Arithmetic Using Word-wise 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. Whereas handling data encrypted at the bit level leads to prohibitively slow algorithms for the arithmetic operations that are essential for cloud computing, the word-based approach hits its bottleneck when operations such as integer comparison are needed. In this work, we tackle this challenging problem, proposing solutions to problems — including comparison and division — in word-based encryption via a leveled FHE 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 [21]. One easily checks that we can model a universal set of gates using addition and multiplication over any non-trivial ring convenient to us. Gentry introduced the first FHE scheme [9, 10] 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., [25, 4, 24]). In 2010, Gentry and Halevi [11] 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; see, e.g., [12, 23, 13]. Nevertheless, bootstrapping [10], 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 [24]; LTV supports inputs from multiple public keys [19]. 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 [8], 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. [15] 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 [20] for instance, Lauter et al. consider the problems of evaluating averages, standard deviations, and logistic 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 [16] 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 [18] where they omit 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. demonstrate in [14] 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. [7] 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 a zero test 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 particular, 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 compare three approaches to field inversion, each with its advantages; these naturally lead to algorithms for division, zero test and equality checking. The first method is exact but slower; the others produce rational approximations, which we scale to integers. Our approach based on Newton-Raphson iterations also gives us an algorithm for square roots. Our convergence-based approach performs better when the characteristic is large due to their amenability to residue number system-based optimizations. Particularly valuable by-products include comparison circuits and threshold functions.
- 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. This technique can be used either individually or to remove the excess bits of the numbers after arithmetic operations. Basically, we show how the numbers are affected 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 operations that output encodings of rational approximations are not compatible with such economies: slight differences in such pre-rounded values can produce vastly different images under the CRT.
– We summarize with an overall comparison of word-wise homomorphic algebraic operations vis a vis their bit-wise counterparts for 32-bit integer domain.
– We implement the proposed methods using an LTV-based homomorphic encryption library and provide the execution times.

3 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 encryption of b_i). In general, all operations are performed over a ring of the form $\mathcal{R} = \mathbb{Z}_q[x]/\langle F(x)\rangle$ with a prime modulus q and an irreducible polynomial $F(x)$ with small coefficients. The schemes also specify an error distribution χ over $\mathcal{R}$, typically a truncated discrete Gaussian distribution, for sampling random polynomials that are B-bounded where $B \ll q$. 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, Encrypt, 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 advanced algebraic operations, we will build homomorphic circuits involving only additions and multiplications, hence our proposed algorithms are not designed for a particular FHE scheme. However, there are some optimization techniques that are specific to the LTV-variant DHS scheme that we used in our experiments.

3.1 DHS Variant of LTV Scheme and FLaSH Library

In 2012 López-Alt, Tromer and Vaikuntanathan proposed a leveled multi-key FHE scheme (LTV) [19]. The scheme is based on a variant of the NTRU encryption scheme proposed by Stehlé and Steinfeld [24]. The LTV scheme uses a new operation called *relinearization* and existing techniques such as modulus switching for noise control. In this work, we use a customized single-key version of LTVproposed by Doröz, Hu, and Sunar [8] along with key size reduction techniques. In this section, we describe an instance L of FLaSH, the software library of the DHS scheme.

Initially, there are three necessary parameters for the setup: message domain p, the multiplicative depth d of the circuit $\mathcal{C}$ to be evaluated and the initial noise factor B. Then we need to select our ring $\mathcal{R} = \mathbb{Z}_q[x]/\langle F(x)\rangle$, however our scheme follows a leveled FHE approach, hence we need a decreasing sequence of moduli, $q_0 > q_1 > \cdots > q_d$ for each level of $\mathcal{C}$. We start with picking a prime q that is large enough to cover the noise in the ciphertext and use $q_i = q^{d-i+1}$ for each circuit level i, i.e. we have a sequence of prime powers, $q^{d+1} > q^d > \cdots > q$. Then we pick a secure degree n for $F(x)$, following the analysis from [8] with respect to the bound on the Hermite factor for a given security level. Once parameters n and q are set and a polynomial $F(x)$ of degree n is fixed (e.g., $F(x) = x^n + 1$ or the m^{th} cyclotomic polynomial $\Psi_m(x)$ with $\varphi(m) = n$), computations are performed in the ring $\mathcal{R}_i = \mathbb{Z}_{q_i}[x]/\langle F(x)\rangle$ for each level i of the evaluation circuit. We write $\mathsf{L} = \mathsf{FLaSH}(\mathcal{C}, p, d, B)$ where

- $\mathsf{L.Keygen}$ first samples polynomials from B-bounded error distribution χ: $f' \leftarrow \chi$ and $g \leftarrow \chi$. Then it sets the secret key $f = pf' + 1$, computes the public key $h = pgf^{-1}$ and finally the evaluation keys $\zeta_\tau = hs_\tau + pe_\tau + w^\tau f^2$ where $s_\tau, e_\tau \leftarrow \chi$, w is the relinearization block size and $\tau \in [0, \lceil \log q_0 / \log w \rceil - 1]$. All operations in this step are performed in ring $\mathcal{R}_0$. [1]
- $\mathsf{L.Encrypt}(m) = hs + pe + m$ where $s \leftarrow \chi$ and $e \leftarrow \chi$ in $\mathcal{R}_0$.
- $\mathsf{L.Decrypt}(c^{(i)}) = \left[\lceil f c^{(i)} \rfloor_{q_i}\right]_p$ where $c^{(i)}$ is the i^{th} level ciphertext and the *balancing* operation $\lceil \cdot \rfloor_{q_i}$ reduces coefficients modulo q to lie between $-q/2$ and $q/2$. The last operation $[\cdot]_p$ reduces the result modulo p to find the encrypted message. The operations are performed in $\mathcal{R}_i$.
- $\mathsf{L.Add}(c_1^{(i)}, c_2^{(i)}) = c_1^{(i)} + c_2^{(i)}$ in $\mathcal{R}_i$.
- $\mathsf{L.Mult}(c_1^{(i)}, c_2^{(i)}) = c_1^{(i)} \cdot c_2^{(i)}$ in $\mathcal{R}_i$.
- $\mathsf{L.Relin}(c^{(i)}) = \sum_\tau \zeta_\tau c_\tau^{(i)}$ in $\mathcal{R}_i$ where $c^{(i)}(x) = \sum_\tau w^\tau c_\tau^{(i)}(x)$ expands ciphertext c as a combination of polynomials with all coefficients in $[0, w-1]$ for $\tau \in [0, \lceil \log q_i / \log w \rceil - 1]$. This operation simulates the homomorphic multiplication with the secret key, hence corrects the encryption mask hs after each multiplication, in addition to reducing the noise.
- $\mathsf{L.ModSwitch}(c^{(i)}) = \lfloor \tilde{c}^{(i)} / q \rceil_p$ decreases the noise by $\log q$ bits by dividing the ciphertext coefficients by q. The operation $\lfloor \cdot \rceil_p$ refers to rounding so as to match all parities with respect to message domain p.
- $\mathsf{L.Batch}(m_1, m_2, \ldots, m_r) = \left[\sum_{i=1}^{r} m_i F_i \left[F_i^{-1}\right]_{f_i}\right]_F$ packs multiple messages into a single plaintext polynomial for parallel evaluations as proposed by Smart and Vercauteren [23, 12]. For this purpose, we select a polynomial $F(x)$ that factors over $\mathbb{F}_p$ into r irreducible polynomials $f_i(x)$ each of degree exactly t. Then messages are embedded using the Chinese Remainder Theorem where $F_i = \frac{F(x)}{f_i(x)}$. When plaintext space $p < n$, we use $F(x) = \Psi_m(x)$, hence we can batch $r = n/t$ messages into one polynomial, where t is the smallest integer that satisfies $\left[p^t - 1\right]_m = 0$ where m is the cyclotomic degree and $\varphi(m) = n$. When $p > n$, we use $F(x) = x^n + 1$ where $[p-1]_{2n} = 0$.

For further details of the scheme, noise growth and security analysis, we refer readers to [8].

4 Beyond Additions and Multiplications

Most homomorphic encryption schemes provide, as basic functionality, addition and multiplication of ciphertexts which encrypt elements in some ring, with the caveat that multiplication gates are considerably "more expensive" than addition gates. At face value, this equips us with the ability to evaluate multivariate polynomials on inputs with a strong preference for low degree polynomials.

In applications such as machine learning, other fundamental operations become essential: **division**, **zero test**, **thresholding** and **comparison**. Bit-level encryption excels at functions with Boolean output but incurs prohibitive cost when required to perform arithmetic even in moderate-sized message domains. Approaching this from the other end, we seek out algebraically efficient algorithms for the operations in the above list. Our solutions fall generally into three categories:

- algebraic and exact algorithms;
- approximation-based algorithms with variable precision outputs;

[1] Note that due to the choice of moduli q_i, all keys can be promoted to the next levels using modular reduction, i.e. $h^{(i)} = [h]_{q_i}$, $f^{(i)} = [f]_{q_i}$ and $\zeta_\tau^{(i)} = [\zeta_\tau]_{q_i}$, when necessary. This reduces the key size significantly as we do not need different keys for each level.

– an ad hoc technique for efficient division of polynomial coeffcients by a prescribed constant.

We begin with three approaches to inversion, which become the basis for more advanced operations later in the section. Section 5 concludes with a summary of how and when RNS-based techniques can be employed to parallelize and/or speed up the execution of these tools as well as expand the message space to avoid overflow.

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-lookahead subtraction circuit and emulate a serial shift division algorithm with depth complexity $\mathcal{O}(n \log(n))$. 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, $P(x, y)$ say, 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-valued 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 $\alpha \equiv \beta \mod \varphi(p)$, 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 $\varphi(p) = p - 1$. It follows that $b^{-1 \mod \varphi(p)} = b^{p-2} \mod p$. Hence we define $P(x) = x^{p-2}$ and $P(x)$ is defined over $\mathbb{Z}_p$.[2]

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, not simply modulo p, but as a real number using a root finding algorithm.

[2] Note that, in case $b = 0$, we will have $P(0) = 0^{p-2} = 0 \mod p$.

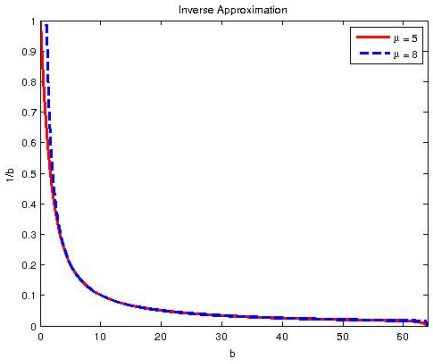
Newton's Root Finding Algorithm We can find the multiplicative inverse of any arbitrary number b using Newton's root finding algorithm. The function $f(z) = 1/z - b$ has a root at $z = 1/b$, hence if we can find the root of $f(z)$, we obtain the reciprocal for b. Iterations start with an initial guess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

For this and upcoming algorithms, we employ *variable precision* representations. More precisely, our ciphertexts will encrypt integers which give correct results only when scaled down and rounded to the nearest integer. We therefore implicitly have a new data type for homomorphic computation. An ordered pair (c, u), where $c = \mathsf{Encrypt}(m)$ and u is a non-negative integer, is viewed as an encryption of the integer $\lceil m/2^u \rfloor$ obtained by rounding $\mathsf{Decrypt}(c)/2^u$. The integer u specifies the location of the (binary) precision point; bits to the right of this point are viewed as a sort of noise. Adding or subtracting two such ciphertexts requires us to align their precision points: for (c, u) and (c', u') with $u' > u$, the sum is represented by $(c2^{u'-u} + c', u')$. Likewise, multiplication of ciphertexts (c, u) and (c', u') yields $(cc', u + u')$. In what follows, the $u + u'$ fractional bits of the corresponding plaintext will be rounded to the nearest integer. Note that, with $\mathbb{Z}_p$ as our message domain, we have $u \leq \log p$.

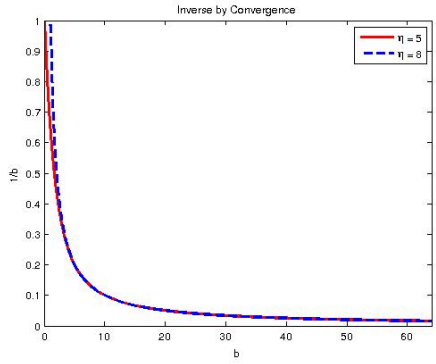
Applying this framework to our Newton iterates, we encode/represent $z_0 = 2^{1-k}$ as $(1, k)$ and set $\bar{z}_0 = 1$. With $\rho = 2^{k-1}$, we then represent the rational number $z_1 = z_0(2 - bz_0)$ as $(\bar{z}_1, 2k)$ where $\bar{z}_1 = \bar{z}_0\,(2\rho - b\bar{z}_0)$ so decryption gives the rounded value $\lceil \mathsf{Decrypt}(\bar{z}_1)/2^{2k} \rfloor \approxeq z_1$ after the $2k$ fractional bits are removed. Continuing in this manner, we represent $z_2 = z_1(2 - bz_1)$ as $(\bar{z}_2, 4k)$ where $\bar{z}_2 = \bar{z}_1\left(2\rho^2 - b\bar{z}_1\right)$ again doubling the number of fractional bits. The general form is then to represent z_{i+1} as $(\bar{z}_{i+1}, 2^{i+1}k)$ where $\bar{z}_{i+1} = \bar{z}_i\left(2\rho^{2^i} - b\bar{z}_i\right)$.

Lemma 2 (Approximate Inverse). *Let* $\mathsf{L} = \mathsf{LTV}(p, q)$ *and* $c = \mathsf{L.Encrypt}(b)$ *with* $0 < b < 2^k$. *We set* $\bar{c}_0 = 1$ *and iteratively compute* $\bar{c}_{i+1} = \bar{c}_i\left(2\rho^{2^i} - c\bar{c}_i\right)$, *where* $\rho = 2^{k-1}$. *Then for sufficiently large* μ, $\mathsf{L.Decrypt}(\bar{c}_\mu) \approxeq (1/b)\,\rho^{2^{\mu-1}} \mod p$. *Let* $d = \mathsf{L.Encrypt}(a)$, *we compute* $\bar{d} = d\bar{c}_\mu$. *Then* $\mathsf{L.Decrypt}(\bar{d}) \approxeq (a/b)\,\rho^{2^{\mu-1}} \mod p$.

The depth of this approximation depends on the number of iterations μ, i.e., it is independent of p. Consider the equation $\bar{c}_{i+1} = \bar{c}_i 2\rho^{2^i} - \bar{c}_i^2 c$, the depth of the function comes from the product $\bar{c}_i^2 c$. Initially $\bar{c}_0$ is a constant, hence the exponent of c in $\bar{c}_1$ becomes 1. In the next iterations, the exponent of c will be $3, 7, \ldots$. Thus, after μ iterations, the exponent will be $2^\mu - 1$ and the circuit depth is μ. This gives a great advantage over the approach based on 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 most 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 $P(x) = 1/x$

Goldschmidt's Convergence Method 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 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 1b.

Given ℓ and $c = \mathsf{Encrypt}(b)$ and setting $\sigma = 2^\ell$, we can mimic the inversion by convergence algorithm to effect a homomorphic division operation. Our variable precision encoding associates the ordered pair (c, ℓ) to $b' = b/\sigma$. Next $z = 1 - b/\sigma$ is represented by $(\bar{z}, \ell)$ where $\bar{z} = \sigma - c$. Likewise, our representation for $r_i = 1 + z^i$ is $(\bar{r}_i, 2^i\ell)$ where decryption of $\bar{r}_i$ is close to $\sigma^{2^i} r_i$.

Putting this all together, we find that $1/b \approxeq r_0 \cdot r_1 \cdots r_\eta/\sigma$ is well approximated by $P(b)/\sigma^{2^{\eta+1}}$ where $P(x) = \prod_{i=0}^{\eta} \left(\sigma^{2^i} + (\sigma - x)^{2^i}\right)$. So our variable precision approach represents our encryption of $1/b$ as $(P(c), 2^{\eta+1}\ell)$ where the fractional part of $P(c)$ consists of the last $2^{\eta+1}\ell$ bits.

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[3]. Therefore, we will need another $\mathcal{O}(\ell)$ iterations for the denominator to reach $2^{i\ell-1} \leq br_0r_1 \ldots r_i < 2^{i\ell}$. 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

[3] 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.

the order of $2^{\mathcal{O}(\log(\ell))}$. 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 (Approximate Inverse). *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 polynomial $P(x)$ has degree $\sum_{i=0}^{\eta} 2^i = 2^{\eta+1} - 1$ requiring a circuit of depth $\log\left(2^{\eta+1} - 1\right) = \eta + 1$. As in the previous approximation method, this is also independent of the message space size 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 also propose another solution, where we describe a homomorphic constant division method, so that we can adjust the precision before decryption. This method decreases the magnitude of p on the fly, hence there are advantages and disadvantages that will be discussed later. (See Section 5.1.)

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 or not the ciphertext is (or rounds to) an encryption of zero. 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 b 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[4] $T(x,t) = \sum_{i\in[t]} (\omega - Z(x-i))$ 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 $c^{\varphi(p)} = c^{\prod \varphi(p_i)}$. 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.

For the first approach we will make use of logistic function and the equation is given as follows, $H(x) = \lim_{k\mapsto\infty} \frac{1}{1+e^{-2kx}} \approxeq (e^x)^{2k} \left(1 + (e^x)^{2k}\right)^{-1}$. 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 $e^x \approxeq \sum_{i=1}^{\infty} \frac{x^i}{i!}$, and we can also use one of the inverse functions that we found in Section 4.1. Thus $H(x)$ becomes: $H(x) \approxeq \left(\sum_{i=1}^{\infty} \frac{x^i}{i!}\right)^{2\kappa} P\left(1 + \left(\sum_{i=1}^{\infty} \frac{x^i}{i!}\right)^{2\kappa}\right)$. 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, $S(x) \approxeq \sum_{i=1}^{\infty} \frac{\sin((2i-1)x)}{(2i-1)}$ For sinus values we can use the approximation, $\sin(x) \approxeq \sum_{j=1}^{\infty} \frac{(-1)^{j-1}x^{2j-1}}{(2j-1)!}$. Embedding this in the previous equation we will have, $S(x) \approxeq \sum_{j=1}^{\infty}\sum_{i=1}^{\infty} \frac{(-1)^{j-1}(2i-1)^{2j-2}}{(2j-1)!} x^{2j-1}$

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 upper limit for j. If we define $i \in [1,\alpha]$ and $j \in [1,\beta]$, then the largest exponent of input x, i.e., the degree of H, becomes $2\beta - 1$. Consequently, the depth of the approximation algorithm becomes $\lceil \log(2\beta - 1) \rceil = \lceil \log \beta \rceil + 1$. For different values of α and β the unit step approximation can be seen in Figure 2.

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 normalize it with $\omega = 2^\ell$, so that the normalized value lies in the input range, i.e.

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

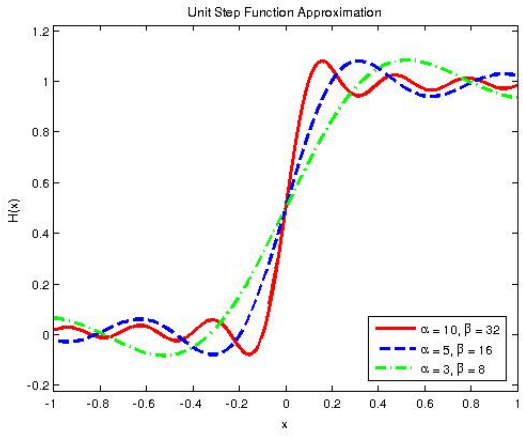


Fig. 2: Unit step function $H(x)$ for various approximation degrees.

$x/\omega \in (-1, 1)$. 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, $R(x)$ say, 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 through 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, κ say; then total depth will be κ times the depth of the inverse computation $P(x)$. Thus this is a much more costly operation relative to inversion.

5 Making Word Arithmetic More Practical

As mentioned before, most of the proposed methods require a large p. By increasing the size of p, we incur a high noise growth in the ciphertexts. As a consequence, this leads to the use of larger coefficient size for the ciphertexts, i.e., the ring $\mathcal{R} = \mathbb{Z}_q[x]/\langle F(x)\rangle$ with a larger q. Increasing q affects security, hence this leads to the use of a cyclotomic polynomial with a larger degree, i.e., even a larger ring $\mathcal{R}$. So even though increasing our message space gives us freedom to handle inputs from a much higher characteristic, this also comes with efficiency problems. In this section we propose two independent methods to make such 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 $\mathsf{L} = \mathsf{FLaSH}(\mathcal{C}, p, d, B)$ with private key $f = pf' + 1$ and public key $h = pgf^{-1}$ and we will examine decryption using $\tilde{\mathsf{L}} = \mathsf{FLaSH}(\mathcal{C}, \tilde{p}, d, B)$ where $\tilde{p}|p$ and $\tilde{\mathsf{L}}$ uses the same private key f (along with the same n and q-sequence) 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 $\mathsf{L}.\mathsf{Decrypt}(c) = m$. 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 $0 \le m_i < p, 0 \le \tilde{m}_i < \tilde{p}, 0 \le r_i < d$. Then, as long as $\|u\|_\infty \le (q - 1 - 2p)/2p$, 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*

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

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 choose 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 precision adjustment technique *cannot* be used along with RNS since CRT cannot recover from rounding errors that occur during decryption.

6 Comparison with Binary Artihmetic

In this section, we will make an overview of all the proposed methods with comparison to their binary equivalents. These operations include *addition* $(+)$, *multiplication* $(*)$, *division* $(/)$, *equality check* $(=)$ and *comparison* $(<)$.

For binary addition we can use a parallel prefix adder such as Kogge-Stone that has a $(1 + \log k)$ depth where k is the bit size of the inputs. For multiplication we can build a Wallace tree multiplier using full and half adders and the circuit has at least $\left(1 + \log_{3/2} k/2\right)$ multiplicative depth. Both addition and multiplication are trivial operations in the word domain. Ciphertext addition does not increase the noise significantly, thus it does not have an effect on the circuit depth. Multiplication increases the circuit depth by adding only one level. Division is by far the most costly of the four

arithmetical operations on binary domains. In order to divide a $2k$ bit number by a k bit divisor, we can build a binary division circuit that involves k cycles of conditional k-bit subtractions. For subtraction, we can use the parallel prefix adders with a delay of $(1 + \log k)$. The condition statement adds one level in each step, thus resulting in an overall circuit depth of $(k\,(2 + \log k))$. For the last two operations we use simple boolean circuits from [6], where equality check has $(\log(k))$ and less than check has $(\log(k) + 1)$ depth.

For 32 bit integer inputs, the parameters that are used in LTV setup can be seen in Table 1. For further details on selection of the security and noise parameters, we refer users to [8].

	Binary					**Wordwise**				
Parameters	$+$	$*$	$=$	$<$	$/$	$+$	$*$	$=$	$<$	$/$
$\log(p)$	1	1	1	1	1	32	32	20	20	20
$\boldsymbol{d}$	**6**	**10**	**5**	**6**	**96**	**–**	**1**	**6**	**5**	**5**
$\log(q)$	23	24	23	23	28	150	150	100	100	100
$\log(q_0)$	161	264	138	161	2716	150	300	700	600	600
n	8190	8190	8190	8190	131070	4096	8192	18432	16384	16384
δ	1.0033	1.0055	1.0028	1.0033	1.0035	1.0062	1.0066	1.0065	1.0063	1.0063
$\log(w)$	3	3	3	3	3	–	30	25	25	25
$\#\zeta_\tau$	20	33	17	20	339	–	10	28	24	24
$\#c^{(0)}$	64	64	64	64	64	2	2	≈ 8	≈ 16	≈ 8
r	630	630	630	630	7710	4096	8192	18432	16384	16384

Table 1: Leveled DHS parameters for bit-wise and word-wise encryption. Key to parameters: $\log(p)$: bit size of the plaintext space; d: multiplicative depth of the circuit; $\log(q)$: bit size of the noise cutting factor; n: degree of the polynomial ring; δ: Hermite factor with respect to the maximum q and n; $\log(w)$: bit size of each relinearization block; $\#\zeta_\tau$: number of relinearization blocks/evaluation keys for the first level; $\#c^{(0)}$: number of total ciphertexts for two operands; r: number of message slots in case of batching enabled.

7 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 9.0.2 [22] compiled with the GMP 5.1.3 package. Our simulations are performed on an Intel Xeon @ 2.9 GHz server running Ubuntu Linux 14.04 LTS. 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. Thus we built our circuits using DHS software library: FLaSH. For parameter selection we utilized the two Hermite factor analysis using the formula in [17], i.e. $1.8/\log\delta - 110$. We used modulus polynomial $x^n + 1$ when $p > n$ for our message embedding otherwise we used $\Psi_m(x)$. For the first test, we evaluated word-wise addition and multiplications. A single 32-bit addition takes 0.8 milliseconds and when batching is enabled it takes 0.19 microseconds per addition. A 32-bit multiplication runs around 1.1 seconds (including the modulus reduction), and is followed by a relinearization which takes 1.2 seconds with a block size of $\log(w) = 30$ bits which gives an amortized time of 0.28 milliseconds per multiplication. Secondly, we evaluated the division circuits for the two proposed methods. For Newton's method

with $\mu = 5$ and inputs in the range $[0, 64]$, we used the RNS method with 4 different 20-bit p values, because p needs to be larger than $2^{(k-1)\left(2^{\mu-1}\right)} = 2^{80}$. In this test, we have a total execution time (including ring operations and relinearizations) of 5.1 minutes. Next, we evaluated Goldschmidt's division by convergence algorithm for $\eta = 5$ and inputs in the range $[0, 64]$, in this case we used 20 different 20-bit p values, because p must be larger than $2^{\ell\left(2^{\eta+1}-1\right)} = 2^{378}$. We also evaluated the equality checks (and/or zero check) using both Fermat's Little Theorem and division by Newton's root finding method. We used two different setup $(p, d) = (17, 4)$ and $(p, d) = (257, 8)$ and disabled batching for this test. For the last test, we computed a comparison with inputs in the range $[0, 32]$ and $\alpha = 5$, $\beta = 16$. In order to have $p > 2^{\ell(2\beta-1)} = 2^{155}$, we set 8 different 20-bit p values. Total execution times can be seen in Table 2.

Operation	**Algorithm**	$(d, \log(q), n, \delta)$	**Total Time**	**Amortized**
Addition	Wordwise	$(0, 150, 4096, 1.0062)$	0.8 ms	0.19 μs
Multiplication	Wordwise	$(1, 150, 8192, 1.0066)$	2.3 sec	0.28 ms
Division	Newton's	$(5, 100, 16384, 1.0063)$	5.1 min	18 ms
	Goldschmidt's	$(6, 100, 18432, 1.0065)$	30.4 min	1.27 sec
Equality Check	Fermat's	$(4, 30, 4096, 1.0061)$	4 sec	-
		$(8, 52, 12288, 1.0065)$	9.09 sec	-
	Newton's	$(6, 100, 18432, 1.0065)$	6.08 min	19 ms
Comparison	Square wave	$(5, 100, 16384, 1.0063)$	10 min	36 ms

Table 2: Parameters and timings for: **Zero Test** using Fermat's Little Theorem with a single message; **Division** first using root finding, then convergence algorithm for multiple packed data; **Comparison** using Square Wave approximation for multiple packed data.

8 Conclusion

This paper explores advances in word-based homomorphic encryption. Directly addressing the weakest points of the current word-based approach, we propose an assortment of solutions to challenging algorithmic bottlenecks that have hampered existing systems from exploiting the full utility of ring operations in large characteristic. As our starting point, we have proposed three distinct approaches to inversion. These lead to efficient algorithms for division, zero test, equality check, thresholding, comparison, and square root, mostly in terms of approximation-based algorithms. We also introduce an extremely efficient technique for constant division and bring in the Chinese Remainder Theorem as a tool to improve the scalability of the proposed approximation algorithms. While many of these operations involve unsurprisingly high degree polynomials (hence require evaluation of deep circuits), our implementation experiments give impressive amortized timings when batching is employed. The most practical use of these techniques remains in applications where all but a small number of gates are addition and multiplication gates, with approximation based algorithms applied only just before decryption.

While we have focused on the DHS variant of the LTV scheme, much of what we explore here is system agnostic and can be adapted to any word-based FHE.

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. Çetin, G.S., Doröz, Y., Sunar, B., Savaş, E.: Depth optimized efficient homomorphic sorting. In: Lauter, K., Rodríguez-Henríquez, F. (eds.) Progress in Cryptology – LATINCRYPT 2015, Lecture Notes in Computer Science, vol. 9230, pp. 61–80. Springer International Publishing (2015)
7. 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)
8. Doröz, Y., Hu, Y., Sunar, B.: Homomorphic AES evaluation using the modified LTV scheme. Designs, Codes and Cryptography pp. 1–26 (2015)
9. Gentry, C.: A Fully Homomorphic Encryption Scheme. Ph.D. thesis, Stanford University (2009)
10. 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)
11. 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)
12. 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/`
13. Gentry, C., Halevi, S., Smart, N.P.: Homomorphic evaluation of the AES circuit. IACR Cryptology ePrint Archive 2012 (2012)
14. Graepel, T., Lauter, K., Naehrig, M.: Ml confidential: Machine learning on encrypted data. Cryptology ePrint Archive: Report 2012/323 (June 2012)
15. 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)
16. 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)
17. Lindner, R., Peikert, C.: Better key sizes (and attacks) for lwe-based encryption. In: CT-RSA. pp. 319–339 (2011)
18. López-Alt, A., Naehrig., M.: Large integer plaintexts in ring-based fully homomorphic encryption. in preparation (2014)
19. 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)
20. 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)
21. Rivest, R.L., Adleman, L., Dertouzos, M.L.: On data banks and privacy homomorphisms. Foundations of Secure Computation pp. 169–180 (1978)
22. Shoup, V.: `http://www.shoup.net/ntl/`, NTL: A Library for doing Number Theory
23. Smart, N.P., Vercauteren, F.: Fully homomorphic SIMD operations. IACR Cryptology ePrint Archive 2011, 133 (2011)
24. 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)
25. 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

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

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, $M_i = \tilde{p}u_i + \tilde{m}_i > -\tilde{p}U > -q/2$ 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 $\lceil M_i \rfloor_q \bmod \tilde{p} = \tilde{m}_i + 1$. □