Enabling Faster Operations for Deeper Circuits in Full RNS Variants of FV-like Somewhat Homomorphic Encryption

Jonathan Takeshita, Matthew Schoenbauer, Ryan Karl, and Taeho Jung

University of Notre Dame, Notre Dame IN 46556, USA
{jtakeshi,mschoenb,rkarl,tjung}@nd.edu

Abstract. Though Fully Homomorphic Encryption (FHE) has been realized, most practical implementations utilize leveled Somewhat Homomorphic Encryption (SHE) schemes, which have limits on the multiplicative depth of the circuits they can evaluate and avoid computationally intensive bootstrapping. Many SHE schemes exist, among which those based on Ring Learning With Error (RLWE) with operations on large polynomial rings are popular. Of these, variants allowing operations to occur fully in Residue Number Systems (RNS) have been constructed. This optimization allows homomorphic operations directly on RNS components without needing to reconstruct numbers from their RNS representation, making SHE implementations faster and highly parallel.
In this paper, we present a set of optimizations to a popular RNS variant of the B/FV encryption scheme that allow for the use of significantly larger ciphertext moduli (e.g., thousands of bits) without increased overhead due to excessive numbers of RNS components or computational overhead, as well as computational optimizations. This allows for the use of larger ciphertext moduli, which leads to a higher multiplicative depth with the same computational overhead. Our experiments show that our optimizations yield runtime improvements of up to 4.48× for decryption and 14.68× for homomorphic multiplication for large ciphertext moduli.

Keywords: Fully Homomorphic Encryption· B/FV Scheme· RNS Variant· Performance Optimization

1 Introduction

1.1 Homomorphic Encryption

Homomorphic encryption (HE) refers to encryption schemes that allow computations to be done on encrypted messages. Fully homomorphic encryption (FHE), first presented by Gentry [15] in 2009, allows arbitrary computation on ciphertexts. The great utility of FHE comes with a price: the computational overhead involved in performing computations on ciphertexts is high, in some cases prohibitively so. While many different schemes and optimizations have

been introduced in the decade after Gentry's breakthrough, a great deal of research is still focused in improving the computational costs of fully homomorphic encryption.

One possible application of FHE is in settings where data privacy is paramount. For example, suppose a hospital wants to outsource statistical computation over patient data to cloud computing. The main obstacle here is the hospital's needs to protect the confidentiality of patients' data. FHE can be used to encrypt the data, allowing the cloud computer to perform the desired computations on the data without being able to learn anything about any patient's medical information. The (encrypted) results can then be returned to the hospital for decryption.

While FHE has been realized theoretically, the bootstrapping procedure required to allow an arbitrary number of operations on ciphertexts is highly complex and computationally intensive. Therefore, most active research (including this work) focuses on improving the efficiency of the underlying Somewhat Homomorphic Encryption (SHE) scheme of an FHE scheme. In SHE, a ciphertext can be operated on arbitrarily up to a certain multiplicative depth before the noise added from successive multiplications makes correct decryption impossible.

1.2 A brief overview of the state of the art

Many modern cryptographic schemes are based on the difficulty of the Ring Learning-With-Errors problem [19]. In these schemes, ciphertexts are polynomials with potentially very large coefficients. One such scheme is the popular Brakerski/Fan-Vercautaren (B/FV) scheme [13]. The original scheme requires division-with-rounding (DWR) and centered modular reductions on the coefficients of ciphertext polynomials. These operations are not efficiently compatible with Residue Number System (RNS) [14] representations of polynomial coefficients, a common optimization.

The Residue Number System representation uses the Chinese Remainder Theorem to decompose large numbers into tuples of smaller, more easily handled (single-precision, as opposed to arbitrarily-sized) numbers. Individual operations are more efficient on the smaller numbers, and operations on numbers in RNS form can be parallelized. However, some operations frequently used in B/FV, such as DWR and centered modular reduction, cannot be performed directly on numbers in RNS form. In order to perform these operations on numbers stored in RNS form, the numbers would have to be reconstructed, operated upon, and decomposed, which requires more computation.

Two different variants of the original B/FV scheme ameliorate the difficulties with RNS representations of B/FV ciphertexts. The scheme by Bajard et al. [4] uses a series of integer-only operations to compute a fast approximate base conversion to compute DWRs and centered reductions. This optimization is very complex, and does slightly affect the noise added to the ciphertext at each operation. The variant by Halevi et al. [16] replaces the integer-only fast approximate base conversion with floating-point operations to exactly compute base conversions, with no changes to the original noise bounds. Efficient software implementations of these schemes have been written: SEAL by Microsoft

Research [10, 9] implements the variant of Bajard et al., and PALISADE by the PALISADE Project [11] implements the variant of Halevi et al. These variants bring approximately the same improvement [5].

In this work, we show how careful parameter selection can be used to improve upon the RNS variant of Bajard et al. In particular, we replace some multiplications (with super-linear complexity) with faster bitshifting operations (with linear complexity), and some modular reductions (with super-linear complexity) with bitshifting and additions/subtractions (all with linear complexity). As bitshifting is more efficient than multiplication, this improves the runtime of the relevant operations (decryption and homomorphic multiplication). A consequence of this parameter selection is the ability to represent the same size of ciphertext modulus more efficiently with fewer RNS components, thus improving the depth of circuit that can be computed.

Summary of contributions:

We present the following improvements that can be applied to Bajard et al.'s RNS variant of the B/FV scheme:

- Parameter choices that allow quicker calculation of products and moduli in base conversion algorithms, as well as faster decomposition in the variant's relinearization.
- An implementation of our improvements and comparison with the previous state-of-the-art.
- An analysis of when these parameter choices are useful, and the resulting tradeoffs.

2 Background

In this section we give a brief description of the B/FV scheme and the RNS variant of Bajard et al. For a full and informative description, the original works should be referenced.

2.1 Notations

For $x \in \mathbb{R}$, $\lfloor x \rceil, \lfloor x \rfloor, \lceil x \rceil$ indicate rounding to the nearest integer, rounding down, and rounding up respectively. Also, we use plain lowercase letters (e.g., x, y) to denote scalar values and bold lowercase letters (e.g., $\mathbf{x}, \mathbf{y}$) to denote polynomials.

We use R to denote a polynomial ring of the form $\mathbb{Z}[x]/\Phi(x)$, where $\Phi(x)$ is a cyclotomic polynomial of degree n which is a power of 2. R_t is the ring R/R_t, with coefficients in the set $\mathbb{Z}_t = [\frac{-t}{2}, \frac{t}{2}) \cap \mathbb{Z}$. $|x|_t$ is the ordinary modular reduction of x into the set $[0, t) \cap \mathbb{Z}$, i.e., $|x|_t = x - \lfloor \frac{x}{t} \rfloor \cdot t$, and $[x]_t$ is the centered modular reduction of x into the range $\mathbb{Z}_t$, i.e., $[x]_t = x - \lfloor \frac{x}{t} \rceil \cdot t$.

For $\mathbf{x} \in R$, we use $[\mathbf{x}]_t$ to denote an element in R_t which is obtained by applying the centered modular reduction to individual coefficients componentwise.

Most additions and multiplications involving $\mathbf{x} \in R$ or $\mathbf{x} \in R_t$ are closed in R or R_t respectively (i.e., polynomial additions and multiplications with the

polynomial reduction to R_t at the end), where a scalar value is treated as a polynomial with a constant coefficient only. Temporarily, a division is applied to $\mathbf{x} \in R$, e.g., $\frac{a}{b} \cdot \mathbf{x}$, which is performed to every coefficient componentwise. Such a polynomial in $\mathbb{Q}[x]/\Phi(x)$ is immediately mapped to R_t by applying the rounding function as follows: $[\lfloor \frac{a}{b} \cdot \mathbf{x} \rceil]_t$. This is referred to as Division-With-Rounding (DWR).

The bit-length of a number q is denoted as $bit(q) = \lceil \log_2(q) \rceil$. The expansion factor of a ring R is $\delta_R = max(\frac{\|\mathbf{x} \cdot \mathbf{y}\|}{\|\mathbf{x}\| \cdot \|\mathbf{y}\|} : \mathbf{x}, \mathbf{y} \in R)$, which is equal to n for the rings we consider [8].

2.2 The B/FV scheme

Suppose we have parameters n (a power of two), $t > 1$, $q > t$. The B/FV scheme operates on plaintexts in R_t and ciphertexts in R_q^2. The secret key $\mathbf{s}$ is a randomly chosen element of R with coefficients from a 1-bounded distribution. The public key is $(\mathbf{p_0}, \mathbf{p_1}) = ([\mathbf{as} + \mathbf{e}]_q, \mathbf{a})$, where $\mathbf{a}$ is chosen uniformly at random from R_q and $\mathbf{e}$ is also chosen from a 1-bounded distribution.

Encryption To encrypt $[\mathbf{m}]_t \in R_t$, first sample $\mathbf{e_1}, \mathbf{e_2}$ from a 1-bounded distribution and $\mathbf{a}$ from the uniform distribution on R_q. Then compute:

$$(\mathbf{c_0}, \mathbf{c_1}) = ([\Delta[\mathbf{m}]_t, +\mathbf{p_0 u} + \mathbf{e_1}]_q, [\mathbf{p_1 u} + \mathbf{e_2}]_q)$$

where $\Delta = \lfloor \frac{q}{t} \rfloor$.

Decryption To decrypt $(\mathbf{c_0}, \mathbf{c_1}) \in R_q^2$, compute:

$$\mathbf{m} = [\lfloor \frac{t}{q} [\mathbf{c_0} + \mathbf{c_1 s}]_\mathbf{q} \rceil]_t$$

Addition Addition on two ciphertexts $(\mathbf{c_0}, \mathbf{c_1})$, $(\mathbf{c'_0}, \mathbf{c'_1})$ that encrypt $\mathbf{m}, \mathbf{m'}$ respectively leads to a ciphertext pair $(\mathbf{c_0}^+, \mathbf{c_1}^+)$, the decryption of which returns $[\mathbf{m} + \mathbf{m'}]_t$.

Given ciphertexts $(\mathbf{c_0}, \mathbf{c_1})$, $(\mathbf{c'_0}, \mathbf{c'_1})$, homomorphic addition is calculated by computing:

$$(\mathbf{c_0}^+, \mathbf{c_1}^+) = ([\mathbf{c_0} + \mathbf{c'_0}]_q, [\mathbf{c_1} + \mathbf{c'_1}]_q)$$

Multiplication Given ciphertexts $(\mathbf{c_0}, \mathbf{c_1})$, $(\mathbf{c'_0}, \mathbf{c'_1})$ that encrypt $\mathbf{m}, \mathbf{m'}$ respectively leads to a ciphertext pair $(\mathbf{c_0}^\times, \mathbf{c_1}^\times)$, decryption of which returns $[\mathbf{m} \cdot \mathbf{m'}]_t$.

To find $(\mathbf{c_0}^\times, \mathbf{c_1}^\times)$ given $(\mathbf{c_0}, \mathbf{c_1})$, $(\mathbf{c'_0}, \mathbf{c'_1})$, we need to first compute:

$$(\tilde{\mathbf{c_0}}, \tilde{\mathbf{c_1}}, \tilde{\mathbf{c_2}}) = \lfloor \frac{t}{q} (\mathbf{c_0} \cdot \mathbf{c'_0},\ \mathbf{c_0} \cdot \mathbf{c'_1} + \mathbf{c_1} \cdot \mathbf{c'_0},\ \mathbf{c_1} \cdot \mathbf{c'_1}) \rceil$$

This gives us three elements, however a ciphertext needs to have only two elements. To reduce them back down to 2 elements, we need to fold $\tilde{\mathbf{c_2}}$ into the other

two components. This is accomplished through a process called *relinearization* that is performed with *relinearization keys*.

A set of relinearization keys $\mathbf{rlk} = \{\mathbf{rlk}[i]\}_{i\in\{0,\cdots,\ell\}}$ includes individual keys that are polynomials in R_q. Each individual key $\mathbf{rlk}[i]$ is of the form $(\mathbf{rlk_0}[i], \mathbf{rlk_1}[i])$, where $\mathbf{rlk_0} = [([\mathbf{s}^2 \cdot w^i]_q - (\mathbf{e_i} + \mathbf{s} \cdot \mathbf{a_i})]_q$ and $\mathbf{rlk_1} = \mathbf{a_i}$, for $i \in \{0, \cdots, \ell\}$. Here the values $\mathbf{a_i}$ are chosen uniformly at random from R_q (just like the value $\mathbf{a}$ used earlier), and similarly $\mathbf{e_i}$ is chosen from a small error distribution. The terms $\mathbf{e_i}$ are small error vectors and the terms $\mathbf{a_i}$ are uniformly chosen from R_q (analogously to encryption). The multiplication by the base w will cancel out later when an inner product is taken with a number written in base w. We let $\ell = \lfloor \log_w(q) \rfloor + 1$.

The relinearization key can be considered as a masking of $\mathbf{s}^2$ (with a small error), separated in pieces of small norm. We need this to find $(\mathbf{c_0}^\times, \mathbf{c_1}^\times)$ such that $[\mathbf{c_0}^\times + \mathbf{c_1}^\times \cdot \mathbf{s}]_q \approx [\tilde{\mathbf{c_0}} + \tilde{\mathbf{c_1}} \cdot \mathbf{s} + \tilde{\mathbf{c_0}} \cdot \mathbf{s}^2]_q$. Relinearization thus proceeds by computing

$$(\mathbf{c_0}^\times, \mathbf{c_1}^\times) = ([\mathbf{c_0} + \langle D_w(\tilde{\mathbf{c_2}}), rlk_0\rangle]_q, [\mathbf{c_1} + \langle D_w(\tilde{\mathbf{c_2}}), rlk_1\rangle]_q)$$

Here, D_w is the base-w decomposition of its input, i.e.

$$x = \sum_{i=0}^{\ell} w^i \cdot D_w(x)[i] \mod q$$

The decomposition D_w is applied coefficientwise to ring operands here.

Parameters, Depth, and Security The parameters n and q affect both the depth and security of the B/FV scheme. The multiplicative depth L of a function ("circuit") that can be evaluated by the B/FV scheme is bounded above by a quantity related to q. This is shown by Theorem 1 of [13], and written more informatively here:

$$L < \frac{\log(\frac{q}{4}) + \log(t) - \log(\delta_R + 1.25)}{\log(\delta_R) + \log(\delta_R + 1.25) + \log(t)}$$

As q increases, so will the depth of circuit that the scheme can correctly evaluate. However, as q increases, the security of the scheme decreases, and n must be increased to make up for the loss of security [3, 2]. It is thus a natural research goal to attempt to improve the depth of circuit an SHE scheme can evaluate by finding efficient ways to increase both n and q.

2.3 The RNS variant of Bajard et al.

In the B/FV cryptosystem, operations take place on polynomials with large coefficients in $\mathbb{Z}_q$. By the Chinese Remainder Theorem, if q is a product of k pairwise coprime numbers q_i, then $\mathbb{Z}_q$ is isomorphic to the Cartesian product of the rings $\mathbb{Z}_{q_i}$. Numbers in $\mathbb{Z}_q$ can then be represented in RNS form by a k-tuple of numbers in $\mathbb{Z}_{q_i}$, i.e. the i^{th} component of the RNS representation of a

number x is $x_i = |x|_{q_i}$ (or $[x]_{q_i}$, in a centered representation). Because of the isomorphism given by the Chinese Remainder Theorem, addition and multiplication on numbers in $\mathbb{Z}_q$ can be performed by performing addition/multiplication componentwise on the numbers' RNS representations.

If each modulus q_i is small enough to fit into a computer word (typically 64 bits), then each individual operation in an RNS addition/multiplication becomes additions, multiplications, and modular reductions on single computer words. Also, operations on numbers in RNS form can be parallelized, as each component is independent. (This is true even when the moduli are larger than a computer word.) As noted in [4], the precise form of q does not impact the security of the cryptosystem - this is a fact that we use to our advantage later, in choosing a special form for q.

Using RNS decomposition to accelerate B/FV encryption has challenges. Some operations (e.g. DWR) are not easily doable on numbers in RNS form, which would neccessitate the overhead of frequent decomposition and recomposition. The RNS variant presnted by Bajard et al. [4] finds various methods to overcome these challenges, enabling the use of RNS decomposition in B/FV encryption. This variant only presents nontrivial versions of decryption and homomorphic multiplication, as the other operations are easily realized in an RNS system.

The RNS variant of Bajard et al. improves the asymptotic complexity of decryption by a factor of n (from $\mathcal{O}(n^3)$ to $\mathcal{O}(n^2 \log(n))$). The asymptotic complexity of homomorphic multiplication remains the same (as polynomial multiplication dominates the operation's theoretical complexity). In both operations, there is a large practical improvement in runtime (up to 20 times faster for decryption, and 4 times faster for multiplication), due in part to faster individual operations on RNS components.

As a preliminary, we define

$$FastBConv(x, q, B) = (\sum_{i=1}^{k} |x_i \cdot \frac{q_i}{q}|_{q_i} \cdot \frac{q}{q_i} \bmod b) \text{ for } b \in B$$

This quickly converts a number x represented in base q to one in base B, and it is faster because 1) it skips performing an intermediate modular reduction by q and 2) it does not require multiprecision arithmetic if all arguments and moduli are single-precision [1]. (We use FastBConv on multiprecision numbers in our optimizations, rendering nugatory the second point of improvement.) As a consequence, the result in base B will be off by some multiple of q. Depending on the scenario, there are different methods of correcting this. FastBConv is applied coefficientwise to ring operands.

Decryption Suppose we are given $(\mathbf{c_0}, \mathbf{c_1})$ encrypting $[\mathbf{m}]_t$, with the secret key $\mathbf{s}$ (both in RNS base q). Let $\mathbf{x} = \mathbf{c_0} + \mathbf{c_1} \cdot \mathbf{s}$. In the first description of the variant, γ is a number coprime to t, though this is not strictly necessary. Decryption then proceeds as follows:

1. Base conversion: find $\mathbf{s}^{(t)} = |-FastBConv(|\gamma t\mathbf{x}|_q, q, t) \cdot |q^{-1}|_t|_t$ and $\mathbf{s}^{(\gamma)} = |-FastBConv(|\gamma t\mathbf{x}|_q, q, \gamma) \cdot |q^{-1}|_\gamma|_\gamma$
2. Find the centered remainder (the additive error): $\tilde{\mathbf{s}} = [\mathbf{s}^{(\gamma)}]_\gamma$
3. Correct the error and remove a factor of γ: $\mathbf{m}^{(t)} = [(\mathbf{s}^{(t)} - \tilde{\mathbf{s}}^{(\gamma)}) \cdot |\gamma^{-1}|_t]_t$.

The result of this is $\mathbf{m}^{(t)} = [\mathbf{m}]_t$.

Multiplication Suppose we are given ciphertexts $\mathbf{c} = (\mathbf{c_0}, \mathbf{c_1})$, $\mathbf{c}' = (\mathbf{c'_0}, \mathbf{c'_1})$. Let $B_{sk} = B \cup m_{sk}$ be a base coprime to q and $\tilde{m}$, such that B_{sk} is large enough to hold numbers of magnitude up to nq^2 (as required to hold the result of products of elements of R_q). Then the RNS variant's version of homomorphic multiplication is as follows:

1. Use FastBConv to convert $\mathbf{c}, \mathbf{c}'$ from q to $B_{sk} \cup \tilde{m}$. (We now have $\mathbf{c}, \mathbf{c}'$ in $q \cup B_{sk} \cup \tilde{m}$.)
2. Reduce extra multiples of q in the $B_{sk} \cup \tilde{m}$-representation using Small Montgomery Reduction [20, 6]. (The intermediate results are now in $q \cup B_{sk}$.)
3. Compute the polynomial products $(\tilde{\mathbf{c_0}}, \tilde{\mathbf{c_1}}, \tilde{\mathbf{c_2}}) = (\mathbf{c_0} \cdot \mathbf{c'_0}, \mathbf{c_0} \cdot \mathbf{c'_1} + \mathbf{c_1} \cdot \mathbf{c'_0}, \mathbf{c_1} \cdot \mathbf{c'_1})$ and scale by t (in $q \cup B_{sk}$).
4. Do a fast floor (using FastBConv) from $q \cup B_{sk}$ to B_{sk}. (This is an approximation of the DWR operation.)
5. Perform a Shenoy and Kumaresan-like reduction from B_{sk} to q [21]. (This also utilizes FastBConv, though it converts to q, not from it.)

This gives us $(\tilde{\mathbf{c_0}}, \tilde{\mathbf{c_1}}, \tilde{\mathbf{c_2}})$ in base q. The next step is relinearization, to reduce this ciphertext back down to 2 elements.

2.4 Relinearization

Suppose we are given $(\tilde{\mathbf{c_0}}, \tilde{\mathbf{c_1}}, \tilde{\mathbf{c_2}})$ as above. Let the relinearization keys $rlk_{RNS}[0], rlk_{RNS}[1]$ be defined as in Section 4.5 of [4]. (This is essentially a different way of decomposing the masking of the secret key.) Define

$$D_{RNS}(\tilde{\mathbf{c_2}}) = (|\tilde{\mathbf{c_2}} \frac{q_1}{q}|_{q_1}, |\tilde{\mathbf{c_2}} \frac{q_2}{q}|_{q_2}, \cdots, |\tilde{\mathbf{c_2}} \frac{q_k}{q}|_{q_k})$$

(Note the similarity between this decomposition operation and the FastBConv operation.)

We then perform relinearization by computing:

$$([\tilde{\mathbf{c_0}} + \langle D_{RNS}(\tilde{\mathbf{c_2}}), rlk_{RNS}[0] \rangle]_q, [\tilde{\mathbf{c_1}} + \langle D_{RNS}(\tilde{\mathbf{c_2}}), rlk_{RNS}[1] \rangle]_q)$$

3 Mathematical identities

In this section, we present mathematical identities that can be used to allow faster computations in either of the above variants. First, we present a choice of the ciphertext modulus q and its factorization that enables our optimizations.

Lemma 1. *Let $q = 2^g - 1$, and suppose g is a power of two (i.e., $g = 2^x$ for some positive integer x) divisible by 2^{f+1} for some positive integer $f \geq 1$. In other words, $q = 2^{2^x} - 1$ for some integer x that is not less than $f + 1$. Then q can be written with $k = f + 1$ coprime factors $q = (2^{\frac{g}{2}} + 1)(2^{\frac{g}{4}} + 1) \cdots (2^{\frac{g}{2^{f-1}}} + 1)(2^{\frac{g}{2^f}} + 1)(2^{\frac{g}{2^f}} - 1)$.*

We denote the i^{th} factor of q as q_i.

Proof. It is trivial to see by induction that

$$\begin{aligned} q = 2^g - 1 &= (2^{\frac{g}{2}} + 1)(2^{\frac{g}{2}} - 1) \\ &= (2^{\frac{g}{2}} + 1)(2^{\frac{g}{4}} + 1)(2^{\frac{g}{4}} - 1) \\ &= \cdots \\ &= (2^{\frac{g}{2}} + 1)(2^{\frac{g}{4}} + 1) \cdots (2^{\frac{g}{2^{f-1}}} + 1)(2^{\frac{g}{2^f}} + 1)(2^{\frac{g}{2^f}} - 1) \end{aligned}$$

A positive number of the form $2^{2^n} + 1$ is called a *Fermat number*. Not all Fermat numbers are prime, however all Fermat numbers are pairwise coprime [12]. In other words, the sequence of Fermat numbers form an infinite sequence of coprime numbers. Then, all factors of q except the last one are pairwise coprime. Furthermore, $2^{\frac{g}{2^f}} - 1$ can be factored into $(2^{\frac{g}{2^{f+1}}} + 1)(2^{\frac{g}{2^{f+2}}} + 1)(2^{\frac{g}{2^{f+3}}} + 1) \cdots (2^2+1)(2^1+1)(2^1-1)$ with the same induction above. Except $2^1 - 1 = 1$, all of these factors are Fermat numbers that are coprime. Since all Fermat numbers are pairwise coprime, we can finally conclude the last factor of q, $2^{\frac{g}{2^f}} - 1$ is also coprime with all other factors of q.

3.1 Modular Inverses for Multicands

In the computation of the FastBConv algorithm, multiplication by modular inverses is one of the steps that needs to be computed. We show that for moduli and multipliers as chosen above, the multicands take the form of a power of two, allowing us to change the multiplication into a bitshift.

Lemma 2. *The modular inverse of q/q_i modulo q_i (i.e. $|\frac{q_i}{q}|_{q_i}$) is $2^{\frac{g}{2^i} - i}$.*

Proof. In the first case, suppose $i \neq f + 1$. Then $q_i = 2^{\frac{g}{2^i}} + 1$, so $2^{\frac{g}{2^i}} \equiv -1 \pmod{q_i}$. Then q/q_i is the product $(2^{\frac{g}{2}} + 1)(2^{\frac{g}{4}} + 1) \cdots (2^{\frac{g}{2^{f-1}}} + 1)(2^{\frac{g}{2^f}} - 1) \equiv ((-1)^{2^i} + 1)((-1)^{2^{i-1}} + 1) \cdots ((-1)^2 + 1)(-1 - 1) \equiv 2^{i-1} \cdot (-2) \equiv -2^i \pmod{q_i}$. Then $-2^i \cdot 2^{\frac{g}{2^i} - i} = -2^{\frac{g}{2^i}} \equiv -(-1) \equiv 1 \pmod{q_i}$.

In the second case, suppose $i = f + 1$, so $2^{\frac{g}{2^i}} \equiv 1 \bmod q_i$. Then $\frac{q}{q_i} = (2^{\frac{g}{2}} + 1)(2^{\frac{g}{4}} + 1) \cdots (2^{\frac{g}{2^{f-1}}} + 1)(2^{\frac{g}{2^f}} + 1) \equiv (1^{2^i} + 1)(1^{2^{i-1}}) \cdots (1^2 + 1)(1 + 1) \equiv 2^i \pmod{q_i}$. Then $2^i \cdot 2^{\frac{g}{2^i} - i} = 2^{\frac{g}{2^i}} \equiv 1 \pmod{q_i}$.

3.2 Simpler Flooring

In computing the FastBConv algorithm, modular reductions of products are computed. When the moduli and multipliers take certain forms as above, both the computation of products and modular reductions can be expedited. The following lemma shows that some terms calculated in a modular reduction can be found quickly (using bitshifts instead of multiplications) when q is chosen as above.

Lemma 3. *When $x \in [0, q_i)$, $\lfloor \frac{x \cdot |\frac{q_i}{q}|_{q_i}}{q_i} \rfloor = \lfloor \frac{x-b}{2^i} \rfloor$, where b is 0 when $i = f+1$ or $x = 0$ and 1 otherwise.*

Proof. From above, $\frac{q_i}{q} \bmod q_i = 2^{\frac{g}{2^i}-i}$. If $x = 0$, then the equality is trivial. We proceed assuming $x > 0$, i.e. $x - b \geq 0$. In the first case, suppose $i \neq f+1$. Then $\frac{x \cdot |\frac{q_i}{q}|_{q_i}}{q_i} = \frac{x \cdot 2^{\frac{g}{2^i}-i}}{2^{\frac{g}{2^i}}+1}$. Consider $\frac{x \cdot 2^{\frac{g}{2^i}-i}}{2^{\frac{g}{2^i}}+1} - \frac{x-1}{2^i} = \frac{x \cdot (2^{\frac{g}{2^i}}) - x \cdot (2^{\frac{g}{2^i}}) - x + 2^{\frac{g}{2^i}}+1}{(2^{\frac{g}{2^i}}+1)2^i} = \frac{2^{\frac{g}{2^i}}+1}{2^i \cdot 2^{\frac{g}{2^i}}+1} - \frac{x}{2^i \cdot 2^{\frac{g}{2^i}}+1} = \frac{1}{2^i}(1 - \frac{x}{2^{\frac{g}{2^i}}+1}) \in [0, \frac{1}{2})$. Then because $\frac{x \cdot 2^{\frac{g}{2^i}-i}}{2^{\frac{g}{2^i}}+1} > \frac{x-1}{2^i}$ and the distance between these terms is less than $\frac{1}{2}$, $\lfloor \frac{x \cdot 2^{\frac{g}{2^i}-i}}{2^{\frac{g}{2^i}}+1} \rfloor = \lfloor \frac{x-1}{2^i} \rfloor$. (Note that $\frac{x-1}{2^i}$ is at most $\frac{1}{2}$ more than $\lfloor \frac{x-1}{2^i} \rfloor$.)

In the second case, suppose $i = f+1$. Then $\frac{x \cdot |\frac{q_i}{q}|_{q_i}}{q_i} = \frac{x \cdot 2^{\frac{g}{2^i}-i}}{2^{\frac{g}{2^i}}-1}$. Consider $\frac{x \cdot 2^{\frac{g}{2^i}-i}}{2^{\frac{g}{2^i}}-1} - \frac{x}{2^i} = \frac{x \cdot 2^{\frac{g}{2^i}} - x \cdot 2^{\frac{g}{2^i}} + x}{2^i \cdot (2^{\frac{g}{2^i}})} = \frac{1}{2^i} \frac{x}{2^{\frac{g}{2^i}}} \in [0, \frac{1}{2})$. By the same reasoning as in the first case, $\lfloor \frac{x \cdot 2^{\frac{g}{2^i}-i}}{2^{\frac{g}{2^i}}-1} \rfloor = \lfloor \frac{x}{2^i} \rfloor$.

4 Our Optimizations

To apply our optimizations to Bajard et al's RNS variant of the B/FV scheme, we choose RNS coefficients q_i as in Section 3. Other choices of parameters in the B/FV scheme are unaffected - we choose B_{sk}, γ, and $\tilde{m}$ as normal, and optimizations from special choices of these parameters are still available.

As noted in Section 2.3, the precise form of q does not affect the security of the cryptosystem. This is because the RLWE problem is (conjectured to be) difficult for arbitrary q [17].

When choosing q in the optimized case, we factor it into $f+1$ coefficients such that the smallest factor $(2^{\frac{g}{2^f}} - 1)$ fits into 64 bits (the size of a typical machine word). Choosing smaller moduli would most likely not yield much additional benefit.

Note that storing and operating on coefficients modulo q_i requires costly multiprecision arithmetic when q_i is larger than a machine word. We do this because the special properties of this form of q and q_i allow for computational optimizations.

4.1 Base Conversion - Multiplication

When the coefficients q_i are chosen as above, this allows us to apply Lemma 2 to compute the product $x_i \cdot |\frac{q_i}{q}|_{q_i}$ quickly. Finding this product is part of computing FastBConv. Because $|\frac{q_i}{q}|_{q_i}$ is known to be a power of two, this product can be efficiently computed by right bitshifting x_i by $\log_2(|\frac{q_i}{q}|_{q_i}) = \frac{g}{2^i} - i$ bits.

In the B/FV scheme, this optimization is applied in the three places that a fast base conversion from base q is applied. In decryption, this is applied to the FastBConv to $\{t, \gamma\}$. In homomorphic multiplication, the optimization is applied to the FastBConv from q to $B_{sk} \cup \{\tilde{m}\}$ (used to convert to a larger base that can hold the result of a ring multiplication). It is also applied in the FastBConv to B_{sk} that is a part of the fast RNS flooring. In decryption, this saves us $n \cdot k$ multiplications, replacing them with bitshifts. In homomorphic multiplication (without relinearization), this saves $7 \cdot n \cdot k$ multiplications (FastBConv is called on two two-component ciphertexts in base extension, and then on a three-component ciphertext in fast flooring).

It should be noted that it is possible that $\frac{g}{2^i} - i < 0$. In practice, this case should not occur. It means that too many RNS components are being used for too small a modulus, and it is disadvantageous to have such a choice parameters because the machine word size is 64 bits. If RNS component becomes smaller than this size, the overhead is dominated by the number of instructions instead of the complexity of the instruction itself.

4.2 Base Conversion - Fast Modular Reduction

Choosing the coefficients q_i as in Section 3 allows us to use Lemma 3 to quickly compute the reduction $|x_i \cdot \frac{q_i}{q}|_{q_i}$, also required to compute FastBConv. Note that for any numbers a, b, c, $|ab|_c = ab - c\lfloor \frac{ab}{c} \rfloor$. The usefulness of Lemma 3 is in calculating the term $\lfloor \frac{ab}{c} \rfloor$, where for our purposes $a = x_i$, $b = |\frac{q_i}{q}|_{q_i}$, $c = q_i$. From Lemma 3, $\lfloor \frac{x_i \cdot |\frac{q_i}{q}|_{q_i}}{q_i} \rfloor = \lfloor \frac{x_i - b}{2^i} \rfloor$ ($b \in \{0, 1\}$). Thus we can quickly calculate the floor of the product by left bitshifting $x - b$ by i bits. (In the case where x is zero, bitshifting $x - b$ yields the desired result even when $x = 0$ and $b = 1$. The exception for $x = 0$ in Lemma 3 is only to account for the difference between shifting and flooring - shifting rounds towards zero, while flooring rounds down.)

This can be further optimized by recalling that q_i is a power of two, plus or minus one. Then in computing the product of q_i and $\lfloor \frac{x \cdot |\frac{q_i}{q}|_{q_i}}{q_i} \rfloor$, the multiplication can be replaced by a shift and one addition or subtraction.

The modular reduction is then computed by subtracting $q_i \cdot \lfloor \frac{x \cdot |\frac{q_i}{q}|_{q_i}}{q_i} \rfloor$ from $x_i \cdot |\frac{q_i}{q}|_{q_i}$.

As in Section 4.1, this optimization is applied in the three places where a FastBConv from base q is computed. Once the term $\lfloor \frac{x \cdot |\frac{q_i}{q}|_{q_i}}{q_i} \rfloor$ has been found quickly, it can be multiplied by q_i, and the result subtracted from the product $|x_i \cdot \frac{q_i}{q}|_{q_i}$ (computed efficiently in Section 4.1). Again, this saves $n \cdot k$ modu-

lar reductions in decryption and $7 \cdot n \cdot k$ modular reductions in homomorphic multiplication (again not counting relinearization).

4.3 Decomposition for Relinearization

Similar applications of Lemma 2 can be applied to the decomposition of $\tilde{\mathbf{c_2}}$. Recall the RNS decomposition formula used in Bajard et al.'s optimizations is given as

$$D_{RNS}(\tilde{\mathbf{c_2}}) = (|\tilde{\mathbf{c_2}}\frac{q_1}{q}|_{q_1}, |\tilde{\mathbf{c_2}}\frac{q_2}{q}|_{q_2}, \cdots, |\tilde{\mathbf{c_2}}\frac{q_k}{q}|_{q_k})$$

We can utilize the strategies of Sections 4.1 and 4.2 to quickly compute the terms $|\tilde{\mathbf{c_2}} \cdot \frac{q_i}{q}|_{q_i}$ (applied coefficientwise). This allows for the faster computation of relinearization.

In each relinearizations, these optimizations save $n \cdot k$ multiplications (changing them to bitshifts) and $n \cdot k$ modular reductions (changing them to bitshifts and additions/subtractions).

It is worth noting that some implementations do not perform relinearization after each multiplication, but defer it so the necessary computations occur less often [10, 9]. In such cases, the effect of this optimization may not be as noticeable.

4.4 Fewer RNS Components

When using our choices of moduli, fewer RNS coefficients are needed to represent the same size of ciphertext modulus, as shown in Table 1. This is because the number of moduli k needed for some $bit(q)$ increases roughly linearly as $bit(q)$ increases in the basic variant, while with our choices of moduli k increases logarithmically.

Consider a machine word size of MW bits (usually 64 in modern computers). Then in the original variant, the number of RNS coefficients required for some ciphertext modulus q is $k = \lceil \frac{bit(q)}{MW} \rceil$ (unless q fits in a single word, in which case we obviously need only one component). However, in our optimized case we need only $k = 1 + \log_2(\lceil \frac{bit(q)}{MW} \rceil)$ components (again with the exception when $bit(q) \leq MW$).

Having fewer RNS components to operate upon can give a significant improvement to runtime, especially in situations where parallelism or multithreading are unavailable or undesirable.

4.5 Complexity

Our optimizations do not make any improvement to the overall asymptotic complexity of the RNS variant, for the same reason that the variant's version of multiplication does not improve on asymptotic complexity. While our optimizations can speed up some parts of operations, the asymptotic complexity of the operations remains the same due to other portions of the operations having the same

Table 1. RNS Components Required with 64-bit Words

$bit(q)$	k (Regular Moduli)	k (Optimized Moduli)
128	2	2
256	4	3
512	8	4
1024	16	5
2048	32	6
4096	64	7
8192	128	8
16384	256	9
32768	512	10

or a higher asymptotic complexity. In decryption, the operation is still dominated by the Number-Theoretic Transform (NTT) [18] used for polynomial multiplication. Thus decryption's complexity remains $\mathcal{O}(k \cdot n \cdot \log(n)) \approx \mathcal{O}(n^2 \log(n))$, though we note that our factor of k is smaller (see Section 4.4). For multiplication, the asymptotic complexity of the operation is also still dominated by the NTT, and has the same complexity as decryption. While our optimizations do not improve the theoretical efficiency of the RNS variant, they greatly improve its practical efficiency.

5 Implementation and Evaluation

5.1 Implementation

We implemented the relevant portions (decryption and multiplication with relinearization) of the RNS variant of Bajard et al. in C++, using NTL [22] for polynomial and numerical operations. NTL provides efficient polynomial and integer arithmetic, including multiplication. Our implementation allows for choosing parameters normally, or in accordance with our optimizations. With our modified parameters, we can optimize multiplications (as in Sections 4.1 and 4.3) and modular reductions (as in Section 4.2).

Our tests were run on a computer with an AMD EPYC 7451 CPU, running at up to 2.3GHz. The computer had 128GB memory. No multithreading or parallelism was used, in order to gain the most accurate representation of total computation.

5.2 Experimental Results

In our experiments, we tested four versions of our scheme: no optimizations, fast multiplication enabled, fast modular reduction enabled, and all optimizations enabled. The runtime of each operation and relevant suboperations was measured (in ns), and averages were taken. All trials had at least 1000 iterations. We then calculated the speedup of the three cases with some optimization, relative to the baseline case without optimization.

Deeper Circuits We were most interested in the impact our optimizations had with the parameter $bit(q)$ increasing, as this allows for an SHE scheme to evaluate circuits with greater multiplicative depth. Our first set of experiments held n constant (to 1024), and observed the improvement of each of the three cases with some optimization. Figure 2 shows the speedup from our optimizations in decryption, and Figure 5 shows the overall improvement in homomorphic multiplication. Figures 1 and 3 show the improvement our optimizations yielded in the FastBConv subroutine (when used in decryption and homomorphic multiplication, respectively), and Figure 4 shows the improvement in D_{RNS}.

In these tests, our optimizations began to improve upon the unoptimized version at $bit(q) = 128$ for optimized multiplication only, and $bit(q) = 1024$ for optimized modular reduction, with or without optimized multiplication. The amount by which our optimizations improved runtime increased as $bit(q)$ increased, which suggests that our optimizations are well-suited for applications requiring larger $bit(q)$, such as when requiring deeper multiplicative depth. Put simply, our optimizations scale well with $bit(q)$.

We note that using the optimization of modular reduction is worse than using only the optimization of multiplication, though it is still better than not using any optimizations. The exception to this was relinearization, where using both optimizations eventually was more efficient than only optimizing multiplication. As this phenomenon occurs repeatedly, we discuss it in Section 6.1.

In these tests, the greatest improvement in both decryption and homomorphic multiplication came from only optimizing multiplication. In that case, when $bit(q) = 16384$, decryption achieved a speedup of 4.48 times and homomorphic multiplication achieved a speedup of 14.68 times.

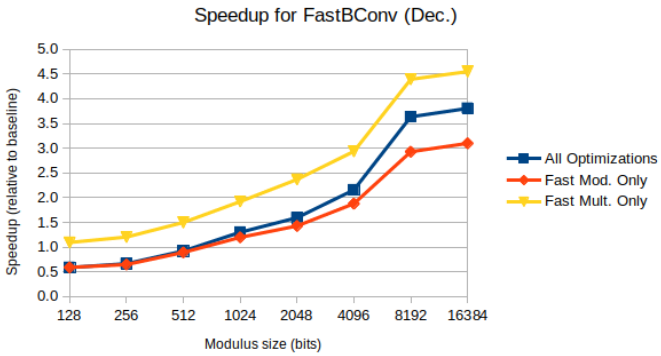

Fig. 1. Speedup for FastBConv (Decryption)

Higher Security As discussed in Section 2.2, increasing the size of the ciphertext modulus decreases the security of the scheme. To counter this, the

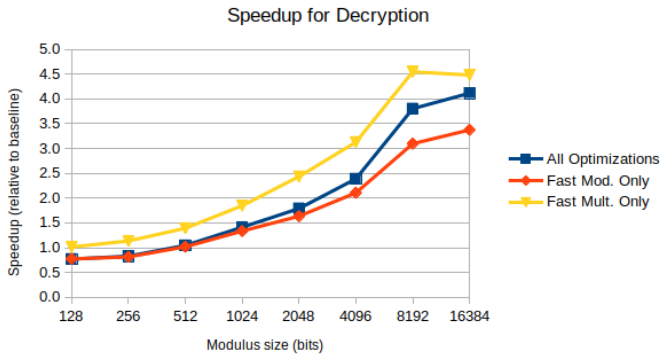

Fig. 2. Speedup for Decryption

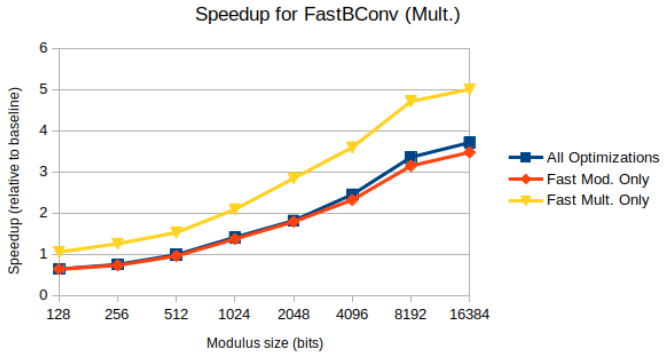

Fig. 3. Speedup for FastBConv (Hom. Mult.)

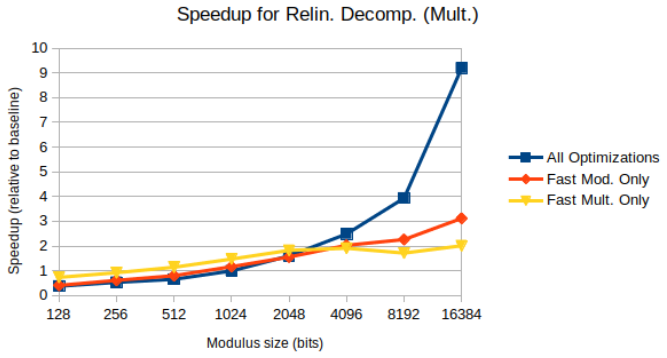

Fig. 4. Speedup for Decomposition in Relinearization (Hom. Mult.)

polynomial modulus degree n (i.e. the degree of $\Phi(x)$) can be increased. It is thus important to discover how our scheme scales with n. To test this, we held

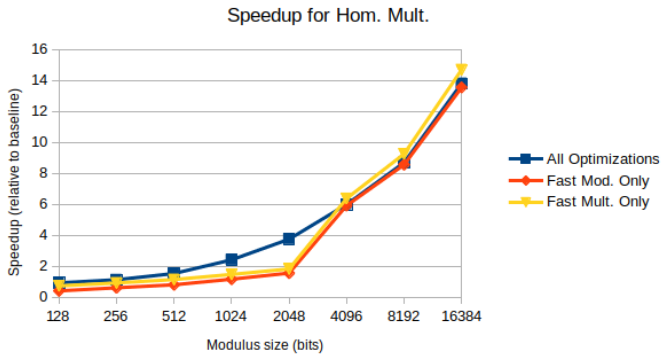

Fig. 5. Speedup for Homomorphic Multiplication

$bit(q)$ constant at 2048 (a value at which our optimizations definitely show some improvement), and tested our improvements with an increasing n in a similar manner to the tests in Section 5.2.

Figures 6 and 7 show how our optimizations scale with n for decryption and homomorphic multiplication, respectively. We note that very little change is seen as n increases. This shows that our optimizations do not incur any additional costs as n increases. Indeed, when n was doubled, the runtimes of the tests almost exactly doubled. Again, we note that optimizing multiplication was far more effective than optimizing modular reduction.

In these tests, the greatest improvement seen was from optimizing only multiplication. In that case, when $n = 16384$, decryption became about 2.15 times faster, and homomorphic multiplication became about 4.07 times faster.

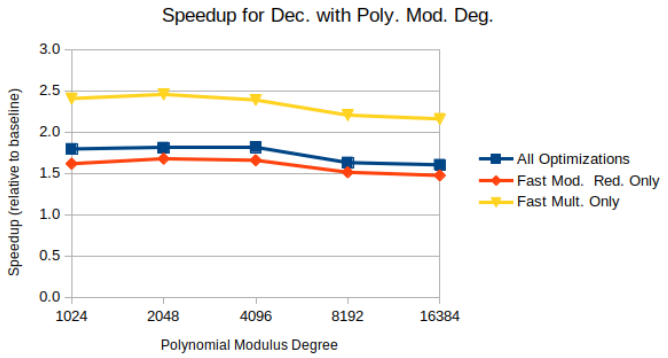

Fig. 6. Speedup in Decryption with respect to Polynomial Modulus Degree

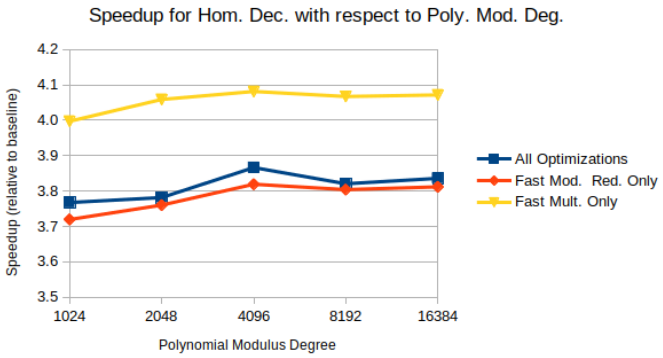

Fig. 7. Speedup in Homomorphic Multiplication with respect to Polynomial Modulus Degree

Using Standard Parameters Having found how our optimizations scale with $bit(q)$ and n, we now turn our attention to how our optimizations improve upon the baseline in standard settings. From the standards given by the Homomorphic Encryption Standardization consortium [2], we choose $(n, bit(q))$ from (8192, 128), (16384, 256), and (32768, 512) - common choices that all guarantee a high security of 256 bits, which is well above the recommended 112 bits [7]. We then conducted testing similarly to the previous tests.

Figures 8 and 9 show the improvement our optimizations brought to decryption and homomorphic multiplication, respectively, for the parameters described above. Once more, optimizing multiplication only brought about the greatest improvement. In that case, when $(n, bit(q)) = (32768, 512)$, decryption became about 1.42 times faster and homomorphic multiplication became about 1.69 times faster. For smaller choices of $(n, bit(q))$, our optimizations were not as useful, and could even result in a worse performance than the baseline. These relatively small improvements (as compared to those in Section 5.2) are because our optimizations bring the greatest improvement with large $bit(q)$, while this set of tests had $bit(q) \leq 512$.

6 Conclusions and Future Work

In this work, we have presented optimizations to the RNS variant of Bajard et al. that allow for faster operations and better scaling with the size of the ciphertext modulus, while preserving good scaling with the size of the polynomial modulus degree.

6.1 Conclusions

1. For larger values of $bit(q)$ (e.g. 1024), our optimizations began to yield much better running times. In some cases with high values for $bit(q)$, our optimizations yielded higher speedups, suggesting that our optimizations are highly

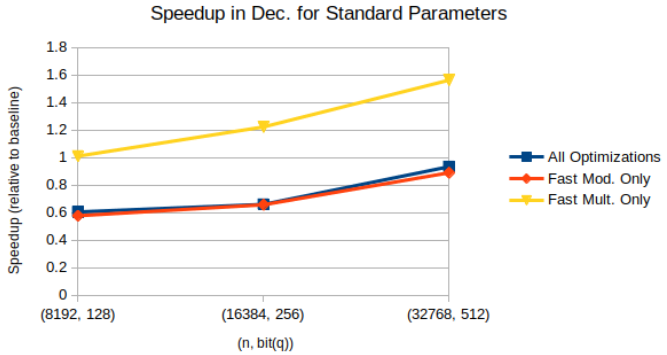

Fig. 8. Improvement in Decryption for Standard Parameters

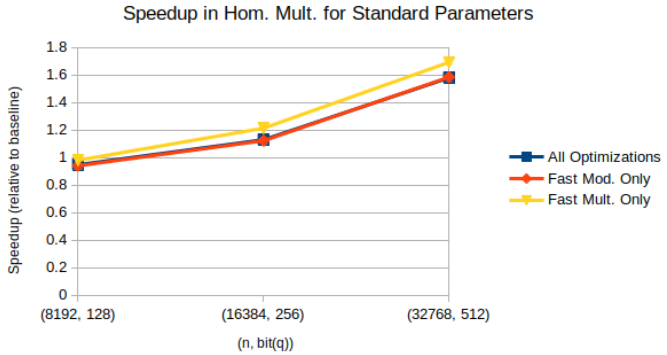

Fig. 9. Improvement in Homomorphic Multiplication for Standard Parameters

effective when a large $bit(q)$ is required for a high multiplicative depth. In most cases, even if $bit(q)$ is not so large, our optimizations can still bring some improvement.

2. For smaller values of q, our optimizations are not as effective as the regular variant. For values of q that we tested less than 512 bits, little significant improvement (in either subroutines or the full procedures of decryption/multiplication) was seen. In a few cases (i.e. for the smallest values of q that we tested), our optimizations led to significant penalties in runtime.
3. In all scenarios, the optimization of multiplication was more effective than the optimization of modular reduction. In many cases, using the fast multiplication alone was faster than using both fast multiplication and fast modular reduction. One possible reason for this is that the number of elementary operations (i.e. high-level functions) required to compute our fast modular reduction is higher than the ordinary method. This necessitates more operations, data movement, and memory allocation, which can slow down computation.

4. Our optimizations also scale as well as could be reasonably expected as n increases, scaling linearly with n with a correlation very close to 1.
5. The previous and first key results combined suggest that our optimizations will scale well as both $bit(q)$ and n increase, which is necessary for both high multiplicative depth and security.

6.2 Future Work

Future related research topics include:

- Finding further optimizations in the choices of parameters such as B_{sk}, γ, and $\tilde{m}$, as well as investigating the best choice of the size of the smallest q_i.
- Further tests of our optimizations' runtime and improvement for very large choices of n and $bit(q)$, to allow for deep circuits without compromising security.
- Investigating the reasons why the optimized modular reduction was not as effective as the optimized multiplication.
- A more lightweight implementation of the B/FV RNS variant with our optimizations, more similar to that of SEAL [10, 9].
- Investigating the effects and relative gains from parallelization with our optimizations.
- Applying this work to allow SHE schemes to use greater multiplicative depth for various privacy-concerned applications.

Acknowledgement

The authors gratefully acknowledge the use of the resources of the Center for Research Computing at The University of Notre Dame. The authors also thank Jean-Claude Bajard (Sorbonne Université) and Kim Laine (Microsoft Research) for their insights.

References

1. Ahmad Qaisar Ahmad Al Badawi, Yuriy Polyakov, Khin Mi Mi Aung, Bharadwaj Veeravalli, and Kurt Rohloff. Implementation and performance evaluation of rns variants of the bfv homomorphic encryption scheme. *IEEE Transactions on Emerging Topics in Computing*, 2019.
2. Martin Albrecht, Melissa Chase, Hao Chen, Jintai Ding, Shafi Goldwasser, Sergey Gorbunov, Shai Halevi, Jeffrey Hoffstein, Kim Laine, Kristin Lauter, Satya Lokam, Daniele Micciancio, Dustin Moody, Travis Morrison, Amit Sahai, and Vinod Vaikuntanathan. Homomorphic encryption security standard. Technical report, HomomorphicEncryption.org, Toronto, Canada, November 2018.
3. Martin R Albrecht, Rachel Player, and Sam Scott. On the concrete hardness of learning with errors. *Journal of Mathematical Cryptology*, 9(3):169–203, 2015.
4. Jean-Claude Bajard, Julien Eynard, M Anwar Hasan, and Vincent Zucca. A full rns variant of fv like somewhat homomorphic encryption schemes. In *International Conference on Selected Areas in Cryptography*, pages 423–442. Springer, 2016.

5. Jean-Claude Bajard, Julien Eynard, Paolo Martins, Leonel Sousa, and Vincent Zucca. Note on the noise growth of the rns variants of the bfv scheme.
6. Jean-Claude Bajard, Julien Eynard, and Nabil Merkiche. Montgomery reduction within the context of residue number system arithmetic. *Journal of Cryptographic Engineering*, 8(3):189–200, 2018.
7. Elaine Barker and Allen Roginsky. Transitioning the use of cryptographic algorithms and key lengths. Technical report, National Institute of Standards and Technology, 2018.
8. Joppe W Bos, Kristin Lauter, Jake Loftus, and Michael Naehrig. Improved security for a ring-based fully homomorphic encryption scheme. In *IMA International Conference on Cryptography and Coding*, pages 45–64. Springer, 2013.
9. Hao Chen, Kyoohyung Han, Zhicong Huang, Amir Jalali, and Kim Laine. Simple encrypted arithmetic library v2. 3.0. *Microsoft*, 2017.
10. Hao Chen, Kim Laine, and Rachel Player. Simple encrypted arithmetic library-seal v2. 1. In *International Conference on Financial Cryptography and Data Security*, pages 3–18. Springer, 2017.
11. Dave Cousins, Kurt Rohloff, Yuriy Polyakov, and Gerard "Jerry" Ryan. The PALISADE lattice cryptography library. `https://palisade-crypto.org/`, 2015–2020.
12. AWF Edwards. Infinite coprime sequences. *The Mathematical Gazette*, 48(366):416–422, 1964.
13. Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. *IACR Cryptology ePrint Archive*, 2012:144, 2012.
14. Harvey L Garner. The residue number system. In *Papers presented at the the March 3-5, 1959, western joint computer conference*, pages 146–153. ACM, 1959.
15. Craig Gentry et al. Fully homomorphic encryption using ideal lattices. In *Stoc*, volume 9, pages 169–178, 2009.
16. Shai Halevi, Yuriy Polyakov, and Victor Shoup. An improved rns variant of the bfv homomorphic encryption scheme. In *Cryptographers' Track at the RSA Conference*, pages 83–105. Springer, 2019.
17. Adeline Langlois and Damien Stehlé. Hardness of decision (r) lwe for any modulus. Technical report, Citeseer, 2012.
18. Patrick Longa and Michael Naehrig. Speeding up the number theoretic transform for faster ideal lattice-based cryptography. In *International Conference on Cryptology and Network Security*, pages 124–139. Springer, 2016.
19. Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 1–23. Springer, 2010.
20. Peter L Montgomery. Modular multiplication without trial division. *Mathematics of computation*, 44(170):519–521, 1985.
21. AP Shenoy and Ramdas Kumaresan. Fast base extension using a redundant modulus in rns. *IEEE Transactions on Computers*, 38(2):292–297, 1989.
22. Victor Shoup et al. NTL: A library for doing number theory (2001). *URL: http://www. shoup. net/ntl*, 2001.