Bootstrapping for HElib*

Shai Halevi
Algorand Foundation

Victor Shoup
NYU

April 20, 2020

Abstract

Gentry's bootstrapping technique is still the only known method of obtaining fully homomorphic encryption where the system's parameters do not depend on the complexity of the evaluated functions. Bootstrapping involves a *recryption* procedure where the scheme's decryption algorithm is evaluated homomorphically. Prior to this work there were very few implementations of recryption, and fewer still that can handle "packed ciphertexts" that encrypt vectors of elements.

In the current work, we report on an implementation of recryption of fully-packed ciphertexts using the HElib library for somewhat-homomorphic encryption. This implementation required extending previous recryption algorithms from the literature, as well as many aspects of the HElib library. Our implementation supports bootstrapping of packed ciphertexts over many extension fields/rings. One example that we tested involves ciphertexts that encrypt vectors of 1024 elements from $\mathrm{GF}(2^{16})$. In that setting, the recryption procedure takes under 3 minutes (at security-level ≈ 80) on a single core, and allows a multiplicative depth-11 computation before the next recryption is needed.

This report updates the results that we reported in Eurocrypt 2015 in several ways. Most importantly, it includes a much more robust method for deriving the parameters, ensuring that recryption errors only occur with negligible probability. Many aspects of this analysis are proven, and for the few well-specified heuristics that we made, we report on thorough experimentation to validate them. The procedure that we describe here is also significantly more efficient than in the previous version, incorporating many optimizations that were reported elsewhere (such as more efficient linear transformations) and adding a few new ones. Finally, our implementation now also incorporates Chen and Han's techniques from Eurocrypt 2018 for more efficient digit extraction (for some parameters), as well as for "thin bootstrapping" when the ciphertext is only sparsely packed.

Keywords: Bootstrapping, Homomorphic Encryption, Implementation

*Work partially done in IBM Research. The second author was supported in part by the Office of the Director of National Intelligence (ODNI), Intelligence Advanced Research Projects Activity (IARPA), via 2019-19-020700006. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of ODNI, IARPA, or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for governmental purposes notwithstanding any copyright annotation therein.

Contents

1 Introduction

Homomorphic Encryption (HE) [35, 16] enables computation of arbitrary functions on encrypted data without knowing the secret key. All current HE schemes follow Gentry's outline from [16], where fresh ciphertexts are "noisy" to ensure security and this noise grows with every operation until it overwhelms the signal and causes decryption errors. This yields a "somewhat homomorphic" scheme (SWHE) that can only evaluate low-depth circuits, which can then be converted to a "fully homomorphic" scheme (FHE) using bootstrapping. Gentry described a *recryption* operation, where the decryption procedure of the scheme is run homomorphically, using an encryption of the secret key that can be found in the public key, resulting in a new ciphertext that encrypts the same plaintext but has smaller noise.

The last decade saw a large body of work improving many aspects of homomorphic encryption in general and recryption in particular, as well as a multitude of implementations of practically usable homomorphic encryption. Some of those implementations even support bootstrapping, most of which were subsequent to the initial report of this work. Early implementations of recryption prior to our work include the Gentry-Halevi implementation of Gentry's cryptosystem [17, 16], the implementation of Coron et al. of the DGHV scheme over the integers [11, 6, 12, 14], and the Rohloff-Cousins implementation of the NTRU-based cryptosystem [36, 29, 31].

Here we report on our implementation of recryption for the cryptosystem of Brakerski, Gentry and Vaikuntanathan (BGV) [4]. We implemented recryption on top of the open-source library `HElib` [26, 23], which implements the ring-LWE variant of BGV. Our implementation includes both new algorithmic designs as well as re-engineering of some aspects of `HElib`. As noted in [23], the choice of homomorphic primitives in `HElib` was guided to a large extent by the desire to support recryption, but nonetheless in the course of our implementation we had to extend the implementation of some of these primitives (e.g., matrix-vector multiplication), and also implement a few new ones (e.g., polynomial evaluation).

The `HElib` library is "focused on effective use of the Smart-Vercauteren ciphertext packing techniques [38] and the Gentry-Halevi-Smart optimizations [19]," so in particular we implemented recryption for "fully-packed" ciphertexts. Specifically, our implementation supports recryption of ciphertexts that encrypt vectors of elements from extension fields (or rings). Importantly, our recryption procedure itself has sufficiently low depth so as to allow significant processing between recryptions while keeping the lattice dimension reasonable to maintain efficiency.

Our experimental results are described in Section 7. Some example settings include: encrypting vectors of 1024 elements from $\mathrm{GF}(2^{16})$ with a security level of 80 bits, where recryption takes under 3 minutes and allows additional computations of multiplicative depth 11 between recryptions; and encrypting vectors of 960 elements from $\mathrm{GF}(2^{24})$ with a security level of 80 bits, where recryption takes under 5 minutes and allows additional computations of multiplicative depth 15 between recryptions.[1]

Compared to the previous recrypt implementations, ours offers several advantages in both flexibility and speed. Our implementation supports packed ciphertexts that encrypt vectors from the more general extension fields (and rings) already supported by `HElib`. Some examples that we tested include vectors over the fields $\mathrm{GF}(2^{16})$, $\mathrm{GF}(2^{25})$, $\mathrm{GF}(2^{24})$, $\mathrm{GF}(2^{36})$, $\mathrm{GF}(17^{40})$, and $\mathrm{GF}(127^{36})$, as well as degree-21 and degree-30 extensions of the ring $\mathbb{Z}_{256}$.

1.1 Concurrent and subsequent work

Concurrently with our work, Ducas and Micciancio described a new bootstrapping procedure [15]. This procedure is applied to Regev-like ciphertexts [34] that encrypt a single bit, using a secret key

[1] The latter setting is conducive to homomorphic AES, see, e.g., the long version of [20].

encrypted similarly to the new cryptosystem of Gentry at al. [22]. They reported on an implementation of their scheme, where they can perform a NAND operation followed by recryption in less than a second. This was later improved and extended by Chillotti et al. [9, 10] that implemented bootstrapping in the TFHE library achieving a single-bit recryption speed of 13 milliseconds. While this wall-clock time is much faster than our work, our implementation is about ten times faster in terms of amortized per-bit running time (see below). It remains a very interesting open problem to combine those techniques with ours, achieving a "best of both worlds" implementation.

Another notable subsequent line of work is bootstrapping for the CKKS approximate-number scheme [8, 7, 28, 27], some of which use optimizations that were introduced in the initial version of the current work.

1.1.1 Improvements subsequent to the Eurocrypt 2015 paper

Since the original publication of our bootstrapping techniques in Eurocrypt 2015 [24], we have made a number of improvements, incorporating many optimizations that were reported elsewhere and adding a few new ones. For example, we have improved our matrix multiplication algorithms significantly, as reported in [25]. We have also significantly improved the overall robustness and efficiency of the noise management in `HElib`. Some of these techniques are specific to bootstrapping, and we report those here (see Section 6).

We have also adapted techniques of Chen and Han [5]. In particular, we adapted their techniques for "digit extraction", which can allow for more noise-efficient bootstrapping (for some parameters). In addition, we adapted their techniques for "thin bootstrapping", where each slot contains an element of the base field (or ring), rather than an extension field (or ring). This can be advantageous in applications where there is no natural way to exploit the extension field (or ring) structure of the slots. We implemented a variant of their technique, details of which may be found in [25]. We ran various experiments with this "thin bootstrapping" algorithm. For the examples above with plaintext space $\mathrm{GF}(2^{16})$ and $\mathrm{GF}(2^{24})$ examples mentioned above, if we restrict the plaintext space to $\mathrm{GF}(2)$, the running times drop to 15 and 19 seconds, respectively. In addition, we calculated the "amortized time" for thin bootstrapping, in which we took the total bootstrapping time, divided that by the number of slots, and divided that by the number of usable multiplicative levels between recryptions. In the two examples mentioned above, the "amortized time" of bootstrapping associated with one multiplication $\mathrm{GF}(2)$ is about 1.1 milliseconds. If we add to this the amortized time of the multiplication itself (i.e., the multiplication time divded by the number of slots), the total amortized running time per multiplication in $\mathrm{GF}(2)$ is about 1.3 milliseconds. In another example, for the plaintext space $\mathbb{Z}_{2^8}$, we can achieve an amortized time for bootstrapping of 2.1 milliseconds. If we add to this the amortized time of the multiplication itself, the total amortized running time per multiplication in $\mathbb{Z}_{2^8}$ is about 2.4 milliseconds.

We have also added support for multi-threading to `HElib`, and have implemented our bootstrapping routine to exploit multiple cores when available. In our experiments, with up to 8 cores, we get nearly linear speedup for our bootstrapping routine. For thin bootstrapping, we get somewhat less speedup (see more details in Section 7.2).

1.2 Algorithmic Aspects

Our recryption procedure follows the high-level structure introduced by Gentry et al. [21], and uses the tensor decomposition of Alperin-Sheriff and Peikert [1] for the linear transformations. However, those two works only dealt with characteristic-2 plaintext spaces so we had to extend some of their algorithmic components to deal with characteristics $p > 2$, see Section 5.

Also, to get an efficient implementation, we had to make the decomposition from [1] explicit, specialize it to cases that support very-small-depth circuits, and align the different representations to reduce the required data-movement and multiplication-by-constant operations. These aspects are described in Section 4. One significant difference between our implementation and the procedure of Alperin-Sheriff and Peikert [1] is that we *do not use* the ring-switching techniques of Gentry et al. [18] (see discussion in Appendix 8).

1.3 Organization

We describe our notations and give some background information on the BGV cryptosystem and the `HElib` library in Section 2. In Section 3 we provide an overview of the high-level recryption procedure from [21] and our variant of it. We then describe in detail our implementation of the linear transformations in Section 4 and the non-linear parts in Section 5. In Section 5.4 we explain how all these parts are put together in our implementation. In Section 6 we describe how various parameters are chosen to ensure a low probability of error. In Section 7 we discuss our performance results. We conclude with directions for future work in Section 9.

2 Notations and Background

For integer z, we denote by $[z]_q$ the reduction of z modulo q into the interval $[-q/2, q/2)$, except that for $q = 2$ we reduce to $(-1, 1]$. This notation extends to vectors and matrices coordinate-wise, and to elements of other algebraic groups/rings/fields by reducing their coefficients in some convenient basis.

For an integer z (positive or negative) we consider the base-p representation of z and denote its digits by $z\langle 0\rangle_p$, $z\langle 1\rangle_p, \cdots$. When p is clear from the context we omit the subscript and just write $z\langle 0\rangle, z\langle 1\rangle, \cdots$. When $p = 2$ we consider a 2's-complement representation of signed integers (i.e., the top bit represents a large negative number). For an odd p we consider balanced mod-p representation where all the digits are in $[-\frac{p-1}{2}, \frac{p-1}{2}]$.

For indexes $0 \le i \le j$ we also denote by $z\langle j, \ldots, i\rangle_p$ the integer whose base-p expansion is $z\langle j\rangle \cdots z\langle i\rangle$ (with $z\langle i\rangle$ the least significant digit). Namely, for odd p we have $z\langle j, \ldots, i\rangle_p = \sum_{k=i}^{j} z\langle k\rangle p^{k-i}$, and for $p = 2$ we have $z\langle j, \ldots, i\rangle_2 = (\sum_{k=i}^{j-1} z\langle k\rangle 2^{k-i}) - z\langle j\rangle 2^{j-i}$. The properties of these representations that we use in our procedures are the following:

- For any $r \ge 1$ and any integer z we have $z = z\langle r-1, \ldots, 0\rangle \pmod{p^r}$.
- If the representation of z is $d_{r-1}, \ldots, d_0$ then the representation of $z \cdot p^r$ is $d_{r-1}, \ldots, d_0, \overbrace{0, \cdots, 0}^{r \text{ zeros}}$.
- If p is odd and $|z| < p^e/2$ then the digits in positions e and up in the representation of z are all zero.
- If $p = 2$ and $|z| < 2^{e-1}$, then the bits in positions $e-1$ and up in the representation of z, are either all zero if $z \ge 0$ or all one if $z < 0$.

2.1 The BGV Cryptosystem

The BGV ring-LWE-based somewhat-homomorphic scheme [4] is defined over a ring $R \stackrel{\text{def}}{=} \mathbb{Z}[X]/(\Phi_m(X))$, where $\Phi_m(X)$ is the mth cyclotomic polynomial. For an arbitrary integer modulus N (not necessarily prime) we denote the ring $R_N \stackrel{\text{def}}{=} R/NR$. We often identify elements in R (or R_N) with their representation is some convenient basis, e.g., their coefficient vectors as polynomials. When dealing with R_N, we assume that the coefficients are in $[-N/2, N/2)$ (except for R_2 where the coefficients are in $\{0, 1\}$). We discuss these representations in some more detail in Section 4.1.

As implemented in HElib, the native plaintext space of the BGV cryptosystem is R_{p^r} for a prime power p^r. The scheme uses a large number of different moduli, and a ciphertext relative to one of these moduli q is a vector $\mathsf{ct} \in (R_q)^2$. At any point, a ciphertext is defined relative to one modulus, but that modulus keeps changing throughout the computation via mod-up and mod-down operations.

The secret keys are elements $\mathfrak{s} \in R$ with "small" coefficients (chosen in $\{0, \pm 1\}$ in HElib), and we view $\mathfrak{s}$ as the second element of the 2-vector $\mathsf{sk} = (1, \mathfrak{s}) \in R^2$. A ciphertext $\mathsf{ct} = (\mathfrak{c}_0, \mathfrak{c}_1)$ encrypts a plaintext element $\mathfrak{m} \in R_{p^r}$ with respect to $\mathsf{sk} = (1, \mathfrak{s})$ and modulus q if we have $[\langle \mathsf{sk}, \mathsf{ct} \rangle]_q = [\mathfrak{c}_0 + \mathfrak{s} \cdot \mathfrak{c}_1]_q = \mathfrak{m} + p^r \cdot \mathfrak{e}$ (in R) for a small noise term $p^r \cdot \mathfrak{e}$ (with norm $\ll q$).

The noise term grows with homomorphic operations of the cryptosystem, and switching from q to $q' < q$ is used to decrease the noise term roughly by the ratio q'/q. Once we have a ciphertext ct relative to the smallest modulus, we can no longer use that technique to reduce the noise. To enable further computation, we need to use Gentry's bootstrapping technique [16], whereby we "recrypt" the ciphertext ct, to obtain a new ciphertext ct^* that encrypts the same element of R_{p^r} with respect to a larger modulus.

In HElib, each modulus q is a product of a number of machine-word sized primes. Elements of the ring R_q are typically represented in DoubleCRT format: as a vector of polynomials modulo each small prime t, where each of these polynomials is represented by its evaluation at the primitive mth roots of unity in $\mathbb{Z}_t$. In DoubleCRT format, elements of R_q may be added and multiplied in linear time. Conversion between DoubleCRT representation and the more natural coefficient representation may be effected in quasi-linear time using the FFT.

2.2 Encoding Vectors in Plaintext Slots

As observed by Smart and Vercauteren [38], an element of the native plaintext space $\alpha \in R_{p^r}$ can be viewed as encoding a vector of "plaintext slots" containing elements from some smaller ring extension of $\mathbb{Z}_{p^r}$ via Chinese remaindering. In this way, a single arithmetic operation on α corresponds to the same operation applied component-wise to all the slots.

Specifically, suppose the factorization of $\Phi_m(X)$ modulo p^r is $\Phi_m(X) \equiv F_1(X) \cdots F_k(X) \pmod{p^r}$, where each F_i has the same degree d, which is equal to the order of p modulo m. (This factorization can be obtained by factoring $\Phi_m(X)$ modulo p and then Hensel lifting.) From the CRT for polynomials, we have the isomorphism

$$R_{p^r} \cong \bigoplus_{i=1}^{k} (\mathbb{Z}[X]/(p^r, F_i(X))).$$

Let us now define $E \stackrel{\text{def}}{=} \mathbb{Z}[X]/(p^r, F_1(X))$, and let ζ be the residue class of X in E, which is a principal mth root of unity, so that $E = \mathbb{Z}/(p^r)[\zeta]$. The rings $\mathbb{Z}[X]/(p^r, F_i(X))$ for $i = 1, \dots, k$ are all isomorphic to E, and their direct product is isomorphic to R_{p^r}, so we get an isomorphism between R_{p^r} and E^k. HElib makes extensive use of this isomorphism, representing it explicitly as follows. It maintains a set $S \subset \mathbb{Z}$ that forms a complete system of representatives for the quotient group $\mathbb{Z}_m^*/\langle p \rangle$, i.e., it contains exactly one element from every residue class. Then we use a ring isomorphism

$$\begin{aligned} R_{p^r} &\to \bigoplus_{h \in S} E \\ \alpha &\mapsto \{\alpha(\zeta^h)\}_{h \in S}. \end{aligned} \tag{1}$$

Here, if α is the residue class $a(X) + (p^r, \Phi_m(X))$ for some $a(X) \in \mathbb{Z}[X]$, then $\alpha(\zeta^h) = a(\zeta^h) \in E$, which is independent of the representative $a(X)$.

This representation allows HElib to effectively pack $k \stackrel{\text{def}}{=} |S| = |\mathbb{Z}_m^*/\langle p\rangle|$ elements of E into different "slots" of a single plaintext. Addition and multiplication of ciphertexts act on the slots of the corresponding plaintext in parallel.

2.3 Hypercube structure and one-dimensional rotations

Beyond addition and multiplications, we can also manipulate elements in R_{p^r} using a set of automorphisms on R_{p^r} of the form $a(X) \mapsto a(X^j)$, or in more detail

$$\begin{aligned} \tau_j : R_{p^r} &\to R_{p^r} \\ a(X) + (p^r, \Phi_m(X)) &\mapsto a(X^j) + (p^r, \Phi_m(X)). \end{aligned} \qquad (j \in \mathbb{Z}_m^*)$$

We can homomorphically apply these automorphisms by applying them to the ciphertext elements and then performing "key switching" (see [4, 19]). As discussed in [19], these automorphisms induce a hypercube structure on the plaintext slots, where the hypercube structure depends on the structure of the group $\mathbb{Z}_m^*/\langle p\rangle$. Specifically, HElib keeps a hypercube basis $g_1, \ldots, g_n \in \mathbb{Z}_m^*$, together with orders $\ell_1, \ldots, \ell_n \in \mathbb{Z}_{>0}$, and then defines the set S of representatives for $\mathbb{Z}_m^*/\langle p\rangle$ (which is used for slot mapping Eqn. (1)) as

$$S \stackrel{\text{def}}{=} \{g_1^{e_1} \cdots g_n^{e_n} : 0 \le e_i < \ell_i,\ i = 1, \ldots, n\}. \tag{2}$$

Note that ℓ_i need not be the order of g_i in $\mathbb{Z}_m^*$. This basis defines an n-dimensional hypercube structure on the plaintext slots, where slots are indexed by tuples $(e_1, \ldots, e_n)$ with $0 \le e_i < \ell_i$. If we fix $e_1, \ldots, e_{i-1}, e_{i+1}, \ldots, e_n$, and let e_i range over $0, \ldots, \ell_i - 1$, we get a set of ℓ_i slots, indexed by $(e_1, \ldots, e_i, \ldots e_n)$, which we refer to as a *hypercolumn in dimension* i (and there are k/ℓ_i such hypercolumns). Using automorphisms, we can efficiently perform rotations in any dimension; a rotation by v in dimension i maps a slot indexed by $(e_1, \ldots, e_i, \ldots, e_n)$ to the slot indexed by $(e_1, \ldots, e_i + v \bmod \ell_i, \ldots, e_n)$. Below we denote this operation by ρ_i^v.

We can implement ρ_i^v by applying either one automorphism or two: if the order of g_i in $\mathbb{Z}_m^*$ is ℓ_i, then we get by with just a single automorphism, $\rho_i^v(\alpha) = \tau_{g_i^v}(\alpha)$. If the order of g_i in $\mathbb{Z}_m^*$ is different from ℓ_i then we need to implement this rotation using two shifts: specifically, we use a constant "0-1 mask value" mask that selects some slots and zeros-out the others, and use two automorphisms with exponents $e = g_i^v \bmod m$ and $e' = g_i^{v-\ell_i} \bmod m$, setting

$$\rho_i^v(\alpha) = \tau_e(\mathsf{mask} \cdot \alpha) + \tau_{e'}((1 - \mathsf{mask}) \cdot \alpha).$$

In the first case (where one automorphism suffices) we call i a "good dimension", and otherwise we call i a "bad dimension".

2.4 Frobenius and linearized polynomials

We define $\sigma \stackrel{\text{def}}{=} \tau_p$, which is the Frobenius map on R_{p^r}. It acts on each slot independently as the Frobenius map σ_E on E, which sends ζ to ζ^p and leaves elements of $\mathbb{Z}_{p^r}$ fixed. (When $r = 1$, σ is the same as the pth power map on E.) For any $\mathbb{Z}_{p^r}$-linear transformation on E, denoted M, there exist unique constants $\theta_0, \ldots, \theta_{d-1} \in E$ such that $M(\eta) = \sum_{f=0}^{d-1} \theta_f \sigma_E^f(\eta)$ for all $\eta \in E$. When $r = 1$, this follows from the general theory of linearized polynomials (see, e.g., Theorem 10.4.4 on p. 237 of [37]), and these constants are readily computable by solving a system of equations mod p; the case of $r > 1$ is similar, and can be thought of as Hensel-lifting these mod-p solutions to a solution mod p^r.

In the special case where the image of M is the sub-ring $\mathbb{Z}_{p^r}$ of E, the constants θ_f are obtained as $\theta_f = \sigma_E^f(\theta_0)$ for $f = 1, \ldots, d-1$; again, this is standard field theory if $r = 1$, and is easily established for $r > 1$ as well.

Using linearized polynomials, we may effectively apply a fixed linear map to each slot of a plaintext element $\alpha \in R_{p^r}$ (either the same or different maps in each slot) by computing $\sum_{f=0}^{d-1} \kappa_f \sigma^f(\alpha)$, where the κ_f's are R_{p^r}-constants obtained by embedding appropriate E-constants in the slots.

3 Overview of the Recryption Procedure

Recall that the recryption procedure is given a BGV ciphertext $\mathsf{ct} = (\mathfrak{c}_0, \mathfrak{c}_1)$, defined relative to secret-key $\mathsf{sk} = (1, \mathfrak{s})$, modulus q, and plaintext space p^r, namely, we have $[\langle \mathsf{sk}, \mathsf{ct}\rangle]_q \equiv \mathfrak{m} \pmod{p^r}$ with $\mathfrak{m}$ being the plaintext. Also we have the guarantee that the noise in ct is still rather small.

The goal of the recryption procedure is to produce another ciphertext ct^* that encrypts the same plaintext element $\mathfrak{m}$ relative to the same secret key, but relative to a much larger modulus $Q \gg q$ and with a much smaller relative noise. Our implementation uses roughly the same high-level structure for the recryption procedure as in [21, 1], below we briefly recall the structure from [21] and then describe our variant of it.

3.1 The GHS Recryption Procedure

The recryption procedure from [21] (for plaintext space $p = 2$) begins by using modulus-switching to compute another ciphertext that encrypts the same plaintext as ct, but relative to a specially chosen modulus $\tilde{q} = 2^e + 1$ (for some integer e).

Denote the resulting ciphertext by ct', the rest of the recryption procedure consists of homomorphic implementation of the decryption formula $\mathfrak{m} \leftarrow [[\langle \mathsf{sk}, \mathsf{ct}'\rangle]_{\tilde{q}}]_2$, applied to an encryption of sk that can be found in the public key. Note that in this formula we know $\mathsf{ct}' = (\mathfrak{c}'_0, \mathfrak{c}'_1)$ explicitly, and it is sk that we process homomorphically. It was shown in [21] that for the special modulus $\tilde{q}$, the decryption procedure can be evaluated (roughly) by computing $\mathfrak{u} \leftarrow [\langle \mathsf{sk}, \mathsf{ct}'\rangle]_{2^{e+1}}$ and then $\mathfrak{m} \leftarrow \mathfrak{u}\langle e\rangle \oplus \mathfrak{u}\langle 0\rangle$.[2]

To enable recryption, the public key is augmented with an encryption of the secret key $\mathfrak{s}$, relative to a (much) larger modulus $Q \gg \tilde{q}$, *and also relative to a larger plaintext space* 2^{e+1}. Namely this is a ciphertext $\tilde{\mathsf{ct}}$ such that $[\langle \mathsf{sk}, \tilde{\mathsf{ct}}\rangle]_Q = \mathfrak{s} \pmod{2^{e+1}}$. Recalling that all the coefficients in $\mathsf{ct}' = (\mathfrak{c}'_0, \mathfrak{c}'_1)$ are smaller than $\tilde{q}/2 < 2^{e+1}/2$, we consider $\mathfrak{c}'_0, \mathfrak{c}'_1$ as plaintext elements modulo 2^{e+1}, and compute homomorphically the inner-product $\mathfrak{u} \leftarrow \mathfrak{c}'_1 \cdot \mathfrak{s} + \mathfrak{c}'_0 \pmod{2^{e+1}}$ by setting

$$\tilde{\mathsf{ct}}' \leftarrow \mathfrak{c}'_1 \cdot \tilde{\mathsf{ct}} + (\mathfrak{c}'_0, 0).$$

This means that $\tilde{\mathsf{ct}}'$ encrypts the desired $\mathfrak{u}$, and to complete the recryption procedure we just need to extract and XOR the top and bottom bits from all the coefficients in $\mathfrak{u}$, thus getting an encryption of (the coefficients of) the plaintext $\mathfrak{m}$. This calculation is the most expensive part of recryption, and it is done in three steps:

Linear transformation. First apply homomorphically a $\mathbb{Z}_{2^{e+1}}$-linear transformation to $\tilde{\mathsf{ct}}'$, converting it into ciphertexts that have the coefficients of $\mathfrak{u}$ in the plaintext slots.

Bit extraction. Next apply a homomorphic (non-linear) bit-extraction procedure, computing two ciphertexts that contain the top and bottom bits (respectively) of the integers stored in the slots. A side-effect of the bit-extraction computation is that the plaintext space is reduced from mod-2^{e+1} to mod-2, so adding the two ciphertexts we get a ciphertext whose slots contain the coefficients of $\mathfrak{m}$ relative to a mod-2 plaintext space.

[2]This is a slight simplification, the actual formula for $p = 2$ is $\mathfrak{m} \leftarrow \mathfrak{u}\langle e\rangle \oplus \mathfrak{u}\langle e-1\rangle \oplus \mathfrak{u}\langle 0\rangle$, see Lemma 5.1.

Inverse linear transformation. Finally apply homomorphically the inverse linear transformation (this time over $\mathbb{Z}_2$), obtaining a ciphertext ct^* that encrypts the plaintext element $\mathfrak{m}$.

An optimization. The deepest part of recryption is bit-extraction, and its complexity — both time and depth — increases with the most-significant extracted bit (i.e., with e). The parameter e can be made somewhat smaller by choosing a smaller $\tilde{q} = 2^e + 1$, but for various reasons $\tilde{q}$ cannot be too small, so Gentry et al. described in [21] an optimization for reducing the top extracted bit without reducing $\tilde{q}$.

After modulus-switching to the ciphertext ct, we can add multiples of $\tilde{q}$ to the coefficients of $\mathfrak{c}_0', \mathfrak{c}_1'$ to make them divisible by $2^{e'}$ for some moderate-size $e' < e$. Let $\mathsf{ct}'' = (\mathfrak{c}_0'', \mathfrak{c}_1'')$ be the resulting ciphertext, clearly $[\langle \mathsf{sk}, \mathsf{ct}' \rangle]_{\tilde{q}} = [\langle \mathsf{sk}, \mathsf{ct}'' \rangle]_{\tilde{q}}$ so ct'' still encrypts the same plaintext $\mathfrak{m}$. Moreover, as long as the coefficients of ct'' are sufficiently smaller than $\tilde{q}^2$, we can still use the same simplified decryption formula $\mathfrak{u}' \leftarrow [\langle \mathsf{sk}, \mathsf{ct}'' \rangle]_{2^{e+1}}$ and $\mathfrak{m} \leftarrow \mathfrak{u}'\langle e \rangle \oplus \mathfrak{u}'\langle 0 \rangle$.

However, since ct'' is divisible by $2^{e'}$ then so is $\mathfrak{u}'$. For one thing this means that $\mathfrak{u}'\langle 0 \rangle = 0$ so the decryption procedure can be simplified to $\mathfrak{m} \leftarrow \mathfrak{u}'\langle e \rangle$. But more importantly, we can divide ct'' by $2^{e'}$ and compute instead $\mathfrak{u}'' \leftarrow [\langle \mathsf{sk}, \mathsf{ct}''/2^{e'} \rangle]_{2^{e-e'+1}}$ and $\mathfrak{m} \leftarrow \mathfrak{u}'\langle e - e' \rangle$. This means that the encryption of $\mathfrak{s}$ in the public key can be done relative to plaintext space $2^{e-e'}$ and we only need to extract $e - e'$ bits rather than e.

3.2 Our Recryption Procedure

We optimize the GHS recryption procedure and extend it to handle plaintext spaces modulo arbitrary prime powers p^r rather than just $p = 2$, $r = 1$. The high-level structure of the procedure remains roughly the same.

To reduce the complexity as much as we can, we use a special recryption key $\tilde{\mathsf{sk}} = (1, \tilde{\mathfrak{s}})$, which is chosen as sparse as possible (subject to security requirements). As we elaborate in Section 6, the number of nonzero coefficients in $\tilde{\mathfrak{s}}$ plays an extremely important role in the complexity of recryption.

To enable recryption of mod-p^r ciphertexts, we include in the public key a ciphertext $\tilde{\mathsf{ct}}$ that encrypts the secret key $\tilde{\mathfrak{s}}$ relative to a large modulus Q and plaintext space mod-p^{e+r} for some $e > r$. Then given a mod-p^r ciphertext ct to recrypt, we perform the following steps:

Modulus-switching. Convert ct into another ct' relative to the special modulus $\tilde{q} = p^e + 1$. We prove in Lemma 5.1 that for the special modulus $\tilde{q}$, the decryption procedure can be evaluated by computing $\mathfrak{u} \leftarrow [\langle \mathsf{sk}, \mathsf{ct}' \rangle]_{p^{e+r}}$ and then $\mathfrak{m} \leftarrow \mathfrak{u}\langle r-1, \ldots, 0 \rangle_p - \mathfrak{u}\langle e+r-1, \ldots, e \rangle_p \pmod{p^r}$.

Optimization. Add multiples of $\tilde{q}$ to the coefficients of ct', making them divisible by $p^{e'}$ for some $r \le e' < e$ without increasing them too much. This is described in Section 5.2. The resulting ciphertext, which is divisible by $p^{e'}$, is denoted $\mathsf{ct}'' = (\mathfrak{c}_0'', \mathfrak{c}_1'')$. It follows from the same reasoning as above that we can now compute $\mathfrak{u}' \leftarrow [\langle \mathsf{sk}, \mathsf{ct}''/p^{e'} \rangle]_{p^{e-e'+r}}$ and then $\mathfrak{m} \leftarrow -\mathfrak{u}'\langle e-e'+r-1, \ldots, e-e' \rangle_p$ (mod p^r).

Multiply by encrypted key. Evaluate homomorphically the inner product (divided by $p^{e'}$), $\mathfrak{u}' \leftarrow (\mathfrak{c}_1' \cdot \mathfrak{s} + \mathfrak{c}_0')/p^{e'} \pmod{p^{e-e'+r}}$, by setting $\tilde{\mathsf{ct}}' \leftarrow (\mathfrak{c}_1'/p^{e'}) \cdot \tilde{\mathsf{ct}} + (\mathfrak{c}_0'/p^{e'}, 0)$. The plaintext space of the resulting $\tilde{\mathsf{ct}}'$ is modulo $p^{e-e'+r}$.

Note that since we only use plaintext space modulo $p^{e-e'+r}$, then we might as well use the same plaintext space also for $\tilde{\mathsf{ct}}$, rather than encrypting it relative to plaintext space modulo p^{e+r} as described above.

Linear transformation. Apply homomorphically a $\mathbb{Z}_{p^{e-e'+r}}$-linear transformation to $\tilde{\mathsf{ct}}'$, converting it into ciphertexts that have the coefficients of $\mathfrak{u}'$ in the plaintext slots. This linear transformation, which is the most intricate part of the implementation, is described in Section 4. It uses a tensor decomposition similar to [1] to reduce complexity, but pays much closer attention to details such as the mult-by-constant depth and data movements.

Digit extraction. Apply a homomorphic (non-linear) digit-extraction procedure, computing r ciphertexts that contain the digits $e-e'+r-1$ through $e-e'$ of the integers in the slots, respectively, relative to plaintext space mod-p^r. This requires that we generalize the bit-extraction procedure from [21] to a digit-extraction procedure for any prime power $p^r \geq 2$, this is done in Section 5.3. Once we extracted all these digits, we can combine them to get an encryption of the coefficients of m in the slots relative to plaintext space modulo p^r.

Inverse linear transformation. Finally apply homomorphically the inverse linear transformation, this time over $\mathbb{Z}_{p^r}$, converting the ciphertext into an encryption ct^* of the plaintext element $\mathfrak{m}$ itself. This too is described in Section 4.

Thin bootstrapping. For "thin bootstrapping", where each slot contains just an integer, we use the Chen-Han procedure [5], which has a somewhat different structure. See Section 7.1 for a brief description.

4 The Linear Transformations

In this section we describe the linear transformations that we apply during the recryption procedure to map the plaintext coefficients into the slots and back. Central to our implementation is imposing a hypercube structure on the plaintext space $R_{p^r} = \mathbb{Z}_{p^r}[X]/(\Phi_m(X))$ with one dimension per factor of m, and implementing the second (inverse) transformation as a sequence of multi-point polynomial-evaluation operations, one for each dimension of the hypercube. We begin with some additional background.

4.1 Algebraic Background

Let m denote the parameter defining the underlying cyclotomic ring in an instance of the BGV cryptosystem with native plaintext space $R_{p^r} = \mathbb{Z}_{p^r}[X]/(\Phi_m(X))$. Throughout this section, we consider a particular factorization $m = m_1 \cdots m_t$, where the m_i's are pairwise co-prime positive integers. We write $\mathrm{CRT}(h_1, \ldots, h_t)$ (with $h_i \in \{0, \ldots, m_i - 1\}$) for the unique element $h \in \{0, \ldots, m-1\}$ satisfying $h \equiv h_i \pmod{m_i}$ $(i = 1, \ldots, t)$ for all $i = 1, \ldots, t$.

Lemma 4.1 *Let p, m and the m_i's be as above, where p is a prime not dividing any of the m_i's. Let d_1 be the order of p modulo m_1 and for $i = 2, \ldots, t$ let d_i be the order of $p^{d_1 \cdots d_{i-1}}$ modulo m_i. Then the order of p modulo m is $d \stackrel{\text{def}}{=} d_1 \cdots d_t$.*

Moreover, suppose that $S_1, \ldots, S_t$ are sets of integers such that each $S_i \subseteq \{0, \ldots, m_i - 1\}$ forms a complete system of representatives for $\mathbb{Z}^*_{m_i}/\langle p^{d_1 \cdots d_{i-1}} \rangle$. Then the set $S \stackrel{\text{def}}{=} \mathrm{CRT}(S_1, \ldots, S_t)$ forms a complete system of representatives for $\mathbb{Z}^*_m/\langle p \rangle$.

Proof. It suffices to prove the lemma for $t = 2$. The general case follows by induction on t.

The fact that the order of p modulo $m \stackrel{\text{def}}{=} m_1 m_2$ is $d \stackrel{\text{def}}{=} d_1 d_2$ is clear by definition. The cardinality of S_1 is $\phi(m_1)/d_1$ and of S_2 is $\phi(m_2)/d_2$, and so the cardinality of S is $\phi(m_1)\phi(m_2)/d_1 d_2 = \phi(m)/d = |\mathbb{Z}^*_m/\langle p \rangle|$. So it suffices to show that distinct elements of S belong to distinct cosets of $\langle p \rangle$ in $\mathbb{Z}^*_m$.

To this end, let $a, b \in S$, and assume that $p^f a \equiv b \pmod{m}$ for some nonnegative integer f. We want to show that $a = b$. Now, since the congruence $p^f a \equiv b$ holds modulo m, it holds modulo m_1 as well, and by the defining property of S_1 and the construction of S, we must have $a \equiv b \pmod{m_1}$. So we may cancel a and b from both sides of the congruence $p^f a \equiv b \pmod{m_1}$, obtaining $p^f \equiv 1 \pmod{m_1}$, and from the defining property of d_1, we must have $d_1 \mid f$. Again, since the congruence $p^f a \equiv b$ holds modulo m, it holds modulo m_2 as well, and since $d_1 \mid f$, by the defining property of S_2 and the construction of S, we must have $a \equiv b \pmod{m_2}$. It follows that $a \equiv b \pmod{m}$, and hence $a = b$. □

The powerful basis. The linear transformations in our recryption procedure make use of the same tensor decomposition that was used by Alperin-Sheriff and Peikert in [1], which in turn relies on the "powerful basis" representation of the plaintext space, due to Lyubashevsky et al. [32, 33]. The "powerful basis" representation is an isomorphism

$$R_{p^r} = \mathbb{Z}[X]/(p^r, \Phi_m(X)) \;\cong\; R'_{p^r} \stackrel{\text{def}}{=} \mathbb{Z}[X_1, \ldots, X_t]/(p^r, \Phi_{m_1}(X_1), \ldots, \Phi_{m_t}(X_t)),$$

defined explicitly by the map

$$\mathsf{PowToPoly} : f(X_1, \ldots, X_t) \to f(X^{m/m_1}, \ldots, X^{m/m_t}),$$

namely $\mathsf{PowToPoly} : R'_{p^r} \to R_{p^r}$ sends (the residue class of) X_i to (the residue class of) X^{m/m_i}.

Recall that we view an element in the native plaintext space R_{p^r} as encoding a vector of plaintext slots from E, where E is an extension ring of $\mathbb{Z}_{p^r}$ that contains a principal mth root of unity ζ. Below let us define $\zeta_i \stackrel{\text{def}}{=} \zeta^{m/m_i}$ for $i = 1, \ldots, t$. It follows from the definitions above that for $h = \mathrm{CRT}(h_1, \ldots, h_t)$ and $\alpha = \mathsf{PowToPoly}(\alpha')$, we have $\alpha(\zeta^h) = \alpha'(\zeta_1^{h_1}, \ldots, \zeta_t^{h_t})$.

Using Lemma 4.1, we can generalize the above to multi-point evaluation. Let $S_1, \ldots, S_t$ and S be sets as defined in the lemma. Then evaluating an element $\alpha' \in R'_{p^r}$ at all points $(\zeta_1^{h_1}, \ldots, \zeta_t^{h_t})$, where $(h_1, \ldots, h_t)$ ranges over $S_1 \times \cdots \times S_t$, is equivalent to evaluating the corresponding element in $\alpha \in R_{p^r}$ at all points ζ^h, where h ranges over S.

4.2 The Evaluation Map

With the background above, we can now describe our implementation of the linear transformations. Recall that these transformations are needed to map the coefficients of the plaintext into the slots and back. Importantly, it is the powerful basis coefficients that we put in the slots during the first linear transformation, and take from the slots in the second transformation.

Since the two linear transformations are inverses of each other (except modulo different powers of p), then once we have an implementation of one we also get an implementation of the other. For didactic reasons we begin by describing in detail the second transformation, and later we explain how to get from it also the implementation of the first transformation.

The second transformation begins with a plaintext element β that contains in its slots the powerful-basis coefficients of some other element α, and ends with the element α itself. Important to our implementation is the view of this transformation as *multi-point evaluation* of a polynomial. Namely, the second transformation begins with an element β whose slots contain the coefficients of the powerful basis $\alpha' = \mathsf{PowToPoly}(\alpha)$, and ends with the element α that holds in the slots the values

$$\alpha(\zeta^h) = \alpha'(\zeta_1^{h_1}, \ldots, \zeta_t^{h_t})$$

where the h_i's range over the S_i's from Lemma 4.1 and correspondingly h range over S. Crucial to this view is that the CRT set S from Lemma 4.1 is the same as the representative-set S from Eqn. (2) that determines the plaintext slots.

Choosing the representatives. Our first order of business is therefore to match up the sets S from Eqn. (2) and Lemma 4.1. To facilitate this (and also other aspects of our implementation), we place some constraints on our choice of the parameter m and its factorization.[3] Recall that we consider the factorization $m = m_1 \cdots m_t$, and denote by d_i the order of $p^{d_1 \cdots d_{i-1}}$ modulo m_i.

I. In choosing m and the m_i's we restrict ourselves to the case where each group $\mathbb{Z}^*_{m_i}/\langle p^{d_1 \cdots d_{i-1}} \rangle$ is cyclic of order k_i, and let its generator be denoted by (the residue class of) $\tilde{g}_i \in \{0, \ldots, m_i - 1\}$. Then for $i = 1, \ldots, t$, we set $S_i \stackrel{\text{def}}{=} \{\tilde{g}_i^e \bmod m_i : 0 \le e < k_i\}$.

   We define $g_i \stackrel{\text{def}}{=} \mathrm{CRT}(1, \ldots, 1, \tilde{g}_i, 1, \ldots, 1)$ (with $\tilde{g}_i$ in the ith position), and use the g_i's as our hypercube basis with the order of g_i set to k_i. In this setting, the set S from Lemma 4.1 coincides with the set S in Eqn. (2); that is, we have $S = \left\{\prod_{i=1}^{t} g_i^{e_i} \bmod m : 0 \le e_i < k_i\right\} = \mathrm{CRT}(S_1, \ldots, S_t)$.

II. We further restrict ourselves to only use factorizations $m = m_1 \cdots m_t$ for which $d_1 = d$. (That is, the order of p is the same in $\mathbb{Z}^*_{m_1}$ as in $\mathbb{Z}^*_m$.) With this assumption, we have $d_2 = \cdots = d_t = 1$, and moreover $k_1 = \phi(m_1)/d$ and $k_i = \phi(m_i)$ for $i = 2, \ldots, t$.

Note that with the above assumptions, the first dimension could be either good or bad, but the other dimensions $2, \ldots, t$ are always good. This is because $p^{d_1 \cdots d_{i-1}} \equiv 1 \pmod{m}$, so also $p^{d_1 \cdots d_{i-1}} \equiv 1 \pmod{m_i}$, and therefore $\mathbb{Z}^*_{m_i}/\langle p^{d_1 \cdots d_{i-1}} \rangle = \mathbb{Z}^*_{m_i}$, which means that the order of g_i in $\mathbb{Z}^*_m$ (which is the same as the order of $\tilde{g}_i$ in $\mathbb{Z}^*_{m_i}$) equals k_i.

Packing the coefficients. In designing the linear transformation, we have the freedom to choose how we want the coefficients of α' to be packed in the slots of β. Let us denote these coefficients by $c_{j_1,\ldots,j_t}$ where each index j_i runs over $\{0, \ldots, \phi(m_i) - 1\}$, and each $c_{j_1,\ldots,j_t}$ is in $\mathbb{Z}_{p^r}$. That is, we have

$$\alpha'(X_1, \ldots, X_t) = \sum_{j_1, j_2, \ldots, j_t} c_{j_1,\ldots,j_t} X_1^{j_1} X_2^{j_2} \cdots X_t^{j_t} = \sum_{j_2,\ldots,j_t} \left(\sum_{j_1} c_{j_1,\ldots,j_t} X_1^{j_1} \right) X_2^{j_2} \cdots X_t^{j_t}.$$

Recall that we can pack d coefficients into a slot, so for fixed $j_2, \ldots, j_t$, we can pack the $\phi(m_1)$ coefficients of the polynomial $\sum_{j_1} c_{j_1,\ldots,j_t} X_1^{j_1}$ into $k_1 = \phi(m_1)/d$ slots. In our implementation we pack these coefficients into the slots indexed by $(e_1, j_2, \ldots, j_t)$, for $e_1 = 0, \ldots, k_1 - 1$. That is, we pack them into a single hypercolumn in dimension 1.

4.2.1 The Eval Transformation

The second (inverse) linear transformation of the recryption procedure beings with the element β whose slots pack the coefficients $c_{j_1,\ldots,j_t}$ as above. The desired output from this transformation is the element whose slots contain $\alpha(\zeta^h)$ for all $h \in S$ (namely the element α itself). Specifically, we need each slot of α with hypercube index $(e_1, \ldots, e_t)$ to hold the value

$$\alpha'\left(\zeta_1^{g_1^{e_1}}, \ldots, \zeta_t^{g_t^{e_t}}\right) = \alpha\left(\zeta^{g_1^{e_1} \cdots g_t^{e_t}}\right).$$

Below we denote $\zeta_{i,e_i} \stackrel{\text{def}}{=} \zeta_i^{g_i^{e_i}}$. We transform β into α in t stages, each of which can be viewed as multi-point evaluation of polynomials along one dimension of the hypercube.

[3] As we discuss in Section 7, there are still sufficiently many settings that satisfy these requirements.

Stage 1. This stage begins with the element β, in which each dimension-1 hypercolumn with index $(\star, j_2, \ldots, j_t)$ contains the coefficients of the univariate polynomial $P_{j_2,\ldots,j_t}(X_1) \stackrel{\text{def}}{=} \sum_{j_1} c_{j_1,\ldots,j_t} X_1^{j_1}$. We transform β into β_1 where that hypercolumn contains the evaluation of the same polynomial in many points. Specifically, the slot of β_1 indexed by $(e_1, j_2, \ldots, j_t)$ contains the value $P_{j_2,\ldots,j_t}(\zeta_{1,e_1})$.

By definition, this stage consists of parallel application of a particular $\mathbb{Z}_{p^r}$-linear transformation M_1 (namely a multi-point polynomial evaluation map) to each of the k/k_1 hypercolumns in dimension 1. In other words, M_1 maps $(k_1 \cdot d)$-dimensional vectors over $\mathbb{Z}_{p^r}$ (each packed into k_1 slots) to k_1-dimensional vectors over E. We elaborate on the efficient implementation of this stage later in this section.

Stages $2, \ldots, t$. The element β_1 from the previous stage holds in its slots the coefficients of the k_1 multivariate polynomials $A_{e_1}(\cdot)$ (for $e_1 = 0, \ldots, k_1 - 1$),

$$A_{e_1}(X_2, \ldots, X_t) \stackrel{\text{def}}{=} \alpha'(\zeta_{1,e_1}, X_2, \ldots, X_t) = \sum_{j_2,\ldots,j_t} \underbrace{\Big(\sum_{j_1} c_{j_1,\ldots,j_t} \zeta_{1,e_1}^{j_1}\Big)}_{\text{slot } (e_1,j_2,\ldots,j_t) = P_{j_2,\ldots,j_t}(\zeta_{1,e_1})} \cdot X_2^{j_2} \cdots X_t^{j_t}.$$

The goal in the remaining stages is to implement multi-point evaluation of these polynomials at all the points $X_i = \zeta_{i,e_i}$ for $0 \le e_i < k_i$. Note that differently from the polynomial α' that we started with, the polynomials A_{e_1} have coefficients from E (rather than from $\mathbb{Z}_{p^r}$), and these coefficients are encoded one per slot (rather than d per slot). As we explain later, this makes it easier to implement the desired multi-point evaluation. Separating out the second dimension we can write

$$A_{e_1}(X_2, \ldots, X_t) = \sum_{j_3,\ldots,j_t} \left(\sum_{j_2} P_{j_2,\ldots,j_t}(\zeta_{1,e_1}) X_2^{j_2} \right) X_3^{j_3} \cdots X_t^{j_t}.$$

We note that each dimension-2 hypercolumn in β_1 with index $(e_1, \star, j_3, \ldots, j_t)$ contains the E-coefficients of the univariate polynomial $Q_{e_1,j_3,\ldots,j_t}(X_2) \stackrel{\text{def}}{=} \sum_{j_2} P_{j_2,\ldots,j_t}(\zeta_{1,e_1}) X_2^{j_2}$. In Stage 2, we transform β_1 into β_2 where that hypercolumn contains the evaluation of the same polynomial in many points. Specifically, the slot of β_2 indexed by $(e_1, e_2, j_3 \ldots, j_t)$ contains the value

$$Q_{e_1,j_3,\ldots,j_t}(\zeta_{2,e_2}) \;=\; \sum_{j_2} P_{j_2,\ldots,j_t}(\zeta_{1,e_1}) \cdot \zeta_{2,e_2}^{j_2} \;=\; \sum_{j_1,j_2} c_{j_1,\ldots,j_t} \zeta_{1,e_1}^{j_1} \zeta_{2,e_2}^{j_2},$$

and the following stages implement the multi-point evaluation of these polynomials at all the points $X_i = \zeta_{i,e_i}$ for $0 \le e_i < k_i$.

Stages $s = 3, \ldots, t$ proceed analogously to Stage 2, each time eliminating a single variable X_s via the parallel application of an E-linear map M_s to each of the k/k_s hypercolumns in dimension s. When all of these stages are completed, we have in every slot with index $(e_1, \ldots, e_t)$ the value $\alpha'(\zeta_{1,e_1}, \ldots, \zeta_{t,e_t})$, as needed.

Implementation and complexity of Eval. The linear transformations M_s for $s = 1, \ldots, t$ are implemented using the MatMul1D and BlockMatMul1D routines that are implemented in `HElib`, and which are described in detail in [25].

The linear transformation M_1 is implemented using the BlockMatMul1D routine. The running time of this routine depends on a number of factors, but will typically be dominated by the running time of at most $c_1\sqrt{\phi(m_1)} + O(1)$ automorphism operations and $c_2\phi(m_1)$ constant-ciphertext multiplications, where $c_1 \in [1, 3]$ and $c_2 \in [1, 2]$. The depth of the computation is one constant-ciphertext multiplication.

For $s = 2, \ldots, t$, the linear transformation M_s is implemented using the $\mathsf{MatMul1D}$ routine. Again, the running time of this routine depends on a number of factors, but will typically be dominated by the running time of at most $\sqrt{\phi(m_s)}+O(1)$ automorphism operations and $\phi(m_s)$ constant-ciphertext multiplications. The depth of the computation is one constant-ciphertext multiplication.

The total running time of Eval will be dominated by the running time of at most

$$c_1\sqrt{\phi(m_1)} + \sqrt{\phi(m_2)} + \cdots + \sqrt{\phi(m_t)} + O(t)$$

automorphism operations and

$$c_2\phi(m_1) + \phi(m_2) + \cdots + \phi(m_t).$$

constant-ciphertext multiplications. The depth of the Eval computation is t.

4.2.2 The Transformation Eval^{-1}

The first linear transformation in the recryption procedure is the inverse of Eval. This transformation can be implemented by simply running the above stages in reverse order and using the inverse linear maps M_s^{-1} in place of M_s. The complexity estimates are identical.

4.3 Unpacking and Repacking the Slots

In our recryption procedure we have the non-linear digit extraction routine “sandwiched” between the linear evaluation map and its inverse. However the evaluation map transformations from above maintain fully-packed ciphertexts, where each slot contains an element of the extension ring E (of degree d), while our digit extraction routine needs “sparsely packed” slots containing only integers from $\mathbb{Z}_{p^r}$.

Therefore, before we can use the digit extraction procedure we need to “unpack” the slots, so as to get d ciphertexts in which each slot contains a single coefficient in the constant term. Similarly, after digit extraction we have to “repack” the slots, before running the second transformation.

Unpacking. Consider the unpacking procedure in terms of the element $\beta \in R_{p^r}$. Each slot of β contains an element of E which we write as $\sum_{i=0}^{d-1} a_i\zeta^i$ with the a_i's in $\mathbb{Z}_{p^r}$. We want to compute $\beta^{(0)}, \ldots, \beta^{(d-1)}$, so that the corresponding slot of each $\beta^{(i)}$ contains a_i. To obtain $\beta^{(i)}$, we need to apply to each slot of β the $\mathbb{Z}_{p^r}$-linear map $L_i : E \to \mathbb{Z}_{p^r}$ that maps $\sum_{i=0}^{d-1} a_i\zeta^i$ to a_i.

Using linearized polynomials, as discussed in Section 2.4, we may write $\beta^{(i)} = \sum_{f=0}^{d-1} \kappa_{i,f}\sigma^f(\beta)$, for constants $\kappa_{i,f} \in R_{p^r}$. Given an encryption of β, we can compute encryptions of all of the $\sigma^f(\beta)$'s and then take linear combinations of these to get encryptions of all of the $\beta^{(i)}$'s. This takes the time of $d-1$ automorphisms and d^2 constant-ciphertext multiplications, and a depth of one constant-ciphertext multiplication.

While the cost in time of constant-ciphertext multiplications is relatively cheap, it cannot be ignored, especially as we have to compute d^2 of them. In our implementation, the cost is dominated the time it takes to convert an element in R_{p^r} to its corresponding DoubleCRT representation. It is possible, of course, to precompute and store all d^2 of these constants in DoubleCRT format, but the space requirement is significant: for typical parameters, our implementation takes about 4MB to store a single constant in DoubleCRT format, so for example with $d = 24$, these constants take up almost 2.5GB of space.

This unappealing space/time trade-off can be improved considerably using somewhat more sophisticated implementations. Suppose that in the first linear transformation Eval^{-1}, instead of packing the coefficients $a_0, \ldots, a_{d-1}$ into a slot as $\sum_i a_i\zeta^i$, we pack them as $\sum_i a_i\sigma_E^i(\theta)$, where $\theta \in E$ is a normal element. Further, let $L'_0 : E \to \mathbb{Z}_{p^r}$ be the $\mathbb{Z}_{p^r}$-linear map that sends $\eta = \sum_i a_i \sigma_E^i(\theta)$ to a_0. Then we have $L'_0(\sigma^{-j}(\eta)) = a_j$ for $j = 0, \ldots, d-1$. If we realize the map L'_0 with linearized polynomials, and if the plaintext γ has the coefficients packed into slots via a normal element as above, then we have $\beta^{(i)} = \sum_{f=0}^{d-1} \kappa_f \cdot \sigma^{f-i}(\gamma)$, where the κ_f's are constants in R_{p^r}. So we have only d constants rather than d^2.

To use this strategy, however, we must address the issue of how to modify the Eval transformation so that Eval^{-1} will give us the plaintext element γ that packs coefficients as $\sum_i a_i \sigma_E^i(\theta)$. As it turns out, in our implementation this modification is for free: recall that the unpacking transformation immediately follows the last stage of the inverse evaluation map Eval^{-1}, and that last stage applies $\mathbb{Z}_{p^r}$-linear maps to the slots; therefore, we simply fold into these maps the $\mathbb{Z}_{p^r}$-linear map that takes $\sum_i a_i \zeta^i$ to $\sum_i a_i \sigma_E^i(\theta)$ in each slot.

It is possible to reduce the number of stored constants even further: since L'_0 is a map from E to the base ring $\mathbb{Z}_{p^r}$, then the κ_f's are related via $\kappa_f = \sigma^f(\kappa_0)$. Therefore, we can obtain all of the DoubleCRTs for the κ_f's by computing just one for κ_0 and then applying the Frobenius automorphisms directly to the DoubleCRT for κ_0. We note, however, that applying these automorphisms directly to DoubleCRTs leads to a slight increase in the noise of the homomorphic computation. We did not use this last optimization in our implementation.

Repacking. Finally, we discuss the reverse transformation, which repacks the slots, taking $\beta^{(0)}, \ldots, \beta^{(d-1)}$ to β. This is quite straightforward: if $\bar{\zeta}$ is the plaintext element with ζ in each slot, then $\beta = \sum_{i=0}^{d-1} \bar{\zeta}^i \beta^{(i)}$. This formula can be evaluated homomorphically with a cost in time of d constant-ciphertext multiplications, and a cost in depth one constant-ciphertext multiplication.

5 Recryption with Plaintext Space Modulo $p > 2$

Below we extend the treatment from [21, 1] to handle plaintext spaces modulo $p > 2$. In Sections 5.1 through 5.3 we generalize the various lemmas to $p > 2$. In Section 5.4 we explain how these lemmas are put together in the recryption procedure. In Section 6 we discuss the choice of parameters.

5.1 Simpler Decryption Formula

We begin by extending the simplified decryption formula [21, Lemma 1] from plaintext space mod-2 to any prime-power p^r. Recall that we denote by $[z]_q$ the mod-q reduction into $[-q/2, q/2)$ (except when $q = 2$ we reduce to $(-1, 1]$). Also $z\langle j, \ldots, i\rangle_p$ denotes the integer whose mod-p expansion consists of digits i through j in the mod-p expansion of z (and we omit the p subscript if it is clear from the context).

Lemma 5.1 *Let $p > 1$, $e > r \geq 1$, and $q = p^e + 1$ be integers. Also let z be an integer such that both z/q and $[z]_q$ are sufficiently smaller than q in magnitude, specifically $|z/q| + |[z]_q| \leq (q-1)/2$.*

- *If p is odd then $[z]_q = z\langle r-1, \ldots, 0\rangle - z\langle e+r-1, \ldots, e\rangle \pmod{p^r}$.*
- *If $p = 2$ then $[z]_q = z\langle r-1, \ldots, 0\rangle - z\langle e+r-1, \ldots, e\rangle - z\langle e-1\rangle \pmod{2^r}$.*

Proof. We begin with the odd-p case. Denote $z_0 = [z]_q$, then $z = z_0 + kq$ (or in other words $k = (z - z_0)/q$). Denoting $w = z_0 + k$, we therefore have

$$|w| = |z_0(1 - 1/q) + z/q| < |z_0| + |z/q| \leq (q-1)/2 = p^e/2. \tag{3}$$

This means that the mod-p representation of w has only 0's in positions e and up. Writing

$$z = z_0 + k(p^e + 1) = z_0 + k + p^e k = w + p^e k, \tag{4}$$

we conclude that the digits $e, e+1, \ldots$ in z are the same as the digits $0, 1, \ldots$ in k (since no carry digits are generated by w). Namely $k\langle r-1, \ldots, 0\rangle = z\langle e+r-1, \ldots, e\rangle$. On the other hand, we have $z_0 = z - k - p^e k = z - k \pmod{p^r}$, so it follows that

$$z_0\langle r-1, \ldots, 0\rangle = z\langle r-1, \ldots, 0\rangle - k\langle r-1, \ldots, 0\rangle = z\langle r-1, \ldots, 0\rangle - z\langle e+r-1, \ldots, e\rangle \pmod{p^r}.$$

The proof for the $p = 2$ case is similar, but we no longer have the guarantee that the high-order bits of the sum $w = z_0 + k$ are all zero. From Eqn. (4) we can still deduce that $z\langle e-1\rangle = w\langle e-1\rangle$ and

$$z\langle e+r-1, \ldots, e\rangle = w\langle e+r-1, \ldots, e\rangle + k\langle r-1, \ldots, 0\rangle \pmod{2^r}.$$

Since $|w| < 2^{e-1}$, then the bits in positions $e-1$ and up in w are either all zero if $w \geq 0$, or all one if $w < 0$. In particular, this means that

$$w\langle e+r-1, \ldots, e\rangle \;=\; \left\{ \begin{array}{rl} 0 & \text{if } w \geq 0 \\ -1 & \text{if } w < 0 \end{array} \right\} \;=\; -\,w\langle e-1\rangle \;=\; -\,z\langle e-1\rangle \pmod{2^r}.$$

Concluding, we therefore have

$$\begin{aligned} z_0\langle r-1, \ldots, 0\rangle &= z\langle r-1, \ldots, 0\rangle - k\langle r-1, \ldots, 0\rangle \\ &= z\langle r-1, \ldots, 0\rangle - \big(z\langle e+r-1, \ldots, e\rangle - w\langle e+r-1, \ldots, e\rangle\big) \\ &= z\langle r-1, \ldots, 0\rangle - z\langle e+r-1, \ldots, e\rangle - z\langle e-1\rangle \pmod{2^r}. \quad \square \end{aligned}$$

Remark. Lemma 5.1 improves upon the corresponding lemma (also Lemma 5.1) in our report from Eurocrypt 2015 [24]. In that lemma, instead of $|z/q| + |[z]_q| \leq (q-1)/2$, we had a pair of inequalities $|z/q| \leq q/4 - 1$ and $|[z]_q| \leq q/4$.

5.2 Making an Integer Divisible By $p^{e'}$

As sketched in Section 3, we use the following lemma to reduce the number of digits that needs to be extracted, hence reducing the time and depth of the digit-extraction step.

Lemma 5.2 *Let $e' \geq 1$ and $q > p > 1$ be integers such that $q \equiv 1 \pmod{p^{e'}}$. Then for every integer z there exist an integer v such that $|v| \leq p^{e'}/2$, such that*

$$z + v \cdot q \equiv 0 \pmod{p^{e'}}.$$

Proof. Let $v = -[z]_{p^{e'}}$, so $|v| \leq p^{e'}/2$. Moreover, since $q \equiv 1 \pmod{p^{e'}}$, we have

$$0 \equiv z + v \equiv z + v \cdot q \pmod{p^{e'}}. \quad \square$$

Remark. Lemma 5.2 is much simpler than the corresponding lemma (also Lemma 5.2) in our report from Eurocrypt 2015 [24]. In that lemma, we added both multiples of q and of p^r to z to make it divisible by $p^{e'}$. However, because of the improved Lemma 5.1, this is no longer helpful. Moreover, in the analysis in Section 6, adding multiples of p^r leads to somewhat worse bounds on error probabilities.

Discussion. Recall that in our recryption procedure we have a ciphertext ct that encrypts some $\mathfrak{m}$ with respect to modulus q and plaintext space mod-p^r, and we use the lemma above to convert it into another ciphertext ct' that encrypts the same thing but is divisible by $p^{e'}$, and by doing so we need to extract e' fewer digits in the digit-extraction step.

Considering the elements $\mathfrak{u} \leftarrow \langle \mathsf{sk}, \mathsf{ct} \rangle$ and $\mathfrak{u}' \leftarrow \langle \mathsf{sk}, \mathsf{ct}' \rangle$ (without any modular reduction), since sk is integral then adding multiples of q to the coefficients of ct does not change $[\mathfrak{u}]_q$, and so ct and ct' still encrypt the same plaintext. However in our recryption procedure we need more: to use our simpler decryption formula from Lemma 5.1, we need to ensure that $|\mathfrak{u}'/q| + |[\mathfrak{u}']_q| \le (q-1)/2$, where $|\cdot|$ denotes the ℓ_∞-norm on the powerful basis.

5.3 Digit-Extraction for Plaintext Space Modulo p^r

The bit-extraction procedure that was described by Gentry et al. in [21] and further optimized by Alperin-Sheriff and Peikert in [1] is specific for the case $p = 2^e$. Namely, for an input ciphertext relative to mod-2^e plaintext space, encrypting some integer z (in one of the slots), this procedure computes the ith top bit of z (in the same slot), relative to plaintext space mod-2^{e-i+1}. Below we show how to extend this bit-extraction procedure to a digit-extraction also when p is an odd prime.

The main observation underlying the original bit-extraction procedure, is that squaring an integer keeps the least-significant bit unchanged but inserts zeros in the higher-order bits. Namely, if b is the least significant bit of the integer z and moreover $z = b \pmod{2^e}$, $e \ge 1$, then squaring z we get $z^2 = b \pmod{2^{e+1}}$. Therefore, $z - z^2$ is divisible by 2^e, and the LSB of $(z-z^2)/2^e$ is the eth bit of z.

Unfortunately the same does not hold when using a base $p > 2$. Instead, we show below that for any exponent e *there exists some degree-p polynomial* $F_e(\cdot)$ (but not necessarily $F_e(X) = X^p$) such that when $z = z_0 \pmod{p^e}$ then $F_e(z) = z_0 \pmod{p^{e+1}}$. Hence $z - F_e(z)$ is divisible by p^e, and the least-significant digit of $(z - F_e(z))/p^e$ is the eth digit of z. The existence of such polynomial $F_e(X)$ follows from the simple derivation below.

Lemma 5.3 *For every prime p and exponent $e \ge 1$, and every integer z of the form $z = z_0 + p^e z_1$ (with z_0, z_1 integers, $z_0 \in [p]$), it holds that $z^p = z_0 \pmod{p}$, and $z^p = z_0^p \pmod{p^{e+1}}$.*

Proof. The first equality is obvious, and the proof of the second equality is just by the binomial expansion of $(z_0 + p^e z_1)^p$. □

Corollary 5.4 *For every prime p there exist a sequence of integer polynomials $f_1, f_2, \ldots$, all of degree $\le p-1$, such that for every exponent $e \ge 1$ and every integer $z = z_0 + p^e z_1$ (with z_0, z_1 integers, $z_0 \in [p]$), we have*

$$z^p = z_0 + \sum_{i=1}^{e} f_i(z_0) p^i \pmod{p^{e+1}}.$$

Proof. From Lemma 5.3 we know that the mod-p digits of z^p modulo-p^{e+1} depend only on z_0, so there exist some polynomials in z_0 that describe them, $f_i(z_0) = z^p\langle i \rangle_p$. Since these f_i's are polynomials from $\mathbb{Z}_p$ to itself, then they have degree at most $p-1$. Moreover, by the 1st equality in Lemma 5.3 we have that the first digit is exactly z_0. □

Corollary 5.5 *For every prime p and every $e \ge 1$ there exist a degree-p polynomial F_e, such that for every integers z_0, z_1 with $z_0 \in [p]$ and every $1 \le e' \le e$ we have $F_e(z_0 + p^{e'} z_1) = z_0 \pmod{p^{e'+1}}$.*

Proof. Denote $z = z_0 + p^{e'} z_1$. Since $z = z_0 \pmod{p^{e'}}$ then $f_i(z_0) = f_i(z) \pmod{p^{e'}}$. This implies that for all $i \ge 1$ we have $f_i(z_0)p^i = f_i(z)p^i \pmod{p^{e'+1}}$, and of course also for $i \ge e'+1$ we have

Digit-Extraction$_p(z, e)$: // Extract eth digit in base-p representation of z

1. $w_{0,0} \leftarrow z$
2. For $k = 0$ to $e - 1$
3. $\quad y \leftarrow z$
4. $\quad$ For $j = 0$ to k
5. $\quad\quad w_{j,k+1} \leftarrow F_e(w_{j,k})$ // F_e from Corollary 5.5, for $p = 2, 3$ we have $F_e(X) = X^p$
6. $\quad\quad y \leftarrow (y - w_{j,k+1})/p$
7. $\quad w_{k+1,k+1} \leftarrow y$
8. Return $w_{e,e}$

Figure 1: The digit extraction procedure

$f_i(z)p^i = 0 \pmod{p^{e'+1}}$. Therefore, setting $F_e(X) = X^p - \sum_{i=1}^{e} f_i(X)p^i$ we get

$$F_e(z) = z^p - \sum_{i=1}^{e} f_i(z)p^i = z^p - \sum_{i=1}^{e'} f_i(z_0)p^i = z_0 \pmod{p^{e'+1}}. \quad \square$$

We know that for $p = 2$ we have $F_e(X) = X^2$ for all e. One can verify that also for $p = 3$ we have $F_e(X) = X^3$ for all e (when considering the balanced mod-3 representation), but for larger primes $F_e(X) \neq X^p$.

The digit-extraction procedure. Just like in the base-2 case, in the procedure for extracting the eth base-p digit from the integer $z = \sum_i z_i p^i$ proceeds by computing integers $w_{j,k}$ ($k \geq j$) such that the lowest digit in $w_{j,k}$ is z_j, and the next $k - j$ digits are zeros. The code in Figure 1 is purposely written to be similar to the code from [1, Appendix B], with the only difference being in Line 5 where we use $F_e(X)$ rather than X^2.

In our implementation we compute the coefficients of the polynomial F_e once and store them for future use. In the procedure itself, we apply a homomorphic polynomial-evaluation procedure to compute $F_e(w_{j,k})$ in Line 5. We note that just as in [21, 1], the homomorphic division-by-p operation is done by multiplying the ciphertext by the constant $p^{-1} \bmod q$, where q is the current modulus. Since the encrypted values are guaranteed to be divisible by p, then this has the desired effect and also it reduces the noise magnitude by a factor of p. Correctness of the procedure from Figure 1 is proved exactly the same way as in [21, 1], the proof is omitted here.

5.3.1 An optimization for $p = 2$, $r \geq 2$.

As it turns out, for $p = 2$ we can sometimes extract several consecutive bits a little cheaper than what the procedure above implies. Specifically, it turns out that for $p = 2, e \geq 0$ and $r \geq 2$ we can compute the integer $z\langle e + r, \ldots, e\rangle$ by extracting only $e + r - 1$ bits (rather than $e + r$ of them). Specifically, when applying the procedure from Figure 1 (which for $p = 2$ is identical to the one from [1, Appendix B]), it turns out that we get

$$z\langle e + r, \ldots, e\rangle = \sum_{j=r}^{e+r-1} 2^{j-r} w_{j,e+r-1} \pmod{2^{e+r+1}}.$$

Note: the above would have been an immediate corollary from the correctness of the bit-extraction procedure if we added the terms $2^{j-r} w_{j,e+r}$ and let the index j go up to $e + r$, but in this case we can stop one step earlier and the result still holds.

To see why this works, observe that (by correctness), when we assign $w_{k+1,k+1} \leftarrow y$ in line 7 then it must be the case that $LSB(y) = z\langle k + 1\rangle$, and in subsequent iterations we just square w_{k+1}

so as to get more zeros in higher-order bits, without changing the LSB. Recall also that squaring indeed has the desired effect since for any $i \geq 1$ and any bit b and integer n we have $(b + 2^i n)^2 = b \pmod{2^{i+1}}$. To prove the optimization, we need two additional observations:

Observation 1. *For any bit b and integer n we have $(b + 2n)^4 = b \pmod{16}$.*

Note that this is *not a corollary* of the squaring property above — that property only gives $b \pmod 8$, but in fact for this particular case we get one extra zero. (This property holds only for that particular step, for later steps we only get one additional zero per squaring.)

Observation 2. *After line 7 in Figure 1, we always have $z = \sum_{j=0}^{k+1} 2^j w_{j,k+1}$.*

This can be verified by inspection: we start in line 3 from $y = z$, and at every step we subtract one w_j and divide by two, so adding them back with their respective powers of two gives back z.

Correctness now follows: Let us denote $w_j \stackrel{\text{def}}{=} w_{j,e+r-1}$ so we will not have to carry this extra index everywhere. Because of the first observation, the w_j's for $j = 0, 1, ..., e + r - 3$ have an extra zero bit, so for these w_j's we have $w_j = z\langle j\rangle \pmod{2^{e+r-j+1}}$, not just $\pmod{2^{e+r-j}}$. Denoting $v_j = 2^j w_j$, this means that the only v_j's that potentially have a nonzero bit in position $e + r$ are v_{e+r-2} and v_{e+r-1}. Also by correctness, for lower bit positions $j < e + r$, only v_j potentially has nonzero bit in position j, and all the other v_j's have zero in that position. Namely, we have

bit position:	$\star$	$e+r$	$e+r-1$	$e+r-2$	$e+r-3$	$\dots$	1	0
$v_0 = w_0 =$	$\star$	0	0	0	0		0	$z\langle 0\rangle$
$v_1 = 2w_1 =$	$\star$	0	0	0	0		$z\langle 1\rangle$	0
	$\vdots$					$\vdots$		
$v_{e+r-3} = 2^{e+r-3} w_{e+r-3} =$	$\star$	0	0	0	$z\langle e+r-3\rangle$		0	0
$v_{e+r-2} = 2^{e+r-2} w_{e+r-2} =$	$\star$	σ	0	$z\langle e+r-2\rangle$	0		0	0
$v_{e+r-1} = 2^{e+r-1} w_{e+r-1} =$	$\star$	τ	$z\langle e+r-1\rangle$	0	0		0	0

for some two bits σ, τ (where the $\star$'s are bits above position $e + r$, which we do not care about).

This means that when adding $\sum_{j=0}^{e+r-1} v_j$, we have no carry bits up to position $e+r$. But by the second observation the sum of all these v_j's is z, so the two top bits σ, τ must satisfy $\sigma \oplus \tau = z\langle e + r\rangle$. We conclude that when adding $\sum_{j=e}^{e+r-1} v_j$, we get all the bits $z\langle e + r, \dots, e\rangle$ which is what we needed to prove.

5.4 Putting Everything Together

Having described all separate parts of our recryption procedure, we now explain how they are combined in our implementation.

Initialization and parameters. Given the ring parameter m (that specifies the mth cyclotomic ring of integers $R = \mathbb{Z}[X]/(\Phi_m(X))$) and the plaintext space p^r, we compute the recryption parameters as explained in Section 6. That is, we set the exponents e, e' from Lemmas 5.1. We also precompute some key-independent tables for use in the linear transformations, with the first transformation using plaintext space $p^{e-e'+r}$ and the second transformation using plaintext space p^r.

Key generation. During key generation we choose in addition to the "standard" secret key sk also a separate secret recryption key $\tilde{\mathsf{sk}} = (1, \tilde{\mathfrak{s}})$. We include in the secret key both a key-switching matrix from sk to $\tilde{\mathsf{sk}}$, and a ciphertext $\tilde{\mathsf{ct}}$ that encrypts $\tilde{\mathfrak{s}}$ under key sk, relative to plaintext space $p^{e-e'+r}$.

The recryption procedure itself. Given a mod-p^r ciphertext ct relative to the "standard" key sk, we first key-switch it to $\tilde{\mathsf{sk}}$ and modulus-switch it to $\tilde{q} = p^e + 1$, then make its coefficients divisible by $p^{e'}$ using the procedure from Lemma 5.2, thus getting a new ciphertext $\mathsf{ct}' = (\mathfrak{c}'_0, \mathfrak{c}'_1)$. We then compute the homomorphic inner-product divided by $p^{e'}$, by setting $\mathsf{ct}'' = (\mathfrak{c}'_1/p^{e'}) \cdot \tilde{\mathsf{ct}} + (0, \mathfrak{c}'_0/p^{e'})$.

Next we apply the first linear transformation (the map Eval^{-1} from Section 4.2), moving to the slots the coefficients of the plaintext $\mathfrak{u}'$ that is encrypted in ct''. The result is a single ciphertext with *fully packed slots*, where each slot holds d of the coefficients from $\mathfrak{u}'$. Before we can apply the digit-extraction procedure from Section 5.3, we therefore need to *unpack* the slots, so as to put each coefficient in its own slot, which results in d "sparsely packed" ciphertexts (as described in Section 4.3).

Next we apply the digit-extraction procedure from Section 5.3 to each one of these d "sparsely packed" ciphertexts. For each one we extract the digits up to $e + r - e'$ (or up to $e + r - e' - 1$ if $p = 2$ and $r > 2$), and combine the top digits as per Lemma 5.1 to get in the slots the coefficients of the plaintext polynomial $\mathfrak{m}$ (one coefficient per slot). The resulting ciphertexts all have plaintext space mod-p^r.

Next we re-combine the d ciphertext into a single fully-packed ciphertext (as described in Section 4.3) and finally apply the second linear transformation (the map Eval described in Section 4.2). This completes the recryption procedure.

6 Parameters for Recryption

Here we explain our choice of parameters for the recryption procedure, in particular e and e'. To a large degree, the running time and depth of the digit extraction procedure depends on the size of $e - e'$, and so the goal is to minimize $e - e'$ while keeping the probability of an error acceptably small. The choice of e and e' depends on several other parameters:

- m, which defines the ring $F = \mathbb{R}[X]/(\Phi_m(X))$, and the number of distinct prime factors of m, denoted by t,
- the plaintext space p^r,
- the Hamming weight h of the secret key, and
- a parameter k that controls the error probability (which should be thought of as a "number of standard deviations").

6.1 Multiplying the Secret Key by a Random Element

Driving our parameter selection is a heuristic high-probability bound on the size of the element $w \cdot s \in F$, where s is the recryption secret key and w is a "random element", whose coefficients in the *powerful basis* are chosen independently from a zero-mean distribution with bounded variance. The recryption secret key s in `HElib` is generated as follows:

- we choose coefficients $s_0, \ldots, s_{m-1}$, where a randomly chosen subset of h coefficients is set to ± 1 uniformly and independently, and the remaining $m - h$ coefficients are set to zero;
- we then form the polynomial $\sum_{i=0}^{m-1} s_i X^i$, and s is the image of this polynomial in F, i.e. the element in F whose power-basis representation is $(\sum_{i=0}^{m-1} s_i X^i) \bmod \Phi_m(X)$.

Let $|x|$ denote the ℓ_∞ norm of a ring element $x \in F$ in the powerful basis. Our heuristic analysis in Section 6.1.1 below, which is validated by experiments, establishes a high-probability bound on

$|ws|$ where w, s are chosen as above. Namely, if the coefficients of w are chosen independently from a zero-mean distribution with variance bounded by σ^2, it suggests that

$$\Pr\left[\ |ws| > B \cdot \sigma\ \right] \lessapprox \phi(m) \cdot \mathrm{erfc}(k/\sqrt{2}), \tag{5}$$

where B is the size bound

$$B \stackrel{\mathrm{def}}{=} k \cdot 2^{t/2} \cdot \sqrt{h} \cdot \sqrt{\frac{\phi(m)}{m}}, \tag{6}$$

and erfc is the complementary error function (so $\mathrm{erfc}(k/\sqrt{2})$ is the probability that a normal random variable takes a value that is more than k standard deviations away from its mean). A useful special case is when the coefficients of w are chosen uniformly at random in $[-\frac{1}{2}, +\frac{1}{2}]$, in which case we have $\sigma^2 = 1/12$ and we get the bound $\Pr\left[\ |ws| > B^*\ \right] \lessapprox \phi(m) \cdot \mathrm{erfc}(k/\sqrt{2})$, with the size bound

$$B^* \stackrel{\mathrm{def}}{=} \frac{B}{2\sqrt{3}} \ = \ k \cdot \frac{2^{t/2}}{2\sqrt{3}} \cdot \sqrt{h} \cdot \sqrt{\frac{\phi(m)}{m}}.$$

In our implementation, we use a default value of $k = 10$, for that value of k we have $\mathrm{erfc}(k/\sqrt{2}) \approx 2^{-76}$. By Eqn. (5), this will keep the probability that $|ws|$ exceeds B bounded by $\approx 2^{-60}$ for all reasonable values of m (with $\phi(m)$ bounded by 2^{16}).

6.1.1 Justifying the Bound (5)

The analysis below considers the powerful basis for F with respect to the factorization into prime powers[4] $m = m_1 \cdots m_t$. We want to bound $|ws|$, where $|\cdot|$ denotes the ℓ_∞-norm in the powerful basis, and where w and s are chosen as described above.

The multiply-by-s matrix. Fix s, and let M_s denote the matrix representing the multiplication-by-s map on the powerful basis. That is, if $\vec{w} = (w_1, \ldots, w_{\phi(m)})^\intercal$ is the powerful basis coordinate vector of w, then the coordinate vector of ws is $M_s \cdot \vec{w}$. The following lemma ties the structure of the matrix M_s to the number of prime-power factors of m:

Lemma 6.1 *Recall that s was chosen in terms of the coefficients $s_0, \ldots, s_{m-1}$. Each entry in M_s is the sum of 2^t distinct coefficients (or their negations). Moreover, for any row of M_s, each coefficient s_i contributes to at most 2^t different entries in that row.*

Proof. As a warm up, consider the case where m is itself a prime (so $t = 1$ and the powerful basis is the same as the usual power basis). In this case, the matrix M_s looks like this:

$$M_s = \begin{pmatrix} s_0 - s_{m-1} & s_{m-1} - s_{m-2} & \cdots & s_2 - s_1 \\ s_1 - s_{m-1} & s_0 - s_{m-2} & \cdots & s_3 - s_1 \\ \vdots & \vdots & \vdots & \vdots \\ s_{m-2} - s_{m-1} & s_{m-3} - s_{m-2} & \cdots & s_0 - s_1 \end{pmatrix}.$$

The jth column of M_s is the coefficient vector of $s \cdot X^j \bmod \Phi_m(X)$. It can be obtained by first rotating the vector $(s_0, \ldots, s_{m-2}, s_{m-1})^\intercal$ by j positions, corresponding to multiplication of s by X^j mod $X^m - 1$, then reducing modulo $\Phi_m(X) = 1 + \cdots + X^{m-2} + X^{m-1}$.

[4]The analysis in Section 4 considered a more general notion of powerful basis, with respect to arbitrary pairwise co-prime factorizations of m. The analysis here does not apply to this more general notion.

The rotation yields the coefficient vector $(s_{-j}, \ldots, s_{m-2-j}, s_{m-1-j})^\intercal$, with subscripts computed modulo m. By virtue of the congruence

$$X^{m-1} \equiv -(1 + \cdots + X^{m-2}) \pmod{\Phi_m(X)},$$

we have that reducing modulo $\Phi_m(X)$ is tantamount to subtracting s_{m-1-j} from the first $m-1$ entries of the rotated vector (and deleting the last entry). Hence the resulting coefficient vector is $(s_{-j} - s_{m-1-j}, \ldots, s_{m-2-j} - s_{m-1-j})^\intercal$. One can verify by direct inspection that the claims of the lemma are satisfied in this case.

In the case of general $m = m_1 \cdots m_t$, where $m_i = u_i^{e_i}$, one can proceed in a similar fashion. As a preliminary matter, we shall work with the coordinate vector of s in the natural basis for the $\mathbb{R}$-algebra

$$A = \mathbb{R}[X]/(X_1^{m_1} - 1, \ldots, X_t^{m_t} - 1),$$

which consists of the monomials $X_1^{j_1}, \ldots, X_t^{j_j}$, where

$$j_1 = 0, \ldots, m_1 - 1, \quad \ldots, \quad j_t = 0, \ldots, m_t - 1.$$

Note that A is isomorphic to $\mathbb{R}[X]/(X^m - 1)$ by the isomorphism that sends X_i to X^{m/m_i}. Because of this isomorphism, the entries of the coordinate vector of s with respect to the standard power basis for $\mathbb{R}[X]/(X^m - 1)$ are just a permutation of the entries with respect to the natural (tensor) basis for A.

Let $\vec{j} = (j_1, \ldots, j_t)$ index a particular column of M_s, corresponding to multiplication by the monomial $X_1^{j_1} \cdots X_t^{j_t}$ in A. That column of M_s is obtained by first permuting the coordinate vector of s, and then reducing modulo $\Phi_{m_1}(X_1), \ldots, \Phi_{m_t}(X_t)$.

The coordinate vector of s is naturally viewed as a t-dimensional hypercube, and multiplying by $X_1^{j_1} \cdots X_t^{j_t}$ (modulo $X^m - 1$) correspond to rotating this hypercube by amounts $j_1, \ldots, j_t$ in each dimension: if $s[i_1, \ldots, i_t]$ denotes one entry in the coordinate vector for s, then the corresponding entry in the coordinate vector of $s' = s \cdot X_1^{j_1} \cdots X_t^{j_t} \bmod (X^m - 1)$ is $s'[i_1, \ldots, i_t] = s[i_1 - j_1, \ldots, i_t - j_t]$ (where each index $i_r - j_r$ is reduced modulo the corresponding modulus m_r), this s' has degree $m-1$. To get the coordinate vector of $s \cdot X_1^{j_1} \cdots X_t^{j_t} \in F$ with respect to the powerful basis, we need to reduce this s' modulo each of $\Phi_{m_1}(X_1), \ldots, \Phi_{m_t}(X_t)$.

Let us denote $L(a,b) \stackrel{\text{def}}{=} (a \bmod b) - b \in [-b, -1]$. Reducing modulo the first polynomial $\Phi_{m_1}(X_1) = 1 + X_1^{m_1/u_1} + X_1^{2m_1/u_1} + \cdots + X_1^{(u_1-1)m_1/u_1}$, the $(i_1, \ldots, i_t)$-entry becomes

$$s''[i_1, i_2, \ldots, i_t] \quad = \quad s'[i_1, i_2, \ldots, i_t] - s'[i_1', i_2, \ldots, i_t],$$

where $i_1' = L(i_1, \frac{m_1}{u_1})$. (Note that when m_1 is itself prime, so $m_1 = u_1$, we have $i_1' = L(i_1, \frac{m_1}{u_1}) = -1$.) Further reducing modulo $\Phi_{m_2}(X_2)$, the $(i_1, \ldots, i_t)$-entry becomes

$$\begin{aligned}
&s''[i_1, i_2, \ldots, i_t] - s''[i_1, L(i_2, \tfrac{m_2}{u_2}), \ldots, i_t] \\
&= s'[i_1, i_2, \ldots, i_t] - s'[L(i_1, \tfrac{m_1}{u_1}), i_2, \ldots, i_t] - s'[i_1, L(i_2, \tfrac{m_2}{u_2}), \ldots, i_t] + s'[L(i_1, \tfrac{m_1}{u_1}), L(i_2, \tfrac{m_2}{u_2}), \ldots, i_t] \\
&= s[i_1 - j_1, i_2 - j_2, \ldots, i_t - j_t] - s[L(i_1, \tfrac{m_1}{u_1}) - j_1, i_2 - j_2, \ldots, i_t - j_t] \\
&\quad - s[i_1 - j_1, L(i_2, \tfrac{m_2}{u_2}) - j_2, \ldots, i_t - j_t] + s[L(i_1, \tfrac{m_1}{u_1}) - j_1, L(i_2, \tfrac{m_2}{u_2}) - j_2, \ldots, i_t - j_t].
\end{aligned}$$

Continuing in this way, the $(i_1, \ldots, i_t)$-entry in the powerful basis of $s \cdot X^{j_1} \ldots X^{j_t}$ (with $0 \leq i_r \leq \phi(m_r) - 1$) is

$$\sum_{\tau_1, \ldots, \tau_t} (-1)^{\tau_1 + \cdots + \tau_t} s\Big[\Delta_{\tau_1}(i_1, L(i_1, \tfrac{m_1}{u_1})) - j_1, \ \ldots, \ \Delta_{\tau_t}(i_t, L(i_t, \tfrac{m_t}{u_t})) - j_t\Big], \tag{7}$$

where the sum is over all $(\tau_1, \ldots, \tau_t) \in \{0,1\}^t$, and $\Delta_\tau(a,b)$ is defined to be a if $\tau = 0$ and b if $\tau = 1$. One row of the matrix M_s, then, is comprised of the entries (7) for all the columns $(j_1, \ldots, j_t)$ (with $0 \le j_r \le \phi(m_r) - 1$). For this row, any one value $s[k_1, \ldots, k_t]$ appears as a term in those columns that are indexed by $(j_1, \ldots, j_t)$ such that

$$\begin{aligned} j_1 &\in \{\ i_1 - k_1 \bmod m_1,\quad L(i_1, \tfrac{m_1}{u_1}) - k_1 \bmod m_1\ \}, \\ &\ldots, \\ j_t &\in \{\ i_t - k_t \bmod m_t,\quad L(i_t, \tfrac{m_t}{u_t}) - k_t \bmod m_t\ \}. \end{aligned} \tag{8}$$

That proves the lemma in the case of a general m. □

The coefficients of ws. Consider now a single coefficient g of ws in the powerful basis, namely an entry in the vector $M_s \cdot \vec{w}$. This coefficient can be expressed as a sum of random variables

$$g = e_1 w_1 + \cdots + e_{\phi(m)} w_{\phi(m)},$$

where the w_i's are coefficients of w and the e_i's are entries in one row of M_s. Recalling that the coefficients w_i's are independent zero-mean random variables with variance σ^2 and conditioning on a fixed s, the random variable g is the sum of independent random variables of bounded variance. By Lemma 6.1, the ℓ_1-norm of this row of M_s is bounded by $2^t h$. It follows that for this fixed s the variance of g itself is at most $2^{2t} h \cdot \sigma^2$.

Thus, we could apply the Central Limit Theorem to argue that for this fixed s, the distribution of g closely approximates a zero-mean Normal random variable with variance at most $2^{2t} h \cdot \sigma^2$, and hence for $B' = k \cdot 2^t \sqrt{h}$, we have $\Pr[|g| > B' \cdot \sigma] \lessapprox \mathrm{erfc}(k/\sqrt{2})$. The approximate inequality (5), for this value of B', would then follow from the union bound.

Note, however, that this argument is quite pessimistic, and $B' = k \cdot 2^t \sqrt{h}$ is significantly larger than the value given in (6). We next argue (a bit heuristically) that the approximate probability bound (5) should indeed hold with the smaller size bound B as defined in (6).

Note that while the w_i's are assumed independent, the e_i's (for s chosen at random as above) are not, and hence the terms $e_i w_i$ are not independent. Nonetheless, they appear "almost independent" (which is backed up by experiments). Hence we compute below the sum of variances $\sum_i \mathsf{Var}[e_i w_i]$ and treat it as if it were the variance of the sum $\mathsf{Var}[\sum_i e_i w_i]$.

Lemma 6.2 *For s, w chosen at random as described above, we have*

$$\sum_i \mathsf{Var}[e_i w_i] = \frac{\phi(m)}{m} \cdot 2^t \cdot h \cdot \sigma^2.$$

Proof. Recall that s is generated at random to have h nonzero coefficients, where each nonzero coefficient is chosen uniformly from $\{-1, 1\}$. We can think of these nonzero coefficients of s as being generated in a series of h rounds: for $\ell = 1, \ldots, h$, in the ℓth round, we choose the position of the ℓth nonzero coefficient of s uniformly from $\{0, \ldots, m-1\}$, repeating as necessary until finding a position that has not already been chosen in one of the previous rounds, and then we choose the value of the ℓth nonzero coefficient unformly from $\{-1, 1\}$.

For $\ell = 1, \ldots, h$, we define X_ℓ to be the number of e_i's to which the ℓth nonzero coefficient of s contributes.

Now, instead of conditioning on a fixed value of s, as we did above, let us instead condition on a fixed choice $\mathcal{C}$ of h positions where coefficients of s are nonzero (and uniform in $\{-1, 1\}$). We can compute the sum of the individual variances, conditioned on the particular choice $\mathcal{C}$:

$$\sum_i \mathsf{Var}[e_i w_i \mid \mathcal{C}] = \sum_i \mathsf{Var}[e_i \mid \mathcal{C}] \mathsf{Var}[w_i] = \sigma^2 \sum_i \mathsf{Var}[e_i \mid \mathcal{C}] = \sigma^2 \sum_\ell \mathsf{E}[X_\ell \mid \mathcal{C}].$$

The first equality follows from independence. The last equality holds because for each i, the value $\mathsf{Var}[e_i \mid \mathcal{C}]$ is equal to the number of nonzero coefficients of s that contribute to e_i. Averaging over all choices $\mathcal{C}$, we have

$$\sum_i \mathsf{Var}[e_i w_i] = \sum_{\mathcal{C}} \Pr[\mathcal{C}] \cdot \sum_i \mathsf{Var}[e_i w_i \mid \mathcal{C}] = \sum_{\mathcal{C}} \Pr[\mathcal{C}] \cdot \sigma^2 \sum_\ell \mathsf{E}[X_\ell \mid \mathcal{C}] = \sigma^2 \sum_\ell \mathsf{E}[X_\ell].$$

Now, for each individual $\ell = 1, \ldots, h$, the position of the ℓth nonzero coefficient of s is uniformly distributed over $\{0, \ldots, m-1\}$, and so it follows that the corresponding coordinate vector $\vec{k} = (k_1, \ldots, k_t)$ in the t-dimensional hypercube introduced in the proof of Lemma 6.1 is uniformly distributed over this hypercube. There are at most 2^t row entries to which this nonzero coefficient contributes, namely the ones corresponding to tuples $\vec{j} = (j_1, \ldots, j_t)$ that satisfy Eqn. (8). But some of these tuples have $j_r > \phi(m_r)$ (for some r), and hence are not valid. In fact for a uniformly chosen coordinate vector $\vec{k}$ in the hypercube, each one of these $\vec{j}$ tuple has only $\phi(m)/m$ probability of also satisfying $0 \le j_r \le \phi(m_r)$ for all r. Hence the expected number of valid tuples to which a uniform $\vec{k}$ contributes is exactly

$$\mathsf{E}[X_\ell] = 2^t \cdot \frac{\phi(m)}{m},$$

and the lemma follows. □

We stress that while Lemma 6.2 gives the precise value of the sum of variances $S = \sum_i \mathsf{Var}[e_i w_i]$, this may not be equal to the variance of the sum $\mathsf{Var}[g] = \mathsf{Var}[\sum_i e_i w_i]$, since the terms $e_i w_i$ are not independent. Nevertheless, based on this analysis, and the results of extensive experimentation, we believe that the distribution of g is well approximated by a normal random variable with variance S. Hence with the size bound B as given in (6), the approximate probability bound from (5) is fairly accurate.

6.2 Using the Bound (5)

The recryption input. Going into the recryption procedure, after key-switching to the recryption key and mod-switching to $q = p^e + 1$, we have a ciphertext (c_0, c_1). Denoting

$$x = c_0 + c_1 s$$

(without mod-q reduction), we recall that for Lemma 5.1 we would need to bound the expression $|x'/q| + |[x']_q|$. We use the analysis from above to bound both terms.

- Eqn. (5) can be used directly to bound the size of $c_1 s$: if we model the powerful-basis coefficients of c_1 as uniformly and independently distributed over the continuous interval $[-\frac{q}{2}, +\frac{q}{2}]$, then we get a heuristic high-probability bound $|c_1 s| \le qB^*$ and therefore $|x|/q \le |c_0|/q + |c_1 s|/q \le (B^* + 0.5)$. While the coefficients of c_1 are integers (and hence not continuous in $[\pm q/2]$), we show in Section 6.2.1 that this discretization introduced at most another small multiplicative factor of $(1 + 1/q^2)$ to this bound

$$|x|/q \le \Big(1 + \frac{1}{q^2}\Big)(B^* + 0.5). \tag{9}$$

- The term $|[x]_q|$, corresponds to the noise in the ciphertext (c_0, c_1), which is dominated by the mod-switching additive noise term $\epsilon_0 + \epsilon_1 s$, with the ϵ_i's the rounding terms. With plaintext space modulo p^r, we can approximately model the coefficients of ϵ_1 as uniform in the continuous interval $[\pm p^r/2]$, so Eqn. (5) yields a heuristic high-probability bound $|\epsilon_0 + \epsilon_1 s| \le p^r(B^* + 0.5)$.

For the term $[x]_q$, we have another contribution due to the scaled noise from before the modulus switching, but as we explain in Section 6.2.1, this term will be smaller so the overall bound is less than doubled

$$|[x]_q| \le 2p^r(B^* + 0.5). \tag{10}$$

The make-divisible operation. After making the ciphertext divisible by $p^{e'}$ using Lemma 5.2, we have a new ciphertext $(c_0', c_1') = (c_0, c_1) + q(v_0, v_1)$. Let $y = v_0 + v_1 s$. Modeling the coefficients of (c_0, c_1) as independently uniform in $[\pm q/2]$, it is reasonable to model the coefficients of (v_0, v_1) on the powerful basis as independently uniform in $[\pm p^{e'}/2]$, and we get from Eqn. (5) a heuristic high-probability bound $|y| \le p^{e'}(B^* + 0.5)$, with another small multiplicative factor due to discretization

$$|y| \le p^{e'}(1 + \delta)(B^* + 0.5), \tag{11}$$

where $\delta = 1/p^{2e'}$ if $p = 2$ and $\delta = 1/q$ if p is odd.

Satisfying the conditions of Lemma 5.1. Denoting $x' = c_0' + c_1' s = x + qy$, applying Lemma 5.1 requires that we satisfy $|x'/q| + |[x']_q| \le (q-1)/2 = p^e/2$. Since $|x'/q| \le |x/q| + |y|$ and $|[x']_q| = |[x]_q|$, then applying the bounds in (9)–(11) we get the heuristic high-probability bound

$$|x'/q| + |[x']_q| \le |x/q| + |y| + |[x]_q| \ \le\ (B^* + 0.5) \cdot \left(p^{e'}(1+\delta) + 2p^r + 1 + \frac{1}{q^2} \right)$$

Our parameter-setting procedure for recryption attempts to minimize $e - e'$ (which corresponds to the recryption depth) subject to the constraint

$$(B^* + 0.5) \cdot \left(p^{e'}(1+\delta) + 2p^r + 1 + \frac{1}{q^2} \right) \ \le\ p^e/2, \tag{12}$$

and also keeping $p^e < 2^{30}$ to avoid integer overflow problems.

6.2.1 Low-level details of the analysis

Two details of the analysis that we still need to address are the discretization effect (since the coefficients must be integers rather than uniformly in some continuous interval), and the initial ciphertext noise from before bootstrapping.

Discretization: the symmetric distribution mod M. Let $M \ge 2$ be an integer, and consider the following probability distribution over the integers in the range $[\pm M/2]$: If M is odd, then the symmetric distribution mod M is simply the uniform distribution on this set of integers $\mathbb{Z} \cap [-\lfloor M/2 \rfloor, +\lfloor M/2 \rfloor]$. If M is even, then the symmetric distribution mod M assigns probability mass $1/2M$ to the integers $\pm M/2$, while the integers of magnitude strictly smaller than $M/2$ are each assigned probability mass $1/M$. Note that for the symmetric distribution mod M, each residue class mod M is equally likely, and the distribution is symmetric about zero (and in particular, its mean is zero). Instead of the variance $M^2/12$ for the continuous distribution on $[\pm M/2]$, we have:

Lemma 6.3 *Let $M \ge 2$ be an integer and X be a random variable that is symmetrically distributed mod M. Then $\mathsf{Var}[X] \le \frac{M^2}{12}$ if M is odd and $\mathsf{Var}[X] \le \frac{M^2}{12} \cdot \left(1 + \frac{2}{M^2}\right)$ if M is even.*

Proof. Let $N = \lfloor M/2 \rfloor$. If M is odd then we have $\mathsf{Var}[X] = \frac{2}{2N+1} \cdot \sum_{i=1}^{N} i^2 \le \frac{M^2}{12}$, and if M is even then $\mathsf{Var}[X] = \frac{N^2}{2N} + \frac{2}{2N} \cdot \sum_{i=1}^{N-1} i^2 = \frac{M^2}{12}\left(1 + \frac{2}{M^2}\right)$. □

We therefore replace the assumption that the coefficients of c_0, c_1 are uniform in the continuous interval $[\pm q/2]$, by the symmetric distribution mod q, which increases the standard deviation used for Eqn. (9) by at most a factor of $\sqrt{1+\frac{2}{q^2}} \le 1+\frac{1}{q^2}$. The factor $(1+\delta)$ from Eqn. (11) and the factor 2 from Eqn. (10) are explained next.

Modular reduction of the symmetric distribution and the bound (11). Recall from the proof of Lemma 5.2 (on Page 14) that the terms v_0, v_1 in the make-divisible operation are set as $v_i = -c_i \bmod p^{e'}$. The coefficients of the c_i's are assumed to be symmetrically distributed mod q, and our goal is to get a bound on the variance of the coefficients of v_i's (which we can use in Eqn. (6)).

Specifically, we are taking an integer coefficient c, which is symmetrically distributed mod q, and reducing it symmetrically mod $p^{e'}$ to get another value d. (If p is even then this reduction chooses between the two endpoints $\pm p^{e'}/2$ at random.) We claim that the value d thus obtained has zero mean and standard deviation is bounded by $\frac{p^{e'}}{2\sqrt{3}} \cdot (1+\delta)$, where $\delta = 1/p^{2e'}$ if $p=2$ and $\delta = 1/q$ if p is odd.

Case 1: $p = 2$. Recall that $q-1$ is divisible by $p^{e'}$ (since $q = p^e + 1$ for some $e \ge e'$). Conditioned on $c \ne 0$, the residue class $c \bmod p^{e'}$ is therefore uniformly distributed over the residue classes mod $p^{e'}$ (while conditioned on $c = 0$ we have $d = 0$). The distribution of d is thus a convex combination of the symmetric distribution mod $p^{e'}$ and the constant zero (which both have zero mean). Hence d has zero mean, and its variance is no more than that of the symmetric distribution mod $p^{e'}$, which is $p^{2e'}/12 \cdot (1 + 2/p^{2e'})$, so the standard deviation is at most $\frac{p^{e'}}{2\sqrt{3}} \cdot (1 + \frac{1}{p^{2e'}})$.

Case 2: p **odd.** In this case, $q = p^e+1$ is even and $p^{e'}$ is odd. Conditioned on $c \ne \pm q/2$, the residue class $c \bmod p^{e'}$ is uniformly distributed over the residue classes mod $p^{e'}$; therefore, the distribution of d is symmetric mod $p^{e'}$, and by Lemma 6.3 its variance is at most $p^{2e'}/12$. Conditioned on $c = \pm q/2$, d is uniform over $\{\pm\lfloor p^{e'}/2\rfloor\}$, which has variance $p^{2e'}/4$. Hence d has zero mean, and since $c = \pm q/2$ with probability $1/q$, then

$$\sigma^2 = \mathsf{Var}[d] \le \frac{p^{2e'}}{12} \cdot \left(1 - \frac{1}{q}\right) + \frac{p^{2e'}}{4} \cdot \frac{1}{q} = \frac{p^{2e'}}{12}\left(1 + \frac{2}{q}\right).$$

The standard deviation is thus bounded by $\frac{p^{e'}}{2\sqrt{3}} \cdot (1 + \frac{1}{q})$.

The ciphertext noise and the bound (10). During the computation in `HElib` we keep track of the ℓ_∞-norm of *the canonical embedding* of the noise. Below we denote this canonical-embedding norm of an element $x \in F$ by $|x|^c$. For bootstrapping, however, we are interested in the ℓ_∞-norm of x on the powerful basis, which is denoted $|x|$.

Heading into recryption, after key-switching to the recryption key but *before modulus switching to* $q = p^e + 1$, we have a decryptable ciphertext $(\tilde{c}_0, \tilde{c}_1) \in R_Q$ (for some $Q \gg q$) with noise magnitude η in the canonical embedding. Namely we have

$$\tilde{x} \stackrel{\text{def}}{=} [\tilde{c}_0 + \tilde{c}_1 s]_Q \text{ with } \eta \stackrel{\text{def}}{=} |\tilde{x}|^c \ll Q.$$

To get a handle on the noise magnitude *in the powerful basis* after modulus switching, we begin by bounding $|\tilde{x}|$ in terms of $|\tilde{x}|^c$. Let $\tan(\cdot)$ be the tangent function, and for a real number u define

$$P(u) \stackrel{\text{def}}{=} \frac{2}{u \cdot \tan(\pi/2u)}.$$

Below we use the values of $P(u)$ at prime numbers u. One can verify that $P(u)$ approaches $4/\pi \approx 1.273$ from below as $u \to \infty$, and the (approximate) values of $P(u)$ for the first few primes are:

u	2	3	5	7	11
$P(u)$	1	1.155	1.231	1.252	1.265

The following lemma generalizes Lemma 5 in [13]. For completeness, we give a self-contained proof.

Lemma 6.4 *For all $w \in F$, we have $|w| \leq |w|^c \cdot \prod_{\text{prime } u \mid m} P(u)$.*

Proof. We first prove the lemma in the case where m is itself prime, where the powerful basis is the same as the standard power basis. Consider the $(m-1) \times (m-1)$ matrix CRT_m, representing the linear map that evaluates a polynomial of degree less than $m-1$ at the $m-1$ primitive mth roots of unity. To prove the lemma for this case, it suffices to show that the ℓ_∞-norm of this matrix, $N = N_\infty(\mathrm{CRT}_m^{-1})$, is equal to $P(m)$. Recall that if $\mathrm{CRT}_m^{-1} = (a_{ij})$, then the ℓ_∞ norm of CRT_m is

$$N = \max_i \sum_j |a_{ij}|.$$

We first calculate the entries a_{ij} of CRT_m^{-1} explicitly. Let DFT_m be the $m \times m$ matrix corresponding the Discrete Fourier Transform, i.e., $\mathrm{DFT}_m = (\omega_m^{ij})$, where the indices i and j range over $\{0, \dots, m-1\}$, and $\omega_m = \exp(2\pi \mathrm{i}/\mathrm{m})$. The matrix CRT_m is obtained by deleting row 0 and column $m-1$ from DFT_m. We know that $\mathrm{DFT}_m^{-1} = m^{-1}(\omega_m^{-ij})$. From this, we can apply general results that express the inverse of a submatrix in terms of the inverse of a matrix. For example, Theorem 2.1 of [30] implies that for $i \neq m-1$ and $j \neq 0$, we have

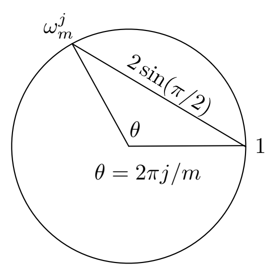
$$a_{ij} = \frac{1}{m}(\omega_m^{-ij} - \omega_m^j) = \frac{\omega_m^j}{m}(\omega_m^{-(i+1)j} - 1) \implies |a_{ij}| = \frac{1}{m} \cdot |\omega_m^{-(i+1)j} - 1|.$$

For every $i \neq m-1$, summing over all $j > 0$ we get $\sum_{j>0} |a_{ij}| = \frac{1}{m} \cdot \sum_{j>0} |\omega_m^j - 1|$, and hence

$$N = \frac{1}{m} \sum_{j>0} |\omega_m^j - 1|.$$

Each term $|\omega_m^j - 1|$ is the length of the chord of the unit circle corresponding to the angle $2\pi j/m$ (which is $2\sin(\frac{\pi j}{m})$). That is,

$$|\omega_m^j - 1| = 2\sin(\pi j/m) = 2\Im(\omega_{2m}^j),$$



where $\omega_{2m} = \exp(2\pi \mathrm{i}/2\mathrm{m})$ and $\Im(c)$ is the imaginary part of the complex number c. If follows that

$$mN = 2\sum_{j=1}^{m-1} \sin(\pi j/m) = 2\Im\left(\sum_{j=0}^{m-1} \omega_{2m}^j\right) = 2\Im\left(\frac{\omega_{2m}^m - 1}{\omega_{2m} - 1}\right) = 2\Im\left(\frac{-2}{\omega_{2m} - 1}\right)$$
$$= -4\Im\left(\frac{\omega_{2m}^{-1} - 1}{|\omega_{2m} - 1|^2}\right) = \frac{\sin(\frac{\pi}{m})}{\sin^2(\frac{\pi}{2m})} = \frac{2\sin(\frac{\pi}{2m})\cos(\frac{\pi}{2m})}{\sin^2(\frac{\pi}{2m})} = 2/\tan(\tfrac{\pi}{2m}).$$

The penultimate equality above uses the standard formula $\sin(2\alpha) = 2\sin(\alpha)\cos(\alpha)$. This completes the proof for the case where m is a prime.

When $m = u^e$ is a prime power, the lemma follows from the result for prime u, along with the fact (see [33, Sec 3]) that CRT_m^{-1} can be expressed as a product of several matrices:

- a block diagonal matrix with CRT_u^{-1} on the diagonal,
- a block diagonal matrix with $\mathrm{DFT}_{m/u}^{-1}$ on the diagonal,
- a diagonal matrix with roots of unity on the diagonal, and
- several permutations matrices,

The first matrix has ℓ_∞-norm $P(u)$, and the remaining matrices have ℓ_∞-norm 1. By the sub-multiplicativity of the ℓ_∞-norm, it follows that the ℓ_∞-norm of CRT_m^{-1} is at most $P(u)$. (Experimentally, the ℓ_∞-norm of CRT_m^{-1} appears to be equal to $P(u)$.)

When $m = m_1 \cdots m_t$ is the product of several prime powers, the result follows from the result for prime powers, and the fact (see [33, Sec 3]) that CRT_m can be expressed as a product of several matrices:

- for each $i = 1, \ldots, t$, one block diagonal matrix with $\mathrm{CRT}_{m_i}^{-1}$ on the diagonal, and
- several permutation matrices.

The lemma follows from the sub-multiplicativity of the ℓ_∞-norm. $\square$

Back to recryption, recall that before modulus-switching we have a ciphertext $(\tilde{c}_0, \tilde{c}_1)$ with noise $\tilde{x} = [\tilde{c}_0 + \tilde{c}_1 s]_Q$ of magnitude $\eta = |\tilde{x}|^c \ll Q$. We then modulus-switch it down to the bootstrapping modulus q to obtain a ciphertext (c_0, c_1) such that $c_i = \lceil \frac{q}{Q}\tilde{c}_i \rfloor = \frac{q}{Q}\tilde{c}_i + \epsilon_i$, where $\epsilon_i \in [\pm p^r/2]$ is a rounding term. The noise term after modulus switching is therefore

$$[x]_q = [c_0 + c_1 s]_q = \frac{q}{Q}\tilde{x} + \epsilon_0 + \epsilon_1 s,$$

and we seek a high-probability upper bound on the norm $|[x]_q|$ (in the powerful basis). The canonical-embedding norm of the first term on the right-hand side is $\frac{q}{Q} \cdot \eta$, and by Lemma 6.4 we can bound its powerful-basis norm by

$$\left|\frac{q}{Q}\tilde{x}\right| \leq \frac{q}{Q} \cdot \eta \cdot D_m$$

where $D_m \stackrel{\text{def}}{=} \prod_{\text{prime } u \mid m} P(u) \leq (4/\pi)^t$ (with t the number of primes that divide m).

When Q is large enough, the coefficients of the rounding terms ϵ_i can be heuristically modeled as independent random variables, each uniform in the continuous interval $[\pm p^r/2]$, hence we get the heuristic high-probability bound $|\epsilon_0 + \epsilon_1 s| \leq p^r(B^* + 0.5)$. If the initial noise magnitude η is small enough so as

$$\frac{q}{Q} \cdot \eta \cdot D_m \leq p^r(B^* + 0.5), \tag{13}$$

then the total noise magnitude is bounded by $2p^r(B^* + 0.5)$ as needed for the bound in Eqn. (10). Currently, `HElib` checks that (13) holds during bootstrapping, and prints a warning if this is not the case (which for typical parameters never happens).

6.3 Experimental validation

There are a several steps in the above analysis that are heuristic:

1. We assumed that certain ciphertext coefficients were essentially independently and uniformly distributed.

2. Beyond just assuming the coefficients of w are independently and uniformly distributed, we modeled the individual coefficients of ws as having a normal distribution with a certain variance. This involves two heuristic steps:

   (a) we expressed each coefficient as a sum of random variables of bounded variance, and calculated the sum of variances, but we ignored the fact that the terms in this sum are not independent;

   (b) even if the terms in this sum were independent, and we could apply the Central Limit Theorem, we did not use a quantitative version of the Central Limit Theorem.

The heuristics 1 and 2(a) in particular involve assuming that many variables "behave as is they were independent" for the purpose of noise growth, and the only way we could justify these assumptions is experimentally.

6.3.1 Coefficient sizes of ws for random w's

Perhaps the most questionable assumption that we made is assuming that the various terms in $\sum e_i w_i$ behave independently when the w_i's are independent and the e_i's are taken from a row of the multiply-by-s matrix (cf. Section 6.1.1), so we ran extensive experiments to validate this assumption. We generated 75 random values of $m \in \{25{,}000, \ldots, 45{,}000\}$ with 1 through 5 prime factors (15 random m's for each number of prime factors). For each of these m's, we ran 100,000 trials, each proceeding as follows:

1. Choose a random secret key s and element w.

   The secret key was chosen with hamming weight $h = 120$, and the coefficients of w on the powerful basis were independently and symmetrically distributed mod $2^{20} + 1$.

2. Compute $x = ws$, then choose one coefficient of x on the powerful basis at random, and output that coefficient.

That gave us 100,000 samples for each value of m, and we computed a few statistics to check if these samples are consistent with a normal random variable with variance $\sigma^2 = \sum_i \mathsf{Var}[e_i w_i]$ from Lemma 6.2:

- For each value of m, we calculated the fraction of the 100,000 samples that fell within 1, 2, and 3 times the predicted standard deviation σ. We got the following results:

	lowest m	predicted fraction	highest m
$1 \times \sigma$	0.6792	0.682689	0.6859
$2 \times \sigma$	0.9530	0.954499	0.9562
$3 \times \sigma$	0.9968	0.997300	0.9978

- For each m, we calculated the *sample variance* of the 100,000 samples, and compared it to the predicted variance σ^2. Of these 75 m's, the highest sample variance was $1.0116 \cdot \sigma^2$, the lowest was $0.9906 \cdot \sigma^2$, and the median was $1.00108 \cdot \sigma^2$. The corresponding p-values[5] are roughly 1/206, 1/57, and 0.40404. Aggregating these 75 experiments into one large experiment, the sample variance is $1.000586\sigma^2$, which has a p-value of 0.128243.

- For each m, we computed the *maximum Z-score* of the 100,000 samples (i.e., the maximum absolute value of the samples, scaled by the predicted standard deviation σ). These 75 maximum Z-scores had a high of 5.2130, a low of 4.0923, and a median of 4.5202. The corresponding p-values are 1/54, 0.986, and 0.461. Aggregating these 75 experiments into one large experiment, the highest Z-score has a p-value of 0.75181.

- For each m, we computed the *Anderson-Darling statistic* [2] of the 100,000 samples. Among these 75 statistics, the two smallest p-values were 1/1028 and 1/24, the largest was 0.98279, and the median was 0.521062.

6.3.2 Coefficient sizes in actual bootstrapping

For experimental evidence to justify heuristic step 1, we collected analogous statistics during runs of the actual bootstrapping routine. The w that we used for this purpose was the ring element v_1 that arises in making the ciphertext divisible by $p^{e'}$ (see discussion above just after (10)). The value of $|v_1 s|$ is really the most critical in the correctness of recryption.

For this experiment, we used 49 odd values of m, in the range 25,000 and 40,000, with between 2 and 4 prime-power factors. For each of these m's, we ran 250 trials, and in each trial we did the following:

1. Choose a random secret key s and compute the element v_1 of the bootstrapping process.

   The secret key was chosen with hamming weight $h = 120$, and the element v_1 was computed as in the "make divisible" operation (see Section 6.2) on a ciphertext with plaintext space $p = 2$.

2. Compute $x = v_1 s$, then choose one coefficient of x on the powerful basis at random, and output that coefficient.

That gave us 250 samples for each value of m, and we computed the same statistics as above to check if these samples are consistent with a normal random variable with variance $\sigma^2 = \sum_i \mathsf{Var}[e_i w_i]$ from Lemma 6.2:

- For each value of m, we calculated the fraction of the 250 samples that fell within 1, 2, and 3 times the predicted standard deviation σ. We got the following results:

	lowest m	predicted fraction	highest m
$1 \times \sigma$	0.612	0.682689	0.752
$2 \times \sigma$	0.912	0.954499	0.980
$3 \times \sigma$	0.988	0.997300	1.000

- For each m, we calculated the *sample variance* of the 250 samples, and compared it to the predicted variance σ^2. Of these 49 m's, the highest sample variance was $1.2472\sigma^2$, the lowest was $0.83913\sigma^2$, and the median was $0.99649\sigma^2$. The corresponding p-values are 1/207, 1/33, and 0.496316. Aggregating these 49 experiments into one large experiment, the sample variance is $0.9986248\sigma^2$, which has a p-value of 0.458818.

[5] The p value of a statistic is the probability of seeing this value under the distribution that we want to test for (i.e., normal with variance σ^2 in our case).

- For each m, we computed the *maximum Z-score* of the 250 samples (i.e., the maximum absolute value of the samples, scaled by the predicted standard deviation σ). These 49 maximum Z-scores had a high of 3.74436, a low of 2.51453, and a median of 2.95041. The corresponding p-values are 0.04421, 0.95010, and 0.54826. Aggregating these 49 experiments into one large experiment, the highest Z-score has a p-value of 0.89092.

- For each m, we computed the *Anderson-Darling statistic* [2] of the 250 samples. Among these 49 statistics, the smallest two p-values were 1/117, and 1/36, the largest was 0.977831, and the median was 0.519054.

6.3.3 Conclusions

The worst p-value we saw for any of the statistics we collected was 1/1028, which was for the Anderson-Darling statistic in the first test suite. As the Anderson-Darling p-values are themselves uniformly distributed, the probability of seeing such a low p-value among all $75 + 49 = 124$ such p-values (in both test suites) is about 1/9, which is not unreasonable.

Arguably, the most important statistic is the maximum Z-value, as this bears directly on the correctness of recryption. As we saw, the aggregate maximum Z-value for the first test suite had a p-value of 0.75181, and for the second, a p-value of 0.89092. These are very reasonable.

Finally, we went back to the values of m that gave rise to the smallest p-values that we saw, and wanted to test if there are any "algebraic reasons" that make these m's particularly bad. Hence we re-ran the tests on the values of m and got the following results:

- For the first suite of tests with a uniform w, the high sample variance (with p-value 1/306) occurred with $m = 32939$, which is a prime. We re-ran the experiment for that m, and got a sample variance of $1.00347 \cdot \sigma^2$, with p-value of 0.218725.

  The Anderson-Darling statistic for this test suite that had the smallest p-value (of 1/1028) came from a run with $m = 3 \cdot 5 \cdot 7 \cdot 13 \cdot 29$. We note that we generated random m's with replacement, and had that very same m appear twice more in our data set. For the other occurrences of this value of m, the corresponding Anderson-Darling p-values were 0.209769 and 0.694718.

- For the second suite of tests with w coming from actual bootstrapping, the p-value 1/207 in the empirical variance test occurred with $m = 3 \cdot 7 \cdot 23 \cdot 67$. We re-ran the experiment for that m and got a sample variance of $0.98939\sigma^2$, which has a p-value of 0.4645803.

  The 1/117 p-value for the Anderson-Darling statistic came from a run with $m = 7 \cdot 23 \cdot 199$. We re-ran this test and got a p-value of 0.184049.

These results seem to indicate that the low p-values for those m's are not due to algebraic reasons.

Summing up, we feel that the results of these experiments provide good evidence to justify our heuristic assumptions.

7 Implementation and Performance

As discussed in Section 4.2, our algorithms for the linear transformations rely on the parameter m having a fairly special form. Moreover, the analysis in Section 6 imposes even more restrictions on m (specifically, we must restrict to prime-power factorizations of m, rather than just pairwise-coprime factorizations as in Section 4.2). Luckily, there are quite a few such m's, which we found by brute-force search. We ran a simple program that searches through a range of possible m's (odd, not divisible by p, not prime). For each such m, we first compute the order d of p mod m. If this exceeds a threshold (we chose a threshold of 100), we skip this m. Next, we compute the factorization of m

into prime powers as $m = m_1 \cdots m_t$. We then find all indexes i such that p has order d mod m_i. If we find none, we skip this m; otherwise, we choose one such index (if there is more than one, we choose one that makes the linear transforms as fast as possible), and the other prime power factors are ordered arbitrarily.

cyclotomic ring m	21845 =257·5·17	18631 =601·31	28679 =241·7·17	35113 = 37·13·73
lattice dim. $\phi(m)$	16384	18000	23040	31104
plaintext space	GF(2^{16})	GF(2^{25})	GF(2^{24})	GF(2^{36})
number of slots	1024	720	960	864
recrypt params e/e'	12/4	16/9	12/4	12/4
before capacity	492	489	578	820
after capacity	174	237	252	495
min capacity	6.3	10.3	6.3	6.2
bits per level	15.2	15.4	15.8	15.9
usable levels	11	14	15	30
linear transforms (sec)	23	15	17	16
total recrypt (sec)	163	167	294	842
space usage (GB)	2.7	3.3	4.0	4.1

Table 1: Experimental results with plaintext space GF(2^d)

cyclotomic ring m	45551 =41·11·101	51319 =73·19·37	42799 = 337·127	42799 w/o Chen-Han	49981 =331·151	49981 w/o Chen-Han
lattice dim. $\phi(m)$	40000	46656	42336		49500	
plaintext space	GF(17^{40})	GF(127^{36})	$R(256, 21)$		$R(256, 30)$	
number of slots	1000	1296	2016		1650	
recrypt params e/e'	4/2	3/1	23/16		23/16	
before capacity	1019	1178	1092		1288	
after capacity	573	422	681	563	872	758
min capacity	7.5	9.2	10.3		10.3	
bits per level	19.7	22.1	23.1		23.1	
usable levels	28	18	29	23	37	32
linear transforms (sec)	29	36	60		75	
total recrypt (sec)	1584	3636	2146	1883	4034	3590
space usage (GB)	7.7	10.3	15.6		21.6	

Table 2: Experimental results with other plaintext spaces

For example, with $p = 2$, we processed all potential m's between 16,000 and 64,000. Among these, there were a total of 192 useful m's with $15,000 \leq \phi(m) \leq 60,016$, with a fairly even spread. So while such useful m's are relatively rare, there are still plenty to choose from. We ran this parameter-generation program to find potential settings for plaintext-space modulo $p = 2, p = 17$, $p = 127$, and $p^r = 2^8$, and manually chose a few of the suggested values of m for our tests.

For each of these values of m, p, r, we then ran a test in which we chose a random key, and performed recryption once per key. These tests were run on an Intel Xeon CPU E5-2698 v3 (Haswell architecture) at 2.30GHz. We ran our tests single-threaded, although `HElib` can also be run multi-threaded to obtain faster speeds. Tables 1 and 2 summarize the results from our experiments. We chose parameters so that the security level, taken from [20, Eqn.(8)], was 80 bits (or just a little more). For all tests we chose a hamming weight of 120 for the secret key (both the bootstrapping

cyclotomic ring m	21845 =257·5·17	18631 =601·31	28679 =241·7·17	35113 = 37·13·73
lattice dim. $\phi(m)$	16384	18000	23040	31104
plaintext space	GF(2)	GF(2)	GF(2)	GF(2)
number of slots	1024	720	960	864
recrypt params e/e'	12/4	16/9	12/4	12/4
before capacity	491	489	578	820
after capacity	247	298	329	580
min capacity	41.1	34.2	41.9	32.6
bits per level	15.2	15.4	15.8	15.9
usable levels	13	17	18	34
linear transforms (sec)	4	3	4	10
total recrypt (sec)	15	11	19	40
amortized time (ms)	1.1	0.9	1.1	1.4
space usage (GB)	1.8	1.8	1.6	3.5

Table 3: Experimental results with plaintext space $GF(2^d)$ – thin bootstrapping

cyclotomic ring m	45551 =41·11·101	51319 =73·19·37	42799 = 337·127	49981 =331·151
lattice dim. $\phi(m)$	40000	46656	42336	49500
plaintext space	GF(17)	GF(127)	$\mathbb{Z}_{256}$	$\mathbb{Z}_{256}$
number of slots	1000	1296	2016	1650
recrypt params e/e'	4/2	3/1	23/16	23/16
before capacity	1019	1178	1091	1288
after capacity	682	551	768	962
min capacity	42.9	67.3	48.8	48.6
bits per level	19.7	22.1	23.1	23.1
usable levels	32	21	31	39
linear transforms (sec)	17	19	19	27
total recrypt (sec)	66	125	130	173
amortized time (ms)	2.1	4.6	2.1	2.7
space usage (GB)	6.4	9.0	8.3	11.5

Table 4: Experimental results with other plaintext spaces — thin bootstrapping

key and the regular decryption key). For all but the $m = 21845$ and $m = 18631$ experiments, we chose the default parameter of 3 for the number of "columns" used in the "break into digits" logic of key switching; for $m = 21845$, we replaced 3 by 9, and for $m = 18631$, we replaced 3 by 5; these non-default settings trade increase security but reduce speed. This "columns" parameter is briefly described in the full version of [20].

In each table, the first row gives m and its factorization into primes. The first factor shows the value that was used in the role of m_1 (as in Section 4.2). The second row gives $\phi(m)$. The third row gives the plaintext space, i.e., the field/ring that is embedded in each slot (here, $R(p^r, d)$ means a ring extension of degree d over $\mathbb{Z}_{p^r}$). The fourth row gives the number of slots packed into a single ciphertext. The fifth row gives the recryption parameters e and e' that were used. The sixth row gives the capacity of the ciphertext just *after* we perform the homomorphic inner product, and just *before* we perform the first linear transformation. Here, capacity is defined as $\log_2(Q/\eta)$, where Q is the current modulus, and η is the current noise bound (measured as the ℓ_∞-norm of the canonical embedding).

The seventh row gives the capacity of the ciphertext after recryption. The eighth row gives the minimum capacity that a ciphertext can have before it is bootstrapped.[6] Thus, the difference between the "after capacity" and "min capacity" in the tables is the residual capacity that can be used to perform real work before we need to bootstrap again. The ninth row gives the "bits per level", which is the number of capacity bits consumed by one squaring operation (determined experimentally). Based on the data in rows 7–9, we compute the number of "usable levels", which is the number of squarings that can be performed between the end of one recryption operation and the start of the next one. The last three rows give the running time (in seconds) of the linear map operations, the total running time (in seconds) of one recryption operation, and the total memory (in gigabytes) used during recryption.

The Chen-Han Digit Extraction Procedure. Currently, `HElib` employs the Chen-Han optimization for digit extraction (cf. [5]) when working with plaintext moduli p^r with $r > 1$.[7] In Table 2, we ran the last two examples, which work with $p^r = 2^8$, either with the Chen-Han optimization (which is the default), or without. One can see that using Chen-Han slows things down a bit, but the noise control is better. By comparing the ratios of the usable levels to the ratios of the running times, one sees that in terms of *amortized* performance, Chen-Han is faster.

7.1 Thin bootstrapping

Tables 3 and 4 summarize the results from analogous experiments using the "thin bootstrapping" technique. Recall that for thin bootstrapping, the assumption is that each slot contains an element of the base field (or ring), rather than an extension field (or ring). `HElib` implements a variant of the technique for thin bootstrapping introduced in [5]. Details of this variant are given in [25]. Briefly, instead of d executions of the digit extracting routine, we only need one execution; moreover, there is no packing or unpacking (which saves a bit of noise, as we avoid a constant-multiplication), and the linear transformations are somewhat more efficient (as we do not need to use the more expensive BlockMatMul1D routine). Note that with this technique, one must perform one of the linear transformations *before* mod switching to the special bootstrapping modulus. Thus, we have added the estimated loss in capacity for this linear transformation (determined experimentally) to the minimum capacity at which a ciphertext should be bootstrapped. Thus, the difference between the "after capacity" and "min capacity" in the tables is still the "residual capacity" that can be used to perform "real work". We also added a row "amortized time", which measures the amortized time (in milliseconds) for the bootstrapping overhead associated per slot and per multiplication. This is computed by taking the total recryption time, and dividing by the number of slots times the number of usable levels.

7.2 Multi-threading

`HElib` supports multi-threading. When multi-threading is activated, the general strategy is to parallelize at the highest level possible. For example, in the bootstrapping routine, we have to perform d different digit extractions, and these can all be done parallel. Linear transformations can also be parallelized: to a large degree, the automorphisms that these transformations need to execute can be run in parallel. In the digit extraction routine (see Figure 1), the computations on lines 5 (for fixed k and $j = 0, \ldots, k$) can be run in parallel, and the cheaper computations on line 6 (for fixed k and $j = 0, \ldots, k$) can then be run sequentially. If none of these higher-level operations are parallelized,

[6] For this, we used a slightly more conservative bound than (13), with $\frac{2}{3}p^r(B^* + 0.5)$ on the right-hand side.

[7] The implementation employs a heuristic method, choosing Chen-Han when it appears that it should save on noise.

conversions between DoubleCRT and polynomial representation are parallelized: integer CRT operations are parallelized across coefficients, and FFT operations are parallelized across small primes. Based on our experiments, parallelizing at the highest level possible yields the best speedup.

Consider the $m = 21845$ bootstrapping example (see the first column of Table 1). On a single core, the running time was 163s. With 4 cores, the running time fell to 45s, and with 8 cores, the running time fell to 26s. So with 4 cores, we attain 90% of the potential speedup, and with 8 cores, we attain 78% of the potential speedup.

Next, consider the $m = 21845$ thin bootstrapping example (see the first column of Table 3). On a single core, the running time was 21s. With 4 cores, the running time fell to 7.9s, and with 8 cores, the running time fell to 5.9s. So with 4 cores, we attain 66% of the potential speedup, and with 8 cores, we attain 45% of the potential speedup. Thus, while we do get some speedup, it is not as effective for thin bootstrapping as it is for bootstrapping.

8 Why We Didn't Use Ring Switching

One difference between our implementation and the procedure described by Alperin-Sheriff and Peikert [1] is that we do not use the ring-switching techniques of Gentry et al. [18] to implement the tensor decomposition of our Eval transformation and its inverse. There are several reasons why we believe that an implementation based on ring switching is less appealing in our context, especially for the smaller parameter settings (say, $\phi(m) < 30000$). The reasoning behind this is as follows:

Rough factorization of m. Since the non-linear part of our recryption procedure takes at least seven levels, and we target having around 10 levels left at the end of recryption, it means that for our smaller examples we cannot afford to spend too many levels for the linear transformations. Since every stage of the linear transformation consumes at least half a level,[8] then for such small parameters we need very few stages. In other words, we have to consider fairly coarse-grained factorization of m, where the factors have sizes m^{ϵ} for a significant ϵ (as large as $\sqrt{m}$ in some cases).

Using large rings. Recall that the first linear transformation during recryption begins with the fresh ciphertext in the public key (after multiplying by a constant). That ciphertext has very low noise, so we have to process it in a large ring to ensure security.[9] This means that we must *switch up* to a much larger ring before we can afford to drop these rough factors of m. Hence we will be spending most of our time on operations in very large rings, which defeats the purpose of targeting these smaller sub-30000 rings in the first place.

We also note that in our tests, the recryption time is dominated by the non-linear part, so our implantation seems close to optimal there. It is plausible that some gains can be made by using ring switching for the second linear transformation, after the non-linear part, but we did not explore this option in our implementation. And as we said above, there is not much to be gained by optimizing the linear transformations.

9 Conclusions and Future work

In this report we described our implementation of bootstrapping in HElib, which can be made to run as fact as amortized 1.3 millisecond per bit. At this rate, we expect fully homomorphic encryption with bootstrapping to already be fast enough for some real applications.

One technical direction to explore is to try to find a better way to represent constants. In `HElib`, the most compact way to store constants in R_{p^r} is also the most natural: as coefficient vectors of

[8] Whether or not we use ring-switching, each stage of the linear transformation has depth of at least one multiply-by-constant, which consumes at least half a level in terms of added noise.

[9] More specifically, the key-switching matrices that allow us to process it must be defined in a large ring.

polynomials over $\mathbb{Z}_{p^r}$. However, in this representation, a surprisingly significant amount of time may be spent in homomorphic computations converting these constants to DoubleCRT format. One could precompute and store these DoubleCRT representations, but this can be quite wasteful of space, as DoubleCRT's occupy much more space than the corresponding polynomials over $\mathbb{Z}_{p^r}$. We may state as an open question: is there a more compact representation of elements of $\mathbb{Z}_{p^r}[X]$ that can be converted to DoubleCRT format in linear time?

Another, more challenging, direction is to find efficient routines to convert between different homomorphic encryption schemes. (In particular between the CKKS approximate number scheme and any of the packed fixed-point scheme such as BGV or B/FV.) While it is obvious that one can use bootstrapping for that purpose, we currently have no effective method for doing this. Some progress along these lines was made recently by Boura et al. [3], but their techniques essentially unpack each ciphertext of one scheme, and bootstrap each bit separately to pack it in the other scheme. A fully optimized cross-scheme bootstrapping is an intriguing possibility that we believe will find many practical applications.

References

[1] J. Alperin-Sheriff and C. Peikert. Practical bootstrapping in quasilinear time. In R. Canetti and J. A. Garay, editors, *Advances in Cryptology - CRYPTO'13*, volume 8042 of *Lecture Notes in Computer Science*, pages 1–20. Springer, 2013.

[2] T. W. Anderson and D. A. Darling. Asymptotic Theory of Certain "Goodness of Fit" Criteria Based on Stochastic Processes. *Ann. Math. Statist.*, 23(2):193–212, 06 1952.

[3] C. Boura, N. Gama, and M. Georgieva. Chimera: a unified framework for B/FV, TFHE and HEAAN fully homomorphic encryption and predictions for deep learning. *IACR Cryptology ePrint Archive*, 2018:758, 2018.

[4] Z. Brakerski, C. Gentry, and V. Vaikuntanathan. (Leveled) fully homomorphic encryption without bootstrapping. *ACM Transactions on Computation Theory*, 6(3):13, 2014.

[5] H. Chen and K. Han. Homomorphic lower digits removal and improved FHE bootstrapping. In *EUROCRYPT 2018*, volume 10820 of *Lecture Notes in Computer Science*, pages 315–337. Springer, 2018.

[6] J. H. Cheon, J. Coron, J. Kim, M. S. Lee, T. Lepoint, M. Tibouchi, and A. Yun. Batch fully homomorphic encryption over the integers. In *Advances in Cryptology - EUROCRYPT 2013, 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings*, pages 315–335, 2013.

[7] J. H. Cheon, K. Han, A. Kim, M. Kim, and Y. Song. Bootstrapping for approximate homomorphic encryption. In *EUROCRYPT 2018*, volume 10820 of *Lecture Notes in Computer Science*, pages 360–384. Springer, 2018.

[8] J. H. Cheon, A. Kim, M. Kim, and Y. Song. Homomorphic encryption for arithmetic of approximate numbers. In *Asiacrypt 2017*, volume 10625 of *Lecture Notes in Computer Science*, pages 409–437. Springer, 2017.

[9] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène. Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. In *ASIACRYPT 2016*, volume 10031 of *Lecture Notes in Computer Science*, pages 3–33. Springer, 2016.

[10] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène. Faster packed homomorphic operations and efficient circuit bootstrapping for TFHE. In *ASIACRYPT 2017*, volume 10624 of *Lecture Notes in Computer Science*, pages 377–408. Springer, 2017.

[11] J. Coron, T. Lepoint, and M. Tibouchi. Batch fully homomorphic encryption over the integers. *IACR Cryptology ePrint Archive*, 2013:36, 2013.

[12] J. Coron, D. Naccache, and M. Tibouchi. Public key compression and modulus switching for fully homomorphic encryption over the integers. In *Advances in Cryptology - EUROCRYPT 2012 - 31st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cambridge, UK, April 15-19, 2012. Proceedings*, pages 446–464, 2012.

[13] I. Damgård, V. Pastro, N. Smart, and S. Zakarias. Multiparty computation from somewhat homomorphic encryption. Cryptology ePrint Archive, Report 2011/535, 2011. `https://eprint.iacr.org/2011/535`.

[14] M. van Dijk, C. Gentry, S. Halevi, and V. Vaikuntanathan. Fully homomorphic encryption over the integers. In *Advances in Cryptology - EUROCRYPT 2010, 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques, French Riviera, May 30 - June 3, 2010. Proceedings*, pages 24–43, 2010.

[15] L. Ducas and D. Micciancio. FHE Bootstrapping in less than a second. Cryptology ePrint Archive, Report 2014/816, 2014. `http://eprint.iacr.org/`.

[16] C. Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the 41st ACM Symposium on Theory of Computing – STOC 2009*, pages 169–178. ACM, 2009.

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

[18] C. Gentry, S. Halevi, C. Peikert, and N. P. Smart. Field switching in BGV-style homomorphic encryption. *Journal of Computer Security*, 21(5):663–684, 2013.

[19] C. Gentry, S. Halevi, and N. Smart. Fully homomorphic encryption with polylog overhead. In *Advances in Cryptology – EUROCRYPT 2012*, volume 7237 of *Lecture Notes in Computer Science*, pages 465–482. Springer, 2012. Full version at `http://eprint.iacr.org/2011/566`.

[20] C. Gentry, S. Halevi, and N. Smart. Homomorphic evaluation of the AES circuit. In *Advances in Cryptology – CRYPTO 2012*, volume 7417 of *Lecture Notes in Computer Science*, pages 850–867. Springer, 2012. Full version at `http://eprint.iacr.org/2012/099`.

[21] C. Gentry, S. Halevi, and N. P. Smart. Better bootstrapping in fully homomorphic encryption. In *Public Key Cryptography – PKC 2012*, volume 7293 of *Lecture Notes in Computer Science*, pages 1–16. Springer, 2012.

[22] C. Gentry, A. Sahai, and B. Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In R. Canetti and J. A. Garay, editors, *Advances in Cryptology - CRYPTO 2013, Part I*, pages 75–92. Springer, 2013.

[23] S. Halevi and V. Shoup. Algorithms in HElib. In J. A. Garay and R. Gennaro, editors, *Advances in Cryptology – CRYPTO 2014, Part I*, pages 554–571. Springer, 2014. Long version at `http://eprint.iacr.org/2014/106`.

[24] S. Halevi and V. Shoup. Bootstrapping for HElib. In *EUROCRYPT 2015*, volume 9056 of *Lecture Notes in Computer Science*, pages 641–670. Springer, 2015.

[25] S. Halevi and V. Shoup. Faster homomorphic linear transformations in helib. In H. Shacham and A. Boldyreva, editors, *Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part I*, volume 10991 of *Lecture Notes in Computer Science*, pages 93–120. Springer, 2018.

[26] S. Halevi and V. Shoup. HElib - An Implementation of homomorphic encryption. `https://github.com/shaih/HElib/`, Accessed September 2014.

[27] K. Han, M. Hhan, and J. H. Cheon. Improved homomorphic discrete fourier transforms and FHE bootstrapping. *IEEE Access*, 7:57361–57370, 2019.

[28] K. Han and D. Ki. Better bootstrapping for approximate homomorphic encryption. *IACR Cryptology ePrint Archive*, 2019:688, 2019.

[29] J. Hoffstein, J. Pipher, and J. H. Silverman. NTRU: A ring-based public key cryptosystem. In J. Buhler, editor, *ANTS*, volume 1423 of *Lecture Notes in Computer Science*, pages 267–288. Springer, 1998.

[30] E. Juárez, R. Cortés-Maldonado, and F. Pérez-Rodriguez. Relationship between the inverses of a matrix and a submatrix. *Computación y Sistemas*, 20:251–262, 2016.

[31] A. López-Alt, E. Tromer, and V. Vaikuntanathan. On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption. In *STOC*, pages 1219–1234, 2012.

[32] V. Lyubashevsky, C. Peikert, and O. Regev. On ideal lattices and learning with errors over rings. *J. ACM*, 60(6):43, 2013. Early version in EUROCRYPT 2010.

[33] V. Lyubashevsky, C. Peikert, and O. Regev. A toolkit for ring-lwe cryptography. Cryptology ePrint Archive, Report 2013/293, 2013. `https://eprint.iacr.org/2013/293`.

[34] O. Regev. On lattices, learning with errors, random linear codes, and cryptography. *J. ACM*, 56(6), 2009.

[35] R. Rivest, L. Adleman, and M. Dertouzos. On data banks and privacy homomorphisms. In *Foundations of Secure Computation*, pages 169–177. Academic Press, 1978.

[36] K. Rohloff and D. B. Cousins. A scalable implementation of fully homomorphic encryption built on NTRU. 2nd Workshop on Applied Homomorphic Cryptography and Encrypted Computing, WAHC'14, 2014. Available at `https://www.dcsec.uni-hannover.de/fileadmin/ful/mitarbeiter/brenner/wahc14_RC.pdf`, accessed September 2014.

[37] S. Roman. *Field Theory*. Springer, 2nd edition, 2005.

[38] N. P. Smart and F. Vercauteren. Fully homomorphic SIMD operations. *Des. Codes Cryptography*, 71(1):57–81, 2014. Early verion at `http://eprint.iacr.org/2011/133`.