,  $[\langle \mathbf{c}_3, \mathbf{s}_j \otimes \mathbf{s}_j \rangle]_{q_j} = [L_{\mathbf{c}_1, \mathbf{c}_2}^{long}(\mathbf{s}_j \otimes \mathbf{s}_j)]_{q_j} = [e_1 \cdot e_2]_{q_j}$ . 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  $\gamma_R$  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 FHE.Refresh) is formally captured in the following lemmas.

**Lemma 6.** *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 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 the linear equation  $L_{\mathbf{c}_1, \mathbf{c}_2}^{long}(\mathbf{x} \otimes \mathbf{x})$  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  $\tilde{O}(dn_j^2 \log q_j)$ . For the RLWE instantiation, since  $n_j = 1$  and since (as we will see)  $\log q_j$  depends logarithmically on the security parameter and linearly on  $L$ , the computation here is only quasi-linear in the security parameter. For the LWE instantiation, the computation is quasi-quadratic.

### 4.3 Correctness and Performance of FHE.Refresh

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

Correctness and Performance of the Expand Step. The Expand step of FHE.Refresh takes as input a long ciphertext  $\mathbf{c}$  under the long tensored key  $\mathbf{s}'_j = \mathbf{s}_j \otimes \mathbf{s}_j$  for modulus  $q_j$ . It simply applies the Powersof2 transformation to  $\mathbf{c}$  to obtain  $\mathbf{c}_1$ . By Lemma 2, we know that

$$\langle \text{Powersof2}(\mathbf{c}, q_j), \text{BitDecomp}(\mathbf{s}'_j, q_j) \rangle = \langle \mathbf{c}, \mathbf{s}'_j \rangle \bmod q_j$$

i.e., we know that if  $\mathbf{s}'_j$  decrypts  $\mathbf{c}$  correctly, then  $\mathbf{s}''_j$  decrypts  $\mathbf{c}_1$  correctly. The noise has not been affected at all.

If implemented naively, the computation in the Expand step is  $\tilde{O}(dn_j^2 \log^2 q_j)$ . The somewhat high computation is due to the fact that the expanded ciphertext is a  $(\binom{n_j+1}{2} \cdot \lceil \log q_j \rceil)$ -dimensional vector over  $R_q$ .

However, recall that  $\mathbf{s}_j$  is drawn using the distribution  $\chi$  – i.e., it has small coefficients of size basically independent of  $q_j$ . Consequently,  $\mathbf{s}'_j$  also has small coefficients, and we can use this *a priori* knowledge in combination with an optimized version of BitDecomp to output a shorter bit decomposition of  $\mathbf{s}'_j$  – in particular, a  $(\binom{n_j+1}{2} \cdot \lceil \log q'_j \rceil)$ -dimensional vector over  $R_q$  where  $q'_j \ll q_j$  is a bound (with overwhelming probability) on the coefficients of elements output by  $\chi$ . Similarly, we can use an abbreviated version of Powersof2( $\mathbf{c}, q_j$ ). In this case, the computation is  $\tilde{O}(dn_j^2 \log q_j)$ .

Correctness and Performance of the Switch-Moduli Step. The Switch Moduli step takes as input a ciphertext  $\mathbf{c}_1$  under the secret bit-vector  $\mathbf{s}''_j$  for the modulus  $q_j$ , and outputs the ciphertext  $\mathbf{c}_2 \leftarrow \text{Scale}(\mathbf{c}_1, q_j, q_{j-1}, 2)$ , which we claim to be a ciphertext under key  $\mathbf{s}''_j$  for modulus  $q_{j-1}$ . Note that  $\mathbf{s}''_j$  is a *short* secret key, since it is a bit vector in  $R_2^{t_j}$  for  $t_j \leq \binom{n_j+1}{2} \cdot \lceil \log q_j \rceil$ . By Corollary 1, and using the fact that  $\ell_1(\mathbf{s}''_j) \leq \sqrt{d} \cdot t_j$ , the following is true: if the noise of  $\mathbf{c}_1$  has length at most  $B < q_j/2 - (q_j/q_{j-1}) \cdot d \cdot \gamma_R \cdot t_j$ , then correctness is preserved and the noise of  $\mathbf{c}_2$  is bounded by  $(q_{j-1}/q_j) \cdot B + d \cdot \gamma_R \cdot t_j$ . 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 8.** *Let  $\mathbf{c}_1$  be a ciphertext under the key  $\mathbf{s}''_j = \text{BitDecomp}(\mathbf{s}_j \otimes \mathbf{s}_j, q_j)$  such that  $e_j \leftarrow [\langle \mathbf{c}_1, \mathbf{s}''_j \rangle]_{q_j}$  has length at most  $B$  and  $m = \lceil e_j \rceil_2$ . Let  $\mathbf{c}_2 \leftarrow \text{Scale}(\mathbf{c}_1, q_j, q_{j-1}, 2)$ , and let  $e_{j-1} = [\langle \mathbf{c}_2, \mathbf{s}''_j \rangle]_{q_{j-1}}$ . Then,  $e_{j-1}$  (the new noise) has length at most  $(q_{j-1}/q_j) \cdot B + d \cdot \gamma_R \cdot \binom{n_j+1}{2} \cdot \lceil \log q_j \rceil$ , and (assuming this noise length is less than  $q_{j-1}/2$ ) we have  $m = \lceil e_{j-1} \rceil_2$ .*

The computation in the Switch-Moduli step is  $\tilde{O}(dn_j^2 \log q_j)$ , using the optimized versions of BitDecomp and Powersof2 mentioned above.

Correctness and Performance of the Switch-Key Step. Finally, in the Switch Keys step, we take as input a ciphertext  $\mathbf{c}_2$  under key  $\mathbf{s}''_j$  for modulus  $q_{j-1}$  and set  $\mathbf{c}_3 \leftarrow \text{SwitchKey}(\tau_{\mathbf{s}''_j \rightarrow \mathbf{s}_{j-1}}, \mathbf{c}_2, q_{j-1})$ , a ciphertext under the key  $\mathbf{s}_{j-1}$  for modulus  $q_{j-1}$ . In Lemma 3, we proved the correctness of key switching and established that the noise grows only by the additive factor 2 ( $\text{BitDecomp}(\mathbf{c}_2, q_{j-1}), \mathbf{e}$ ), where  $\text{BitDecomp}(\mathbf{c}_2, q_{j-1})$  is

a (short) bit-vector and  $\mathbf{e}$  is a (short and fresh) noise vector. In particular, if the noise originally had length  $B$ , then after the Switch Keys step is has length at most  $B + 2 \cdot \gamma_R \cdot \sum_{i=1}^{u_j} \|\text{BitDecomp}(\mathbf{c}_2, q_{j-1})[i]\| \cdot B_\chi \leq B + 2 \cdot \gamma_R \cdot u_j \cdot \sqrt{d} \cdot B_\chi$ , where  $u_j \leq \binom{n_j+1}{2} \cdot \lceil \log q_j \rceil \cdot \lceil \log q_{j-1} \rceil$  is the dimension of  $\text{BitDecomp}(\mathbf{c}_2)$ .

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

**Lemma 9.** *Let  $\mathbf{c}_2$  be a ciphertext under the key  $\mathbf{s}_j'' = \text{BitDecomp}(\mathbf{s}_j \otimes \mathbf{s}_j, q_j)$  for modulus  $q_{j-1}$  such that  $e_1 \leftarrow [\langle \mathbf{c}_2, \mathbf{s}_j'' \rangle]_{q_{j-1}}$  has length at most  $B$  and  $m = [e_1]_2$ . Let  $\mathbf{c}_3 \leftarrow \text{SwitchKey}(\tau_{\mathbf{s}_j' \rightarrow \mathbf{s}_{j-1}}, \mathbf{c}_2, q_{j-1})$ , and let  $e_2 = [\langle \mathbf{c}_3, \mathbf{s}_{j-1} \rangle]_{q_{j-1}}$ . Then,  $e_2$  (the new noise) has length at most  $B + 2 \cdot \gamma_R \cdot \binom{n_j+1}{2} \cdot \lceil \log q_j \rceil^2 \cdot \sqrt{d} \cdot B_\chi$  and (assuming this noise length is less than  $q_{j-1}/2$ ) we have  $m = [e_2]_2$ .*

In terms of computation, the Switch-Key step involves multiplying the transpose of  $u_j$ -dimensional vector  $\text{BitDecomp}(\mathbf{c}_2)$  with a  $u_j \times (n_{j-1} + 1)$  matrix  $B$ . Assuming  $n_j \geq n_{j-1}$  and  $q_j \geq q_{j-1}$ , and using the optimized versions of  $\text{BitDecomp}$  and  $\text{Powersof2}$  mentioned above to reduce  $u_j$ , this computation is  $\tilde{O}(dn_j^3 \log^2 q_j)$ . Still this is quasi-linear in the RLWE instantiation.

#### 4.4 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  $\lambda$  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  $\text{FHE.Enc}$ . (Recall the discussion from Section 3.1 where we explained that we use a noise distribution  $\chi$  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  $\mathbf{c}' \leftarrow \text{FHE.Mult}(pk, \mathbf{c}_1, \mathbf{c}_2)$  at level  $j - 1$ . ( $\text{FHE.Mult}$  increases the noise strictly more in the worst-case than  $\text{FHE.Add}$  for any reasonable choice of parameters.)

Specifically, after the first step of  $\text{FHE.Mult}$  (without the Refresh step), the noise has length at most  $\gamma_R \cdot B^2$ . Then, we apply the Scale function, after which the noise length is at most  $(q_{j-1}/q_j) \cdot \gamma_R \cdot B^2 + \eta_{\text{Scale},j}$ , where  $\eta_{\text{Scale},j}$  is some additive term. Finally, we apply the SwitchKey function, which introduces another additive term  $\eta_{\text{SwitchKey},j}$ . Overall, after the entire  $\text{FHE.Mult}$  step, the noise length is at most  $(q_{j-1}/q_j) \cdot \gamma_R \cdot B^2 + \eta_{\text{Scale},j} + \eta_{\text{SwitchKey},j}$ . 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:  $B \geq 2 \cdot (\eta_{\text{Scale},j} + \eta_{\text{SwitchKey},j})$  for all  $j$ .
- Property 2:  $q_j/q_{j-1} \geq 2 \cdot B \cdot \gamma_R$  for all  $j$ .

Then we have

$$(q_{j-1}/q_j) \cdot \gamma_R \cdot B^2 + \eta_{\text{Scale},j} + \eta_{\text{SwitchKey},j} \leq \frac{1}{2 \cdot B \cdot \gamma_R} \cdot \gamma_R \cdot B^2 + \frac{1}{2} \cdot B \leq B$$

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  $\eta_{\text{Scale},L} + \eta_{\text{SwitchKey},L}$ , which is polynomial in  $\lambda$  and  $\log q_L$ , and hence polynomial in  $\lambda$  and  $L$  (Property 1 is satisfied). Since  $q_j/q_{j-1}$  is super-polynomial in both  $\lambda$  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  $d \cdot n_L$  must be approximately that number times  $\lambda$  for  $2^\lambda$  security.

**Theorem 3.** *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  $\tilde{O}(d \cdot n_L^3 \cdot \log^2 q_j) = \tilde{O}(d \cdot n_L^3 \cdot L^2)$ . 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.

## 4.5 Security

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

## 5 Optimizations

Despite the fact that our new FHE scheme has per-gate computation only quasi-linear in the security parameter, we present several significant ways of optimizing it. We focus primarily on the RLWE-based scheme, since it is much more efficient.

Our first optimization is *batching*. Batching allows us to reduce the per-gate computation from quasi-linear in the security parameter to *polylogarithmic*. In more detail, we show that evaluating a function  $f$  homomorphically on  $\ell = \Omega(\lambda)$  blocks of encrypted data requires only *polylogarithmically* (in terms of the security parameter  $\lambda$ ) more computation than evaluating  $f$  on the unencrypted data. (The overhead is still polynomial in the depth  $L$  of the circuit computing  $f$ .) Batching works essentially by packing multiple plaintexts into each ciphertext.

Next, we reintroduce *bootstrapping* as an optimization rather than a necessity (Section 5.2). Bootstrapping allows us to achieve per-gate computation *quasi-quadratic* in the security parameter, *independent* of the number levels in the circuit being evaluated.

In Section 5.3, we show that *batching the bootstrapping function* is a powerful combination. With this optimization, circuits whose levels mostly have width at least  $\lambda$  can be evaluated homomorphically with only  $\tilde{O}(\lambda)$  per-gate computation, independent of the number of levels.

Finally, Section 5.5 presents a few other miscellaneous optimizations.

### 5.1 Batching

Suppose we want to evaluate the same function  $f$  on  $\ell$  blocks of encrypted data. (Or, similarly, suppose we want to evaluate the same encrypted function  $f$  on  $\ell$  blocks of plaintext data.) Can we do this using less than

$\ell$  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 5.2), 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 [21] 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  $\lambda$ ) 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 mostly have width at least  $\lambda$ , *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  $R_p$  that is isomorphic to the direct product  $R_{p_1} \times \cdots \times R_{p_d}$  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  $p_i$ ’s will be *ideals* in our ring  $R = \mathbb{Z}[x]/(x^d + 1)$ . (One could still use  $R_2$  as in [21], but the number theory there is a bit more involved.)

### 5.1.1 Some Number Theory

Let us take a very brief tour of algebraic number theory. Suppose  $p$  is a prime number satisfying  $p \equiv 1 \pmod{2d}$ , 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,  $x^d + 1 = \prod_{i=1}^d (x - a_i) \pmod{p}$  where  $a_i = a^{2i-1} \pmod{p}$ . 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  $0 + \mathfrak{p}_i, \dots, (p - 1) + \mathfrak{p}_i$  partition  $R$ . These ideals are relative prime, and so they behave like relative prime integers. In particular, the Chinese Remainder Theorem applies:  $R_p \cong R_{p_1} \times \cdots \times R_{p_d}$ .

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  $\sigma_{i \rightarrow j}$  over  $R$  that maps elements of  $\mathfrak{p}_i$  to elements of  $\mathfrak{p}_j$ . Specifically,  $\sigma_{i \rightarrow j}$  works by mapping an  $R$ -element  $r = r(x) = r_{d-1}x^{d-1} + \cdots + r_1x + r_0$  to  $r(x^{e_{ij}}) = r_{d-1}x^{e_{ij}(d-1) \bmod 2d} + \cdots + r_1x^{e_{ij}} + r_0$  where  $e_{ij}$  is some odd number in  $[1, 2d]$ . Notice that this automorphism just permutes the coefficients of  $r$  and fixes the free coefficient. Notationally, we will use  $\sigma_{i \rightarrow j}(\mathbf{v})$  to refer to the vector that results from applying  $\sigma_{i \rightarrow j}$  coefficient-wise to  $\mathbf{v}$ .

### 5.1.2 How Batching Works

Deploying batching inside our scheme FHE is quite straightforward. First, we pick a prime  $p = 1 \pmod{2d}$  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,  $\text{E.PublicKeyGen}(params, sk)$  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  $\text{SwitchKeyGen}$ .) Decryption becomes  $m = \lfloor \langle \mathbf{c}, \mathbf{s} \rangle \rfloor_p$ . 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  $\text{Scale}(\mathbf{c}_1, q_j, q_{j-1}, p)$  rather than  $\text{Scale}(\mathbf{c}_1, q_j, q_{j-1}, 2)$ . 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  $R_{p_i}$  on input  $\mathbf{x}$  projected down to  $R_{p_i}^n$ . The evaluations modulo the various prime ideals do not “mix” or interact with each other.

**Theorem 4.** *Let  $p = 1 \pmod{2d}$  be a prime of size polynomial in  $\lambda$ . The RLWE-based instantiation of FHE using the ring  $R = \mathbb{Z}[x]/(x^d + 1)$  can be adapted to use the plaintext space  $R_p = \otimes_{i=1}^d R_{p_i}$  while preserving correctness and the same asymptotic performance. For any boolean circuit  $f$  of depth  $L$ , the scheme can homomorphically evaluate  $f$  on  $\ell$  sets of inputs with per-gate computation  $\tilde{O}(\lambda \cdot L^3 / \min\{d, \ell\})$ .*

When  $\ell \geq \lambda$ , the per-gate computation is only polylogarithmic in the security parameter (still cubic in  $L$ ).

## 5.2 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 – to be de-compressed to longer ciphertexts that permit homomorphic operations. Bootstrapping allows us to save memory by storing data encrypted in the compressed form – e.g., under AES.

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.

### 5.2.1 Decryption as a Circuit of Quasi-Linear Size and Logarithmic Depth

Recall that the decryption function is  $m = [[\langle \mathbf{c}, \mathbf{s} \rangle]_q]_2$ . 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-quartic.)

First, let us consider the LWE case, where  $\mathbf{c}$  and  $\mathbf{s}$  are  $n$ -dimensional integer vectors. 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 [8], 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 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  $\text{polylog}(q)$  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  $O(\log \lambda)$ .

For the RLWE case, we can use the  $R_2$  plaintext space to emulate the simpler plaintext space  $\mathbb{Z}_2$ . Using  $\mathbb{Z}_2$ , the analysis is basically the same as above, except that we mention that the DFT is used to multiply elements in  $R$ .

In practice, it would be useful to tighten up this analysis by reducing the polylogarithmic factors in the computation and the constants in the depth. Most likely this could be done by evaluating decryption using symmetric polynomials [8, 9] or with a variant of the “grade-school addition” approach used in the Gentry-Halevi implementation [10].

### 5.2.2 Bootstrapping Lazily

Bootstrapping is rather expensive computationally. In particular, the cost of bootstrapping a ciphertext is greater than the cost of a homomorphic operation by approximately a factor of  $\lambda$ . This suggests the question: can we lower per-gate computation of a bootstrapped scheme by bootstrapping *lazily* – i.e., applying the refresh procedure only at a  $1/L$  fraction of the circuit levels for some well-chosen  $L$  [11]? Here we show that the answer is yes. By bootstrapping lazily for  $L = \theta(\log \lambda)$ , we can lower the per-gate computation by a logarithmic factor.

Let us present this result somewhat abstractly. Suppose that the per-gate computation for a  $L$ -level no-bootstrapping FHE scheme is  $f(\lambda, L) = \lambda^{a_1} \cdot L^{a_2}$ . (We ignore logarithmic factors in  $f$ , since they will not affect the analysis, but one can imagine that they add a very small  $\epsilon$  to the exponent.) Suppose that bootstrapping a ciphertext requires a  $c$ -depth circuit. Since we want to be capable of evaluation depth  $L$  *after* evaluating the  $c$  levels need to bootstrap a ciphertext, the bootstrapping procedure needs to begin with ciphertexts that can be used in a  $(c + L)$ -depth circuit. Consequently, let us say that the computation needed a bootstrap a ciphertext is  $g(\lambda, c + L)$  where  $g(\lambda, x) = \lambda^{b_1} \cdot x^{b_2}$ . The overall per-gate computation is approximately  $f(\lambda, L) + g(\lambda, c + L)/L$ , a quantity that we seek to minimize.

We have the following lemma.

**Lemma 10.** *Let  $f(\lambda, L) = \lambda^{a_1} \cdot L^{a_2}$  and  $g(\lambda, L) = \lambda^{b_1} \cdot L^{b_2}$  for constants  $b_1 > a_1$  and  $b_2 > a_2 \geq 1$ . Let  $h(\lambda, L) = f(\lambda, L) + g(\lambda, c + L)/L$  for  $c = \theta(\log \lambda)$ . Then, for fixed  $\lambda$ ,  $h(\lambda, L)$  has a minimum for  $L \in [(c - 1)/(b_2 - 1), c/(b_2 - 1)]$  – i.e., at some  $L = \theta(\log \lambda)$ .*

*Proof.* Clearly  $h(\lambda, L) = +\infty$  at  $L = 0$ , then it decreases toward a minimum, and finally it eventually increases again as  $L$  goes toward infinity. Thus,  $h(\lambda, L)$  has a minimum at some positive value of  $L$ . Since  $f(\lambda, L)$  is monotonically increasing (i.e., the derivative is positive), the minimum must occur where the derivative of  $g(\lambda, c + L)/L$  is negative. We have

$$\begin{aligned} \frac{d}{dL}g(\lambda, c + L)/L &= g'(\lambda, c + L)/L - g(\lambda, c + L)/L^2 \\ &= b_2 \cdot \lambda^{b_1} \cdot (c + L)^{b_2-1}/L - \lambda^{b_1} \cdot (c + L)^{b_2}/L^2 \\ &= (\lambda^{b_1} \cdot (c + L)^{b_2-1}/L^2) \cdot (b_2 \cdot L - c - L), \end{aligned}$$

which becomes positive when  $L \geq c/(b_2 - 1)$  – i.e., the derivative is negative only when  $L = O(\log \lambda)$ . For  $L < (c - 1)/(b_2 - 1)$ , we have that the above derivative is less than  $-\lambda^{b_1} \cdot (c + L)^{b_2-1}/L^2$ , which dominates the positive derivative of  $f$ . Therefore, for large enough value of  $\lambda$ , the value  $h(\lambda, L)$  has its minimum at some  $L \in [(c - 1)/(b_2 - 1), c/(b_2 - 1)]$ .  $\square$

This lemma basically says that, since homomorphic decryption takes  $\theta(\log \lambda)$  levels and its cost is super-linear and dominates that of normal homomorphic operations (FHE.Add and FHE.Mult), it makes sense to bootstrap lazily – in particular, once every  $\theta(\log \lambda)$  levels. (If one bootstrapped even more lazily than this, the super-linear cost of bootstrapping begins to ensure that the (amortized) per-gate cost of bootstrapping alone is increasing.) It is easy to see that, since the per-gate computation is dominated by bootstrapping, bootstrapping lazily every  $\theta(\log \lambda)$  levels reduces the per-gate computation by a factor of  $\theta(\log \lambda)$ .

### 5.3 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 5.1, 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  $R_{p_1}$ . Roughly, the Pack function should take a bunch of ciphertexts  $\mathbf{c}_1, \dots, \mathbf{c}_d$  that encrypt messages  $m_1, \dots, m_d \in \mathbb{Z}_p$  under key  $\mathbf{s}_1$  for modulus  $q$  and plaintext slot  $R_{p_1}$ , and then *aggregate* them into a single ciphertext  $\mathbf{c}$  under some possibly different key  $\mathbf{s}_2$  for modulus  $q$  and plaintext slot  $R_p = \otimes_{i=1}^d R_{p_i}$ , so that correctness holds with respect to all of the different plaintext slots – i.e.  $m_i = [[\langle \mathbf{c}, \mathbf{s}_2 \rangle]_q]_{p_i}$  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  $\mathbf{c}'_1, \dots, \mathbf{c}'_d$  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  $\lambda$ . 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  $\sigma_{i \rightarrow j}$  over  $R$  that map elements of  $\mathfrak{p}_i$  to elements of  $\mathfrak{p}_j$ . (See Section 5.1.1.) We note that Smart and Vercauteren [21] 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 [14] 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, our main idea is the following. If  $m$  is encoded in the free term as a number in  $\{0, \dots, p-1\}$  and if  $m = [[\langle \mathbf{c}, \mathbf{s} \rangle]_q]_{\mathfrak{p}_i}$ , then  $m = [[\langle \sigma_{i \rightarrow j}(\mathbf{c}), \sigma_{i \rightarrow j}(\mathbf{s}) \rangle]_q]_{\mathfrak{p}_j}$ . 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  $\sigma_{i \rightarrow j}$  is a homomorphism, that it maps elements of  $\mathfrak{p}_i$  to elements of  $\mathfrak{p}_j$ , and that it fixes free terms.) 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 and 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  $p$ .

**PackSetup( $\mathbf{s}_1, \mathbf{s}_2$ ):** Takes as input two secret keys  $\mathbf{s}_1, \mathbf{s}_2$ . For all  $i \in [1, d]$ , it runs  $\tau_{\sigma_{1 \rightarrow i}(\mathbf{s}_1) \rightarrow \mathbf{s}_2} \leftarrow \text{SwitchKeyGen}(\sigma_{1 \rightarrow i}(\mathbf{s}_1), \mathbf{s}_2)$ .

**Pack( $\{\mathbf{c}_i\}_{i=1}^d, \{\tau_{\sigma_{1 \rightarrow i}(\mathbf{s}_1) \rightarrow \mathbf{s}_2}\}_{i=1}^d$ ):** Takes as input ciphertexts  $\mathbf{c}_1, \dots, \mathbf{c}_d$  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 PackSetup. For all  $i$ , it does the following:

- Computes  $\mathbf{c}_i^* \leftarrow \sigma_{1 \rightarrow i}(\mathbf{c}_i)$ . (Observe:  $m_i = [[\langle \mathbf{c}_i^*, \sigma_{1 \rightarrow i}(\mathbf{s}_1) \rangle]_q]_{\mathfrak{p}_i}$  while  $0 = [[\langle \mathbf{c}_i^*, \sigma_{1 \rightarrow i}(\mathbf{s}_1) \rangle]_q]_{\mathfrak{p}_j}$  for all  $j \neq i$ .)
- Runs  $\mathbf{c}_i^\dagger \leftarrow \text{SwitchKey}(\tau_{\sigma_{1 \rightarrow i}(\mathbf{s}_1) \rightarrow \mathbf{s}_2}, \mathbf{c}_i^*)$  (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 that  $m_i = [[\langle \mathbf{c}, \mathbf{s}_2 \rangle]_q]_{\mathfrak{p}_i}$  for all  $i$ .)

**UnpackSetup( $\mathbf{s}_1, \mathbf{s}_2$ ):** Takes as input two secret keys  $\mathbf{s}_1, \mathbf{s}_2$ . For all  $i \in [1, d]$ , it runs  $\tau_{\sigma_{i \rightarrow 1}(\mathbf{s}_1) \rightarrow \mathbf{s}_2} \leftarrow \text{SwitchKeyGen}(\sigma_{i \rightarrow 1}(\mathbf{s}_1), \mathbf{s}_2)$ .

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

- Computes  $\mathbf{c}_i \leftarrow u \cdot \sigma_{i \rightarrow 1}(\mathbf{c})$ . (Observe: Assuming the noise does not wrap,  $m_i = [[\langle \mathbf{c}_i, \sigma_{i \rightarrow 1}(\mathbf{s}_1) \rangle]_q]_{p_1}$  and  $0 = [[\langle \mathbf{c}_i, \sigma_{i \rightarrow 1}(\mathbf{s}_1) \rangle]_q]_{p_j}$  for all  $j \neq 1$ .)
- Outputs  $\mathbf{c}_i^* \leftarrow \text{SwitchKey}(\tau_{\sigma_{i \rightarrow 1}(\mathbf{s}_1) \rightarrow \mathbf{s}_2}, \mathbf{c}_i)$ . (Observe: Assuming the noise does not wrap,  $m_i = [[\langle \mathbf{c}_i^*, \mathbf{s}_2 \rangle]_q]_{p_1}$  and  $0 = [[\langle \mathbf{c}_i^*, \mathbf{s}_2 \rangle]_q]_{p_j}$  for all  $j \neq 1$ .)

Splicing the Pack and Unpack procedures into our scheme FHE is tedious but pretty straightforward. Although these procedures introduce many more encrypted secret keys, this does not cause a circular security problem as long as the chain of encrypted secret keys is acyclic; then the standard hybrid argument applies. After applying Pack or Unpack, one may apply modulus reduction to reduce the noise back down to normal.

#### 5.4 More Fun with Funky Plaintext Spaces

In some cases, it might be nice to have a plaintext space isomorphic to  $\mathbb{Z}_p$  for some *large* prime  $p$  – e.g., one *exponential* in the security parameter. So far, we have been using  $R_p$  as our plaintext space, and (due to the rounding step in modulus switching) the size of the noise after modulus switching is proportional to  $p$ . When  $p$  is exponential, our previous approach for handling the noise (which keeps the magnitude of the noise polynomial in  $\lambda$ ) obviously breaks down.

To get a plaintext space isomorphic to  $\mathbb{Z}_p$  that works for exponential  $p$ , we need a new approach. Instead of using an integer modulus, we will use an *ideal* modulus  $I$  (an ideal of  $R$ ) whose *norm* is some large prime  $p$ , but such that we have a basis  $B_I$  of  $I$  that is very short – e.g.  $\|B_I\| = O(\text{poly}(d) \cdot p^{1/d})$ . Using an ideal plaintext space forces us to modify the modulus switching technique nontrivially.

Originally, when our plaintext space was  $R_2$ , each of the moduli in our “ladder” was odd – that is, they were all congruent to each other modulo 2 and relatively prime to 2. Similarly, we will have to choose each of the moduli in our new ladder so that they are all congruent to each other modulo  $I$ . (This just seems necessary to get the scaling to work, as the reader will see shortly.) This presents a difficulty, since we wanted the norm of  $I$  to be large – e.g., exponential in the security parameter. If we choose our moduli  $q_j$  to be *integers*, then we have that the integer  $q_{j+1} - q_j \in I$  – in particular,  $q_{j+1} - q_j$  is a multiple of  $I$ ’s norm, implying that the  $q_j$ ’s are exponential in the security parameter. Having such large  $q_j$ ’s does not work well in our scheme, since the underlying lattice problems becomes easy when  $q_j/B$  is exponential in  $d$  where  $B$  is a bound of the noise distribution of fresh ciphertexts, and since we need  $B$  to remain quite small for our new noise management approach to work effectively. So, instead, our ladder of moduli will also consist of ideals – in particular, principle ideals  $(q_j)$  generated by an element of  $q_j \in R$ . Specifically, it is easy to generate a ladder of  $q_j$ ’s that are all congruent to 1 moduli  $I$  by sampling appropriately-sized elements  $q_j$  of the coset  $1 + I$  (using our short basis of  $I$ ), and testing whether the principal ideal  $(q_j)$  generated by the element has appropriate norm.

Now, let us reconsider modulus switching in light of the fact that our moduli are now principal ideals. We need an analogue of Lemma 4 that works for ideal moduli.

Let us build up some notation and concepts that we will need in our new lemma. Let  $\mathcal{P}_q$  be the half-open *parallelepiped* associated to the *rotation basis* of  $q \in R$ . The rotation basis  $\mathbf{B}_q$  of  $q$  is the  $d$ -dimensional basis formed by the coefficient vectors of the polynomials  $x^i q(x) \bmod f(x)$  for  $i \in [0, d-1]$ . The associated parallelepiped is  $\mathcal{P}_q = \{\sum z_i \cdot \mathbf{b}_i : \mathbf{b}_i \in \mathbf{B}_q, z_i \in [-1/2, 1/2)\}$ . We need two concepts associated to this parallelepiped. First, we will still use the notation  $[a]_q$ , but where  $q$  is now an  $R$ -element rather than integer. This notation refers to  $a$  reduced modulo the *rotation basis* of  $a$  – i.e., the element  $[a]_q$  such that  $[a]_q - a \in qR$  and  $[a]_q \in \mathcal{P}_q$ . Next, we need notions of the *inner radius*  $r_{q,in}$  and *outer radius*  $r_{q,out}$  of  $\mathcal{P}_q$  – that is, the

largest radius of a ball that is circumscribed by  $\mathcal{P}_q$ , and the smallest radius of a ball that circumscribes  $\mathcal{P}_q$ . It is possible to choose  $q$  so that the ratio  $r_{q,out}/r_{q,in}$  is  $\text{poly}(d)$ . For example, this is true when  $q$  is an integer. For a suitable value of  $f(x)$  that determines our ring, such as  $f(x) = x^d + 1$ , the expected value of ratio will be  $\text{poly}(d)$  even if  $q$  is sampled uniformly (e.g., according to discrete Gaussian distribution centered at 0). More generally, we will refer to  $r_{\mathbf{B},out}$  as the outer radius associated to the parallelepiped determined by basis  $\mathbf{B}$ . Also, in the field  $\mathbb{Q}(x)/f(x)$  overlying this ring, it will be true with overwhelming probability, if  $q$  is sampled uniformly, that  $\|q^{-1}\| = 1/\|q\|$  up to a  $\text{poly}(d)$  factor. For convenience, let  $\alpha(d)$  be a polynomial such that  $\|q^{-1}\| = 1/\|q\|$  up to a  $\alpha(d)$  factor and moreover  $r_{q,out}/r_{q,in}$  is at most  $\alpha(d)$  with overwhelming probability. For such an  $\alpha$ , we say  $q$  is  $\alpha$ -good. Finally, in the lemma,  $\gamma_R$  denotes the expansion factor of  $R$  – i.e.,  $\max\{\|\mathbf{a} \cdot \mathbf{b}\|/\|\mathbf{a}\|\|\mathbf{b}\| : \mathbf{a}, \mathbf{b} \in R\}$ .

**Lemma 11.** *Let  $q_1$  and  $q_2$ ,  $\|q_1\| < \|q_2\|$ , be two  $\alpha$ -good elements of  $R$ . Let  $\mathbf{B}_I$  be a short basis (with outer radius  $r_{\mathbf{B}_I,out}$ ) of an ideal  $I$  of  $R$  such that  $q_1 - q_2 \in I$ . Let  $\mathbf{c}$  be an integer vector and  $\mathbf{c}' \leftarrow \text{Scale}(\mathbf{c}, q_2, q_1, I)$  – that is,  $\mathbf{c}'$  is an  $R$ -element at most  $2r_{\mathbf{B}_I,out}$  distant from  $(q_1/q_2) \cdot \mathbf{c}$  such that  $\mathbf{c}' - \mathbf{c} \in I$ . Then, for any  $\mathbf{s}$  with*

$$\|[\langle \mathbf{c}, \mathbf{s} \rangle]_{q_2}\| < \left( r_{q_2,in}/\alpha(d)^2 - (\|q_2\|/\|q_1\|)\gamma_R \cdot 2r_{\mathbf{B}_I,out} \cdot \ell_1^{(R)}(\mathbf{s}) \right) / (\alpha(d) \cdot \gamma_R^2)$$

we have

$$[\langle \mathbf{c}', \mathbf{s} \rangle]_{q_1} = [\langle \mathbf{c}, \mathbf{s} \rangle]_{q_2} \bmod I \quad \text{and} \quad \|[\langle \mathbf{c}', \mathbf{s} \rangle]_{q_1}\| < \alpha(d) \cdot \gamma_R^2 \cdot (\|q_1\|/\|q_2\|) \cdot \|[\langle \mathbf{c}, \mathbf{s} \rangle]_{q_2}\| + \gamma_R \cdot 2r_{\mathbf{B}_I,out} \cdot \ell_1^{(R)}(\mathbf{s})$$

where  $\ell_1^{(R)}(\mathbf{s})$  is defined as  $\sum_i \|\mathbf{s}[i]\|$ .

*Proof.* We have

$$[\langle \mathbf{c}, \mathbf{s} \rangle]_{q_2} = \langle \mathbf{c}, \mathbf{s} \rangle - kq_2$$

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

$$e_{q_1} = \langle \mathbf{c}', \mathbf{s} \rangle - kq_1 \in R$$

Note that  $e_{q_1} = [\langle \mathbf{c}', \mathbf{s} \rangle]_{q_1} \bmod q_1$ . We claim that  $\|e_{q_1}\|$  is so small that  $e_{q_1} = [\langle \mathbf{c}', \mathbf{s} \rangle]_{q_1}$ . We have:

$$\begin{aligned} \|e_{q_1}\| &= \| -kq_1 + \langle (q_1/q_2) \cdot \mathbf{c}, \mathbf{s} \rangle + \langle \mathbf{c}' - (q_1/q_2) \cdot \mathbf{c}, \mathbf{s} \rangle \| \\ &\leq \| -kq_1 + \langle (q_1/q_2) \cdot \mathbf{c}, \mathbf{s} \rangle \| + \| \langle \mathbf{c}' - (q_1/q_2) \cdot \mathbf{c}, \mathbf{s} \rangle \| \\ &\leq \gamma_R \cdot \|q_1/q_2\| \cdot \|[\langle \mathbf{c}, \mathbf{s} \rangle]_{q_2}\| + \gamma_R \cdot 2r_{\mathbf{B}_I,out} \cdot \ell_1^{(R)}(\mathbf{s}) \\ &\leq \gamma_R^2 \cdot \|q_1\| \cdot \|q_2^{-1}\| \cdot \|[\langle \mathbf{c}, \mathbf{s} \rangle]_{q_2}\| + \gamma_R \cdot 2r_{\mathbf{B}_I,out} \cdot \ell_1^{(R)}(\mathbf{s}) \\ &\leq \alpha(d) \cdot \gamma_R^2 \cdot (\|q_1\|/\|q_2\|) \cdot \|[\langle \mathbf{c}, \mathbf{s} \rangle]_{q_2}\| + \gamma_R \cdot 2r_{\mathbf{B}_I,out} \cdot \ell_1^{(R)}(\mathbf{s}) \end{aligned}$$

By the final expression above, we see that the magnitude of  $e_{q_1}$  may actually be less than the magnitude of  $e_{q_2}$  if  $\|q_1\|/\|q_2\|$  is small enough. Let us continue with the inequalities, substituting in the bound for  $\|[\langle \mathbf{c}, \mathbf{s} \rangle]_{q_2}\|$ :

$$\begin{aligned} \|e_{q_1}\| &\leq \alpha(d) \cdot \gamma_R^2 \cdot (\|q_1\|/\|q_2\|) \cdot \left( r_{q_2,in}/\alpha(d)^2 - (\|q_2\|/\|q_1\|)\gamma_R \cdot 2r_{\mathbf{B}_I,out} \cdot \ell_1^{(R)}(\mathbf{s}) \right) / (\alpha(d) \cdot \gamma_R^2) \\ &\quad + \gamma_R \cdot 2r_{\mathbf{B}_I,out} \cdot \ell_1^{(R)}(\mathbf{s}) \\ &\leq (\|q_1\|/\|q_2\|) \cdot \left( r_{q_2,in}/\alpha(d)^2 - (\|q_2\|/\|q_1\|)\gamma_R \cdot 2r_{\mathbf{B}_I,out} \cdot \ell_1^{(R)}(\mathbf{s}) \right) + \gamma_R \cdot 2r_{\mathbf{B}_I,out} \cdot \ell_1^{(R)}(\mathbf{s}) \\ &\leq \left( r_{q_1,in} - \gamma_R \cdot 2r_{\mathbf{B}_I,out} \cdot \ell_1^{(R)}(\mathbf{s}) \right) + \gamma_R \cdot 2r_{\mathbf{B}_I,out} \cdot \ell_1^{(R)}(\mathbf{s}) \\ &= r_{q_1,in} \end{aligned}$$

Since  $\|e_{q_1}\| < r_{q_1, in}$ ,  $e_{q_1}$  is inside the parallelepiped  $\mathcal{P}_{q_1}$  and it is indeed true that  $e_{q_1} = [\langle \mathbf{c}', \mathbf{s} \rangle]_{q_1}$ . Furthermore, we have  $[\langle \mathbf{c}', \mathbf{s} \rangle]_{q_1} = e_{q_1} = \langle \mathbf{c}', \mathbf{s} \rangle - kq_1 = \langle \mathbf{c}, \mathbf{s} \rangle - kq_2 = [\langle \mathbf{c}, \mathbf{s} \rangle]_{q_2} \pmod I$ .  $\square$

The bottom line is that we can apply the modulus switching technique to moduli that are ideals, and this allows us to use, if desired, plaintext spaces that are very large (exponential in the security parameter) and that have properties that are often desirable (such as being isomorphic to a large prime field).

## 5.5 Other Optimizations

If one is willing to assume circular security, the keys  $\{\mathbf{s}_j\}$  may all be the same, thereby permitting a public key of size independent of  $L$ .

While it is not necessary, squashing may still be a useful optimization in practice, as it can be used to lower the depth of the decryption function, thereby reducing the size of the largest modulus needed in the scheme, which may improve efficiency.

For the LWE-based scheme, one can use key switching to gradually reduce the dimension  $n_j$  of the ciphertext (and secret key) vectors as  $q_j$  decreases – that is, as one traverses to higher levels in the circuit. As  $q_j$  decreases, the associated LWE problem becomes (we believe) progressively harder (for a fixed noise distribution  $\chi$ ). This allows one to gradually reduce the dimension  $n_j$  without sacrificing security, and reduce ciphertext length faster (as one goes higher in the circuit) than one could simply by decreasing  $q_j$  alone.

## 6 Summary and Future 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  $\lambda$ .

While these schemes should perform significantly better than previous FHE schemes, we caution that the polylogarithmic factors in the per-gate computation are large. One future direction toward a truly practical scheme is to tighten up these polylogarithmic factors considerably.

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

## References

- [1] 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.
- [2] 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.
- [3] Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) lwe. Manuscript, to appear in FOCS 2011, available at <http://eprint.iacr.org/2011/344>.
- [4] Zvika Brakerski and Vinod Vaikuntanathan. Fully homomorphic encryption from ring-lwe and security for key dependent messages. Manuscript, to appear in CRYPTO 2011.
- [5] 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.

- [6] 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>.
- [7] Craig Gentry. *A fully homomorphic encryption scheme*. PhD thesis, Stanford University, 2009. [crypto.stanford.edu/craig](http://crypto.stanford.edu/craig).
- [8] Craig Gentry. Fully homomorphic encryption using ideal lattices. In Michael Mitzenmacher, editor, *STOC*, pages 169–178. ACM, 2009.
- [9] Craig Gentry and Shai Halevi. Fully homomorphic encryption without squashing using depth-3 arithmetic circuits. Manuscript, to appear in FOCS 2011, available at <http://eprint.iacr.org/2011/279>.
- [10] 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.
- [11] Shai Halevi, 2011. Personal communication.
- [12] 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.
- [13] Kristin Lauter, Michael Naehrig, and Vinod Vaikuntanathan. Can homomorphic encryption be practical? Manuscript at <http://eprint.iacr.org/2011/405>, 2011.
- [14] 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.
- [15] 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.
- [16] Chris Peikert. Public-key cryptosystems from the worst-case shortest vector problem: extended abstract. In *STOC*, pages 333–342. ACM, 2009.
- [17] 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.
- [18] Oded Regev. The learning with errors problem (invited survey). In *IEEE Conference on Computational Complexity*, pages 191–204. IEEE Computer Society, 2010.
- [19] Ron Rivest, Leonard Adleman, and Michael L. Dertouzos. On data banks and privacy homomorphisms. In *Foundations of Secure Computation*, pages 169–180, 1978.
- [20] 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.
- [21] Nigel P. Smart and Frederik Vercauteren. Fully homomorphic SIMD operations. Manuscript at <http://eprint.iacr.org/2011/133>, 2011.

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