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 evaluate a Discrete Fourier Transform (DFT) on data which has been encrypted using a Somewhat Homomorphic Encryption (SHE) scheme, which is over 200 times faster than other methods. The technique utilizes the fact that the entire DFT algorithm is an algebraic operation over the underlying ring of the SHE scheme (for a suitably chosen ring). We then go on to show that the same technique can be used to perform homomorphic operations on encryptions of approximations to arbitrary complex numbers, which dramatically simplifies earlier methodologies.

1 Introduction

Since its introduction by Gentry in 2009 [7] most work on Fully Homomorphic (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 numbers, 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, 5] 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 [6], is 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.

Prior work [4, 5] in encoding real numbers (and hence complex numbers) has 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 [4] where this growth in coefficient sizes of the representing polynomials is considered in depth.

In [4] 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. 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.

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 $\zeta = \zeta_{2 \cdot M}$.

This leads to our immediate improvement on the method to homomorphically evaluate the DFT given in [4]. 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), thus the input can easily be encoded as required.

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.

A key improvement in our method is that we evaluate directly on encodings of complex numbers, thus a homomorphic multiplication requires only one ciphertext multiplication, as opposed to four when one encodes via encodings of fixed point values. This immediately leads one to consider if the same methodology can be applied to perform homomorphic operations on arbitrary complex numbers. With the DFT algorithm we perform an exact computation homomorphically, since the DFT is defined as an algebraic function over R. Clearly, not

all complex-arithmetic algorithms can be evaluated exactly. However, if we are prepared to give up exact computation, we can utilize the same trick.

For example, 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 [4] 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 [10], in a relatively standard way. See Section 4 for more details on this methodology.

2 New Homomorphic DFT Method

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 3. 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. In Section 4, we provide a new method to encode general complex numbers in the power-of-two cyclotomic rings.

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 $2 \cdot M$.

2.1 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

$$\|U(X)\|_\infty := \max_k\{|u_k|\}, \quad \text{and} \quad \|U(X)\|_1 := \sum_{k=0}^{d} |u_k|.$$

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 [4].

From (4), we obtain, using the above properties, that

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

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 as it provides a trade-off between the number of scalar multiplications and the depth of the circuit. The resulting so called Mixed Fourier Transform (MFT) has been investigated in this context [4]. The divide-and-conquer procedure is applied for instances of size greater than, say, $\mathfrak{B}$, and for instances of size lesser than or equal to $\mathfrak{B}$, the naive matrix-vector multiplication method is applied.

In all the above methods, any intermediate intermediate 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 (ensuring which is the whole point of this analysis), 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, while our bounds are invariant of the method used to compute the DFT, this is not the case with the previous method found in [4].

2.2 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 2.1, 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.

3 Homomorphic Image Processing

In this section, we apply the bounds obtained in Section 2 to the case of homomorphic image processing. Previously, homomorphic image processing has been investigated in the works of [1, 2, 4]. 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 [11]. In [4], 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 [4] compared to [1, 2] even for the operation of homomorphically performing a single DFT only. See [4, Section A.2] for a detailed comparison of their work with that of [1, 2].

3.1 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 2, 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

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

3.2 Comparison of Concrete Parameters

In [4, 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, where, as previously mentioned, the Hadamard vector is 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 [4]. As in [4], we chose $b_1 = 8$ bits of precision for the magnitude of each input, including the entries of the Hadamard vector. Unlike our case, in [4], the precision b_2 of the roots of unity had to be adjusted so that the result has a precision of 32 bits. In [4], all the inputs and the roots of unity were encoded as pairs of

polynomials encoding the real and imaginary parts. These encoding polynomial pairs correspond to encodings of integers with respect to a balanced base-3 representation, and each having degree d_1 and d_2 for the inputs and the roots of unity, respectively.

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 2.1). We remark that the size of the plaintext modulus in our method is close to that required in the case of NFT for [4]. Recall, 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 [4] 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 $\mathfrak{B}=n$	
					$\log_2 p$	$\deg(R)$	$\log_2 p$	$\deg(R)$	$\log_2 p$	$\deg(R)$
Method	n	b_2	d_1	d_2	$\geq$	$\geq$	$\geq$	$\geq$	$\geq$	$\geq$
[4]	16	29	5	18	54	190	37	118	25	46
This paper	16	-	-	-	26	8	26	8	26	8
[4]	64	27	5	17	74	248	49	146	29	44
This paper	64	-	-	-	30	32	30	32	30	32
[4]	256	25	5	16	93	298	61	170	33	42
This paper	256	-	-	-	34	128	34	128	34	128
[4]	1024	23	5	15	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.

3.3 Comparison of Implementation Timings

Like [4], we implemented the full pipeline using the HElib library [9] that implements the BGV Somewhat Homomorphic Encryption scheme [3, 8]. Table 2 compares the performance of our method with that of [4]. 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 [4] we are unable to obtain any form of amortization via SIMD packing.

The parameter instances for which the above results are reported were chosen to be the same as reported in [4]. Since HElib has a restriction of at most 60

Method	N	$\mathfrak{B}$	$\deg(R)$	$\log_2(q)$	HElib Levels	CPU Time
[4]	16	1	32768	710	33	188
This paper	16	1	16384	363	15	2.1
[4]	16	4	32768	451	19	147
This paper	16	4	16384	319	12	1.95
[4]	16	16	16384	192	9	106
This paper	16	16	16384	192	7	2.63
[4]	64	8	32768	622	30	1500
This paper	64	8	32768	407	17	32.5
[4]	64	64	16384	192	10	1582
This paper	64	64	16384	192	8	30.37
[4]	256	256	16384	278	11	34876
This paper	256	256	16384	235	10	449.32

Table 2. Comparison of timing results for homomorphically evaluating a full image processing pipeline.

bits for the plaintext modulus p, not all instances of the MFT could be run with the method from [4] for comparison. 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 all possible instances of the MFT for the chosen values of N. We mark in bold the value of $\mathfrak{B}$ which produced the fastest runtime for a given value of N. We could, however, only perform experiments until $N = 512$ due to insufficient RAM memory, which is still twice the size possible with the previous method.

4 Encoding Arbitrary Complex Numbers

The method of the previous section for implementing algorithms in

$$R = \mathbb{Z}[X]/(X^{2^m} + 1)$$

can also be applied to implement (approximate) homomorphic arithmetic for arbitrary complex numbers. The standard method to encode complex numbers is to use a pair of real numbers, and using previous schemes, one would therefore hold the encrypted complex number as the encryption of two real numbers. The real numbers would then be encrypted using the methods suggested in [4, 5] 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.

In this section we present the folklore method of approximating an arbitrary complex number by an element in R. We then can encode the complex number

N	$\mathfrak{B}$	$\lceil \log_2 p \rceil$	$\deg(R)$	$\lceil \log_2 q \rceil$	HElib Levels	CPU Time
16	1	26	16384	363	15	2.1
16	2	26	16384	363	15	2.16
16	4	26	16384	319	12	1.95
16	**8**	**26**	**16384**	**235**	**11**	**1.86**
16	16	26	16384	192	7	2.63
64	1	30	32768	537	24	31.7
64	2	30	32768	537	24	29.09
64	4	30	32768	494	22	29.3
64	8	30	32768	407	17	32.5
64	**16**	**30**	**16384**	**363**	**15**	**16.62**
64	32	30	16384	278	11	23.21
64	64	30	16384	192	8	30.37
256	**1**	**34**	**32768**	**753**	**34**	**152.36**
256	2	34	32768	753	34	154.94
256	4	34	32768	710	31	156.08
256	8	34	32768	622	28	170.92
256	16	34	32768	537	23	199.52
256	32	34	32768	451	20	275.99
256	64	34	32768	363	17	395.93
256	128	34	16384	278	13	305.67
256	256	34	16384	235	10	449.32
512	1	36	65536	884	41	905.58
512	2	36	65536	884	41	906.2
512	4	36	65536	797	36	881.36
512	**8**	**36**	**32768**	**710**	**33**	**395.79**
512	16	36	32768	666	28	530.36
512	32	36	32768	537	25	627.18
512	64	36	32768	451	21	973.67
512	128	36	32768	407	17	1651.82
512	256	36	16384	319	13	1253.21
512	512	36	16384	235	10	1198.48

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

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 then use the method in [4] to bound the growth of the plaintext coefficients as we perform homomorphic operations. Thus we use the method in [4] to bound coefficients of polynomials representing complex numbers, as opposed to polynomials representing fixed-point numbers. The only difference being how we interpret the underlying polynomial/element of R.

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, 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 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 [10] 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). We expect, for a suitably large (but not too large) choice of T 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

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

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

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

In other words we get, for large enough C, 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 [4].

4.1 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

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

Acknowledgements

This work has been supported in part by ERC Advanced Grant ERC IMPaCT and by the European Union's H2020 Programme under grant agreement number ICT-644209 (HEAT).

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. 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`.
5. 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`.
6. Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. *IACR Cryptology ePrint Archive*, 2012:144, 2012.
7. Craig Gentry. *A fully homomorphic encryption scheme*. PhD thesis, Stanford University, 2009. `http://crypto.stanford.edu/craig`.
8. 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.
9. 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`.
10. Arjen K. Lenstra, Hendrik W. Lenstra, and Laszlo Lovasz. Factoring polynomials with rational coefficients. *Math. Annalen.*, 261:515–534, 1982.
11. Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Proc. of EUROCRYPT'99*, pages 223–238, 1999.