A Full RNS Variant of FV like Somewhat Homomorphic Encryption Schemes

Jean-Claude Bajard[1], Julien Eynard[2], M. Anwar Hasan[2], and Vincent Zucca[1]

[1]Sorbonne Universités, UPMC, CNRS, LIP6, Paris, France.
{jean-claude.bajard, vincent.zucca}@lip6.fr
[2]Department of Electrical and Computer Engineering, University of Waterloo.
{jeynard, ahasan}@uwaterloo.ca

Abstract. Since Gentry's breakthrough work in 2009, homomorphic cryptography has received a widespread attention. Implementation of a fully homomorphic cryptographic scheme is however still highly expensive. Somewhat Homomorphic Encryption (SHE) schemes, on the other hand, allow only a limited number of arithmetical operations in the encrypted domain, but are more practical. Many SHE schemes have been proposed, among which the most competitive ones rely on Ring Learning With Errors (RLWE) and operations occur on high-degree polynomials with large coefficients. This work focuses in particular on the Chinese Remainder Theorem representation (a.k.a. Residue Number Systems) applied to the large coefficients. In SHE schemes like that of Fan and Vercauteren (FV), such a representation remains hardly compatible with procedures involving coefficient-wise division and rounding required in decryption and homomorphic multiplication. This paper suggests a way to entirely eliminate the need for multi-precision arithmetic, and presents techniques to enable a full RNS implementation of FV-like schemes. For dimensions between 2^{11} and 2^{15}, we report speed-ups from $5\times$ to $20\times$ for decryption, and from $2\times$ to $4\times$ for multiplication.

Keywords: Lattice-based Cryptography, Homomorphic Encryption, FV, Residue Number Systems, Software Implementation

1 Introduction

Cryptographers' deep interests in lattices are for multiple reasons. Besides possessing highly desirable post-quantum security features, lattice-based cryptography relies on simple structures, allowing efficient asymptotic complexities, and is quite flexible in practice. In addition to encryption/signature schemes ([15, 25, 18, 7, 21, 22]), identity-based encryption [8], multilinear maps [10, 16], lattices are also involved in homomorphic encryption (HE). The discovery of this property by Gentry in 2009 [12], through the use of ideal rings, is a major breakthrough which has opened the door to many opportunities in terms of applications, especially when coupled with cloud computing.

HE is generally composed of a basic layer, which is a Somewhat Homomorphic Encryption scheme (SHE). Such a scheme allows us to compute a limited

number of additions and multiplications on ciphertexts. This can be explained by the fact that any ciphertext contains an inherent noise which increases after each homomorphic operation. Beyond a certain limit, the noise becomes too large to allow a correct decryption. This drawback may be tackled by using bootstrapping, which however constitutes a bottleneck in terms of efficiency. Further improvements of noise management [6, 5] have been suggested so that, in practice, and given an applicative context, it may be wiser to select an efficient SHE with parameters enabling a sufficient number of operations. For instance, schemes like `FV` [9] and `YASHE` [4] have been implemented and tested for evaluating the SIMON Feistel Cipher [17]. Among the today's more practical SHE schemes, `FV` is arguably one of the most competitive. This scheme is being currently considered by major stakeholders, such as the European H2020 HEAT consortium [26], as a viable candidate for practical homomorphic encryption.

Our Contribution This work focuses on practical improvement of SHE schemes, in particular `FV`. Despite the fact that the security of `YASHE` has been called into question recently [1], this scheme can also benefit from the present work. These schemes handle elements of a polynomial ring $\mathbb{Z}_q[X]/(X^n+1)$. The modulus q can be chosen as the product of some small moduli fitting with practical hardware requirements (machine word, etc.). This enables us to avoid the need of multi-precision arithmetic in almost the whole scheme. However, this CRT representation (a.k.a. Residue Number Systems, or RNS) is hardly compatible with a couple of core operations: coefficient-wise division and rounding, occurring in multiplication and decryption, and a noise management technique within homomorphic multiplication, relying on the access to a positional number system.

We show how to efficiently avoid any switch between RNS and the positional system for performing these operations. We present a full RNS variant of `FV` and analyse the new bounds on noise growth. A software implementation highlights the practical benefits of the new RNS variant.

It is important to note that this work is related to the arithmetic at the coefficient level. Thus, the security features of the original scheme are not modified.

Outline Section 2 provides some preliminaries about `FV` and RNS. Section 3 provides a full RNS variant of decryption. Section 4 gives a full RNS variant of homomorphic multiplication. Results of a software implementation are presented in Section 5. Finally, some conclusions are drawn.

2 Preliminaries

For the sake of clarity, proofs of lemmas, propositions and theorems within Sect. 3 and 4 are provided in an extended version [2].

Context High-level operations occur in a polynomial ring $\mathcal{R} = \mathbb{Z}[X]\ /(X^n+1)$ with n a power of 2. $\mathcal{R}$ is one-to-one mapped to integer polynomials of degree

strictly smaller than n. Most of the time, elements of $\mathcal{R}$ are denoted by lowercase boldface letters and identified by their coefficients. Polynomial arithmetic is done modulo (X^n+1). The infinity norm of $\boldsymbol{a} = (a_0, \ldots, a_{n-1}) \in \mathcal{R}$ is defined by $\|\boldsymbol{a}\|_\infty = \max_{0 \leqslant i \leqslant n-1}(|a_i|)$. Ciphertexts will be managed as polynomials (of degree 1) in $\mathcal{R}[Y]$. For $\mathtt{ct} \in \mathcal{R}[Y]$, we note $\|\mathtt{ct}\|_\infty = \max_i \|\mathtt{ct}[i]\|_\infty$, $\mathtt{ct}[i]$ being the coefficient of degree i in Y. The multiplicative law of $\mathcal{R}[Y]$ is denoted by $\star$.

Behind lattice-based cryptosystems in general, and FV in particular, lies the principle of noisy encryption. Additionally to the plaintext, a ciphertext contains a noise (revealed by using the secret key) which grows after each homomorphic operation. Since the homomorphic multiplication involves multiplications in $\mathcal{R}$, it is crucial that the size of a product in $\mathcal{R}$ does not increase too much. This increase is related to the ring constant $\delta = \sup\{\|\boldsymbol{f} \cdot \boldsymbol{g}\|_\infty / \|\boldsymbol{f}\|_\infty \cdot \|\boldsymbol{g}\|_\infty : (\boldsymbol{f}, \boldsymbol{g}) \in (\mathcal{R} \setminus \{\boldsymbol{0}\})^2\}$. It means that $\|\boldsymbol{f} \cdot \boldsymbol{g}\|_\infty \leqslant \delta \|\boldsymbol{f}\|_\infty \cdot \|\boldsymbol{g}\|_\infty$. For the specific ring $\mathcal{R}$ used here, δ is equal to n.

For our subsequent discussions on decryption and homomorphic multiplication, we denote the "Division and Rounding" in $\mathcal{R}[Y]$ (depending on parameters t, q which are defined thereafter) by:

$$\mathtt{DR}_i : \mathtt{ct} = \sum_{j=0}^{i} \mathtt{ct}[j] Y^j \in \mathcal{R}[Y] \mapsto \sum_{j=0}^{i} \left\lfloor \frac{t}{q} \mathtt{ct}[j] \right\rceil Y^j \in \mathcal{R}[Y] \ . \tag{1}$$

The notation $\lfloor \frac{t}{q} \boldsymbol{c} \rceil$, for any $\boldsymbol{c} \in \mathcal{R}$ (e.g. $\mathtt{ct}[j]$ in (1)), means a coefficient-wise division-and-rounding.

Plaintext and Ciphertext Spaces The plaintext space is determined by an integer parameter t ($t \geqslant 2$). A message is an element of $\mathcal{R}_t = \mathcal{R}/(t\mathcal{R})$, i.e. a polynomial of degree at most $n-1$ with coefficients in $\mathbb{Z}_t$. The notation $[\boldsymbol{m}]_t$ (resp. $|\boldsymbol{m}|_t$) means that coefficients lie in $[-t/2, t/2)$ (resp. $[0, t)$). Ciphertexts will lie in $\mathcal{R}_q[Y]$ with q a parameter of the scheme. On one side, some considerations about security imply a relationship between q and n which, for a given degree n, establish an upper bound for $\log_2(q)$ (cf. (6) in [9]). On the other side, the ratio $\Delta = \lfloor \frac{q}{t} \rfloor$ will basically determine the maximal number of homomorphic operations which can be done in a row to ensure a correct decryption.

RNS Representation Beyond the upper bound on $\log_2(q)$ due to security requirements, the composition of q has no restriction. So, q can be chosen as a product of small pairwise coprime moduli $q_1 \ldots q_k$. The reason for such a choice is the Chinese Remainder Theorem (CRT) which offers a ring isomorphism $\mathbb{Z}_q \xrightarrow{\sim} \prod_{i=1}^{k} \mathbb{Z}_{q_i}$. Thus, the CRT implies the existence of a non-positional number system (RNS) in which large integers (mod q) are mapped to sets of small residues. Furthermore, the arithmetic modulo q over large integers can be substituted by k independent arithmetics in the small rings $\mathbb{Z}_{q_i}$. The isomorphism can be naturally extended to polynomials: $\mathcal{R}_q \simeq \mathcal{R}_{q_1} \times \ldots \times \mathcal{R}_{q_k}$. It means that RNS can be used at the coefficient level to accelerate the arithmetic in $\mathcal{R}_q$.

In the rest of the paper, the letter q may refer either to the product $q_1 \dots q_k$ or to the "RNS base" $\{q_1, \dots, q_k\}$. Symbol ν denotes the "width" of the moduli. In other words, from now on, any modulus m (should it belong to q or to any other RNS base) is assumed to verify $m < 2^\nu$.

Asymmetric Keys The *secret key* $\boldsymbol{s}$ is picked up in $\mathcal{R}$ according to a discrete distribution χ_{key} on $\mathcal{R}$ (in practice, bounded by $B_{key} = 1$, i.e. $\|\boldsymbol{s}\|_\infty \leqslant 1$).

For creating the public key, an "error" distribution χ_{err} over $\mathcal{R}$ is used. In practice, this is a discrete distribution statistically close to a Gaussian (with mean 0 and standard deviation σ_{err}) truncated at B_{err} (e.g. $B_{err} = 6\sigma_{err}$). χ_{err} is related to the hardness of the underlying (search version of) RLWE problem (for which the purpose is, given samples $([-(\boldsymbol{a}_i\boldsymbol{s} + \boldsymbol{e}_i)]_q, \boldsymbol{a}_i)$ with $\boldsymbol{e}_i \leftarrow \chi_{err}$ and $\boldsymbol{a} \leftarrow \mathcal{U}(\mathcal{R}_q)$, to find $\boldsymbol{s}$; $\mathcal{U}(\mathcal{R}_q)$ is the uniform distribution on $\mathcal{R}_q$). The *public key* $\mathtt{pk}$ is created as follows: sample $\boldsymbol{a} \leftarrow \mathcal{U}(\mathcal{R}_q)$ and $\boldsymbol{e} \leftarrow \chi_{key}$, then output $\mathtt{pk} = (\boldsymbol{p}_0, \boldsymbol{p}_1) = ([-(\boldsymbol{a}\boldsymbol{s} + \boldsymbol{e})]_q, \boldsymbol{a})$.

Encryption, Addition, Inherent Noise of a Ciphertext Encryption and homomorphic addition are already fully compliant with RNS arithmetic. They are recalled hereafter:

- $\mathtt{Enc}_{\mathtt{FV}}([\boldsymbol{m}]_t)$: from samples $\boldsymbol{e}_1, \boldsymbol{e}_2 \leftarrow \chi_{err}$, $\boldsymbol{u} \leftarrow \chi_{key}$, output $\mathtt{ct} = (\mathtt{ct}[0], \mathtt{ct}[1]) = ([\Delta[\boldsymbol{m}]_t + \boldsymbol{p}_0\boldsymbol{u} + \boldsymbol{e}_1]_q, [\boldsymbol{p}_1\boldsymbol{u} + \boldsymbol{e}_2]_q)$.
- $\mathtt{Add}_{\mathtt{FV}}(\mathtt{ct}_1, \mathtt{ct}_2)$: output $([\mathtt{ct}_1[0] + \mathtt{ct}_2[0]]_q, [\mathtt{ct}_1[1] + \mathtt{ct}_2[1]]_q)$.

By definition, the *inherent noise* of $\mathtt{ct}$ (encrypting $[\boldsymbol{m}]_t$) is the polynomial $\boldsymbol{v}$ such that $[\mathtt{ct}(\boldsymbol{s})]_q = [\mathtt{ct}[0] + \mathtt{ct}[1]\boldsymbol{s}]_q = [\Delta[\boldsymbol{m}]_t + \boldsymbol{v}]_q$. Thus, it is revealed by evaluating $\mathtt{ct} \in \mathcal{R}_q[Y]$ on the secret key $\boldsymbol{s}$.

Elementary Operations A basic word will fit in ν bits. In RNS, an "inner modular multiplication" (`IMM`) in a small ring like $\mathbb{Z}_m$ is a core operation. If `EM` stands for an elementary multiplication of two words, in practice an `IMM` is costlier than an `EM`. But it can be well controlled. For instance, the moduli provided in `NFLlib` library [19] (cf. Sect. 5) enable a modular reduction which reduces to one `EM` followed by a multiplication modulo 2^ν. Furthermore, the cost of an inner reduction can be limited by using lazy reduction, e.g. during RNS base conversions used throughout this paper. `NTT` and `invNTT` denote the Number Theoretic Transform and its inverse in a ring $\mathcal{R}_m$ for a modulus m. They enable an efficient polynomial multiplication ($\mathtt{NTT}, \mathtt{invNTT} \in \mathcal{O}(n \log_2(n))$).

3 Towards a Full RNS Decryption

This section deals with the creation of a variant of the original decryption function $\mathtt{Dec}_{\mathtt{FV}}$, which will only involve RNS representation. The definition of $\mathtt{Dec}_{\mathtt{FV}}$ is recalled hereafter.

– $\texttt{Dec}_{\texttt{FV}}(\texttt{ct} = (\boldsymbol{c}_0, \boldsymbol{c}_1) \in \mathcal{R}_q[Y])$: compute $[\texttt{DR}_0([\texttt{ct}(\boldsymbol{s})]_q)]_t = \left[\left\lfloor \frac{t}{q}[\boldsymbol{c}_0 + \boldsymbol{c}_1\boldsymbol{s}]_q \right\rceil\right]_t$.

The idea is that computing $[\boldsymbol{c}_0 + \boldsymbol{c}_1\boldsymbol{s}]_q = [\Delta[\boldsymbol{m}]_t + \boldsymbol{v}]_q$ reveals the noise. If this noise is small enough, and given that $[\boldsymbol{m}]_t$ has been scaled by Δ, the function $\texttt{DR}_0$ allows to cancel the noise while scaling down $\Delta[\boldsymbol{m}]_t$ to recover $[\boldsymbol{m}]_t$. Concretely, decryption is correct as long as $\|\boldsymbol{v}\|_\infty < (\Delta - |q|_t)/2$, i.e. the size of the noise should not go further this bound after homomorphic operations.

The division-and-rounding operation makes $\texttt{Dec}_{\texttt{FV}}$ hardly compatible with RNS at a first sight. Because RNS is of non positional nature, only exact integer division can be naturally performed (by multiplying by a modular inverse). But it is not the case here. And the rounding operation involves comparisons which require to switch from RNS to another positional system anyway, should it be a classical binary system or a mixed-radix one [11]. To get an efficient RNS variant of $\texttt{Dec}_{\texttt{FV}}$, we use an idea of [3]. To this end, we introduce relevant RNS tools.

3.1 Fast RNS Base Conversion

At some point, the decryption requires, among others, a polynomial to be converted from $\mathcal{R}_q$ to $\mathcal{R}_t$. To achieve such kind of operations as efficiently as possible, we suggest to use a "fast base conversion". In order to convert residues of $x \in [0, q)$ from base q to a base $\mathcal{B}$ (e.g. $\{t\}$) coprime to q, we compute:

$$\texttt{FastBconv}(x, q, \mathcal{B}) = \left(\sum_{i=1}^{k} \left| x_i \frac{q_i}{q} \right|_{q_i} \times \frac{q}{q_i} \bmod m \right)_{m \in \mathcal{B}} . \tag{2}$$

This conversion is relatively fast. This is because the sum should ideally be reduced modulo q in order to provide the exact value x; instead, (2) provides $x + \alpha_x q$ for some integer $\alpha_x \in [0, k-1]$. Computing α_x requires costly operations in RNS. So this step is by-passed, at the cost of an approximate result.

$\texttt{FastBconv}$ naturally extends to polynomials of $\mathcal{R}$ by applying it coefficient-wise.

3.2 Approximate RNS Rounding

The above mentioned fast conversion allows us to efficiently compute an approximation of $\lfloor \frac{t}{q}[\boldsymbol{c}_0 + \boldsymbol{c}_1\boldsymbol{s}]_q \rceil$ modulo t. The next step consists in correcting this approximation.

First, we remark that $|\texttt{ct}(\boldsymbol{s})|_q$ can be used instead of $[\texttt{ct}(\boldsymbol{s})]_q$. Indeed, the difference between these two polynomials is a multiple of q. So, the division-and-rounding turns it into a polynomial multiple of t, which is cancelled by the last reduction modulo t. Second, a rounding would involve, at some point, a comparison. This is hardly compatible with RNS, so it is avoided. Therefore, we propose to simplify the computation, albeit at the price of possible errors, by replacing rounding by flooring. To this end, we use the following formula:

$$\left\lfloor \frac{t}{q} |\texttt{ct}(\boldsymbol{s})|_q \right\rfloor = \frac{t|\texttt{ct}(\boldsymbol{s})|_q - |t \cdot \texttt{ct}(\boldsymbol{s})|_q}{q} .$$

The division is now exact, so it can be done in RNS. Since this computation has to be done modulo t, the term $t|\mathtt{ct}(\boldsymbol{s})|_q$ cancels. Furthermore, the term $(|t \cdot \mathtt{ct}(\boldsymbol{s})|_q \bmod t)$ is obtained through a fast conversion.

Lemma 1 sums up the strategy by replacing $|\mathtt{ct}(\boldsymbol{s})|_q$ by $\gamma|\mathtt{ct}(\boldsymbol{s})|_q$, where γ is an integer which will help in correcting the approximation error.

Lemma 1. *Let* $\mathtt{ct}$ *be such that* $[\mathtt{ct}(\boldsymbol{s})]_q = \Delta[\boldsymbol{m}]_t + \boldsymbol{v} + q\boldsymbol{r}$*, and let* $\boldsymbol{v_c} := t\boldsymbol{v} - [\boldsymbol{m}]_t|q|_t$*. Let* γ *be an integer coprime to* q*. Then, for* $m \in \{t, \gamma\}$*, the following equalities hold modulo* m*:*

$$\begin{aligned} \mathtt{FastBconv}(|\gamma t \cdot \mathtt{ct}(\boldsymbol{s})|_q, q, \{t, \gamma\}) \times |-q^{-1}|_m &= \left\lfloor \gamma \frac{t}{q} [\mathtt{ct}(\boldsymbol{s})]_q \right\rceil - \boldsymbol{e} \\ &= \gamma\left([\boldsymbol{m}]_t + t\boldsymbol{r}\right) + \left\lfloor \gamma \frac{\boldsymbol{v_c}}{q} \right\rceil - \boldsymbol{e} \end{aligned} \tag{3}$$

where each integer coefficient of the error polynomial $\boldsymbol{e} \in \mathcal{R}$ *lies in* $[0, k]$*.*

The error $\boldsymbol{e}$ is due to the fast conversion and the replacement of rounding by flooring. It is the same error for residues modulo t and γ. The residues modulo γ will enable a fast correction of it and of the term $\lfloor \gamma \frac{\boldsymbol{v_c}}{q} \rceil$ at a same time. Also, note that $\boldsymbol{r}$ vanishes since it is multiplied by both t and γ.

3.3 Correcting the Approximate RNS Rounding

The next step is to show how γ in (3) can be used to correct the term $(\lfloor \gamma \frac{\boldsymbol{v_c}}{q} \rceil - \boldsymbol{e})$. It can be done efficiently when the polynomial $\boldsymbol{v_c}$ is such that $\|\boldsymbol{v_c}\|_\infty \leqslant q(\frac{1}{2} - \varepsilon)$, for some real number $\varepsilon \in (0, 1/2]$.

Lemma 2. *Let* $\|\boldsymbol{v_c}\|_\infty \leqslant q(\frac{1}{2} - \varepsilon)$*,* $\boldsymbol{e} \in \mathcal{R}$ *with coefficients in* $[0, k]$*, and* γ *an integer. Then,*

$$\gamma\varepsilon \geqslant k \Rightarrow \left[\left\lfloor \gamma \frac{\boldsymbol{v_c}}{q} \right\rceil - \boldsymbol{e} \right]_\gamma = \left\lfloor \gamma \frac{\boldsymbol{v_c}}{q} \right\rceil - \boldsymbol{e} \ . \tag{4}$$

Lemma 2 enables an efficient and correct RNS rounding as long as $k(\frac{1}{2} - \frac{\|\boldsymbol{v_c}\|_\infty}{q})^{-1} \sim \gamma$ has the size of a modulus. Concretely, one computes (3) and uses the centered remainder modulo γ to obtain $\gamma\left([\boldsymbol{m}]_t + t\boldsymbol{r}\right)$ modulo t, which reduces to $\gamma[\boldsymbol{m}]_t \bmod t$. And it remains to multiply by $|\gamma^{-1}|_t$ to recover $[\boldsymbol{m}]_t$.

3.4 A Full RNS Variant of $\mathtt{Dec_{FV}}$

The new variant of the decryption is detailed in Alg. 1. The main modification for the proposed RNS decryption is due to Lem. 2. As stated by Thm. 1, given a γ, the correctness of rounding requires a new bound on the noise to make the γ-correction technique successful.

Theorem 1. *Let* $\mathtt{ct}(\boldsymbol{s}) = \Delta[\boldsymbol{m}]_t + \boldsymbol{v} \pmod q$. *Let* γ *be a positive integer coprime to* t *and* q *such that* $\gamma > 2k/(1 - \frac{t|q|_t}{q})$. *For Alg. 1 returning* $[\boldsymbol{m}]_t$, *it suffices that* $\boldsymbol{v}$ *satisfies the following bound:*

$$\|\boldsymbol{v}\|_\infty \leqslant \frac{q}{t}\left(\frac{1}{2} - \frac{k}{\gamma}\right) - \frac{|q|_t}{2} \ . \tag{5}$$

There is a trade-off between the size of γ and the bound in (5). Ideally, $\gamma \sim 2k$ at the price of a (*a priori*) quite small bound on the noise. But by choosing $\gamma \sim 2^{p+1}k$ for $p < \nu - 1 - \lceil \log_2(k) \rceil$ (i.e. $\gamma < 2^\nu$ is a standard modulus), the bound $(\Delta(1 - 2^{-p}) - |q|_t)/2$ for a correct decryption should be close to the original bound $(\Delta - |q|_t)/2$ for practical values of ν. A concrete estimation of γ in Sect. 5.1 will show that γ can be chosen very close to $2k$ in practice, and thus fitting on a basic word by far.

Algorithm 1 $\mathtt{Dec}_{\mathtt{RNS}}(\mathtt{ct}, \boldsymbol{s}, \gamma)$

Require: $\mathtt{ct}$ an encryption of $[\boldsymbol{m}]_t$, and $\boldsymbol{s}$ the secret key, both in base q; an integer γ coprime to t and q
Ensure: $[\boldsymbol{m}]_t$
1: **for** $m \in \{t, \gamma\}$ **do**
2: $\quad \boldsymbol{s}^{(m)} \leftarrow \mathtt{FastBconv}(|\gamma t \cdot \mathtt{ct}(\boldsymbol{s})|_q, q, \{m\}) \times |-q^{-1}|_m \bmod m$
3: **end for**
4: $\tilde{\boldsymbol{s}}^{(\gamma)} \leftarrow [\boldsymbol{s}^{(\gamma)}]_\gamma$
5: $\boldsymbol{m}^{(t)} \leftarrow [(\boldsymbol{s}^{(t)} - \tilde{\boldsymbol{s}}^{(\gamma)}) \times |\gamma^{-1}|_t]_t$
6: **return** $\boldsymbol{m}^{(t)}$

3.5 Staying in RNS Is Asymptotically Better

In any decryption technique, $(\mathtt{ct}(\boldsymbol{s}) \bmod q)$ has to be computed first. To optimize this polynomial product, one basically performs $k\mathtt{NTT} \rightarrow kn\mathtt{IMM} \rightarrow k\mathtt{invNTT}$. For next steps, a simple strategy is to compute $(\lfloor \frac{t}{q}[\mathtt{ct}(\boldsymbol{s})]_q \rceil \bmod t)$ by doing an RNS-to-binary conversion in order to perform the division and rounding. By denoting $\boldsymbol{x}_i = |\mathtt{ct}(\boldsymbol{s})\frac{q_i}{q}|_{q_i}$, one computes $\sum_{i=1}^k \boldsymbol{x}_i \frac{q}{q_i} \bmod q$, compares it to $q/2$ so as to center the result, and performs division and rounding. Hence, the division-and-rounding requires $\mathcal{O}(k^2 n)\mathtt{EM}$. In practice, security analysis (cf. e.g. [9, 4, 17]) requires that $k\nu = \lceil \log_2(q) \rceil \in \mathcal{O}(n)$. So, the cost of leaving RNS to access a positional system is dominant in the asymptotic computational complexity.

Staying in RNS enables to get a better asymptotic complexity. Indeed, it is easy to see that Alg. 1 requires $\mathcal{O}(kn)$ operations (excluding the polynomial product). Thus, the cost of $\mathtt{NTT}$ is dominant in this case. By considering $k \in \mathcal{O}(n)$, we deduce $\mathcal{C}(\mathtt{Dec}_{\mathtt{FV}}) \in \mathcal{O}(n^3)$, while $\mathcal{C}(\mathtt{Dec}_{\mathtt{RNS}}) \in \mathcal{O}(n^2 \log_2(n))$. But the hidden constant in "$k \in \mathcal{O}(n)$" is small, and the $\mathtt{NTT}$, common to both variants, should avoid any noticeable divergence (cf. 5.4) for practical ranges for parameters.

In order to provide optimized RNS variants of decryption, we make two remarks. First, the reduction modulo q is unnecessary. Indeed, any extra multiple of q in the sum $\sum_{i=1}^{k} \boldsymbol{x}_i \frac{q}{q_i}$ is multiplied by $\frac{t}{q}$, making the resulting term a multiple of t, which is not affected by the rounding and is finally cancelled modulo t. Second, it is possible to precompute $\frac{t}{q}$ as a multiprecision floating point number in order to avoid a costly integer division. But given the first remark, it suffices to precompute the floating point numbers $\mathcal{Q}_i \sim \frac{t}{q_i}$ with $2\nu + \log_2(k) - \log_2(t)$ bits (~ 2 words) of precision. In this case, using standard double or quadruple (depending on ν) precision is sufficient. Finally, it is sufficient to compute $\lfloor \sum_{i=1}^{k} \boldsymbol{x}_i \mathcal{Q}_i \rceil \bmod t$. This represents about $2kn\texttt{EM}$. Reducing modulo t is nearly free of cost when t is a power of 2.

A second optimized RNS variant, with only integer arithmetic, is based on Alg. 1, in which γ is assumed to be coprime to t. It is possible to be slightly more efficient by noticing that the coprimality assumption can be avoided. This is because the division by γ is exact. To do it, the `for` loop can be done modulo $\gamma \times t$. For instance, even if t is a power of 2, one can choose γ as being a power of 2 too, and use the following lemma to finish the decryption efficiently.

Lemma 3. *Let γ be a power of 2. Let $\boldsymbol{z} := |\gamma[\boldsymbol{m}]_t + \lfloor \gamma \frac{\boldsymbol{v_c}}{q} \rceil - \boldsymbol{e}|_{\gamma t}$ coming from (3) when computed modulo γt. If γ satisfies (4), then ($\gg$ denotes the right bit-shifting, and $\boldsymbol{v}_1$ is the polynomial with all its coefficients being equal to 1)*

$$\left[(\boldsymbol{z} + \tfrac{\gamma}{2}\boldsymbol{v}_1) \gg \log_2(\gamma)\right]_t = [\boldsymbol{m}]_t \ . \tag{6}$$

Lemma 3 can be adapted to other values for γ. The important remark is that $[\boldsymbol{m}]_t$ is contained in the $\lfloor \log_2(t) \rfloor + 1$ most significant bits of $(\boldsymbol{z} + \frac{\gamma}{2}\boldsymbol{v}_1) \bmod \gamma t$. So, by choosing γ as a power of 2, a simple bit shifting enables to recover these bits. Finally, as soon as γt fits in 1 word, the cost of such variant (besides the polynomial product) reduces to $kn\texttt{IMM}$, or simply to $kn\texttt{EM}$ modulo $2^{\log_2(\gamma t)}$ whenever t is a power of 2.

Remark 1. In previous discussion, the product γt is assumed fitting in one machine word to simplify complexity analysis. However, for some applications, the plaintext modulus t can be bigger than a machine word (e.g. homomorphic neural networks [13], where $t > 2^{80}$). In such cases, either the plaintexts directly lie in $\mathcal{R}_t$, or t can be decomposed in a product of smaller moduli $t_1, \ldots, t_\ell$, enabling the use of RNS for encoding plaintexts (and then allowing better homomorphic multiplicative depth for a given dimension n). In the first case, the optimized RNS decryption (given by Lem. 3) remains available, but the residues modulo t should be handled with several words. In the second case, a plaintext is recovered by decrypting its residue modulo each of the t_i. These ℓ decryptions can be done as in Lem. 3, by using γ as a power of 2 (whatever the t_i's are). Finally, the plaintext is reconstructed from residues modulo the t_i's by using a classical RNS to binary conversion. However, this conversion is only related to the way the plaintexts are encoded. This is not handled by RNS decryption described in this paper, which only deals with representation of ciphertexts (i.e. modulo q).

4 Towards a Full RNS Homomorphic Multiplication

4.1 Preliminaries About $\texttt{Mult}_{\texttt{FV}}$

Below, we recall the main mechanisms of the homomorphic multiplication $\texttt{Mult}_{\texttt{FV}}$ from [9]. More precisely, we focus on the variant with version 1 for relinearisation step. First, two functions, of which the purpose is to limit a too rapid noise growth during a multiplication, are recalled (these functions will be denoted as in [4]). They are applicable to any $\boldsymbol{a} \in \mathcal{R}$, for any radix ω, and with the subsequent parameter $\ell_{\omega,q} = \lfloor \log_\omega(q) \rfloor + 1$. $\mathcal{D}_{\omega,q}$ is a decomposition in radix base ω, while $\mathcal{P}_{\omega,q}$ gets back powers of ω which are lost within the decomposition process.

$$\forall \boldsymbol{a} \in \mathcal{R}, \begin{cases} \mathcal{D}_{\omega,q}(\boldsymbol{a}) = ([\boldsymbol{a}]_\omega, [\lfloor \boldsymbol{a}\omega^{-1} \rfloor]_\omega, \ldots, [\lfloor \boldsymbol{a}\omega^{-(\ell_{\omega,q}-1)} \rfloor]_\omega) \in \mathcal{R}_\omega^{\ell_{\omega,q}} \\ \mathcal{P}_{\omega,q}(\boldsymbol{a}) = ([\boldsymbol{a}]_q, [\boldsymbol{a}\omega]_q, \ldots, [\boldsymbol{a}\omega^{\ell_{\omega,q}-1}]_q) \in \mathcal{R}_q^{\ell_{\omega,q}} \end{cases} . \quad (7)$$

In particular, for any $(\boldsymbol{a}, \boldsymbol{b}) \in \mathcal{R}^2$, $\langle \mathcal{D}_{\omega,q}(\boldsymbol{a}), \mathcal{P}_{\omega,q}(\boldsymbol{b}) \rangle \equiv \boldsymbol{ab} \bmod q$.

Next, $\texttt{Mult}_{\texttt{FV}}$ is built as follows ($\texttt{rlk}_{\texttt{FV}}$ is a public relinearisation key):

- $\texttt{rlk}_{\texttt{FV}} = \left([\mathcal{P}_{\omega,q}(\boldsymbol{s}^2) - (\overrightarrow{\boldsymbol{e}} + \boldsymbol{s}\overrightarrow{\boldsymbol{a}})]_q, \overrightarrow{\boldsymbol{a}}\right)$ where $\overrightarrow{\boldsymbol{e}} \leftarrow \chi_{err}^{\ell_{\omega,q}}$, $\overrightarrow{\boldsymbol{a}} \leftarrow \mathcal{U}(\mathcal{R}_q)^{\ell_{\omega,q}}$,
- $\texttt{Relin}_{\texttt{FV}}(\boldsymbol{c}_0, \boldsymbol{c}_1, \boldsymbol{c}_2, \texttt{rlk}_{\texttt{FV}})$:
  compute $([\boldsymbol{c}_0 + \langle \mathcal{D}_{\omega,q}(\boldsymbol{c}_2), \texttt{rlk}_{\texttt{FV}}[0] \rangle]_q, [\boldsymbol{c}_1 + \langle \mathcal{D}_{\omega,q}(\boldsymbol{c}_2), \texttt{rlk}_{\texttt{FV}}[1] \rangle]_q)$,
- $\texttt{Mult}_{\texttt{FV}}(\texttt{ct}_1, \texttt{ct}_2)$: denote $\texttt{ct}_\star = \texttt{ct}_1 \star \texttt{ct}_2$ (degree-2 element of $\mathcal{R}[Y]$),
  - Step 1: $\widetilde{\texttt{ct}}_{mult} = [\texttt{DR}_2(\texttt{ct}_\star)]_q = ([\texttt{DR}_0(\texttt{ct}_\star[i])]_q)_{i \in \{0,1,2\}}$,
  - Step 2: $\texttt{ct}_{mult} = \texttt{Relin}_{\texttt{FV}}(\widetilde{\texttt{ct}}_{mult})$.

There are two main obstacles to a full RNS variant. First, the three calls to $\texttt{DR}_0$ in Step 1, for which the context is different than for the decryption. While in the decryption we are working with a noise whose size can be controlled, and while we are reducing a value from q to $\{t\}$, here the polynomial coefficients of the product $\texttt{ct}_1 \star \texttt{ct}_2$ have kind of random size modulo q (for each integer coefficient) and have to be reduced towards q. Second, the function $\mathcal{D}_{\omega,q}$ (in $\texttt{Relin}_{\texttt{FV}}$) requires, by definition, an access to a positional system (in radix base ω), which is hardly compatible with RNS.

4.2 Auxiliary RNS Bases

Step 1 requires to use enough moduli to contain any product, in $\mathcal{R}[Y]$ (i.e. on $\mathbb{Z}$), of degree-1 elements from $\mathcal{R}_q[Y]$. So, we need an auxiliary base $\mathcal{B}$, additionally to the base q. We assume that $\mathcal{B}$ contains ℓ moduli (while q owns k elements). A sufficient size for ℓ will be given later. An extra modulus $m_{\texttt{sk}}$ is added to $\mathcal{B}$ to create an extended base $\mathcal{B}_{\texttt{sk}}$. It will be used for a transition between the new steps 1 and 2. Computing the residues of ciphertexts in $\mathcal{B}_{\texttt{sk}}$ is done through a fast conversion from q. In order to reduce the extra multiples of q (called "q-overflows" in further discussions) this conversion can produce, a single-modulus base $\tilde{m}$ is introduced. All these bases are assumed to be pairwise coprime.

Reducing (mod q) a Ciphertext in $\mathcal{B}_{\mathtt{sk}}$ A $\mathtt{FastBconv}$ from q can create q-overflows (i.e. unnecessary multiples of q) in the output. To limit the impact on noise growth (because of division by q in step 1), we give an efficient way to reduce a polynomial $\boldsymbol{c}+q\boldsymbol{u}$ in $\mathcal{B}_{\mathtt{sk}}$. It should be done prior to each multiplication. For that purpose, we use the residues modulo $\tilde{m}$ as it is described in Alg. 2.

Algorithm 2 $\mathtt{SmMRq}_{\tilde{m}}((\boldsymbol{c}''_m)_{m\in\mathcal{B}_{\mathtt{sk}}\cup\{\tilde{m}\}})$: Small Montgomery Reduction mod q

Require: $\boldsymbol{c}''$ in $\mathcal{B}_{\mathtt{sk}}\cup\{\tilde{m}\}$
Ensure: $\boldsymbol{c}'$ in $\mathcal{B}_{\mathtt{sk}}$, with $\boldsymbol{c}'\equiv\boldsymbol{c}''\tilde{m}^{-1} \bmod q$, $\|\boldsymbol{c}'\|_\infty \leqslant \frac{\|\boldsymbol{c}''\|_\infty}{\tilde{m}}+\frac{q}{2}$
1: $\boldsymbol{r}_{\tilde{m}} \leftarrow [-\boldsymbol{c}''_{\tilde{m}}/q]_{\tilde{m}}$
2: **for** $m\in\mathcal{B}_{\mathtt{sk}}$ **do**
3: $\quad\boldsymbol{c}'_m \leftarrow |(\boldsymbol{c}''_m+q\boldsymbol{r}_{\tilde{m}})\tilde{m}^{-1}|_m$
4: **end for**
5: **return** $\boldsymbol{c}'$ in $\mathcal{B}_{\mathtt{sk}}$

Lemma 4. *On input $\boldsymbol{c}''_m=|[\tilde{m}\boldsymbol{c}]_q+q\boldsymbol{u}|_m$ for all $m\in\mathcal{B}_{sk}\cup\{\tilde{m}\}$, with $\|\boldsymbol{u}\|_\infty\leqslant\tau$, and given a parameter $\rho>0$, then Alg. 2 returns $\boldsymbol{c}'$ in $\mathcal{B}_{sk}$ with $\boldsymbol{c}'\equiv\boldsymbol{c} \bmod q$ and $\|\boldsymbol{c}'\|_\infty\leqslant\frac{q}{2}(1+\rho)$ if $\tilde{m}$ satisfies:*

$$\tilde{m}\rho\geqslant 2\tau+1\ . \tag{8}$$

To use this fast reduction, the ciphertexts have to be handled in base q through the Montgomery [20] representation with respect to $\tilde{m}$ (i.e. $|\tilde{m}\boldsymbol{c}|_q$ instead of $|\boldsymbol{c}|_q$). This can be done for free of cost during the base conversions (in (2), multiply residues of $\boldsymbol{c}$ by precomputed $|\frac{\tilde{m}q_i}{q}|_{q_i}$ instead of $|\frac{q_i}{q}|_{q_i}$). Since $\{\tilde{m}\}$ is a single-modulus base, conversion of $\boldsymbol{r}_{\tilde{m}}$ from $\{\tilde{m}\}$ to $\mathcal{B}_{\mathtt{sk}}$ (l. 3 of Alg. 2) is a simple copy-paste when $\tilde{m}<m_i$. Finally, if $\mathtt{SmMRq}_{\tilde{m}}$ is performed right after a $\mathtt{FastBconv}$ from q (for converting $|\tilde{m}\boldsymbol{c}|_q$), τ is nothing but k.

4.3 Adapting the First Step

We recall that originally this step is the computation of $[\mathtt{DR}_2(\mathtt{ct}_\star)]_q$. Unlike the decryption, a γ-correction technique does not guarantee an exact rounding. Indeed, for the decryption we wanted to get $\mathtt{DR}_0([\mathtt{ct}(\boldsymbol{s})]_q)$, and through $\boldsymbol{s}$ we had access to the noise of $\mathtt{ct}$, on which we have some control. In the present context, we cannot ensure a condition like $\|[t\cdot\mathtt{ct}_\star]_q\|_\infty\leqslant q(\frac{1}{2}-\varepsilon)$, for some $\varepsilon^{-1}\sim 2^\nu$, which would enable the use of an efficient γ-correction. Thus, we suggest to perform a simple uncorrected RNS flooring. For that purpose, we define:

$$\forall\boldsymbol{a}\in\mathcal{R}, \mathtt{fastRNSFloor}_q(\boldsymbol{a},m):=(\boldsymbol{a}-\mathtt{FastBconv}(|\boldsymbol{a}|_q,q,m))\times|q^{-1}|_m \bmod m.$$

Algorithm 2 should be executed first. Consequently, by Lem. 4, if $\tilde{m}$ satisfies the bound in (8) for a given parameter $\rho>0$, we assume having, in $\mathcal{B}_{\mathtt{sk}}$, the residues of $\mathtt{ct}'_i$ ($\equiv\mathtt{ct}_i \bmod q$) such that:

$$\|\mathtt{ct}'_\star:=\mathtt{ct}'_1\star\mathtt{ct}'_2\|_\infty\leqslant\delta\tfrac{q^2}{2}(1+\rho)^2\ . \tag{9}$$

The parameter ρ is determined in practice. Notice that, in base q, $\mathtt{ct}'_i$ and $\mathtt{ct}_i$ are equal.

Lemma 5. *Let the residues of $\mathtt{ct}'_i \equiv \mathtt{ct}_i \bmod q$ be given in base $q \cup \mathcal{B}_{sk}$, with $\|\mathtt{ct}'_i\|_\infty \leqslant \frac{q}{2}(1+\rho)$ for $i \in \{1,2\}$. Let $\mathtt{ct}'_\star = \mathtt{ct}'_1 \star \mathtt{ct}'_2$. Then, for $j \in \{0,1,2\}$,*

$$\mathit{fastRNSFloor}_q(t \cdot \mathtt{ct}'_\star[j], \mathcal{B}_{sk}) = \left\lfloor \frac{t}{q} \mathtt{ct}'_\star[j] \right\rceil + \boldsymbol{b}_j \text{ in } \mathcal{B}_{sk}, \text{ with } \|\boldsymbol{b}_j\|_\infty \leqslant k \ . \quad (10)$$

A first part of the noise growth is detailed in the following proposition.

Proposition 1. *Let $\widetilde{\mathtt{ct}}_{mult} = \mathit{DR}_2(\mathtt{ct}'_\star)$ with (9) satisfied, and $r_\infty := \frac{1+\rho}{2}(1+\delta B_{key})+1$. Let $\boldsymbol{v}_i$ be the inherent noise of $\mathtt{ct}'_i$. Then $\widetilde{\mathtt{ct}}_{mult}(\boldsymbol{s}) = \Delta\,[\boldsymbol{m}_1\boldsymbol{m}_2]_t + \tilde{\boldsymbol{v}}_{mult} \pmod q$ with:*

$$\begin{aligned}\|\tilde{\boldsymbol{v}}_{mult}\|_\infty < \delta t(r_\infty + \tfrac{1}{2})(\|\boldsymbol{v}_1\|_\infty + \|\boldsymbol{v}_2\|_\infty) + \tfrac{\delta}{2}\min\|\boldsymbol{v}_i\|_\infty + \delta t|q|_t(r_\infty+1) \\ + \tfrac{1}{2}(3 + |q|_t + \delta B_{key}(1+\delta B_{key})) \ . \end{aligned} \quad (11)$$

4.4 Transitional Step

Lemma 5 states that we have got back $\mathtt{DR}_2(\mathtt{ct}'_\star) + \mathtt{b}$ in $\mathcal{B}_{\mathtt{sk}}$ so far, where we have denoted $(\boldsymbol{b}_0, \boldsymbol{b}_1, \boldsymbol{b}_2)$ by $\mathtt{b}$. To perform the second step of multiplication, we need to convert it in base q. However, the conversion has to be exact because extra multiples of $M = m_1 \dots m_\ell$ cannot be tolerated. $m_{\mathtt{sk}}$ allows us to perform a complete Shenoy and Kumaresan like conversion [24]. The next lemma describes such kind of conversion for a more general context where the input can be either positive or negative, and can be larger, in absolute value, than M.

Lemma 6. *Let $\mathcal{B}$ be an RNS base and m_{sk} be a modulus coprime to $M = \prod_{m\in\mathcal{B}} m$. Let x be an integer such that $|x| < \lambda M$ (for some real number $\lambda \geqslant 1$) and whose residues are given in $\mathcal{B}_{sk}$. Let's assume that m_{sk} satisfies $m_{sk} \geqslant 2(|\mathcal{B}| + \lceil\lambda\rceil)$. Let $\alpha_{sk,x}$ be the following integer:*

$$\alpha_{sk,x} := \left[(\mathit{FastBconv}(x, \mathcal{B}, \{m_{sk}\}) - x_{sk})M^{-1}\right]_{m_{sk}} \ . \quad (12)$$

Then, for x being either positive or negative, the following equality holds:

$$\mathit{FastBconvSK}(x, \mathcal{B}_{sk}, q) := (\mathit{FastBconv}(x, \mathcal{B}, q) - \alpha_{sk,x}M) \bmod q = x \bmod q \ . \quad (13)$$

Consequently, since $\|\mathtt{DR}_2(\mathtt{ct}'_\star) + \mathtt{b}\|_\infty \leqslant \delta t\frac{q}{2}(1+\rho)^2 + \frac{1}{2} + k$, we can establish the following proposition.

Proposition 2. *Given a positive real number λ, let m_{sk} and $\mathcal{B}$ be such that:*

$$\lambda M > \delta t\tfrac{q}{2}(1+\rho)^2 + \tfrac{1}{2} + k, \ m_{sk} \geqslant 2(|\mathcal{B}| + \lceil\lambda\rceil) \ . \quad (14)$$

Let's assume that $\mathit{DR}_2(\mathtt{ct}'_\star) + \mathtt{b}$ is given in $\mathcal{B}_{sk}$, with $\|\mathtt{b}\|_\infty \leqslant k$. Then,

$$\mathit{FastBconvSK}(\mathit{DR}_2(\mathtt{ct}'_\star) + \mathtt{b}, \mathcal{B}_{sk}, q) = (\mathit{DR}_2(\mathtt{ct}'_\star) + \mathtt{b}) \bmod q \ .$$

4.5 Adapting the Second Step

At this point, $\widetilde{\mathtt{ct}}_{mult} + \mathtt{b} = (\overline{c}_0, \overline{c}_1, \overline{c}_2)$ is known in base q ($\widetilde{\mathtt{ct}}_{mult} := \mathtt{DR}_2(\mathtt{ct}'_\star)$). We recall that the original second step of homomorphic multiplication would be done as follows:

$$\mathtt{ct}_{mult} = \left([\overline{c}_0 + \langle \mathcal{D}_{\omega,q}(\overline{c}_2), \mathcal{P}_{\omega,q}(\boldsymbol{s}^2) - (\overrightarrow{\boldsymbol{e}} + \boldsymbol{s}\overrightarrow{\boldsymbol{a}})\rangle]_q, [\overline{c}_1 + \langle \mathcal{D}_{\omega,q}(\overline{c}_2), \overrightarrow{\boldsymbol{a}}\rangle]_q\right) \tag{15}$$

where $\overrightarrow{\boldsymbol{e}} \leftarrow \chi_{err}^{\ell_{\omega,q}}$, $\overrightarrow{\boldsymbol{a}} \leftarrow \mathcal{U}(\mathcal{R}_q)^{\ell_{\omega,q}}$. The decomposition of $\overline{c}_2$ in radix ω enables a crucial reduction of the noise growth due to the multiplications by the terms $\boldsymbol{e}_i + \boldsymbol{s}\boldsymbol{a}_i$. It cannot be done directly in RNS as is. Indeed, it would require a costly switch between RNS and positional representation in radix ω. However, we can do something very similar. We recall that we can write $\overline{c}_2 = \sum_{i=1}^{k} |\overline{c}_2 \frac{q_i}{q}|_{q_i} \times \frac{q}{q_i} (\mathrm{mod}\ q)$. If ω has the same order of magnitude than 2^ν (size of moduli in q), we obtain a similar limitation of the noise growth by using the vectors $\xi_q(\overline{c}_2) = (|\overline{c}_2 \frac{q_1}{q}|_{q_1}, \ldots, |\overline{c}_2 \frac{q_k}{q}|_{q_k})$ and $\mathcal{P}_{\mathtt{RNS},q}(\boldsymbol{s}^2) = (|\boldsymbol{s}^2 \frac{q}{q_1}|_q, \ldots, |\boldsymbol{s}^2 \frac{q}{q_k}|_q)$, both in $\mathcal{R}^k$. This is justified by the following lemma.

Lemma 7. *For any $\boldsymbol{c} \in \mathcal{R}$, $\langle \xi_q(\boldsymbol{c}), \mathcal{P}_{RNS,q}(\boldsymbol{s}^2)\rangle \equiv \boldsymbol{c}\boldsymbol{s}^2 \bmod q$.*

Thus, we replace the public relinearisation key $\mathtt{rlk}_{\mathtt{FV}}$ by the following one: $\mathtt{rlk}_{\mathtt{RNS}} = \left([\mathcal{P}_{\mathtt{RNS},q}(\boldsymbol{s}^2) - (\overrightarrow{\boldsymbol{e}} + \boldsymbol{s}\overrightarrow{\boldsymbol{a}})]_q, \overrightarrow{\boldsymbol{a}}\right)$. The next lemma helps for providing a bound on the extra noise introduced by this step.

Lemma 8. *Let $\overrightarrow{\boldsymbol{e}} \leftarrow \chi_{err}^k$, $\overrightarrow{\boldsymbol{a}} \leftarrow \mathcal{U}(\mathcal{R}_q)^k$, and $\boldsymbol{c} \in R$. Then,*

$$\| \left(\langle \xi_q(\boldsymbol{c}), -(\overrightarrow{\boldsymbol{e}} + \overrightarrow{\boldsymbol{a}}\boldsymbol{s})\rangle + \langle \xi_q(\boldsymbol{c}), \overrightarrow{\boldsymbol{a}}\rangle \boldsymbol{s}\right) \bmod q\|_\infty < \delta B_{err} k 2^\nu \ . \tag{16}$$

Remark 2. It is still possible to add a second level of decomposition (like in original approach, but applied on the residues) to limit a bit more the noise growth. Furthermore, Sect. 4.6 details how the size of $\mathtt{rlk}_{\mathtt{RNS}}$ can be reduced in a similar way that $\mathtt{rlk}_{\mathtt{FV}}$ could be through the method described in ([4], 5.4).

Finally, the output of the new variant of multiplication is the following one:

$$\mathtt{ct}_{mult} = \left(\left[\overline{c}_0 + \langle \xi_q(\overline{c}_2), \mathcal{P}_{\mathtt{RNS},q}(\boldsymbol{s}^2) - (\overrightarrow{\boldsymbol{e}} + \overrightarrow{\boldsymbol{a}}\boldsymbol{s})\rangle\right]_q, \left[\overline{c}_1 + \langle \xi_q(\overline{c}_2), \overrightarrow{\boldsymbol{a}}\rangle\right]_q\right) \ . \tag{17}$$

Proposition 3. *Let $\mathtt{ct}_{mult}$ be as in (17), and $\boldsymbol{v}_{mult}$ (resp. $\widetilde{\boldsymbol{v}}_{mult}$) the inherent noise of $\mathtt{ct}_{mult}$ (resp. $\widetilde{\mathtt{ct}}_{mult}$). Then $\mathtt{ct}_{mult}(\boldsymbol{s}) = \Delta [\boldsymbol{m}_1\boldsymbol{m}_2]_t + \boldsymbol{v}_{mult} (\mathrm{mod}\ q)$ with:*

$$\|\boldsymbol{v}_{mult}\|_\infty < \|\widetilde{\boldsymbol{v}}_{mult}\|_\infty + k(1 + \delta B_{key}(1 + \delta B_{key})) + \delta k B_{err} 2^{\nu+1} \ . \tag{18}$$

Algorithm 3 depicts the scheme of the new full RNS variant $\mathtt{Mult}_{\mathtt{RNS}}$.

Algorithm 3 RNS homomorphic multiplication $\texttt{Mult}_{\texttt{RNS}}$

Require: $\texttt{ct}_1, \texttt{ct}_2$ in q
Ensure: $\texttt{ct}_{mult}$ in q
S0: Convert fast $\texttt{ct}_1$ and $\texttt{ct}_2$ from q to $\mathcal{B}_{\texttt{sk}} \cup \{\tilde{m}\}$: $\rightsquigarrow \texttt{ct}_i'' = \texttt{ct}_i + q$-overflows
S1: Reduce q-overflows in $\mathcal{B}_{\texttt{sk}}$: $(\texttt{ct}_i' \text{ in } \mathcal{B}_{\texttt{sk}}) \leftarrow \texttt{SmMRq}_{\tilde{m}}(((\texttt{ct}_i'')_m)_{m\in\mathcal{B}_{\texttt{sk}}\cup\{\tilde{m}\}})$
S2: Compute the product $\texttt{ct}_\star' = \texttt{ct}_1' \star \texttt{ct}_2'$ in $q \cup \mathcal{B}_{\texttt{sk}}$
S3: Convert fast from q to $\mathcal{B}_{\texttt{sk}}$ to achieve first step (approximate rounding) in $\mathcal{B}_{\texttt{sk}}$:
$(\widetilde{\texttt{ct}}_{mult} + \texttt{b} = \texttt{DR}_2(\texttt{ct}_\star') + \texttt{b} \text{ in } \mathcal{B}_{\texttt{sk}}) \leftarrow \ldots \leftarrow \texttt{FastBconv}(t \cdot \texttt{ct}_\star', q, \mathcal{B}_{\texttt{sk}})$
S4: Convert exactly from $\mathcal{B}_{\texttt{sk}}$ to q to achieve transitional step:
$(\widetilde{\texttt{ct}}_{mult} + \texttt{b} \text{ in } q) \leftarrow \texttt{FastBconvSK}(\widetilde{\texttt{ct}}_{mult} + \texttt{b}, \mathcal{B}_{\texttt{sk}}, q)$
S5: Perform second step (relinearisation) in q:
$\texttt{ct}_{mult} \leftarrow \texttt{Relin}_{\texttt{RNS}}(\widetilde{\texttt{ct}}_{mult} + \texttt{b}) \bmod (q_1, \ldots, q_k)$

4.6 Reducing the Size of the Relinearization Key $\texttt{rlk}_{\texttt{RNS}}$

In [4], Sect. 5.4, a method to significantly reduce the size of the public evaluation key $\texttt{evk}$ is described (by truncating the ciphertext) and it is applicable to the original $\texttt{FV}$ scheme. We provide an efficient adaptation of such kind of optimization to the RNS variant of the relinearisation step.

We recall that the relinearisation is applied to a degree-2 ciphertext denoted here by $(\boldsymbol{c}_0, \boldsymbol{c}_1, \boldsymbol{c}_2)$. The initial suggestion was to set to zero, say, the i lowest significant components of the vector $\mathcal{D}_{\omega,q}(\boldsymbol{c}_2)$. Doing so is equivalent to replacing $\boldsymbol{c}_2$ by $\boldsymbol{c}_2' = \omega^i \lfloor \boldsymbol{c}_2 \omega^{-i} \rfloor = \boldsymbol{c}_2 - |\boldsymbol{c}_2|_{\omega^i}$. Thus, only the $\ell_{\omega,q} - i$ most significant components of $\texttt{rlk}_{\texttt{FV}}[0]$ (and then of $\texttt{rlk}_{\texttt{FV}}[1]$) are required (in other words, when $\texttt{rlk}_{\texttt{FV}}[0]$ is viewed as an $(\ell_{q,\omega}, k)$ RNS matrix by decomposing each component in base q, ik entries are set to zero like this). This optimization causes a greater noise than the one in Lem. 4 of [4]. Given $(\boldsymbol{c}_0, \boldsymbol{c}_1, \boldsymbol{c}_2)$ decryptable under $\boldsymbol{s}$, the relinearisation step provides the following ciphertext:

$$(\tilde{\boldsymbol{c}}_0, \tilde{\boldsymbol{c}}_1) := (\boldsymbol{c}_0 + \langle \mathcal{D}_{\omega,q}(\boldsymbol{c}_2'), \mathcal{P}_{\omega,q}(\boldsymbol{s}^2) - (\overrightarrow{\boldsymbol{e}} + \overrightarrow{\boldsymbol{a}}\,\boldsymbol{s}) \rangle,\ \ \boldsymbol{c}_1 + \langle \mathcal{D}_{\omega,q}(\boldsymbol{c}_2'), \overrightarrow{\boldsymbol{a}} \rangle)\ .$$

Thus, $(\tilde{\boldsymbol{c}}_0, \tilde{\boldsymbol{c}}_1)(\boldsymbol{s}) = \boldsymbol{c}_0 + \boldsymbol{c}_1\boldsymbol{s} + \boldsymbol{c}_2'\boldsymbol{s}^2 - \langle \mathcal{D}_{\omega,q}(\boldsymbol{c}_2'), \overrightarrow{\boldsymbol{e}} \rangle \bmod q$. Consequently, the extra noise comes from the following term:

$$\begin{aligned} \| - |\boldsymbol{c}_2|_{\omega^i}\boldsymbol{s}^2 - \langle \mathcal{D}_{\omega,q}(\boldsymbol{c}_2'), \overrightarrow{\boldsymbol{e}} \rangle\|_\infty &= \| - |\boldsymbol{c}_2|_{\omega^i}\boldsymbol{s}^2 - \textstyle\sum_{j=i}^{\ell_{\omega,q}-1} \mathcal{D}_{\omega,q}(\boldsymbol{c}_2)_j \boldsymbol{e}_j\|_\infty \\ &< \delta^2 \omega^i B_{key}^2 + (\ell_{\omega,q} - i)\delta\omega B_{err}\ . \end{aligned} \tag{19}$$

In the present RNS variant, the computation of $\lfloor \boldsymbol{c}_2 \omega^{-i} \rfloor$ is not straightforward. This could be replaced by $\lfloor \boldsymbol{c}_2 (q_1 \ldots q_i)^{-1} \rfloor$ through a Newton like interpolation (also known as mixed-radix conversion [11]). Though the result would be quite similar to the original optimization in terms of noise growth, its efficiency is not satisfying. Indeed, despite ik entries of the RNS matrix $\texttt{rlk}_{\texttt{RNS}}[0]$ can be set to zero like this, such a Newton interpolation is intrinsically sequential, while the division by ω^i followed by a flooring is simply achieved by an immediate zeroing of the lowest significant coefficients in radix ω representation.

For our approach, we rely on the fact that $\texttt{rlk}_{\texttt{RNS}}$ contains the RLWE-encryptions of the polynomials $|\boldsymbol{s}^2 \frac{q}{q_j}|_q$. Then, we notice that only the j^{th}-residue

of $|\boldsymbol{s}^2 \frac{q}{q_j}|_q$ can be non zero. So, let's assume that we want to cancel ik entries in $\texttt{rlk}_{\texttt{RNS}}[0]$ (as it has been done in $\texttt{rlk}_{\texttt{FV}}$ with the previous optimization). Then we choose, for each index j, a subset of index-numbers $\mathcal{I}_j \subseteq [1,k] \setminus \{j\}$ with cardinality i (i.e. at line j of $\texttt{rlk}_{\texttt{RNS}}$, choose i columns, except the diagonal one; these terms will be set to zero). Next, for each j, we introduce an RLWE-encryption of $|\boldsymbol{s}^2 \frac{q}{q_j q_{\mathcal{I}_j}}|_q$, where $q_{\mathcal{I}_j} = \prod_{s\in\mathcal{I}_j} q_s$, which is $(|\boldsymbol{s}^2 \frac{q}{q_j q_{\mathcal{I}_j}} - (\boldsymbol{e}_j + \boldsymbol{s}\boldsymbol{a}_j)|_q, \boldsymbol{a}_j)$. So far, the underlying security features are still relevant. Now, it remains to multiply this encryption by $q_{\mathcal{I}_j}$, which gives in particular $|\boldsymbol{s}^2 \frac{q}{q_j} - q_{\mathcal{I}_j}(\boldsymbol{e}_j + \boldsymbol{s}\boldsymbol{a}_j)|_q$. This is the j^{th}-line of the new matrix $\texttt{rlk}'_{\texttt{RNS}}[0]$. It is clear that this line contains zeros at columns index-numbered by $\mathcal{I}_j$. $\texttt{rlk}_{\texttt{RNS}}[1] = (\boldsymbol{a}_1, \ldots, \boldsymbol{a}_k)$ is replaced by $\texttt{rlk}'_{\texttt{RNS}}[1] = (|q_{\mathcal{I}_1}\boldsymbol{a}_1|_q, \ldots, |q_{\mathcal{I}_k}\boldsymbol{a}_k|_q)$.

Let's analyse the new noise growth. By evaluating in $\boldsymbol{s}$ the output of relinearisation with this new $\texttt{rlk}'_{\texttt{RNS}}$, we obtain:

$$\begin{aligned}
&\boldsymbol{c}_0 + \langle \xi_q(\boldsymbol{c}_2), \texttt{rlk}'_{\texttt{RNS}}[0]\rangle + (\boldsymbol{c}_1 + \langle \xi_q(\boldsymbol{c}_2), \texttt{rlk}'_{\texttt{RNS}}[1]\rangle)\,\boldsymbol{s} \\
&= \boldsymbol{c}_0 + \textstyle\sum_{j=1}^k |\boldsymbol{c}_2 \frac{q_j}{q}|_{q_j} \left(\boldsymbol{s}^2 \frac{q}{q_j} - q_{\mathcal{I}_j}(\boldsymbol{e}_j + \boldsymbol{s}\boldsymbol{a}_j)\right) + \left(\boldsymbol{c}_1 + \textstyle\sum_{j=1}^k |\boldsymbol{c}_2 \frac{q_j}{q}|_{q_j} q_{\mathcal{I}_j}\boldsymbol{a}_j\right)\boldsymbol{s} \\
&= \boldsymbol{c}_0 + \boldsymbol{c}_1\boldsymbol{s} + \boldsymbol{c}_2\boldsymbol{s}^2 - \textstyle\sum_{j=1}^k |\boldsymbol{c}_2 \frac{q_j}{q}|_{q_j} q_{\mathcal{I}_j}\boldsymbol{e}_j \ .
\end{aligned}$$

Consequently, the cancellation of ik terms in the public matrix $\texttt{rlk}_{\texttt{RNS}}[0]$ by using this method causes an extra noise growth bounded by (this can be fairly compared to (19) in the case where $\omega = 2^\nu$, i.e. $k = \ell_{\omega,q}$):

$$\| \textstyle\sum_{j=1}^k |\boldsymbol{c}_2 \frac{q_j}{q}|_{q_j} q_{\mathcal{I}_j}\boldsymbol{e}_j\|_\infty < \sum_{j=1}^k \delta q_j q_{\mathcal{I}_j} B_{err} < \delta k 2^{\nu(i+1)} B_{err} \ .$$

Therefore, the truncation of ciphertexts can be efficiently adapted to RNS representation without causing more significant noise growth.

4.7 About Computational Complexity

In a classical multi-precision (MP) variant, for the purpose of efficiency the multiplication can involve NTT-based polynomial multiplication to implement the ciphertext product (e.g. [23] for such kind of implementation on FPGA). This approach requires the use of a base $\mathcal{B}'$ (besides q) with $|\mathcal{B}'| = k+1$ for storing the product. Notice that, in RNS variant, we also have $|\mathcal{B}_{sk}| = k+1$ (the same amount of information is kept, but used differently). Thus, it is easy to show that RNS and MP variants involve the same number of NTT and invNTT operations in the case $\ell_{\omega,q} = k$ (e.g. the keys $\texttt{rlk}_{\texttt{FV}}$ and $\texttt{rlk}_{\texttt{RNS}}$ have the same size in this situation). In other words, the same number of polynomial products is performed in both cases when $\omega = 2^\nu$.

Above all, the RNS variant enables us to decrease the cost of other parts of the computation. Despite the fact that the asymptotic computational complexity of these parts remains identical for both variants, i.e. $\mathcal{O}(k^2 n)$ elementary multiplications, the RNS variant only involves single-precision integer arithmetic.

To sum up, because of a complexity of $\mathcal{O}(k^2 n \log_2(n))$ due to the NTT's, we keep the same asymptotic computational complexity $\mathcal{C}(\texttt{Mult}_{\texttt{FV}}) \underset{n\to+\infty}{\sim} \mathcal{C}(\texttt{Mult}_{\texttt{RNS}})$.

However, the most important fact is that multi-precision multiplications within MP variant are replaced in RNS by fast base conversions, which are simple matrix-vector products. Thus, $\texttt{Mult}_{\texttt{RNS}}$ retains all the benefits of RNS properties and is highly parallelizable.

5 Software Implementation

The C++ `NFLlib` library [19] was used for efficiently implementing the arithmetic in $\mathcal{R}_q$. It provides an efficient `NTT`-based product in $\mathcal{R}_q$ for q a product of 30 or 62-bit prime integers, and with degree n a power of 2, up to 2^{15}.

5.1 Concrete Examples of Parameter Settings

In this part, we analyse which depth can be reached in a multiplicative tree, and for which parameters.

The initial noise is at most $V = B_{err}(1 + 2\delta B_{key})$ [17]. The output of a tree of depth L has a noise bounded by $C^L_{\texttt{RNS},1}V + LC^{L-1}_{\texttt{RNS},1}C_{\texttt{RNS},2}$ (cf. [4], Lem. 9) with, for the present RNS variant:

$$\begin{cases} C_{\texttt{RNS},1} = 2\delta^2 t\frac{(1+\rho)}{2}B_{key} + \delta t(4+\rho) + \frac{\delta}{2} \ , \\ C_{\texttt{RNS},2} = (1+\delta B_{key})(\delta t|q|_t\frac{1+\rho}{2} + \delta B_{key}(k+\frac{1}{2})) + 2\delta t|q|_t + k(\delta B_{err}2^{\nu+1}+1) \\ \qquad\qquad +\frac{1}{2}(3+|q|_t) \ . \end{cases} \tag{20}$$

We denote by $L_{\texttt{RNS}} = \max\{L \in \mathbb{N} \mid C^L_{\texttt{RNS},1}V + LC^{L-1}_{\texttt{RNS},1}C_{\texttt{RNS},2} \leqslant \frac{q}{t}(\frac{1}{2} - \frac{k}{\gamma}) - \frac{|q|_t}{2}\}$ the depth allowed by $\texttt{Mult}_{\texttt{RNS}}$, when $\texttt{Dec}_{\texttt{RNS}}$ is used for decryption.

n	k	t	$L_{\texttt{RNS}}$ ($L_{\texttt{std}}$)	ρ	$\tilde{m}$	$\lceil\log_2(m_{\texttt{sk}})\rceil$	γ
2^{11}	3	2	2 (2)	5	(no need)	18	7
		2^{10}	1 (1)	5	(no need)	27	7
2^{12}	6	2	5 (6)	11	(no need)	21	13
		2^{10}	4 (4)	10	2	29	54
2^{13}	13	2	13 (13)	$\frac{1}{3}$	81	15	36
		2^{10}	9 (9)	13	3	31	58
2^{14}	26	2	25 (25)	$\frac{1}{2}$	106	17	53
		2^{10}	19 (19)	1	53	27	53
2^{15}	53	2	50 (50)	$\frac{1}{16}$	1712	20	753
		2^{10}	38 (38)	$\frac{1}{2}$	214	30	107

Table 1: Examples of parameter settings, by using 30-bit moduli of `NFLlib`.

For an 80-bit security level and parameters $B_{key} = 1$, $\sigma_{err} = 8$, $B_{err} = 6\sigma_{err}$, we consider the security analysis in [17], which provides ranges for $(\log_2(q), n)$ (cf. [17], Tab. 2). We analyse our parameters by using the moduli given in `NFLlib`, because those were used for concrete testing. For a 32-bit (resp. 64) implementation, a set of 291 30-bit (resp. 1000 62-bit) moduli is available. These moduli are chosen to enable an efficient modular reduction (cf. [19], Alg. 2).

Table 1 lists parameters when q and $\mathcal{B}$ are built with the 30-bit moduli of `NFLlib`. These parameters were determined by choosing the largest ρ (up to $2k-1$) allowing to reach the depth $L_{\texttt{RNS}}$. $L_{\texttt{std}}$ corresponds to the bounds given in [17] for a classical approach. Sufficient sizes for γ and $m_{\texttt{sk}}$ (allowing, for $m_{\texttt{sk}}$, to have $|\mathcal{B}| = k$ through (14) after having chosen for q the k greatest available moduli) are provided. For these specific parameters, the new bounds on the noise for RNS variant cause a smaller depth in only one case.

5.2 Influence of $\tilde{m}$ Over Noise Growth

After a fast conversion from q, ciphertexts in $\mathcal{B}_{\mathtt{sk}}$ can contain q-overflows and verify $\|\mathtt{ct}'_i\|_\infty < \frac{q}{2}(1+\tau)$. In a multiplicative tree without any addition, $\tau \leqslant 2k-1$. By applying Alg. 2, this bound decreases to $\frac{q}{2}(1+\rho)$, for some $0 < \rho \leqslant 2k-1$. Having $\rho = 2k-1$ in Tab. 1 means it is unnecessary to use $\mathtt{SmMRq}_{\tilde{m}}$ to reach the best depth. It happens only three times. Most of the time, doing such reduction is necessary before a multiplication so as to reach the highest depth. Moreover, choosing a lower ρ (i.e. higher $\tilde{m}$) than necessary can decrease the size of γ and $m_{\mathtt{sk}}$ (as shown in Tab. 1, one set of parameters leads to $\lceil \log_2(m_{\mathtt{sk}}) \rceil = 31$, avoiding the use of a 30-bit modulus; this can be solved by taking a larger $\tilde{m}$).

To illustrate the impact of $\mathtt{SmMRq}_{\tilde{m}}$, Fig. 1 depicts the noise growth for $\tilde{m} \in \{0, 2^8, 2^{16}\}$. Given Tab. 1, $\tilde{m} = 2^8$ is sufficient in such scenario to reach $L_{\mathtt{RNS}} = 13$. Against a computation with no reduction at all ($\tilde{m} = 0$, implying $L_{\mathtt{RNS}} = 11$ in this case), choosing $\tilde{m} = 2^8$ implies an average reduction of 25%. By using $\tilde{m} = 2^{16}$, we gain around 32%. Therefore, $\mathtt{SmMRq}_{\tilde{m}}$ has been systematically integrated within the implementation of $\mathtt{Mult}_{\mathtt{RNS}}$ for timing measurements in next part.

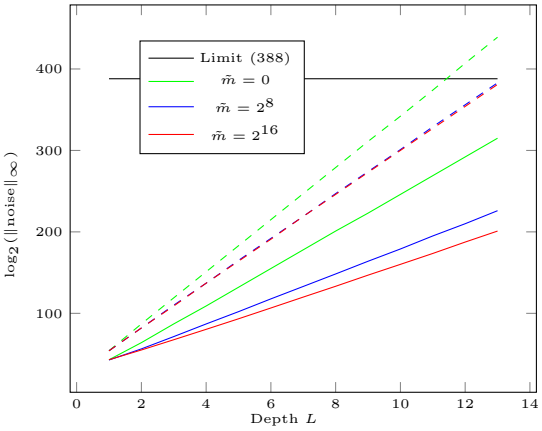

Fig. 1: Example of noise growth; $n = 2^{13}$, $\log_2(q) = 390$ ($\nu = 30, k = 13$), $t = 2$, $\sigma_{err} = 8$, $B_{key} = 1$ (dashed line: bound using (20); plain line: measurements).

5.3 Some Remarks

Convenient $\tilde{m}$ and γ Given values of ρ in Tab. 1, $\tilde{m} = 2^8$ (resp. $\tilde{m} = 2^{16}$) satisfies, by far, any set of analysed parameters. This enables an efficient and straightforward modular arithmetic through standard types like `uint8_t` (resp. `uint16_t`) and a type conversion towards the signed `int8_t` (resp. `int16_t`) immediately gives the centered remainder. Parameter analysis with such $\tilde{m}$ shows that $\gamma = 2^8$ is sufficient to ensure a correct decryption for all configurations. A reduction modulo γ can then be achieved by a simple type conversion to `uint8_t`.

Tested Algorithms The code[1] we compared with was implemented in the context of HEAT [26]. It is based on `NFLlib` too. Multi-precision arithmetic is handled with `GMP` 6.1.0 [14], and multiplications by $\frac{t}{q}$ are performed with integer divisions. $\mathtt{Mult}_{\mathtt{MP}}$ and $\mathtt{Dec}_{\mathtt{MP}}$ denote functions from this code.

[1] `https://github.com/CryptoExperts/FV-NFLlib`

$\texttt{Mult}_{\texttt{RNS}}$ was implemented as described by Alg. 3. Could the use of $\texttt{SmMRq}_{\tilde{m}}$ be avoided to reach the maximal theoretical depth, it was however systematically used. Its cost is negligible and it enables a noticeable decrease of noise growth.

Two variants of $\texttt{Dec}_{\texttt{RNS}}$ (cf. Sect. 3.5) have been implemented. Depending on ν, the one with floating point arithmetic (named $\texttt{Dec}_{\texttt{RNS-flp}}$ thereafter) uses `double` (resp. `long double`) for double (resp. quadruple) precision, and then does not rely on any other external library at all.

5.4 Results

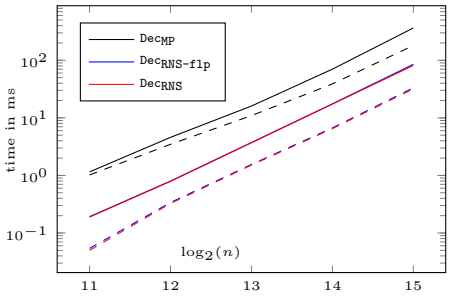

Fig. 2: Decryption time ($t = 2^{10}$), with $\nu = 30$ (plain lines) and $\nu = 62$ (dashed lines).

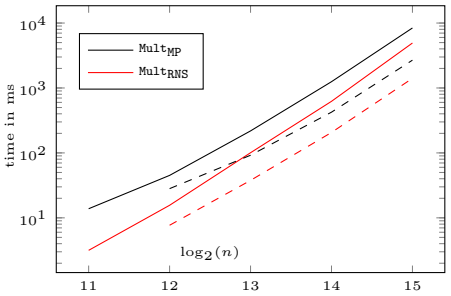

Fig. 3: Multiplication time ($t = 2^{10}$), with $\nu = 30$ (plain lines) and $\nu = 62$ (dashed lines).

The tests have been run on a laptop, running under Fedora 22 with Intel® Core™ i7-4810MQ CPU @ 2.80GHz and using GNU compiler g++ version 5.3.1. Hyper-Threading and Turbo Boost were deactivated. The timings have been measured on 2^{12} decryptions/multiplications for each configuration.

Figure 2 presents timings for $\texttt{Dec}_{\texttt{MP}}$, $\texttt{Dec}_{\texttt{RNS}}$ and $\texttt{Dec}_{\texttt{RNS-flp}}$, and Fig. 3 depicts timings for $\texttt{Mult}_{\texttt{MP}}$ and $\texttt{Mult}_{\texttt{RNS}}$. Both figures gather data for two modulus sizes: $\nu = 30$ and $\nu = 62$. Step 2 of $\texttt{Mult}_{\texttt{MP}}$ uses a decomposition in radix-base $\omega = 2^{32}$ when $\nu = 30$, and $\omega = 2^{62}$ when $\nu = 62$. The auxiliary bases $\mathcal{B}_{sk}$ and $\mathcal{B}'$ involved in $\texttt{Mult}_{\texttt{RNS}}$ and $\texttt{Mult}_{\texttt{MP}}$ contain $k + 1$ moduli each. Table 2 shows which values of k have been tested (depending on n). Multiplication timing for $(n, \nu, k) = (2^{11}, 62, 1)$ is not given since $L = 1$ already causes decryption failures.

$\log_2(n)$	11	12	13	14	15
k ($\nu = 30$)	3	6	13	26	53
k ($\nu = 62$)	1	3	6	12	25

Table 2: Parameter k used in the tests (i.e. $\lceil \log_2(q) \rceil = k\nu$).

It has to be noticed that the performance are mainly due to `NFLlib`. The contributions appear in the speed-ups between RNS and MP variants.

In Fig. 3, the convergence of complexities of $\texttt{Mult}_{\texttt{RNS}}$ and $\texttt{Mult}_{\texttt{MP}}$ (as explained in Sect. 4.7) is noticeable. The new algorithm presented in this paper allows speed-ups from 4.3 to 1.7 for degree n from 2^{11} to 2^{15} when $\nu = 30$, and from 3.6 to 1.9 for n from 2^{12} to 2^{15} when $\nu = 62$.

In Fig. 2, the two variants of decryption described in 3.5 are almost equally fast. Indeed, they perform the same number of elementary (floating point or integer) operations. Between degree 2^{11} and 2^{15}, the RNS variants allow speed-ups varying from 6.1 to 4.4 when $\nu = 30$, and from 20.4 to 5.6 when $\nu = 62$. All the implemented decryption functions take as input a ciphertext in `NTT` representation. Thus, only one `invNTT` is performed (after the product of residues) within each decryption. As explained (cf. 3.5), despite a better asymptotic computational complexity for RNS decryption, the efficiency remains in practice highly related to this `invNTT` procedure, even maybe justifying the slight convergence between MP and RNS decryption times observed in Fig. 2.

6 Conclusion

In this paper, the somewhat homomorphic encryption scheme `FV` has been fully adapted to Residue Number Systems. Prior to this work, RNS was used to accelerate polynomial additions and multiplications. However, the decryption and the homomorphic multiplication involve operations at the coefficient level which are hardly compatible with RNS, such as division and rounding.

Our proposed solutions overcome these incompatibilities, without modifying the security features of the original scheme. As a consequence, we have provided a SHE scheme which only involves RNS arithmetic. It means that only single-precision integer arithmetic is required, and the new variant fully benefits from the properties of RNS, such as parallelization.

The proposed scheme has been implemented in sotfware using C++. Because arithmetic on polynomials (in particular polynomial product) is not concerned by the new optimizations provided here, the implementation has been based on the `NFLlib` library, which embeds a very efficient `NTT`-based polynomial product. Our implementation has been compared to a classical version of `FV` (based on `NFLlib`, and `GMP`). For degrees from 2^{11} to 2^{15}, the new decryption (resp. homomorphic multiplication) offers speed-ups from 20 to 5 (resp. 4 to 2) folds for cryptographic parameters.

Further work should demonstrate the high potential of the new variant by exploiting all the concurrency properties of RNS, in particular through dedicated hardware implementations.

Acknowledgement

We thank anonymous reviewers for their helpful comments and remarks.

This work was partially supported by the European Union's H2020 Programme under grant agreement # ICT-644209, and by the Natural Sciences and Engineering Research Council (NSERC).

References

1. M. Albrecht, S. Bai, and L. Ducas. A Subfield Lattice Attack on Overstretched NTRU Assumptions: Cryptanalysis of some FHE and Graded Encoding Schemes.

Cryptology ePrint Archive, Report 2016/127, 2016. http://eprint.iacr.org/2016/127.
2. J.-C. Bajard, J. Eynard, M. A. Hasan, and V. Zucca. A Full RNS Variant of FV like Somewhat Homomorphic Encryption Schemes. Cryptology ePrint Archive, Report 2016/510, 2016. http://eprint.iacr.org/2016/510.
3. J.-C. Bajard, J. Eynard, N. Merkiche, and T. Plantard. RNS Arithmetic Approach in Lattice-Based Cryptography: Accelerating the "Rounding-off" Core Procedure. In *22nd IEEE Symposium on Computer Arithmetic, ARITH 2015, Lyon, France, June 22-24, 2015*, pages 113–120. IEEE, 2015.
4. J. W. Bos, K. E. Lauter, J. Loftus, and M. Naehrig. Improved Security for a Ring-Based Fully Homomorphic Encryption Scheme. In Martijn Stam, editor, *Cryptography and Coding - 14th IMA International Conference, IMACC 2013, Oxford, UK, December 17-19, 2013. Proceedings*, volume 8308 of *Lecture Notes in Computer Science*, pages 45–64. Springer, 2013.
5. Z. Brakerski. Fully Homomorphic Encryption without Modulus Switching from Classical GapSVP. Cryptology ePrint Archive, Report 2012/078, 2012. http://eprint.iacr.org/2012/078.
6. Z. Brakerski, C. Gentry, and V. Vaikuntanathan. (Leveled) Fully Homomorphic Encryption without Bootstrapping. In Shafi Goldwasser, editor, *Innovations in Theoretical Computer Science 2012, Cambridge, MA, USA, January 8-10, 2012*, pages 309–325. ACM, 2012.
7. L. Ducas, A. Durmus, T. Lepoint, and V. Lyubashevsky. Lattice Signatures and Bimodal Gaussians. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part I*, volume 8042 of *Lecture Notes in Computer Science*, pages 40–56. Springer, 2013.
8. L. Ducas, V. Lyubashevsky, and T. Prest. Efficient Identity-Based Encryption over NTRU Lattices. In Palash Sarkar and Tetsu Iwata, editors, *Advances in Cryptology - ASIACRYPT 2014 - 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, R.O.C., December 7-11, 2014, Proceedings, Part II*, volume 8874 of *Lecture Notes in Computer Science*, pages 22–41. Springer, 2014.
9. J. Fan and F. Vercauteren. Somewhat Practical Fully Homomorphic Encryption. Cryptology ePrint Archive, Report 2012/144, 2012. http://eprint.iacr.org/2012/144.
10. S. Garg, C. Gentry, and S. Halevi. Candidate Multilinear Maps from Ideal Lattices. In Thomas Johansson and Phong Q. Nguyen, editors, *Advances in Cryptology - EUROCRYPT 2013, 32nd Annual Intern. Conf. on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings*, volume 7881 of *Lecture Notes in Computer Science*, pages 1–17. Springer, 2013.
11. H. L. Garner. The Residue Number System. In *Papers Presented at the the March 3-5, 1959, Western Joint Computer Conference*, IRE-AIEE-ACM '59 (Western), pages 146–153, New York, NY, USA, 1959. ACM.
12. C. Gentry. Fully Homomorphic Encryption Using Ideal Lattices. In *Proceedings of the Forty-first Annual ACM Symposium on Theory of Computing*, STOC '09, pages 169–178, New York, NY, USA, 2009. ACM.
13. R. Gilad-Bachrach, N. Dowlin, K. Laine, K. E. Lauter, M. Naehrig, and J. Wernsing. CryptoNets: Applying Neural Networks to Encrypted Data with High Throughput and Accuracy. In Maria-Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of the 33nd International Conference on Machine Learning,*

ICML 2016, New York City, NY, USA, June 19-24, 2016, volume 48 of *JMLR Workshop and Conference Proceedings*, pages 201–210. JMLR.org, 2016.
14. T. Granlund and the GMP development team. *GNU MP: The GNU Multiple Precision Arithmetic Library*, 6.1.0 edition, 2015. `http://gmplib.org/`.
15. J. Hoffstein, J. Pipher, and J. H. Silverman. NTRU: A Ring-Based Public Key Cryptosystem. In Joe Buhler, editor, *Algorithmic Number Theory, Third Intern. Symp., ANTS-III, Portland, Oregon, USA, June 21-25, 1998, Proceedings*, volume 1423 of *Lecture Notes in Computer Science*, pages 267–288. Springer, 1998.
16. A. Langlois, D. Stehlé, and R. Steinfeld. GGHLite: More Efficient Multilinear Maps from Ideal Lattices. In Phong Q. Nguyen and Elisabeth Oswald, editors, *Advances in Cryptology - EUROCRYPT 2014 - 33rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Copenhagen, Denmark, May 11-15, 2014. Proceedings*, volume 8441 of *Lecture Notes in Computer Science*, pages 239–256. Springer, 2014.
17. T. Lepoint and M. Naehrig. A Comparison of the Homomorphic Encryption Schemes FV and YASHE. In David Pointcheval and Damien Vergnaud, editors, *Progress in Cryptology - AFRICACRYPT 2014 - 7th International Conference on Cryptology in Africa, Marrakesh, Morocco, May 28-30, 2014. Proceedings*, volume 8469 of *Lecture Notes in Computer Science*, pages 318–335. Springer, 2014.
18. V. Lyubashevsky. Lattice Signatures without Trapdoors. In *EUROCRYPT*, volume 7237 of *Lecture Notes in Computer Science*, pages 738–755. Springer, 2012.
19. C. Aguilar Melchor, J. Barrier, S. Guelton, A. Guinet, M.-O. Killijian, and T. Lepoint. NFLlib: NTT-Based Fast Lattice Library. In Kazue Sako, editor, *Topics in Cryptology - CT-RSA 2016 - The Cryptographers' Track at the RSA Conference 2016, San Francisco, CA, USA, February 29 - March 4, 2016, Proceedings*, volume 9610 of *Lecture Notes in Computer Science*, pages 341–356. Springer, 2016.
20. P. L. Montgomery. Modular Multiplication without Trial Division. *Mathematics of Computation*, 44(170):519–521, 1985.
21. T. Oder, T. Pöppelmann, and T. Güneysu. Beyond ECDSA and RSA: Lattice-based Digital Signatures on Constrained Devices. In *DAC*, pages 110:1–110:6. ACM, 2014.
22. C. Peikert. Lattice Cryptography for the Internet. In Michele Mosca, editor, *Post-Quantum Cryptography - 6th International Workshop, PQCrypto 2014, Waterloo, ON, Canada, October 1-3, 2014. Proceedings*, volume 8772 of *Lecture Notes in Computer Science*, pages 197–219. Springer, 2014.
23. S. S. Roy, K. Järvinen, F. Vercauteren, V. S. Dimitrov, and I. Verbauwhede. Modular Hardware Architecture for Somewhat Homomorphic Function Evaluation. In Tim Güneysu and Helena Handschuh, editors, *Cryptographic Hardware and Embedded Systems - CHES 2015 - 17th International Workshop, Saint-Malo, France, September 13-16, 2015, Proceedings*, volume 9293 of *Lecture Notes in Computer Science*, pages 164–184. Springer, 2015.
24. A. P. Shenoy and R. Kumaresan. Fast Base Extension Using a Redundant Modulus in RNS. *IEEE Trans. Computers*, 38(2):292–297, 1989.
25. D. Stehlé, R. Steinfeld, K. Tanaka, and K. Xagawa. Efficient Public Key Encryption Based on Ideal Lattices. In Mitsuru Matsui, editor, *Advances in Cryptology - ASIACRYPT 2009, 15th Intern. Conf. on the Theory and Application of Cryptology and Information Security, Tokyo, Japan, December 6-10, 2009. Proceedings*, volume 5912 of *Lecture Notes in Computer Science*, pages 617–635. Springer, 2009.
26. European Union. Homomorphic Encryption, Applications and Technology (HEAT). `https://heat-project.eu`. H2020-ICT-2014-1, Project reference: 644209.

A Proofs

A.1 Lemma 1

According to (2), $\texttt{FastBconv}(|t\gamma \cdot \texttt{ct}(\boldsymbol{s})|_q, q, \{t, \gamma\})$ provides $|t\gamma \cdot \texttt{ct}(\boldsymbol{s})|_q + q\boldsymbol{a}$, where each coefficient $\boldsymbol{a}_i$ is an integer lying in $[0, k-1]$. Let m be t or γ. Then,

$$\begin{aligned}
&\texttt{FastBconv}(|t\gamma \cdot \texttt{ct}(\boldsymbol{s})|_q, q, \{t, \gamma\}) \times |-q^{-1}|_m \bmod m \\
&= (|t\gamma \cdot \texttt{ct}(\boldsymbol{s})|_q + q\boldsymbol{a}) \times |-q^{-1}|_m \bmod m \\
&= \frac{t\gamma[\texttt{ct}(\boldsymbol{s})]_q - |t\gamma \cdot \texttt{ct}(\boldsymbol{s})|_q - q\boldsymbol{a}}{q} \bmod m \text{ (exact division)} \\
&= \left(\left\lfloor \frac{t\gamma[\texttt{ct}(\boldsymbol{s})]_q}{q} \right\rfloor - \boldsymbol{a} \right) \bmod m \\
&= \left(\left\lfloor \frac{t\gamma[\texttt{ct}(\boldsymbol{s})]_q}{q} \right\rceil - \boldsymbol{e} \right) \bmod m
\end{aligned}$$

where $\boldsymbol{e}_i \in \{\boldsymbol{a}_i, \boldsymbol{a}_i + 1\}$, i.e. $\boldsymbol{e}_i \in [0, k]$. To conclude the proof, it suffices to use the equality $\Delta t = q - |q|_t$. That way, one can write $t[\texttt{ct}(\boldsymbol{s})]_q = q([\boldsymbol{m}]_t + t\boldsymbol{r}) + \boldsymbol{v_c}$, and the second equality of (3) follows.

A.2 Lemma 2

By hypothesis, we have $-\gamma(\frac{1}{2}-\varepsilon)-k \leqslant (\gamma\frac{\boldsymbol{v_c}}{q}-\boldsymbol{e})_i \leqslant \gamma(\frac{1}{2}-\varepsilon)$ for any $i \in [0, n-1]$. To have $[\lfloor \gamma\frac{\boldsymbol{v_c}}{q} \rceil - \boldsymbol{e}]_\gamma = \lfloor \gamma\frac{\boldsymbol{v_c}}{q} \rceil - \boldsymbol{e}$, we then require that $-\lfloor \frac{\gamma}{2} \rfloor - \frac{1}{2} \leqslant (\gamma\frac{\boldsymbol{v_c}}{q} - \boldsymbol{e})_i < \lfloor \frac{\gamma-1}{2} \rfloor + \frac{1}{2}$. Hence, a sufficient condition is given by:

$$\begin{cases} \gamma(\frac{1}{2} - \varepsilon) < \lfloor \frac{\gamma-1}{2} \rfloor + \frac{1}{2} \\ -\lfloor \frac{\gamma}{2} \rfloor - \frac{1}{2} \leqslant -\gamma(\frac{1}{2} - \varepsilon) - k \end{cases} \Leftrightarrow (\gamma \text{ odd}) \begin{cases} \gamma\varepsilon > 0 \\ \gamma\varepsilon \geqslant k \end{cases}, (\gamma \text{ even}) \begin{cases} \gamma\varepsilon > \frac{1}{2} \\ \gamma\varepsilon \geqslant k - \frac{1}{2} \end{cases} .$$

A.3 Theorem 1

According to Lem. 2, the γ-correction technique works as long as $\gamma(\frac{1}{2} - \frac{\|\boldsymbol{v_c}\|_\infty}{q}) \geqslant k \Leftrightarrow \|\boldsymbol{v_c}\|_\infty \leqslant q(\frac{1}{2} - \frac{k}{\gamma})$. Moreover, $\|\boldsymbol{v_c}\|_\infty = \|t\boldsymbol{v} - |q|_t[\boldsymbol{m}]_t\|_\infty \leqslant t\|\boldsymbol{v}\|_\infty + |q|_t\frac{t}{2}$. Then, the bound (5) follows. The lower bound on γ guarantees that the bound (5) for the noise is positive.

A.4 Lemma 3

Let's denote $\tilde{\boldsymbol{v}}_{\boldsymbol{c}} := \lfloor \gamma\frac{\boldsymbol{v_c}}{q} \rceil - \boldsymbol{e}$. By computing (3) modulo γt, we obtain $\boldsymbol{z} = |\gamma[\boldsymbol{m}]_t + \tilde{\boldsymbol{v}}_{\boldsymbol{c}}|_{\gamma t}$. We notice that $\boldsymbol{z} = |\gamma|\boldsymbol{m}|_t + \tilde{\boldsymbol{v}}_{\boldsymbol{c}}|_{\gamma t}$. Indeed, $\gamma|\boldsymbol{m}|_t = \gamma([\boldsymbol{m}]_t + t\boldsymbol{a})$, where $\boldsymbol{a}_i \in \{0, 1\}$. Thus, $\gamma t\boldsymbol{a}$ vanishes modulo γt. More generally, $|\gamma|\boldsymbol{m}|_t + \tilde{\boldsymbol{v}}_{\boldsymbol{c}}|_{\gamma t} = \gamma t\boldsymbol{b} + \gamma|\boldsymbol{m}|_t + \tilde{\boldsymbol{v}}_{\boldsymbol{c}}$ with $\boldsymbol{b}_i \in \{0, 1\}$ ($\boldsymbol{b}_i = 1 \Leftrightarrow ((\gamma|\boldsymbol{m}|_t)_i = 0$ and $(\tilde{\boldsymbol{v}}_{\boldsymbol{c}})_i < 0)$).

The integer γ has been chosen such that each coefficient of $\tilde{\boldsymbol{v}}_{\boldsymbol{c}}$ lies in $[-\gamma/2, \gamma/2)$. Thus, it is clear that any coefficient of $\tilde{\boldsymbol{v}}_{\boldsymbol{c}} + \frac{\gamma}{2}\boldsymbol{v}_1$ lies in $[0, \gamma)$.

Consequently, we have $\boldsymbol{z} + \frac{\gamma}{2}\boldsymbol{v}_1 = \gamma t\boldsymbol{b} + \gamma|\boldsymbol{m}|_t + \boldsymbol{w}$ where $\boldsymbol{w}_i \in [0, \gamma)$ and $\boldsymbol{b}_i \geqslant 0$ for any i. Furthermore, it can be noticed that $|\boldsymbol{z} + \frac{\gamma}{2}\boldsymbol{v}_1|_{\gamma t} = \gamma|\boldsymbol{m}|_t + \boldsymbol{w}$. Finally, since γ is a power of 2, we have $(\boldsymbol{z} + \frac{\gamma}{2}\boldsymbol{v}_1) \gg \log_2(\gamma) = t\boldsymbol{b} + |\boldsymbol{m}|_t$ (and also that $|\boldsymbol{z} + \frac{\gamma}{2}\boldsymbol{v}_1|_{\gamma t} \gg \log_2(\gamma) = |\boldsymbol{m}|_t$). By applying a centered reduction modulo t, (6) follows.

A.5 Lemma 4

Algorithm 2 performs in $\mathcal{B}_{\mathtt{sk}}$ the computation of $\frac{[\boldsymbol{c}\tilde{m}]_q + q\boldsymbol{u} + q[-([\boldsymbol{c}\tilde{m}]_q + q\boldsymbol{u})/q]_{\tilde{m}}}{\tilde{m}}$. This quantity is clearly congruent to $\boldsymbol{c}$ modulo q. In accordance with hypothesis (8), its norm is bounded by $\frac{q(1/2+\tau+\tilde{m}/2)}{\tilde{m}} \leqslant \frac{q}{2}(1+\rho)$.

A.6 Lemma 5

We recall that $\mathtt{FastBconv}(|t \cdot \mathtt{ct}'_\star[j]|_q, q, \mathcal{B}_{\mathtt{sk}})$ outputs $|t \cdot \mathtt{ct}'_\star[j]|_q + q\boldsymbol{u}$ with $\|\boldsymbol{u}\|_\infty \leqslant k-1$. Then, the proof is complete by using the general equalities $\frac{x-|x|_q}{q} = \lfloor \frac{x}{q} \rfloor = \lfloor \frac{x}{q} \rceil + \tau$, $\tau \in \{-1, 0\}$.

A.7 Proposition 1

The following noise analysis is inspired from the one provided in [4]. Some of the tools and bounds from there are re-used here. In the following, we write $\mathtt{ct}'_i = (\boldsymbol{c}'_{i,0}, \boldsymbol{c}'_{i,1})$. In particular, we have $\mathtt{ct}'_\star = (\boldsymbol{c}'_{1,0}\boldsymbol{c}'_{2,0}, \boldsymbol{c}'_{1,1}\boldsymbol{c}'_{2,0} + \boldsymbol{c}'_{1,0}\boldsymbol{c}'_{2,1}, \boldsymbol{c}'_{1,1}\boldsymbol{c}'_{2,1})$. By hypothesis, each $\boldsymbol{c}'_{i,j}$ satisfies $\|\boldsymbol{c}'_{i,j}\|_\infty \leqslant \frac{q}{2}(1+\rho)$. In particular, the bound in (9) comes from the fact that $\|\boldsymbol{c}'_{1,1}\boldsymbol{c}'_{2,0} + \boldsymbol{c}'_{1,0}\boldsymbol{c}'_{2,1}\|_\infty \leqslant \delta\frac{q^2}{2}(1+\rho)^2$.

Since $\mathtt{ct}'_i = \mathtt{ct}_i \bmod q$ and $\|\mathtt{ct}'_i\|_\infty \leqslant \frac{q}{2}(1+\rho)$, and by using $\|\boldsymbol{s}\|_\infty \leqslant B_{key}$, $\|\boldsymbol{v}_i\|_\infty < \frac{\Delta}{2} < \frac{q}{2t}$, $\|\Delta[\boldsymbol{m}]_t\|_\infty < \frac{t\Delta}{2} < \frac{q}{2}$ and $t \geqslant 2$, we can write:

$$\mathtt{ct}'_i(\boldsymbol{s}) = \boldsymbol{c}'_{i,0} + \boldsymbol{c}'_{i,1}\boldsymbol{s} = \Delta[\boldsymbol{m}_i]_t + \boldsymbol{v}_i + q\boldsymbol{r}_i$$

with $\|\boldsymbol{r}_i\|_\infty < r_\infty := \frac{1+\rho}{2}(1+\delta B_{key})+1$. For our purpose, we use some convenient notations and bounds from [4], but applicable to the present context:

$$\begin{cases} \boldsymbol{v}_1\boldsymbol{v}_2 = [\boldsymbol{v}_1\boldsymbol{v}_2]_\Delta + \Delta\boldsymbol{r}_v, \ \|\boldsymbol{r}_v\|_\infty < \frac{\delta}{2}\min\|\boldsymbol{v}_i\|_\infty + \frac{1}{2} \ , \\ [\boldsymbol{m}_1]_t\,[\boldsymbol{m}_2]_t = [\boldsymbol{m}_1\boldsymbol{m}_2]_t + t\boldsymbol{r}_m, \ \|\boldsymbol{r}_m\|_\infty < \frac{1}{2}\delta t \ . \end{cases} \tag{21}$$

By noticing that $\mathtt{ct}'_\star(\boldsymbol{s}) = (\mathtt{ct}'_1 \star \mathtt{ct}'_2)(\boldsymbol{s}) = \mathtt{ct}'_1(\boldsymbol{s}) \times \mathtt{ct}'_2(\boldsymbol{s})$, we obtain:

$$\begin{aligned} \mathtt{ct}'_\star(\boldsymbol{s}) = {} & \Delta^2[\boldsymbol{m}_1]_t[\boldsymbol{m}_2]_t + \Delta([\boldsymbol{m}_1]_t\boldsymbol{v}_2 + [\boldsymbol{m}_2]_t\boldsymbol{v}_1) \\ & + q\Delta([\boldsymbol{m}_1]_t\boldsymbol{r}_2 + [\boldsymbol{m}_2]_t\boldsymbol{r}_1) + \boldsymbol{v}_1\boldsymbol{v}_2 + q^2\boldsymbol{r}_1\boldsymbol{r}_2 + q(\boldsymbol{v}_1\boldsymbol{r}_2 + \boldsymbol{v}_2\boldsymbol{r}_1) \ . \end{aligned}$$

Then, by using (21) and $\Delta t = q - |q|_t$, we deduce that:

$$\begin{aligned} \tfrac{t}{q}\mathtt{ct}'_\star(\boldsymbol{s}) = {} & \Delta[\boldsymbol{m}_1\boldsymbol{m}_2]_t + q\,(\boldsymbol{r}_m + [\boldsymbol{m}_1]_t\boldsymbol{r}_2 + [\boldsymbol{m}_2]_t\boldsymbol{r}_1 + t\boldsymbol{r}_1\boldsymbol{r}_2) \\ & - \tfrac{|q|_t\Delta}{q}[\boldsymbol{m}_1\boldsymbol{m}_2]_t + (\tfrac{|q|_t}{q} - 2)|q|_t\boldsymbol{r}_m \\ & + (1 - \tfrac{|q|_t}{q})([\boldsymbol{m}_1]_t\boldsymbol{v}_2 + [\boldsymbol{m}_2]_t\boldsymbol{v}_1) - |q|_t([\boldsymbol{m}_1]_t\boldsymbol{r}_2 + [\boldsymbol{m}_2]_t\boldsymbol{r}_1) \\ & + \tfrac{t}{q}[\boldsymbol{v}_1\boldsymbol{v}_2]_\Delta + (1 - \tfrac{|q|_t}{q})\boldsymbol{r}_v + t(\boldsymbol{v}_1\boldsymbol{r}_2 + \boldsymbol{v}_2\boldsymbol{r}_1) \ . \end{aligned}$$

Thus,

$$\widetilde{\mathtt{ct}}_{mult}(\boldsymbol{s}) = \mathtt{DR}_2(\mathtt{ct}'_\star)(\boldsymbol{s}) = \tfrac{t}{q}\mathtt{ct}'_\star(\boldsymbol{s}) + \boldsymbol{r}_a = \Delta\,[\boldsymbol{m}_1\boldsymbol{m}_2]_t + \tilde{\boldsymbol{v}}_{mult} \bmod q$$

with $\boldsymbol{r}_a := (\mathtt{DR}_2(\mathtt{ct}'_\star) - \frac{t}{q}\mathtt{ct}'_\star)(\boldsymbol{s}) = \sum_{i=0}^{2}\left(\lfloor \frac{t}{q}\mathtt{ct}'_\star[i]\rceil - \frac{t}{q}\mathtt{ct}'_\star[i]\right)\boldsymbol{s}^i$ and

$$\begin{aligned}\tilde{\boldsymbol{v}}_{mult} := \boldsymbol{r}_a &- \tfrac{|q|_t\Delta}{q}[\boldsymbol{m}_1\boldsymbol{m}_2]_t + (\tfrac{|q|_t}{q} - 2)|q|_t\boldsymbol{r}_m \\ &+(1 - \tfrac{|q|_t}{q})([\boldsymbol{m}_1]_t\boldsymbol{v}_2 + [\boldsymbol{m}_2]_t\boldsymbol{v}_1) - |q|_t([\boldsymbol{m}_1]_t\boldsymbol{r}_2 + [\boldsymbol{m}_2]_t\boldsymbol{r}_1) \\ &+\tfrac{t}{q}[\boldsymbol{v}_1\boldsymbol{v}_2]_\Delta + (1 - \tfrac{|q|_t}{q})\boldsymbol{r}_v + t(\boldsymbol{v}_1\boldsymbol{r}_2 + \boldsymbol{v}_2\boldsymbol{r}_1)\ .\end{aligned} \tag{22}$$

Below, some useful bounds are given.

$$\begin{cases} \|\,[\boldsymbol{m}_1]_t\,\boldsymbol{v}_2 + [\boldsymbol{m}_2]_t\,\boldsymbol{v}_1\|_\infty \leqslant \frac{\delta t}{2}(\|\boldsymbol{v}_1\|_\infty + \|\boldsymbol{v}_2\|_\infty)\ , \\ \|\boldsymbol{v}_1\boldsymbol{r}_2 + \boldsymbol{v}_2\boldsymbol{r}_1\|_\infty < \delta r_\infty(\|\boldsymbol{v}_1\|_\infty + \|\boldsymbol{v}_2\|_\infty)\ . \end{cases}$$

Next, we set a bound for each term of (22), then it suffices to put them all together to obtain (11).

$$\begin{aligned} \|\boldsymbol{r}_a\|_\infty &\leqslant \tfrac{1}{2}(1 + \delta B_{key} + \delta^2 B_{key}^2)\ , \\ \|-\tfrac{|q|_t\Delta}{q}[\boldsymbol{m}_1\boldsymbol{m}_2]_t + \tfrac{t}{q}[\boldsymbol{v}_1\boldsymbol{v}_2]_\Delta\|_\infty &\leqslant \tfrac{|q|_t\Delta t}{2q} + \tfrac{t\Delta}{2q} < \tfrac{1}{2}(|q|_t + 1)\ , \\ \|(\tfrac{|q|_t}{q} - 2)|q|_t\boldsymbol{r}_m\|_\infty &\leqslant 2|q|_t\|\boldsymbol{r}_m\|_\infty < \delta t|q|_t\ , \\ \|(1 - \tfrac{|q|_t}{q})\boldsymbol{r}_v\|_\infty &\leqslant \|\boldsymbol{r}_v\|_\infty < \tfrac{\delta}{2}\min\|\boldsymbol{v}_i\|_\infty + \tfrac{1}{2}\ , \\ \|t(\boldsymbol{v}_1\boldsymbol{r}_2 + \boldsymbol{v}_2\boldsymbol{r}_1)\|_\infty &< \delta t r_\infty(\|\boldsymbol{v}_1\|_\infty + \|\boldsymbol{v}_2\|_\infty)\ , \\ \|(1 - \tfrac{|q|_t}{q})([\boldsymbol{m}_1]_t\boldsymbol{v}_2 + [\boldsymbol{m}_2]_t\boldsymbol{v}_1)\|_\infty &< \tfrac{\delta t}{2}(\|\boldsymbol{v}_1\|_\infty + \|\boldsymbol{v}_2\|_\infty)\ , \\ \|-|q|_t([\boldsymbol{m}_1]_t\boldsymbol{r}_2 + [\boldsymbol{m}_2]_t\boldsymbol{r}_1)\|_\infty &< |q|_t\delta t r_\infty\ . \end{aligned}$$

A.8 Lemma 6

We set $\mathcal{B} = \{m_1, \ldots, m_\ell\}$ ($|\mathcal{B}| = \ell$). The case $x \geqslant 0$ and $\lambda = 1$ is the classical case of Shenoy and Kumaresan's conversion. We recall that, by definition, $\mathtt{FastBconv}(x, \mathcal{B}, \cdot) = \sum_{i=1}^{\ell} |x\frac{m_i}{M}|_{m_i}\frac{M}{m_i}$. There exists an integer $0 \leqslant \alpha \leqslant \ell - 1$ such that $\sum_{i=1}^{\ell} |x\frac{m_i}{M}|_{m_i}\frac{M}{m_i} = x + \alpha M$. By inverting this equality modulo $m_{\mathtt{sk}}$, it would suffice, in this case, that $m_{\mathtt{sk}} > \ell$ to enable us to recover α by noticing that $\alpha = |\alpha|_{m_{\mathtt{sk}}} = |\alpha_{\mathtt{sk},x}|_{m_{sk}}$.

Let's consider the general case $|x| < \lambda M$. Here, the possible negativity of x is the reason why we have to compute a centered remainder modulo m_{sk} in (12).

First, we notice that the residues of x in $\mathcal{B}$ are actually those of $|x|_M = x + \mu M \in [0, M)$, where μ is an integer lying in $[-\lfloor\lambda\rfloor, \lceil\lambda\rceil]$. Therefore, we deduce that

$$\textstyle\sum_{i=1}^{\ell} |x\frac{m_i}{M}|_{m_i}\frac{M}{m_i} = |x|_M + \alpha_{\mathtt{sk},|x|_M}M = (x + \mu M) + \alpha_{\mathtt{sk},(x+\mu M)}M$$

with $0 \leqslant \alpha_{\mathtt{sk},(x+\mu M)} \leqslant \ell - 1$. Denoting $|x|_{m_{\mathtt{sk}}}$ by $x_{\mathtt{sk}}$, it follows that the quantity computed in (12) is the following one:

$$\begin{aligned}[(\textstyle\sum_{i=1}^{\ell} |x\frac{m_i}{M}|_{m_i}\frac{M}{m_i} - x_{\mathtt{sk}})M^{-1}]_{m_{\mathtt{sk}}} &= [(\textstyle\sum_{i=1}^{\ell} |x\frac{m_i}{M}|_{m_i}\frac{M}{m_i} - (x_{\mathtt{sk}} + \mu M))M^{-1} + \mu]_{m_{\mathtt{sk}}} \\ &= [\alpha_{\mathtt{sk},(x+\mu M)} + \mu]_{m_{\mathtt{sk}}}\ .\end{aligned}$$

It remains to show that $[\alpha_{\mathtt{sk},(x+\mu M)}+\mu]_{m_{\mathtt{sk}}} = \alpha_{\mathtt{sk},(x+\mu M)}+\mu$ or, in other words, that $-\lfloor\frac{m_{sk}}{2}\rfloor \leqslant \alpha_{\mathtt{sk},(x+\mu M)}+\mu \leqslant \lfloor\frac{m_{sk}-1}{2}\rfloor$. But by hypothesis on $m_{\mathtt{sk}}$, and because $\ell \geqslant 1$, we can write $m_{\mathtt{sk}} \geqslant 2(\ell+\lceil\lambda\rceil) \geqslant 2\lfloor\lambda\rfloor+1$. Then,

$$\begin{cases} \alpha_{\mathtt{sk},(x+\mu M)}+\mu \leqslant \ell-1+\lceil\lambda\rceil \leqslant \frac{m_{sk}}{2}-1 \leqslant \lfloor\frac{m_{sk}-1}{2}\rfloor \ , \\ \alpha_{\mathtt{sk},(x+\mu M)}+\mu \geqslant -\lfloor\lambda\rfloor \geqslant -\frac{m_{sk}-1}{2} \geqslant -\lfloor\frac{m_{sk}}{2}\rfloor \ . \end{cases}$$

Thus, $[\alpha_{\mathtt{sk},(x+\mu M)}+\mu]_{m_{\mathtt{sk}}} = \alpha_{\mathtt{sk},(x+\mu M)}+\mu$, and it follows that, by computing the right member of (13), we obtain the following equalities:

$$\begin{aligned} \sum_{i=1}^{\ell} |x\tfrac{m_i}{M}|_{m_i}\tfrac{M}{m_i} - [\alpha_{\mathtt{sk},(x+\mu M)}+\mu]_{m_{\mathtt{sk}}}M &= (x+\mu M)+\alpha_{\mathtt{sk},(x+\mu M)}M \\ &\quad -(\alpha_{\mathtt{sk},(x+\mu M)}+\mu)M \\ &= x \ . \end{aligned}$$

A.9 Proposition 2

By using (9), we have $\|\mathtt{DR}_2(\mathtt{ct}'_\star)\|_\infty \leqslant \|\frac{t}{q}\mathtt{ct}'_\star\|_\infty + \frac{1}{2} \leqslant \delta t\frac{q}{2}(1+\rho)^2+\frac{1}{2}$. Lemma 6 concludes the proof.

A.10 Lemma 8

First, we have

$$\big(\langle\xi_q(\boldsymbol{c}), -(\overrightarrow{\boldsymbol{e}}+\overrightarrow{\boldsymbol{a}}\boldsymbol{s})\rangle + \boldsymbol{s}\langle\xi_q(\boldsymbol{c}), \overrightarrow{\boldsymbol{a}}\rangle\big) \bmod q = -\langle\xi_q(\boldsymbol{c}), \overrightarrow{\boldsymbol{e}}\rangle \bmod q \ .$$

Second, by using $q_i < 2^\nu$, we obtain

$$\begin{aligned} \|\langle\xi_q(\boldsymbol{c}), \overrightarrow{\boldsymbol{e}}\rangle\|_\infty = \|\textstyle\sum_{i=1}^k |\boldsymbol{c}\frac{q_i}{q}|_{q_i}\boldsymbol{e}_i\|_\infty < \sum_{i=1}^k \delta q_i\|\boldsymbol{e}_i\|_\infty &\leqslant \delta B_{err}\textstyle\sum_{i=1}^k q_i \\ &< \delta B_{err}k2^\nu \ . \end{aligned}$$

A.11 Proposition 3

At this point, we are evaluating $\mathtt{Relin}_{\mathtt{RNS}}(\widetilde{\mathtt{ct}}_{mult}+\mathtt{b})$. We denote $\widetilde{\mathtt{ct}}_{mult} = (\boldsymbol{c}_0, \boldsymbol{c}_1, \boldsymbol{c}_2)$. A first remark is that we can write

$$\xi_q(\boldsymbol{c}_2+\boldsymbol{b}_2) = \xi_q(\boldsymbol{c}_2)+\xi_q(\boldsymbol{b}_2)-\overrightarrow{\boldsymbol{u}}$$

where $\overrightarrow{\boldsymbol{u}} = (\boldsymbol{u}_1, \ldots, \boldsymbol{u}_k)$ is such that, for any $(i,j) \in [1,k]\times[0,n-1]$, $(\boldsymbol{u}_i)_j \in \{0, q_i\}$. In particular, it can be noticed that $\||\boldsymbol{b}_2\frac{q_i}{q}|_{q_i} - \boldsymbol{u}_i\|_\infty < q_i$, and that $\langle\overrightarrow{\boldsymbol{u}}, (\boldsymbol{s}^2\frac{q}{q_1}, \ldots, \boldsymbol{s}^2\frac{q}{q_k})\rangle \equiv 0 \bmod q$. This latter fact leads to the following equality:

$$\langle\overrightarrow{\boldsymbol{u}}, \mathtt{rlk}_{\mathtt{RNS}}[0]\rangle + \langle\overrightarrow{\boldsymbol{u}}, \mathtt{rlk}_{\mathtt{RNS}}[1]\rangle\boldsymbol{s} = \langle\overrightarrow{\boldsymbol{u}}, \overrightarrow{\boldsymbol{e}}\rangle = \textstyle\sum_{i=1}^k \boldsymbol{u}_i\boldsymbol{e}_i \ (\mathrm{mod} q) \ .$$

Consequently, we have that, in $\mathcal{R}_q \times \mathcal{R}_q$,

$$\begin{aligned} \mathtt{Relin}_{\mathtt{RNS}}(\widetilde{\mathtt{ct}}_{mult}+\mathtt{b}) = \mathtt{Relin}_{\mathtt{RNS}}(\widetilde{\mathtt{ct}}_{mult}) &+ \mathtt{Relin}_{\mathtt{RNS}}(\mathtt{b}) \\ &-(\langle\overrightarrow{\boldsymbol{u}}, \mathtt{rlk}_{\mathtt{RNS}}[0]\rangle, \langle\overrightarrow{\boldsymbol{u}}, \mathtt{rlk}_{\mathtt{RNS}}[1]\rangle) \ . \end{aligned}$$

Thus, a part of the noise comes from the following extra term:

$$\begin{aligned}
&\left\|\texttt{Relin}_{\texttt{RNS}}(\texttt{b})(\boldsymbol{s}) - \langle\overrightarrow{\boldsymbol{u}}, \texttt{rlk}_{\texttt{RNS}}[0]\rangle - \langle\overrightarrow{\boldsymbol{u}}, \texttt{rlk}_{\texttt{RNS}}[1]\rangle\boldsymbol{s} \pmod q\right\|_\infty \\
= &\left\|\boldsymbol{b}_0 + \langle\xi_q(\boldsymbol{b}_2) - \overrightarrow{\boldsymbol{u}}, \texttt{rlk}_{\texttt{RNS}}[0]\rangle + \boldsymbol{s}(\boldsymbol{b}_1 + \langle\xi_q(\boldsymbol{b}_2) - \overrightarrow{\boldsymbol{u}}, \texttt{rlk}_{\texttt{RNS}}[1]\rangle) \pmod q\right\|_\infty \\
= &\left\|\boldsymbol{b}_0 + \boldsymbol{b}_1\boldsymbol{s} + \boldsymbol{b}_2\boldsymbol{s}^2 - \textstyle\sum_{i=1}^k(|\boldsymbol{b}_2\frac{q_i}{q}|_{q_i} - \boldsymbol{u}_i)\boldsymbol{e}_i \pmod q\right\|_\infty \\
< &\ k(1 + \delta B_{key} + \delta^2 B_{key}^2) + \delta B_{err}\textstyle\sum_{i=1}^k q_i \\
< &\ k(1 + \delta B_{key} + \delta^2 B_{key}^2) + \delta B_{err} k 2^\nu \ .
\end{aligned}$$

Lemma 8 brings the rest of the noise.

B Additional elements about RNS homomorphic multiplication

B.1 Combining two levels of decomposition within step 2

To reduce the noise growth due to the relinearisation step a bit more, we can integrate another level of decomposition in radix ω where $\omega = 2^\theta << 2^\nu$ as efficiently as in the original scheme by doing it on the residues, because they are handled through the classical binary positional system. By denoting $\ell_{\omega,2^\nu} = \lceil\frac{\nu}{\theta}\rceil$, each polynomial $|\boldsymbol{c}\frac{q_i}{q}|_{q_i}$ is decomposed into the vector of polynomials $([\lfloor|\boldsymbol{c}\frac{q_i}{q}|_{q_i}\omega^{-z}\rfloor]_\omega)_{z\in[0,\ldots,\ell_{\omega,2^\nu}-1]}$, and the new decomposition function is defined by:

$$\mathcal{D}_{\texttt{RNS},\omega,q}(\boldsymbol{c}) = \left(\boldsymbol{d}_i^z = \left[\left\lfloor\left|\boldsymbol{c}\frac{q_i}{q}\right|_{q_i}\omega^{-z}\right\rfloor\right]_w\right)_{i\in[1,k],z\in[0,\ldots,\ell_{\omega,2^\nu}-1]} .$$

Therefore, each term $|\boldsymbol{s}^2\frac{q}{q_i} - (\boldsymbol{e}_i + \boldsymbol{s}\boldsymbol{a}_i)|_{q_j}$ in $\texttt{rlk}_{\texttt{RNS}}[0]$ has to be replaced by

$$\left(|\boldsymbol{s}^2\frac{q}{q_i}\omega^z - (\boldsymbol{e}_i^z + \boldsymbol{s}\boldsymbol{a}_i^z)|_{q_j}\right)_z,\ z = 0,\ldots,\ell_{\omega,2^\nu}-1,\ \boldsymbol{e}_i^z \leftarrow \chi_{err}, \boldsymbol{a}_i^z \leftarrow \mathcal{U}(\mathcal{R}_q) \ .$$

It follows that the extra noise is now bounded by:

$$\|\langle\mathcal{D}_{\texttt{RNS},\omega,q}(\boldsymbol{c}), \overrightarrow{\boldsymbol{e}}\rangle\|_\infty = \|\textstyle\sum_{i=1}^k\sum_{z=0}^{\ell_{\omega,2^\nu}-1}\boldsymbol{d}_i^z\boldsymbol{e}_i^z\|_\infty < \delta B_{err}\omega k\ell_{\omega,2^\nu} \ .$$

In other words, the term 2^ν in (16) is replaced by $\omega\ell_{\omega,2^\nu}$. Even if such optimization may be not really relevant in practice, this is another argument for saying that any trick and technique of the original scheme can be efficiently adapted to a pure RNS approach.

B.2 Some details about complexity

We analyse the cost of a multi-precision variant, in order to estimate the benefits of the new RNS variant of multiplication in terms of computational cost.

The product $\mathtt{ct}_\star = \mathtt{ct}_1 \star \mathtt{ct}_2 = (\boldsymbol{c}_{1,0}, \boldsymbol{c}_{1,1}) \star (\boldsymbol{c}_{2,0}, \boldsymbol{c}_{2,1})$ in MP variant is advantageously performed in RNS, in order to benefit from NTT. So, the MP variant considered here is assumed to use a second base $\mathcal{B}'$ such that $qM' > \|\mathtt{ct}_\star\|_\infty$. By taking centered remainders modulo q, we consider $\|\mathtt{ct}_i\|_\infty \leqslant \frac{q}{2}$. Then $\mathcal{B}'$ must verify in particular that $\|\mathtt{ct}_\star[1] = \boldsymbol{c}_{1,0}\boldsymbol{c}_{2,1} + \boldsymbol{c}_{1,1}\boldsymbol{c}_{2,0}\|_\infty \leqslant 2\delta\frac{q^2}{4} < qM'$. Thus, $|\mathcal{B}'|$ has to be at least equal to $k+1$ (notice that, in RNS variant, we also need to have $|\mathcal{B}_{sk}| = k+1$ for dealing with the multiplication).

The conversion, from q to $\mathcal{B}'$, of each $\mathtt{ct}_i$ has to be as exact as possible in order to reduce the noise growth. It can be done by computing $[\sum_{i=1}^{k} |\boldsymbol{c}_{a,b}|_{q_i} \times (|\frac{q_i}{q}|_{q_i}\frac{q}{q_i})]_q$ in $\mathcal{B}'$. The terms $(|\frac{q_i}{q}|_{q_i}\frac{q}{q_i})$ are precomputable and their size is k words ($\log_2(q)$ bits). Thus, the sum involves $k^2 n\mathtt{EM}$. The reduction modulo q can be performed by using an efficient reduction as described in [19], reducing to around 2 multiplications of k-word integers, that is $\mathcal{O}(k^{1+\varepsilon})n\mathtt{EM}$ (where ε stands for complexity of multi-precision multiplication in radix-base 2^ν; e.g. $\varepsilon = 1$ for the schoolbook multiplication). Next, the k-word value is reduced modulo each 1-word element of $\mathcal{B}'$, through around $2kn\mathtt{EM}$ for the whole set of coefficients. Finally, this procedure has to be made four times. Its total cost is around $(4k^2 + \mathcal{O}(k^{1+\varepsilon}))n\mathtt{EM}$.

Next, the product $\mathtt{ct}_1 \star \mathtt{ct}_2$ is done in $q \cup \mathcal{B}'$. First, $4(2k+1)\mathtt{NTT}$ are applied. Second, by using a Karatsuba like trick, the product is achieved by using only $3\times(2k+1)n\mathtt{IMM}$. Third, $3(2k+1)\mathtt{invNTT}$ are applied to recover $\mathtt{ct}_\star = (\boldsymbol{c}_{\star,0}, \boldsymbol{c}_{\star,1}, \boldsymbol{c}_{\star,2})$ in coefficient representation.

The next step is the division and rounding of the three polynomials $\boldsymbol{c}_{\star,i}$'s. A lift from $q \cup \mathcal{B}'$ to $\mathbb{Z}$ is required, for a cost of $3(2k+1)^2 n\mathtt{EM}$. $\frac{t}{q}$ can be precomputed with around $3k+1$ words of precision to ensure a correct rounding. Thus, a product $\frac{t}{q} \times \boldsymbol{c}_{\star,i}$ is achieved with $\mathcal{O}(k^{1+\varepsilon})n\mathtt{EM}$. After, the rounding of $\boldsymbol{c}_{\star,0}$ and $\boldsymbol{c}_{\star,1}$ are reduced in RNS base q by $2 \times 2kn\mathtt{EM}$.

The relinearisation step (15) can be done in each RNS channel of q. By assuming that $\omega = 2^\nu$, we would have $\ell_{\omega,q} = k$. The computation of the vector $\mathcal{D}_{\omega,q}(\lfloor\frac{t}{q}\boldsymbol{c}_2\rceil)$ reduces to shifting. The two scalar products in $\mathtt{Relin}_{\mathtt{Fv}}$, with an output in coefficient representation, require $k\ell_{\omega,q}\mathtt{NTT} + 2k^2 n\mathtt{IMM} + 2k\mathtt{invNTT}$. Thus, the total cost is at most the following one:

$$\begin{aligned}\mathtt{Cost}(\mathtt{Mult}_{\mathtt{MP}}) = (k\ell_{\omega,q} + 8k + 4)\mathtt{NTT} + (8k+3)\mathtt{invNTT} \\ +[2k^2 + 6k + 3]n\mathtt{IMM} + [40k^2 + \mathcal{O}(k^{1+\varepsilon})]n\mathtt{EM}.\end{aligned}$$

Let's analyse the cost of Alg. 3. The fast conversions at S0 from q to $\mathcal{B}_{sk} \cup \{\tilde{m}\}$ require $4 \times k(k+2)n\mathtt{IMM}$. Next, the reduction of q-overflows at step S1 requires $4 \times (k+1)n\mathtt{IMM}$. The product of ciphertexts $\mathtt{ct}'_1 \star \mathtt{ct}'_2$ (S2) in $q \cup \mathcal{B}_{sk}$ requires the same cost as for MP variant, that is $4(2k+1)\mathtt{NTT} + 3(2k+1)n\mathtt{IMM} + 3(2k+1)\mathtt{invNTT}$.

Let's analyse the cost of steps S3, S4 and S5. With adequate pre-computed data, the base conversion in S3 can integrate the flooring computation in $\mathcal{B}_{\mathtt{sk}}$. So, S3 is achievable with $3 \times k(k+1)n\mathtt{IMM}$. The exact FastBconvSK at step S4 basically reduces to a fast conversion from $\mathcal{B}$ to $q_{\mathtt{sk}}$, followed by a second one

from $m_{\mathtt{sk}}$ to q. So, this is achieved with $3 \times k(k+2)n\mathtt{IMM}$. In `S5`, we already have the vector $\xi_q(\boldsymbol{c}_2)$ which is involved in the fast conversion in step `S3`. Indeed, the function ξ_q is an automorphism of $\mathcal{R}_q$. So, data in q can stay in this form throughout the computations. The two scalar products in $\mathtt{Relin}_{\mathtt{RNS}}$ involve $k^2\mathtt{NTT}+2k^2n\mathtt{IMM}+2k\mathtt{invNTT}$, exactly like the relinearisation step in MP variant. And finally $2kn\mathtt{IMM}$ are needed to manage the Montgomery representation, in q, with respect to $\tilde{m}$.

$$\begin{aligned}\mathtt{Cost}(\mathtt{Mult}_{\mathtt{RNS}}) = (k^2+8k+4)\mathtt{NTT} + (8k+3)\mathtt{invNTT}\\ +[10k^2+25k+7]n\mathtt{IMM}.\end{aligned}$$

To summarize, the RNS variant decreases the computational cost of the whole homomorphic multiplication algorithm except the parts concerning polynomial multiplications. Also, it involves as many `NTT` and `invNTT` as the MP variant. Even by considering an optimized multi-precision multiplication algorithm in MP variant (with sub-quadratic complexity), the asymptotic computational complexity remains dominated by the $(k^2+\mathcal{O}(k))n\mathtt{NTT}$. Finally, the MP and RNS variants are asymptotically equivalent when $n \to +\infty$, but the pure RNS variant is more flexible in terms of parallelization.

C Timing results for decryption and multiplication

Tables 3 and 4 summarize the values chosen for $\tilde{m}$ for all the configurations which have been implemented and tested, and they list sufficient sizes for $m_{\mathtt{sk}}$ and γ in these cases.

n	k	t	$\tilde{m}$	$L_{\mathtt{RNS}}$	$\lceil\log_2(m_{\mathtt{sk}})\rceil$	γ
2^{11}	3	2	2^8	2	13	7
		2^{10}		1	22	7
2^{12}	6	2	2^8	5	14	13
		2^{10}		4	23	13
2^{13}	13	2	2^8	13	15	27
		2^{10}		9	24	27
2^{14}	26	2	2^8	25	17	53
		2^{10}		19	26	53
2^{15}	53	2	2^{16}	50	20	112
		2^{10}		38	29	107

Table 3: Parameters based on 30-bit moduli of `NFLlib` and depending on an *a priori* chosen $\tilde{m}$.

n	k	t	$\tilde{m}$	$L_{\mathtt{RNS}}$ ($L_{\mathtt{std}}$)	$\lceil\log_2(m_{\mathtt{sk}})\rceil$	γ
2^{11}	1	2	2^8	0 (0)	–	–
		2^{10}		0 (0)	–	–
2^{12}	3	2	2^8	4 (5)	14	7
		2^{10}		3 (3)	23	7
2^{13}	6	2	2^8	11 (11)	15	13
		2^{10}		8 (8)	24	13
2^{14}	12	2	2^8	23 (23)	16	25
		2^{10}		17 (17)	25	25
2^{15}	25	2	2^{16}	47 (47)	17	51
		2^{10}		37 (37)	26	52

Table 4: Parameters based on 62-bit moduli of `NFLlib` and depending on an *a priori* chosen $\tilde{m}$.

Table 5 lists timings and speed-ups of RNS vs MP variants. Also, timings of an RNS decryption and multiplication including the use of SIMD (Single Instruction Multiple Data) have been added.

In RNS variants (as well decryption as multiplication), the replacement of division and rounding by base conversions (i.e. matrix-vector multiplications) allows to benefit, easily and naturally, from concurrent computation. An RNS vector-matrix multiplication naturally owns two levels of parallelization: along the RNS channels, and along the dimension of the result. In `NFLlib`, an element of $\mathcal{R}_q$ is stored in a (32-byte aligned) array `_data` in which the n first values are the coefficients of the polynomial in $\mathcal{R}_{q_1}$, and so forth and so on. Advanced Vector Extensions (`AVX2`) have been used to accelerate the computations.

An `AVX2` register is handled (for our purpose) through the type `__m256i`. This enables us to handle either 8x32-bit, or 4x64-bit, or again 16x16-bit integers concurrently. Given the configuration of the `_data` array, reading/writing communications between 256-bit `AVX2` registers and `_data` are the most efficient (through `_mm256_store_si256` and `_mm256_load_si256` intrinsics, which require the 32-byte alignment of `_data`) when the base conversion is parallelized along the dimension n. This is the way it has been implemented and tested. Moreover, this has been only done within the 32-bit implementations, for which more convenient intrinsic instructions are available.

Regarding the timings, the impact of `AVX2` remains quite moderate. This is because it is used to accelerate the parts of algorithms besides the `NTT`-based polynomial products which constitute the main cost. For decryption, the RNS variants, floating point and integer, are already very efficient, whereas the performance of `AVX2` variant depends on time-consuming loading operations from/to vectorial registers, explaining the small differences. About the multiplication, as expected the timings are converging when n grows, because of the cost of `NTT`'s.

n	ν	k	variant	Decryption (ms)	Speed-up	Multiplication (ms)	Speed-up
2^{11}	30	3	MP	1.153	–	13.809	–
			RNS-flp	0.192	6.005	–	–
			RNS	0.189	6.101	3.159	4.371
			RNS-AVX2	0.188	6.133	2.710	5.096
	62	1	MP	1.020	–	–	–
			RNS-flp	0.054	18.880	–	–
			RNS	0.050	20.390	–	–
2^{12}	30	6	MP	4.587	–	45.055	–
			RNS-flp	0.798	5.748	–	–
			RNS	0.789	5.814	15.614	2.886
			RNS-AVX2	0.775	5.919	13.737	3.280
	62	3	MP	3.473	–	28.168	–
			RNS-flp	0.339	10.245	–	–
			RNS	0.326	10.653	7.688	3.664
2^{13}	30	13	MP	16.051	–	218.103	–
			RNS-flp	3.732	4.301	–	–
			RNS	3.691	4.349	100.625	2.167
			RNS-AVX2	3.637	4.413	88.589	2.462
	62	6	MP	10.945	–	92.093	–
			RNS-flp	1.552	7.052	–	–
			RNS	1.513	7.234	37.738	2.440
2^{14}	30	26	MP	70.154	–	1,249.400	–
			RNS-flp	17.497	4.009	–	–
			RNS	17.333	4.047	622.596	2.007
			RNS-AVX2	16.818	4.171	617.846	2.022
	62	12	MP	38.910	–	424.014	–
			RNS-flp	6.702	5.806	–	–
			RNS	6.494	5.992	206.511	2.053
2^{15}	30	53	MP	364.379	–	8,396.080	–
			RNS-flp	85.165	4.279	–	–
			RNS	81.225	4.486	4,923.220	1.705
			RNS-AVX2	72.665	5.015	5,063.920	1.658
	62	25	MP	180.848	–	2,680.535	–
			RNS-flp	33.310	5.429	–	–
			RNS	31.895	5.670	1,406.960	1.905

Table 5: Timing results.