An Investigation of Complex Operations with Word-Size Homomorphic Encryption

Gizem S. Çetin[1], Yarkın Doröz[1], Berk Sunar[1], and William J. Martin[1]

Worcester Polytechnic Institute
{gscetin,ydoroz,sunar,martin}@wpi.edu

Abstract. Homomorphic encryption has progressed rapidly in both efficiency and versatility since its emergence in 2009. Meanwhile, a multitude of pressing privacy needs — ranging from cloud computing to healthcare management to the handling of shared databases such as those containing genomics data — call for immediate solutions that apply fully homomorpic encryption (FHE) and somewhat homomorphic encryption (SHE) technologies. Further progress towards these ends requires new ideas for the efficient implementation of algebraic operations on word-based (as opposed to bit-wise) encrypted data. While the arithmetic operations that are essential for cloud computing have known boolean representations, most of them require access to each bit individually, such as comparison and we lack solutions on a word domain. In this work, we tackle this challenging problem of finding ways to solve algebraic problems -including comparison and homomorphic division- in word-based encryption via an SHE scheme. We present concrete performance figures for all proposed primitives.

Keywords: Homomorphic encryption, word-size comparison, homomorphic division.

1 Introduction

A *fully homomorphic encryption scheme* (FHE scheme) is one which permits the efficient evaluation of any boolean circuit or arithmetic function on ciphertexts [20]. Gentry introduced the first FHE scheme [8, 9] in 2009; this lattice-based scheme was the first to support the efficient evaluation of arbitrary-depth boolean circuits. This was followed by a rapid progression of new FHE schemes (e.g., [24, 4, 23]). In 2010, Gentry and Halevi [10] presented the first actual FHE implementation along with a wide array of optimizations to tackle the infamous efficiency bottleneck of FHE schemes. Further optimizations for FHE which also apply to somewhat homomorphic encryption (SHE) schemes followed including batching and SIMD optimizations, e.g. see [11, 22, 12]. Nevertheless, bootstrapping [9], relinearization [5], and modulus reduction [5, 4] remain as indispensable tools for most HE schemes.

Most relevant to the present work, López-Alt, Tromer and Vaikuntanathan proposed SHE and FHE schemes (which we denote LTV) based on the Stehlé and Steinfeld variant of the NTRU scheme [23] that support inputs from multiple

public keys [18]. Bos et al. [1] introduced a variant of the LTV FHE scheme along with an implementation. The authors of [1] modify the LTV scheme by adopting a tensor product technique introduced earlier by Brakerski [3] thereby providing a security reduction to that of standard lattice-based problems. Their scheme affords enhanced flexibility by use of the Chinese Remainder Theorem on the message space and obviates the need for modulus switching. Doröz, Hu and Sunar propose another variant of the LTV scheme in [7], putting forward a batched, bit-sliced implementation that features modulus switching techniques.

With these improved primitives as a springboard, homomorphic encryption schemes have been used to build a variety of higher level security applications. For example, Lagendijk et al. [14] give a summary of homomorphic encryption and MPC techniques to realize key signal processing operations such as evaluating linear operations, inner products, distance calculation, dimension reduction, and thresholding. Meanwhile SHE tools, developed mainly to achieve FHE, have not been sufficiently explored for use in applications in their own right. In [19] for instance, Lauter et al. consider the problems of evaluating averages, standard deviations, and logistical regression which provide basic tools for a number of real-world applications in the medical, financial, and advertising domains. The same work also presents a proof-of-concept Magma implementation of an SHE scheme, offering basic arithmetic functionality, based on the ring learning with errors (RLWE) problem proposed earlier by Brakerski and Vaikuntanathan. Later, Lauter et al. show in [15] that it is possible to implement genomic data computation algorithms where the patients' data are encrypted to preserve patient privacy. The authors used a leveled SHE scheme which is a modified version of [17] where they omit of the costly relinearization operation. In [2] Bos et al. show how to privately perform predictive analysis tasks on encrypted medical data. These authors use the SHE implementation of [1] to provide timing results. Around the same time, Graepel et al. in [13] demonstrate that it is possible to homomorphically execute machine learning algorithms in a service while protecting the confidentiality of the training and test data. They, too, provide benchmarks for a small scale data set to show that their scheme is practical. Cheon et al. [6] present a method along with implementation results to compute encrypted dynamic programming algorithms such as Hamming distance, edit distance, and the Smith-Waterman algorithm on genomic data encrypted using a somewhat homomorphic encryption algorithm.

2 Motivation

With word size message domains we gain the ability to homomorphically multiply and add integers via simple ciphertext multiplications and additions, respectively. This significant gain comes at a severe price. We can no longer homomorphically compute comparisons via direct evaluation of a standard boolean comparator circuit, since the input bits are no longer accessible via our homomorphic evaluation operations. The same applies to more complex operations such as comparison evaluations, thresholding and division. Division, in particu-

lar, requires heavy computations and is challenging to evaluate in either bit or higher characteristic encryption. Therefore, it is commonly avoided by selecting division free algorithms or by postponing the computation to the client side after decryption whenever possible.

Our Contribution. In this work we present an array of solutions to improve the versatility of higher characteristic SHE/FHE schemes along with new abilities, specifically we

- We introduce convergence based iterative division and comparison algorithms and an algebraic technique for zero check and thresholding. Whenever the characteristic is large, the convergence algorithms perform better. This is due to the simple fact that convergence based algorithms work alongside residue number system based optimizations while algebraic techniques do not. Further, we introduce a new technique to perform constant division which is used to adjust the precision on-the fly. When the ciphertext is decrypted the *rounded* message is recovered.
- We introduce a technique to remove the excess bits of the numbers after homomorphic divisions and thresholding operations. Basically, we show how the numbers are effected by the noise levels, parity of the messages so that they round up or down when they are decrypted.
- We discuss the usage of residue number system (RNS) representation for proposed arithmetic operations to achieve a large message space. The algorithms that causes rounding operations (causing precision bits) on the messages are not recoverable since the error destroys the CRT operations.

3 Leveled FHE Background

An FHE scheme is an encryption method where one is capable of performing these two primitive operations: $\mathsf{Decrypt}(c_1 + c_2) = b_1 + b_2$ and $\mathsf{Decrypt}(c_1 \cdot c_2) = b_1 \cdot b_2$ (where c_i is the corresponding ciphertext of b_i). In general, all operations are performed over a ring of the form $\mathrm{R}_q = \mathbb{Z}_q[x]/\langle F(x)\rangle$ with a prime modulus q and an irreducible polynomial $F(x)$ of degree n. The schemes also specify an error distribution χ, typically a truncated discrete Gaussian distribution, for sampling random polynomials that are B-bounded. The term B-bounded means that the coefficients of the polynomial are selected in the range $[-B, B]$ according to distribution χ: for $g \leftarrow \chi$, we have $\|g\|_\infty \leq B$. There are usually four primitive functions, namely Keygen, $\mathsf{Encrypt}$, $\mathsf{Decrypt}$ and Eval. Among these, Eval involves homomorphic multiplication, which creates significant noise growth in ciphertexts and in order to cope with this, there are also several noise cutting operations.

For complex homomorphic operations, we will build homomorphic circuits involving only addition and multiplication operations, hence our proposed algorithms are not designed for a particular FHE scheme. However, there are some optimization techniques that are specific to LTV Scheme that we used in our experiments.

3.1 LTV Scheme

In 2012 López-Alt, Tromer and Vaikuntanathan proposed a leveled multi-key FHE scheme (LTV) [18]. The scheme is based on a variant of the NTRU encryption scheme proposed by Stehlé and Steinfeld [23]. The LTVscheme uses a new operation called *relinearization* and existing techniques such as modulus switching for noise control. Here we use a customized single-key version of LTVproposed by Doröz, Hu, and Sunar [7] along with key size reduction techniques.

Here we describe an instance L of the LTV encryption scheme. Once parameters n and q are chosen, and a polynomial $F(x)$ of degree n in $\mathbb{Z}_q[x]$ is fixed (e.g., $F(x) = x^n \pm 1$ or m^{th} cyclotomic polynomial with $\Phi(m) = n$), computations are performed in some ring $\mathcal{R}_q = \mathbb{Z}_q[x]/\langle F(x)\rangle$ of size q^n for each level i of the evaluation circuit. We write $\mathsf{L} = \mathsf{LTV}(p, q)$ where

- $\mathsf{L.Keygen}(n, B)$ first generates integers q, p, and B-bounded polynomials $f'(x) \leftarrow \chi$ and $g(x) \leftarrow \chi$ and then sets $f = pf' + 1$ and computes $h = pgf^{-1}$ in ring $\mathcal{R}_q$. When modulus switching is to be enabled, q is a decreasing sequence of moduli and evaluation keys are computed as $\zeta_\tau(x) = hs_\tau + pe_\tau + w^\tau f^2$ where $s_\tau(x), e_\tau(x) \leftarrow \chi$ and $\tau \in [\lfloor \log q_i / \log w \rfloor]$.
- $\mathsf{L.Encrypt}(m) = hs + pe + m$ where $s(x) \leftarrow \chi$ and $e(x) \leftarrow \chi$
- $\mathsf{L.Decrypt}(c) = \lceil fc \rfloor_q \bmod p$ reduces fc modulo q, balancing coefficients to lie between $-q/2$ and $q/2$ and then reduces the result modulo p
- $\mathsf{L.Relinearize}(c) = \sum_\tau \zeta_\tau(x) c_\tau(x)$ in $\mathcal{R}_q$ where $c(x) = \sum_\tau w^\tau c_\tau(x)$ expands ciphertext c as a combination of polynomials with all coefficients in $\{0, \ldots, w-1\}$; this operation is to be computed on each product $c(x)$.

There are two more ingredients we need for performance:

Modulus Switching. As indicated above, the noise scaling performance of this scheme can be improved dramatically by choosing a sequence of prime powers $q^{d+1} > q^d > \cdots > q$ and employing a different $q_i = q^{d+1-i}$ in each level i of our arithmetic circuit of depth d. With this special choice of moduli, the evaluation keys are promoted to the next level via modular reduction: $\zeta_\tau^{(i)}(x) = \zeta_\tau^{(0)}(x) \bmod q_i$. This technique drastically reduces the key storage requirement and obviates the need for key switching even for complex fixed-depth circuits. Applying modulus switching between levels as $\tilde{c}^{(i)}(x) = \lfloor \frac{q_i}{q_{i-1}} \tilde{c}^{(i-1)}(x) \rceil_p$ decreases the noise by $\log q$ bits by dividing and multiplying the new ciphertext with the previous and current moduli, respectively. In each level the arithmetic is performed over $\mathcal{R}_{q_i}$. The operation $\lfloor \cdot \rceil_p$ refers to rounding so as to match all parities.

Batching. We can also batch multiple messages into a single plaintext for parallel evaluations as proposed by Smart and Vercauteren [22, 11]. For this purpose, the specially selected polynomial $F(x)$ is factorized over $\mathbb{F}_p$ into equal degree polynomials $F_i(x)$ which define the message slots in which message bits are embedded using the Chinese Remainder Theorem. In this way, we can batch $\ell = n/t$ messages into one polynomial, where t is the smallest integer that satisfies $m|(p^t - 1)$.

4 Homomorphic Complex Arithmetic - Beyond Additions and Multiplications

In this section, we will focus on a number of operations that are essential for many machine learning algorithms such as division, zero test, thresholding and comparison. For homomorphic applications, the main goal is to represent any operation as a polynomial function so that they can be applied to encrypted inputs. implement parallel circuits for the mentioned procedures, but with complex operations such as division the evaluation of the circuit becomes too inefficient. Even though finding solutions to these problems is algebraically possible with word-based encryption, the challenge lies in finding low degree polynomials for efficiency. Therefore, in this work, we first focus on finding an algebraic solution to the proposed complex problems, then we go further and try to find more efficient approaches by using approximation algorithms. We start with finding the multiplicative inverse of a number, then by using the inverse function we will derive division and zero test, consequently equality check. After that, we will provide solutions to thresholding using zero test and other approximation techniques. Then using a thresholding approximation, we will have a look at the comparison problem. Additionally, we will propose a way for constant division which is especially useful to get rid of the excess bits in approximation/convergence algorithms and finally we describe how to use RNS in order to handle the overflow in these proposed methods.

4.1 Multiplicative Inverse and Division

One of the most difficult, and currently open, questions is how to implement homomorphic division efficiently. With bit-level encryption, one could implement a parallel division circuit by unrolling the shift and subtract operations. However the depth of this division circuit would be very high; the best we can do is to use a costly carry look ahead subtraction circuit and emulate a serial shift division algorithm with $\mathcal{O}(n \log(n))$ depth complexity. In the case of higher characteristic, we run into the aforementioned comparison and sign-detection problems.

The problem is as follows: Given two inputs a and b which are both defined in $\mathbb{Z}_p$, can we find a polynomial function, let it be $P(x, y)$, such that $P(a, b) = a/b$? This question can be reduced to this: Can we find a polynomial $P(x)$ such that $P(b) = 1/b$? Because if there exists such $P(x)$, then in order to find a/b, we can compute $aP(b) = a/b$. Furthermore the same polynomial can be used to execute a simple zero test which we will describe later. In this section we will construct such a polynomial using three different methods. The first one gives an exact algebraic solution, but works as a modular operation, whereas the next two use approximation algorithms and they output real results with respect to a preinitialized precision.

Fermat's Little Theorem We can obtain a polynomial function via Fermat's Little Theorem that permits homomorphic evaluation of the multiplicative inverse b^{-1} of a number b, modulo p. Note that a generalization of Fermat's Little

Theorem states that $b^{\alpha} \equiv b^{\beta} \mod p$ as long as $\alpha \equiv \beta \mod \phi(p)$, where b and p are coprime. If we pick p a prime, b can be any number from $\mathbb{Z}_p$ and it is known that $\phi(p) = p - 1$. It follows that $b^{-1 \mod \phi(p)} = b^{p-2} \mod p$. Hence we define $P(x) = x^{p-2}$ and $P(x)$ is defined over $\mathbb{Z}_p$.

Lemma 1 (Modular Inverse and Modular Division). *Let* $\mathsf{L} = \mathsf{LTV}(p, q)$ *where* $p \in \mathbb{Z}$ *is prime. For* $c = \mathsf{L.Encrypt}(b)$*, we compute* $\tilde{c} = c^{p-2}$*. Then* $\mathsf{L.Decrypt}(\tilde{c}) = b^{-1} \mod p$*. For* $c_1 = \mathsf{L.Encrypt}(a)$*,* $c_2 = \mathsf{L.Encrypt}(b)$*, if we compute* $\tilde{c} = c_1 c_2^{p-2}$*, then* $\mathsf{L.Decrypt}(\tilde{c}) = ab^{-1} \mod p$*.*

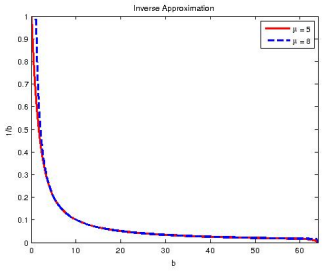
As we have a polynomial of degree $p - 2$, this method is not very efficient due to the fact that we have to compute a homomorphic exponentiation of multiplicative depth $\mathcal{O}(\log(p))$. Unless p is small without further customization this approach will not be very practical. Additionally note that this method does not provide a multiplicative inverse over real numbers since this is a modular operation. On the bright side, the output is an exact arithmetic solution, i.e there is no approximation, no fractions or precisions to handle. In the next approach we will find the reciprocal without a modulo p, as a real number using a root finding algorithm.

Newton's Root Finding Algorithm We can find the multiplicative inverse of any arbitrary number b using Newton's root finding algorithm. The function $f(z) = 1/z - b$ has a root at $z = 1/b$, hence if we can find the root of $f(z)$, we obtain the reciprocal for b. Iterations start with an initial guess z_0 and follows by finding $z_{i+1} = z_i - \frac{f(z_i)}{f'(z_i)} = z_i\,(2 - bz_i)$. Assuming b has a range of $[0, 2^k]$, we set the initial value z_0 to 2^{1-k} and fix the number of iterations to μ, the approximation can be seen in Figure 1a.

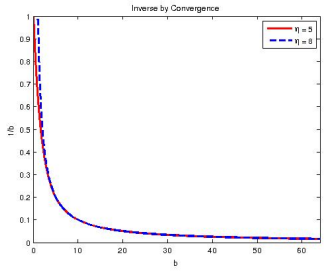
In homomorphic applications, we deal with integers, therefore we need to represent fractions using an initially chosen precision value. Since we initialized $z_0 = 2^{1-k}$, we need at least $k - 1$ bits in order to cover the fractional part. If we use a fixed constant $\rho = 2^{k-1}$ and let $\bar{z_0} = z_0\rho = 1$, we can represent our initial guess as an integer. In the following iteration steps, whenever there is an addition/subtraction we need to re-arrange the precision point so that both operands are represented with respect to the same precision, similarly whenever there is a multiplication we need to allow the precision point to move to the left so that we can mimic the fractional multiplication. For instance, in order to compute z_1 we also need to multiply the constant 2 in the equation with ρ, so that the precision points of 2 and $b\bar{z_0}$ are aligned. Let the first iteration result be $\bar{z_1}$, then $\bar{z_1} = \bar{z_0}\,(2\rho - b\bar{z_0}) = z_0\rho\,(2\rho - bz_0\rho) = z_1\rho^2$. Similarly in the next iteration, we need to align the precision point of constant 2 with the precision point of product $b\bar{z_1}$. Hence we have $\bar{z_2} = \bar{z_1}\left(2\rho^2 - b\bar{z_1}\right) = z_1\rho^2\left(2\rho^2 - bz_1\rho^2\right) = z_2\rho^4$. Notice that with one iteration step, we doubled the number of fractional bits since we executed a multiplication. Finally, at the end of each iteration we will have $\bar{z_{i+1}} = \bar{z_i}\left(2\rho^{2^i} - b\bar{z_i}\right) = z_i\rho^{2^i}\left(2\rho^{2^i} - bz_i\rho^{2^i}\right) = z_{i+1}\rho^{2^i}$.

Lemma 2 (Approximate Inverse). *Let* $\mathsf{L} = \mathsf{LTV}(p, q)$ *and* $c = \mathsf{L.Encrypt}(b)$. *We compute* $c_{i+1} = c_i \left(2\rho^{2^i} - c_i c\right)$, *where* $\rho = 2^{k-1}$ *with an integer* k *with* $b \in [0, 2^k]$ *and* $c_0 = 1$. *Then for a chosen number of iterations* μ, $\mathsf{L.Decrypt}(c_\mu) \approxeq (1/b)\,\rho^{2^{\mu-1}} \mod p$. *Let* $d = \mathsf{L.Encrypt}(a)$, *we compute* $\tilde{d} = dc_\mu$. *Then* $\mathsf{L.Decrypt}(\tilde{d}) \approxeq (a/b)\,\rho^{2^{\mu-1}} \mod p$.

The depth of this approximation depends on the number of iterations μ, i.e. it is independent of p. Consider the equation $c_{i+1} = c_i 2\rho^i - c_i^2 c$, the depth of the function comes from the product $c_i^2 c$. Initially c_0 is a constant, hence the exponent of c in c_1 becomes 1. In the next iterations, the exponent of c will be doubled by the square operation c_i^2, plus one increased due to the multiplication with c, hence the exponents will increase as $3, 7, \cdots$. Thus, after μ iterations, the exponent will be $2^\mu - 1$ and the depth is μ. This gives a great advantage over the first scheme where we use Fermat's Little Theorem, when the inputs come from a small subset of the plaintext space (assuming $\mu < \log(p)$). Note that the algorithm is flexible in the sense that we can keep iterating to increase the precision, or terminate early if less precision suffices for the application. Once the iterations have been completed, the precision has changed where the least significant $\log(\rho^{2^\mu})$ bits of the result represent the desired reciprocal. This means that any further computation requires other operands that will interact with the reciprocal need to be shifted to align with the segment representing the fractional part. On the other hand, the down-side of this algorithm is, since we need to fix the precision variable ρ, we need to have an upper limit for input b. The next algorithm finds an approximate reciprocal using a convergence algorithm.



(a) Using Newton's root finding algorithm, where $b \in [0, 64]$, $z_0 = 1/32$ and $\mu \in \{5, 8\}$.



(b) By convergence, where $b \in [0, 64]$ and $\eta \in \{5, 8\}$.

Fig. 1: Multiplicative inverse approximation function $P(x) = 1/x$

By Convergence We briefly and informally describe how to find the inverse by convergence as follows: Assume we want to compute the reciprocal $1/b$. The

algorithm works by multiplying both the numerator and denominator by a series of values $r_0, r_1, \ldots$ so as to make the denominator converge to a 1. Thus, at the end of the computation the numerator yields the desired division result:

$$\frac{1}{b} = \frac{1}{b} \cdot \frac{r_0}{r_0} \cdot \frac{r_1}{r_1} \cdots \frac{r_\eta}{r_\eta}, \qquad b \cdot r_0 r_1 \cdots r_\eta \to 1 \ .$$

The standard approach starts by normalizing 1 and b to become fractions in the unit interval, in particular $b \in [\frac{1}{2}, 1)$. Then we can write $z = 1 - b$ where $z \in \left[0, \frac{1}{2}\right]$. Then setting $r_0 = 1 + z$, $r_1 = 1 + z^2$, $\ldots$, $r_i = 1 + z^{2^i}$ will yield the desired result. We can show that $b \cdot r_0 \in [1 - 2^{-2}, 1]$, $b \cdot r_0 r_1 \in [1 - 2^{-4}, 1]$, $b \cdot r_0 r_1 r_2 \in [1 - 2^{-8}, 1]$, etc., with products $b \cdot r_0 \cdots r_\eta$ converging to one. The approximated inverse values for different η can be seen in Figure **??**.

Given the encrypted value of b, we can mimic the division by convergence algorithm to effect a homomorphic division operation. As in the last method, we are not able to multiply fractions, hence we need to allow the products of integers r_0, $r_0 r_1$, $r_0 r_1 r_2$, $\ldots$ to grow. We again view this as allowing the precision point to move to the left with each multiplication. Another restriction is, we need to roughly know the magnitude of b, i.e. assume $b \in \left[2^{\ell-1}, 2^\ell\right)$, then we need to normalize b with the constant $\sigma = 2^\ell$, so that b/σ is in the $[\frac{1}{2}, 1)$ interval. This means that b/σ will have ℓ fractional bits and in order to represent the fraction as an integer we need to shift it to the left for ℓ times. Thus we have $\bar{b} = 2^\ell b/\sigma = b$. We also need to normalize the numerator, so it becomes $1/\sigma$. Here, this fractional multiplication can be left to after decryption, since it is a constant. But if it is needed to be done before decryption we need to represent it as an integer, hence we need to left shift it ℓ times. Now, instead of setting $z = 1 - b/\sigma$, we align the precision point in constant 1 and set $\bar{z} = \sigma - b$. Similarly for each iteration step, we need to set $\bar{r_0} = \sigma + \bar{z}$, $\bar{r_1} = \sigma^2 + \bar{z}^2$ and in general $\bar{r_i} = \sigma^{2^i} + \bar{z}^{2^i}$. Assume we need η iterations, then our polynomial becomes $P(x) = \sigma \prod_{i=0}^{\eta} \left(\sigma^{2^i} + (\sigma - x)^{2^i}\right)$.[1] The grow of the fractional part can be computed as $\sigma^{1+\sum_{i=0}^{\eta} 2^i} = \sigma^{2^{\eta+1}}$.

For the most significant ℓ bits to stabilize, we need $\log(\ell) + \log\log(\ell) = \mathcal{O}(\log(\ell))$ iterations which also represents the depth of the computation. Now if we cannot estimate the magnitude of b, due to repeated squaring the power of z will double in precision in every iteration moving bit by bit closer to the end of the precision window[2]. Therefore, we will need another $\mathcal{O}(\ell)$ iterations for the denominator to reach $2^{i\ell-1} \leq b r_0 r_1 \ldots r_i < 2^{i\ell}$. In practice the number of iterations required by the division by convergence algorithm will depend on the distribution of the data. For uniformly distributed data of precision ℓ, the expected value of the deviation in the magnitude will be in the order of $2^{\mathcal{O}(\log(\ell))}$.

[1] Notice that the first σ in the equation comes from the numerator and it can be handled after decryption.

[2] Note that in the special case of a constant division we can always finish the result in $\log(b)$ iterations. Therefore we can very efficiently divide by small constants with compact representation.

Therefore, the average case and worst case complexities of the division by convergence algorithm are in the order of $\mathcal{O}(\log(\ell))$ and $\mathcal{O}(\ell)$, respectively.

Lemma 3 (Inverse by Convergence). *Let* $\mathsf{L} = \mathsf{LTV}(p, q)$ *and* $c = \mathsf{L.Encrypt}(b)$. *We compute* $\tilde{c} = \prod_{i=0}^{\eta} \left(\sigma^{2^i} + (\sigma - c)^{2^i}\right)$, *for a chosen number of iterations* η, *which depends on the predetermined precision factor* ℓ, *with inputs* $b \in [0, 2^\ell]$ *and* $\sigma = 2^\ell$. *We have* $\mathsf{L.Decrypt}(\tilde{c}) \approxeq (1/b)\,\sigma^{2^{\eta+1}-1} \mod p$. *Let* $d = \mathsf{L.Encrypt}(a)$, *we compute* $\tilde{d} = d\tilde{c}$. *Then* $\mathsf{L.Decrypt}(\tilde{d}) \approxeq (a/b)\,\sigma^{2^{\eta+1}-1} \mod p$.

The degree of the polynomial $P(x)$ becomes $\sum_{i=0}^{\eta} 2^i = 2^{\eta+1} - 1$. This means that the depth of this method is $\log\left(2^{\eta+1} - 1\right) = \eta + 1$. As in the previous approximation method, this is also independent from encryption factor p. But both algorithms suffer from the growth in the fractions, i.e. p should be large enough to cover the magnitude of the end result in order to avoid overflows. Even with small precision, after a few iteration steps we end up with a large fraction. This is a generic problem in any approximation based algorithm where we have to use real numbers. Thus, later in Section 5.2, we propose a method to make these schemes, that require large p, more practical using RNS. We propose another solution, where we describe a homomorphic constant division method, so that we can adjust the precision before decryption. This method lets p size to decrease on the fly, hence there are advantages and disadvantages that will be discussed later in Section 5.1.

From now on, whenever we use the approximation based algorithms we will refer the excess bits as ω in general. For instance, $\omega = \sigma^{2^{\eta+1}-1}$ if we use the convergence based inverse algorithm, similarly $\omega = \rho^{2^{\mu-1}}$ when Newton's root finding method is used. For the sake of generality, when we use an algebraic method, i.e. when there is no excess bit, we will take $\omega = 1$. Additionally, whenever we refer to $P(x)$, it is the generic polynomial that works as an inverse function, i.e. $P(x = b) = \omega/b$.

4.2 Zero Test and Equality Check

We can obtain a polynomial function that permits homomorphic evaluation of a zero test. The test returns a zero or one depending on whether the encrypted message content holds a zero or not. Let this polynomial be $Z(x)$, then we want to have $Z(a) = 0$ if a is equal to zero, $Z(a) = 1$ otherwise. We can retrieve this functionality using Fermat's Little Theorem by computing $x^{p-1} \mod p$. This can be interpreted as multiplying the input x with its inverse modulo p, which is $x^{p-2} \mod p$. Inspired by the same idea, we can create a zero test polynomial by using any inverse polynomial as follows: $Z(x) = xP(x)$. Then the output will give us a 0 or ω depending on the chosen inverse finding method.

The zero test may be used trivially to homomorphically perform an equality check on two messages a, b by computing $Z(a - b)$. Note that this is a much simpler operation than magnitude comparison which we will address later in Section 4.3.

Lemma 4 (Zero Test and Equality Check). *Let* $\mathsf{L} = \mathsf{LTV}(p, q)$ *and* $c = \mathsf{L.Encrypt}(b)$. *We compute* $\tilde{c} = cP(c)$. *Then* $\mathsf{L.Decrypt}(\tilde{c}) = 0$ *if* $b = 0 \mod p$ *and* $\mathsf{L.Decrypt}(\tilde{c}) \approxeq \omega$ *if* $b \neq 0 \mod p$. *Let* $c_1 = \mathsf{L.Encrypt}(a)$ *and* $c_2 = \mathsf{L.Encrypt}(b)$, *then if we compute* $c = c_1 - c_2$ *and* $\tilde{c} = cP(c)$, *we will retrieve* $\mathsf{L.Decrypt}(\tilde{c}) = 0$ *if* $a = b \mod p$ *and* $\mathsf{L.Decrypt}(\tilde{c}) \approxeq \omega$ *if* $a \neq b \mod p$.

The degree of $Z(x)$ is always one more than the degree of $P(x)$. Thus, the complexity of a zero test depends on the underlying inverse polynomial. Due to the same reason, the zero test also suffers from the same problems of the chosen inverse method.

4.3 Thresholding and Comparison

Using the zero test we can compute thresholding operations easily albeit inefficiently. Assume we want to homomorphically evaluate the check $b \leq t$ for some data m and threshold $t \in \mathbb{Z}_p$. As earlier we are given the encryption of b while t is presumed available as cleartext and again we are seeking a polynomial to represent this operation. Let it be $T(x, t)$, then we want $T(a, t) = 0$ when $a < t$ whereas $T(a, t) = 1$ otherwise. We can devise this algorithm by testing the equality over the range of integers $i = 0, \ldots, t-1$ and aggregate the result as[3] $T(x, t) = \sum_{i \in [t]} (\omega - Z(x - i))$ where $Z(x)$ is a zero test polynomial that is described in the previous section. Clearly, we can instead compute the complement if t is closer to p than to 0.

If we compute it using Fermat's Little Theorem, although it is not efficient, this presents a viable and exact technique for evaluating thresholds. A significant positive aspect of the formulation is that the multiplicative depth of the threshold computation is independent of the threshold constant t and is the same as the depth of an equality check: $\mathcal{O}(\log(p))$. On the other hand, the summation becomes computationally expensive — with complexity $\mathcal{O}(t \log(p))$ — as p and the range of t grow. Lookup tables and selection of special moduli can be used to increase the efficiency.

Unless p is small without further customization this approach will not be very practical. To gain some economy over the prime p case, we may chose p to be highly composite $p = \prod_{i \in [k]} p_i$ in such a way that the zero test simply becomes $c^{\phi(p)} = c^{\prod \phi(p_i)}$. Then the multiplicative depth complexity of a zero test (or comparison) becomes $\sum_{i \in [k]} \log(p_i - 1)$.

Approximation Methods In order to retrieve a threshold polynomial $T(x)$, we will make use of the Unit Step Function, i.e $H(x) = 0$ when $x < 0$ and $H(x) = 1$ when $x > 0$, then we can just compute $T(x) = H(x - t)$ where t is a fixed cleartext threshold. Furthermore, the same polynomial can be used to compare two encrypted values a, b by computing $H(a - b)$. We propose two different methods to create a step function.

[3] Since the zero tests are exclusive, we may aggregate the result using a standard homomorphic addition operation instead of a boolean OR.

For the first approach we will make use of logistic function and the equation is given as follows, $H(x) = \lim_{k \mapsto \infty} \frac{1}{1+e^{-2kx}} \cong (e^x)^{2k} \left(1 + (e^x)^{2k}\right)^{-1}$. By limiting k to a small constant, we can get a smooth approximation and we can use Taylor Series approximation to compute the exponential function $e^x \cong \sum_{i=1}^{\infty} \frac{x^i}{i!}$, and we can also use one of the inverse functions that we found in Section 4.1. Thus $H(x)$ becomes: $H(x) \cong \left(\sum_{i=1}^{\infty} \frac{x^i}{i!}\right)^{2\kappa} P \left(1 + \left(\sum_{i=1}^{\infty} \frac{x^i}{i!}\right)^{2\kappa}\right)$. Even though we can get a threshold polynomial using this approach, it is computationally expensive considering the input to the inverse function has already a large exponent. Therefore, we use another approach which is constructing a square wave using sine waves. Square wave function $S(x)$ can be approximated as, $S(x) \cong \sum_{i=1}^{\infty} \frac{\sin((2i-1)x)}{(2i-1)}$ For sinus values we can use the approximation, $\sin(x) \cong \sum_{j=1}^{\infty} \frac{(-1)^{j-1} x^{2j-1}}{(2j-1)!}$. Embedding this in the previous equation we will have, $S(x) \cong \sum_{j=1}^{\infty} \sum_{i=1}^{\infty} \frac{(-1)^{j-1}(2i-1)^{2j-2}}{(2j-1)!} x^{2j-1}$

The output of the square wave function is in the range of $[-0.8, 0.8]$ in a period, thus we compute $H(x)$ as: $\frac{S(x)+0.8}{1.6}$. The degree of $H(x)$ depends on the limit of j. If we define $i \in [1, \alpha]$ and $j \in [1, \beta]$, then the largest exponent of input x, thus the degree of H, becomes $2\beta - 1$. Consequently, the depth of the approximation algorithm becomes $\lceil \log(2\beta - 1) \rceil = \log \beta + 1$. For different values of α and β the unit step approximation can be seen in Figure 2.

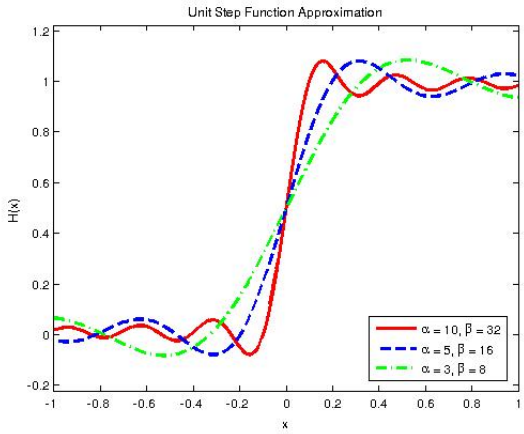


Fig. 2: Unit step function $H(x)$ for various approximation degrees.

To make use of this approximation algorithm, we also need to associate message space elements to discrete samples of the input range $[-1, 1]$ of $H(x)$. Assume we handle elements of precision ℓ bits and we want to find $H(b-t)$, where $b, t \in [0, 2^\ell)$. Then we have an input $x = b - t \in \left(-2^\ell, 2^\ell\right)$ and we have to nor-

malize it with $\omega = 2^\ell$, so that the normalized value lies in the input range, i.e. $x/\omega \in (-1, 1)$. As in the previous approximation methods, we need to represent ℓ fractional bits with a binary point placed right after leaving a single bit for the integer part. During evaluation we need to keep track of the precision point which moves to the left, with each multiplication by x. Once the evaluation is completed the approximation result resides in the most significant precision bit(s) ready to be used for subsequent evaluation and the maximum number of fraction bits can be found in the term with the highest exponent, $\omega^{2\beta-1}$.

4.4 Square Root

We can find an approximation to the square root of a number by using a root finding algorithm. As before, we seek a polynomial, let it be $R(x)$, such that $R(b) = \sqrt{b}$. The function $f(y) = y^2 - b$ has a root at $y = \sqrt{b}$, hence if we can find the root of $f(y)$, we obtain the square root of b. If we use Newton's Root Finding method as in Section 4.1, we can iterate the values $y_{i+1} = y_i - \frac{f(y_i)}{f'(y_i)} = \frac{1}{2}\left(y_i + \frac{b}{y_i}\right)$ with an initial guess of y_0. For the inverse computation $\frac{b}{y_i}$, we can use the inverse approximation polynomial that we retrieved before, $y_{i+1} = \frac{1}{2}\left(y_i + bP(y_i)\right)$. In order to handle fractions, again we need to consider an imaginary precision point. The depth of the algorithm depends on the number of iterations, let it be κ, then total depth will be κ times the depth of the inverse computation $P(x)$. Thus, we can conclude that this is a relatively much more costly operation.

5 Making Complex Arithmetic More Practical

As mentioned before, most of the proposed methods require a large p. By increasing the size of p, we introduce a high noise growth in the ciphertexts. As a consequence, this leds to the use of larger coefficient size for the ciphertexts, i.e. the ring $\mathcal{R}_q = \mathbb{Z}_q[x]/\langle F(x)\rangle$ with a larger q. Increasing q size affects security, hence this leads to the use of a cyclotomic polynomial with a larger degree, i.e. even a larger ring $\mathcal{R}_q$. So even though, increasing message space gives us freedom of handling inputs from a much higher characteristic, it also comes with the efficiency problems. In this section we propose two independent methods to make the arithmetic with large p more practical.

5.1 Constant Division - Adjusting the Precision

Here we introduce a technique that may be used to remove excess bits (at decryption) after division and thresholding operations. We consider $\mathsf{L} = \mathsf{LTV}(p, q)$ with private key $f = pf' + 1$ and public key $h = pgf^{-1}$ and we will examine decryption using $\tilde{\mathsf{L}} = \mathsf{LTV}(\tilde{p}, q)$ where $\tilde{p}|p$ and $\tilde{\mathsf{L}}$ uses the same private key f (along with the same n and q) as does L.

Lemma 5 (Constant Division). *Suppose the plaintext m is LTV-encrypted using L as $c = c(x) = hs + pe + m$ so that $\mathsf{L.Decrypt}(c) = m$. Suppose $p = d \cdot \tilde{p}$ and $q \equiv 1 \pmod{p}$. Let $u = gs + fe + f'm$ and write $m = d \cdot \tilde{m} + r$ where $0 \leq m_i < p, 0 \leq \tilde{m}_i < \tilde{p}, 0 \leq r_i < d$. Then, as long as $\|u\|_\infty \leq (q - 1 - 2p)/2p$, the scaled ciphertext $\tilde{c} = d^{-1} \cdot c$ in $\mathcal{R}$ satisfies $\tilde{\mathsf{L}}.\mathsf{Decrypt}(\tilde{c}) = \sum_{i=0}^{n-1} \hat{m}_i x^i$ where*

$$\hat{m}_i = \begin{cases} \tilde{m}_i, & \text{if } 0 \leq r_i \leq d/2 + \frac{u_i}{2\|u\|_\infty}; \\ \tilde{m}_i + 1, & \text{otherwise.} \end{cases}$$

When decrypted we obtain our results with reduced precision afforded by $\tilde{p}$. However, we can perform deeper computations with as much precision allowed by $d\tilde{p}$. We may chose to divide the message by any divisor s of d by multiplying it with $s^{-1} \in \mathbb{Z}_q$.

5.2 Using RNS with Approximation Algorithms

As shown in Sections 4.1 and 4.3 we can efficiently compute divisions and approximate thresholds using convergence. While asymptotically efficient, both require many levels of multiplication and a large message space, i.e. p, to prevent overflow. This is where the residue number system (RNS) can make a significant difference. Since both algorithms use only constant scaling, additions and multiplication operations and therefore can be used in conjunction with RNS representation. For this, we create parallel LTV encryptions of the same message by computing its residues using a set of distinct prime moduli $p_1, p_2, \ldots, p_k$. The product $p = \prod p_i$ should be large enough to contain the result even after division or thresholding and any subsequent evaluations. This creates k parallel evaluation paths where the same evaluation is performed including any divisions and threshold computations. The resulting ciphertexts are decrypted individually. The result is recovered using CRT. With this approach noise growth can be curbed and parameter sizes can be kept in a reasonable range. Finally, we note that the prevision adjustment technique *cannot* be used along with RNS since CRT cannot recover from rounding errors that occur during decryption.

6 Implementation Results

We implemented the proposed division, zero test, thresholding and comparison algorithms using the leveled single key LTV scheme using Shoup's NTL library version 9.0 [21] compiled with the GMP 5.1.3 package. Our simulations are performed on an Intel Xeon @ 2.9 GHz server running Ubuntu Linux 13.10. Note that the proposed homomorphic algebraic operations in Section 4, are generic, i.e. they can be implemented using any FHE scheme that supports word size encryption, but the optimizations defined in Section 5 are LTV-specific. For parameter selection we utilized the two Hermite factor analysis using the formula in [16], i.e. $1.8/\log\delta - 110$. We used modulus polynomial $x^n - 1$ for our message embedding. For the first test, we picked $p = \{17, 257\}$ and $n = 16384$

and applied Fermat's Little Theorem for a zero test. In this scenario, we have a single message, because for packing multiple data with degree n we need a much larger p value for NTT operations. For the second test we used 20-bit p values, we first applied Newton's Root Finding method $\mu = 5$ and inputs in the range $[0, 64]$ and we used RNS method with 4 different p values, because p needs to be larger than $2^{(k-1)\left(2^{\mu-1}\right)} = 2^{80}$. Secondly, we applied division by convergence algorithm for $\eta = 5$ and inputs in the range $[0, 64]$, in this case we used 20 p values, because p must be larger than $2^{\ell\left(2^{\eta+1}-1\right)} = 2^{378}$. For the last test, we computed a comparison with inputs in the range $[0, 32]$ and $\alpha = 5$, $\beta = 16$. For relinearization and evaluation keys, we use a block size of $\omega = 2^{\log(q)/2}$ where q is the noise cutting factor for each level $i = 0, \cdots, d-1$.

	d	n	$\log q$	**Total Time**	**Amortized**
Zero Test	4	16384	30	4 sec	N/A
	8	16384	45	9.09 sec	N/A
Division	5	16384	80	36 min	130 msec
	6	16384	100	1.18 h	259 msec
Comparison	5	16384	80	27 min	90 msec

Table 1: Zero Test using Fermat's Little Theorem with a single message, Division first using root finding, then convergence algorithm for multiple packed data and finally comparison using Square Wave approximation for multiple packed data.

References

1. Bos, J.W., Lauter, K., Loftus, J., Naehrig, M.: Improved security for a ring-based fully homomorphic encryption scheme. In: Stam, M. (ed.) Cryptography and Coding, Lecture Notes in Computer Science, vol. 8308, pp. 45–64. Springer Berlin Heidelberg (2013)
2. Bos, J.W., Lauter, K., Naehrig, M.: Private predictive analysis on encrypted medical data. Journal of biomedical informatics 50, 234–243 (2014)
3. Brakerski, Z.: Fully homomorphic encryption without modulus switching from classical gapSVP. IACR Cryptology ePrint Archive 2012, 78 (2012)
4. Brakerski, Z., Gentry, C., Vaikuntanathan, V.: Fully homomorphic encryption without bootstrapping. Electronic Colloquium on Computational Complexity (ECCC) 18, 111 (2011)
5. Brakerski, Z., Vaikuntanathan, V.: Efficient fully homomorphic encryption from (standard) LWE. In: Ostrovsky, R. (ed.) FOCS. pp. 97–106. IEEE (2011)
6. Cheon, J., Kim, M., Lauter, K.: Homomorphic computation of edit distance. In: Brenner, M., Christin, N., Johnson, B., Rohloff, K. (eds.) Financial Cryptography and Data Security, Lecture Notes in Computer Science, vol. 8976, pp. 194–212. Springer Berlin Heidelberg (2014)
7. Doröz, Y., Hu, Y., Sunar, B.: Homomorphic AES evaluation using the modified LTV scheme. Designs, Codes and Cryptography pp. 1–26 (2015)

8. Gentry, C.: A Fully Homomorphic Encryption Scheme. Ph.D. thesis, Stanford University (2009)
9. Gentry, C.: Fully homomorphic encryption using ideal lattices. In: Proceedings of the Forty-first Annual ACM Symposium on Theory of Computing. pp. 169–178. STOC '09, ACM (2009)
10. Gentry, C., Halevi, S.: Implementing Gentry's fully-homomorphic encryption scheme. In: Paterson, K.G. (ed.) Advances in Cryptology–EUROCRYPT 2011. Lecture Notes in Computer Science, vol. 6632, pp. 129–148. Springer (2011)
11. Gentry, C., Halevi, S., Smart, N.P.: Fully homomorphic encryption with polylog overhead. IACR Cryptology ePrint Archive Report 2011/566 (2011), `http://eprint.iacr.org/`
12. Gentry, C., Halevi, S., Smart, N.P.: Homomorphic evaluation of the AES circuit. IACR Cryptology ePrint Archive 2012 (2012)
13. Graepel, T., Lauter, K., Naehrig, M.: Ml confidential: Machine learning on encrypted data. Cryptology ePrint Archive: Report 2012/323 (June 2012)
14. Lagendijk, R., Erkin, Z., Barni, M.: Encrypted signal processing for privacy protection: Conveying the utility of homomorphic encryption and multiparty computation. Signal Processing Magazine, IEEE 30(1), 82–105 (Jan 2013)
15. Lauter, K., López-Alt, A., Naehrig, M.: Private computation on encrypted genomic data. In: Aranha, D.F., Menezes, A. (eds.) Progress in Cryptology - LATINCRYPT 2014, Lecture Notes in Computer Science, vol. 8895, pp. 3–27. Springer International Publishing (2015)
16. Lindner, R., Peikert, C.: Better key sizes (and attacks) for lwe-based encryption. In: CT-RSA. pp. 319–339 (2011)
17. López-Alt, A., Naehrig., M.: Large integer plaintexts in ring-based fully homomorphic encryption. in preparation (2014)
18. López-Alt, A., Tromer, E., Vaikuntanathan, V.: On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption. In: Proceedings of the Forty-fourth Annual ACM Symposium on Theory of Computing. pp. 1219–1234. STOC '12, ACM (2012)
19. Naehrig, M., Lauter, K., Vaikuntanathan, V.: Can homomorphic encryption be practical? In: Proceedings of the 3rd ACM Workshop on Cloud Computing Security Workshop. pp. 113–124. CCSW '11, ACM (2011)
20. Rivest, R.L., Adleman, L., Dertouzos, M.L.: On data banks and privacy homomorphisms. Foundations of Secure Computation pp. 169–180 (1978)
21. Shoup, V.: `http://www.shoup.net/ntl/`, NTL: A Library for doing Number Theory
22. Smart, N.P., Vercauteren, F.: Fully homomorphic SIMD operations. IACR Cryptology ePrint Archive 2011, 133 (2011)
23. Stehlé, D., Steinfeld, R.: Making NTRU as secure as worst-case problems over ideal lattices. In: Paterson, K.G. (ed.) Advances in Cryptology EUROCRYPT 2011, Lecture Notes in Computer Science, vol. 6632, pp. 27–47. Springer Berlin Heidelberg (2011)
24. Van Dijk, M., Gentry, C., Halevi, S., Vaikuntanathan, V.: Fully homomorphic encryption over the integers. In: Advances in cryptology–EUROCRYPT 2010, pp. 24–43. Springer (2010)

Appendix

Proof of Lemma 5.

Remark 1. We note that cases $2r_i \equiv 0 \pmod{d}$ become simpler in the case when distribution χ generates only polynomials with non-negative coefficients.

Proof. Set $q = p\ell + 1$ and $U = \frac{\ell}{2} - 1$ so that $\|u\|_\infty \leq U$. Observe that $d' = q - \tilde{p}\ell$ is the inverse of d in $\mathbb{Z}_q$ and write $m' = d'm$.

We begin by expanding $f\tilde{c}$ and, where possible, reducing modulo q to find

$$\tilde{c} = d'hs + d'pe + d'm = \tilde{p}gf^{-1}s + \tilde{p}e + d'm$$
$$f\tilde{c} = \tilde{p}gs + \tilde{p}fe + (pf' + 1)d'm = \tilde{p}gs + \tilde{p}fe + \tilde{p}f'm + d'm = \tilde{p}u + d'm \ .$$

Next, we substitute $m = d\tilde{m} + r$ and $d' = q - \tilde{p}\ell$ as above: $f\tilde{c} = \tilde{p}u + d'd\tilde{m} + d'r = \tilde{p}u + \tilde{m} + (q - \tilde{p}\ell)\, r = \tilde{p}u + \tilde{m} - \tilde{p}\ell r$ in $\mathbb{Z}_q[x]$. So we can write

$$\tilde{\mathsf{L}}.\mathbf{Decrypt}(\tilde{c}) = \lceil f\tilde{c} \rfloor_q \bmod \tilde{p} = \lceil \tilde{p}u + \tilde{m} - \tilde{p}\ell r \rfloor_q \bmod \tilde{p}$$

That is, $\hat{m} = \lceil M \rfloor_q \bmod \tilde{p}$ where $M(x) = \tilde{p}u(x) + \tilde{m}(x) - \tilde{p}\ell r(x)$ with coefficients $M_i = \tilde{p}u_i + \tilde{m}_i - \tilde{p}\ell r_i$ for $0 \leq i < n$. We will consider various cases and compute $\lceil M_i \rfloor_q \bmod \tilde{p}$ in each case.

First we observe that, in all cases, $-q < M_i < q/2$. Since $u_i \geq -U$, $\tilde{m}_i \geq 0$ and $r_i \leq d - 1$, we have $M_i \geq -\tilde{p}U - \tilde{p}\ell(d-1) = -\tilde{p}U - \tilde{p}\ell d + \tilde{p}\ell = \tilde{p}(\ell - U) - (q-1) > -q$ since $U < \ell$ by hypothesis. Likewise, $u_i \leq U$ and $\tilde{m}_i < \tilde{p}$ give $M_i < q/2$. So the balanced reduction modulo q takes a very simple form:

$$\lceil M_i \rfloor_q = \begin{cases} M_i + q, & \text{if } M_i \leq -q/2; \\ M_i, & \text{if } -q/2 < M_i \leq q/2. \end{cases} \tag{1}$$

Case 1: $r_i = 0$: Here, we have $M_i = \tilde{p}u_i + \tilde{m}_i > -\tilde{p}U > -q/2$ so that $\lceil M_i \rfloor_q = M_i$ and $\lceil M_i \rfloor_q \bmod \tilde{p} = \tilde{m}_i$.

Case 2: d even, $r_i = d/2$: First note that M_i is close to our boundary $-q/2$:

$$M_i = \tilde{p}u_i + \tilde{m}_i - \frac{p\ell}{2} = \tilde{p}u_i + \tilde{m}_i - \frac{q-1}{2} \ .$$

If $u_i \geq 0$, we obtain $-q/2 < M_i < q/2$ and $\lceil M_i \rfloor_q = M_i$. Since d is even, $2\tilde{p}$ divides $q - 1$ and we have $\lceil M_i \rfloor_q \bmod \tilde{p} = \tilde{m}_i$. On the other hand, if $u_i < 0$, $\tilde{m}_i < \tilde{p}$ gives $\tilde{p}u_i + \tilde{m}_i < 0$ and $M_i < -q/2$ so that $\lceil M_i \rfloor_q = M_i + q$ and

$$\lceil M_i \rfloor_q \bmod \tilde{p} = \left(\tilde{p}u_i + \tilde{m}_i - \frac{p\ell}{2} + p\ell + 1 \right) \bmod \tilde{p} = \tilde{m}_i + 1 \ .$$

This dependence on u_i is reflected in the statement of the theorem by replacing r_i by $r_i - u_i/2U$.

Case 3: $0 < r_i < d/2$: Since $U < \frac{\ell}{2}$ and $d \geq 2$, $\tilde{p}U + \tilde{p}\ell \lfloor \frac{d-1}{2} \rfloor < \frac{q}{2}$ and $\tilde{p}U + \tilde{p}\ell r_i < \frac{q}{2}$, thus $M_i = \tilde{p}u_i + \tilde{m}_i - \tilde{p}\ell r_i > -\frac{q}{2}$. So $\lceil M_i \rfloor_q = M_i$ and $\lceil M_i \rfloor_q \bmod \tilde{p} = \tilde{m}_i$ in this case.

Case 4: $d/2 < r_i < d$: Here, we have $u_i \leq U$, $\tilde{m}_i < \tilde{p}$, and $r_i \geq \frac{d+1}{2}$ so that our bound $U = \frac{\ell}{2} - 1$ gives $\tilde{p}U \leq \tilde{p}\frac{\ell}{2} - \tilde{p}$ and $\tilde{p}U + (\tilde{p} - 1) - \frac{p\ell}{2} - \frac{\tilde{p}\ell}{2} < -\frac{q}{2}$, thus $M_i = \tilde{p}u_i + \tilde{m}_i - \tilde{p}\ell r_i < -\frac{q}{2}$ so that $\lceil M_i \rfloor_q = \tilde{p}u_i + \tilde{m}_i - \tilde{p}\ell r_i + q$ and $\lceil M_i \rfloor_q \bmod \tilde{p} = \tilde{m}_i + 1$. □