Revisiting Homomorphic Encryption Schemes for Finite Fields

Andrey Kim, Yuriy Polyakov, Vincent Zucca

Revisiting Homomorphic Encryption Schemes for Finite Fields

Andrey Kim[1,2], Yuriy Polyakov[1,3], and Vincent Zucca[4,5,6]

[1]New Jersey Institute of Technology, Newark, USA
[2]Samsung Advanced Institute of Technology, Suwon, Republic of Korea
[3]Duality Technologies, Newark, USA
[4]DALI, Université de Perpignan Via Domitia, France
[5]LIRMM, Univ Montpellier, Montpellier, France
[6]imec-COSIC, KU Leuven, Belgium

September 18, 2021

Abstract

The Brakerski-Gentry-Vaikuntanathan (BGV) and Brakerski/ Fan-Vercauteren (BFV) schemes are the two main homomorphic encryption (HE) schemes to perform exact computations over finite fields and integers. Although the schemes work with the same plaintext space, there are significant differences in their noise management, algorithms for the core homomorphic multiplication operation, message encoding, and practical usability. The main goal of our work is to revisit both schemes, focusing on closing the gap between the schemes by improving their noise growth, computational complexity of the core algorithms, and usability. The other goal of our work is to provide both theoretical and experimental performance comparison of BGV and BFV.

More precisely, we propose an improved variant of BFV where the encryption operation is modified to significantly reduce the noise growth, which makes the BFV noise growth somewhat better than for BGV (in contrast to prior results showing that BGV has smaller noise growth for larger plaintext moduli). We also modify the homomorphic multiplication procedure, which is the main bottleneck in BFV, to reduce its algorithmic complexity. Our work introduces several other novel optimizations, including lazy scaling in BFV homomorphic multiplication and an improved BFV decryption procedure in the Residue Number System (RNS) representation. We also develop a usable variant of BGV as a more efficient alternative to BFV for common practical scenarios.

We implement our improved variants of BFV and BGV in PALISADE and evaluate their experimental performance for several benchmark computations. The experimental results suggest that our BGV implementation is faster for intermediate and large plaintext moduli, which are often used in practical scenarios with ciphertext packing, while our BFV implementation is faster for small plaintext moduli.

Contents

1 Introduction

Homomorphic encryption (HE) is a powerful cryptographic primitive that enables performing computations over encrypted data without having access to the secret key. The HE research area has seen a lot of progress since the formulation of the first fully homomorphic encryption construction by Gentry in 2009 [19], and the schemes implemented in modern HE libraries are multiple orders of magnitude faster than the initial implementation of Gentry's scheme [20]. The most common HE schemes are typically grouped into three classes based on the data types they support computations on. The first class primarily works with Boolean circuits and decision diagrams, similar to the original Gentry scheme, and includes the FHEW and TFHE schemes [13, 17]. The second class supports modular arithmetic over finite fields, which typically correspond to vectors of integers mod t, where t is a prime power commonly called as the plaintext modulus. The second class is also sometimes used for small-integer arithmetic. This class includes Brakerski-Gentry-Vaikuntantan (`BGV`) and Brakerski/Fan-Vercauteren (`BFV`) schemes [9, 10, 18]. The third, and most recent, class supports approximate computations over vectors of real and complex numbers, and is represented by the Cheon-Kim-Kim-Song (CKKS) scheme [12]. All these schemes are based on the hardness of the Ring Learning With Errors (RLWE) problem, where noise is added during encryption and key generation to achieve the hardness properties. The noise grows as encrypted computations are performed, and the main functional parameter in all these schemes, the ciphertext modulus Q, needs to be large enough to accommodate the noise growth, or a special bootstrapping procedure may be used to reset the noise and keep the value of Q relatively small.

Our work focuses on the HE schemes of the second class. Although the `BGV` and `BFV` schemes work with the same plaintext algebra, they use different strategies for encoding the message composed of integers in $\mathbb{Z}_t$ and controlling the noise. The `BGV` scheme encodes the message in the least significant digit (LSD) of integers in $\mathbb{Z}_Q$ and applies the modulus switching technique to keep the noise magnitude constant, i.e., it scales down Q by a factor that corresponds to the noise added after the previous modulus switching call. The `BFV` scheme encodes the message in the most significant digit (MSD) of integers in $\mathbb{Z}_Q$ and uses a special form of homomorphic multiplication, where ciphertext polynomials are multiplied without modular reduction and then scaled down by Q/t. In `BFV`, the value of Q is typically constant and the noise magnitude increases at a rate similar to how Q decreases in `BGV`. The difference in noise management strategies between `BGV` and `BFV` affects the noise growth and efficiency of the schemes. Costache and Smart performed a noise growth comparison, which suggested that `BGV` has better noise growth for larger t than `BFV` [15]. However, the authors did not examine the computational complexity difference, and it has not been clear up to this moment how the schemes compare in terms of practical performance, both from the perspective of computational complexity and actual experimental measurements.

The main goal of this paper is to present improved variants of `BFV` and `BGV` schemes, which also close the gap between the schemes. The other goal is to compare the theoretical complexity of their primitive operations, and experimental performance of `BGV` and `BFV` for several different scenarios using our software implementation in the PALISADE library [2].

Modified BFV Scheme. We propose two modifications for the `BFV` scheme. The first modification deals with encryption, and the second modification revises the homomorphic multiplication operation. The net effects of these modifications are smaller noise growth and faster homomorphic multiplication in `BFV`.

The encryption in BFV can be represented as $\boldsymbol{a} \cdot \boldsymbol{s} + \boldsymbol{e} + \Delta \boldsymbol{m}$ (for simplicity, we focus here on the secret-key formulation), where $\boldsymbol{a}$ is a uniformly random ring element in cyclotomic ring $\mathcal{R}_Q$, $\boldsymbol{s}$ and $\boldsymbol{e}$ are the secret key and Gaussian noise ring elements in $\mathcal{R}$, $\boldsymbol{m}$ is a message in $\mathcal{R}_t$, and $\Delta = \lfloor Q/t \rfloor$ is the scaling factor. Our analysis shows that the difference between Δ and Q/t, which is often described in terms of $r_t(Q) := Q - t\Delta$, brings about a significant error (proportional to $r_t(Q)$) that affects the first homomorphic multiplication and increases the noise growth in BFV as compared to BGV for larger t. If this error is removed, i.e., $r_t(Q) \approx 0$, the noise growth in BFV becomes the same, or actually somewhat better, as in BGV. In view of this, our first modification suggested for BFV is to replace the encryption operation in BFV with $\boldsymbol{a} \cdot \boldsymbol{s} + \boldsymbol{e} + \left\lceil \frac{Q}{t} \boldsymbol{m} \right\rfloor$, which is a more natural choice as compared to the one in the original BFV. This encryption function also significantly simplifies the noise analysis and estimates for BFV homomorphic multiplication. Note that this modification can be likewise applied to the original Brakerski LWE scheme [9].

The most expensive operation in BFV is homomorphic multiplication as it requires a multiplication of two ciphertexts $\mathbf{c}_1$ and $\mathbf{c}_2$ without modular reduction, followed by scaling the results of the tensor product by t/Q. Algorithmically, this requires extending both ciphertexts to modulus QP, where P is sufficiently larger than Q, performing a tensor product which involves expensive Number Theoretic Transforms (NTTs), scaling down the result by Q/t, and finally switching the scaling result from P to Q. We propose a more efficient procedure for homomorphic multiplication where the values of $P \approx Q$, which saves some expensive modulus extension operations and NTTs. The main idea is to apply modulus switching to one of the ciphertexts, e.g., $\mathbf{c}_2$, to change it from Q to P (denote it as $\mathbf{c}_2'$), and then do scaling by t/P after the tensor product. This removes the requirement for extending the scaling result from P to Q (it will already be in Q) at the expense of doing a smaller number of modulus extensions during the modulus switching of $\mathbf{c}_2$. The other benefit is that the tensor product of $\mathbf{c}_1$ and $\mathbf{c}_2'$ can be scaled by t/P directly in PQ, i.e., we have a tensor product mod PQ instead of a tensor product without modular reduction.

We also introduce a leveled version of BFV homomorphic multiplication, where ciphertexts modulo a larger modulus Q are internally scaled down to a smaller modulus Q_ℓ (or P_ℓ), the standard homomorphic multiplication operations are performed, and then the results are scaled back up to Q. The benefit of this approach is that the ciphertexts still look the same (modulo Q) outside the homomorphic multiplication operations, but we get BGV-like benefits of working with smaller moduli in multiplication. The combined effect of our improvements in homomorphic multiplication is the speed-up of up to 4x, as compared to a prior state-of-the-art BFV implementation, when dealing with multiplications at deeper levels of computation.

BFV Scheme Optimizations. We also introduce several algorithmic optimizations that equally apply to the classical BFV and our modified variant. The first optimization is for the scenarios where we need to add multiple BFV ciphertexts that were just obtained by BFV multiplication. The standard way is to perform many expensive BFV multiplications and then add up the result. However, we can delay the scaling by t/Q (or by t/P in our BFV variant) in each homomorphic multiplication until the sum is computed, and then just do one scaling at the end. This saves many expensive NTTs and modulus extension operations. We denote this optimization as *lazy scaling*. The lazy scaling can be combined with previously known lazy relinearization to push most of the expensive computations in a homomorphic multiplication, i.e., scaling and relinearization, to the end, after the aggregation is done.

Some of the other optimizations apply to Residue Number Systems (RNS) variants of BFV,

where multi-precision integers in $\mathbb{Z}_Q$ are split into vectors of smaller integers using the Chinese Remainder Theorem (CRT) to perform operations efficiently using native (64-bit) integer types. The RNS variants are now predominately used in practice, and are implemented in the SEAL [31], PALISADE [2], and Lattigo [1] software libraries. There are two main RNS variants of `BFV`: the Bajard-Eynard-Hasan-Zucca (BEHZ) variant based on modular integer arithmetic and Montgomery reductions and the Halevi-Polyakov-Shoup (HPS) variant based on a combination of modular integer arithmetic and floating-point approximations [6, 23].

A significant limitation of the HPS approach is that high-precision ("long double" or even quad-precision) floating-point arithmetic is required to support larger CRT moduli: long doubles are needed for CRT moduli from 47 to 58 bits, and quad-precision floats are needed for higher CRT moduli [23]. We introduce a general-purpose digit decomposition technique (inspired by digit decomposition in key switching) and apply it to the HPS decryption procedure to add support for arbitrary CRT moduli using only regular double-precision floating-point arithmetic, thus overcoming this limitation of the HPS variant. This digit decomposition technique can be applied to other mixed integer/floating-point RNS operations to reduce precision requirements for floating-point arithmetic.

We also apply the full RNS variant of hybrid key switching [26] recently proposed for the CKKS scheme to both `BFV` RNS variants, and demonstrate how some auxiliary CRT moduli needed for homomorphic multiplication can be reused for hybrid key switching. This key switching method has some benefits (smaller noise growth, better efficiency for deeper computations) over the residue decomposition key switching method previously used in both RNS variants of `BFV`.

BGV Scheme Optimizations and Usability Improvements. We use the Gentry-Halevi-Smart (GHS) variant of `BGV` as the basis for our `BGV` instantiation [22, 25]. Although the original GHS variant performs some operations in RNS, it still uses multiprecision integer arithmetic for key switching and some scenarios of modulus switching. For instance, although the GHS paper originally introduced the hybrid key switching technique, the authors used multiprecision arithmetic for the digit decomposition step. We apply the full RNS version of hybrid key switching to our `BGV` instantiation and eliminate any multiprecision arithmetic from our `BGV` implementation, thus significantly improving its efficiency.

One of the challenges in the GHS variant is the need to perform dynamic noise estimation, which makes the `BGV` implementation less robust and usable as compared to the `BFV` variants where noise estimation is typically needed only at the parameter generation phase. We develop a more usable and robust variant of `BGV` that is essentially as simple to use as the current `BFV` implementations. This variant only needs to know the multiplicative depth and maximum number of additions per level for many common scenarios. The main advantage of this `BGV` variant is that it is significantly faster than our `BFV` implementations for certain practical scenarios, yet its usability matches that of `BFV`.

Implementation and Performance Comparison. We implement the improved variants of `BFV` and `BGV` in PALISADE, and provide their comparison for specific benchmark computations. To the best of our knowledge, this is the first publicly available implementation of both schemes in the same software library. We also perform theoretical comparison of the computational complexity for the operations that differ between `BFV` and `BGV`.

The comparison results can be summarized as follows:

- Our improved variant of BFV has somewhat better noise growth than BGV, in contrast to prior results for the original BFV scheme that showed better noise growth for BGV at larger plaintext moduli [15].
- Our best variant of BFV is faster than BGV for small plaintext moduli, while BGV is faster for intermediate and large plaintext moduli used in many practical scenarios.
- The speed-up in homomorphic multiplication of our best BFV variant compared to a prior state-of-the-art RNS implementation of BFV goes up to 4x for deeper computations.

Related work. Costache et al. further examine the difference between the noise growth in BFV and BGV [14] to improve the analysis presented in [15]. They explore an alternative heuristic noise analysis approach to obtain tighter noise bounds. We point out that this new analysis has some inacurracies, e.g., the effect of extra noise due to $r_t(Q)$ in BFV homomorphic multiplication is not accounted for. We show that this extra noise determines the higher noise growth in BFV for large plaintext moduli, and demonstrate how this noise is removed in our BFV variant. Moreover, we show that this analysis can be carried out independently of the chosen heuristic for noise analysis. The authors also do not consider the difference in the complexity of homomorphic encryption operations between BGV and BFV, which affects their conclusions. In view of the above, we primarily compare our results with the prior work [15].

The encoding of a message in the MSD of a ciphertext as $\lceil \frac{Q}{t}\boldsymbol{m} \rfloor$ was already used in the Key Encapsulation Mechanism (KEM) Kyber [8]. But in the case of Kyber, the plaintext modulus $t = 2$, i.e., the coefficients of the messages are either 0 or 1. Therefore, the messages can be recovered directly during decryption by checking whether the coefficients are closer to $\lceil Q/2 \rfloor$ or 0, and the noise is not affected.

SEAL also independently added to v3.4.0 a modification of BFV encryption similar to what we describe in our work [31]. However, no underlying noise analysis was presented, and the prior paper related to SEAL [14] included noise analysis inaccuracies involving the rounding term $r_t(Q)$, suggesting that the full effect of this change was not well-understood.

Note that the FHEW and TFHE schemes can also support arithmetic over finite fields for small plaintext moduli (typically up to 4 bits) [30], and can be considered as an alternative to BGV and BFV for these scenarios. These schemes support fast bootstrapping (the latency is much lower than for BGV and BFV bootstrapping [24]), but their main limitation is the lack of support for CRT packing, which makes the BGV/BFV approach much more appealing when large arrays of numbers need to be computed on/bootstrapped because one ciphertext operation can perform thousands of integer operations at once.

Organization. The rest of the paper is organized as follows. In Section 2 we provide the necessary background on BGV and BFV. In Sections 3 and 4, we present our improved variants of BFV and BGV, respectively. Section 5 includes the theoretical comparison of the schemes, and discussion of the experimental results. Section 6 provides the conclusions and outlines the ideas for future work.

2 Background

All logarithms are expressed in base 2 if not indicated otherwise. Let N be a power of two. We denote the $2N$-th cyclotomic ring $\mathcal{R} = \mathbb{Z}[X]/(X^N + 1)$ and $\mathcal{R}_Q := \mathcal{R}/Q\mathcal{R}$. [1] Ring elements are indicated in bold, e.g. $\boldsymbol{a}$. For an integer $Q > 1$, we identify the ring $\mathbb{Z}_Q$ with $(-Q/2, Q/2] \cap \mathbb{Z}$ as a representative interval and for $z \in \mathbb{Z}$, $[z]_Q \in \mathbb{Z}_Q$ denotes the centered remainder of z modulo Q, while $r_Q(z)$ denotes the classical Euclidean remainder in $[0, Q) \cap \mathbb{Z}$. For $x \in \mathbb{Q}$, $\lfloor x \rfloor$, $\lfloor x \rceil$ and $\lceil x \rceil$ denote the rounding to the lower, closest and higher integer, respectively. We extend these notations to elements of $\mathcal{R}$ by applying them coordinate-wise. For $\boldsymbol{a} = a_0 + a_1 \cdot X + \cdots + a_{N-1} \cdot X^{N-1} \in \mathcal{R}$, we denote the ℓ_∞ norm of $\boldsymbol{a}$ as $\|\boldsymbol{a}\|_\infty = \max_{0 \leq i < N}\{|a_i|\}$. There exists a constant $\delta_\mathcal{R}$ such that $\|\boldsymbol{a} \cdot \boldsymbol{b}\|_\infty \leq \delta_\mathcal{R} \|\boldsymbol{a}\|_\infty \|\boldsymbol{b}\|_\infty$ for any $(a, b) \in \mathcal{R}^2$. It is well-known that for $\mathcal{R} = \mathbb{Z}[X]/(X^N + 1)$, $\delta_\mathcal{R} = N$. However in practice this bound is only reached with exponentially low probability. As shown in [23], the bound $\delta_\mathcal{R} = 2\sqrt{N}$ is much closer to what we observe experimentally, and can be used to achieve tighter noise bounds. Another approach consists in estimating the noise size using the canonical embedding norm [22], as currently done in HElib [25]. Nonetheless, in this work we estimate the noise size using the expansion factor $\delta_\mathcal{R}$ with the method of [23] as it is simpler and precise enough for our purpose.

We use $\boldsymbol{a} \leftarrow \chi$ to denote the sampling of $\boldsymbol{a} \in \mathcal{R}$ according to a distribution χ. χ_{key} denotes the *uniform ternary* distribution, i.e., all the coefficients of $\boldsymbol{a} \leftarrow \chi_{\text{key}}$ are selected uniformly and independently from $\{-1, 0, 1\}$. This distribution is commonly used for secret key generation as it is the most efficient option conforming to the HE standard [4]. $\chi_{\texttt{err}}$ denotes a *discrete Gaussian* distribution with standard deviation $\sigma_{\texttt{err}}$, i.e. all the coefficients of $\boldsymbol{a} \leftarrow \chi_{\texttt{err}}$ are selected independently from a truncated discrete Gaussian distribution with standard deviation σ_{err}. Truncated discrete Gaussian distributions are commonly used to generate error polynomials to meet the desired hardness requirement [4]. We assume that the polynomials sampled from $\chi_{\texttt{key}}$ and $\chi_{\texttt{err}}$ have their coefficients bounded by $B_{\texttt{key}} = 1$ and $B_{\texttt{err}} = 6\sigma_{\texttt{err}}$, respectively. Although a Gaussian distribution is not bounded by nature, the probability for a Gaussian coefficient to be larger than $B_{\texttt{err}} = 6\sigma_{\texttt{err}}$, is less than 2^{-30}, therefore the two distributions are very close in practice. $\mathcal{U}_Q$ denotes the *uniform distribution* over $\mathcal{R}_Q$, where every coefficient of $\boldsymbol{a}$ is sampled uniformly and independently from $\mathbb{Z}_Q$.

2.1 Plaintext Space

We are interested in the `BGV` and `BFV` homomorphic encryption schemes which both share the same plaintext space $\mathcal{R}_t$ for some integer $t > 1$. Hence, the most natural way to represent plaintext messages of these schemes is to think of them as vectors of size N with their coefficients taken modulo t. However, $\mathcal{R}_t$ has many algebraic properties, in particular when $t = p^r$ is a prime power with p coprime to $2N$. In this case $\mathcal{R}_t$ is actually a $\mathbb{Z}_t$-algebra, which means that it contains a subring isomorphic to $\mathbb{Z}_t$. In this paper we focus on the case $r = 1$, where $t = p$ is a prime. The interested reader can nonetheless refer to [25] for further details regarding the general case. $\mathbb{Z}_t$-algebra supports efficent Single-Instruction Multiple-Data (SIMD) packing/batching. For more details on the packing, the reader is referred to [32].

[1] more general cyclotomic rings are also supported, and all results of our work equally apply to these non-power-of-two rings; please see [25] for more details on general cyclotomic rings

2.2 Homomorphic Encryption Schemes for Finite Field Arithmetic

The two schemes studied in this work: `BGV` and `BFV` are actually two instantiations of the same idea, and share, therefore, many common features. First, according to the desired security level λ and the targeted application, one starts by selecting public parameters for the considered scheme: ring dimension $N = 2^d$, the plaintext modulus t, a ciphertext modulus Q and two probability distributions χ_{key} and χ_{err} on the ring $\mathcal{R}$. In both cases, the secret key will be an element $\boldsymbol{s} \leftarrow \chi_{\texttt{key}}$. Note that `BGV` and `BFV` may be viewed as different modes of a unified scheme, where the ciphertexts may be switched from one mode/scheme to the other (see the full version for details).

2.2.1 Original BGV Scheme

In 2011, Brakerski et al. designed a leveled homomorphic scheme, namely capable of evaluating circuits of arbitrary size, but known beforehand [10]. The key tool of their construction is the *modulus switching* procedure which allows to switch a ciphertext `ct` encrypted under a modulus Q to a smaller modulus Q' in order to maintain the noise level "constant". As a consequence, one must select a chain of $L+1$ moduli $Q_0 \mid Q_1 \mid \ldots \mid Q_L = Q$ such that t and Q_L are coprime. The public key is formed as:

$$\texttt{pk} = \left([\boldsymbol{a} \cdot \boldsymbol{s} + t\boldsymbol{e}]_{Q_L}, -\boldsymbol{a} \right) \in \mathcal{R}_{Q_L}^2,$$

which is equivalent to the Ring-LWE sample $([\boldsymbol{a}/t \cdot \boldsymbol{s} + \boldsymbol{e}]_{Q_L}, [-\boldsymbol{a}/t]_{Q_L})$ (since t and Q_L are coprime) associated to $\boldsymbol{s}$ and Q_L with $\boldsymbol{a} \leftarrow \mathcal{U}_{Q_L}$ and $\boldsymbol{e} \leftarrow \chi_{\texttt{err}}$.

A ciphertext $\texttt{ct} = (\boldsymbol{c}_0, \boldsymbol{c}_1) \in \mathcal{R}_Q^2$ corresponds to a degree 1 polynomial whose coefficients lie in $\mathcal{R}_Q$. The message $\boldsymbol{m} \in \mathcal{R}_t$ is hidden in the LSD of the first coefficient $\boldsymbol{c}_0$ of the ciphertext as follows:

$$\texttt{ct} = \left([[\boldsymbol{m}]_t + \boldsymbol{u} \cdot \mathbf{pk}_0 + t\boldsymbol{e}_0]_{Q_L}, [\boldsymbol{u} \cdot \mathbf{pk}_1 + t\boldsymbol{e}_1]_{Q_L} \right)$$

with $\boldsymbol{u} \leftarrow \chi_{\texttt{key}}$ and $\boldsymbol{e}_0, \boldsymbol{e}_1 \leftarrow \chi_{\texttt{err}}$. The noise contained in a ciphertext $\texttt{ct} = (\boldsymbol{c}_0, \boldsymbol{c}_1)$ appears explicitly once the ciphertext is evaluated on the secret key $\boldsymbol{s}$:

$$\boldsymbol{c}_0 + \boldsymbol{c}_1 \cdot \boldsymbol{s} = [\boldsymbol{m}]_t + t(\boldsymbol{u} \cdot \boldsymbol{e} + \boldsymbol{e}_1 \cdot \boldsymbol{s} + \boldsymbol{e}_0) = [\boldsymbol{m}]_t + t\boldsymbol{v}_{\texttt{fresh}} \bmod Q_L, \tag{1}$$

where the term $\boldsymbol{v}_{\texttt{fresh}} = \boldsymbol{u} \cdot \boldsymbol{e} + \boldsymbol{e}_1 \cdot \boldsymbol{s} + \boldsymbol{e}_0$ is the noise inherent to a "freshly" encrypted ciphertext. Since $Q_0 \mid Q_1 \mid \ldots \mid Q_L$, encryptions can be performed equivalently at any level i, i.e., modulo Q_i.

To decrypt a ciphertext $\texttt{ct} = (\boldsymbol{c}_0, \boldsymbol{c}_1) \in \mathcal{R}_{Q_i}^2$ with $i \in [0, L]$, one computes $\boldsymbol{m}' = [\boldsymbol{c}_0 + \boldsymbol{c}_1 \cdot \boldsymbol{s}]_{Q_i}$ and then outputs $[\boldsymbol{m}']_t$. To ensure correctness of the decryption, the noise $\boldsymbol{v}$ must be "small enough" such that $\boldsymbol{m}' = [\boldsymbol{m}]_t + t\boldsymbol{v}$ does not wrap-around modulo Q_i. As a consequence, decryption remains correct as long as:

$$\|\boldsymbol{v}\|_\infty < \frac{Q_0}{2t} - \frac{1}{2}.$$

One can add two ciphertexts `ct` and `ct'` encrypting $\boldsymbol{m}$ and $\boldsymbol{m}'$, respectively, at the same level i to yield:

$$\boldsymbol{c}_0 + \boldsymbol{c}_0' + (\boldsymbol{c}_1 + \boldsymbol{c}_1') \cdot \boldsymbol{s} = [\boldsymbol{m} + \boldsymbol{m}']_t + t(\boldsymbol{v} + \boldsymbol{v}' + \boldsymbol{u}) \bmod Q_i,$$

with $\|\boldsymbol{u}\|_\infty \leq 1$. This means that

$$\texttt{ct}_{\texttt{add}} = \left([\boldsymbol{c}_0 + \boldsymbol{c}_0']_{Q_i}, [\boldsymbol{c}_1 + \boldsymbol{c}_1']_{Q_i} \right)$$

is a level-i encryption of $[\boldsymbol{m}+\boldsymbol{m}']_t$ and its noise is almost the sum of the noises of $\mathtt{ct}$ and $\mathtt{ct}'$:

$$\|\boldsymbol{v}_{\mathtt{add}}\|_\infty = \|\boldsymbol{v}+\boldsymbol{v}'+\boldsymbol{u}\|_\infty \leq \|\boldsymbol{v}\|_\infty + \left\|\boldsymbol{v}'\right\|_\infty + 1.$$

Similarly to addition, we can multiply two level-i ciphertexts $\mathtt{ct}$ and $\mathtt{ct}'$ to obtain the following congruence modulo Q_i:

$$(\boldsymbol{c}_0+\boldsymbol{c}_1\cdot\boldsymbol{s})\cdot(\boldsymbol{c}'_0+\boldsymbol{c}'_1\cdot\boldsymbol{s}) = [\boldsymbol{m}\cdot\boldsymbol{m}']_t + t([\boldsymbol{m}]_t\cdot\boldsymbol{v}'+\boldsymbol{v}\cdot[\boldsymbol{m}']_t+t\boldsymbol{v}\cdot\boldsymbol{v}'+\boldsymbol{r}_m)$$

with $[\boldsymbol{m}]_t\cdot[\boldsymbol{m}']_t = [\boldsymbol{m}\cdot\boldsymbol{m}']_t + t\boldsymbol{r}_m$ and $\|\boldsymbol{r}_m\|_\infty \leq \delta_{\mathcal{R}} t/2$. This means that

$$\mathtt{ct}_{\mathtt{mult}} = \left([\boldsymbol{c}_0\cdot\boldsymbol{c}'_0]_{Q_i}, [\boldsymbol{c}_0\cdot\boldsymbol{c}'_1+\boldsymbol{c}_1\cdot\boldsymbol{c}'_0]_{Q_i}, [\boldsymbol{c}_1\cdot\boldsymbol{c}'_1]_{Q_i}\right) \in \mathcal{R}^3_{Q_i}$$

is a degree-2 ciphertext encrypting $[\boldsymbol{m}\cdot\boldsymbol{m}']_t$ and its noise is bounded by

$$\begin{aligned}\|\boldsymbol{v}_{\mathtt{mult}}\|_\infty &= \left\|[\boldsymbol{m}]_t\cdot\boldsymbol{v}'+\boldsymbol{v}\cdot[\boldsymbol{m}']_t+t\boldsymbol{v}\cdot\boldsymbol{v}'+\boldsymbol{r}_m\right\|_\infty \\ &\leq \frac{\delta_{\mathcal{R}} t}{2}\left(2\|\boldsymbol{v}\|_\infty\left\|\boldsymbol{v}'\right\|_\infty + \|\boldsymbol{v}\|_\infty + \left\|\boldsymbol{v}'\right\|_\infty + 1\right).\end{aligned}$$

Remark 2.1 *The reader can notice that the degree, and thus the size, of a ciphertext increases after each multiplication, increasing therefore the future communication and computational costs. Since this is something one wants to avoid in practice, the degree-2 ciphertexts are "relinearized" after a homomorphic multiplication to degree-1 ciphertexts using a key-switching procedure (see the full version).*

The main issue with homomorphic multiplication is its quadratic noise growth, which implies that by choosing $Q_L \approx \|\boldsymbol{v}_{\mathtt{fresh}}\|_\infty^L$ one could only perform $\log_2 L$ consecutive multiplications. The idea of modulus switching is to reduce the size of the noise after each multiplication to keep it constant and prevent the quadratic blow-up. This is achieved by scaling the ciphertext $\mathtt{ct}$ by Q_i/Q_j for $i<j$, which scales down the noise by roughly the same factor. More precisely, let $\mathtt{ct}=(\boldsymbol{c}_0,\boldsymbol{c}_1)$ be a level $j\in(0,L]\cap\mathbb{Z}$ encryption of a message $\boldsymbol{m}$ with noise $\boldsymbol{v}$ and let i be an integer smaller than j, then set:

$$\boldsymbol{\delta} = \left(t[-\boldsymbol{c}_0/t]_{Q_j/Q_i}, t[-\boldsymbol{c}_1/t]_{Q_j/Q_i}\right) \in \mathcal{R}^2.$$

Then one can compute

$$\mathtt{ct}' = \frac{Q_i}{Q_j}\cdot(\boldsymbol{c}_0+\boldsymbol{\delta}_0, \boldsymbol{c}_1+\boldsymbol{\delta}_1) \bmod Q_i.$$

Brakerski et al. showed that if $\mathtt{ct}=(\boldsymbol{c}_0,\boldsymbol{c}_1)$ is such that

$$\left\|[\boldsymbol{c}_0+\boldsymbol{c}_1\cdot\boldsymbol{s}]_{Q_j}\right\|_\infty < \frac{Q_j}{2} - \frac{tQ_j}{2Q_i}(1+\delta_{\mathcal{R}} B_{\mathtt{key}}),$$

then $\mathtt{ct}'$ is an encryption of $[Q_i/Q_j\boldsymbol{m}]_t$ whose noise $\boldsymbol{v}'$ is bounded by

$$\left\|\boldsymbol{v}'\right\|_\infty \leq \frac{Q_i}{Q_j}\|\boldsymbol{v}\|_\infty + \|\boldsymbol{v}_{\mathtt{ms}}\|_\infty$$

with $\|\boldsymbol{v}_{\mathtt{ms}}\|_\infty \leq (1+\delta_{\mathcal{R}} B_{\mathtt{key}})/2$. Therefore by choosing the encryption parameters such that performing modulus switching after a homomorphic multiplication (plus a key-switching) brings the noise back to its initial level, one can perform L consecutive multiplications instead of approximately $\log_2 L$ initially.

Gentry-Halevi-Smart (GHS) variant of BGV. Since the modulus-switching procedure outputs an encryption of the message scaled by a factor $[Q_i/Q_j]_t$, Brakerski et al. proposed to choose the moduli $Q_i = 1 \bmod t$. This approach, although very convenient in theory, becomes challenging in practice when using a large t since it reduces significantly the range of possible moduli for the Q_i. As a consequence, Gentry, Halevi, and Smart proposed to keep track of the scaling factor for each ciphertext instead [22]. In particular, they suggested to encrypt $[Q_L \boldsymbol{m}]_t$ instead of $[\boldsymbol{m}]_t$ in Equation (1), which provides natural downscaling to $[Q_i \boldsymbol{m}]_t$ as modulus switching operations are applied. However in this case, one has to pay attention when adding two ciphertexts with different scaling factors. Nonetheless this can be achieved without impacting significantly the noise by following the methodology of [27].

Gentry et al. also proposed several optimizations related to noise management. The first one is to perform modulus switching after encryption and before first multiplication, in order to reduce the noise from $\|\boldsymbol{v}_{\texttt{fresh}}\|_\infty$ to $\|\boldsymbol{v}_{\texttt{ms}}\|_\infty$. The second one is to perform modulus switching just before the next multiplication instead of just after a multiplication. This permits to reduce the noise accumulated due to other operations, such as additions or key switching, that are performed between two subsequent multiplications.

2.2.2 Original BFV Scheme

In [9], Brakerski proposed a *scale-invariant* construction that achieves asymptotically the same noise growth as `BGV`, but does not *explicitly* call the modulus-switching procedure, embedding it internally in the homomorphic multiplication. Fan and Vercauteren then ported Brakerski's construction to the Ring-LWE setting [18], and the scheme is now commonly referred to as `BFV`. The `BFV` scheme uses a public key

$$\texttt{pk} = \left([\boldsymbol{a} \cdot \boldsymbol{s} + \boldsymbol{e}]_Q \, , -\boldsymbol{a} \right) \in \mathcal{R}_Q^2,$$

which corresponds exactly to a Ring-LWE sample associated to $\boldsymbol{s}$ and Q with $\boldsymbol{a} \leftarrow \mathcal{U}_Q$ and $\boldsymbol{e} \leftarrow \chi_{\texttt{err}}$. The main difference between `BGV` and `BFV` is that `BFV` ciphertexts encode messages in their MSD instead of LSD:

$$\texttt{ct} = \left([\Delta[\boldsymbol{m}]_t + \boldsymbol{u} \cdot \mathbf{pk}_0 + \boldsymbol{e}_0]_Q \, , [\boldsymbol{u} \cdot \mathbf{pk}_1 + \boldsymbol{e}_1]_Q \right)$$

with $\Delta = \lfloor Q/t \rfloor$, $\boldsymbol{u} \leftarrow \chi_{\texttt{key}}$ and $\boldsymbol{e}_0, \boldsymbol{e}_1 \leftarrow \chi_{\texttt{err}}$. Similarly to `BGV`, the noise contained in a ciphertext $\texttt{ct} = (\boldsymbol{c}_0, \boldsymbol{c}_1)$ appears explicitly once the ciphertext is evaluated on the secret key $\boldsymbol{s}$:

$$\boldsymbol{c}_0 + \boldsymbol{c}_1 \cdot \boldsymbol{s} = \Delta[\boldsymbol{m}]_t + \boldsymbol{u} \cdot \boldsymbol{e} + \boldsymbol{e}_1 \cdot \boldsymbol{s} + \boldsymbol{e}_0 = \Delta[\boldsymbol{m}]_t + \boldsymbol{v}_{\texttt{fresh}} \bmod Q,$$

where the "fresh" noise $\boldsymbol{v}_{\texttt{fresh}} = \boldsymbol{u} \cdot \boldsymbol{e} + \boldsymbol{e}_1 \cdot \boldsymbol{s} + \boldsymbol{e}_0$ is the same as for `BGV`.

To decrypt the ciphertext `ct`, one needs to scale and round $\texttt{ct}(\boldsymbol{s})$ by t/Q to remove the factor Δ. Hence the decryption procedure requires computing

$$\boldsymbol{m}' = \left\lfloor \frac{t}{Q} \left[\boldsymbol{c}_0 + \boldsymbol{c}_1 \cdot \boldsymbol{s} \right]_Q \right\rceil,$$

and the decryption will be correct as long as:

$$\|\boldsymbol{v}\|_\infty < \frac{Q}{2t} - \frac{r_t(Q)}{2}. \tag{2}$$

Note that the term $r_t(Q)/2$ is the error inherited from the difference between Δ^{-1} and t/Q: $\Delta t/Q = 1 - r_t(Q)/Q$. Addition of two ciphertexts $\mathtt{ct}$ and $\mathtt{ct}'$ is done like in BGV, but the noise growth is slightly different since the carry of the addition of the two messages is scaled by Δ:

$$\boldsymbol{c}_0 + \boldsymbol{c}'_0 + (\boldsymbol{c}_1 + \boldsymbol{c}'_1) \cdot \boldsymbol{s} = \Delta[\boldsymbol{m} + \boldsymbol{m}']_t + \boldsymbol{v} + \boldsymbol{v}' - r_t(Q)\boldsymbol{u} \bmod Q,$$

with $\|\boldsymbol{u}\|_\infty \leq 1$. This implies that

$$\mathtt{ct}_{\mathtt{add}} = \left([\boldsymbol{c}_0 + \boldsymbol{c}'_0]_Q, [\boldsymbol{c}_1 + \boldsymbol{c}'_1]_Q\right)$$

is an encryption of $[\boldsymbol{m} + \boldsymbol{m}']_t$, and its noise is bounded by

$$\|\boldsymbol{v}_{\mathtt{add}}\|_\infty = \|\boldsymbol{v} + \boldsymbol{v}' + r_t(Q)\boldsymbol{u}\|_\infty \leq \|\boldsymbol{v}\|_\infty + \left\|\boldsymbol{v}'\right\|_\infty + r_t(Q). \tag{3}$$

The BFV multiplication of two ciphertexts $\mathtt{ct}$ and $\mathtt{ct}'$ is done differently, as compared to BGV, since once the product is computed, it gets scaled by Δ^2, which has two important consequences. First, the product of $\mathtt{ct}$ and $\mathtt{ct}'$ cannot be reduced modulo Q, therefore it must be done in $\mathcal{R}$, i.e., without any modular reduction. Second, the product must be scaled down by $t/Q \approx \Delta^{-1}$ to remove the extra Δ factor and reduce the noise similarly to modulus switching in BGV. We describe the two steps of the homomorphic multiplication separately. The first part, called the *tensoring*, consists in computing the product of two ciphertexts in $\mathcal{R}$:

$$\mathtt{ct}_{\mathtt{tensor}} = (\boldsymbol{c}_0 \cdot \boldsymbol{c}'_0, \boldsymbol{c}_0 \cdot \boldsymbol{c}'_1 + \boldsymbol{c}_1 \cdot \boldsymbol{c}'_0, \boldsymbol{c}_1 \cdot \boldsymbol{c}'_1) \in \mathcal{R}^3.$$

When evaluating $\mathtt{ct}_{\mathtt{tensor}}$ on the secret key, one obtains:

$$\begin{aligned}(\boldsymbol{c}_0 \ + \boldsymbol{c}_1 \cdot \boldsymbol{s}) \cdot (\boldsymbol{c}'_0 \ + \boldsymbol{c}'_1 \cdot \boldsymbol{s}) &= (\Delta[\boldsymbol{m}]_t + \boldsymbol{v} + Q\boldsymbol{k}) \cdot (\Delta[\boldsymbol{m}']_t + \boldsymbol{v}' + Q\boldsymbol{k}') \\ &= \frac{Q}{t}\Delta\left[\boldsymbol{m} \cdot \boldsymbol{m}'\right]_t + \frac{Q}{t}\boldsymbol{v}_{\mathtt{tensor}} + \frac{Q^2}{t}\boldsymbol{k}_{\mathtt{tensor}},\end{aligned}$$

where

$$\begin{aligned}\boldsymbol{v}_{\mathtt{tensor}} = &\frac{t\boldsymbol{v} \cdot \boldsymbol{v}'}{Q} + \frac{t\Delta}{Q}\left([\boldsymbol{m}]_t \cdot \boldsymbol{v}' + [\boldsymbol{m}']_t \cdot \boldsymbol{v}\right) + t(\boldsymbol{v} \cdot \boldsymbol{k}' + \boldsymbol{v}' \cdot \boldsymbol{k}) \\ &- r_t(Q)\left([\boldsymbol{m}]_t \cdot \boldsymbol{k}' + [\boldsymbol{m}']_t \cdot \boldsymbol{k} + \boldsymbol{r}_m + \frac{\Delta}{Q}[\boldsymbol{m}]_t \cdot [\boldsymbol{m}']_t\right),\end{aligned}$$

$$\boldsymbol{k}_{\mathtt{tensor}} = [\boldsymbol{m}]_t \cdot \boldsymbol{k}' + [\boldsymbol{m}']_t \cdot \boldsymbol{k} + t\boldsymbol{k} \cdot \boldsymbol{k}' + \boldsymbol{r}_m$$

with $\|\boldsymbol{r}_m\|_\infty \leq \delta_{\mathcal{R}} t/2$ like for BGV. Also note that $\boldsymbol{k} = (\boldsymbol{c}_0 + \boldsymbol{c}_1 \cdot \boldsymbol{s} - \Delta[\boldsymbol{m}]_t - \boldsymbol{v})/Q$ and $\boldsymbol{k}' = (\boldsymbol{c}'_0 + \boldsymbol{c}'_1 \cdot \boldsymbol{s} - \Delta[\boldsymbol{m}']_t - \boldsymbol{v}')/Q$ have their norm bounded by $(\delta_{\mathcal{R}} B_{\mathtt{key}} + 3)/2$.

The scaling operation is done in $\mathcal{R}_Q$ and outputs a result modulo Q:

$$\mathtt{ct}_{\mathtt{scale}} = \left\lfloor\left\lceil \frac{t}{Q}\mathtt{ct}_{\mathtt{tensor}}\right\rceil\right\rfloor_Q \in \mathcal{R}_Q^3.$$

The scaling leads to

$$\frac{t}{Q}\left(\boldsymbol{c}_{\mathtt{tensor}_0} + \boldsymbol{c}_{\mathtt{tensor}_1} \cdot \boldsymbol{s} + \boldsymbol{c}_{\mathtt{tensor}_2} \cdot \boldsymbol{s}^2\right) = \Delta\left[\boldsymbol{m} \cdot \boldsymbol{m}'\right] + \boldsymbol{v}_{\mathtt{tensor}} + Q\boldsymbol{k}_{\mathtt{tensor}}.$$

The rounding of scaled terms introduces an additional error $\boldsymbol{v}_r$ such that

$$\mathtt{ct}_{\mathtt{scale}_0} + \mathtt{ct}_{\mathtt{scale}_1} \cdot \boldsymbol{s} + \mathtt{ct}_{\mathtt{scale}_2} \cdot \boldsymbol{s}^2 = \Delta \left[\boldsymbol{m} \cdot \boldsymbol{m}'\right] + \boldsymbol{v}_{\mathtt{tensor}} + \boldsymbol{v}_{\mathtt{r}} \bmod Q,$$

with $\|\boldsymbol{v}_r\|_\infty \leq (1 + \delta_{\mathcal{R}} B_{\mathtt{key}} + \delta_{\mathcal{R}}^2 B_{\mathtt{key}}^2)/2$. Hence the total multiplication noise $\boldsymbol{v}_{\mathtt{mult}} = \boldsymbol{v}_{\mathtt{tensor}} + \boldsymbol{v}_{\mathtt{r}}$ is bounded by

$$\begin{aligned}\left\|\boldsymbol{v}_{\mathtt{mult}}\right\|_\infty \leq & \frac{\delta_{\mathcal{R}} t}{2} \left(\frac{2 \left\|\boldsymbol{v}\right\|_\infty \left\|\boldsymbol{v}'\right\|_\infty}{Q} + \left(4 + \delta_{\mathcal{R}} B_{\mathtt{key}}\right) \left(\left\|\boldsymbol{v}\right\|_\infty + \left\|\boldsymbol{v}'\right\|_\infty\right) \right. \\ & \left. + r_t(Q) \left(\delta_{\mathcal{R}} B_{\mathtt{key}} + 5\right) \right) + \frac{1 + \delta_{\mathcal{R}} B_{\mathtt{key}} + \delta_{\mathcal{R}}^2 B_{\mathtt{key}}^2}{2}. \end{aligned} \tag{4}$$

For the same reasons as for BGV, one needs to perform a key-switching operation to relinearize the resulting degree-2 ciphertext. The key-switching procedure is the same for both schemes, and we refer to the full version for further details.

2.3 RNS Representation

The Chinese Remainder Theorem (CRT) permits decomposing multi-precision integers in $\mathbb{Z}_Q$ into vectors of smaller integers to perform operations efficiently using native (64-bit) integer data types. The integer CRT representation is also often referred to as the Residue-Number-System (RNS) representation. As a consequence, the ciphertext modulus is usually chosen as a product of "small", i.e., fitting in a machine word, co-prime moduli so that elements of $\mathcal{R}_Q$ are represented with their residues modulo the different q_i's. For BGV, we choose $Q = q_0 \cdots q_L$ and we denote $Q_i = q_0 \cdots q_i$ for $0 \leq i \leq L$, where each $q_i = 1 \bmod 2N$, to use the efficient NTT algorithm for the multiplication of elements in $\mathcal{R}_Q$. For BFV, we choose $Q = q_1 \cdots q_k$, with $q_i = 1 \bmod 2N$ for $1 \leq i \leq k$ and similarly to BGV we denote $Q_i = q_1 \cdots q_i$ for $1 \leq i \leq k$. Note that we have chosen different notations on purpose since in the original BGV the size of the moduli is directly related to the noise reduction we want to achieve by modulus switching, and is therefore dependent on the circuit one wants to evaluate. However, this constraint can be removed by considering a granular approach with dynamic noise estimation, as implemented in HElib [25] (see the discussion in Section 4 for more details on dynamic vs static noise estimation in BGV). On the other hand, in BFV the size of the moduli is independent of the circuit and, hence, the moduli are usually chosen as large as possible within the limit of a machine word.

When performing computations in RNS, and more particularly when implementing BGV and BFV, it is sometimes needed to switch the RNS basis, i.e., convert $\boldsymbol{a} \in \mathcal{R}_Q$ from its residues modulo $Q = q_1 \cdots q_k$ to $[\boldsymbol{a}]_Q$ modulo P for some $P = p_1 \cdots p_{k'}$. This can be achieved using a *basis extension* formulated as

$$\mathtt{FastBaseExtension}(\boldsymbol{a}, Q, P) = \sum_{i=1}^{k} \left[\boldsymbol{a} \left(\frac{Q}{q_i}\right)^{-1} \right]_{q_i} \frac{Q}{q_i} \bmod p_j. \tag{5}$$

Note that the basis extension does not yield $[\boldsymbol{a}]_Q \bmod P$ but rather $[\boldsymbol{a}]_Q + \boldsymbol{u}Q \bmod P$ with $\|\boldsymbol{u}\|_\infty < k/2$. When the result of this extension is divided by Q, as in many procedures of BGV and BFV, the error caused by this Q-overflow $\boldsymbol{u}$ can be neglected most of the times. However in certain cases, as in the BFV decryption procedure, this overflow cannot be tolerated and needs to be removed/corrected. This can be achieved either using integer instructions with the so-called γ-*correction* technique of

[6], or using floating-point instructions to retrieve $\boldsymbol{u}$ as in [23] since

$$\boldsymbol{u} = \left\lfloor \sum_{i=1}^{k} \left[\boldsymbol{a} \left(\frac{Q}{q_i} \right)^{-1} \right]_{q_i} \frac{1}{q_i} \right\rceil . \tag{6}$$

The same problem occurs during BFV homomorphic multiplication, and if it is not handled using either of the techniques of [6] or [23], the impact of $\boldsymbol{u}$ on the noise growth will be significant [7].

2.4 Hybrid Key Switching in RNS

Key switching transforms a ciphertext $\mathtt{ct} = (\boldsymbol{c}_0, \boldsymbol{c}_1) \in \mathcal{R}_Q^2$, which can be decrypted with $\boldsymbol{s}_A$, into a ciphertext $\mathtt{ct}' = (\boldsymbol{c}'_0, \boldsymbol{c}'_1) \in \mathcal{R}_Q^2$ encrypting the same message as $\mathtt{ct}$, but decryptable with another secret key $\boldsymbol{s}_B$. This procedure is needed to compute automorphisms (rotations) of the ciphertexts [21], or to relinearize ciphertexts after a homomorphic multiplication. Note that this procedure adds a noise $\boldsymbol{v}_{\mathtt{ks}}$ to the ciphertexts.

Several ways of performing the key-switching procedure have been found over the years. The first one was formulated by Brakerski and Vaikuntanathan (BV) in [11] and extended to RNS in [6]. This technique is based on digit decomposition of one ring element in the ciphertext. Unfortunately the BV key switching requires a quadratic number of NTTs to be computed, and hence becomes the main bottleneck of the scheme (asymptotically, and often in practice), and causes a relatively large noise growth. In [22], Gentry, Halevi, and Smart proposed another alternative for key switching. Their method, which we refer to as the GHS key switching, has a smaller noise growth than BV, and is also more efficient (asymptotically, and in many practical cases) since it only requires a linear number of NTTs. The drawback of this method is that one either needs to double the dimension N or reduce the size of Q by a factor of 2 for security reasons. Gentry, Halevi, and Smart also presented a hybrid key switching technique, which combines BV digit decomposition and larger modulus from GHS to provide the best tradeoff between the two techniques. The RNS versions of hybrid key switching were later derived for the CKKS scheme in [28] (for one small special prime) and in [26] for the more general case. The hybrid key switching technique [26] is the most efficient one in practice, both in terms of performance and noise growth, as our detailed comparison of the BV, GHS, and Hybrid key switching in the full version shows. Hence we use the hybrid key switching in our implementation.

3 Improved BFV Scheme

One can notice from Equations (2), (3) and (4) that the noise of BFV is impacted by the $r_t(Q)$ factor which does not appear in BGV. This factor causes faster noise growth for BFV when using larger plaintext moduli, as compared to BGV. In this section we show that this problem is not inherent to the MSD encoding of BFV, but rather comes from the choice for its instantiation in [18] and prior LWE-based Brakerski scheme [9]. We show that by instantiating the scheme in a more natural way, we can get rid of this $r_t(Q)$ term. We also present a modified homomorphic multiplication procedure that significantly improves the complexity of BFV homomorphic multiplication, as compared to all prior variants of BFV.

In this section, $\mathtt{ct} = (\boldsymbol{c}_0, \boldsymbol{c}_1)$ and $\mathtt{ct}' = (\boldsymbol{c}'_0, \boldsymbol{c}'_1)$ denote two BFV ciphertexts encrypting, respectively, the messages $[\boldsymbol{m}]_t$ and $[\boldsymbol{m}']_t$ with noise $\boldsymbol{v}$ and $\boldsymbol{v}'$.

3.1 Noise Reduction

To fully understand the problem with faster noise growth in BFV for larger plaintext moduli, let us examine more carefully the noise bound after a multiplication in BFV (Equation (4)). This bound can be simplified by analyzing only the dominant terms, which determine the noise magnitude. More precisely, if we assume that the two ciphertexts have their noise bounded by V, and $B_{\texttt{key}} = 1$, the size of the noise of their multiplication can be reasonably approximated by

$$\delta_{\mathcal{R}} t \left((5 + \delta_{\mathcal{R}}) V + \frac{r_t(Q)}{2} (\delta_{\mathcal{R}} + 5) \right) + \frac{\delta_{\mathcal{R}}^2}{2} \approx \delta_{\mathcal{R}}^2 t \left(V + \frac{r_t(Q)}{2} \right). \tag{7}$$

Since the noise grows significantly with homomorphic multiplications, V becomes larger than $r_t(Q)/2 < t$ after we perform the first multiplication. However, this is not necessarily true for the first multiplication itself since, like in BGV, the noise of fresh ciphertexts in BFV is bounded by $B_{\texttt{err}}(2\delta_{\mathcal{R}} B_{\texttt{key}} + 1) \approx 2\delta_{\mathcal{R}} B_{\texttt{err}}$. The homomorphic encryption standard ([4]) recommends using an error distribution with $\sigma_{\texttt{err}} = 3.2$. Therefore, the noise of a fresh ciphertext can be estimated as $V_{\texttt{init}} = 2 \times 6 \times 3.2 \times 2\sqrt{N} < 77\sqrt{N}$. Since in practice the dimension N typically does not exceed 2^{16}, a fresh ciphertext always has its noise size not higher than 14 bits, while $r_t(Q)$ can be as large as t. As a consequence, when $r_t(Q)$ is larger than 2^{15}, it becomes responsible for the larger noise growth after the first multiplication in BFV. For instance, if $t = 2^{32}$ and $r_t(Q) \approx t/2 \approx 2^{31}$, the noise after the first multiplication will be at least 16 bits larger than in the case when $r_t(Q) < V_{\texttt{init}}$. Note that this difference will not lead to a larger noise growth on the next multiplications since, as shown in Equation (7), the noise growth after a multiplication is linear in V. However, this difference of 16+ bits will be carried through until the end of the computation. In the case of $t = 2^{60}$, this difference would become at least 44 bits.

The easiest way to circumvent this problem would be to choose the moduli q_i, as in the original BGV, i.e., such that $q_i = 1 \bmod t$, which would lead to $r_t(Q) = 1$. However, for the same reasons as in BGV, this restriction would make the finding of the moduli challenging for large t. Although it is possible to relax this condition by choosing, for instance, $r_t(Q) < \sqrt{N}$, i.e., finding $r_t(Q)$ through trial and error, we believe this would be a patch rather than a real solution. We show next that there is a more natural way to fix this problem.

Indeed, the $r_t(Q)$ term comes from the difference between Δ^{-1} and t/Q since when computing $\Delta t/Q$ (during decryption and homomorphic multiplication), one obtains $1 - r_t(Q)/Q$. Therefore, to solve this issue we propose to modify the original BFV encryption algorithm by encoding $[\boldsymbol{m}]_t$ in the ciphertext in a more natural way as $\lfloor Q[\boldsymbol{m}]_t/t \rceil$ instead of $\Delta[\boldsymbol{m}]_t$. The first benefit is seen in the decryption bound since now

$$\begin{aligned} \left\lfloor \frac{t}{Q}[\boldsymbol{c}_0 + \boldsymbol{c}_1 \cdot \boldsymbol{s}]_Q \right\rceil &= \left\lfloor \frac{t}{Q} \left(\frac{Q}{t}[\boldsymbol{m}]_t + \boldsymbol{v} + \boldsymbol{\varepsilon} + \boldsymbol{k}Q \right) \right\rceil = [\boldsymbol{m}]_t + \left\lfloor \frac{t}{Q} (\boldsymbol{v} + \boldsymbol{\varepsilon}) \right\rceil + t\boldsymbol{k} \\ &= [\boldsymbol{m}]_t + \left\lfloor \frac{t}{Q} (\boldsymbol{v} + \boldsymbol{\varepsilon}) \right\rceil \bmod t, \end{aligned}$$

where $\boldsymbol{k} \in \mathcal{R}$ and $\boldsymbol{\varepsilon}$ is the rounding error coming from $\lfloor Q[\boldsymbol{m}]_t/t \rceil = Q[\boldsymbol{m}]_t/t + \boldsymbol{\varepsilon}$, such that $\|\boldsymbol{\varepsilon}\|_\infty \leq 1/2$. Therefore the decryption will be correct as long as:

$$\frac{t}{Q} \|\boldsymbol{v} + \boldsymbol{\varepsilon}\|_\infty < \frac{1}{2},$$

which is satisfied if

$$\|\boldsymbol{v}\|_\infty < \frac{Q}{2t} - \frac{1}{2}. \tag{8}$$

Remark 3.1 *Note that one can compute* $\lfloor Q[\boldsymbol{m}]_t/t \rceil \bmod Q$ *directly in RNS as long as* t *and* Q *are coprime since*

$$\left\lfloor \frac{Q[\boldsymbol{m}]_t}{t} \right\rceil = \frac{Q[\boldsymbol{m}]_t - [Q\boldsymbol{m}]_t}{t} = -\frac{[Q\boldsymbol{m}]_t}{t} \bmod Q.$$

The second benefit is observed in the addition since now we have

$$\begin{aligned}
\boldsymbol{c}_0 + \boldsymbol{c}_1 \cdot \boldsymbol{s} + \boldsymbol{c}'_0 + \boldsymbol{c}'_1 \cdot \boldsymbol{s} &= \frac{Q}{t}\left([\boldsymbol{m}]_t + [\boldsymbol{m}']_t\right) + \boldsymbol{v} + \boldsymbol{v}' + \boldsymbol{\varepsilon} + \boldsymbol{\varepsilon}' \\
&= \frac{Q}{t}\left([\boldsymbol{m} + \boldsymbol{m}']_t + t\boldsymbol{u}\right) + \boldsymbol{v} + \boldsymbol{v}' + \boldsymbol{\varepsilon} + \boldsymbol{\varepsilon}' \\
&= \frac{Q}{t}[\boldsymbol{m} + \boldsymbol{m}']_t + \boldsymbol{v} + \boldsymbol{v}' + \boldsymbol{\varepsilon} + \boldsymbol{\varepsilon}' \bmod Q.
\end{aligned}$$

Hence the noise after a homomorphic addition is bounded by

$$\|\boldsymbol{v}_{\texttt{new-add}}\|_\infty \leq \|\boldsymbol{v}\|_\infty + \left\|\boldsymbol{v}'\right\|_\infty + 1. \tag{9}$$

Note that the decryption bound (8) and the addition bound (9) are now exactly the same as for `BGV`.

The equations for homomorphic multiplication can be simplified the same way. Denoting $\tilde{\boldsymbol{v}} = \boldsymbol{v} + \boldsymbol{\varepsilon}$, $\texttt{ct}_{\texttt{tensor}}$ is computed as

$$\begin{aligned}
(\boldsymbol{c}_0 + \boldsymbol{c}_1 \cdot \boldsymbol{s}) \cdot (\boldsymbol{c}'_0 + \boldsymbol{c}'_1 \cdot \boldsymbol{s}) &= \left(\frac{Q}{t}[\boldsymbol{m}]_t + \tilde{\boldsymbol{v}} + \boldsymbol{k}Q\right) \cdot \left(\frac{Q}{t}[\boldsymbol{m}']_t + \tilde{\boldsymbol{v}}' + \boldsymbol{k}'Q\right) \\
&= \frac{Q^2}{t^2}[\boldsymbol{m} \cdot \boldsymbol{m}']_t + \frac{Q}{t}\boldsymbol{v}_{\texttt{new-tensor}} + \frac{Q^2}{t}\boldsymbol{k}_{\texttt{new-tensor}},
\end{aligned}$$

where

$$\begin{aligned}
\boldsymbol{v}_{\texttt{new-tensor}} &= [\boldsymbol{m}]_t \cdot \tilde{\boldsymbol{v}}' + [\boldsymbol{m}']_t \cdot \tilde{\boldsymbol{v}} + \frac{t}{Q}\tilde{\boldsymbol{v}} \cdot \tilde{\boldsymbol{v}}' + t(\tilde{\boldsymbol{v}} \cdot \boldsymbol{k}' + \tilde{\boldsymbol{v}}' \cdot \boldsymbol{k}), \\
\boldsymbol{k}_{\texttt{new-tensor}} &= [\boldsymbol{m}]_t \cdot \boldsymbol{k}' + [\boldsymbol{m}']_t \cdot \boldsymbol{k} + t\boldsymbol{k} \cdot \boldsymbol{k}' + \boldsymbol{r}_m.
\end{aligned}$$

Then after the scaling by t/Q and rounding, similarly to the original `BFV` scheme, the noise of the multiplication is given by: $\boldsymbol{v}_{\texttt{new-mult}} = \boldsymbol{v}_{\texttt{new-tensor}} + \boldsymbol{v}_r$, and is bounded by:

$$\begin{aligned}
\|\boldsymbol{v}_{\texttt{new-mult}}\|_\infty \leq \frac{\delta_{\mathcal{R}} t}{2}\left(\frac{2\|\tilde{\boldsymbol{v}}\|_\infty \|\tilde{\boldsymbol{v}}'\|_\infty}{Q} + \left(4 + \delta_{\mathcal{R}} B_{\texttt{key}}\right)\left(\|\tilde{\boldsymbol{v}}\|_\infty + \left\|\tilde{\boldsymbol{v}}'\right\|_\infty\right)\right) \\
+ \frac{1 + \delta_{\mathcal{R}} B_{\texttt{key}} + \delta_{\mathcal{R}}^2 B_{\texttt{key}}^2}{2}.
\end{aligned} \tag{10}$$

We will see in Section 5 that this bound is similar to the bound for `BGV`.

Similarly to [29], we can derive a bound on the noise after having evaluated a binary tree of depth L from (10). By assuming that the size of the noise of ciphertexts is bounded by V before the first multiplication, the noise of the resulting ciphertext will be bounded by

$$C_1^L V + C_2 \sum_{i=0}^{L-1} C_1^i \leq C_1^L V + L C_2 C_1^{L-1}$$

with $C_1 = \delta_{\mathcal{R}} t(5 + \delta_{\mathcal{R}} B_{\mathtt{key}})$ and $C_2 = (1 + \delta_{\mathcal{R}} B_{\mathtt{key}} + \delta_{\mathcal{R}}^2 B_{\mathtt{key}}^2)/2 + V_{\mathtt{ks}}$ where $V_{\mathtt{ks}}$ represents the noise added by the key-switching (see the full version).

Last we want to highlight further the similarity between BGV and BFV. Just like in the GHS variant of BGV, one can encrypt a ciphertext ct in BFV using a slightly larger modulus Qp and then rescale the ciphertext by $1/p$, which will have the same effect as modulus switching in BGV:

$$\mathtt{ct}_{\mathtt{scale}} = \left\lfloor \frac{1}{p} \mathtt{ct} \right\rceil \bmod Q,$$

so that its noise after scaling is bounded by

$$\|\boldsymbol{v}_{\mathtt{scale}}\|_\infty \leq \frac{\|\boldsymbol{v}\|_\infty}{p} + \frac{1}{2p} + \frac{1 + \delta_{\mathcal{R}} B_{\mathtt{key}}}{2}.$$

Therefore, like in BGV, this allows to reduce the noise of a freshly encrypted to $(1 + \delta_{\mathcal{R}} B_{\mathtt{key}})/2 \approx \delta_{\mathcal{R}}/2$. Note that the noise benefit of the BFV encryption proposed in our work will become more significant as the fresh noise is several bits larger than the modulus switching noise. Moreover, when using GHS or Hybrid key switching, p can be chosen as one of the moduli of the key-switching basis, and therefore this technique will not impact the selection of Ring-LWE security parameters.

3.2 Modified Homomorphic Multiplication

In the previous subsection, we showed how to instantiate BFV in such a way that it is not worse than BGV in terms of noise growth. Now the main difference left between the two schemes is in the complexity of their homomorphic multiplication procedure. In a nutshell, in BGV the tensoring can be done directly modulo Q while in BFV it must be done without any modular reduction. In practice, this requires using a second RNS basis $P = p_1 \cdots p_{k'}$ such that $\mathtt{ct}(\boldsymbol{s}) \cdot \mathtt{ct}'(\boldsymbol{s}')$ does not wrap around modulo PQ. More precisely, the critical part in practice is to avoid the wraparound of the dominant term $Qt\boldsymbol{k} \cdot \boldsymbol{k}'$ modulo P during the scaling (see Section 2.2.2). This requires to choose $P > t\delta_{\mathcal{R}}^3 Q/4$, which in practice is satisfied by setting $k' = k + 1$ for the RNS instantiation. Algorithm 1 recalls the original homomorphic multiplication procedure of BFV.

We propose a new homomorphic multiplication algorithm, with a reduced computational complexity. Instead of multiplying two ciphertexts modulo Q and dealing with a multiple of Q^2 modulo QP, the idea is to switch one of the two ciphertexts to modulus P so that after the tensoring we obtain a multiple of PQ that vanishes modulo PQ. As explained in the above paragraph, this would allow us to reduce the size of P since the original dominant term would now disappear. More precisely, the procedure starts as usual with two ciphertexts encrypted modulo Q:

$$\mathtt{ct}(\boldsymbol{s}) = \frac{Q}{t}[\boldsymbol{m}]_t + \tilde{\boldsymbol{v}} + \boldsymbol{k}Q \text{ and } \mathtt{ct}'(\boldsymbol{s}) = \frac{Q}{t}[\boldsymbol{m}']_t + \tilde{\boldsymbol{v}}' + \boldsymbol{k}'Q.$$

with $\tilde{\boldsymbol{v}} = \boldsymbol{v} + \boldsymbol{\varepsilon}$ and $\tilde{\boldsymbol{v}}' = \boldsymbol{v}' + \boldsymbol{\varepsilon}'$, like in Section 3.1.

Then one can perform the modulus switching of one of the two ciphertexts, say ct, to convert it to modulus P by computing

$$\hat{\mathtt{ct}} = \left\lfloor \frac{P}{Q} \mathtt{ct} \right\rceil \bmod P,$$

Algorithm 1 Original BFV Multiplication Algorithm

procedure OriginalMult($\mathtt{ct} = (\boldsymbol{c}_0, \boldsymbol{c}_1) \in \mathcal{R}_Q^2, \mathtt{ct}' = (\boldsymbol{c}_0', \boldsymbol{c}_1') \in \mathcal{R}_Q^2$)

Expand: $\mathtt{ct} \in \mathcal{R}_Q^2$ and $\mathtt{ct}' \in \mathcal{R}_Q^2 \to \mathtt{ct} \in \mathcal{R}_{QP}^2$ and $\mathtt{ct}' \in \mathcal{R}_{QP}^2$:

▷ $\mathtt{ct}(\boldsymbol{s}) = \Delta[\boldsymbol{m}]_t + \boldsymbol{v} + Q\boldsymbol{k} \pmod{\mathcal{R}_{QP}}$

▷ $\mathtt{ct}'(\boldsymbol{s}) = \Delta[\boldsymbol{m}']_t + \boldsymbol{v}' + Q\boldsymbol{k}' \pmod{\mathcal{R}_{QP}}$

Tensor: $\mathtt{ct}_{\mathtt{tensor}} = (\boldsymbol{c}_0 \cdot \boldsymbol{c}_0', \boldsymbol{c}_0 \cdot \boldsymbol{c}_1' + \boldsymbol{c}_1 \cdot \boldsymbol{c}_0', \boldsymbol{c}_1 \cdot \boldsymbol{c}_1') \in \mathcal{R}_{QP}^3$:

▷ $\mathtt{ct}_{\mathtt{tensor}}(\boldsymbol{s}) = \frac{Q}{t}\Delta\left[\boldsymbol{m} \cdot \boldsymbol{m}'\right]_t + \frac{Q}{t}\boldsymbol{v}_{\mathtt{tensor}} + \frac{Q^2}{t}\boldsymbol{k}_{\mathtt{tensor}} \pmod{\mathcal{R}_{QP}}$

ScaleDown: $\mathtt{ct}_{\mathtt{scale}} = \left\lfloor \frac{t}{Q}\mathtt{ct}_{\mathtt{tensor}} \right\rceil \in \mathcal{R}_P^3$

▷ $\mathtt{ct}_{\mathtt{scale}}(\boldsymbol{s}) = \Delta\left[\boldsymbol{m} \cdot \boldsymbol{m}'\right]_t + \boldsymbol{v}_{\mathtt{tensor}} + Q\boldsymbol{k}_{\mathtt{tensor}} + \boldsymbol{v}_r \pmod{\mathcal{R}_P}$

SwitchBasis: $\mathtt{ct}_{\mathtt{scale}} \in \mathcal{R}_P^3 \to \mathtt{ct}_{\mathtt{scale}} \in \mathcal{R}_Q^3$:

▷ $\mathtt{ct}_{\mathtt{scale}}(\boldsymbol{s}) = \Delta\left[\boldsymbol{m} \cdot \boldsymbol{m}'\right]_t + \boldsymbol{v}_{\mathtt{tensor}} + \boldsymbol{v}_r \pmod{\mathcal{R}_Q}$

which satisfies:

$$\hat{\mathtt{ct}}(\boldsymbol{s}) = \frac{P}{t}[\boldsymbol{m}]_t + \frac{P\tilde{\boldsymbol{v}}}{Q} + \boldsymbol{k}P + \boldsymbol{\varepsilon}_{\mathtt{round}} \bmod P,$$

where the rounding error $\boldsymbol{\varepsilon}_{\mathtt{round}} = \hat{\mathtt{ct}}(\boldsymbol{s}) - P\mathtt{ct}(\boldsymbol{s})/Q$ has its norm bounded by $(1 + \delta_{\mathcal{R}}B_{\mathtt{key}})/2$ as usual. From there, $\hat{\mathtt{ct}}$ is expanded from P to QP, $\mathtt{ct}'$ is expanded from Q to QP, and one can perform the tensoring as usual to obtain

$$\begin{aligned}\hat{\mathtt{ct}}_{\mathtt{tensor}}(\boldsymbol{s}) = \ & \frac{PQ}{t^2}[\boldsymbol{m} \cdot \boldsymbol{m}']_t + \frac{P}{t}\hat{\boldsymbol{v}}_{\mathtt{tensor}} + \frac{QP}{t}\hat{\boldsymbol{k}}_{\mathtt{tensor}} \\ & + \boldsymbol{\varepsilon}_{\mathtt{round}} \cdot \left(\frac{Q}{t}[\boldsymbol{m}']_t + \tilde{\boldsymbol{v}}' + \boldsymbol{k}'Q\right) \bmod PQ,\end{aligned}$$

with $\hat{\boldsymbol{v}}_{\mathtt{tensor}} = \boldsymbol{v}_{\mathtt{new\text{-}tensor}}$ from Section 3.1 and

$$\hat{\boldsymbol{k}}_{\mathtt{tensor}} = [\boldsymbol{m}]_t \cdot \boldsymbol{k}' + [\boldsymbol{m}']_t \cdot \boldsymbol{k} + \boldsymbol{r}_m \in \mathcal{R}.$$

Note that this time $\hat{\boldsymbol{k}}_{\mathtt{tensor}}$ does not contain a multiple of $\boldsymbol{k} \cdot \boldsymbol{k}'$. Then to get back a valid ciphertext modulo Q, one must scale down the result by t/P, instead of t/Q in the original case, which leads to

$$\frac{t}{P}\hat{\mathtt{ct}}_{\mathtt{tensor}}(\boldsymbol{s}) = \frac{Q}{t}[\boldsymbol{m} \cdot \boldsymbol{m}']_t + \hat{\boldsymbol{v}}_{\mathtt{tensor}} + Q\hat{\boldsymbol{k}}_{\mathtt{tensor}} + \frac{t\boldsymbol{\varepsilon}_{\mathtt{round}}}{P} \cdot \left(\frac{Q}{t}[\boldsymbol{m}']_t + \tilde{\boldsymbol{v}}' + \boldsymbol{k}'Q\right).$$

After the rounding, the multiple of Q will vanish modulo Q and one will have to take into account the rounding error term $\boldsymbol{v}_r$ of norm bounded by $(1 + \delta_{\mathcal{R}}B_{\mathtt{key}} + \delta_{\mathcal{R}}^2B_{\mathtt{key}}^2)/2$, which adds to the noise, like in the original BFV scheme. Therefore, the noise of this variant of homomorphic multiplication is bounded by

$$\|\hat{\boldsymbol{v}}_{\mathtt{new\text{-}mult}}\|_\infty \leq \|\boldsymbol{v}_{\mathtt{new\text{-}mult}}\|_\infty + \frac{t\delta_{\mathcal{R}}(\delta_{\mathcal{R}}B_{\mathtt{key}} + 1)}{2P}\left(\|\tilde{\boldsymbol{v}}\|_\infty' + \frac{Q(\delta_{\mathcal{R}}B_{\mathtt{key}} + 4)}{2}\right). \tag{11}$$

Notice that the only difference in the noise growth between Equation (10) and Equation (11) is due to rounding error $\boldsymbol{\varepsilon}_{\mathtt{round}}$ occuring during the first modulus switching. However, we can control the size of this noise with P. As explained in Section 3.1, the norm of $\boldsymbol{v}_{\mathtt{new\text{-}mult}}$ is dominated by $\delta_{\mathcal{R}}^2 tV$, where V is the bound on the size of the noise of $\tilde{\boldsymbol{v}}$ and $\tilde{\boldsymbol{v}}'$. If we look carefully at the other

term and choose $P \approx Q$, it will be dominated by $\delta_{\mathcal{R}}^3 t/4$. Since the noise of a fresh ciphertext, even scaled-down after encryption, has always a size larger than $\delta_{\mathcal{R}}/2$, this error term will add at most half a bit to the noise of the first multiplication in the worst case, which can be considered as neglible in practice as a larger cushion is typically added to the heuristic expression for δ_R, or, more precisely, δ_R^3 in this case. This means that one can choose $P \approx Q$ and hence $k' = k$, instead of $k' = k + 1$ in the original case, which reduces the size of P and still achieves the same noise growth as in Section 3.1. We summarize our new multiplication algorithm in Algorithm 2.

Algorithm 2 New BFV Multiplication Algorithm

procedure NEWMULT($\mathtt{ct} = (\boldsymbol{c}_0, \boldsymbol{c}_1) \in \mathcal{R}_Q^2, \mathtt{ct}' = (\boldsymbol{c}_0', \boldsymbol{c}_1') \in \mathcal{R}_Q^2$)

$\mathtt{ModSwitch}$: $\mathtt{ct} \in \mathcal{R}_Q^2 \to \hat{\mathtt{ct}} \in \mathcal{R}_P^2$

$\mathtt{Expand}$: $\hat{\mathtt{ct}} \in \mathcal{R}_P^2 \to \hat{\mathtt{ct}} \in \mathcal{R}_{QP}^2$ and $\mathtt{ct}' \in \mathcal{R}_Q^2 \to \mathtt{ct}' \in \mathcal{R}_{QP}^2$

$\triangleright$ $\hat{\mathtt{ct}}(\boldsymbol{s}) = \frac{P}{t}[\boldsymbol{m}]_t + \frac{P}{Q}\tilde{\boldsymbol{v}} + P\boldsymbol{k} + \boldsymbol{\varepsilon}_{\mathtt{round}} \pmod{\mathcal{R}_{QP}}$

$\triangleright$ $\mathtt{ct}'(\boldsymbol{s}) = \frac{Q}{t}[\boldsymbol{m}']_t + \tilde{\boldsymbol{v}}' + Q\boldsymbol{k}' \pmod{\mathcal{R}_{QP}}$

$\mathtt{Tensor}$: $\hat{\mathtt{ct}}_{\mathtt{tensor}} = (\hat{\boldsymbol{c}}_0 \cdot \boldsymbol{c}_0', \hat{\boldsymbol{c}}_0 \cdot \boldsymbol{c}_1' + \hat{\boldsymbol{c}}_1 \cdot \boldsymbol{c}_0', \hat{\boldsymbol{c}}_1 \cdot \boldsymbol{c}_1') \in \mathcal{R}_{QP}^3$:

$\triangleright$ $\hat{\mathtt{ct}}_{\mathtt{tensor}}(\boldsymbol{s}) = \frac{QP}{t^2}[\boldsymbol{m} \cdot \boldsymbol{m}']_t + \frac{P}{t}\hat{\boldsymbol{v}}_{\mathtt{tensor}} + \frac{QP}{t}\hat{\boldsymbol{k}}_{\mathtt{tensor}} \pmod{\mathcal{R}_{QP}}$

$\mathtt{ScaleDown}$: $\mathtt{ct}_{\mathtt{scale}} = \lfloor \frac{t}{P}\hat{\mathtt{ct}}_{\mathtt{tensor}} \rceil \in \mathcal{R}_Q^3$

$\triangleright$ $\mathtt{ct}_{\mathtt{scale}}(\boldsymbol{s}) = \frac{Q}{t}[\boldsymbol{m} \cdot \boldsymbol{m}']_t + \hat{\boldsymbol{v}}_{\mathtt{tensor}} + \boldsymbol{v}_r \pmod{\mathcal{R}_Q}$

Remark 3.2 *Notice that since one must scale the tensored ciphertext by t/P instead of t/Q, it can be done directly in the basis Q. Therefore the homomorphic multiplication procedure requires 2 basis extensions from Q to P for $\mathtt{ct} = (\boldsymbol{c}_0, \boldsymbol{c}_1)$ in the beginning, 4 more basis extensions to expand the two ciphertexts $\hat{\mathtt{ct}} = (\hat{\boldsymbol{c}}_0, \hat{\boldsymbol{c}}_1)$ and $\mathtt{ct}' = (\boldsymbol{c}_0', \boldsymbol{c}_1')$ from P and Q, respectively, to PQ. Finally one needs 3 additional basis extensions from PQ to Q for $\hat{\mathtt{ct}}_{tensor} \in \mathcal{R}_{PQ}^3$ to perform the downscaling modulo Q. Thus it requires a total of 9 basis extensions instead of $4+3+3 = 10$ in the original BFV algorithm. Moreover, since P is slightly smaller, each basis extension will be cheaper to compute.*

Leveled BFV multiplication. If one wants to make BFV even closer to BGV in terms of performance, one could consider a "leveled" approach to BFV by working with ciphertexts modulo $Q_\ell = q_1 \cdots q_\ell$ and performing modulus switching as the computation progresses. However, as in BGV, one would have to manage ciphertexts at different levels and deal with more challenging noise estimation.

To keep the usability of BFV, we propose instead a "leveled" multiplication that pre-scales both ciphertexts by $\frac{Q_\ell}{Q}$ (using internal modulus switching to Q_ℓ), and then multiplies the result by $\frac{Q}{Q_\ell}$ after the multiplication procedure. In this case, the ciphertexts will always stay modulo Q outside the multiplication procedure, but the multiplication will be done internally modulo $Q_\ell < Q$ and hence will be more efficient.

In this case, the noise of input ciphertexts after internal modulus switching from Q to Q_ℓ will be equal to

$$\hat{\boldsymbol{v}} = \frac{Q_\ell}{Q}\boldsymbol{v} + \boldsymbol{\varepsilon}_{\mathtt{round}} \text{ and } \hat{\boldsymbol{v}}' = \frac{Q_\ell}{Q}\boldsymbol{v}' + \boldsymbol{\varepsilon}'_{\mathtt{round}},$$

where $\boldsymbol{\varepsilon}_{\mathtt{round}}$ and $\boldsymbol{\varepsilon}'_{\mathtt{round}}$ have their norm bounded by $(1 + \delta_{\mathcal{R}} B_{\mathtt{key}})/2 \approx \delta_{\mathcal{R}}/2$. On the one hand,

	# NTTs	# integer-mult	# floating-point-oper
MultOld	$14k+7$	$(10k^2+26k+9)n$	$(10k+3)n$
MultNew	$14k$	$(9k^2+15k)n$	$7kn$
MultNewLeveled	14ℓ	$(4k\ell+5\ell^2+2k+18\ell)n$	$(2k+5\ell)n$

Table 1: Complexities of different multiplication methods.

the main consideration here is to choose ℓ such that $\frac{Q_\ell}{Q}\|\boldsymbol{v}\|_\infty$ remains significantly larger than $\|\boldsymbol{\varepsilon}_{\mathtt{round}}\|_\infty \approx \frac{\delta_{\mathcal{R}}}{2}$, so that the noise brought about by the modulus switching procedure will not significantly impact the overall noise growth. This is equivalent to

$$\|\boldsymbol{v}\|_\infty \gg \frac{Q\delta_{\mathcal{R}}}{2Q_\ell} \text{ and } \|\boldsymbol{v}\|'_\infty \gg \frac{Q\delta_{\mathcal{R}}}{2Q_\ell},$$

or in practice

$$\|\boldsymbol{v}\|_\infty > 8\frac{Q\delta_{\mathcal{R}}}{Q_\ell} \text{ and } \|\boldsymbol{v}\|'_\infty > 8\frac{Q\delta_{\mathcal{R}}}{Q_\ell}. \tag{12}$$

On the other hand, in order to gain as much as possible in efficiency, Q_ℓ must be chosen as the smallest modulus satisfying inequalities (12). Theoretically this requires to have a precise estimate of the average (or lower bound within a certain confidence interval) noise size. But in practice it is enough to add a heuristic "cushion" to our worst-case bound. See the full version for details.

Algorithm 3 New Leveled BFV Multiplication Algorithm

procedure LEVELEDNEWMULT($\mathtt{ct}=(\boldsymbol{c}_0,\boldsymbol{c}_1)\in\mathcal{R}_Q^2, \mathtt{ct}'=(\boldsymbol{c}'_0,\boldsymbol{c}'_1)\in\mathcal{R}_Q^2$)
- ModSwitchDown: $\hat{\mathtt{ct}}=\lfloor\frac{Q_\ell}{Q}\mathtt{ct}\rceil\in\mathcal{R}_{Q_\ell}^2$ and $\hat{\mathtt{ct}}'=\lfloor\frac{Q_\ell}{Q}\mathtt{ct}'\rceil\in\mathcal{R}_{Q_\ell}^2$
  - $\triangleright$ $\hat{\boldsymbol{v}}=\frac{Q_\ell}{Q}\boldsymbol{v}+\boldsymbol{\varepsilon}_{\mathtt{round}}$ and $\triangleright$ $\hat{\boldsymbol{v}}'=\frac{Q_\ell}{Q}\boldsymbol{v}'+\boldsymbol{\varepsilon}'_{\mathtt{round}}$
- $\hat{\mathtt{ct}}_{\mathtt{m}}=\mathtt{NewMult}(\hat{\mathtt{ct}},\hat{\mathtt{ct}}')\in\mathcal{R}_{Q_\ell}^3$
  - $\triangleright$ $\hat{\mathtt{ct}}_{\mathtt{m}}(\boldsymbol{s})=\frac{Q_\ell}{t}[\boldsymbol{m}_1\boldsymbol{m}_2]_t+\hat{\boldsymbol{v}}_{\mathtt{m}} \pmod{\mathcal{R}_{Q_\ell}}$
- ModSwitchUp: $\mathtt{ct}_{\mathtt{m}}=\lfloor\frac{Q}{Q_\ell}\hat{\mathtt{ct}}_{\mathtt{m}}\rceil\in\mathcal{R}_Q^3$
  - $\triangleright$ $\mathtt{ct}_{\mathtt{m}}(\boldsymbol{s})=\frac{Q}{t}[\boldsymbol{m}_1\boldsymbol{m}_2]_t+\boldsymbol{v}_{\mathtt{m}} \pmod{\mathcal{R}_Q}$
  - $\triangleright$ $\boldsymbol{v}_{\mathtt{m}}=\frac{Q}{Q_\ell}\hat{\boldsymbol{v}}_{\mathtt{m}}+\boldsymbol{v}_r$

Remark 3.3 *Note that this "leveled" optimization can be equally applied to key switching. The only difference in this case would be to ensure that the noise of the scaled ciphertext remains larger than the noise brought about by the key-switching procedure itself.*

Remark 3.4 *Modulus switching from Q to Q_ℓ and then from Q_ℓ to P_ℓ in Algorithm 3 can be combined into a single modulus switching from Q directly to P_ℓ. This reduces the number of integer multiplications from $(k\ell+\ell^2+2\ell)n$ to $(k\ell+\ell)n$. Note that an approximate modulus switching (instead of an exact one with a floating-point correction technique from [23]) can be employed by adding extra $\log k$ bits to the noise estimate used for reducing the number of levels inside homomorphic multiplication. Both exact and approximate procedures for switching the moduli of ciphertexts between arbitrary RNS bases are described in the full version.*

Table 1 summarizes the computational complexities of different multiplication algorithms by assuming that the extension from Q to QP is performed using the technique from [23] with floating-point operations.

Lazy scaling in BFV multiplication. An additional optimization can be implemented by noticing that tensoring and scaling can be separated to optimize some evaluation circuits. For example, consider the inner product circuit of two vectors of ciphertexts. We evaluate it by multiplying (tensoring and scaling) the pairs of ciphertexts and then adding the results $\pmod{\mathcal{R}_Q}$. It is more efficient to do this in a different way: first we apply the tensoring subroutine to the pairs of ciphertexts, then add the results $\pmod{\mathcal{R}_{QP}}$, and finally perform the expensive scaling subroutine only once. Indeed, after tensoring we already have the information about the multiplicative noise $\boldsymbol{v}_{\texttt{tensor}}$, thus changing the order of scaling and additions does not affect the $\boldsymbol{v}_{\texttt{tensor}}$ noise. Moreover, as we perform the scaling down only once instead of doing it for each pair of ciphertexts, the total noise from the inner product is actually reduced. We call this technique *lazy scaling* and describe the pseudocode in the full version. The experimental results in the full version suggest that this optimization can speed up inner products by more than 2x in practice.

3.3 Improved Decryption in the HPS RNS variant

A significant practical limitation of the HPS approach is that high-precision ("long double" or even quad-precision) floating-point arithmetic is required to support larger CRT moduli [23]. We introduce a general-purpose digit decomposition technique (inspired by digit decomposition in key switching) and apply it to the HPS decryption procedure to add support for arbitrary CRT moduli using only regular double-precision floating-point arithmetic, hence overcoming this limitation of the HPS variant.

The idea of HPS scaling [23] for decryption can be briefly explained as follows: for $x \in \mathbb{Z}_Q$ with CRT representation $(x_1, \ldots, x_k)$ we want to compute an integer $y = \lceil t/Qx \rfloor \in \mathbb{Z}_t$, and use a CRT composition formula to derive the following expression:

$$y := \left\lceil \frac{t}{Q} x \right\rfloor = \left\lceil \left(\sum_{i=1}^{k} x_i \cdot \left[\left(\frac{Q}{q_i} \right)^{-1} \right]_{q_i} \cdot \frac{Q}{q_i} \cdot \frac{t}{Q} \right) - u \cdot Q \cdot \frac{t}{Q} \right\rfloor =$$

$$= \left[\left\lceil \left(\sum_{i=1}^{k} x_i \cdot \left(\left[\left(\frac{Q}{q_i} \right)^{-1} \right]_{q_i} \cdot \frac{t}{q_i} \right) \right) \right\rfloor \right]_t = \left[\left(\sum_{i=1}^{k} x_i \cdot \omega_i \right) + \left\lceil \sum_{i=1}^{k} x_i \cdot \theta_i \right\rfloor \right]_t,$$

$$\text{where } \left[\left(\frac{Q}{q_i} \right)^{-1} \right]_{q_i} \cdot \frac{t}{q_i} = \omega_i + \theta_i \text{ with } \omega_i \in \mathbb{Z}_t \text{ and } \theta_i \in \left[-\frac{1}{2}, \frac{1}{2} \right).$$

As we can only store approximate values $\tilde{\theta}_i = \theta_i + \epsilon_i$, the magnitude of the error term $|\epsilon'| = |\sum_i x_i \epsilon_i|$ in the fractional part is limited by $k q_m \epsilon_m$, where $q_m = \max_i(q_i)$ and $\epsilon_m = \max_i(\epsilon_i)$. If we restrict the floating-point precision to "doubles", which are natively supported by modern CPUs, we have to introduce a constraint $k q_m < 2^{51}$. To support larger CRT moduli, we need "long doubles" or even quad-precision arithmetic: long doubles are needed for CRT moduli from 47 to 58 bits, and quad-precision floats are needed for higher CRT moduli [23].

Our main idea is to perform digit decomposition, somewhat similar to how digit decomposition is done in BV key-switching, to replace the factor q_m with a smaller digit of it. For base $B_s \in \mathbb{Z}$, $B_s \geq 2$, let $d_s = \lceil \log(q_m)/\log(B_s) \rceil$. Let $x_i = \sum_{j=0}^{d_s-1} x_{i,j} \cdot B_s^j$ be the B_s base decomposition of x_i. Then the expression for y can be rewritten as

$$y = \left[\left[\left(\sum_{i=1}^{k} \sum_{j=0}^{d_s-1} x_{i,j} \cdot \left(\frac{t}{q_i} \cdot \left[\left[\left(\frac{Q}{q_i} \right)^{-1} \right]_{q_i} \cdot B_s^j \right]_{q_i} \right) \right) \right] \right]_t$$
$$= \left[\left(\sum_{i=1}^{k} \sum_{j=0}^{d_s-1} \left[x_{i,j} \cdot \omega_{i,j} \right]_t \right) + \left[\sum_{i=1}^{k} \sum_{j=0}^{d_s-1} x_{i,j} \cdot \theta_{i,j} \right] \right]_t ,$$

$$\text{where } \frac{t}{q_i} \cdot \left[\left[\left(\frac{Q}{q_i} \right)^{-1} \right]_{q_i} \cdot B_s^j \right]_{q_i} = \omega_{i,j} + \theta_{i,j}, \quad \text{with } \omega_{i,j} \in \mathbb{Z}_t \text{ and } \theta_{i,j} \in \left[-\frac{1}{2}, \frac{1}{2} \right).$$

Note that $\omega_{i,j}$ and $\theta_{i,j}$ are the new precomputation factors instead of ω_i and θ_i.

Error analysis. The magnitude of the error term $|\epsilon'| = |\sum_{i,j} x_{i_j} \epsilon_{i,j}|$ is now limited by $\left| \sum_{i,j} x_{i,j} \epsilon_{i,j} \right| < k d_s B_s \epsilon_m$. In practice, our moduli q_i are normally bounded by 64 (or often by 60) bits. We have already considered the case of $kq_m < 2^{51}$. If $kq_m > 2^{51}$, we can take $d_s = 2$, $B_s = 2^{\lceil \log_2 q_m/2 \rceil}$. Then $|\epsilon'| < 2^{-19} k < 1/4$ for $k < 2^{17}$. Hence the floating-point error will have no effect on the result for any practically reasonable value of k.

Complexity analysis. The procedure takes kd_s floating-point multiplications, kd_s modular integer multiplications, some modular additions, and one rounding to compute $\lceil u \rfloor$. However, if $tkd_sB_s < 2^{64}$, then we can replace modular multiplications and modular additions by plain integer multiplications and additions respectively, and do one modular reduction at the end.

Remark 3.5 *Note that this digit decomposition technique can be applied to other mixed integer/floating-point RNS operations to reduce precision requirements for floating-point arithmetic or avoid extra noise due to floating-point rounding. For instance, it can be used in the scaling for homomorphic multiplication.*

4 More Usable BGV Scheme

The practical use of the BGV scheme requires accurate dynamic noise estimation to decide when the modulus operation should be executed, and what scaling factor should be chosen for modulus switching [22]. Each modulus switching decision may significantly impact the noise not only for the current operation, but also for all subsequent operations. An error in a noise estimate may eventually lead even to a decryption failure. Therefore, fine-tuned noise estimation techniques are used to estimate the noise for various operations (see [25] for a more detailed discussion). In contrast, the BFV scheme is much more robust to inaccuracies in noise estimation and typically only requires an upper bound on the error for the desired multiplicative depth. This robustness of BFV is related to the use of the MSD encoding and scaling down by a large factor Q/t during BFV homomorphic multiplication, and the "fragility" of BGV is caused by the LSD encoding and

scaling down by a small factor, comparable in magnitude to the noise incurred in operations after the previous modulus switching. For this reason, many modern homomorphic encryption libraries implement `BFV` as the scheme for finite fields, while only HELib and PALISADE (since quite recently) provide efficient implementations of `BGV`.

We present an alternative approach for instantiating `BGV`, which does not require dynamic noise estimation. For this instantiation, one only needs to specify the multiplicative depth of the computation, maximum number of additions per multiplicative level, and the number of additions and automorphisms before first multiplication. Then all moduli $Q_L, Q_{L-1}, \ldots, Q_1, Q_0$ are chosen so that a small, constant level of noise can be maintained throughout the computation. All modulus switching operations are automatically performed right before a homomorphic multiplication. Conceptually, this `BGV` instantiation is similar in usability to `BFV`, where only the multiplicative depth needs to be specified upfront and all "modulus switching" operations are performed automatically.

The logic for choosing the moduli is as follows. We start with a fresh encryption that has a noise $\|\boldsymbol{v}_{\mathtt{fresh}}\|_\infty = B_{\mathtt{err}}(2\delta_{\mathcal{R}} B_{\mathtt{key}} + 1)$. Then we perform automorphism operations and additions, and apply modulus switching right before the first multiplication. This additional modulus switching before first multiplication allows us to reset the noise to a value comparable to the modulus switching noise, which will be the constant noise level $\|\boldsymbol{v}_{\mathtt{c}}\|_\infty$ we will maintain throughout the computation. This can be expressed as

$$\frac{Q_L}{Q_{L+1}}\left((n_{\text{add}}+1)\|\boldsymbol{v}_{\mathtt{fresh}}\|_\infty + n_{\text{ks}}\|\boldsymbol{v}_{\mathtt{ks}}\|_\infty\right) + \frac{1+\delta_{\mathcal{R}} B_{\mathtt{key}}}{2} \leq \|\boldsymbol{v}_{\mathtt{c}}\|_\infty,$$

where n_{add} and n_{ks} are the numbers of additions and automorphism operations, respectively, that are performed before first multiplication, and $\|\boldsymbol{v}_{\mathtt{ks}}\|_\infty$ is the bound on key switching noise. Note that here we introduced a new level and corresponding new modulus Q_{L+1} to account for an extra level we added before first multiplication. It is best to choose Q_{L+1}/Q_L such that $\|\boldsymbol{v}_{\mathtt{c}}\|_\infty \approx 1 + \delta_{\mathcal{R}} B_{\mathtt{key}}$ to achieve the smallest constant error because this error will allow us to minimize the subsequent values of Q_{i+1}/Q_i for most practical scenarios, hence resulting in the minimum value of ciphertext modulus Q_{L+1} (see the full version for more details, and more general expression for optimal $\|\boldsymbol{v}_{\mathtt{c}}\|_\infty$).

Then for multiplication levels (from L to 1), we have to satisfy

$$\frac{Q_i}{Q_{i+1}}\left((n'_{\text{add}}+1)\frac{\delta_{\mathcal{R}} t}{2}(2\|\boldsymbol{v}_{\mathtt{c}}\|_\infty^2 + 2\|\boldsymbol{v}_{\mathtt{c}}\|_\infty + 1) + (n'_{\text{ks}}+1)\|\boldsymbol{v}_{\mathtt{ks}}\|_\infty\right) + \frac{1+\delta_{\mathcal{R}} B_{\mathtt{key}}}{2} \leq \|\boldsymbol{v}_{\mathtt{c}}\|_\infty,$$

where n'_{add} and n'_{ks} are the maximum numbers of additions and key switching operations, respectively, allowed per any multiplication level (going from L down to 1). For simplicity we use these maximum values across all levels so that Q_{i+1}/Q_i could have roughly same value for all $i \in \{1, \ldots, L-1\}$. Note that for `Hybrid` key switching and relatively large plaintext moduli, such as $t = 2^{16} + 1$, which is often used for CRT packing, the multiplication noise is always much higher than $\|\boldsymbol{v}_{\mathtt{ks}}\|_\infty$ (see derivations in the full version). Hence for this case we can rewrite the expression as

$$\frac{Q_i}{Q_{i+1}}\left((n'_{\text{add}}+1)\frac{\delta_{\mathcal{R}} t}{2}(2\|\boldsymbol{v}_{\mathtt{c}}\|_\infty^2 + 2\|\boldsymbol{v}_{\mathtt{c}}\|_\infty + 1)\right) + \frac{1+\delta_{\mathcal{R}} B_{\mathtt{key}}}{2} \leq \|\boldsymbol{v}_{\mathtt{c}}\|_\infty.$$

The last modulus Q_0 is chosen such that decryption is correct for a ciphertext with noise bounded by $\|\boldsymbol{v}_{\mathtt{c}}\|_\infty$. This implies that $Q_0 > 2t\|\boldsymbol{v}_{\mathtt{c}}\|_\infty - t$.

It is easy to show that once $L, n_{\text{add}}, n_{\text{ks}}, n'_{\text{add}}$, and n'_{ks} (only needed for small t) are given, all moduli Q_i can be derived. First we compute Q_0, then all values $Q_1, \ldots, Q_L$, and finally we can find Q_{L+1}.

This logic is simple to implement and avoids any dynamic noise estimation during the computation. It is also robust to inaccurate estimates as long as the upper bound for δ_R is chosen appropriately, which is very similar to what is done for BFV. There is a cost for this simplicity and robustness. The moduli Q_{L+1} may be larger than the values obtained using the more granular approach with dynamic noise estimation [25] because we use the maximum values of n'_{add} and n'_{ks} over all intermediate levels. However, our experimental results show that this instantiation of BGV can be significantly faster than the improved BFV implementation described in Section 3.

Remarks on the RNS instantiation. Recall that for original BGV, we choose $Q = q_0 \cdots q_L$ and denote $Q_i = q_0 \cdots q_i$ for $0 \leq i \leq L$, where all $q_i = 1 \bmod 2N$ and co-prime to each other. In the case of our BGV variant, an extra q_{L+1} is introduced to reset the "fresh" noise to modulus switching noise. It is easy to show that for this setup, $Q_0 = q_0$, $Q_{i+1}/Q_i = q_{i+1}$, and $Q_{L+1}/Q_L = q_{L+1}$.

The expressions for finding q_0, q_i, q_{L+1}, where $i \in \{1, \ldots, L\}$, can be written as follows:

$$q_0 > 2t \left\| \boldsymbol{v}_{\mathtt{c}} \right\|_\infty - t, \tag{13}$$

$$q_i > 2 \left((n'_{\text{add}} + 1) \frac{\delta_{\mathcal{R}} t}{2} (2 \left\| \boldsymbol{v}_{\mathtt{c}} \right\|_\infty + 2 + \frac{1}{\left\| \boldsymbol{v}_{\mathtt{c}} \right\|_\infty}) + (n'_{\text{ks}} + 1) \frac{\left\| \boldsymbol{v}_{\mathtt{ks}} \right\|_\infty}{\left\| \boldsymbol{v}_{\mathtt{c}} \right\|_\infty} \right), \tag{14}$$

$$q_{L+1} > 2 \left((n_{\text{add}} + 1) \frac{\left\| \boldsymbol{v}_{\mathtt{fresh}} \right\|_\infty}{\left\| \boldsymbol{v}_{\mathtt{c}} \right\|_\infty} + n_{\text{ks}} \frac{\left\| \boldsymbol{v}_{\mathtt{ks}} \right\|_\infty}{\left\| \boldsymbol{v}_{\mathtt{c}} \right\|_\infty} \right),$$

where we take $\left\| \boldsymbol{v}_{\mathtt{c}} \right\|_\infty = 1 + \delta_{\mathcal{R}} B_{\mathtt{key}}$.

Handling crosslevel operations and scaling factors. The GHS variant implemented in HElib uses ciphertext-specific scaling factors, which introduces some complications that may affect the usability and may require additional scalar multiplications to bring two ciphertexts to the same scaling factor. In our BGV variant, we chose a simpler approach where the same scaling factor is used for all ciphertexts at a specific level, which reduces the number of scalar multiplications. This approach was originally introduced for the CKKS scheme in [27], and in our work we adapt it to BGV.

5 Comparison of BFV and BGV

5.1 Noise Growth

When comparing BGV and BFV, it is convenient to use the leveled approach of BGV, first comparing Q_0, then Q_i, and finally Q_{L+1}.

For Q_0, our modified variant of BFV has identical noise as BGV, i.e., Equation (8) is exactly the same as Equation (13).

For Q_{i+1}/Q_i, which corresponds to each multiplicative level, the dominant term in the BFV expression given by Equation (11) is $\delta_R^2 t B_{\mathtt{key}} V$, where V is the largest of the errors in two multiplied ciphertexts. For BGV, Equation (14) suggests that the dominant term is $2\delta_R^2 t B_{\mathtt{key}} V$. In other words, the expressions for BFV and BGV are identical except for the extra multiplicative factor of 2. This

factor appears in BGV because we ensure that at each level the downscaled noise matches the added modulus switching noise, keeping the noise level constant at twice the modulus switching noise (see the full version for details). In the case of BFV, the quadratic noise (product of prior noises for each ciphertext) is negligible as we downscale the ciphertext by a large factor Q/t, and we only observe the pure modulus switching noise. In other words, BFV has a small benefit of using one bit less per multiplication level.

There is also an extra advantage of BFV for small plaintext moduli, e.g., $t = 2$. As the analysis in the full version shows, the key switching noise in this case becomes comparable to multiplication noise for BGV, which implies higher values of Q_{i+1}/Q_i. In contrast, the key switching noise may only affect the initial level in BFV, as afterwards the accumulated noise from prior multiplications will be much higher than additive key switching noise, which is independent of current ciphertext noise. When we switch to larger plaintext moduli, this BFV advantage disappears as the key-switching noise in BGV becomes negligible compared to multiplication noise (as shown in the full version).

Using the $(L+1)$-th level (q_{L+1} in the RNS version) is preferred in BGV to achieve the smallest constant noise (twice the modulus switching noise). If $(L+1)$-th level is not used, then the fresh noise will make each Q_{i+1}/Q_i larger by a factor $\|\boldsymbol{v}_{\mathtt{fresh}}\|_\infty / \|\boldsymbol{v}_{\mathtt{c}}\|_\infty \approx 2B_{\mathtt{err}} \approx 37$. Although one could use an auxiliary modulus in hybrid key switching during encryption instead (see the end of Section 3.1), extra noise can be accumulated from additions and/or key switching operations performed before first multiplication, which would increase all subsequent Q_{i+1}/Q_i. So the least level of constant noise in BGV, and hence smallest Q_{i+1}/Q_i, can be guaranteed only by introducing a relatively small extra "noise budget" for pseudo-level $L+1$. Note that in BFV it is best to use an auxiliary modulus to reset the fresh noise to smaller modulus switching noise, without increasing the ciphertext modulus (see Section 3.1). Hence no pseudo-level $L+1$ is needed in BFV, which is another small advantage of BFV over BGV.

In summary, the improved variants of BFV and BGV presented in this work have very similar noise growth, but BFV has some minor advantages over BGV, resulting in somewhat reduced ciphertext moduli needed to support the same computations.

5.2 Computational Complexity

The main difference between BFV and BGV in terms of computational complexity is due to the scaling method used in the multiplication operation. As was previously mentioned, BFV uses the MSD encoding and scales down the tensor product by a large Q/t factor, while BGV uses the LSD encoding technique to scale the tensor product only by a relatively small factor, comparable to the noise of previous multiplication. Considering that the noise growth is very similar in both schemes, one can expect that the computational complexity of BFV multiplication will be significantly higher. The purpose of this section is to quantify this difference, and examine the effect of plaintext moduli on this difference. Note that all other operations, such as addition and automorphism, use the same approach in both schemes, and do not have any significant difference in terms of theoretical complexity. Hence we focus on the operation of multiplication.

The analysis in Section 3.2 shows that our leveled BFV multiplication takes 14ℓ NTTs and $(4k\ell + 5\ell^2 + 2k + 18\ell)n$ integer multiplications (we ignore for simplicity a much smaller contribution of floating-point operations). We also add the computational cost of hybrid key switching used for relinearization as there is a difference in its cost between BFV and BGV. For key-switching we assume that the ciphertext element is decomposed into $d_{\text{num}} = \ell/\alpha$ digits, i.e. each digit is considered modulo α moduli. The cost of relinearization for BFV is $4\ell + 2\alpha$ NTTs and $n(3\alpha\ell + 2d_{\text{num}}\ell + 2\alpha + 5\ell)$

integer multiplications (see the full version for details). Here, for simplicity of analysis we assume that ℓ is the same for leveled BFV multiplication and subsequent relinearization. The total cost of multiplication and relinearization in BFV is $17\ell + 2\alpha$ NTTs and $n(4k\ell + 5\ell^2 + 2k + 23\ell + 3\alpha\ell + 2d_{\text{num}}\ell + 2\alpha)$ integer multiplications.

In the case of our BGV variant, the total cost of multiplication includes two modulus switching operations for input ciphertexts, the tensor product, and, finally, the relinearization. The cost of modulus switching is $4(\ell' + 1)$ NTTs and $4n(\ell' + 2)$ integer multiplications, where ℓ' is the number of CRT moduli after modulus switching. The cost of tensor product is $4n\ell'$ integer multiplications. The cost of relinearization in the case of BGV is: $4\ell' + 2\alpha'$ NTTs and $n(3\alpha'\ell' + 2d_{\text{num}}\ell' + 4\alpha' + 7\ell')$ integer multiplications. Hence the total cost of multiplication and relinearization is $8\ell' + 4 + 2\alpha'$ NTTs and $n(3\alpha'\ell' + 2d_{\text{num}}\ell' + 4\alpha' + 15\ell' + 8)$ integer multiplications.

One can observe that the number of NTTs needed for BFV multiplication appears to be 2x or even higher than for BGV. But we should keep in mind that typically $\ell' > \ell$. For example, when $t = 2$, we can even have $\ell' > 3\ell$ since in BFV we work with large (60-bit) moduli vs the moduli of size $\delta_R^2 t$ (less than 20 bits) in BGV. On the other hand, the cost of integer multiplications in BFV appears to be significantly higher due to multiple basis extension operations. The above may suggest that the complexity of BFV could be lower than for BGV at small t, while more significant benefits of BGV are expected as t is increased, when the ratio of ℓ'/ℓ becomes smaller than 2, which corresponds to the typical value of $t = 2^{16} + 1$ used for CRT packing. One could argue that this is essentially due to the assumption that the computations modulo each CRT moduli are implemented on different machine words, which is typically true for practical implementations of homomorphic encryption. As a consequence, while BGV might be practically slower than BFV at small t for classical implementations, we stress that this is only due to the way the CRT representation is usually implemented and that BGV still has a lower theoretical complexity than BFV even for small plaintext moduli.

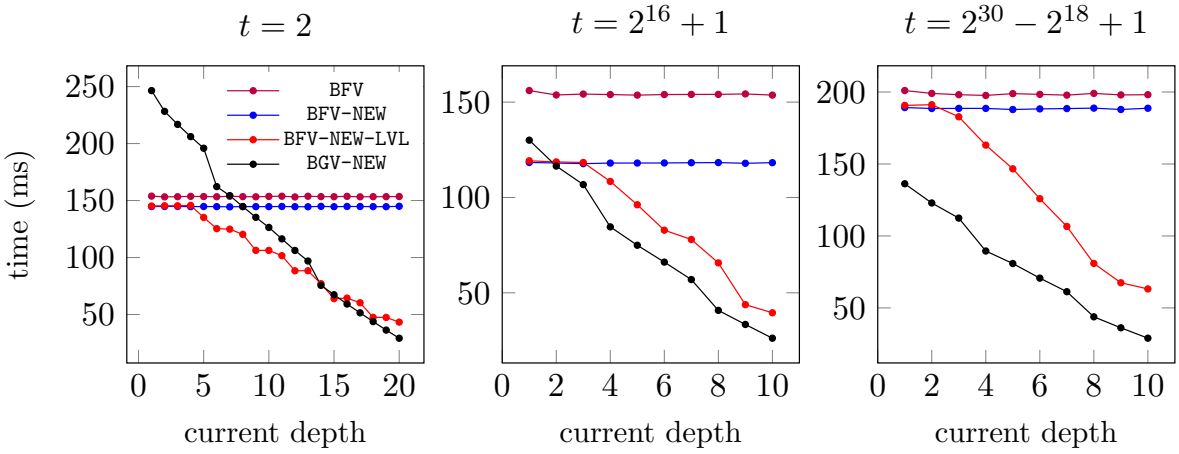
Remark 5.1 *To reduce even further the computational cost of BGV, one could trunk some CRT moduli together in the same 64-bit machine word. This would allow one to divide the number of moduli ℓ', and thus of NTTs, by a factor of 2 when the moduli are smaller than 30 bits ($t \approx 2^{11}$) and by a factor of 3 when they are smaller than 20 bits ($t \approx 2$).*

5.3 Software Implementation and Experimentation Setup

We implemented all variants of BFV and BGV in PALISADE v1.10.4. The evaluation environment was a commodity desktop computer system with an Intel(R) Core(TM) i7-9700 CPU @ 3.00GHz and 64 GB of RAM, running Ubuntu 18.04 LTS. The compiler was g++ 9.3.0. All experiments were executed in the single-threaded mode.

PALISADE includes the implementation of both BEHZ and HPS variants of BFV. The runtime results and noise growth for both variants are roughly the same (as shown in Section 5.4). We chose the HPS variant as the main RNS variant for our BFV modifications due to its relative simplicity. We denote our modified BFV variant as BFV-NEW, our modified BFV variant with leveled multiplication as BFV-NEW-LVL, and our BGV variant as BGV-NEW. Note that our implementation does not trunk small CRT moduli in BGV for small values of t, i.e., it does not include the optimization suggested in Remark 5.1.

Figure 1: Comparison of homomorphic multiplication runtimes for BFV and BGV variants at various depths as a function of plaintext modulus t. Hybrid key switching with 3 digits, i.e., $d_{\text{num}} = 3$, was used, and N was set to 2^{15}.



5.4 Performance Comparison

Figure 1 illustrates the comparison of homomorphic multiplication runtimes for the BFV and BGV variants developed in this work to the baseline for the prior state-of-the-art BFV implementation of the HPS variant [23]. The first major observation is that BFV-NEW-LVL outperforms BGV-NEW for small plaintext moduli (at least up to depth 20), while BGV-NEW runs significantly faster than BFV-NEW-LVL for intermediate and large plaintext moduli, i.e., $t = 2^{16} + 1$ and $t = 2^{30} - 2^{18} + 1$. This observation is in agreement with our theoretical complexity analysis in Section 5.2 since our implementation does not include the optimization suggested in Remark 5.1, i.e., small moduli are not trunked together. The second significant observation is that our best BFV variant, labeled as BFV-NEW-LVL, speeds up the runtime of deeper multiplications (depth-20 for $t = 2$ and depth-10 for higher t) by 3x-4x, as compared to the BFV baseline.

Table 2 shows the comparison of noise growth and runtimes for a binary tree computation ranging in multiplicative depth from 1 to 7. First, we want to point out that the noise growth and runtimes for the BEHZ and HPS variants are very close, with HPS having somewhat better runtime efficiency, which agrees well with the noise analysis in [7] and runtime comparison in [3]. In view of this, we chose HPS as the main variant for our BFV improvements (but similar gains can be expected for the BEHZ variant). Our next observation is that BGV has a slightly faster noise growth, as compared to all BFV variants, with the difference in noise increasing with depth (as predicted in Section 5.1). Note that the original BFV variants have somewhat higher noise (by almost constant number of bits) as compared to our BFV variants because they do not use the technique of encrypting with a slightly larger modulus Qp, followed with scaling by p. Our final observation is that BGV-NEW has a minor speed-up over the best BFV variant for the chosen plaintext modulus $t = 2^{16} + 1$. Note that the speed-up is observed only for this or higher plaintext moduli, with BFV-NEW-LVL becoming faster for $t = 2$ (see the full version for details). Tables in the appendix of the full version also show the more significant effect of $r_t(Q)$ on noise magnitude at larger plaintext moduli for the original BFV, as theoretically predicted in Section 3.

Table 3 illustrates the comparison of noise growth and runtimes for a polynomial evaluation

Table 2: Comparison of noise growth and runtimes of BFV and BGV variants for a benchmark computation $\prod_{i=1}^{2^k} x_i$. Hybrid key switching with 3 digits, i.e., $d_{\text{num}} = 3$, was used, t was set to $2^{16}+1$, and $\lambda \geq 128$. Here, e denotes the current noise magnitude, $\log Q$, the size of the BFV ciphertext modulus, and $\log Q_L$, the equivalent ciphertext modulus in BGV without the last CRT modulus q_{L+1}.

	Original BFV							Our BFV							Our BGV				
	params			BEHZ		HPS		params			BFV-NEW		BFV-NEW-LVL		params			BGV-NEW	
k	$\log N$	$\log q_i$	$\log Q$	$\log e$	time(s)	$\log e$	time(s)	$\log N$	$\log q_i$	$\log Q$	$\log e$	time(s)	$\log e$	time(s)	$\log N$	$\log q_i$	$\log Q_L$	$\log e$	time(s)
1	13	31	62	45	0.011	44	0.01	13	59	59	36	0.004	35	0.004	13	33	58	34	0.005
2	13	47	94	66	0.034	66	0.03	13	45	90	63	0.025	63	0.025	13	33	91	67	0.02
3	14	43	129	102	0.24	103	0.21	14	41	123	95	0.19	96	0.18	13	33	124	100	0.063
4	14	53	159	131	0.52	132	0.45	14	52	156	125	0.4	125	0.39	13	33	157	133	0.17
5	14	48	192	158	1.41	161	1.2	14	47	188	155	1.07	155	1.04	14	34	196	171	0.8
6	14	56	224	189	2.85	189	2.44	14	55	220	184	2.18	184	2.13	14	34	230	205	2.03
7	14	51	255	221	7.61	220	6.51	14	50	250	214	5.98	214	5.73	14	34	264	239	4.86

benchmark. Our first observation is that BGV-NEW has a significantly higher noise than all BFV variants because the moduli q_i in this case require extra room for the additions at each level (the deepest level has the most significant effect on all q_i's). BGV-NEW again has a minor advantage in terms of runtime as compared to our best BFV variant for $t = 2^{16} + 1$, but BFV-NEW-LVL becomes faster when we decrease t to smaller values (see the full version for details). Note that for $k = 8$, BGV-NEW has a smaller ring dimension than all BFV variants, which is an effect of the automated logic for hybrid key switching used in the implementation, rather than a result of better noise growth in BGV (since $\log Q$ in BFV is significantly smaller than $\log Q_L$ in BGV).

Table 3: Comparison of noise growth and runtimes of BFV and BGV variants for a benchmark computation $\prod_{i=0}^{k} a_i x^i$: $|a_i| < 16$. Hybrid key switching with 3 digits, i.e., $d_{\text{num}} = 3$, was used, t was set to $2^{16}+1$, and $\lambda \geq 128$. Here, e denotes the current noise magnitude, $\log Q$, the BFV ciphertext modulus, and $\log Q_L$, the equivalent ciphertext modulus in BGV without the last CRT modulus q_{L+1}.

	Original BFV							Our BFV							Our BGV				
	params			BEHZ		HPS		params			BFV-NEW		BFV-NEW-LVL		params			BGV-NEW	
k	$\log N$	$\log q_i$	$\log Q$	$\log e$	time(s)	$\log e$	time(s)	$\log N$	$\log q_i$	$\log Q$	$\log e$	time(s)	$\log e$	time(s)	$\log N$	$\log q_i$	$\log Q_L$	$\log e$	time(s)
2	13	34	68	41	0.012	40	0.01	13	32	64	35	0.009	36	0.009 s	13	38	68	38	0.007
4	13	50	100	76	0.034	76	0.03	13	48	96	67	0.026	67	0.025 s	13	38	107	74	0.024
8	14	45	135	106	0.25	107	0.22	14	43	129	100	0.19	100	0.18 s	13	39	148	116	0.061
16	14	56	168	138	0.53	138	0.46	14	54	162	130	0.4	130	0.33 s	14	41	197	163	0.28
32	14	50	200	166	1.43	167	1.22	14	49	196	161	1.1	161	0.78 s	14	42	244	208	0.61
48	14	58	232	197	2.16	198	1.85	14	57	228	191	1.66	190	1.22 s	14	42	286	251	1.07
64	14	58	232	199	2.89	199	2.48	14	57	228	191	2.22	191	1.54 s	14	43	293	256	1.27

6 Concluding Remarks

Our theoretical analysis and experimental results show that the modified BFV variant has somewhat better noise growth than BGV for all plaintext moduli, though previous results suggested that BGV has

a better noise growth than BFV for larger plaintext moduli [14, 15]. This result is mainly due to our modification of the BFV encryption procedure. The other major conclusion is that, when the moduli of BGV are not trunked together, BFV is significantly faster for small plaintext moduli, e.g., $t = 2$, with BGV becoming faster as the plaintext modulus is increased.

The variant of BGV presented in this paper was mainly motivated by improving the usability of the scheme, which is known to be more challenging for use than BFV. From this perspective, this BGV variant is as easy to use as the implementation of BFV in PALISADE. However, the usability also has some performance cost, e.g., we have to choose the size of CRT moduli more conservatively. It would be interesting to examine how the performance of our BGV variant compares to the BGV design with dynamic noise estimation, which is implemented in HElib. It would not be fair to compare the PALISADE implementation directly with the HElib implementation as one would mostly observe the effect of differences in the efficiency of primitive ring operations, such as NTTs, rather than the differences between the BGV variants. For a fair comparison, a PALISADE implementation of the dynamic-noise BGV variant would be needed. Another potential improvement for BGV is to consider the idea of trunking multiple small CRT moduli mentioned in Remark 5.1. We plan to examine both ideas in our future work.

7 Acknowledgments

Andrey Kim and Yuriy Polyakov's NJIT work was supported in part by the Defense Advanced Research Projects Agency (DARPA) and the US Navy SPAWAR Systems Center Pacific (SSC-PAC) under Contract Number N66001-17-1-4043 and the Office of the Director of National Intelligence (ODNI), Intelligence Advanced Research Projects Activity (IARPA), via Contract No. 2019-1902070006. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the Department of Defense, ODNI, IARPA, or the U.S. Government.

Vincent Zucca's KU Leuven work was supported in part by the Research Council KU Leuven grant C14/18/067, CyberSecurity Research Flanders with reference number VR20192203, and the IARPA HECTOR project under the solicitation number IARPA-BAA-17-05.

We also thank Charlotte Bonte for a careful review of the first version of the paper, her feedback, and fruitful discussions that helped us to improve the paper.

References

[1] Lattigo v2.1.1. Online: http://github.com/ldsec/lattigo, Dec. 2020. EPFL-LDS.

[2] PALISADE Lattice Cryptography Library (release 1.10.6). https://palisade-crypto.org/, Dec. 2020.

[3] A. Al Badawi, Y. Polyakov, K. M. M. Aung, B. Veeravalli, and K. Rohloff. Implementation and performance evaluation of rns variants of the bfv homomorphic encryption scheme. *IEEE Transactions on Emerging Topics in Computing*, 9(2):941–956, 2021.

[4] M. Albrecht, M. Chase, H. Chen, and et al. Homomorphic encryption security standard. Technical report, HomomorphicEncryption.org, Toronto, Canada, November 2018.

[5] J. Alperin-Sheriff and C. Peikert. Practical bootstrapping in quasilinear time. In R. Canetti and J. A. Garay, editors, *Advances in Cryptology – CRYPTO 2013*, pages 1–20, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[6] J.-C. Bajard, J. Eynard, M. A. Hasan, and V. 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.

[7] J. C. Bajard, J. Eynard, P. Martins, L. Sousa, and V. Zucca. Note on the noise growth of the RNS variants of the BFV scheme. Cryptology ePrint Archive, Report 2019/1266, 2019. `https://eprint.iacr.org/2019/1266`.

[8] J. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. M. Schanck, P. Schwabe, G. Seiler, and D. Stehle. Crystals - kyber: A cca-secure module-lattice-based kem. In *2018 IEEE European Symposium on Security and Privacy (EuroS P)*, pages 353–367, 2018.

[9] Z. Brakerski. Fully homomorphic encryption without modulus switching from classical GapSVP. In *Annual Cryptology Conference*, pages 868–886. Springer, 2012.

[10] Z. Brakerski, C. Gentry, and V. Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. *ACM Transactions on Computation Theory (TOCT)*, 6(3):1–36, 2014.

[11] Z. Brakerski and V. Vaikuntanathan. Fully homomorphic encryption from ring-LWE and security for key dependent messages. In *Annual cryptology conference*, pages 505–524. Springer, 2011.

[12] J. H. Cheon, A. Kim, M. Kim, and Y. Song. Homomorphic encryption for arithmetic of approximate numbers. In *ASIACRYPT 2017*, pages 409–437, 2017.

[13] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène. Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. In J. H. Cheon and T. Takagi, editors, *ASIACRYPT 2016*, pages 3–33, 2016.

[14] A. Costache, K. Laine, and R. Player. Evaluating the Effectiveness of Heuristic Worst-Case Noise Analysis in FHE. In L. Chen, N. Li, K. Liang, and S. Schneider, editors, *Computer Security – ESORICS 2020*, pages 546–565, 2020.

[15] A. Costache and N. P. Smart. Which Ring Based Somewhat Homomorphic Encryption Scheme is Best? In K. Sako, editor, *Topics in Cryptology - CT-RSA 2016*, pages 325–340, Cham, 2016. Springer International Publishing.

[16] E. Crockett and C. Peikert. $\lambda \circ \lambda$: Functional lattice cryptography. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, page 9931005, New York, NY, USA, 2016. Association for Computing Machinery.

[17] L. Ducas and D. Micciancio. FHEW: Bootstrapping Homomorphic Encryption in Less Than a Second. In E. Oswald and M. Fischlin, editors, *Advances in Cryptology – EUROCRYPT 2015*, pages 617–640, 2015.

[18] J. Fan and F. Vercauteren. Somewhat practical fully homomorphic encryption. *IACR Cryptol. ePrint Arch.*, 2012:144, 2012.

[19] C. Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the forty-first annual ACM symposium on Theory of computing*, pages 169–178, 2009.

[20] C. Gentry and S. Halevi. Implementing gentry's fully-homomorphic encryption scheme. In K. G. Paterson, editor, *Advances in Cryptology – EUROCRYPT 2011*, pages 129–148, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[21] C. Gentry, S. Halevi, and N. P. Smart. Fully Homomorphic Encryption with Polylog Overhead. In D. Pointcheval and T. Johansson, editors, *Advances in Cryptology – EUROCRYPT 2012*, pages 465–482, 2012.

[22] C. Gentry, S. Halevi, and N. P. Smart. Homomorphic evaluation of the AES circuit. In *Annual Cryptology Conference*, pages 850–867. Springer, 2012.

[23] S. Halevi, Y. Polyakov, and V. Shoup. An improved RNS variant of the BFV homomorphic encryption scheme. In *Cryptographers Track at the RSA Conference*, pages 83–105. Springer, 2019.

[24] S. Halevi and V. Shoup. Bootstrapping for HElib. Cryptology ePrint Archive, Report 2014/873, 2014. `https://eprint.iacr.org/2014/873`.

[25] S. Halevi and V. Shoup. Design and implementation of HElib: a homomorphic encryption library. Cryptology ePrint Archive, Report 2020/1481, 2020.

[26] K. Han and D. Ki. Better bootstrapping for approximate homomorphic encryption. In *Cryptographers Track at the RSA Conference*, pages 364–390. Springer, 2020.

[27] A. Kim, A. Papadimitriou, and Y. Polyakov. Approximate homomorphic encryption with reduced approximation error. Cryptology ePrint Archive, Report 2020/1118, 2020. `https://eprint.iacr.org/2020/1118`.

[28] M. Kim, Y. Song, B. Li, and D. Micciancio. Semi-parallel logistic regression for GWAS on encrypted data. *BMC Medical Genomics*, 13(7):1–13, 2020.

[29] T. Lepoint and M. Naehrig. A Comparison of the Homomorphic Encryption Schemes FV and YASHE. In *Progress in Cryptology – AFRICACRYPT 2014*, pages 318–335, 2014.

[30] D. Micciancio and Y. Polyakov. Bootstrapping in FHEW-like Cryptosystems. Cryptology ePrint Archive, Report 2020/086, 2020. `https://eprint.iacr.org/2020/086`.

[31] Microsoft SEAL, 2020. `https://github.com/Microsoft/SEAL`.

[32] N. P. Smart and F. Vercauteren. Fully Homomorphic SIMD Operations. *Des. Codes Cryptography*, 71(1):5781, Apr. 2014.

A Unified View of BGV and BFV

The BGV and BFV schemes can be viewed as different modes of the same unified scheme using the methodology described in [5, 16]. The BGV encryption corresponds to the Least Significant Digit (LSD) message encoding, and the BFV encryption can be interpreted as the Most Significant Digit (MSD) message encoding. One can losslessly switch (without increasing the noise) between the two encodings as long as Q and t are coprime, using the techniques discussed in Appendix A of [5]. This unified scheme can support three modes based on the type of tensor product in the homomorphic multiplication. The first mode is LSD $\times$ LSD, i.e., it corresponds to the product of an encryption in the LSD encoding by an encryption in the LSD encoding, and represents the BGV scheme. The MSD $\times$ MSD mode directly corresponds to the BFV scheme. The last (intermediate) mode, where we deal with the LSD $\times$ MSD product, is a variant of BGV discussed in [16]. Note that both the LSD $\times$ LSD and LSD $\times$ MSD modes incur the BGV noise growth while the MSD $\times$ MSD mode has the BFV noise growth behavior. This unified view simplifies the analysis of noise growth and computational complexity as we can focus on the comparison of homomorphic multiplication and encryption/decryption procedures between BGV and BFV. All other operations incur essentially the same noise growth. This unified view also helps understand the differences between the schemes, e.g., the better noise growth in BGV for large plaintext moduli reported in [15].

B Key Switching

In this section we recall different variants of key switching used in the litterature and compare their efficiency in terms of noise growth and performance. Key-switching procedures can be applied equivalently to BGV or BFV ciphertexts. In the following, we detail them for BGV ciphertexts.

Let $\mathtt{ct}^A = (\boldsymbol{c}_0^A, \boldsymbol{c}_1^A)$ be a ciphertext encrypted modulo $Q \in \{Q_i\}_{i=0}^L$ under a public key $\mathtt{pk}_A$ whose associated secret key is $\mathtt{sk}_A = \boldsymbol{s}_A$, we have

$$\boldsymbol{c}_0^A + \boldsymbol{c}_1^A \cdot \boldsymbol{s}_A \equiv \boldsymbol{m} + t\boldsymbol{v} \bmod Q.$$

It is possible to transform $\mathtt{ct}_A$ into another cipertext $\mathtt{ct}_B$ which will decrypt under a secret key $\mathtt{sk}_B = \boldsymbol{s}_B$. The high level idea is to multiply $\boldsymbol{c}_1^A$ by an encryption of $\boldsymbol{s}_A$ under a public key associated to $\boldsymbol{s}_B$

$$\mathtt{ks}_{A\to B} = \left(\left[\boldsymbol{s}_A + \boldsymbol{a} \cdot \boldsymbol{s}_B + t\boldsymbol{e}\right]_Q, -\boldsymbol{a} \right) \in \mathcal{R}_Q^2$$

with $\boldsymbol{a} \in\!\leftarrow \mathcal{U}_Q$ and $\boldsymbol{e} \leftarrow \chi_{\mathtt{err}}$. Then by setting

$$\mathtt{ct}^B = \left(\left[\boldsymbol{c}_0^A + \boldsymbol{c}_1^A \cdot (\boldsymbol{s}_A + \boldsymbol{a} \cdot \boldsymbol{s}_B + t\boldsymbol{e})\right]_Q, \left[-\boldsymbol{c}_1^A \cdot \boldsymbol{a}\right]_Q \right) \in \mathcal{R}_Q^2,$$

we would have

$$\begin{aligned} \boldsymbol{c}_0^B + \boldsymbol{c}_1^B \cdot \boldsymbol{s}_B &\equiv \boldsymbol{c}_0^A + \boldsymbol{c}_1^A \cdot (\boldsymbol{s}_A + \boldsymbol{a} \cdot \boldsymbol{s}_B + t\boldsymbol{e}) - \boldsymbol{c}_1^A \cdot \boldsymbol{a} \cdot \boldsymbol{s}_B \\ &\equiv \boldsymbol{c}_0^A + \boldsymbol{c}_1^A \cdot \boldsymbol{s}_A + t\boldsymbol{c}_1^A \cdot \boldsymbol{e} \\ &\equiv \boldsymbol{m} + t(\boldsymbol{v} + \boldsymbol{c}_1^A \cdot \boldsymbol{e}) \bmod Q, \end{aligned}$$

which is exactly what we wanted. Unfortunately, this cannot be done directly this way because the added noise $\boldsymbol{c}_1^A \cdot \boldsymbol{e}$ would be too high: $\|\boldsymbol{c}_1^A \cdot \boldsymbol{e}\|_\infty \leq \delta_{\mathcal{R}} Q B_{\mathtt{err}}/2 > \lfloor (Q-t)/2t \rfloor$. Therefore one has to find ways to reduce the size of the product $\boldsymbol{c}_1^A \cdot \boldsymbol{e}$.

B.1 Different Variants of Key Switching

B.1.1 Brakerski-Vaikuntanathan

Since the problem of the size of $\boldsymbol{c}_1^A \cdot \boldsymbol{e}$ comes mainly from $\boldsymbol{c}_1^A$ whose coefficients can be of size $Q/2$, Brakerski and Vaikuntanathan proposed to decompose $\boldsymbol{c}_1^A$ into small coefficients [11]. This can be done by using the decomposition in radix base $\omega \ll Q$. Let $\ell_{\omega,Q} = \lfloor \log_\omega(Q) \rceil + 1$. We define the functions $\mathcal{D}_{\omega,Q}$ as the decomposition in radix base ω and $\mathcal{P}_{\omega,Q}$ that retrieves powers of ω lost within the decomposition process. So that for any $\boldsymbol{a} \in \mathcal{R}$

$$\mathcal{D}_{\omega,Q}(\boldsymbol{a}) = \left([\boldsymbol{a}]_\omega, \left[\left\lfloor \frac{\boldsymbol{a}}{\omega} \right\rceil \right]_\omega, \ldots, \left[\left\lfloor \frac{\boldsymbol{a}}{\omega^{\ell_{\omega,Q}-1}} \right\rceil \right]_\omega \right) \in \mathcal{R}_\omega^{\ell_{\omega,Q}}$$

$$\mathcal{P}_{\omega,Q}(\boldsymbol{a}) = \left([\boldsymbol{a}]_Q, [\boldsymbol{a}\omega]_Q, \ldots, [\boldsymbol{a}\omega^{\ell_{\omega,Q}-1}]_Q \right) \in \mathcal{R}_Q^{\ell_{\omega,Q}}$$

Lemma B.1 *For any* $(\boldsymbol{a}, \boldsymbol{b}) \in \mathcal{R}^2$, $\langle \mathcal{D}_{\omega,Q}(\boldsymbol{a}), \mathcal{P}_{\omega,Q}(\boldsymbol{b}) \rangle \equiv \boldsymbol{a} \cdot \boldsymbol{b} \bmod Q$.

Therefore if we use a key-switching key

$$\mathtt{ks}_{A\to B}^{\mathtt{BV}} = \left(\left[\mathcal{P}_{\omega,Q_L}(\boldsymbol{s}_A) + \overrightarrow{\boldsymbol{a}} \cdot \boldsymbol{s}_B + t\overrightarrow{\boldsymbol{e}} \right]_{Q_L}, -\overrightarrow{\boldsymbol{a}} \right) \in \mathcal{R}_{Q_L}^{\ell_{\omega,Q_L}} \times \mathcal{R}_{Q_L}^{\ell_{\omega,Q_L}}$$

with $\overrightarrow{\boldsymbol{a}} \in\leftarrow \mathcal{U}_Q^{\ell_{\omega,Q}}$ and $\overrightarrow{\boldsymbol{e}} \leftarrow \chi_{\mathtt{err}}^{\ell_{\omega,Q}}$, we can compute

$$\mathtt{ct}_B = \left(\left[\boldsymbol{c}_0^A + \left\langle \mathcal{D}_{\omega,Q}(\boldsymbol{c}_1^A), \mathtt{ks}_{A\to B,0}^{\mathtt{BV}} \right\rangle \right]_Q, \left[\left\langle \mathcal{D}_{\omega,Q}(\boldsymbol{c}_1^A), \mathtt{ks}_{A\to B,1}^{\mathtt{BV}} \right\rangle \right]_Q \right).$$

Thanks to the linearity of the inner product we obtain in this case

$$\begin{aligned}
\boldsymbol{c}_0^B + \boldsymbol{c}_1^B \cdot \boldsymbol{s}_B &\equiv \boldsymbol{c}_0^A + \left\langle \mathcal{D}_{\omega,Q}(\boldsymbol{c}_1^A), \mathtt{ks}_{A\to B,0}^{\mathtt{BV}} \right\rangle + \left\langle \mathcal{D}_{\omega,Q}(\boldsymbol{c}_1^A), \mathtt{ks}_{A\to B,1}^{\mathtt{BV}} \right\rangle \cdot \boldsymbol{s}_B \\
&\equiv \boldsymbol{c}_0^A + \left\langle \mathcal{D}_{\omega,Q}(\boldsymbol{c}_1^A), \mathcal{P}_{\omega,Q}(\boldsymbol{s}_A) \right\rangle + t \left\langle \mathcal{D}_{\omega,Q}(\boldsymbol{c}_1^A), \overrightarrow{\boldsymbol{e}} \right\rangle \\
&\equiv \boldsymbol{c}_0^A + \boldsymbol{c}_1^A \cdot \boldsymbol{s}_A + t \left\langle \mathcal{D}_{\omega,Q}(\boldsymbol{c}_1^A), \overrightarrow{\boldsymbol{e}} \right\rangle \\
&\equiv \boldsymbol{m} + t \left(\boldsymbol{v} + \left\langle \mathcal{D}_{\omega,Q}(\boldsymbol{c}_1^A), \overrightarrow{\boldsymbol{e}} \right\rangle \right) \bmod Q
\end{aligned}$$

with

$$\|\boldsymbol{v}_{\mathtt{BV}}\|_\infty = \left\| \left\langle \mathcal{D}_{\omega,Q}(\boldsymbol{c}_1^A), \overrightarrow{\boldsymbol{e}} \right\rangle \right\|_\infty \leq \sum_{i=0}^{\ell_{\omega,Q}-1} \left\| \left[\left\lfloor \frac{\boldsymbol{c}_1^A}{\omega^i} \right\rceil \right]_\omega \cdot \boldsymbol{e}_i \right\|_\infty \leq \frac{\ell_{\omega,Q} \delta_{\mathcal{R}} \omega B_{\mathtt{err}}}{2}.$$

B.1.2 Gentry-Halevi-Smart

Another way to reduce the size of the added noise $\boldsymbol{c}_1^A \cdot \boldsymbol{e}$ was proposed by Gentry, Halevi, and Smart in [22]. Their idea was to temporarily extend the size of Q with another modulus P and modify the key-switching key by shifting $\boldsymbol{s}_A$ of P

$$\mathtt{ks}_{A\to B}^{\mathtt{GHS}} = \left([P\boldsymbol{s}_A + \boldsymbol{a} \cdot \boldsymbol{s}_B + t\boldsymbol{e}]_{QP}, -\boldsymbol{a} \right) \in \mathcal{R}_{QP}^2.$$

Then one can perform the product with the key-switching key modulo QP and obtain:

$$\tilde{\mathtt{ct}}_B = \left(\left[\boldsymbol{c}_1^A \cdot (P\boldsymbol{s}_A + \boldsymbol{a} \cdot \boldsymbol{s}_B + t\boldsymbol{e}) \right]_{QP}, \left[-\boldsymbol{c}_1^A \cdot \boldsymbol{a} \right]_{QP} \right) \in \mathcal{R}_{QP}^2,$$

which satisfies

$$\begin{aligned}\tilde{\boldsymbol{c}}_0^B + \tilde{\boldsymbol{c}}_1^B \cdot \boldsymbol{s}_B &\equiv \boldsymbol{c}_1^A \cdot (P\boldsymbol{s}_A + \boldsymbol{a} \cdot \boldsymbol{s}_B + t\boldsymbol{e}) - \boldsymbol{c}_1^A \cdot \boldsymbol{a} \cdot \boldsymbol{s}_B \\ &\equiv P\boldsymbol{c}_1^A \cdot \boldsymbol{s}_A + t\boldsymbol{c}_1^A \cdot \boldsymbol{e} \bmod QP\end{aligned}$$

Therefore at this point we can scale everything down by P using modulus switching to get back $\boldsymbol{c}_1^A \cdot \boldsymbol{s}_A$ and reduce the error by a factor P, so that in the end

$$\mathtt{ct}_B = \left(\left[\boldsymbol{c}_0^A + \frac{\tilde{\boldsymbol{c}}_0^B + \boldsymbol{\delta}_0}{P} \right]_Q, \left[\frac{\tilde{\boldsymbol{c}}_1^B + \boldsymbol{\delta}_1}{P} \right]_Q \right),$$

with $\boldsymbol{\delta}_i = t[-t^{-1}\tilde{\boldsymbol{c}}_i^B]_P$, satisfies

$$\begin{aligned}\boldsymbol{c}_0^B + \boldsymbol{c}_1^B \cdot \boldsymbol{s}_B &\equiv \boldsymbol{c}_0^A + \frac{\boldsymbol{c}_1^A \cdot (P\boldsymbol{s}_A + \boldsymbol{a} \cdot \boldsymbol{s}_B + t\boldsymbol{e}) + \boldsymbol{\delta}_0}{P} + \frac{-\boldsymbol{c}_1^A \cdot \boldsymbol{a} + \boldsymbol{\delta}_1}{P} \cdot \boldsymbol{s}_B \\ &\equiv \boldsymbol{c}_0^A + \frac{\boldsymbol{c}_1^A \cdot (P\boldsymbol{s}_A + \boldsymbol{a} \cdot \boldsymbol{s}_B + t\boldsymbol{e}) + \boldsymbol{\delta}_0}{P} + \left(-\frac{\boldsymbol{c}_1^A \cdot \boldsymbol{a} + \boldsymbol{\delta}_1}{P} \right) \cdot \boldsymbol{s}_B \\ &\equiv \boldsymbol{c}_0^A + \boldsymbol{c}_1^A \cdot \boldsymbol{s}_A + t\frac{\boldsymbol{c}_1^A \cdot \boldsymbol{e}}{P} + \frac{\boldsymbol{\delta}_0 + \boldsymbol{\delta}_1 \cdot \boldsymbol{s}_B}{P} \\ &\equiv \boldsymbol{m} + t\left(\boldsymbol{v} + \frac{\boldsymbol{c}_1^A \cdot \boldsymbol{e}}{P} + \frac{\boldsymbol{\delta}_0 + \boldsymbol{\delta}_1 \cdot \boldsymbol{s}_B}{tP} \right) \bmod Q.\end{aligned}$$

with

$$\|\boldsymbol{v}_{\mathtt{GHS}}\|_\infty = \left\| \frac{\boldsymbol{c}_1^A \cdot \boldsymbol{e}}{P} + \frac{\boldsymbol{\delta}_0 + \boldsymbol{\delta}_1 \cdot \boldsymbol{s}_B}{tP} \right\|_\infty \leq \frac{\delta_{\mathcal{R}} Q B_{\mathtt{err}}}{2P} + \frac{1 + \delta_R B_{\mathtt{key}}}{2}.$$

Therefore by chosing $P \approx Q$ the noise added by the operation will be relatively close to modulus switching noise. However the drawback of this method is that the security of the scheme now depends on a Ring-LWE sample modulo $QP \approx Q^2$. Therefore to maintain the same level of security one must either divide the size of Q by 2, which implies to lose half the multiplicative depth, or double the ring dimension which will degrade the performance.

B.1.3 Hybrid

`BV` method overcomes the limitations of `GHS` technique since it does not require to halve the multiplicative depth. However the drawback of this method is that one has to perform a quadratic number of NTTs: $\ell_{\omega,Q_i}(i+1)$ at level i. This becomes very quickly the efficiency bottleneck of the scheme especially in the first levels. On the other hand, the `GHS` technique requires only $2(i+1)$ NTTs. In order to obtain a tradeoff between efficiency and noise growth, one can use a hybrid method which takes advantage of both previous methods [22].

Using this method, one starts by decomposing $\boldsymbol{c}_1^A$ as in `BV` but using a relatively large radix basis ω so that $\ell_{\omega,Q}$ remains quite small. Then to compensate for the noise growth caused by the size of the digits, the product with the key-switching key is performed in an extended basis PQ as in `GHS`. Therefore in this case the key-switching key will be:

$$\mathtt{ks}_{A\to B}^{\mathtt{Hybrid}} = \left(\left[P\mathcal{P}_{\omega,Q}(\boldsymbol{s}_A) + \overrightarrow{\boldsymbol{a}} \cdot \boldsymbol{s}_B + t\overrightarrow{\boldsymbol{e}} \right]_{PQ}, -\overrightarrow{\boldsymbol{a}} \right) \in \mathcal{R}_{PQ}^{\ell_{\omega,Q}} \times \mathcal{R}_{PQ}^{\ell_{\omega,Q}}.$$

We compute

$$\tilde{\mathtt{ct}}_B = \left(\left[\left\langle \mathcal{D}_{\omega,Q}(\boldsymbol{c}_1^A), \mathtt{ks}_{A\to B,0}^{\mathtt{Hybrid}} \right\rangle \right]_{PQ}, \left[\left\langle \mathcal{D}_{\omega,Q}(\boldsymbol{c}_1^A), \mathtt{ks}_{A\to B,1}^{\mathtt{Hybrid}} \right\rangle \right]_{PQ} \right) \in \mathcal{R}_{PQ}^2,$$

which satisfies

$$\tilde{\boldsymbol{c}}_0^B + \tilde{\boldsymbol{c}}_1^B \cdot \boldsymbol{s}_B \equiv P\boldsymbol{c}_1^A \cdot \boldsymbol{s}_A + t \left\langle \mathcal{D}_{\omega,Q}(\boldsymbol{c}_1^A), \overrightarrow{\boldsymbol{e}} \right\rangle \bmod PQ.$$

Then, as in GHS, we can scale down everything by P using modulus switching and sum the result to $\boldsymbol{c}_0^A$ to obtain

$$\mathtt{ct}_B = \left(\left[\boldsymbol{c}_0^A + \frac{\tilde{\boldsymbol{c}}_0^B + \boldsymbol{\delta}_0}{P} \right]_Q, \left[\frac{\tilde{\boldsymbol{c}}_1^B + \boldsymbol{\delta}_1}{P} \right]_Q \right),$$

with $\boldsymbol{\delta}_i = t[-t^{\tilde{-1}}\boldsymbol{c}_i^B]_P$ which satisfies

$$\boldsymbol{c}_0^B + \boldsymbol{c}_1^B \cdot \boldsymbol{s}_B \equiv \boldsymbol{m} + t \left(\boldsymbol{v} + \frac{\left\langle \mathcal{D}_{\omega,Q}(\boldsymbol{c}_1^A), \overrightarrow{\boldsymbol{e}} \right\rangle}{P} + \frac{\boldsymbol{\delta}_0 + \boldsymbol{\delta}_1 \cdot \boldsymbol{s}_B}{tP} \right) \bmod Q,$$

with

$$\left\| \boldsymbol{v}_{\mathtt{Hybrid}} \right\|_\infty = \left\| \frac{\left\langle \mathcal{D}_{\omega,Q}(\boldsymbol{c}_1^A), \overrightarrow{\boldsymbol{e}} \right\rangle}{P} + \frac{\boldsymbol{\delta}_0 + \boldsymbol{\delta}_1 \cdot \boldsymbol{s}_B}{tP} \right\|_\infty \leq \frac{\ell_{\omega,Q}\delta_{\mathcal{R}}\omega B_{\mathtt{err}}}{2P} + \frac{1 + \delta_{\mathcal{R}} B_{\mathtt{key}}}{2}$$

In this case we only need $P \approx \ell_{\omega,Q}\omega/2 \approx \log_Q(\omega)$, which means that we do not have to sacrifice half the multiplicative levels as in the original GHS technique. Futhermore the products can be done using $2\ell_{\omega,Q}(i+1)$ products with a relatively small $\ell_{\omega,Q}$ (between 3 and 5 in practice).

B.2 RNS Instantiation

In practice BGV and BFV are implemented using RNS, which does not allow to perform certain operations natively, e.g., the decomposition in radix-base ω. In this section we detail the RNS variants of the previous key-switching techniques. In this section we assume that $Q = Q_i = q_0 \cdots q_i$ is a product of $(i+1)$ small primes.

B.2.1 Brakerski-Vaikuntanathan

The BV technique can be adapted to RNS by decomposing the values according to each residue as done in [6]

$$\mathcal{D}_{Q_i}(\boldsymbol{a}) = \left(\left[\boldsymbol{a} \left(\frac{Q_i}{q_0} \right)^{-1} \right]_{q_0}, \ldots, \left[\boldsymbol{a} \left(\frac{Q_i}{q_i} \right)^{-1} \right]_{q_i} \right) \in \mathcal{R}^{i+1}$$

$$\mathcal{P}_{Q_i}(\boldsymbol{a}) = \left(\left[\boldsymbol{a} \frac{Q_i}{q_0} \right]_{Q_i}, \ldots, \left[\boldsymbol{a} \frac{Q_i}{q_i} \right]_{Q_i} \right) \in \mathcal{R}_{Q_i}^{i+1}$$

This method was later improved by Halevi, Polyakov, and Shoup [23] who noticed that one could move the $[(Q_i/q_j)^{-1}]_{q_i}$ factors to $\mathcal{P}_{Q_i}$ saving therefore $(i+1)$ multiplications when computing $\mathcal{D}_{Q_i}(\boldsymbol{c}_1^A)$.

Lemma B.2 *For any $(\boldsymbol{a}, \boldsymbol{b}) \in \mathcal{R}^2$, $\langle \mathcal{D}_{Q_i}(\boldsymbol{a}), \mathcal{P}_{Q_i}(\boldsymbol{b}) \rangle \equiv \boldsymbol{a} \cdot \boldsymbol{b} \bmod Q_i$.*

So, by denoting $\tilde{q}_i = \max_{0\leq j\leq i}\{q_j\}$ in this case the noise added by the key-switching is bounded by:

$$\|\boldsymbol{v}_{\texttt{RNS-BV}}\|_\infty \leq \frac{(i+1)\delta_{\mathcal{R}}\tilde{q}_i B_{\texttt{err}}}{2}.$$

Note that one can add a second level of decomposition by decomposing each digits (which fits on a machine word) in radix base $\omega < \tilde{q}_i$. In this case the noise will be bounded by

$$\|\boldsymbol{v}_{\texttt{RNS-BV}}\|_\infty \leq \frac{\ell_{\omega,\tilde{q}_i}(i+1)\delta_{\mathcal{R}}\omega B_{\texttt{err}}}{2}$$

with $\ell_{\omega,\tilde{q}_i} = \lfloor \log_\omega(\tilde{q}_i)\rfloor + 1$. However the computational cost is now dominated by the $\ell_{\omega,\tilde{q}}(i+1)^2$ NTTs instead of $(i+1)^2$.

B.2.2 Gentry-Halevi-Smart

To take advantage of the RNS representation, P is also chosen as a product of small prime moduli $P = p_1\cdots p_k$, such that P and Q are coprime so that one can perform the division by P modulo Q. In original `GHS` one has to temporarily extend the size of the modulus Q to perform the product with the key-switching key. In order to avoid a costly CRT reconstruction, this is done by a fast base extension from the basis $\mathcal{Q}_i = \{q_0,\ldots,q_i\}$ to the basis $\mathcal{P} = \{p_1,\ldots,p_k\}$ using the classical CRT technique:

$$\forall \boldsymbol{a}\in\mathcal{R}_{Q_i},\ \texttt{FastBaseExtension}(\boldsymbol{a},\mathcal{Q}_i,\mathcal{P}) = \left(\sum_{j=0}^{i}\left[\boldsymbol{a}\left(\frac{Q_i}{q_j}\right)^{-1}\right]_{q_j}\frac{Q_i}{q_j} \bmod p_l\right)_{l=1}^{k}$$

Note that this conversion is not exact and one gets $[\boldsymbol{a}]_{Q_i} + \boldsymbol{u}Q_i$ in base $\mathcal{P}$ with $\|\boldsymbol{u}\|_\infty \leq i/2$ instead of exactly $[\boldsymbol{a}]_{Q_i}$. So overall one has the residues of $[\boldsymbol{a}]_{Q_i} + \boldsymbol{u}Q_i$ in the basis $\mathcal{P}\cup\mathcal{Q}_i$. From there one can compute

$$\tilde{\texttt{ct}}_B = \left(\left[(\boldsymbol{c}_1^A + Q_i\boldsymbol{u})\cdot(P\boldsymbol{s}_A + \boldsymbol{a}\cdot\boldsymbol{s}_B + t\boldsymbol{e})\right]_{PQ_i}, \left[-(\boldsymbol{c}_1^A + Q_i\boldsymbol{u})\cdot\boldsymbol{a}\right]_{PQ_i}\right) \in \mathcal{R}_{PQ_i}^2$$

which satisfies

$$\tilde{\boldsymbol{c}}_0^B + \tilde{\boldsymbol{c}}_1^B\cdot\boldsymbol{s}_B \equiv P\boldsymbol{c}_1^A\cdot\boldsymbol{s}_A + t(\boldsymbol{c}_1^A\cdot\boldsymbol{e} + Q_i\boldsymbol{u}\cdot\boldsymbol{e}) \bmod PQ_i$$

Then to perform the modulus switching, one has to first get $\boldsymbol{\delta} = t[-t^{-1}\tilde{\texttt{ct}}_B]_P$ in base $\mathcal{Q}_i$. This is done by converting back $\tilde{\texttt{ct}}_B$ from $\mathcal{P}$ to $\mathcal{Q}_i$, once again one can use `FastBaseExtension`. Hence one obtains $\tilde{\boldsymbol{\delta}} = t([-t^{-1}\tilde{\texttt{ct}}_B]_P + P\boldsymbol{u}')$ with $\|\boldsymbol{u}'\|_\infty \leq (k-1)/2$ in the basis $\mathcal{Q}_i$. Therefore in the end one obtains

$$\texttt{ct}_B = \left(\left[\boldsymbol{c}_0^A + (\tilde{\boldsymbol{c}}_0^B + \tilde{\boldsymbol{\delta}}_0)/P\right]_{Q_i}, \left[(\tilde{\boldsymbol{c}}_1^B + \tilde{\boldsymbol{\delta}}_1)/P\right]_{Q_i}\right)$$

which satisfies

$$\boldsymbol{c}_0^B + \boldsymbol{c}_1^B\cdot\boldsymbol{s}_B \equiv [\boldsymbol{m}]_t + t\left(\boldsymbol{v} + \frac{(\boldsymbol{c}_1^A + Q_i\boldsymbol{u})\cdot\boldsymbol{e}}{P} + \frac{\boldsymbol{\delta}_0 + \boldsymbol{\delta}_1\cdot\boldsymbol{s}_B}{tP} + \boldsymbol{u}_0' + \boldsymbol{u}_1'\cdot\boldsymbol{s}_B\right)$$

with

$$\|\boldsymbol{v}_{\texttt{RNS-GHS}}\|_\infty \leq \frac{\delta_{\mathcal{R}}Q_i(i+1)B_{err}}{2P} + \frac{k+\delta_{\mathcal{R}}kB_{key}}{2}.$$

To summarize, the approximate RNS techniques increase the first part of the worst-case bound by a factor $i+1$ and the second one by a factor k. This can be softened by using techniques from BEHZ or removed with techniques from HPS but the noise growth is quite small in practice and thus barely noticed.

B.2.3 Hybrid

For `Hybrid` key-switching in RNS we use the same methodology and tools as for `BV` and `GHS` techniques. We start by decomposing $\boldsymbol{c}_1^A$ in d_{num} digits $\tilde{Q}_0, \dots \tilde{Q}_{d_{\text{num}}-1}$, where each digit is the product of α moduli $\tilde{Q}_j = q_{\alpha j} \cdots q_{\alpha(j+1)-1}$ with $\alpha = \lceil (L+1)/d_{\text{num}} \rceil$. Therefore the key-switching key will be:

$$\mathtt{ks}_{A\to B}^{\texttt{RNS-Hybrid}} = ([P\tilde{\mathcal{P}}_{Q_i}(\boldsymbol{s}_A) + \overrightarrow{\boldsymbol{a}} \cdot \boldsymbol{s}_B + t\overrightarrow{\boldsymbol{e}}]_{PQ_i}, -\overrightarrow{\boldsymbol{a}}) \in \mathcal{R}_{PQ_i}^{d_{\text{num}}} \times \mathcal{R}_{PQ_i}^{d_{\text{num}}},$$

with

$$\tilde{\mathcal{P}}_Q(\boldsymbol{s}_B) = \left(\left[\boldsymbol{s}_B \frac{Q}{\tilde{Q}_0} \right]_Q, \dots, \left[\boldsymbol{s}_B \frac{Q_i}{\tilde{Q}_{d_{\text{num}}-1}} \right]_Q \right) \in \mathcal{R}_Q^{d_{\text{num}}}.$$

Then each digit is extended from $\tilde{\mathcal{Q}}_j = \{q_{\alpha j}, \dots, q_{\alpha(j+1)-1}\}$ to $\mathcal{P} \cup \mathcal{Q}_i$ which causes an overflow $\boldsymbol{u}_j \tilde{Q}_j$, where $\|\boldsymbol{u}_j\|_\infty \leq (\alpha-1)/2$. As in GHS, the second source of errors comes from the conversion from $\mathcal{P}$ to $\mathcal{Q}_i$ to perform the modulus switching. In this case the overflow will remain the same as in GHS $\|\boldsymbol{u}'\| \leq (k-1)/2$.

Therefore by denoting $\tilde{Q} = \max_{0\leq j\leq d_{\text{num}}-1}\{\tilde{Q}_j\}$ the noise added by the hybrid key-switching in RNS is bounded by

$$\|\boldsymbol{v}_{\text{RNS-Hybrid}}\|_\infty \leq \frac{\alpha d_{\text{num}} \delta_{\mathcal{R}} \tilde{Q} B_{\mathtt{err}}}{2P} + \frac{k + k\delta_{\mathcal{R}} B_{\mathtt{key}}}{2}$$

Thus overall one can take $P \approx \tilde{Q}$ i.e. $k \approx \alpha$.

Remark B.3 *Note that for `BGV` while the moduli q_i must be chosen between 20 and 60 bits depending on the targeted application, the moduli p_i can be chosen of maximal size ≈ 60 bits which should reduce k and hence the computational complexity.*

B.3 Complexity of Key Switching Methods in RNS and Key Sizes.

In this section we need to distinguish whether we are using `BGV` or `BFV` since in `BGV` the ciphertexts are kept in NTT format during homomorphic multiplication while the `BFV` ciphertexts are kept in coefficient representation.

Therefore we assume that for `BFV`, the ciphertext modulus Q is made of ℓ moduli, the key-switching modulus P is made of k moduli and the digits fit on α moduli. For `BGV`, we assume that the ciphertext modulus Q is made of ℓ' moduli, the key-switching modulus P is made of k' moduli and the digits fit on α' moduli.

The costs are given in number of NTTs (we assume that NTTs and invNTTs have the same cost) and integer multiplications (Mult). Last we recall that the maximal number of moduli L, for GHS and Hybrid key-switching techniques, is smaller than for `BV` ($L/2$ and $L-\alpha$ respectively).

Brakerski-Vaikuntanathan

- put $\boldsymbol{c}_1$ in coefficient representation: ℓ' invNTTs (not required for `BFV`);
- decompose $\boldsymbol{c}_1$ in small digits (for free with HPS trick);
- put $\mathcal{D}(\boldsymbol{c}_1)$ in NTT form: $\ell_{\omega,\tilde{q}}\ell^2$ NTTs for `BFV` and $\ell_{\omega,\tilde{q}}\ell'^2$ NTTs for `BGV`;
- perform the product with the evaluation key: $2n\ell_{\omega,\tilde{q}}\ell^2$ Mults for `BFV` and $2n\ell_{\omega,\tilde{q}}\ell'^2$ Mults for `BGV`;
- put the result back in coefficient form 2ℓ `invNTT`s (not required for `BGV`)
- sum the results to $\boldsymbol{c}_0$;

Total:

`BGV`: $\ell'(\ell_{\omega,\tilde{q}}\ell' + 1)$ NTTs and $2n\ell_{\omega,\tilde{q}}\ell'^2$ Mults.

`BFV`: $\ell(\ell_{\omega,\tilde{q}}\ell + 2)$ NTTs and $2n\ell_{\omega,\tilde{q}}\ell^2$ Mults.

GHS

- put $\boldsymbol{c}_1$ in coefficient representation: ℓ' invNTTs (not required for `BFV`);
- extend $\boldsymbol{c}_1$ from base $\mathcal{Q}$ to base $\mathcal{P}$: $n\ell(k+1)$ Mults for `BFV` and $n\ell'(k+1)$ Mults for `BGV`;
- put $\boldsymbol{c}_1$ in NTT form: $\ell + k$ NTTs for `BFV` and k' NTTs for `BGV`;
- perform the product with the evaluation key: $2n(\ell + k)$ Mults for `BFV` and $2n(\ell' + k')$ Mults for `BGV`;
- inverse the NTT in base $\mathcal{P}$: $2k$ invNTTs for `BFV` and $2k'$ invNTTs for `BGV`;
- inverse the NTT in base $\mathcal{Q}$: 2ℓ invNTTs for `BFV` (not required for `BGV`);
- multiply the result by $-t^{-1} \bmod P$ for the computation of $\boldsymbol{\delta}$: $2nk'$ Mults (not required for `BFV`);
- convert back from base $\mathcal{P}$ to base $\mathcal{Q}$: $2nk(\ell+1)$ Mults for `BFV` and $2nk'(\ell'+1)$ Mults for `BGV`;
- multiply by t for the computation of $\boldsymbol{\delta}$ (no modular reduction): $2n\ell'$ Mults (not required for `BFV`);
- put the results in NTT format: $2\ell'$ NTTs (not required for `BFV`);
- substract the result to $\boldsymbol{c}_0$ and multiply the results by $P^{-1} \bmod Q$: $2n\ell$ Mults for `BFV` and $2n\ell'$ Mults for `BGV`.

Total:

`BGV`: $3(\ell' + k')$ NTTs and $n(3\ell'k' + 7\ell' + 6k')$ Mults.

`BFV`: $3(\ell + k)$ NTTs and $n(3\ell k + 5\ell + 4k)$ Mults.

Given that for GHS key-switching one must choose $k \approx \ell$ (resp. $k' \approx \ell'$) the total cost is given by:

$$\texttt{BGV}\text{: } 6\ell' \text{ NTTs and } n\ell(3\ell' + 13) \text{ Mults.}$$

$$\texttt{BFV}\text{: } 6\ell \text{ NTTs and } n\ell(3\ell + 9) \text{ Mults.}$$

Hybrid

- put $\boldsymbol{c}_1$ in coefficient representation: ℓ' invNTTs (not required for `BFV`);
- decompose $\boldsymbol{c}_1$ in small digits : for free with HPS trick;
- extend the digits of $\boldsymbol{c}_1$ from base $\mathcal{Q}_i$ to base $\mathcal{P} \cup \mathcal{Q}$: $nd_{\text{num}}\alpha(\ell - \alpha + k + 1) = n\ell(\ell - \alpha + k + 1)$ Mults for `BFV` and $nd_{\text{num}}\alpha'(\ell' - \alpha' + k' + 1) = n\ell'(\ell' - \alpha' + k' + 1)$ Mults for `BGV`;
- put the digits of $\boldsymbol{c}_1$ in NTT form: $d_{\text{num}}(\ell + k)$ NTTs for `BFV` and $d_{\text{num}}(\ell' - \alpha' + k')$ NTTs for `BGV`;
- perform the product with the evaluation key: $2nd_{\text{num}}(\ell + k)$ Mults for `BFV` and $2nd_{\text{num}}(\ell' + k')$ Mults for `BGV`;
- inverse the NTT in base $\mathcal{P}$: $2k$ invNTTs for `BFV` and $2k'$ invNTTs for `BGV`;
- inverse the NTT in base $\mathcal{Q}$: 2ℓ invNTT (not required for `BGV`);
- multiply the result by $-t^{-1} \bmod P$ for the computation of $\boldsymbol{\delta}$: $2nk'$ Mults (not required for `BFV`);
- convert back from base $\mathcal{P}$ to base $\mathcal{Q}$: $2nk(\ell + 1)$ Mults for `BFV` and $2nk'(\ell' + 1)$ Mults for `BGV`;
- multiply by t for the computation of $\boldsymbol{\delta}$ (no modular reduction): $2n\ell'$ Mults (not required for `BFV`);
- put the results in NTT format: $2\ell'$ NTTs (not required for `BFV`);
- substract the result to $\boldsymbol{c}_0$ and multiply the results by $P^{-1} \bmod Q$: $2n\ell$ Mults for `BFV` and $2n\ell'$ Mults for `BGV`.

Total:

$$\texttt{BGV}\text{: } 3\ell' + 2k' + d_{\text{num}}(\ell' + k' - \alpha') \text{ NTTs and } n(\ell'^2 + (3k' - \alpha' + 2d_{\text{num}} + 5)\ell' + +2d_{\text{num}}k' + 4k') \text{ Mults.}$$

$$\texttt{BFV}\text{: } 2(\ell + k) + d_{\texttt{num}}(\ell + k) \text{ NTTs and } n(\ell^2 + (3k - \alpha + 2d_{\text{num}} + 3)\ell + 2d_{\text{num}}k + 2k) \text{ Mults.}$$

Given that for hybrid key-switching one must choose $k \approx \alpha$ (resp. $k' \approx \alpha'$) and that $d_{\text{num}}\alpha \approx \ell$ (resp. $d_{\text{num}}\alpha' \approx \ell'$) we have

	BGV		BFV	
	# NTTs	# Mults	# NTTs	# Mults
BV	$\ell'(\ell'\ell_{\omega,\tilde{q}}+1)$	$2n\ell_{\omega,\tilde{q}}\ell'^2$	$\ell(\ell\ell_{\omega,\tilde{q}}+2)$	$2n\ell_{\omega,\tilde{q}}\ell^2$
GHS	$6\ell'$	$n\ell'(3\ell'+13)$	6ℓ	$n\ell(3\ell+9)$
Hybrid	$3\ell'+\alpha'(d_{\text{num}}^2+2)$	$n((\ell'+2\alpha'+2d_{\text{num}}+7)\ell'+4\alpha')$	$2\ell+\alpha(d_{\text{num}}^2+3)$	$n((\ell+2\alpha+2d_{\text{num}}+5)\ell+2\alpha)$

Table 4: Complexities of different key-switching methods for BGV and BFV.

BV	GHS	Hybrid
$2N\ell_{\omega,Q}\log_2 Q$	$4N\log_2 Q$	$4N\log_2 Q$

Table 5: Size in bits for different key-switching methods

BGV: $3\ell'+\alpha'(d_{\text{num}}^2+2)$ NTTs and $n(\ell'^2+(2\alpha'+2d_{\text{num}}+7)\ell'+4\alpha')$ Mults.

BFV: $2\ell+\alpha(d_{\text{num}}^2+3)$ NTTs and $n(\ell^2+(2\alpha+2d_{\text{num}}+5)\ell+2\alpha)$ Mults.

There is another important parameter that needs to be considered if one wants to have an efficient implementation of the schemes: the key-switching key size. For simplicity, we assume here that the ciphertext modulus Q has the same size for BGV and BFV.

In BV, the key-switching is made of two vectors of $\ell_{\omega,Q}$ elements of $\mathcal{R}_Q$. Hence it is usually larger than for GHS, where it is made of two elements of $\mathcal{R}_{PQ}$ with $P\approx Q$. In the case of hybrid key switching, we have two vectors of d_{num} elements of $\mathcal{R}_{Q_iP}$ with $\log_2 Q_i\approx\log_2 Q/d_{\text{num}}$ and $Q_i\approx P$. Therefore, one obtains roughly the same size as for the GHS key.

For the sake of completeness, we provide in Table 5 a comparison of different key-switching key sizes in bits.

C Noise Estimates for BGV Multiplication

C.1 Setting the Optimal Constant Noise Level

Let us approximate the noise expression for BGV (here we drop negligible terms)

$$\frac{Q_i}{Q_{i+1}}\left((n'_{\text{add}}+1)\frac{\delta_{\mathcal{R}}t}{2}(2\left\|\boldsymbol{v}_{\mathtt{c}}\right\|_\infty^2+2\left\|\boldsymbol{v}_{\mathtt{c}}\right\|_\infty+1)+(n'_{\text{ks}}+1)\left\|\boldsymbol{v}_{\mathtt{ks}}\right\|_\infty\right)+\frac{1+\delta_{\mathcal{R}}B_{\mathtt{key}}}{2}\leq\left\|\boldsymbol{v}_{\mathtt{c}}\right\|_\infty$$

as

$$\frac{Q_i}{Q_{i+1}}\left(A\cdot c^2\left\|\boldsymbol{v}_{\mathtt{ms}}\right\|_\infty^2+B\left\|\boldsymbol{v}_{\mathtt{ks}}\right\|_\infty\right)+\left\|\boldsymbol{v}_{\mathtt{ms}}\right\|_\infty\leq c\left\|\boldsymbol{v}_{\mathtt{ms}}\right\|_\infty,$$

where $\left\|\boldsymbol{v}_{\mathtt{ms}}\right\|_\infty=\frac{1+\delta_{\mathcal{R}}B_{\mathtt{key}}}{2}$, $A=\delta_{\mathcal{R}}t\cdot(n'_{\text{add}}+1)$, $B=(n'_{\text{ks}}+1)$, $\left\|\boldsymbol{v}_{\mathtt{c}}\right\|_\infty=c\left\|\boldsymbol{v}_{\mathtt{ms}}\right\|_\infty$, and c is the sought parameter corresponding to the minimum value of $\frac{Q_{i+1}}{Q_i}$.

Rearranging the terms, we obtain

$$\frac{Q_{i+1}}{Q_i}>\frac{A\cdot c^2\left\|\boldsymbol{v}_{\mathtt{ms}}\right\|_\infty^2+B\left\|\boldsymbol{v}_{\mathtt{ks}}\right\|_\infty}{(c-1)\left\|\boldsymbol{v}_{\mathtt{ms}}\right\|_\infty}=f(c).$$

Taking a derivative of the right-hand side w.r.t. c yields

$$f'(c) = \frac{c \cdot (c-2) \cdot A \left\|\boldsymbol{v}_{\mathtt{ms}}\right\|_\infty^2 - B \left\|\boldsymbol{v}_{\mathtt{ks}}\right\|_\infty}{\left\|\boldsymbol{v}_{\mathtt{ms}}\right\|_\infty \cdot (c-1)^2}.$$

Solving the quadratic equation in the numerator for c gives us the following expression for optimal c, corresponding to the smallest value of $\frac{Q_{i+1}}{Q_i}$:

$$c = 1 + \sqrt{1 + \frac{B \left\|\boldsymbol{v}_{\mathtt{ks}}\right\|_\infty}{A \left\|\boldsymbol{v}_{\mathtt{ms}}\right\|_\infty^2}}.$$

If $\frac{B\|\boldsymbol{v}_{\mathtt{ks}}\|_\infty}{A\|\boldsymbol{v}_{\mathtt{ms}}\|_\infty^2} \ll 1$, which corresponds to many practical cases where the plaintext modulus is relatively large (see Appendix C.2), then $c \approx 2$.

C.2 Effect of Key-Switching Noise

We show that the hybrid-key-switching noise is much smaller than the multiplication noise for larger plaintext moduli in the noise expression for `BGV`:

$$\frac{Q_i}{Q_{i+1}} \left((n'_{\text{add}} + 1) \frac{\delta_{\mathcal{R}} t}{2} (2 \left\|\boldsymbol{v}_{\mathtt{c}}\right\|_\infty^2 + 2 \left\|\boldsymbol{v}_{\mathtt{c}}\right\|_\infty + 1) + (n'_{\text{ks}} + 1) \left\|\boldsymbol{v}_{\mathtt{ks}}\right\|_\infty \right) + \frac{1 + \delta_{\mathcal{R}} B_{\mathtt{key}}}{2} \leq \left\|\boldsymbol{v}_{\mathtt{c}}\right\|_\infty.$$

Note that $\left\|\boldsymbol{v}_{\mathtt{c}}\right\|_\infty \approx 1 + \delta_{\mathcal{R}} B_{\mathtt{key}}$ and $\left\|\boldsymbol{v}_{\text{ks}}\right\|_\infty \leq \frac{\alpha d_{\text{num}} \delta_{\mathcal{R}} \tilde{Q} B_{\mathtt{err}}}{2P} + \frac{k_d + k_d \delta_{\mathcal{R}} B_{\mathtt{key}}}{2}$.
As $P \approx \tilde{Q}$, $k \approx \alpha d_{\text{num}}$, $B_{\mathtt{key}} = 1$, and $B_{\mathtt{err}} = 6\sigma$, we have

$$\left\|\boldsymbol{v}_{\text{ks}}\right\|_\infty \approx \frac{k \delta_{\mathcal{R}} B_{\mathtt{err}}}{2} + \frac{k_d + k_d \delta_{\mathcal{R}}}{2} \approx \frac{k \delta_{\mathcal{R}} B_{\mathtt{err}}}{2} = 3k \delta_{\mathcal{R}} \sigma.$$

For the multiplication noise, we have

$$\left\|\boldsymbol{v}_{\text{m}}\right\|_\infty \approx \frac{\delta_{\mathcal{R}} t}{2} (2 \left\|\boldsymbol{v}_{\mathtt{c}}\right\|_\infty) \approx \delta_{\mathcal{R}} t \left\|\boldsymbol{v}_{\mathtt{c}}\right\|_\infty \approx \delta_{\mathcal{R}}^2 t.$$

Hence we have

$$\frac{\left\|\boldsymbol{v}_{\text{m}}\right\|_\infty}{\left\|\boldsymbol{v}_{\text{ks}}\right\|_\infty} \approx \frac{\delta_{\mathcal{R}}^2 t}{3k \delta_{\mathcal{R}} \sigma} \approx \frac{\delta_{\mathcal{R}} t}{9.6k} \approx \frac{2\sqrt{N} t}{9.6k},$$

where we use that $\sigma \approx 3.2$ and $\delta_R \approx 2\sqrt{N}$. Note that $N \geq 2^{10}$ (often higher).

If t is relatively large, e.g., $2^{16}+1$ often used in scenarios of CRT packing, then the multiplication noise dominates the key-switching noise. If t is small, e.g., $t = 2$, the key-switching noise makes a significant (non-negligible) contribution to the total noise related to multiplication.

D Noise Estimates for Leveled BFV Multiplication

Let us rewrite expression (12):

$$\|\boldsymbol{v}\|_\infty > 8\frac{Q\delta_{\mathcal{R}}}{Q_\ell} \text{ and } \|\boldsymbol{v}\|'_\infty > 8\frac{Q\delta_{\mathcal{R}}}{Q_\ell}.$$

The main challenge here is that we need precise estimates of $\|\boldsymbol{v}\|_\infty$ and $\|\boldsymbol{v}\|'_\infty$ to make sure we do not drop more levels than we need to. If we use our worst-case bounds $\|\boldsymbol{v}\|_\infty^{\texttt{ws}}$ and $\|\boldsymbol{v}\|_\infty^{\prime\ \texttt{ws}}$, which may be significantly higher than the actual noise, then we may loose some extra bits to noise, approaching the worst-case bound. While this would not necessarily lead to decryption errors (as we would still not exceed the worst-case bound), we would observe somewhat higher actual noise growth than in `BFV` without leveled multiplication.

This can be mitigated in practice by computing $\|\boldsymbol{v}\|_\infty = C \cdot \|\boldsymbol{v}\|_\infty^{\texttt{ws}}$ and $\|\boldsymbol{v}\|'_\infty = C \cdot \|\boldsymbol{v}\|_\infty^{\prime\ \texttt{ws}}$, where C is the "cushion" accounting for the maximum deviation of the worst-case bounds from the actual noise. For our implementation, we heuristically found $C = 2^{30}$ adequate up to at least depth-10 computations. A higher value of C may of be used if deeper computations need to be supported, e.g., $C = 2^{60}$ is sufficient for depth-20 computations.

E Modulus Switching between Arbitrary RNS Bases

Consider the modulus switching $\left[\left\lceil \frac{P}{Q} \cdot [x]_Q \right\rfloor\right]_P$, where P and Q are arbitrary co-prime moduli. Observe that

$$\left\lceil \frac{P}{Q} \cdot [x]_Q \right\rfloor = \frac{P \cdot [x]_Q - [Px]_Q}{Q}.$$

This implies that

$$\left[\left\lceil \frac{P}{Q} \cdot [x]_Q \right\rfloor\right]_P = \left[(P \cdot [x]_Q - [Px]_Q) \cdot Q^{-1}\right]_P = \left[-[Px]_Q \cdot Q^{-1}\right]_P.$$

Using the CRT composition formula, we can write $[[Px]_Q]_P$ as

$$[Px]_Q = \sum_{i=1}^{k} [Px]_{q_i} \cdot \left[\left(\frac{Q}{q_i}\right)^{-1}\right]_{q_i} \cdot \frac{Q}{q_i} - u \cdot Q.$$

Then we have

$$\begin{aligned}\left[\left\lceil \frac{P}{Q} \cdot [x]_Q \right\rfloor\right]_{p_j} &= \left[-Q^{-1} \cdot \left(\textstyle\sum_{i=1}^{k} [Px]_{q_i} \cdot \left[\left(\frac{Q}{q_i}\right)^{-1}\right]_{q_i} \cdot \frac{Q}{q_i} + u \cdot Q\right)\right]_{p_j} \\ &= \left[-\textstyle\sum_{i=1}^{k} [Px]_{q_i} \cdot \left[\left(\frac{Q}{q_i}\right)^{-1}\right]_{q_i} \cdot {q_i}^{-1} + u\right]_{p_j}.\end{aligned}$$

Note that the value of $\|u\|_\infty$ is at most $k/2$. In other words, if we use `FastBaseExtension` given by Equation (5), the extra noise introduced by scaling will be $\log(k/2)$ bits. This noise can be included in the noise estimate used in the leveled `BFV` multiplication. Alternatively, u can be computed exactly using the floating-point expression (6).

F Inner Product with Lazy Scaling

The algorithm for inner product computation with *lazy scaling* is depicted in Algorithm 4. We apply tensoring to pairs of ciphertexts, then add the results $\pmod{\mathcal{R}_{QP}}$, and finally perform the scaling only once. The part $\sum_i \boldsymbol{v}_{\mathtt{tensor}_i}$ of the resulting noise is unchanged, but as we perform scaling only once, we both reduce the complexity and noise coming from the scaling subroutine. The part $\frac{QP}{t}(\sum_i \boldsymbol{k}_{\mathtt{tensor}_i} + \boldsymbol{r}_{\mathtt{ip}})$, where $\boldsymbol{r}_{\mathtt{ip}}$ comes from $\sum_i [\boldsymbol{x}_i \boldsymbol{y}_i]_t = [\sum_i \boldsymbol{x}_i \boldsymbol{y}_i]_t + t\boldsymbol{r}_{\mathtt{ip}}$, disappears, as was explained in Section 3.2.

Note that we can also apply lazy scaling to the original `BFV` multiplication algorithm, but in this case we have to control the part $\frac{Q^2}{t}(\sum_i \boldsymbol{k}_{\mathtt{tensor}_i} + \boldsymbol{r}_{\mathtt{ip}})$, so that it does not overflow $\pmod{\mathcal{R}_{QP}}$. This could require increasing the number of primes in the RNS basis for P, reducing the benefit of this optimization.

Algorithm 4 Inner Product with Lazy Scaling

procedure InnerProdLazy($\mathtt{ct}_{x_i}, \mathtt{ct}_{y_i}$, $i = 1, \ldots, n$)
  for $i = 1, \ldots, n$ **do**
    $\mathtt{ct}_{x_i y_i} = \mathtt{Tensor}(\mathtt{Expand}(\mathtt{ct}_{x_i}, \mathtt{ct}_{y_i})) \in \mathcal{R}^3_{QP}$
      ▷ $\mathtt{ct}_{\mathtt{tensor}_i}(\boldsymbol{s}) = \frac{QP}{t^2}[\boldsymbol{x}_i \boldsymbol{y}_i]_t + \boldsymbol{v}_{\mathtt{tensor}_i} + \frac{QP}{t}\boldsymbol{k}_{\mathtt{tensor}_i} \pmod{\mathcal{R}_{QP}}$
    $\boldsymbol{c}_{\mathtt{ip}} = \boldsymbol{c}_{\mathtt{ip}} + \boldsymbol{c}_{\mathtt{tensor}_i} \in \mathcal{R}^3_{QP}$
      ▷ $\mathtt{ct}_{\mathtt{ip}}(\boldsymbol{s}) = \frac{QP}{t^2}[\sum_i \boldsymbol{x}_i \boldsymbol{y}_i]_t + \sum_i \boldsymbol{v}_{\mathtt{tensor}_i} + \frac{QP}{t}(\sum_i \boldsymbol{k}_{\mathtt{tensor}_i} + \boldsymbol{r}_{\mathtt{ip}}) \pmod{\mathcal{R}_{QP}}$
  `ScaleDown`: $\hat{\mathtt{ct}}_{\mathtt{ip}} = \left\lceil \frac{t}{P}\mathtt{ct}_{\mathtt{ip}} \right\rfloor \in \mathcal{R}^3_Q$
    ▷ $\hat{\mathtt{ct}}_{\mathtt{ip}}(\boldsymbol{s}) = \frac{Q}{t}[\sum_i \boldsymbol{x}_i \boldsymbol{y}_i]_t + \sum_i \boldsymbol{v}_{\mathtt{tensor}_i} + \boldsymbol{v}_{\mathtt{r}} \pmod{\mathcal{R}_Q}$

G Additional Experimental Results

G.1 Binary Tree Multiplicaton

Table 6: Comparison of noise growth and runtimes of BFV and BGV variants for a benchmark computation $\prod_{i=1}^{2^k} x_i$ at different plaintext moduli. Hybrid key switching with d_{num} digits, and $\lambda \geq 128$. Here, e denotes the current noise magnitude, $\log Q$, the BFV ciphertext modulus, and $\log Q_L$, the equivalent ciphertext modulus in BGV without the last CRT modulus q_{L+1}.

			Original BFV							Our BFV							Our BGV				
			params			BEHZ		HPS		params			BFV-NEW		BFV-NEW-LVL		params			BGV-NEW	
t	d_{num}	k	$\log N$	$\log q_i$	$\log Q$	$\log e$	time(s)	$\log e$	time(s)	$\log N$	$\log q_i$	$\log Q$	$\log e$	time(s)	$\log e$	time(s)	$\log N$	$\log q_i$	$\log Q$	$\log e$	time(s)
2	3	1	12	32	32	22	0.003	22	0.003	12	30	30	18	0.002	18	0.002	12	19	34	25	0.002
		2	12	47	47	34	0.009	34	0.008	12	43	43	30	0.006	31	0.006	13	20	56	25	0.018
		3	13	32	64	50	0.079	49	0.07	13	30	60	47	0.058	47	0.059	13	20	75	39	0.058
		4	13	40	80	63	0.17	63	0.15	13	38	76	60	0.13	60	0.13	13	20	94	58	0.15
		5	13	48	96	76	0.35	76	0.31	13	46	92	73	0.26	74	0.26	13	20	113	77	0.35
		6	13	56	112	91	0.71	90	0.63	13	54	108	87	0.53	87	0.52	13	20	131	94	0.84
		7	14	45	135	111	4.42	111	3.84	14	44	132	109	3.35	109	3.34	14	22	165	126	4.44
2	L	1	12	32	32	22	0.003	21	0.003	12	30	30	19	0.002	19	0.002	12	19	34	25	0.003
		2	12	47	47	34	0.009	34	0.009	12	43	43	30	0.006	31	0.006	13	20	56	25	0.024
		3	13	32	64	50	0.079	50	0.07	13	30	60	46	0.058	46	0.058	13	20	75	39	0.076
		4	13	40	80	63	0.17	63	0.15	13	38	76	60	0.12	60	0.12	13	20	94	57	0.21
		5	13	48	96	77	0.35	76	0.31	13	46	92	73	0.26	74	0.26	13	20	113	77	0.56
		6	13	56	112	90	0.71	90	0.63	13	54	108	87	0.53	87	0.52	13	20	131	94	1.42
		7	13	43	129	104	2.15	104	1.86	13	41	123	101	1.63	101	1.62	14	22	165	125	7.33
$2^{16}+1$	3	1	13	31	62	45	0.011	44	0.01	13	59	59	36	0.004	35	0.004	13	33	58	34	0.005
		2	13	47	94	66	0.034	66	0.03	13	45	90	63	0.025	63	0.025	13	33	91	67	0.02
		3	14	43	129	102	0.24	103	0.21	14	41	123	95	0.19	96	0.18	13	33	124	100	0.063
		4	14	53	159	131	0.52	132	0.45	14	52	156	125	0.4	125	0.39	13	33	157	133	0.17
		5	14	48	192	158	1.41	161	1.2	14	47	188	155	1.07	155	1.04	14	34	196	171	0.8
		6	14	56	224	189	2.85	189	2.44	14	55	220	184	2.18	184	2.13	14	34	230	205	2.03
		7	14	51	255	221	7.61	220	6.51	14	50	250	214	5.98	214	5.73	14	34	264	239	4.86
$2^{16}+1$	L	1	13	31	62	44	0.011	44	0.01	13	59	59	35	0.004	35	0.004	13	33	58	34	0.006
		2	13	47	94	66	0.034	66	0.03	13	45	90	63	0.025	63	0.025	13	33	91	67	0.022
		3	13	41	123	100	0.12	100	0.1	13	40	120	92	0.091	92	0.09	13	33	124	100	0.07
		4	13	52	156	127	0.25	127	0.22	13	50	150	120	0.2	120	0.19	13	33	157	133	0.2
		5	14	48	192	159	1.41	158	1.31	14	47	188	155	1.18	155	1.13	14	34	196	171	1.09
		6	14	56	224	189	2.85	190	2.65	14	55	220	184	2.4	184	2.3	14	34	230	205	2.78
		7	14	51	255	220	7.64	221	7.11	14	50	250	214	6.49	214	6.19	14	34	264	239	6.89
$2^{30}-2^{18}+1$	3	1	13	52	104	71	0.011	71	0.01	13	43	86	49	0.008	50	0.008	13	46	85	48	0.005
		2	14	51	153	116	0.11	115	0.091	14	45	135	95	0.08	94	0.077	14	47	133	96	0.046
		3	14	49	196	159	0.32	159	0.27	14	45	180	138	0.24	137	0.23	14	47	180	143	0.13
		4	14	49	245	200	0.9	200	0.77	14	56	224	183	0.52	181	0.5	14	47	227	190	0.36
		5	14	58	290	246	1.87	245	1.59	14	54	270	225	1.46	226	1.34	14	47	274	238	0.86
		6	15	57	342	294	9.66	294	8.15	14	53	318	269	3.56	269	3.34	15	48	328	291	4.62
		7	15	55	385	340	23.79	340	19.89	15	53	371	321	18.72	320	17.13	15	48	376	339	10.84
$2^{30}-2^{18}+1$	L	1	13	52	104	71	0.011	71	0.01	13	43	86	49	0.009	49	0.009 s	13	46	85	48	0.006
		2	13	50	150	114	0.052	113	0.045	13	44	132	91	0.04	92	0.038 s	13	46	131	94	0.023
		3	14	49	196	158	0.33	158	0.3	14	45	180	137	0.27	137	0.25 s	14	47	180	143	0.15
		4	14	49	245	200	0.93	200	0.85	14	56	224	181	0.59	181	0.54 s	14	47	227	190	0.42
		5	14	58	290	245	1.92	245	1.76	14	54	270	225	1.61	225	1.48 s	14	47	274	237	1.11
		6	14	56	336	288	5.26	288	4.51	14	53	318	269	4.19	268	3.79 s	14	47	321	284	2.87
		7	15	55	385	341	24.06	340	23.2	14	52	364	312	10.57	312	9.6 s	14	47	368	331	7.06

G.2 Polynomial Evaluation

Table 7: Comparison of noise growth and runtimes of BFV and BGV variants for a benchmark computation $\prod_{i=0}^{k} a_i x^i$: $|a_i| < 16$ at different plaintext moduli (for $t = 2$ we set $a_i = 1$). Hybrid key switching with d_{num} digits, and $\lambda \geq 128$. Here, e denotes the current noise magnitude, $\log Q$, the BFV ciphertext modulus, and $\log Q_L$, the equivalent ciphertext modulus in BGV without the last CRT modulus q_{L+1}.

			Original BFV							Our BFV							Our BGV				
			params			BEHZ		HPS		params			BFV-NEW		BFV-NEW-LVL		params			BGV-NEW	
t	d_{num}	k	$\log N$	$\log q_i$	$\log Q$	$\log e$	time(s)	$\log e$	time(s)	$\log N$	$\log q_i$	$\log Q$	$\log e$	time(s)	$\log e$	time(s)	$\log N$	$\log q_i$	$\log Q$	$\log e$	time(s)
2	3	2	12	33	33	22	0.003	22	0.003	12	30	30	18	0.002	19	0.002 s	12	20	35	25	0.002
		4	12	49	49	35	0.009	35	0.009	12	45	45	31	0.006	31	0.006 s	13	22	60	23	0.023
		8	13	34	68	51	0.08	52	0.071	13	32	64	48	0.059	48	0.059 s	13	23	85	43	0.063
		16	13	42	84	66	0.17	66	0.15	13	40	80	62	0.13	62	0.12 s	13	23	108	66	0.15
		32	13	51	102	80	0.35	80	0.31	13	48	96	76	0.26	76	0.24 s	13	24	136	93	0.32
		48	13	59	118	94	0.53	94	0.47	13	57	114	90	0.39	90	0.38 s	14	26	173	126	1.19
		64	13	59	118	95	0.72	94	0.64	13	57	114	91	0.53	91	0.49 s	14	26	173	127	1.43
2	L	2	12	33	33	22	0.003	22	0.003	12	30	30	19	0.002	19	0.002 s	12	20	35	26	0.003
		4	12	49	49	35	0.009	35	0.009	12	45	45	31	0.006	31	0.006 s	13	22	60	23	0.027
		8	13	34	68	52	0.079	51	0.07	13	32	64	48	0.059	48	0.059 s	13	23	85	43	0.076
		16	13	42	84	66	0.17	65	0.15	13	40	80	63	0.13	62	0.13 s	13	23	108	66	0.18
		32	13	51	102	81	0.35	80	0.31	13	48	96	77	0.26	77	0.24 s	13	24	136	93	0.4
		48	13	59	118	93	0.53	93	0.47	13	57	114	91	0.39	91	0.38 s	14	26	173	126	1.45
		64	13	59	118	94	0.72	95	0.64	13	57	114	91	0.53	92	0.49 s	14	26	173	127	1.74
$2^{16}+1$	3	2	13	34	68	41	0.012	40	0.01	13	32	64	35	0.009	36	0.009 s	13	38	68	38	0.007
		4	13	50	100	76	0.034	76	0.03	13	48	96	67	0.026	67	0.025 s	13	38	107	74	0.024
		8	14	45	135	106	0.25	107	0.22	14	43	129	100	0.19	100	0.18 s	13	39	148	116	0.061
		16	14	56	168	138	0.53	138	0.46	14	54	162	130	0.4	130	0.33 s	14	41	197	163	0.28
		32	14	50	200	166	1.43	167	1.22	14	49	196	161	1.1	161	0.78 s	14	42	244	208	0.61
		48	14	58	232	197	2.16	198	1.85	14	57	228	191	1.66	190	1.22 s	14	42	286	251	1.07
		64	14	58	232	199	2.89	199	2.48	14	57	228	191	2.22	191	1.54 s	14	43	293	256	1.27
$2^{16}+1$	L	2	13	34	68	42	0.012	42	0.01	13	32	64	38	0.009	39	0.009	13	38	68	37	0.007
		4	13	50	100	76	0.034	74	0.031	13	48	96	66	0.026	67	0.026	13	38	107	72	0.025
		8	13	44	132	104	0.12	105	0.1	13	42	126	98	0.092	97	0.086	13	39	148	115	0.064
		16	14	56	168	137	0.53	138	0.47	13	53	159	126	0.2	127	0.16	14	41	197	163	0.3
		32	14	50	200	166	1.42	166	1.32	14	49	196	161	1.2	161	0.79	14	42	244	209	0.64
		48	14	58	232	198	2.16	198	2.02	14	57	228	191	1.82	190	1.25	14	42	286	250	1.14
		64	14	58	232	198	2.89	199	2.7	14	57	228	191	2.44	191	1.58	14	43	293	256	1.33
$2^{30}-2^{18}+1$	3	2	13	55	110	71	0.012	71	0.01	13	46	92	51	0.008	51	0.009 s	13	52	96	51	0.007
		4	14	52	156	114	0.11	115	0.093	14	46	138	98	0.081	53	0.075 s	14	53	151	105	0.051
		8	14	51	204	164	0.32	163	0.28	14	46	184	143	0.25	142	0.21 s	14	54	208	162	0.13
		16	14	50	250	203	0.92	204	0.78	14	58	232	187	0.53	185	0.43 s	14	55	267	219	0.29
		32	14	59	295	253	1.9	253	1.62	14	56	280	231	1.48	231	0.97 s	15	57	333	284	1.29
		48	15	58	348	301	7.31	301	6.15	15	55	330	281	5.8	281	3.74 s	15	57	391	341	2.3
		64	15	58	348	302	9.95	302	8.46	15	55	330	282	7.79	282	4.71 s	15	58	397	346	2.74
$2^{30}-2^{18}+1$	L	2	13	55	110	70	0.012	70	0.011	13	46	92	53	0.009	53	0.009	13	52	96	51	0.007
		4	13	52	156	116	0.053	118	0.046	13	46	138	96	0.04	95	0.037	13	52	149	103	0.025
		8	14	51	204	164	0.33	164	0.31	14	46	184	142	0.28	141	0.22	14	54	208	161	0.13
		16	14	50	250	203	0.95	204	0.87	14	58	232	187	0.6	187	0.45	14	55	267	219	0.31
		32	14	59	295	253	1.96	252	1.79	14	56	280	231	1.64	231	1.03	14	56	328	278	0.66
		48	14	57	342	295	3.98	296	3.4	14	54	324	275	3.18	275	1.88	15	57	391	341	2.47
		64	14	57	342	295	5.39	295	4.59	14	54	324	275	4.27	276	2.34	15	58	397	347	2.88

G.3 Inner Product

Tables 8 illustrates the results for the inner product computation. We compare our BFV-NEW variant (BFV-NEW-LVL has same performance as no levels will be dropped at multiplication steps) without optimizations, with lazy key switching optimization, and with lazy scaling optimization.

Table 8: Runtimes of inner product $\sum_{i=1}^{k} x_i y_i$. Hybrid key switching with d_{num} digits, and $\lambda \geq 128$. Here e denotes the current noise magnitude.

			`params`			`BFV-NEW`		`BFV-NEW` Lazy KS		`BFV-NEW` Lazy Scale	
t	d_{num}	k	$\log N$	$\log q_i$	$\log Q$	$\log e$	time(s)	$\log e$	time(s)	$\log e$	time(s)
2	3	128	13	35	35	23	0.56	23	0.38	24	0.28
2	L	128	13	35	35	23	0.56	23	0.38	23	0.28
$2^{16}+1$	3	128	13	33	66	38	1.07	38	0.72	38	0.5
$2^{16}+1$	L	128	13	33	66	38	1.07	39	0.72	39	0.5
$2^{30}-2^{18}+1$	3	128	13	47	94	52	1.09	52	0.73	52	0.51
$2^{30}-2^{18}+1$	L	128	13	47	94	53	1.09	53	0.73	53	0.5