Faster Homomorphic Evaluation of Discrete Fourier Transforms

Anamaria Costache, Nigel P. Smart, and Srinivas Vivek

University of Bristol, Bristol, UK

Abstract. We present a methodology to achieve low latency homomorphic operations on approximations to complex numbers, by encoding a complex number as an evaluation of a polynomial at a root of unity. We then use this encoding to evaluate a Discrete Fourier Transform (DFT) on data which has been encrypted using a Somewhat Homomorphic Encryption (SHE) scheme, with up to three orders of magnitude improvement in latency over previous methods. We are also able to deal with much larger input sizes than previous methods. Due to the fact that the entire DFT algorithm is an algebraic operation over the underlying ring of the SHE scheme (for a suitably chosen ring), our method for the DFT utilizes exact arithmetic over the complex numbers, as opposed to approximations.

1 Introduction

Since its introduction by Gentry in 2009 [8] most work on Fully (resp. Somewhat) Homomorphic Encryption (FHE/SHE) has focused on evaluating binary or arithmetic circuits. However, for many applications one needs to evaluate functions over more complex data types. In many areas of scientific processing one requires operations on real or complex number, and many applications consist of evaluation of functions of relatively low multiplicative depth. For example, basic statistical calculations are often linear (such as means) or quadratic (such as standard deviations).

This need to process real and complex arithmetic homomorphically has led some authors to propose encoding methods for such numbers [4–6] in the context of encryption schemes based on Ring-LWE. Such schemes are typified by the BGV scheme [3]. The BGV scheme and its extensions [7] are based on a ring

$$R = \mathbb{Z}[X]/\Phi_M(X),$$

where $\Phi_M(X)$ is some cyclotomic polynomial. The ring is considered with respect to two moduli, the plaintext modulus p and the ciphertext modulus q. Writing R_p and R_q for the ring reduced modulo p and q respectively, we have that R_p represents the space of all possible plaintexts and R_q^2 is the ciphertext space.

The first methodology [5, 6] to perform homomorphic operations on real numbers (and hence complex numbers) used a fixed point representation based on the polynomial expansion of the real number with respect to some "base". This

polynomial is then embedded into the plaintext space, and homomorphic operations on the polynomials map into homomorphic operations on the underlying fixed point number. During a homomorphic operation the degree of the representing polynomial increases, as does the size of the coefficients. These two increases imply lower bounds on the degree of the ring R and on the plaintext modulus p. It should be noted that we therefore need to track both noise growth (as in all SHE operations) as well as plaintext growth in such an encoding. See [5] where this growth in coefficient sizes of the representing polynomials is considered in depth. This method uses (in most cases) a single ciphertext to represent a single real number, thus no ciphertext "packing", i.e. amortization, is generally supported. On the other hand, once a given approximation is used for an input plaintext, future homomorphic operations are computed exactly; i.e. floating point precision does not decrease.

A second methodology to perform operations on approximations to complex numbers was presented in [4]. In this methodology a set of $\deg(R)$ approximations to complex numbers are encrypted via a single polynomial. In more detail, for each element in the plaintext space $a \in R_p$, we consider the associated polynomial $a(X)$ and then associate this with the $\deg(R)$ complex numbers $a(\theta_i)$, where θ_i are the roots of the polynomial defining R. In other words, the associated complex numbers are precisely the canonical embedding of the plaintext polynomial. This methodology allows one to produce amortized homomorphic operations via packing. A drawback is however that the associated plaintext polynomial, for a given set of input complex numbers, can have relatively large height.

In both [4] and [5] this ability to homomorphically evaluate on real and complex numbers is demonstrated via a toy example of evaluating a simple image processing pipeline consisting of a DFT, followed by the multiplication of a secret Hadamard transform, followed by an inverse DFT. The results in [4] is particularly interesting, especially when throughput is considered. However, in many applications the main impediment to using homomorphic encryption techniques is low latency; i.e. we are more interested in the time to wait for a single answer than the amortized time over multiple executions.

In this paper, we take this motivating algorithm and show that one can evaluate it over two orders of magnitude faster, by utilizing a completely different representation of the complex numbers. Our method is particularly tailored for DFT operations, however we also show that it can be applied to other more general operations on complex numbers. Note that there are potentially many applications for evaluating DFTs on homomorphic data, as it is widely used in a variety of applications such as signal processing of sound waves and radio signals, or processing of other recurrent data in which determining periodic properties is of interest.

Our techniques make use of the special cyclotomic ring

$$R = \mathbb{Z}[X]/(X^M + 1)$$

where $M = 2^m$ is a power of two. We note that in the ring R the value X corresponds to a formal primitive $2 \cdot M$-th root of unity. Thus by selecting a

mapping $X \mapsto \zeta_{2 \cdot M}$ we can interpret a polynomial in R as being an integer linear combination of the powers of the complex number $\zeta = \zeta_{2 \cdot M}$. Thus our method can be seen as associating a polynomial with a single complex number corresponding to a single component of the canonical embedding, as opposed to the set of complex numbers used in [4]. This means our associated input plaintext polynomials can have smaller height; but we will not be able to deal with ciphertext packing. The effect of this is to improve latency, at the expense of throughput.

For example, if we take a complex number α and then approximate it via the sum

$$\alpha \approx \sum_{i=0}^{N-1} a_i \cdot \zeta^{i \cdot 2 \cdot M/N},$$

then we can use this polynomial to encode the complex number. If the coefficients a_i are selected to be relatively small then the methodology in [5] can be applied to estimate the associated coefficient growth of the encoding polynomials as homomorphic operations are performed. Finding suitably small a_i values can be obtained for an arbitrary complex number via the use of the LLL algorithm [11], in a relatively standard way. See Section 2 for more details on this general methodology.

For the evaluation of the DFT pipeline our method can also dispense with the associated approximations of complex numbers, and we find we can evaluate the DFT pipeline using exact operations on encodings of exact complex numbers. If N is a power of two which divides M then

$$Y = X^{2 \cdot M/N} \qquad (\text{resp. } \zeta_N = \zeta_{2 \cdot M}^{2 \cdot M/N}) \tag{1}$$

is a primitive N-root of unity lying in R (resp. $\mathbb{C}$). Recall that the DFT operation takes an input vector and applies a linear operation (defined over R) to the input vector. Thus, as long as we encode our input in R, we can perform the DFT using only algebraic operations in R. Thus we can homomorphically evaluate the DFT, as long as the coefficient growth of the underlying polynomials can be supported by our plaintext modulus p. When applying DFT in many applications the input can be scaled to be an integer (e.g. in image processing), therefore the input can easily be encoded in an exact manner as well.

This methodology enables us to achieve a considerable improvement in the ability to homomorphically evaluate a DFT. Notice that despite the DFT being linear, the large number of additions and scalar multiplications means that the often heard mantra of "only multiplications matter" does not apply. We need to be careful not only of the growth of the coefficients of the ring elements which encode our values, but also of the homomorphic noise.

We are able to evaluate a DFT-Hadamard-iDFT pipeline of input size 8192 elements, as opposed to 1024 elements for [4] and [5]. In terms of latency we were able to evaluate a pipeline for 256 elements in 9.43 *seconds*, compared to a latency of 581 *minutes* for [5] and 87 *minutes* for [4]. Our amortized times are however much worse; since our method does not allow packing our amortized

time for the same calculation is still 9.43 seconds, compared to 89.4 seconds for [5] and 0.31 seconds for [4]. So whilst we obtain faster latency (and exact computations), for high throughput calculations the method of [4] is still to be preferred.

2 Encoding Approximations to Arbitrary Complex Numbers

As discussed in the introduction, there are two prior methods used to encode complex numbers. The first encodes the complex number as a pair of real numbers and therefore holds the encrypted complex number as the encryption of two real numbers. The real numbers would then be encrypted using the methods suggested in [5, 6] to encode fixed-point numbers. A major downside of this methodology is that to add two encrypted complex numbers requires two homomorphic additions, and to multiply two encrypted complex numbers requires four homomorphic multiplications. The second method suggested in [4] encodes a set of approximations to complex numbers. It looks at these in the canonical embedding of the ring R and then pulls the element back in the canonical embedding to a polynomial. This second method allows multiple complex numbers to be encoded, but the height of the pulled back ring element can be high if only a single complex number is required to be approximated (as would be the case in applications focused on latency).

In this section we present an analogue of the second method where one only wishes to approximate a single complex number. We do this by presenting the folklore method of finding a good approximation to an arbitrary complex number by an element in R. We then can encode the complex number by the associated element in R. As long as we can bound the coefficients of the associated element (in terms of the power basis of R), we can use the method in [5] to bound the growth of the plaintext coefficients as we perform homomorphic operations. Thus we use the method in [5] to bound coefficients of polynomials representing complex numbers, as opposed to polynomials representing fixed-point numbers. The only difference is how we interpret the underlying polynomial/element of R. In comparison to [4] we pull back a single coordinate in the canonical embedding, which allows us a greater degree of freedom in selecting a "small" polynomial to perform the approximation; hence our use of LLL [11] below to find this approximation.

We pick a value n such that n divides $M = 2^m$. This is purely to reduce the size of the associated lattice below from M to the smaller value n, in order to make lattice reduction more manageable. However, a larger value of n will result in an approximation polynomial with smaller coefficients (heuristically, although not provably). We let ζ denote a primitive n-th root of unity, so that ζ is a fixed primitive root of the polynomial $Z^n - 1$, where $Z = X^{M/n}$. Our basic idea for

encoding (an approximation to) the complex number α is to write

$$\alpha \approx \overline{\alpha} = \sum_{i=0}^{n-1} z_i \cdot \zeta^i$$

for some "small" integer values z_i, thus we can approximate α by $\overline{\alpha} \in R$.

We first fix a "large" integer C, say $C = 10^{10}$, which encodes how close we want the approximation to be. We then set, for $i = 0, \ldots, n-1$,

$$a_i = \lceil \Re e(C \cdot \zeta^i) \rfloor \text{ and } b_i = \lceil \Im m(C \cdot \zeta^i) \rfloor,$$

and

$$a = \lceil \Re e(C \cdot \alpha) \rfloor \text{ and } b = \lceil \Im m(C \cdot \alpha) \rfloor.$$

We form the rank $n+1$ lattice $\mathcal{L}$ in $\mathbb{R}^{n+3}$ generated by the columns of the matrix

$$A = \begin{pmatrix} 1 & & & & \\ & \ddots & & & \\ & & 1 & & \\ & & & T & \\ a_0 & \ldots & a_{n-1} & -a & \\ b_0 & \ldots & b_{n-1} & -b & \end{pmatrix},$$

for some non-zero constant T. The determinant of this lattice $\Delta(\mathcal{L})$ is given by $\sqrt{\det(A^T \cdot A)} \approx n \cdot T \cdot C^2/2$, assuming $|a|, |b| < T \cdot C$. We then apply the LLL [11] algorithm to the lattice generated by the columns of A. We let j denote the index of the shortest LLL basis vector which is non-zero in the n-th position (when the basis is ordered in increasing order of size). For a suitably large (but not too large) choice of T, we expect that the j-th basis vector will have $\pm T$ in its $n+1$-st position, and hence will be of the form

$$\mathbf{y} = \left(z_0, \ \ldots, \ z_{n-1}, \ \pm T, \ \sum_{i=0}^{n-1} z_i \cdot a_i \mp a, \ \sum_{i=0}^{n-1} z_i \cdot b_i \mp b \right).$$

We then have, by the usual bounds on LLL basis vectors, that the z_i values for $i = 0, \ldots, n-1$, to be of size bounded by

$$\left(2^{n \cdot (n+1)/4 - 1} \cdot n \cdot T \cdot C^2\right)^{1/(n+2-j)},$$

resulting in an approximation $\overline{\alpha}$ such that

$$\begin{aligned} C \cdot \left|\alpha - \overline{\alpha}\right| &\leq \left|\Re e(C \cdot \alpha) - \sum_{i=0}^{n-1} \Re e(C \cdot z_i \cdot \zeta^i)\right| + \left|\Im m(C \cdot \alpha) - \sum_{i=0}^{n-1} \Im m(C \cdot z_i \cdot \zeta^i)\right| \\ &\approx \left|\sum_{i=0}^{n-1} z_i \cdot a_i - a\right| + \left|\sum_{i=0}^{n-1} z_i \cdot b_i - b\right| \\ &\lessapprox 2 \cdot \left(2^{n \cdot (n+1)/4 - 1} \cdot n \cdot T \cdot C^2\right)^{1/(n+2-j)}. \end{aligned}$$

In other words, for large enough C, we get a good approximation $\overline{\alpha}$ of α. In addition, since LLL usually behaves much better than the theoretical bounds predict, we expect the actual bound on the approximation and the z_i values to grow roughly as $C^{2/n}$. Thus for fixed C, increasing the rank of the lattice - i.e. increasing n - will result in an approximately linear decrease in the coefficient sizes.

Our estimates of the accuracy of the method above depended on the fact that a and b are not too big. In particular we assumed that $|a|, |b| < T \cdot C$, so that they produce a negligible effect on the determinant of the lattice we are reducing. Thus in practice it helps to scale α down so that $|\alpha|$ is close to one, assuming this is enabled by the application in hand. This may require the appropriate scaling to be tracked through the homomorphic operation; much like was proposed in the method from [5]. A similar scaling is needed in the method from [4].

2.1 Numerical Example

Suppose we are given the complex number

$$\alpha = 0.655981733221013 + 0.923883055400882 \cdot \sqrt{-1} = a + b \cdot \sqrt{-1},$$

and we want to produce an approximation which is correct up to ten decimal digits of accuracy using a lattice of dimension $n = 16$. We apply the above method with $C = 10^{10}$ and $T = 10$, and find that the LLL reduced basis of the above rank $n + 1$ lattice in $\mathbb{R}^{n+3}$ has its first basis vector given by

$$(0, -5, 0, 1, -4, 12, 8, -6, -1, -2, -1, -8, -2, 8, 0, 1, 10, -5, -1).$$

Thus if we form the polynomial

$$\begin{aligned} P(Z) = Z^{15} + 8 \cdot Z^{13} - 2 \cdot Z^{12} - 8 \cdot Z^{11} - Z^{10} - 2 \cdot Z^9 - Z^8 \\ - 6 \cdot Z^7 + 8 \cdot Z^6 + 12 \cdot Z^5 - 4 \cdot Z^4 + Z^3 - 5 \cdot Z, \end{aligned}$$

then

$$\begin{aligned} \overline{\alpha} &= P\Big(\exp(\pi \cdot \sqrt{-1}/16) \Big) \\ &\approx 0.65598173270304 + 0.923883055555970 \cdot \sqrt{-1}. \end{aligned}$$

3 New Homomorphic DFT Method

The prior method to approximate complex numbers allows us to homomorphically evaluate operations on complex numbers, as long as there is no wrap-around in the plaintext modulus space, i.e. the plaintext modulus p is chosen appropriately large. Suppose we want to evaluate DFT on an input vector

$$\boldsymbol{v} = (v_0, ...v_{N-1}) \tag{2}$$

of N integers in the range $(-B, \ldots, B)$. For most of this section, we will restrict ourselves to the integer input case because it suffices for our application to homomorphic image processing that we consider in Section 4. However, later in this section, we deal with the case when the DFT inputs are integer polynomials representing elements of the power-of-two cyclotomic rings, i.e. when we want to apply the DFT to the general approximations obtained in the previous section. When the inputs are integers, the nature of the DFT algorithm is such that (as long as the plaintext modulus p is large enough) we obtain an exact computation over the complex numbers. This is because the DFT is an algebraic (in fact linear) operation over the ring R.

For simplicity, let us assume that $N = 2^n$ for some $n \geq 0$. Recall that the ith element $(0 \leq i < N)$ of the DFT output vector is computed as

$$\mathrm{DFT}(\boldsymbol{v})[i] = \sum_{j=0}^{N-1} v_j \cdot \zeta_N^{ij}, \tag{3}$$

where ζ_N is a primitive complex Nth root of unity. We require that ζ_N can be represented by an element in R, and so we must have N dividing $2 \cdot M$. This ensures that the DFT is evaluated exactly.

3.1 Bounding Coefficients

To simulate complex arithmetic in R, the plaintext modulus p must be chosen to be greater than the largest occurring intermediate coefficient in the DFT computation. Hence it is necessary to choose p such that the magnitude of the largest coefficient is less than $p/2$, when we represent the modulo p integers in the interval $(-p/2, \ldots, p/2)$. If this is not done then decrypting the result of a homomorphic operation will not result in the correct value; regardless as to whether the homomorphic noise has swamped the computation.

Substituting ζ_N in (3) by Y from (1), we obtain a vector of polynomials in the indeterminate X,

$$(D_0(X), \ldots, D_{N-1}(X)),$$

where

$$D_i(X) = \sum_{j=0}^{N-1} \boldsymbol{v}_i \cdot Y^{ij}, \tag{4}$$

for $0 \leq i < N$. This corresponds to the set of polynomials that encodes $\mathrm{DFT}(\boldsymbol{v})$ using our encoding scheme. It is this set of polynomials that we wish to homomorphically compute.

For a polynomial $U(X) = \sum_{k=0}^{d} u_k \cdot X^k \in \mathbb{Z}[X]$ define $\|U(X)\|_\infty := \max_k\{|u_k|\}$ and $\|U(X)\|_1 := \sum_{k=0}^{d} |u_k|$. Recall that

$$\left\|a \cdot X^k\right\|_\infty = \|a\|_\infty, \tag{5}$$

$$\|a + b\|_\infty \leq \|a\|_\infty + \|b\|_\infty, \tag{6}$$

where $a, b, k \in \mathbb{Z}$ and $k \geq 0$. The first of the above two properties is crucial to ensure that our encoding scheme leads to much slower growth of coefficients than previous analysis in [5].

From (4) and using the above properties we obtain

$$\|D_i(X)\|_\infty \leq \sum_{j=0}^{N-1} |\boldsymbol{v}_i| = \sum_{j=0}^{N-1} \|\boldsymbol{v}_i\|_\infty < N \cdot B. \tag{7}$$

Invariance. While (7) bounds only the size of the coefficients in the final output, we need to bound the intermediate values as well. But this depends on the method used to compute DFT. In the following, we argue that the bound in (7) also holds for intermediate variables in most of the well-known methods to compute DFT. Two popular methods to compute DFT are:

1. *Naive Fourier Transform* (NFT): the encoded input vector $\boldsymbol{v}$ is multiplied with a matrix A of encoded powers of the primitive Nth root of unity Y, where $A[i, j] = Y^{ij} \pmod{p, X^M + 1}$. This matrix-vector multiplication is usually carried out for small dimensions using either the row approach (scalar product of a column vector and $\boldsymbol{v}$) or the column approach (as a span of column vectors).
2. *Fast Fourier Transform* (FFT): this is a recursive divide-and-conquer procedure, where the ith element $\mathrm{DFT}(\boldsymbol{v})[i]$ $(0 \leq i < N)$ is computed as

$$\mathrm{DFT}(\boldsymbol{v})[i] = \mathrm{DFT}(\boldsymbol{v}[0, \ldots, N/2 - 1])[i] + Y^i \cdot \mathrm{DFT}(\boldsymbol{v}[N/2, \ldots, N - 1])[i]).$$

A hybrid of NFT and FFT is particularly interesting in the context of homomorphic evaluation. This is because it provides a trade-off between the number of scalar multiplications and the depth of the circuit. Here, we count scaler multiplications as contributing to homomorphic depth. The resulting so-called Mixed Fourier Transform (MFT) has been investigated in this context [5]. The divide-and-conquer procedure is applied for instances of size greater than some $\mathfrak{B}$ and for instances of size lesser than or equal to $\mathfrak{B}$, the naive matrix-vector multiplication method is applied. However, in our methodology there is no difference between the homomorphic depth required of the naive or the fast Fourier transform. Since all scalar multiplications are by roots of unity in our algorithms, the noise associated to a ciphertext is never increased by a scalar multiplication. In particular, the noise vector is simply rotated by the scalar multiplication. This means that scalar multiplication in our DFT algorithms does not increase homomorphic ciphertext noise, and so does not contribute to the number of levels required of our underlying SHE scheme.

In all the above methods, any intermediate intermediate plaintext polynomial $U(X)$ is of the form

$$U(X) = \sum_{i=0}^{N-1} u_i \cdot \boldsymbol{v}_i \cdot X^{t_i} \pmod{p, X^M + 1},$$

where $u_i = 0$ or $u_i = 1$, depending upon whether the corresponding summand should be present or not. Assuming no wrap around the modulus p, then using properties (5) and (6), we obtain the same bound as in (7). That is,

$$\|U(X)\|_\infty < N \cdot B. \tag{8}$$

Note that our bounds on the plaintext are also invariant of the method used to compute the DFT, this is not the case with the previous method found in [5], since the output is exact no matter which method is used.

3.2 Extending the Analysis to the Ring of Algebraic Integers

Now suppose that the DFT input vector $\boldsymbol{v}$ (cf. (2)) now contains integer polynomials representing elements of R, instead of just elements from $\mathbb{Z}$. Following an analysis similar to that in Section 3.1, we obtain that any intermediate variable $U(X)$ in the DFT computation satisfies

$$\|U(X)\|_\infty \leq \sum_{i=0}^{N-1} \|\boldsymbol{v}_i(X)\|_\infty \leq N \cdot \max_i \|\boldsymbol{v}_i(X)\|_\infty . \tag{9}$$

Note that the above bound is independent of the number of non-zero terms in the input polynomials. This approximation is useful when we discuss a DFT-Hadamard-iDFT pipeline in the next section.

4 Homomorphic Image Processing

In this section, we apply the bounds obtained in Section 3 to the case of homomorphic image processing. Previously, homomorphic image processing has been investigated in the works of [1, 2, 4, 5]. The works [1, 2] investigate the problem of performing radix-2 DFT in the encrypted domain using additively homomorphic encryption schemes. Because DFT is a linear operation, the authors manage to perform this homomorphically using Paillier encryption scheme [12].

In [5], the authors homomorphically implement a standard image processing pipeline of DFT, followed by Hadamard component-wise multiplication by a fixed but encrypted matrix/vector, and finally inverse DFT to move back from the Fourier domain. The fact that the Hadamard vector is encrypted makes the whole operation non-linear and hence prevents the use of additively homomorphic encryption schemes for this purpose. Yet much smaller parameters size is achieved in [5] compared to [1, 2], even for the operation of homomorphically performing a single DFT only. See [5, Section A.2] for a detailed comparison of their work with that of [1, 2].

In [4] this application is considered again using the packed approximations to complex numbers considered earlier. Here the authors were able to evaluate a degree 256 DFT pipeline in 87 minutes latency (compared to 581) for the method in [5]. In terms of throughput the authors of [4] obtained an amortized

time of 0.31 seconds, compared to 89.4 seconds for [5]. Our results below show we can achieve a latency for the same calculation of 9.43 seconds, but with no improvement possible due to amortization. The largest DFT pipeline reported in previous works was that of a degree 1024 DFT pipeline in [4]. We achieve a pipeline of degree 8192, which we executed with a latency of 1026 seconds.

4.1 DFT-Hadamard-iDFT Pipeline

Inputs to the (homomorphic evaluation of) a DFT-Hadamard-iDFT pipeline are usually (encrypted) integers in some interval $[0, \ldots, B := 2^{b_1})$ representing, for instance, the colour encoding of a pixel. Assume that there are $N = 2^n$ integer DFT inputs and as many in the Hadamard vector for component-wise multiplication. Using our encoding scheme from Section 3, we encode the input integers as themselves in the ring $R = \mathbb{Z}[X]/(p, X^M + 1)$, where as $Y = X^{2M/N}$ encodes a complex primitive Nth root of unity. Because the powers of a primitive root of unity are encoded as monic monomials in R, we do not need to bother to specify the precision for the roots of unity.

From (8), we obtain that during the computation of DFT, the largest occuring intermediate coefficient is bounded above by $N \cdot B$. After the Hadamard component-wise multiplication by a vector of (encrypted) integer entries, the new upper bound is $N \cdot B^2$. Finally, using (9), we obtain the following bound for any intermediate polynomial $U(x)$

$$\|U(X)\|_\infty < N^2 \cdot B^2.$$

Hence we need to choose a plaintext modulus p of the ring R such that

$$p \geq 2 \cdot N^2 \cdot B^2.$$

4.2 Comparison of Concrete Parameters

In [5, Section A.3], the authors use a computational procedure to compute concrete lower bounds for the sizes of p and $\deg(R)$ chosen to homomorphically evaluate the above DFT-Hadamard-iDFT pipeline. As previously mentioned, this is with the Hadamard vector also encrypted. This computational approach was followed because obtaining sharp closed form bounds seems to be out of reach for their encoding technique. Our technique by contrast enables us to obtain tight bounds on the resulting coefficients relatively easily.

Table 1 compares concrete lower bounds for our method and those from [5]. As in [5], we chose $b_1 = 8$ bits of precision for the magnitude of each input, including the entries of the Hadamard matrix. Unlike our case, in [5], the precision b_2 of the roots of unity had to be adjusted so that the final result has a precision of 32 bits. Since our computation is exact this is not a concern.

Note that, as remarked before, the lower bounds on p are independent of the method used to compute DFT. The parameter $\mathfrak{B}$ corresponds to the depth of the MFT method used (cf. Section 3.1). We remark that the size of the plaintext

modulus in our method is close to that required in the case of NFT for [5]. Recall that we also need to lower bound the degree of R by $\deg(R) = M \geq N/2$, which is a much higher bound than that required in [5] for large values of N. However, in practice the degree will need to be much larger than this lower bound to ensure security of the underlying homomorphic encryption scheme. So this increase in the lower bound on the degree is unlikely to be a problem in practice.

			FFT $\mathfrak{B} = 1$		$\mathfrak{B} = \sqrt{N}$		NFT $\mathfrak{B} = N$	
Method	N	b_2	$\log_2 p \geq$	$\deg(R) \geq$	$\log_2 p \geq$	$\deg(R) \geq$	$\log_2 p \geq$	$\deg(R) \geq$
[5]	16	29	54	190	37	118	25	46
This paper	16	-	26	8	26	8	26	8
[5]	64	27	74	248	49	146	29	44
This paper	64	-	30	32	30	32	30	32
[5]	256	25	93	298	61	170	33	42
This paper	256	-	34	128	34	128	34	128
[5]	1024	23	112	340	72	190	37	40
This paper	1024	-	38	512	38	512	38	512

Table 1. Comparison of the parameters for the DFT-Hadamard-iDFT pipeline.

4.3 Comparison of Implementation Timings

As [5], we implemented the full pipeline using the HElib library [10] that implements the BGV Somewhat Homomorphic Encryption scheme [3, 9]. Table 2 compares the performance of our method with that of [5]. The experiments were run on a machine with six Intel Xeon E5 2.7GHz processors with 64 GB RAM. The time, measured in seconds, is that required to evaluate the DFT-Hadamard-iDFT pipeline in the encrypted domain. The parameter $\log_2(q)$ corresponds to the size of the fresh ciphertexts, and "HElib Levels" report the actual number of levels consumed by HElib due to its internal choice of ciphertext moduli. In particular, HElib was allowed to choose by default half-sized primes for the ciphertext modulus chain. Unlike [5] and [4] we are unable to obtain any form of amortization via SIMD packing.

Since HElib has a restriction of at most 60 bits for the plaintext modulus p, not all instances of the MFT could be run with the method from [5] for our comparison. Thus we compare only against the best possible values for $\mathfrak{B}$ for the method from [5] in the table below. Note that this restriction of HElib does not affect our method at all. We are thus able to cope with a much larger range of parameter choices, as described in Table 3. In this table we report the timing results for our method for select instances of the MFT for the chosen values of N. Indeed the fastest run time for our method always occurred when utilizing the full DFT, i.e. for setting $\mathfrak{B} = 1$ in the MFT algorithm. For $N = 1024$, 4096

Method	N	$\mathfrak{B}$	$\deg(R)$	$\lceil \log_2(q) \rceil$	HElib Levels	CPU Time (sec)
[5]	16	16	16384	192	9	106
This paper	16	1	8192	150	7	0.46
[5]	64	8	32768	622	30	1500
This paper	64	1	8192	150	7	2.08
[5]	256	256	16384	278	11	34876
This paper	256	1	8192	150	7	9.43

Table 2. Comparison of the *best* timing results, for a given N, for homomorphically evaluating a full image processing pipeline.

and 8192, we do not report timings for large values of $\mathfrak{B}$ since we did not run the computations until completion as the time taken was too long.

N	$\mathfrak{B}$	$\lceil \log_2 p \rceil$	$\deg(R)$	$\lceil \log_2 q \rceil$	HElib Levels	CPU Time (sec)
16	1	26	8192	150	7	0.46
16	4	26	8192	150	7	0.51
16	16	26	8192	150	7	0.90
64	1	30	8192	150	7	2.08
64	8	30	8192	150	7	2.75
64	64	30	8192	150	7	11.05
256	1	34	8192	150	7	9.43
256	16	34	8192	150	7	16.48
256	256	34	8192	150	7	165.85
1024	1	38	16384	192	9	104.12
4096	1	42	16384	192	10	464.44
8192	1	44	16384	192	10	1026.1

Table 3. Timing results for select instances of MFT for homomorphically evaluating a full image processing pipeline.

Acknowledgements

This work has been supported in part by ERC Advanced Grant ERC-2015-AdG-IMPaCT and by the European Union's H2020 Programme under grant agreement number ICT-644209 (HEAT). We thank the referee for helpful comments on an earlier version of this paper, and pointing out a few optimizations which we had missed.

References

1. Tiziano Bianchi, Alessandro Piva, and Mauro Barni. Comparison of different FFT implementations in the encrypted domain. In *2008 16th European Signal Processing Conference, EUSIPCO 2008, Lausanne, Switzerland, August 25-29, 2008*, pages 1–5. IEEE, 2008.
2. Tiziano Bianchi, Alessandro Piva, and Mauro Barni. On the implementation of the discrete fourier transform in the encrypted domain. *IEEE Transactions on Information Forensics and Security*, 4(1):86–97, 2009.
3. Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (Leveled) fully homomorphic encryption without bootstrapping. In Shafi Goldwasser, editor, *ITCS*, pages 309–325. ACM, 2012.
4. Jung Hee Cheon, Andrey Kim, Miran Kim, and Yong Soo Song. Homomorphic encryption for arithmetic of approximate numbers. *IACR Cryptology ePrint Archive*, 2016:421, 2016.
5. Anamaria Costache, Nigel P. Smart, Srinivas Vivek, and Adrian Waller. Fixed-point arithmetic in SHE scheme. In *Selected Areas in Cryptography - SAC*, 2016. Full version available at `http://eprint.iacr.org/2016/250`.
6. Nathan Dowlin, Ran Gilad-Bachrach, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing. Manual for using homomorphic encryption for bioinformatics, 2015. Available at `http://www.microsoft.com/en-us/research/publication/manual-for-using-homomorphic-encryption-for-bioinformatics`.
7. Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. *IACR Cryptology ePrint Archive*, 2012:144, 2012.
8. Craig Gentry. *A fully homomorphic encryption scheme*. PhD thesis, Stanford University, 2009. `http://crypto.stanford.edu/craig`.
9. Craig Gentry, Shai Halevi, and Nigel P. Smart. Homomorphic evaluation of the AES circuit. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology – CRYPTO 2012*, volume 7417 of *Lecture Notes in Computer Science*, pages 850–867. Springer, 2012.
10. Shai Halevi and Victor Shoup. Design and implementation of a homomorphic-encryption library. Manuscript available at `http://people.csail.mit.edu/shaih/pubs/he-library.pdf`.
11. Arjen K. Lenstra, Hendrik W. Lenstra, and Laszlo Lovasz. Factoring polynomials with rational coefficients. *Math. Annalen.*, 261:515–534, 1982.
12. Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Proc. of EUROCRYPT'99*, pages 223–238, 1999.