On-the-fly Homomorphic Batching/Unbatching

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

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

Abstract. We introduce a homomorphic batching technique that can be used to pack multiple ciphertext messages into one ciphertext for parallel processing. One is able to use the method to batch or unbatch messages homomorphically to further improve the flexibility of encrypted domain evaluations. In particular, we show various approaches to implement Number Theoretic Transform (NTT) homomorphically in FFT speed. Also, we present the limitations that we encounter in application of these methods. We implement homomorphic batching in various settings and present concrete performance figures. Finally, we present an implementation of a homomorphic NTT method which we process each element in an independent ciphertext. The advantage of this method is we are able to batch independent homomorphic NTT evaluations and achieve better amortized time.

Keywords: Homomorphic encryption, homomorphic batching, homomorphic number theoretic transform.

1 Introduction

Fully Homomorphic Encryption (FHE) is an encryption method that allows to perform arbitrary circuit or function evaluations on encrypted data without the need for decryption of the ciphertexts. The first FHE scheme is lattice-based construction introduced by Gentry [12] in 2009. In 2010, Gentry and Halevi [15] simplified the construction and completed the first practical FHE implementation. Even with the optimizations the FHE scheme lacked in performance, since a crucial operation called recryption had to be performed after each bit AND operation which was taking 30 seconds. After the first FHE implementation many various schemes [13, 9, 4, 5, 14, 3] have emerged with different optimization techniques on fully or somewhat homomorphic encryption (SHE). In [26] batching and SIMD operations were introduced to pack multiple messages into a ciphertext and thereby allow for parallel homomorphic evaluations. Other operations such as bootstrapping [12], relinearization [23], modulus reduction [5, 3], key switching [3] and flattening [17] are used as key and noise management techniques permitting the evaluation of deeper circuits with similar parameter sizes.

In [3] Brakerski, Gentry and Vaikuntanathan implemented a leveled FHE scheme that is capable of evaluating polynomial-size circuits by using noise management techniques. Their scheme is based on learning with error (LWE) problem. Later, the BGV scheme was implemented as a software library HElib [19]

using C++. The library was used to re-implement the homomorphic evaluation of an earlier AES circuit [16] by Gentry, Halevi and Smart. They achieved an amortized time of 2 seconds for 120 blocks of AES implementation. Later, [2] presented a new tensor product technique that reduces the noise from quadratic to linear growth. The technique is applicable to LWE schemes, i.e. BGV style schemes. Later, López-Alt, Tromer and Vaikuntanathan (LTV) [23] proposed an FHE scheme based on a variant of NTRU [28] that has multi key support. Doröz, Hu and Sunar implemented the proposed LTV scheme and using it evaluated a custom AES circuit and a level optimized Prince block cipher circuit [10, 11] homomorphically. These implementations were later accelerated using a GPU by Dai et. al. [7, 8]. With GPU support, amortized timings of homomorphic Prince and AES evaluations reduced to 24 msec and 7.3 sec respectively. Recently, a new *approximate eigenvector* FHE scheme with reduction to LWE was proposed by Gentry, Shai and Waters (GSW) [17]. The approximate eigenvector, eigenvalue pairs are used in the construction and they introduce a new noise management technique called flattening. GSW is asymptotically faster due to the use of standard matrix operations in order to apply homomorphic addition and multiplications. With flatting the need for costly relinearization operations and any association storage of massive evaluation keys is eliminated.

Applications. The increasing number of new FHE schemes proposed along with a variety of optimizations, motivated researchers to conduct experiments on their practicality in applications. For example, Lagendijk et al. [20] give details on applicability of homomorphic encryption and multi-party computation for signal processing operations. These signal processing operations include but are not limited to linear filters, correlation evaluations, thresholding, signal transformations, inner product calculations and dimension reduction.

In [24], Lauter et al. focus on simple statistical operations that can be used in real-life cloud services for medical or financial applications such as finding the mean, the standard deviation and the logistical regression. Since these functions do not have high multiplicative depth, they are not necessarily required to be implemented using a FHE scheme, but an SHE construction is sufficient. In the same work, they also implement the SHE scheme of Brakerski and Vaikuntanathan [4] using Magma algebra program. The same reference discussed how to pack multiple message bits into a ciphertext. As noted, even though it is possible to pack multiple ciphertexts into a single ciphertext, there are some problems. First of all, they state that there is no known technique to unpack the messages in the encrypted form, so they cannot retrieve the messages within a packed ciphertext. Secondly, arithmetic operations become limited, i.e. we cannot perform multiplication without destroying the messages in the ciphertext.

Later, in [21] Lauter et al. investigate another homomorphic application: genomic data computation algorithms. They measure the performance of algorithms such as Pearson Goodness-of-Fit test, the D' and r^2-measures of linkage disequilibrium, the Estimation Maximization algorithm for haplotyping, and the Cochran-Armitage Test for Trend. Another homomorphic encryption application on medical data is performed in [1] by Bos et al. The technique is applied on

medical data to perform private predictive analysis on the probability of cardiovascular disease.

There are many other homomorphic applications from various fields that are implemented by various groups of researchers. A machine learning algorithm, i.e. Linear Means Classifier and Fisher's Linear Discriminant Classifier on the Wisconsin Breast Cancer Data set, is implemented in [18] by Graepel et al.. Another application is dynamic programming that is presented by Cheon et al. [6]. They implemented algorithms such as Hamming distance, edit distance, and the Smith-Waterman algorithm on genomic data.

2 Motivation

The recent progress in fully homomorphic encryption schemes motivated researchers to investigate applications of FHE schemes in real life problems. In these applications, researchers face difficulties to evaluate some of the basic primitive operations homomorphically. The lack of these homomorphic primitive operations limits the applications or forces protocol changes, e.g. by moving some of the more difficult operations to the client side. Among these primitive operations an important and yet still open one is the *homomorphic unbatching* operation. Another important operation we wish to support using a similar approach is the homomorphic evaluation of a NTT operation with applications in signal processing. The details are as follows:

- **Homomorphic Unbatching.** This problem was explicitly posed by Lauter et al. in [24]: How can we unpack information belonging to numerous clients packed at the beginning of a homomorphic evaluation session into a single ciphertext for efficiency. The authors mention that if there was a method for *homomorphic unbatching*, a server might easily batch messages of different clients on a single ciphertext, process the ciphertext and later it can homomorphically unbatch the individual results to different ciphertexts to be delivered to the respective clients. Basically, this method helps to significantly improve the computational performance on servers by compressing the different ciphertext messages from users into a single ciphertext for parallel processing. In addition it gives the option to separate these results into different ciphertexts so that the result is only send to the owner. Here we show a way to achieve *homomorphic unbatching* by using Number Theoretic Transform homomorphically. We focus on ways to implement the homomorphic NTT and show the difficulties of achieving Fast Fourier Transform (FFT) speed for *homomorphic unbatching*.
- **Homomorphic NTT.** Later, we implement homomorphic NTT using another method with a focus on achieving FFT speed by holding each input and output elements of the homomorphic NTT in a different ciphertext. Although it is not usable for *homomorphic unbatching*, it can be used for any homomorphic NTT/FFT application, e.g. filtering, large integer and polynomial multiplication, Chebyshev approximation, efficient matrix-vector multiplication, etc.

Our Contribution. In this work we present an array of solutions to improve the versatility of homomorphic NTT, specifically:

- We tackle the problem of computing a number theoretical transform homomorphically over the domain defined by the message space. It turns out that noise growth is a significant issue and FFT speed evaluation is difficult to achieve without homomorphic modular reduction. We work out a solution and provide concrete performance figures.
- Empowered by homomorphic NTT we define homomorphic batching / unbatching which allows us to move the coefficients of encrypted message polynomials into message slots and vice versa. Using homomorphic batching one may **unpack** message polynomials, i.e. extract coefficients from encrypted message polynomials; and more broadly change the processing domain on-the-fly while evaluation proceeds.
- From a security perspective, homomorphic batching / unbatching allows us to prevent information leakage through partial evaluation results that accumulate in batched messages. This is of utmost concern in multi-user settings where multiple streams of information are bundled together and processed simultaneously.
- We implement homomorphic NTT using another method in which we encrypt the elements of the NTT in different ciphertexts and perform levels of NTT computations on these ciphertexts. In the end we achieve the elements of NTT result in different ciphertexts. This way we are able to achieve the FFT speed, batch independent NTT operations for parallel processing and achieve amortized time. However we are unable to compute batch/unbatch homomorphically.
- Also, we give run-time complexity analysis on both of the proposed homomorphic NTT methods.
- Finally we note that homomorphic NTT is of independent interest to numerous applications, e.g. filtering in digital signal processing, spectral decomposition and analysis, etc.

3 Background

In this work, we use customized leveled FHE implementation propsed by Doröz, Hu and Sunar (DHS) [10]. The library is written in C++ and it uses NTL software with GMP support. The library supports the leveled multi-key FHE scheme implementation proposed by 2012 López-Alt, Tromer and Vaikuntanathan (LTV) [23]. It is based on a variant of Stehlé and Steinfeld's [27] NTRU encryption with new operation called relinearization and existing operation modulus switching to control noise. Although, the scheme can support support multi-keys (users) the implemented library focuses on single-key (user) implementation.

The LTV scheme is build on using the following primitives; they select a polynomial ring as $\mathrm{R}_q = \mathbb{Z}_q[x]/\langle x^N + 1\rangle$ with N being the polynomial degree and q being the prime modulus. They also select a message prime modulus p.

In the scheme, polynomials are sampled using a truncated discrete Gaussian distribution χ. These are also B-bounded polynomials which each coefficient of the polynomial is selected between $[-B, B]$. The scheme consist of the following primitive functions: KeyGen, Encrypt, Decrypt and Eval. The primitive Eval consist of a multiplication operation which is later followed by a relinearization and modulus switch operation. The relinearization and modulus switch operations are used to reduce the noise caused by the multiplication. Here we describe the primitive functions of the LTV scheme:

KeyGen. In the scheme modulus q is decreasing sequence of prime numbers for each level: $q_0 > q_1 > q_2 \cdots > q_d$. We select the two consecutive modulus q_i by the size of noise at that level. We sample two polynomials $g^{(i)} \in \chi$ and $u^{(i)} \in \chi$ to compute secret keys $f^{(i)} = pu^{(i)} + 1$ and public key $h^{(i)} = pg^{(i)}f^{-(i)}$ for each level. We evaluate the evaluation keys $\zeta_\rho^{(i)}(x) = h^{(i)}s_\rho^{(i)} + pe_\rho^{(i)} + 2^\rho f^{2(i-1)}$ which $\{s_\rho^{(i)}, e_\rho^{(i)}\} \in \chi$ and $\rho = [0, \lfloor \log(q_i) \rfloor]$ for each level i.
Encrypt. We encrypt a message $b \in \mathbb{Z}_p$ (or it can be a message polynomial $b(x) \in \mathbb{Z}_p[x]$) by evaluating $c^{(i)} = h^{(i)}s^{(i)} + pe^{(i)} + b$ by sampling $\{s^{(i)}, e^{(i)}\} \in \chi$ for i^{th} level.
Decrypt. The decryption for i^{th} level is simply achieved by evaluating: $m = c^{(i)}f^{(i)} \pmod p$.
Eval. The multiplication and summation of the ciphertexts corresponds to multiplication and addition of messages in modular p. These operations increase the noise level of the ciphertexts and multiplication explicitly outweighs addition in terms of noise creation. The scheme uses two significant operations to control noise growth; relinearization and modulus switching. We summarize these two operations as follows:

- **Modulus Switching.** This operation is a way of reducing the existing noise in the ciphertexts. Basically, we perform $\tilde{c}^{(i)}(x) = \lfloor \frac{q_i}{q_{i-1}} \tilde{c}^{(i-1)}(x) \rceil_p$ on each coefficient of the ciphertxt. We achieve following two things; a reduction in the noise by $\log(q_i/q_{i-1})$ bits and a new field $\mathbb{Z}_{q_i}$ for modular arithmetic. The ceil/floor operation $\lfloor \cdot \rceil_p$ refers to rounding to match the parity for modular p. An advantage of the scheme is that its performance is increased as we switch levels due to a smaller modulus q_i.
- **Relinearization.** This operations is necessary after each multiplication operation in order to prevent the noise growth and the increase of inverse powers of secret keys $f^{(i)}$. Simply we are switching square power of secret key $f^{-2(i-1)}$ for level $i-1$ with the new secret key $f^{(-i)}$ of level i. We evaluate the relinearization operation by computing: $\tilde{c}^{(i)}(x) = \sum_\rho \zeta_\rho^{(i)}(x)\tilde{c}_\rho^{(i-1)}(x)$. In the equation $\tilde{c}_\rho^{(i-1)}(x)$ are binary polynomials that forms $\tilde{c}^{(i-1)}(x) = \sum_\rho 2^\rho \tilde{c}_\rho^{(i-1)}(x)$.

Specializations. In DHS library [10], the decreasing modulus sequence is selected as a power of a fixed prime, i.e. $q_i = \sigma^{k-i}$. Here prime number σ is equal to noise cutting size for each level and k is circuit depth plus one. This special ring structure is used to promote evaluation keys to the next level when needed,

i.e. $\zeta_\rho^{(i)}(x) = \zeta_\rho^{(0)}(x) \bmod q_i$. This reduce the key store significantly from $k+1$ level of evaluation keys to 1 level of evaluation key.

3.1 Fourier Transform

Fourier Transform (FT) is a signal transformation method that is used in many mathematical and scientific applications such as filtering, time domain and frequency domain conversions, large integer multiplications and sine/cosine wave transformations. For practical applications, Discrete Fourier Transform (DFT), finite version of the FT, is used. DFT can be computed by simply evaluating:

$$X_k = \sum_{j=0}^{N-1} x_j e^{-2\pi i k \frac{j}{N}},$$

for each $k \in N$. This operation can also be represented as a matrix multiplication of a vector input $\overrightarrow{x} = [x_0, x_1, x_2, \ldots, x_{N-2}, x_{N-1}]$ with a special Fourier Transform matrix $\mathbf{W}$, i.e. $X = \mathbf{W} \cdot \overrightarrow{x}$. This special Fourier Transform matrix $\mathbf{W}$ has the structure of a Vandermonde matrix with entries $\alpha_{i,j} = \left(\alpha^i\right)^j$ and can be represented as follows:

$$\begin{bmatrix} \alpha^0 & \alpha^0 & \alpha^0 & \ldots & \alpha^0 & \alpha^0 \\ \alpha^0 & \alpha^1 & \alpha^2 & \ldots & \alpha^{N-2} & \alpha^{N-1} \\ \alpha^{2\cdot 0} & \alpha^{2\cdot 1} & \alpha^{2\cdot 2} & \ldots & \alpha^{2\cdot(N-2)} & \alpha^{2\cdot(N-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \alpha^{(N-2)\cdot 0} & \alpha^{(N-2)\cdot 1} & \alpha^{(N-2)\cdot 2} & \ldots & \alpha^{(N-2)\cdot(N-2)} & \alpha^{(N-2)\cdot(N-1)} \\ \alpha^{(N-1)\cdot 0} & \alpha^{(N-1)\cdot 1} & \alpha^{(N-1)\cdot 2} & \ldots & \alpha^{(N-1)\cdot(N-2)} & \alpha^{(N-1)\cdot(N-1)} \end{bmatrix}$$

where $\alpha = e^{\frac{-2\pi i}{N}}$. In a naïve implementation the time complexity of the DFT becomes $\mathcal{O}(N^2)$. However, by using a Fast Fourier Transform (FFT) algorithm namely Cooley-Tukey method, we can reduce the cost of the evaluation to $\mathcal{O}(N \log(N) \log\log(N))$. The Cooley-Tukey algorithm is based on re-expressing the DFT equation into summation of two sub-DFT equations:

$$X_k = \sum_{j=0}^{N-1} x_j e^{-i2\pi k \frac{j}{N}} = \underbrace{\sum_{m=0}^{N/2-1} x_{2m} e^{-i2\pi k \frac{m}{N/2}}}_{Even} + \underbrace{e^{\frac{-2\pi i k}{N}} \sum_{m=0}^{N/2-1} x_{2m+1} e^{-i2\pi k \frac{m}{N/2}}}_{Odd}$$

As shown on the equation above, summation on the left calculates the DFT of the even indices and the summation on the right calculates the DFT of the odd indicies. These odd/even DFT summations can also be re-expressed as summation of sub -odd and -even indicies. This procedure can be applied recursively until the DFT size is small enough to be evaluated fast enough. Later, the FFT can be calculated by reconstructing the calculated sub-DFT's by going into upper levels in the recursive function.

3.2 Number Theoretic Transform

Number Theoretic Transform is a specialization of DFT over the ring $\mathbb{Z}/p\mathbb{Z}$ by replacing $e^{-i2\pi k\frac{j}{N}}$ with N^{th} primitive root of unity ω. One of the most common usage of the method is to evaluate large integer or polynomial multiplications. It prevents the errors that might be caused by the floating point arithmetic of FFT and provides precise arithmetic evaluations. We can compute the NTT by simply evaluating:

$$X_k = \sum_{j=0}^{N-1} x_j \omega^{k \cdot j} \pmod{p},$$

where $\overrightarrow{x}$ is again the input vector, p is the prime modulus and $k \in N$. The inverse-NTT of the evaluated vector $\overrightarrow{X}$ is computed using the same equation by replacing ω with $\omega^{-1} \pmod{p}$:

$$x_k = \sum_{j=0}^{N-1} X_j \omega^{-k \cdot j} \pmod{p}.$$

Thus the transformation matrix $\mathbf{W}$ and the inverse transformation matrix $\mathbf{W^{-1}}$ becomes:

$$\begin{bmatrix} 1 & 1 & \dots & 1 \\ 1 & \omega & \dots & \omega^{N-1} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{N-1} & \dots & \omega^{(N-1)\cdot(N-1)} \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} 1 & 1 & \dots & 1 \\ 1 & \omega^{-1} & \dots & \omega^{-(N-1)} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{-(N-1)} & \dots & \omega^{-(N-1)\cdot(N-1)} \end{bmatrix}$$

respectively. Using the Cooley-Tukey approach that is explained in the previous section, the NTT conversion can achieve runtime of $\mathcal{O}(N \log(N) \log\log(N))$.

4 Homomorphic NTT

In this section we give two methods to perform homomorphic NTT and discuss their advantages and disadvantages. In the first method we show that we are able to homomorphically batch/unbatch messages on a single ciphertext, but also show that it has limitations to achieve FFT speed. In the second method we show that we can overcome the problem and achieve FFT speed, but we need to use N ciphertexts for the input of the NTT. Also, we are able to batch the messages to form a single ciphertext result but we are unable to unbatch the messages for further processing.

4.1 Homomorphic Batching/Unbatching

Batching is a powerful data encoding technique when used in homomorphic computing, yields great versatility in the computations and greatly improves performance. To this end the data is first embedded into *message slots*. The

message slot contents are then *encoded* into a polynomial representation with the help of the (inverse) Chinese Remainder Theorem. The encoded message polynomial is then encrypted. Once batched messages are encoded, then they are processed in independent homomorphic evaluation paths. Once the evaluations are completed, the output message polynomial is *decoded* and the message slot contents are retrieved using the CRT residue computation.

To overcome this difficulty we need a transformation that is capable to bring the message slot contents into the coefficients in the polynomial representation and back while all data is maintained in encrypted form. The homomorphic evaluation needed here is the equivalent to a CRT and its inverse computation. We briefly define homomorphic batching as follows.

Definition 1 (Homomorphic Batching). *The isomorphism* $\mathbb{Z}_p[x]/(\Phi(x)) \cong \mathbb{Z}_p[x]/(x-\zeta) \times \mathbb{Z}_p[x]/(x-\zeta^2) \times \cdots \times \mathbb{Z}_p[x]/(x-\zeta^{N-1})$ *where* $\Phi(x) = \prod(x-\zeta^i)$ *denotes the characteristic polynomial in* $\mathbb{Z}_p$ *of degree* $N-1$ *and* ζ *denotes* N^{th} *root of unity in* $\mathbb{Z}_p$*. We refer to the homomorphic evaluation of the isomorphism and its inverse as homomorphic unbatching and batching, respectively.*

From a computational perspective, the encoding/decoding operations both amount to the evaluation of a linear transformation on the message slot/polynomial coefficients, respectively. For instance, the message polynomial $m(x)$ is decoded as $\mathbf{M} = \langle m(\zeta), m(\zeta^2), \ldots, m(\zeta^{N-1})\rangle$. The encoding function may be computed, for example, using Lagrange interpolation. For computations may be expressed as linear transformations as $\mathsf{Decode}(m(x)) = \mathbf{W}\bar{m}$ and $\mathsf{Encode}(\bar{m}) = \mathbf{W^{-1}M}$ where $[\mathbf{W}]_{ij} = \zeta^{ij} \in \mathbb{Z}_p$. The operation appears simple enough since modulo p operations is the natural domain of the homomorphic evaluations and since all we need to compute is constant multiplications by powers of ζ. Typically, when batching in cleartext we compute the encoding and decoding operation with the aid of an N^{th} root of unity $\zeta \in \mathbb{Z}_p$ via a number theoretical transform (NTT) to gain *FFT speed*, i.e. $\mathcal{O}(N\log(N))$ encoding/decoding performance. However, the structure of the cyclotomic polynomial have limitations that prevents us from directly evaluating NTT.

Limitations. The message slots present independent computation paths. To compute the linear transformation we need to sum the scaled message slot contents. Thus we need a means to move the message slot contents. We therefore use $\Phi(x) = \prod_{i\in[N-1]}(x-\zeta^{b^i})$ where b is a primitive element of $\mathbb{Z}_N^*$. Then by evaluating $m(x^b)$ will rotate the message slot contents. The side-effect of this shift operation on a ciphertext is that the key is altered during the evaluation process:

$$c(x^b) = pg(x^b)s(x^b)f^{-1}(x^b) + pe(x^b) + m(x^b)$$

The ciphertext will still decrypt correctly since $g(x^b), s(x^b)$ and $e(x^b)$ will have small norm. However, to decrypt the ciphertext the key needs to be updated to $f(x^b)$. To restore the original key we may use *key switching*: $\mathsf{L.KeySwitch}(c(x^b), \theta)$ where $\theta = \left\{\mathsf{L.Encrypt}(w^\tau f(x^b)_\tau) \text{ for } \tau \in [\log q]\right\}$. With this approach we can rotate the message slot contents an arbitrary i positions by evaluating the ci-

phertext polynomial as $c(x^{b^i})$ and then by applying a key switching operation with $f(x^{b^i})$.

Here the problem lies with the selection of cyclotomic polynomial $\Phi(x)$ as the modulus. It gives a decoding matrix $\mathbf{W}$ as:

$$\begin{bmatrix} \alpha^0 & \alpha^1 & \alpha^2 & \dots & \alpha^{N-2} & \alpha^{N-1} \\ \alpha^{2\cdot 0} & \alpha^{2\cdot 1} & \alpha^{2\cdot 2} & \dots & \alpha^{2\cdot(N-2)} & \alpha^{2\cdot(N-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \alpha^{(N-2)\cdot 0} & \alpha^{(N-2)\cdot 1} & \alpha^{(N-2)\cdot 2} & \dots & \alpha^{(N-2)\cdot(N-2)} & \alpha^{(N-2)\cdot(N-1)} \\ \alpha^{(N-1)\cdot 0} & \alpha^{(N-1)\cdot 1} & \alpha^{(N-1)\cdot 2} & \dots & \alpha^{(N-1)\cdot(N-2)} & \alpha^{(N-1)\cdot(N-1)} \end{bmatrix}$$

and an encoding matrix $\mathbf{W^{-1}} \pmod p$. The formed matrices $\mathbf{W}$ and $\mathbf{W^{-1}}$ of $\Phi(x)$ are not Vandermonde matrices, therefore we are unable to apply Cooley-Tukey's algorithm. Since we cannot apply the even-odd splitting trick, we are unable to apply fast NTT.

We can solve $\mathbf{W}$ and $\mathbf{W^{-1}}$ not being Vandermonde matrices by switching the cyclotomic polynomial $\Phi(x)$ with $x^N - 1$ which has the following form:

$$x^N - 1 = (x-1) \cdot \Phi(x) = \prod_{i=0}^{i=N-1} (x - \zeta^i),$$

where N is power of 2. This converts the batching operation to be applicable using Vandermonde matrix multiplication which is suitable for fast NTT using Cooley-Tukey. Although scheme is suitable for fast NTT, we are not able to rotate the messages as in cyclotomic polynomials. The message in the first slot, i.e. in $(x - \zeta^0)$, never rotates in function $f(x^{b^i})$ for any i.

Homomorphic Batch/Unbatch. With the issues addressed above, we are able to compute homomorphic unbatching by the following equation:

$$\mathsf{L.Unbatch}(c) = \sum_{s \in N} c^{(s)} \cdot \mathsf{Encode}\left((\mathbf{W}^{\mathrm{rot}})^{\top}[\mathrm{s}]\right).$$

Here $c^{(s)}$ represents the rotated versions of the ciphertext (and message) coefficients by s positions. $\mathbf{W}^{\mathrm{rot}}$ is the transformation matrix where each row index i is rotated by i. The symbol $\top$ represents the transpose of the matrix and [s] is used for the row index s of the matrix. In case of batching we only replace $\mathbf{W}$ with $\mathbf{W^{-1}}$. The all operation requires only one level of circuit depth for evaluation.

Packing/Unpacking. Packing and unpacking messages in homomorphic encryption is useful for processing the information in parallel by batching/unbatching multi-user information. In multi-user scenarios where we have many users that provides input for a process, we can pack the informations to efficiently process. In word message space we can input N user information into the same chipertext that will provide N times speedup for information processing. In [24] Lauter et

al. show how to pack the messages from multiple ciphertexts into one ciphertext. They mention that they cannot present a technique to unpack messages which restricts their computations. After a packing operation the polynomial multiplications rounds and deforms the information because of the polynomial modulus. Our main motivation here was to transfer the message slot contents into the polynomial coefficients and back. However, we achieve unpacking, which is regarded as difficult to achieve, with the aid of homomorphic batching. We may unpack the k^{th} coefficient $c^{(k)} = \mathsf{L.Encrypt}(m_k x^k)$ and $c = \left(\sum_{i\in[N]} m_i x^i\right)$ with the following steps:

- Push coefficients into the message slots $\tilde{c} = \mathsf{L.Unbatch}(c) \in \mathcal{R}$,
- Filter desired coefficient(s) by multiplying with constant cleartext mask $\mu_k(x) = \mathsf{NTT}^{-1}(I_k)$,
- Push message slot contents back into coefficients by homomorphic batching $c^{(k)} = \mathsf{L.Batch}(c\tilde{\mu_k}(x)) \in \mathcal{R}$.

The packing/unpacking operation enables to do *privatization* in the homomorphic encryption. We may easily batch the information for parallel processing and later send the result informations for filtering the results for each users in multi-user scenarios. This prevents the information leaks while returning the results to the users, since we are able to eliminate the results of other users from the ciphertext.

4.2 Homomorphic NTT Using Parallel Batching

There is an alternative and straightforward way to implement homomorphic NTT that is not limited by the issues given in the previous section. We can encrypt each message to be used in the NTT separately: $c_i = hs_i + pe_i + m_i$. Then, we can compute the fast NTT using the Cooley-Tukey as:

$$C_k = \sum_{j=0}^{N-1} c_j \zeta^{jk} = \underbrace{\sum_{j=0}^{N/2-1} c_{2j}\zeta^{2jk}}_{Even} + \underbrace{\zeta^{jk} \sum_{j=0}^{N/2-1} c_{2j+1}\zeta^{2jk}}_{Odd}$$

, where ζ is the N^{th} primitive root of p. Since each message is in an independent ciphertext, we can easily divide them into even and odd indicies. This way we can easily compute the fast NTT of the input. However there are two main issues with the scheme that limits the operation:

- The modulo p reduction does not take place until the very end of the decryption step, i.e. $\mathsf{L.Decrypt}(c) = \lceil cf \rfloor_q \bmod p$. Therefore, intermediate results will accumulate powers of w, which likely will cause a wraparound and decryption failure. One alternative is to aggressively apply noise reduction, e.g. modulus switching, even for the constant multiplications. However, this will increase the evaluation levels significantly. For instance, even a moderate $N = 2^{13}$ would add 13 evaluation levels. To overcome noise accumulation we

abandon FFT style evaluation and instead only multiply with precomputed $\mathbf{W}, \mathbf{W^{-1}} \in \mathbb{Z}_{\tilde{p}}^{(N-1)\times(N-1)}$.

– The number of ciphertexts increase to the number of NTT elements, i.e. N in our case, from a single ciphertext. This increases the ciphertext input size by N times and it is equal to $N^2 \log q$. More than the computational complexity, it increases the I/O transactions of the scheme significantly. Although we have N ciphertexts at the end, we can simply batch them by evaluating:

$$\sum_{i=0}^{i=N-1} C_k \cdot x^i.$$

This solves the issue of having many ciphertexts. However, we are unable to unbatch the values in the equation which limits further processing. We can access the values individually only after a decryption operation.

Although we are not able to batch the dependent elements in a fast NTT operation, we are able to batch N independent fast NTT operations. Basically, we are able to use the empty message slots to evaluate N parallel fast NTT operations. This way we are able to achieve an amortized time that is N times better than the total runtime.

5 Complexity Analysis

Here we discuss the complexity of the two proposed algorithms. In homomorphic batching we need to compute N multiplications of ciphertext with a polynomial formed by the row values of $\mathbf{W}$. Using a large polynomial multiplication algorithm, such as Schönhage-Strassen algorithm, we achieve a run-time complexity of $\mathcal{O}(N \log N \log\log N)$. Furthermore, we have to perform key-switching operations to the ciphertexts to correct the public keys that are corrupted in rotation operation. This is a similar operation to the relinearization, so we can apply the time complexity of relinearization in [10] for key-switching as well. We have N key-switching operations with run-time complexity of $\mathcal{O}(\log(q) N \log N \log\log N)$. In total the algorithm has a run-time complexity of $\mathcal{O}(\log(q) N^2 \log N \log\log N)$.

In the second algorithm, i.e. homomorphic NTT using parallel batching, we have $\log N$ stages of NTT operations. Each stage N multiplications of a constant with a ciphertext which makes N^2 coefficient multiplications per stage. In total the algorithm has run-time of $\mathcal{O}(N^2 \log N)$.

An important thing to note is that the complexity analysis takes into account only the number of coefficient multiplications. It does not include the run-time complexity of the coefficient multiplications. In the first case we have small and fix size coefficients which gives an advantage in real time applications against the second method. The second method has larger coefficients because of the leveled implementation. Thus it takes longer time to process the second method even though the run-time complexity of the method is smaller in terms of number of coefficient multiplications.

6 Implementation Results

We implemented the algorithms using a leveled LTV scheme using Shoup's NTL library version 9.0 [25] compiled with the GMP 5.1.3 package. For parameter selection we utilized the two Hermite factor analysis using the formula in [22], i.e. $1.8/\log\delta - 110$. The security level of the experiments varies on the settings, but each setting has at least 100-bit security.

In the homomorphic NTT using homomorphic batching we use special cyclotomic polynomial $\Phi_m(x)$, where we set m as a prime number to have $\Phi_m(x) = x^0 + x^1 + x^2 + \cdots + x^N$, to perform faster modular reduction. The results are summarized in Table 1. In the algorithm we have one constant polynomial multiplication and N additions, so our prime modulus q does not grow too large. The values of N are chosen to be close to as powers of two, i.e. 2048, 4096, 8192. The message slots are used for the same NTT operation so there is no amortized time.

In the second case, we compute homomorphic NTT by using parallel batching. We choose the polynomial degree $N = 16384$ and modulus bitsize $\log q = 512$ which are slightly higher values compared to the first algorithm. The reason behind this is that we need to handle the noise in stages, so the modulus q grows significantly. Our implementation achieves a runtime of 108 minutes. Since we are able to batch N independent homomorphic NTT computation, we achieve 0.4 second of amortized time.

N	$\log q$	Security (in bits)	Total Time (in minutes)
2080	64	140	2.5
4252	64	400	10.7
8782	64	943	43

Table 1. Timings for Homomorphic Batching/Unbatching operation.

7 Conclusion

To improve the versatility of homomorphic encryption applications, we tackled another challenging problem, i.e. the problem of moving data in encrypted form from the message slots into the message polynomial coefficients and back. We called this operation homomorphic batching/unbatching. Via homomorphic batching one can extract coefficients and achieve unpacking operations easily. In addition, the batching operation enabled via a homomorphic NTT operation, which will be of interest for numerous signal processing applications.

References

1. Bos, J.W., Lauter, K., Naehrig, M.: Private predictive analysis on encrypted medical data. Tech. Rep. MSR-TR-2013-81 (September 2013), http://research.microsoft.com/apps/pubs/default.aspx?id=200652
2. Brakerski, Z.: Fully homomorphic encryption without modulus switching from classical gapSVP. IACR Cryptology ePrint Archive 2012, 78 (2012)
3. Brakerski, Z., Gentry, C., Vaikuntanathan, V.: (leveled) fully homomorphic encryption without bootstrapping. In: Proceedings of the 3rd Innovations in Theoretical Computer Science Conference. pp. 309–325. ACM (2012)
4. Brakerski, Z., Vaikuntanathan, V.: Fully homomorphic encryption from ring-LWE and security for key dependent messages. In: Rogaway, P. (ed.) CRYPTO. Lecture Notes in Computer Science, vol. 6841, pp. 505–524. Springer (2011)
5. Brakerski, Z., Vaikuntanathan, V.: Efficient fully homomorphic encryption from (standard) LWE. SIAM Journal on Computing 43(2), 831–871 (2014)
6. Cheon, Jung, H., Miran, K., Kristin, L.: Secure dna-sequence analysis on encrypted dna nucleotides. (2014), http://media.eurekalert.org/aaasnewsroom/MCM/\-FIL_000000001439/EncryptedSW.pdf
7. Dai, W., Doröz, Y., Sunar, B.: Accelerating NTRU based homomorphic encryption using GPUs. In: High Performance Extreme Computing Conference (HPEC), 2014 IEEE. pp. 1–6 (2014)
8. Dai, W., Sunar, B.: cuHE: A homomorphic encryption accelerator library (2015)
9. van Dijk, M., Gentry, C., Halevi, S., Vaikuntanathan, V.: Fully homomorphic encryption over the integers. In: Gilbert, H. (ed.) Advances in Cryptology EUROCRYPT 2010, Lecture Notes in Computer Science, vol. 6110, pp. 24–43. Springer Berlin Heidelberg (2010)
10. Doröz, Y., Hu, Y., Sunar, B.: Homomorphic AES evaluation using the modified LTV scheme. Designs, Codes and Cryptography pp. 1–26 (2015)
11. Doröz, Y., Shahverdi, A., Eisenbarth, T., Sunar, B.: Toward practical homomorphic evaluation of block ciphers using Prince. In: Böhme, R., Brenner, M., Moore, T., Smith, M. (eds.) Financial Cryptography and Data Security, Lecture Notes in Computer Science, vol. 8438, pp. 208–220. Springer Berlin Heidelberg (2014)
12. Gentry, C.: A Fully Homomorphic Encryption Scheme. Ph.D. thesis, Stanford University (2009)
13. 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)
14. Gentry, C., Halevi, S.: Fully homomorphic encryption without squashing using depth-3 arithmetic circuits. IACR Cryptology ePrint Archive 2011, 279 (2011)
15. 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)
16. Gentry, C., Halevi, S., Smart, N.P.: Homomorphic evaluation of the AES circuit. IACR Cryptology ePrint Archive 2012 (2012)
17. Gentry, C., Sahai, A., Waters, B.: Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In: CRYPTO. pp. 75–92. Springer (2013)
18. Graepel, T., Lauter, K., Naehrig, M.: ML confidential: Machine learning on encrypted data. In: Kwon, T., Lee, M.K., Kwon, D. (eds.) Information Security and Cryptology ICISC 2012, Lecture Notes in Computer Science, vol. 7839, pp. 1–21. Springer Berlin Heidelberg (2013)

19. Halevi, S., Shoup, V.: HElib, homomorphic encryption library. Internet Source (2012)
20. 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)
21. 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)
22. Lindner, R., Peikert, C.: Better key sizes (and attacks) for LWE-based encryption 6558, 319–339 (2011), http://dx.doi.org/10.1007/978-3-642-19074-2_21
23. 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)
24. 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)
25. Shoup, V.: NTL: A library for doing number theory. http://www.shoup.net/ntl/ (2001)
26. Smart, N.P., Vercauteren, F.: Fully homomorphic SIMD operations. Designs, codes and cryptography 71(1), 57–81 (2014)
27. Stehlé, D., Steinfeld, R.: Making ntru as secure as worst-case problems over ideal lattices. Advances in Cryptology – EUROCRYPT '11 pp. 27–4 (2011)
28. 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)