An HPR variant of the FV scheme: Computationally Cheaper, Asymptotically Faster

Jean-Claude Bajard[1], Julien Eynard[2], Paulo Martins[3], Leonel Sousa[3], and Vincent Zucca[4]

[1] Sorbonne Université, CNRS, LIP6, Paris, France
jean-claude.bajard@lip6.fr
[2] Univ. Grenoble Alpes, CEA, LETI, DSYS, CESTI, F-38000 Grenoble
julien.eynard@cea.fr
[3] INESC-ID, Instituto Superior Técnico, Universidade de Lisboa
paulo.sergio@netcabo.pt, las@inesc-id.pt
[4] Institute of Cybersecurity and Cryptology, University of Wollongong
vzucca@uow.edu.au

Abstract. State-of-the-art implementations of homomorphic encryption exploit the Fan and Vercauteren (FV) scheme and the Residue Number System (RNS). While the RNS breaks down large integer arithmetic into smaller independent channels, its non-positional nature makes operations such as division and rounding hard to implement, and makes the representation of small values inefficient. In this work, we propose the application of the Hybrid Position-Residues Number System representation to the FV scheme. This is a positional representation of large radix where the digits are represented in RNS. It inherits the benefits from RNS and allows to accelerate the critical division and rounding operations while also making the representation of smaller values more compact. This directly benefits the decryption and the homomorphic multiplication procedures, reducing their asymptotic complexity, in dimension n, from $\mathcal{O}(n^2 \log n)$ to $\mathcal{O}(n \log n)$ and from $\mathcal{O}(n^3 \log n)$ to $\mathcal{O}(n^3)$, respectively. This has also resulted in noticeable speedups when experimentally compared to related art RNS implementations.

Keywords: Fan-Vercauteren · Residue Number System · Homomorphic Encryption

1 Introduction

Homomorphic Encryption (HE) enables the processing of encrypted data and has a large number of applications including secure electronic voting [1], verifiable computing [10] and multi-party computation [8]. One of the most efficient Somewhat Homomorphic Encryption (SHE) schemes, that allows for a predetermined number of operations, was due to Fan and Vercauteren [9]. Ciphertexts

[1] was initially supported by Sorbonne Université for this project.
[2] was initially supported by University of Toulon (France) for this project.

are represented as a vector of polynomials that grows as homomorphic multiplications are applied. In order to prevent the continuous growth of this vector, one has to apply *Relinearisation* [9]. This operation is currently the main bottleneck of the scheme, and has thus been the focus of study of previous works [12,5,11].

The Residue Number System (RNS) is a number representation that breaks down large integer arithmetic into many independent smaller channels [15]. With it, additions and multiplications are computed independently for each channel, resulting in a reduction of complexity when compared to traditional positional representations. In particular, since the Fan-Vercauteren (FV) scheme operates on polynomials with large coefficients, the RNS is useful to accelerate homomorphic processing [12,5,11]. However, while for positional notations the size of the representation of a number is proportional to its magnitude, the RNS representation of a number is as large as the largest number representable in it. During the relinearisation procedure, an operation requires polynomial coefficients to be decomposed to values of a smaller magnitude. Hence, after applying this decomposition, the RNS becomes a suboptimal representation. Furthermore, the non-positional nature of RNS makes it more difficult to implement operations such as division and rounding, which are required by both FV homomorphic multiplication and decryption procedures.

This work proposes to exploit instead the Hybrid Position-Residues Number System (HPR) [6] representation of numbers to accelerate the cryptographic operations of FV. This hybrid representation is of a positional nature, but each digit is represented in RNS. It not only inherits the parallel nature of RNS but it also makes the relinearisation, and division and rounding operations more efficient. As a result, in dimension n, the complexity of FV decryption (resp. homomorphic multiplication) is asymptotically reduced from $\mathcal{O}(n^2 \log n)$ to $\mathcal{O}(n \log n)$ (resp. $\mathcal{O}(n^3 \log n)$ to $\mathcal{O}(n^3)$). Complexity gains are translated in practice into speedups of up to 8.0 and 8.6 for the decryption operation and of up to 2.0 and 1.6 for the homomorphic multiplication, for the parameters considered in this article and two state-of-the-art schemes, with an upward trend for larger parameters.

2 Background

The ring $\mathcal{R} = \mathbb{Z}[X]/(X^n + 1)$ can be seen as the set of polynomials with integer coefficients and degree smaller than n. $\mathcal{R}_q$ is used to denote the polynomials in $\mathcal{R}$ whose coefficients are reduced modulo an integer q. When $q = 1 \pmod{2n}$, elements of $\mathcal{R}_q$ can be represented in the Number Theoretic Transform (NTT)-domain, which allows for coefficient-wise additions and multiplications [2]. Henceforth, boldface letters are used to denote polynomials and the infinity norm $\|\cdot\|_\infty$ returns the largest absolute value of the polynomial's coefficients. The expansion factor associated to $\mathcal{R}$ is defined as $\delta_\mathcal{R} = \sup\left\{\frac{\|\boldsymbol{a}\cdot\boldsymbol{b}\|_\infty}{\|\boldsymbol{a}\|_\infty\|\boldsymbol{b}\|_\infty}, \right.$ for $\left.\boldsymbol{a}, \boldsymbol{b} \in \mathcal{R} - \{0\}\right\}$. $\boldsymbol{a} \leftarrow D$ and $\boldsymbol{a} \leftarrow \chi$ are used to denote that $\boldsymbol{a}$ is drawn uniformly at random from the set D or according to a distribution χ, respectively. The notation $|\cdot|_q$ (resp. $[\cdot]_q$) is used to denote the residue modulo q in $[0, q)$ (resp. in $[-q/2, q/2)$). Moreover, for $a \in \mathbb{Q}$, $b = \lfloor a \rceil$ is used to denote the

closest integer $b \in \mathbb{Z}$ to a. The computational complexity is herein evaluated in terms of the amount of Elementary Modular Multiplications (EMMs), floating point operations (FPOs) and forward and inverse NTTs required.

2.1 FV cryptosystem

A private-key of the FV scheme [9] is generated as a polynomial $\boldsymbol{s} \leftarrow \chi_{key}$, and the corresponding public-key is generated as $\boldsymbol{b} = (\left[-(\boldsymbol{a} \cdot \boldsymbol{s} + \boldsymbol{e})\right]_q, \boldsymbol{a}) \in \mathcal{R}_q^2$; with $\boldsymbol{a} \leftarrow \mathcal{R}_q$ drawn uniformly at random and $\boldsymbol{e} \leftarrow \chi_{err}$. χ_{key} is such that for $\boldsymbol{s} \leftarrow \chi_{key}$, $\|\boldsymbol{s}\|_\infty \leqslant B_{key}$ for a small B_{key}, while the distribution χ_{err} has standard deviation σ_{err} such that elements sampled from this distribution have their infinite norm smaller than $B_{err} = 6\sigma_{err}$ with very high probability.

A relinearisation key is defined with respect to a pair of functions $\mathfrak{D}$ and $\mathfrak{P}$, such that $[\langle \mathfrak{D}(\boldsymbol{a}), \mathfrak{P}(\boldsymbol{b}) \rangle]_q = [\boldsymbol{a} \cdot \boldsymbol{b}]_q$ and the infinite norm of $\mathfrak{D}(\boldsymbol{a})$ is "small". The relinearisation key, which can be interpreted as a vector of pseudo-encryptions of $\mathfrak{P}(\boldsymbol{s}^2)$, is defined as: $\overrightarrow{\texttt{rlk}} = (\left[\mathfrak{P}(\boldsymbol{s}^2) - (\overrightarrow{\boldsymbol{e}} + \overrightarrow{\boldsymbol{a}} \cdot \boldsymbol{s})\right]_q, \overrightarrow{\boldsymbol{a}}) \in \mathcal{R}_q^{2l}$, where $\overrightarrow{\boldsymbol{e}} \leftarrow \chi_{err}^l$ and $\overrightarrow{\boldsymbol{a}} \leftarrow \mathcal{R}_q^l$. A ciphertext $\texttt{ct} = (\boldsymbol{c}_0, \boldsymbol{c}_1) \in \mathcal{R}_q^2$, encrypting $[\boldsymbol{m}]_t$, satisfies $\boldsymbol{c}_0 + \boldsymbol{c}_1 \cdot \boldsymbol{s} = \Delta[\boldsymbol{m}]_t + \boldsymbol{v} \bmod q$, where $\Delta = \lfloor q/t \rfloor$. ct can be correctly decrypted as long as the noise $\boldsymbol{v}$ remains small enough – $\|\boldsymbol{v}\|_\infty < \Delta/2 - |q|_t/2$, through:

$$\boldsymbol{m} = \left[\left\lfloor \frac{t}{q}[\boldsymbol{c}_0 + \boldsymbol{c}_1 \cdot \boldsymbol{s}]_q \right\rfloor\right]_t. \tag{1}$$

The homomorphic addition of two ciphertexts corresponds to the pairwise addition of their coefficients modulo q. The homomorphic multiplication of ct and $\texttt{ct}'$ is performed in two steps. First, we compute the three elements ciphertext:

$$\tilde{\texttt{ct}}_{\text{mult}} = \left(\left[\left\lfloor \frac{t}{q}\boldsymbol{c}_0 \cdot \boldsymbol{c}_0' \right\rfloor\right]_q, \left[\left\lfloor \frac{t}{q}(\boldsymbol{c}_0 \cdot \boldsymbol{c}_1' + \boldsymbol{c}_1 \cdot \boldsymbol{c}_1) \right\rfloor\right]_q, \left[\left\lfloor \frac{t}{q}\boldsymbol{c}_1 \cdot \boldsymbol{c}_1' \right\rfloor\right]_q \right) \in \mathcal{R}_q^3, \tag{2}$$

satisfying $\tilde{\boldsymbol{c}}_0 + \tilde{\boldsymbol{c}}_1\boldsymbol{s} + \tilde{\boldsymbol{c}}_1\boldsymbol{s}^2 = \Delta[\boldsymbol{m} \cdot \boldsymbol{m}']_t + \boldsymbol{v}' \bmod q$. In a second step, $\tilde{\texttt{ct}}_{\text{mult}}$ is converted back to a two element ciphertext, by multiplying $\mathfrak{D}(\tilde{\boldsymbol{c}}_2)$ by $\overrightarrow{\texttt{rlk}}$ and adding the result to $(\tilde{\boldsymbol{c}}_0, \tilde{\boldsymbol{c}}_1)$:

$$\texttt{ct}_{\text{mult}} = \left(\left[\tilde{\boldsymbol{c}}_0 + \left\langle \mathfrak{D}(\tilde{\boldsymbol{c}}_2), \overrightarrow{\texttt{rlk}_0} \right\rangle\right]_q, \left[\tilde{\boldsymbol{c}}_0 + \left\langle \mathfrak{D}(\tilde{\boldsymbol{c}}_2), \overrightarrow{\texttt{rlk}_1} \right\rangle\right]_q \right) \tag{3}$$

The result $\texttt{ct}_{\text{mult}}$, can be decrypted with (1) to produce $[\boldsymbol{m} \cdot \boldsymbol{m}']_t$. As homomorphic operations are applied, the noise associated with ciphertexts grows, limiting the number of operations that can be applied.

2.2 Hybrid Position-Residue Number System

Using [6] and considering a modulus q such that $q = p^d$, one can represent polynomials $\boldsymbol{a} \in \mathcal{R}_q$ in a positional system of large radix base p – i.e. such that $\boldsymbol{a} = \sum_{i=0}^{d-1} \boldsymbol{a}_i p^i$ with $(\boldsymbol{a}_0, \ldots, \boldsymbol{a}_{d-1}) \in (\mathcal{R}_p)^d$.

Moreover, if the modulus p is chosen as a product of small distinct primes $p = p_1 \cdots p_k$, the Chinese Remainder Theorem (CRT) states that there is a ring isomorphism between $\mathcal{R}_p$ and $\mathcal{R}_{p_1} \times \ldots \times \mathcal{R}_{p_k}$. In this case, the digits $\boldsymbol{a}_i$s can be represented in RNS as $f_p(\boldsymbol{a}_i) = ([\boldsymbol{a}_i]_{p_1}, \ldots, [\boldsymbol{a}_i]_{p_k})$ which allows to break down the arithmetic over the large coefficients into several smaller channels and thus enhance performance. The RNS representation can be reversed, and $\boldsymbol{a}_i$ recovered, by computing:

$$f_p^{-1}([\boldsymbol{a}_i]_{p_1}, \ldots, [\boldsymbol{a}_i]_{p_k}) = \left[\sum_{i=1}^{k} \left[[\boldsymbol{a}_i]_{p_i} \frac{p_i}{p}\right]_{p_i} \frac{p}{p_i}\right]_p = \sum_{i=1}^{k} \left[\boldsymbol{a}_i \frac{p_i}{p}\right]_{p_i} \frac{p}{p_i} - \boldsymbol{\alpha}_i p \quad (4)$$

However, since the HPR is of a positional nature ($\boldsymbol{a} = \sum_{i=0}^{d-1} \boldsymbol{a}_i p^i$), one has to consider the propagation of carries over the digits after additions or multiplications. In order to compute the carries, the digits should be able to grow larger than p. Thus, the digits arithmetic cannot be performed directly modulo p and we must adjoined a second RNS basis $\mathcal{B} = \{b_1, \ldots, b_k\}$ to $\mathcal{P} = \{p_1, \ldots, p_k\}$, so that the digits can grow larger than p to compute the carries.

In order to approximate the carry caused by the i^{th} digit, we add a modulus b_{sk} to $\mathcal{B}$ to form $\mathcal{B}_{\text{sk}}$, and use a Shenoy-Kumaresan approach [14]. First, we use a fast base extension (FBE),

$$\texttt{FBE}(\boldsymbol{a}_i, \mathcal{P}, \mathcal{B}_{\text{sk}}) = \left(\sum_{i=1}^{k} \left[\boldsymbol{a}_i \frac{p_i}{p}\right]_{p_i} \cdot \frac{p}{p_i} \bmod b\right)_{b \in \mathcal{B}_{\text{sk}}},$$

to obtain $[\boldsymbol{a}_i]_p + \boldsymbol{\alpha}_i \cdot p$ in base $\mathcal{B}_{\text{sk}}$ with $\|\boldsymbol{\alpha}_i\|_\infty \leqslant (k-1)/2$. Then, the approximated carry $\tilde{\boldsymbol{c}}_i = \frac{\boldsymbol{a}_i - \texttt{FBE}(\boldsymbol{a}_i, \mathcal{P}, \mathcal{B}_{\text{sk}})}{p} = \lfloor \frac{\boldsymbol{a}_i}{p} \rceil - \boldsymbol{\alpha}_i$ is produced in $\mathcal{B}_{\text{sk}}$. To finalise the procedure, the value of $\tilde{\boldsymbol{c}}_i$ is extended exactly to $\mathcal{P}$. An inexact extension first computes $\texttt{FBE}(\tilde{\boldsymbol{c}}_i, \mathcal{B}, \mathcal{P}) = \tilde{\boldsymbol{c}}_i + \boldsymbol{\beta}_i \cdot b$ in basis $\mathcal{P}$ with $b = b_1 \cdots b_k$ and $\|\boldsymbol{\beta}_i\|_\infty \leqslant (k-1)/2$. Then, the extra residue modulo b_{sk} is used to correct $\boldsymbol{\beta}_i \cdot b$, by computing $\boldsymbol{\beta}_i$ with:

$$[\boldsymbol{\beta}_i]_{b_{\text{sk}}} = \left[\frac{1}{b}\left(\sum_{i=1}^{\ell} \left[\boldsymbol{a}_i \frac{b_i}{b}\right]_{b_i} \frac{b}{b_i} - [\tilde{\boldsymbol{c}}_i]_{b_{\text{sk}}}\right)\right]_{b_{\text{sk}}}. \quad (5)$$

With this procedure, values $\|\tilde{\boldsymbol{c}}_i\| < \lambda B$ can be exactly represented and extended from $\mathcal{B}$ so long as $b_{sk} \geq 2(k+\lambda)$ [5, Lemma 6]. For the parameters considered in this article, b_{sk} can have the same bitwidth as the other moduli in $\mathcal{P}$ and $\mathcal{B}$, which is enough to get the residues of $[\boldsymbol{a}_i]_p + \boldsymbol{\alpha}_i \cdot p$ and of $\tilde{\boldsymbol{c}}_i = \boldsymbol{c}_i - \boldsymbol{\alpha}_i$ in $\mathcal{P} \cup \mathcal{B}_{\text{sk}}$.

Afterwards, $\tilde{\boldsymbol{c}}_i$ can be added to the digit of index $i+1$ and one can re-run the procedure to compute the carry $\tilde{\boldsymbol{c}}_{i+1}$ caused by the digit of index $i+1$. If one assumes an element to belong to $\mathcal{R}_q$, the representation of the $(d-1)$th digit in basis $\mathcal{B}$ can be discarded. The $(d-1)$th digit is implicitly multiplied by p^{d-1}, so any value larger than $p/2$ in absolute value would lead to representations larger than $q/2 = p \times p^{d-1}/2$, which are redundant.

Practical complexity The computation of a carry for a single HPR digit requires a first fast base extension from $\mathcal{P}$ to the $k+1$ moduli of $\mathcal{B}_{\text{sk}}$ ($nk(k+2)$) `EMMs`), followed by the product by $1/p$ in base $\mathcal{B}_{\text{sk}}$ ($n(k+1)$ `EMMs`), and a Shenoy-Kumaresan extension ($n(k^2+3k+1)$ `EMMs`). Finally, since, when operating modulo q, carries are consecutively computed for all the digits but the last, the carry propagation procedure applied to an element in $\mathcal{R}_p^d$ requires the following amount of EMMs:

$$\begin{cases} \mathcal{C}_{1\_\text{carry}} = n(2k^2+6k+2) \text{ EMMs} \\ \mathcal{C}_{\text{all\_carries}} = (d-1)\cdot \mathcal{C}_{1\_\text{carry}}. \end{cases} \tag{6}$$

3 Proposed HPR-based FHE System

In this section, we propose an efficient HPR variant of the FV scheme. Key generation, encryption and homomorphic addition take place as in the original scheme [9], but with polynomial coefficients represented in HPR.

3.1 Proposed HPR-based Decryption Algorithm

The computational costs of RNS-based approaches to FV decryption are dominated by the amount of NTTs one needs to compute [11,5]. Since polynomial coefficients are therein represented in RNS, NTTs are computed for moduli $\{q_1, \ldots, q_{K_{RNS}}\}$ such that $q_1 \cdots q_{K_{RNS}} = q$. With an HPR approach, NTTs can be computed only for the most-significant digit of the ciphertext polynomials. Since each digit is represented in an RNS basis d times smaller than K_{RNS}, the amount of NTTs one needs to compute is also reduced d times.

We assume that the HPR representation of a ciphertext $(\boldsymbol{c}_0, \boldsymbol{c}_1) \in \mathcal{R}_q^2$ has digits satisfying $\|\boldsymbol{c}_{i,j}\| \leqslant \beta \cdot p$ for $i = 0, 1$ and $0 \leqslant j \leqslant d-2$: β may take the value of $k/2$ if carry propagation has been applied; or might be even slightly larger for cases where the ciphertext represents an homomorphic sum. Lemma 1 proves that, for the considered setting, the value of $\boldsymbol{c}_0(p) + \boldsymbol{c}_1(p) \cdot \boldsymbol{s}$ can be efficiently approximated having only access to the mod-$\mathcal{P}$ representation of the $(d-1)$th digits of $\boldsymbol{c}_0$ and $\boldsymbol{c}_1$ without causing a large error.

Lemma 1. *For $i = 0, 1$, let $\boldsymbol{c}_i$ be such that $\|\boldsymbol{c}_{i,j}\|_\infty \leqslant \beta p$ for $j \in [\![0, d-2]\!]$, then:*

$$[\boldsymbol{c}_{0,d-1} + \boldsymbol{c}_{1,d-1} \cdot \boldsymbol{s}]_p = \frac{\Delta[\boldsymbol{m}]_t + \boldsymbol{v} + q\boldsymbol{u}}{p^{d-1}} + \boldsymbol{e} + \boldsymbol{\epsilon} \cdot p \tag{7}$$

for some $\boldsymbol{\epsilon} \in \mathcal{R}$, with the error $\boldsymbol{e}$ verifying $\|\boldsymbol{e}\|_\infty \leqslant \frac{p}{p-1}\beta(1 + \delta_{\mathcal{R}} B_{key})$.

Proof. In Appendix A.

Decryption consists in efficiently computing (8) modulo t based on the values of $\boldsymbol{c}_{0,d-1}$ and $\boldsymbol{c}_{1,d-1}$.

$$\left\lfloor t \cdot \frac{[\boldsymbol{c}_{0,d-1} + \boldsymbol{c}_{1,d-1} \cdot \boldsymbol{s}]_p}{p} \right\rceil = [\boldsymbol{m}]_t + \lfloor \tilde{\boldsymbol{e}} \rceil + t(\boldsymbol{u} + \boldsymbol{\epsilon}) \tag{8}$$

with $\tilde{\boldsymbol{e}} = \left(t \cdot \boldsymbol{v} - |q|_t[\boldsymbol{m}]_t + tp^{d-1}\boldsymbol{e}\right)/q$ which has a norm bounded up by:

$$\|\tilde{\boldsymbol{e}}\|_\infty \leqslant \frac{t}{q}\|\boldsymbol{v}\|_\infty + \frac{t|q|_t}{2q} + \frac{t}{p-1}\beta(1+\delta_{\mathcal{R}}B_{\text{key}}) \tag{9}$$

We use a strategy similar to [5] to compute (8) with a fast basis extension but without any error. By scaling the computation in (8) by an integer γ, any extension errors will be detected since they will be nonzero modulo γ. After efficiently getting $[\gamma t(\boldsymbol{c}_{0,d-1} + \boldsymbol{c}_{1,d-1} \cdot \boldsymbol{s})]_p + \boldsymbol{\alpha}p$ through a fast conversion from $\mathcal{P}$ to $\{\gamma t\}$, (10) can be computed.

$$\begin{aligned}
\boldsymbol{r} &= \frac{\gamma t[\boldsymbol{c}_{0,d-1} + \boldsymbol{c}_{1,d-1} \cdot \boldsymbol{s}]_p - ([\gamma t(\boldsymbol{c}_{0,d-1} + \boldsymbol{c}_{1,d-1} \cdot \boldsymbol{s})]_p + \boldsymbol{\alpha}p)}{p} \bmod \gamma t \\
&= \frac{\gamma\left(t[\boldsymbol{c}_{0,d-1} + \boldsymbol{c}_{1,d-1} \cdot \boldsymbol{s}]_p - [t(\boldsymbol{c}_{0,d-1} + \boldsymbol{c}_{1,d-1} \cdot \boldsymbol{s})]_p\right)}{p} + \\
&\quad \frac{\gamma[t(\boldsymbol{c}_{0,d-1} + \boldsymbol{c}_{1,d-1} \cdot \boldsymbol{s})]_p - [\gamma t(\boldsymbol{c}_{0,d-1} + \boldsymbol{c}_{1,d-1} \cdot \boldsymbol{s})]_p}{p} - \boldsymbol{\alpha} \bmod \gamma t \\
&= \gamma\left\lfloor t\frac{[\boldsymbol{c}_{0,d-1} + \boldsymbol{c}_{1,d-1} \cdot \boldsymbol{s}]_p}{p}\right\rceil + \left\lfloor \gamma\frac{[t(\boldsymbol{c}_{0,d-1} + \boldsymbol{c}_{1,d-1} \cdot \boldsymbol{s})]_p}{p}\right\rceil - \boldsymbol{\alpha} \bmod \gamma t \\
&= \gamma\left([\boldsymbol{m}]_t + \lfloor\tilde{\boldsymbol{e}}\rceil\right) + \left\lfloor \gamma\frac{[t(\boldsymbol{c}_{0,d-1} + \boldsymbol{c}_{1,d-1} \cdot \boldsymbol{s})]_p}{p}\right\rceil - \boldsymbol{\alpha} \bmod \gamma t \\
&= \gamma[\boldsymbol{m}]_t + \gamma\lfloor\tilde{\boldsymbol{e}}\rceil + \left\lfloor \gamma\frac{[p\tilde{\boldsymbol{e}}]_p}{p}\right\rceil - \boldsymbol{\alpha} \bmod \gamma t
\end{aligned} \tag{10}$$

Now, through the application of Lemma 2 to (10), $[\boldsymbol{m}]_t$ can be computed as $(\boldsymbol{r} - [\boldsymbol{r}]_\gamma)/\gamma \bmod t$.

Lemma 2. *Let $\gamma > k/2$ and $\|\boldsymbol{\alpha}\|_\infty \leqslant (k-1)/2$. If we have*

$$\|\boldsymbol{v}\|_\infty < \frac{q}{2t}\left(1 - \frac{k}{\gamma} - \frac{2t\beta}{p-1}(1+\delta_{\mathcal{R}}B_{key})\right) - \frac{|q|_t}{2} \tag{11}$$

then $\lfloor\tilde{\boldsymbol{e}}\rceil = 0$, and $\left\lfloor\left\lfloor\gamma\frac{[p\tilde{\boldsymbol{e}}]_p}{p}\right\rceil - \boldsymbol{\alpha}\right\rceil_\gamma = \left\lfloor\gamma\frac{[p\tilde{\boldsymbol{e}}]_p}{p}\right\rceil - \boldsymbol{\alpha}$. Hence, the correction technique from [5] works.

Proof. In Appendix B.

3.2 Summarising

The above-described decryption steps are featured in Algorithm 1. First, $\boldsymbol{c}_{0,d-1} + \boldsymbol{c}_{1,d-1} \cdot \boldsymbol{s}$ is computed as an approximation to $(\boldsymbol{c}_0(p) + \boldsymbol{c}_1(p) \cdot \boldsymbol{s})/p^{d-1}$. Then (10) is computed modulo γt. Notice that since $\gamma t \cdot [\boldsymbol{c}_{0,d-1} + \boldsymbol{c}_{1,d-1} \cdot \boldsymbol{s}]_p = 0 (\mathrm{mod} \gamma t)$, the expression for $\boldsymbol{r}$ can be simplified as

$$\boldsymbol{r} = \frac{-([\gamma t(\boldsymbol{c}_{0,d-1} + \boldsymbol{c}_{1,d-1} \cdot \boldsymbol{s})]_p + \boldsymbol{\alpha}p)}{p} \bmod \gamma t$$

In order to get the value of $[\gamma t(\boldsymbol{c}_{0,d-1} + \boldsymbol{c}_{1,d-1} \cdot \boldsymbol{s})]_p + \boldsymbol{\alpha} p$, $\boldsymbol{c}_{0,d-1} + \boldsymbol{c}_{1,d-1} \cdot \boldsymbol{s}$ is scaled by γt modulo p and extended to basis γt. The function $\xi_{\mathcal{P}}$ computes $\xi_{\mathcal{P}}(\boldsymbol{a}) = ([\boldsymbol{a}p_1/p]_{p_1}, \ldots, [\boldsymbol{a}p_k/p]_{p_k})$ for a polynomial $\boldsymbol{a} \in \mathcal{R}_p$ and is used to prepare the extension to γt. In particular,

$$\texttt{FBE}(\boldsymbol{a}, \mathcal{P}, \{\gamma t\}) = \langle \xi_{\mathcal{P}}(\boldsymbol{a}), ([p/p_1]_{\gamma t}, \ldots, [p/p_k]_{\gamma t}) \rangle \bmod \gamma t$$

After $-\texttt{FBE}([\gamma t(\boldsymbol{c}_{0,d-1} + \boldsymbol{c}_{1,d-1} \cdot \boldsymbol{s})], \mathcal{P}, \gamma t)/p \bmod \gamma t$ is computed, the extension result is corrected using the centred remainder modulo γ. Lemma 2 ensures that $\boldsymbol{r}$ will hold $\gamma[\boldsymbol{m}]_t$ plus some noise. This noise can be recovered through the centred residue of $\boldsymbol{r}$ modulo γ, and then used to extract the message.

Algorithm 1 HPR-based Decryption

Require: $\boldsymbol{c}_{0,d-1}$, $\boldsymbol{c}_{1,d-1}$ in base $\mathcal{P} \times \mathcal{B}$, $\texttt{NTT}(\boldsymbol{s})$ in base $\mathcal{P}$
Ensure: $\boldsymbol{m} \bmod t$.
1: $([\boldsymbol{h}]_{p_1}, \ldots, [\boldsymbol{h}]_{p_k}) \leftarrow \xi_{\mathcal{P}}(\gamma t \cdot (\boldsymbol{c}_{0,d-1} + \texttt{NTT}^{-1}(\texttt{NTT}(\boldsymbol{c}_{1,d-1}) \odot \texttt{NTT}(\boldsymbol{s})))) \bmod p$
2: $\boldsymbol{r} \bmod \gamma t \leftarrow \langle ([\boldsymbol{h}]_{p_1}, \ldots, [\boldsymbol{h}]_{p_k}), (|-1/p_1|_{\gamma t}, \ldots, |-1/p_k|_{\gamma t}) \rangle \bmod \gamma t$
3: **return** $(\boldsymbol{r} - [\boldsymbol{r}]_\gamma)/\gamma \bmod t$

Notice that as in [5], if γ is chosen as a power of two, division by γ can be replaced with a shift. Hence, Algorithm 1 has a computational cost of:

$$k \ \texttt{NTT} + 2kn \ \texttt{EMM} + k \ \texttt{NTT}^{-1} + kn \ \texttt{EMM}_{\gamma t}.$$

4 Proposed HPR-based Homomorphic Multiplication

A main issue with RNS approaches to the homomorphic multiplication of FV has to do with the multiplication in $\mathbb{Z}$ it requires, that is followed by a division and rounding. Since RNS operations are inherently modular, one needs to extend the polynomial coefficients to large enough bases to handle the multiplications without any modular reduction. In our proposal, the extension of the most significant digit suffices to lift elements from $\mathcal{R}_q$ to $\mathcal{R}$. In addition, with RNS approaches, other expensive basis extensions are required to achieve the division and rounding. Given the positional nature of HPR, herein division and rounding is achieved by only computing the most significant digits of a product. Finally, the RNS representation of any value is as large as the representation of the largest value representable in the RNS basis. Since an NTT has to be computed for each modulus in the RNS basis, a large amount of NTTs is required even for small values. With HPR, small values can be represented as a single digit, reducing the amount of NTTs one has to compute d-fold.

For homomorphic multiplications in HPR we consider two ciphertexts $\texttt{ct} = (\boldsymbol{c}_0(Y), \boldsymbol{c}_1(Y))$ and $\texttt{ct}' = (\boldsymbol{c}'_0(Y), \boldsymbol{c}'_1(Y))$, with their elements represented in HPR, encrypting respectively $\boldsymbol{m}$ and $\boldsymbol{m}'$, such that $\boldsymbol{c}_0 + \boldsymbol{c}_1 \cdot \boldsymbol{s} = \Delta[\boldsymbol{m}]_t + \boldsymbol{v} + \boldsymbol{r}q$ and $\boldsymbol{c}'_0 + \boldsymbol{c}'_1 \cdot \boldsymbol{s} = \Delta[\boldsymbol{m}']_t + \boldsymbol{v}' + \boldsymbol{r}'q$ where $\boldsymbol{r}$ and $\boldsymbol{r}'$ satisfy the following bound:

$$\|\boldsymbol{r}\|_\infty, \|\boldsymbol{r}'\|_\infty \leqslant \underbrace{\frac{\delta_{\mathcal{R}} B_{key} + 1}{2} + 1}_{r_\infty}. \tag{12}$$

The polynomial digits are assumed to be bounded in a similar way to how they were bounded during decryption in Section 3.1, i.e. $\forall (i,j) \in \{0,1\} \times \{0,\dots,d-1\}$, $\|\boldsymbol{c}_{i,j}\|_\infty, \|\boldsymbol{c}'_{i,j}\|_\infty \leqslant \beta \cdot p$ with $\beta \simeq k/2$.

Until now, we only needed to represent the digit of index $d-1$ in basis $\mathcal{P}$, since we were dealing with elements of $\mathcal{R}$ modulo q. However, during homomorphic multiplication, the first product in (2) is not reduced modulo q. Therefore we have to extend the digits of index $d-1$ to the basis $\mathcal{P} \times \mathcal{B}_{\text{sk}}$. This extension is done exactly with the approach proposed in [11]. First, $\texttt{FBE}(\tilde{\boldsymbol{c}}_i, \mathcal{P}, \mathcal{B}_{\text{sk}}) = [\boldsymbol{c}_{i,d-1}]_p + \boldsymbol{\alpha}_i \cdot p$ is computed. Then, $\boldsymbol{\alpha}_i$ is produced as

$$\boldsymbol{\alpha}_i = \left\lfloor \sum_{j=1}^{k} \frac{\left[\tilde{\boldsymbol{c}}_i \frac{p_j}{p}\right]_{p_j}}{p_j} \right\rceil, \tag{13}$$

using floating point operations to compute the quotients and rounding to the nearest integer. Finally, the term $\boldsymbol{\alpha}_i \cdot p$ is subtracted from the intermediate result. The lifting procedure has a practical complexity of $n(k^2+3k+1)$ `EMMs` and $n(k+1)$ Floating-Point Operations (`FPOs`).

Notice that the approximation of (13) with floating point arithmetic may lead to an erroneous computation of $\boldsymbol{\alpha}_i$. There is a region $\mathbb{Z} + \frac{1}{2} \pm \epsilon$, where ϵ is related to the used precision and the number of moduli in the basis $\mathcal{P}$ ($|\epsilon| < 2^{-51}$ for IEEE 754 double precision and $k \leq 4$), to which the value computed in (13) may belong before rounding, causing a possible error of at most 1. While one can detect when this happens, and redo the computation with a higher precision to reduce ϵ, we have chosen, as in [11], to skip this error detection. We will discuss the possible effects of this choice in Remark 2.

At this point, we aim at computing (2). To do so we start by computing the `NTT`s of all the digits of the two input ciphertexts ($4d(2k+1)$ `NTT`s in total). Then the product is computed through a Karatsuba-like pattern:

$$\begin{cases} \widetilde{\boldsymbol{c}}_0(Y) = t \cdot \boldsymbol{c}_0(Y) \cdot \boldsymbol{c}'_0(Y) \\ \widetilde{\boldsymbol{c}}_2(Y) = t \cdot \boldsymbol{c}_1(Y) \cdot \boldsymbol{c}'_1(Y) \\ \widetilde{\boldsymbol{c}}_1(Y) = t \cdot (\boldsymbol{c}_0(Y) + \boldsymbol{c}'_1(Y)) \cdot (\boldsymbol{c}_1(Y) + \boldsymbol{c}'_0(Y)) - \widetilde{\boldsymbol{c}}_0(Y) - \widetilde{\boldsymbol{c}}_2(Y) \end{cases}$$

The products of polynomials in Y are performed as usual, i.e. in the power basis, and the products of digits, which are in `NTT` form, are made component-wise. Since the product is not reduced modulo q, no reduction modulo Y^d is done. Thus we obtain polynomials of degree $(2d-2)$ in Y. The division by $q = p^d$ and the rounding can be approximated by computing only the $d-1$ most significant digits of the products. Nonetheless, in order to reduce the noise added by this approximation, we also keep the carry coming from the digits of index $d-1$.

Remark 1. The products of polynomials in Y required for the computations of the $\widetilde{\boldsymbol{c}}_i$s, for $i = 0, 1, 2$, can be written as a vector-matrix product with the matrix in Toeplitz form. Therefore, they can be performed with subquadratic asymptotic complexity: $\mathcal{O}(d^{\log_2(3)})$ products of digits, i.e. with $O(d^{\log_2(3)}) \times n(2k+1)$ products of residues. However, the small values of d considered for practical parameters can make a naive (quadratic) algorithm more efficient.

Lemma 3. *Let $i \in \{0, 1, 2\}$ and $\lfloor \widetilde{\boldsymbol{c}}_{i,d-1}/p \rceil - \boldsymbol{\alpha}_{i,d-1}$ be the carry coming from $\widetilde{\boldsymbol{c}}_{i,d-1}$ with $\|\boldsymbol{\alpha}_{i,d-1}\|_\infty \leqslant (k-1)/2$ (see Section 2.2). We have:*

$$\left\lfloor \frac{\widetilde{\boldsymbol{c}}_{i,d-1}}{p} \right\rceil - \boldsymbol{\alpha}_{i,d-1} + \widetilde{\boldsymbol{c}}_{i,d} + \cdots + \widetilde{\boldsymbol{c}}_{i,2d-2} \cdot p^{d-2} = \frac{\widetilde{\boldsymbol{c}}_i}{q} + \boldsymbol{e}_i, \tag{14}$$

with:

$$\|\boldsymbol{e}_i\|_\infty \leqslant \underbrace{6\delta_{\mathcal{R}} t \beta^2 p \left(\frac{d-1}{p-1} - \frac{1-p^{-d}}{(p-1)^2} \right) + \frac{k}{2}}_{B_e}. \tag{15}$$

Proof. In Appendix C.

At this point, three polynomials of degree $d-1$ in Y have been produced, in `NTT` form, which approximate $\mathtt{ct}_{\text{mult}}$ with an error $\boldsymbol{e}_i$. They may be interpreted as encrypting $[\boldsymbol{m} \cdot \boldsymbol{m}']_t$. The next proposition states the inherent noise of this encryption, depending on the input noises $\boldsymbol{v}$ and $\boldsymbol{v}'$.

Proposition 1. *For $i \in \{0, 1, 2\}$, let $\hat{\boldsymbol{c}}_i = \left\lfloor \frac{\widetilde{\boldsymbol{c}}_{i,d-1}}{p} \right\rceil - \boldsymbol{\alpha}_{i,d-1} + \sum_{j=d}^{2d-2} \widetilde{\boldsymbol{c}}_{i,d} \cdot p^{j-d}$ as in (14), then we have:*

$$\hat{\boldsymbol{c}}_0 + \hat{\boldsymbol{c}}_1 \cdot \boldsymbol{s} + \hat{\boldsymbol{c}}_2 \cdot \boldsymbol{s}^2 \equiv \Delta \cdot [\boldsymbol{m} \cdot \boldsymbol{m}']_t + \hat{\boldsymbol{v}} \bmod q,$$

with:

$$\begin{aligned} \|\hat{\boldsymbol{v}}\|_\infty \leqslant\ & \delta_{\mathcal{R}} t \left(r_\infty + \frac{1}{2} \right) (\|\boldsymbol{v}\|_\infty + \|\boldsymbol{v}'\|_\infty) + \frac{\delta_{\mathcal{R}}}{2} \min(\|\boldsymbol{v}\|_\infty, \|\boldsymbol{v}'\|_\infty) \\ & + \frac{1}{2} + t \left(\frac{1}{2} + t\delta_{\mathcal{R}}(1 + r_\infty) \right) + B_e \left(1 + \delta_{\mathcal{R}} B_{key} + \delta_R^2 B_{key}^2 \right) \end{aligned} \tag{16}$$

with B_e as in (15) and r_∞ as in (12).

Proof. In Appendix D.

Remark 2. In the case where an error has occured in the `HPS` basis extension, before the multiplication, then we would obtain $\tilde{\boldsymbol{c}}_i = \boldsymbol{c}_i + q\boldsymbol{u}$ with $\|\boldsymbol{u}\|_\infty = 1$ (or $\tilde{\boldsymbol{c}'}_i = \boldsymbol{c}'_i + q\boldsymbol{u}'$ with $\|\boldsymbol{u}'\|_\infty = 1$). In this case r_∞ would increase to $3(\delta_{\mathcal{R}} B_{key} + 1)/2 + 1$. As explained in [11, Section 4.5], this would only impact the size of $\delta_{\mathcal{R}} t \, (r_\infty + 1/2) \, (\|\boldsymbol{v}\|_\infty + \|\boldsymbol{v}'\|_\infty)$ by less than 3% in the worst case. Considering that this error occurs with probability smaller than $4n \cdot 2^{-51} \leq 2^{-43}$ on each multiplication ($n \leq 2^{16}$, on our parameters), its effect on the noise growth in practice would be negligible.

After having performed the products, the carries of $\widehat{\boldsymbol{c}}_{0,d-1}$ and $\widehat{\boldsymbol{c}}_{1,d-1}$ are computed to achieve an accurate approximation. These can be stored apart in coefficient representation and added after the relinearisation step to the final output. The computation of the two carries requires $2(2k+1)$ $\texttt{NTT}^{-1}$ to leave the $\texttt{NTT}$ representation, and $2 \times \mathcal{C}_{1\_\text{carry}}$. Therefore, at this point the three elements ciphertext has the $2(d-1)$ digits of $\hat{\boldsymbol{c}}_0$ and $\hat{\boldsymbol{c}}_1$ and the d digits of $\hat{\boldsymbol{c}}_2$ in $\texttt{NTT}$ representation; and the two carries stored apart in coefficient representation.

Practical complexity All in all, if we consider that each vector-matrix product is performed with a naive algorithm requiring $nkd(d+1)$ $\texttt{EMM}$, the homomorphic multiplication (before relinearisation) requires:

$$\begin{aligned}\mathcal{C}_{\text{mult.}} = n(k^2+3k+1)\texttt{EMMs} + n(k+1)\texttt{FPOs} + 4d(2k+1)\ \texttt{NTTs} \\ +2(2k+1)\ \texttt{NTTs}^{-1} + 3nd(2k+1)(d+3)/2\ \texttt{EMMs} + 2\cdot\mathcal{C}_{1\_\text{carry}}\end{aligned}$$

4.1 Relinearisation

As stated in Section 2.1, relinearisation starts by decomposing $\hat{\boldsymbol{c}}_2$ into a vector of digits with small norm, according to a function $\mathcal{D}$. These values are afterwards multiplied by the relinearisation-key, and added to $(\hat{\boldsymbol{c}}_0, \hat{\boldsymbol{c}}_1)$. The relinearisation-key contains pseudo-encryptions of $\mathfrak{P}(s^2)$, for a function $\mathfrak{P}$ satisfying $\left[\langle \mathfrak{D}(\hat{\boldsymbol{c}}_2), \mathfrak{P}(\boldsymbol{s}^2)\rangle\right]_q = [\hat{\boldsymbol{c}}_2 \cdot \boldsymbol{s}^2]_q$. In the end, a two-element ciphertext is produced encrypting $[\boldsymbol{m}\cdot\boldsymbol{m}']_t$. The main goal of this section is to set up efficient $\mathfrak{D}$ and $\mathfrak{P}$ functions that take advantage of the properties of HPR.

The value of $\hat{\boldsymbol{c}}_2$ is represented as an array of d digits resulting from the computation in Proposition 1: $\sum_{i=-1}^{d-2} \hat{\boldsymbol{c}}_{2,i} p^i$. At the beginning of relinearisation, these digits are converted from the NTT to the canonical domain at a cost of $d(2k+1)$ $\texttt{NTT}^{-1}$. Then, carry propagation is applied and the digit of weight p^{-1} is discarded, producing a vector: $\hat{\boldsymbol{c}}'_2(p) = \sum_{i=0}^{d-1} \hat{\boldsymbol{c}}'_{2,i} p^i$. By construction, the digits $\boldsymbol{a} = ([\boldsymbol{a}'_{0,\mathcal{P}}, \boldsymbol{a}'_{0,\mathcal{B}}], \ldots, [\boldsymbol{a}'_{d-1,\mathcal{P}}, \boldsymbol{a}'_{d-1,\mathcal{B}}])_{\text{HPR}}$ that result from propagating carries satisfy:

$$\left[Y^i f^{-1}_{\mathcal{P}\cup\mathcal{B}}([\boldsymbol{a}'_{i,\mathcal{P}}, \boldsymbol{a}'_{i,\mathcal{B}}])\right]_q = \left[Y^i \texttt{FBE}(\boldsymbol{a}'_{i,\mathcal{P}}, \mathcal{P}, \mathbb{Z})\right]_q \tag{17}$$

where $f^{-1}_{\mathcal{P}\cup\mathcal{B}}$ corresponds to reverting the RNS representation with respect to the $(\mathcal{P}, \mathcal{B})$ basis (see (4)). Concretely, this means that once the carry propagation has been done, the residues of $\hat{\boldsymbol{c}}'_2$ in base $\mathcal{B}$ are no longer required for the relinearisation process. Based on (17), $[\hat{\boldsymbol{c}}'_2 \cdot \boldsymbol{s}^2]_q$ may be rewritten as:

$$[\hat{\boldsymbol{c}}'_2 \cdot \boldsymbol{s}^2]_q = \left[\sum_{i=0}^{d-1} \texttt{FBE}(\hat{\boldsymbol{c}}'_{2,i}, \mathcal{P}, \mathbb{Z}) \cdot \boldsymbol{s}^2 Y^i\right]_q = \left[\sum_{i=0}^{d-1}\sum_{j=1}^{k} \left[\hat{\boldsymbol{c}}'_{2,i}\frac{p_j}{p}\right]_{p_j} \frac{p}{p_j} Y^i \boldsymbol{s}^2\right]_q . \tag{18}$$

Thus, $\mathfrak{D}(\hat{\boldsymbol{c}}_2)$ is defined as the vector $(\boldsymbol{r}_{i,j})$ satisfying: $\boldsymbol{r}_{i,j} = \left[\hat{\boldsymbol{c}}'_{2,i}\frac{p_j}{p}\right]_{p_j}$ and $\mathfrak{P}(\boldsymbol{s}^2)$ is defined as the vector $(\boldsymbol{\psi}_{i,j})$ with $\boldsymbol{\psi}_{i,j} = \left[Y^i \frac{p}{p_j}\boldsymbol{s}^2\right]_q$, for $0 \leqslant i \leqslant d-1$ and $1 \leqslant j \leqslant k$. Clearly, the $\|\boldsymbol{r}_{i,j}\|_\infty < p_j/2$ are small, and $\left[\langle \mathfrak{D}(\hat{\boldsymbol{c}}'_2), \mathfrak{P}(\boldsymbol{s}^2)\rangle\right]_q = [\hat{\boldsymbol{c}}'_2 \cdot \boldsymbol{s}^2]_q$.

As explained in Section 2.1, the relinearisation-key ($\overrightarrow{\texttt{rlk}}$) is composed of pseudo-encryptions of the $\boldsymbol{\psi}_{i,j}$ values. To produce the pseudo-encryption with respect to $\boldsymbol{\psi}_{i,j}$, a Ring Learning with Errors (RLWE) sample $([-(\boldsymbol{a}_{i,j} \cdot \boldsymbol{s} + \boldsymbol{e}_{i,j}]_q)$, $\boldsymbol{a}_{i,j})$ is drawn, and $\frac{p}{p_j} \cdot \boldsymbol{s}^2$ is added to the i-th digit of the first element of the pair. Then, the whole relinearisation key is stored in `NTT` form.

Remark 3. Note that, since $\|\boldsymbol{s}^2\|_\infty \leqslant \delta_{\mathcal{R}} B_{key}^2 < p_j$ for moduli p_j of practical size, $p/p_j \boldsymbol{s}^2$ is representable as a single HPR digit. Thus, the RLWE samples used to mask the $\boldsymbol{\psi}_{i,j} = \frac{p}{p_j} \cdot Y^i \cdot \boldsymbol{s}^2$, for a fixed i and $1 \leq j \leq k$, do not need to have a degree higher than i in Y. The $d-1-i$ most significant digits of the k elements of the relinearisation-key associated with these $\psi_{i,j}$ are set to zero. In practice, RLWE samples are drawn with respect to a modulus $p^i \leqslant p^d$, but with an error of same standard deviation σ_{err}. Thus, the security of the scheme is not reduced. This technique allows us to not only reduce the size of $\overrightarrow{\texttt{rlk}}$, but also to further reduce the practical complexity of the relinearisation.

Remark 4. As in [5, Appendix B1], it is still possible to use another level of decomposition by using the binary representation of the $\boldsymbol{r}_{i,j}$s. It may also be possible to use the ciphertext truncation methods from [7, Section 5.4] and/or the key-size reduction as explained in [5, Section 4.6] by applying them to the least significant digits. The method from [7] would consist in truncating $\boldsymbol{c}_2'$ by p^i, i.e. of removing the i least significant digits of $\boldsymbol{c}_2'$ and of $\overrightarrow{\texttt{rlk}}$. With the method from [5] we would ignore certain residues of the digits by setting them to zero. It should be noted that both methods will still cause a larger noise growth during the relinearisation.

When a small polynomial $\boldsymbol{r}_{i,j}$ has been extracted from a residue modulo p_j, it is easily extended to the base $(\mathcal{P}, \mathcal{B}_{\text{sk}})$ through a simple copy-paste operation. Then, since its representation is a single digit, the computation of its `NTT` form only requires $(2k+1)$ `NTT`s. In the next step, the $\boldsymbol{r}_{i,j}$s are multiplied by the corresponding $\overrightarrow{\texttt{rlk}}$ elements and the results are accumulated in $\hat{\boldsymbol{c}}_0, \hat{\boldsymbol{c}}_1$.

The relinearisation step causes an increase in the noise associated with the ciphertext. More concretely, for a homomorphic multiplication result $(\boldsymbol{c}_0, \boldsymbol{c}_1)$

$$\begin{aligned}
[\boldsymbol{c}_0 + \boldsymbol{c}_1 \boldsymbol{s}]_q &= \left[\hat{\boldsymbol{c}}_0 + \hat{\boldsymbol{c}}_1 \cdot \boldsymbol{s} + \langle \mathfrak{D}(\hat{\boldsymbol{c}}_2'), \mathfrak{P}(\boldsymbol{s}^2) - (\overrightarrow{\boldsymbol{e}} + \overrightarrow{\boldsymbol{a}} \cdot \boldsymbol{s}) \rangle + \langle \mathfrak{D}(\hat{\boldsymbol{c}}_2'), \overrightarrow{\boldsymbol{a}} \rangle \cdot \boldsymbol{s} \right]_q \\
&= \left[\hat{\boldsymbol{c}}_0 + \hat{\boldsymbol{c}}_1 \cdot \boldsymbol{s} + \hat{\boldsymbol{c}}_2' \cdot \boldsymbol{s}^2 - \langle \mathfrak{D}(\hat{\boldsymbol{c}}_2'), \overrightarrow{\boldsymbol{e}} \rangle \right]_q \\
&= \left[\hat{\boldsymbol{c}}_0 + \hat{\boldsymbol{c}}_1 \cdot \boldsymbol{s} + \hat{\boldsymbol{c}}_2' \cdot \boldsymbol{s}^2 + \boldsymbol{e}_{\text{relin}} \right]_q
\end{aligned}$$

The extra noise caused by this step is bounded by:

$$\|\boldsymbol{e}_{\text{relin}}\|_\infty \leqslant \delta_{\mathcal{R}} \cdot d \cdot B_{err} \cdot (p_1 + \cdots + p_k) \tag{19}$$

Practical complexity After the carry propagation of $\boldsymbol{c}_2$, we have to compute the `NTT` of 1-digit polynomials $\boldsymbol{r}_{i,j}$ in base $(\mathcal{P}, \mathcal{B}_{\text{sk}})$. Since there are $d \times k$ such polynomials, this step requires $(2k+1) \cdot dk$ `NTT`s. Then, for each i, the single-digit

polynomial $\boldsymbol{r}_{i,j}$ is computed with n elementary products and is multiplied by a relinearisation-key element of degree i, resulting in a total of $d(d+1)\times(2k+1)\times n$ elementary products. The results of these products are added to $\widehat{\boldsymbol{c}}_0$ and $\widehat{\boldsymbol{c}}_1$. Once all intermediate products have been accumulated, $2 \times d(2k+1)$ $\texttt{NTT}^{-1}$s are used to obtain the result in the canonical domain. Finally, carry propagation is applied. This operation limits the size of the ciphertext polynomials digits, enabling the application of further homomorphic multiplications or decryption.

Thus, the total cost of the relinearisation step is:

$$\begin{aligned}\mathcal{C}_{\text{relin.}} = {} & dk(2k+1)\ \texttt{NTTs} + 3d(2k+1)\ \texttt{NTTs}^{-1} \\ & + dkn((d+1)(2k+1)+1)\ \texttt{EMMs} + (3d-2)\mathcal{C}_{1\_\text{carry}}\end{aligned} \tag{20}$$

Remark 5. As for any other variant of FV, the bit size of the noise grows linearly with the multiplicative depth L. The maximal depth $L_{\max}$ is determined by the worst case scenario about the size of fresh noise and is reached when the next multiplication makes the noise larger than $\simeq \frac{\Delta}{2}$ (see Lem. 2). Consequently, when the current depth L reaches around $\frac{L_{\max}}{d}$, the noise affects more than one digit in the worst case scenario. Knowing that, it is possible to decrease the asymptotic and practical complexity of the relinearisation by using a coarse grain decomposition.

Instead of relinearising ciphertexts at the residue level, the residues can be combined into a single digit during the decomposition step. In other words, the sum with index j in (18) can be computed by the decomposition function prior to the product with the relinearisation key. With this technique, the decomposition polynomials become $\boldsymbol{r}_i = \sum_{j=1}^{k} \left[\widehat{\boldsymbol{c}}'_{2,i}\frac{p_j}{p}\right]_{p_j} \times \frac{p}{p_j}$, and the new relinearisation key contains $\boldsymbol{\psi}_i = \left[Y^i \boldsymbol{s}^2\right]_q$.

Since $\boldsymbol{r}_i$ is not a single residue, its conversion toward base $\mathcal{B}$ is not a simple copy-paste. However, the carry propagation performed prior to the relinearisation already involves such a base conversion. So, $\boldsymbol{r}_i$ is obtained in the full RNS base at no extra cost. Moreover, one may disregard an increasingly larger number of least significant digits during relinearisation as noise grows. If one assumes that only the d' most significant digits are taken into account for $\hat{\boldsymbol{c}}'_{2,i}$ (in the case where $L \simeq (d-d')\frac{L_{\max}}{d}$), the total cost boils down to

$$\begin{aligned}\mathcal{C}_{\text{relin.},d'} = {} & d'(k+\ell+1)\ \texttt{NTTs} + 3d'(k+\ell+1)\ \texttt{NTTs}^{-1} \\ & + nd'(d'+1)(k+\ell+1)\ \texttt{EMMs} + (3d'-2)\mathcal{C}_{1\_\text{carry}}\end{aligned} \tag{21}$$

Notice that a smoother approach may be taken by progressively reducing the number of considered residues of the least significant digit used for relinearisation, instead of entirely removing whole digits at once.

4.2 Overall Procedure

An overview of the proposed HPR-based homomorphic multiplication procedure can be found in Fig. 1. Two ciphertexts $(\boldsymbol{c}_0, \boldsymbol{c}_1), (\boldsymbol{c}'_0, \boldsymbol{c}'_1) \in \mathcal{R}_q^2$, encrypting $[\boldsymbol{m}]_t$

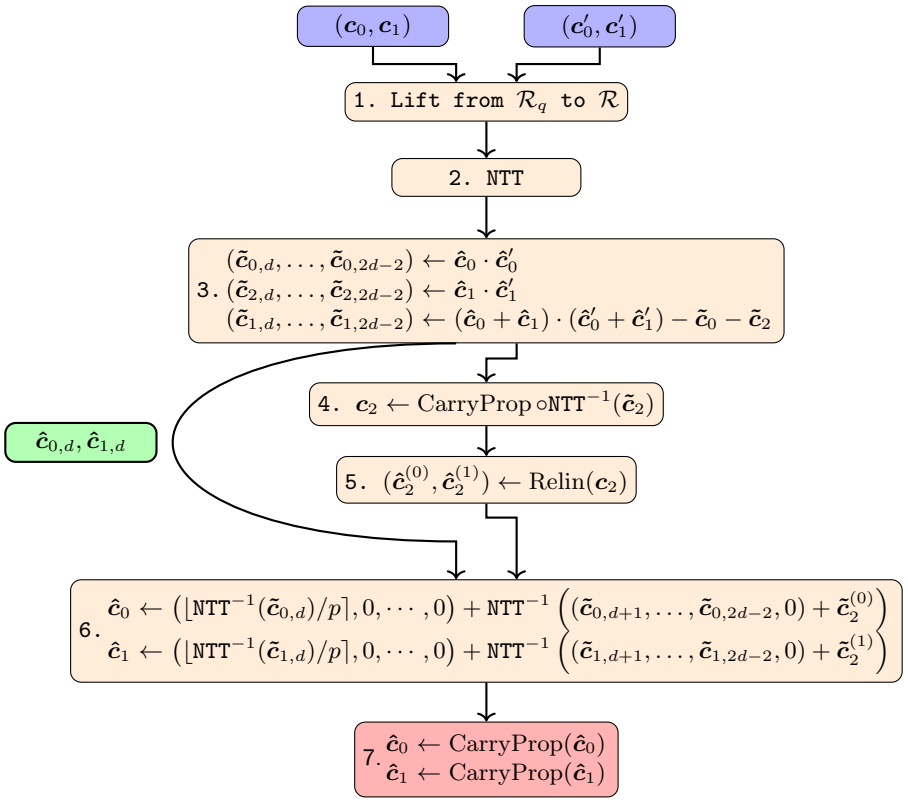


Fig. 1: HPR-based homomorphic multiplication procedure: the ciphertexts $(\boldsymbol{c}_0, \boldsymbol{c}_1)$ and $(\boldsymbol{c}'_0, \boldsymbol{c}'_1)$ encrypting $[\boldsymbol{m}]_t$ and $[\boldsymbol{m}']_t$ are homomorphic multiplied to produce $(\widehat{\boldsymbol{c}}_0, \widehat{\boldsymbol{c}}_1)$ encrypting $[\boldsymbol{m} \cdot \boldsymbol{m}']_t$.

and $[\boldsymbol{m}']_t$, are the input. Since the first product between the ciphertexts' elements must not be reduced modulo q, $\boldsymbol{c}_0$, $\boldsymbol{c}_1$, $\boldsymbol{c}'_0$ and $\boldsymbol{c}'_1$ are lifted to $\mathcal{R}$ by extending their most significant digits to $(\mathcal{P}, \mathcal{B}_{\mathrm{sk}})$. This is done in Step 1 through the base extension proposed in [11].

Then, the digits of $(\boldsymbol{c}_0, \boldsymbol{c}_1), (\boldsymbol{c}'_0, \boldsymbol{c}'_1)$ are converted to the NTT domain in Step 2, as a preparation for the computation of $(\widetilde{\boldsymbol{c}}_0, \widetilde{\boldsymbol{c}}_1, \widetilde{\boldsymbol{c}}_2) = (\lfloor t/q\boldsymbol{c}_0 \cdot \boldsymbol{c}_0 \rceil, \lfloor t/q(\boldsymbol{c}_0 \cdot \boldsymbol{c}'_1 + \boldsymbol{c}_1 \cdot \boldsymbol{c}'_0) \rceil, \lfloor t/q\boldsymbol{c}_1 \cdot \boldsymbol{c}'_1 \rceil)$. One of the advantages of the HPR is that, since $p^d = q$, a division by q can be achieved by discarding the d least significant digits. More concretely, we only need to compute the d most significant digits of the $\boldsymbol{c}'_i, \boldsymbol{c}'_j$ products in Step 3. Due to the regularity of the operations, these can be achieved with a Vector-Toeplitz Matrix Product (VTMP) with a reduced complexity. However, for the small values of d considered in this article, a naive vector-matrix multiplication is of sufficient performance. We keep the first of the d produced digits, of weight p^{-1}, to improve accuracy and we discard it after its carry has been computed. Notice that by adding the carries at the end of the algorithm, as depicted in Fig. 1 in Step 6, instead of when they are computed, the amount of NTTs is further reduced.

Afterwards, the three-element ciphertext $(\widetilde{\boldsymbol{c}}_0, \widetilde{\boldsymbol{c}}_1, \widetilde{\boldsymbol{c}}_2)$ is converted back to a two-elements ciphertext. First, we convert $\widetilde{\boldsymbol{c}}_2$ from the NTT to the canonical domain and propagate its carries in Step 4. Then, we can use the residues of $\widetilde{\boldsymbol{c}}_2$ in basis $\mathcal{P}$ to extract the small elements $\boldsymbol{r}_{i,j} = [\widetilde{\boldsymbol{c}}_2 p_j/p]_{p_j}$ which will be used for the relinearisation. Unlike previous RNS approaches where, even though the norm of the elements of $\mathcal{D}_{RNS}(\widetilde{\boldsymbol{c}}'_2)$ was small, their representation was large, here the elements of $\mathcal{D}(\widetilde{\boldsymbol{c}}'_2)$ are represented as a single HPR digit. This difference allows to reduce by a factor of d the number of NTTs one needs to compute. The inner products of $\mathcal{D}(\widetilde{\boldsymbol{c}}'_2)$ and $\overrightarrow{\texttt{rlk}}$ are computed in Step 5, and the result is accumulated in $(\widehat{\boldsymbol{c}}_0, \widehat{\boldsymbol{c}}_1)$ in Step 6.

Finally, we convert the result back to the canonical domain, and a full carry propagation is conducted to reduce the size of the resulting coefficients. Lemma 4 quantifies the noise associated with this result.

Lemma 4. *Given two ciphertexts encrypting $[\boldsymbol{m}]_t$ and $[\boldsymbol{m}']_t$ with respective noises $\boldsymbol{v}$ and $\boldsymbol{v}'$, then the HPR-FV homomorphic multiplication provides an encryption of $[\boldsymbol{m} \cdot \boldsymbol{m}']_t$ with noise $\boldsymbol{v}_{mult}$ such that*

$$\begin{aligned} \|\boldsymbol{v}_{mult}\|_\infty \leqslant\ & \delta_{\mathcal{R}} t \left(r_\infty + \tfrac{1}{2}\right)\left(\|\boldsymbol{v}\|_\infty + \|\boldsymbol{v}'\|_\infty\right) + \tfrac{\delta_{\mathcal{R}}}{2} \min(\|\boldsymbol{v}\|_\infty, \|\boldsymbol{v}'\|_\infty) \\ & + \tfrac{1}{2} + t\left(\tfrac{1}{2} + t\delta_{\mathcal{R}}(1 + r_\infty)\right) + B_e\left(1 + \delta_{\mathcal{R}} B_{key} + \delta_R^2 B_{key}^2\right) \\ & + \delta_{\mathcal{R}} d B_{err}(p_1 + \cdots + p_k) \end{aligned} \tag{22}$$

Practical complexity By adding the costs of the multiplication and relinearisation procedures, homomorphic multiplication has the following computational cost:

$$\begin{aligned} \mathcal{C}_{\text{hom. mult.}} = & \ d(k+4)(2k+1)\ \texttt{NTTs} + (3d+2)(2k+1)\ \texttt{NTTs}^{-1} + 3 \cdot d\mathcal{C}_{1\_\text{carry}} \\ & + n(2k+1)\left(1 + 3d\tfrac{d+3}{2} + dk(d+1)\right) + nk(k+d+1)\ \texttt{EMMs} \\ & + n(k+1)\texttt{FPOs} \end{aligned}$$

5 Related Art

A first adaptation of FV [9] to the RNS was achieved in [5]. Due to the reduced complexity, speedups from 2 up to 4 and from 5 up to 20 were achieved for the homomorphic multiplication and decryption operations, respectively, in comparison to an implementation based on a generic multiprecision library. Later, [11] proposed modifications to [5], namely by computing part of the RNS basis extensions with floating point arithmetic.

There has been experimental work [4] comparing [11] and [5]. However, the performance results of [4] are of limited applicability. The λ factor in [5, Lemma 6] is therein ignored, leading to RNS bases larger than what was originally proposed in [5]. Moreover, implementation optimisations are ignored, like considering as an invariant that the polynomials are multiplied by x with $x = q_i/q \bmod q_i$ or $x = b_i/b \bmod b_i$. If these factors are included in the precomputations one needs to store, while more multiplications are needed to remove this factor from one of the operands before polynomial products are performed, the complexity of bases extensions is reduced, leading to an overall improvement in performance. Also, due to a reduction in the size of the constants one needs to store for the decryption operation of [11] (instead of $[q_i/q]_{q_i} q/q_i$, only the q/q_i need to be precomputed), the moduli bit-size restriction therein is removed. In addition, if one selects γ as a power of two and q_i with $q_i = 1 \bmod \gamma t$, a large number of the `EMMs` in the decryption of [5] are unnecessary.

It has also been noticed in [4] that for similar cryptographic parameters, [5] offers a lower multiplicative depth than [11]. While we were able to determine that this was partially due to the use in [4] of a positive remainder instead of a centred remained during the $\texttt{SmMRq}_{\tilde{m}}$ algorithm of [5], this did not completely solve the issue. Hence, as discussed in [4], we herein base our lifting from $\mathcal{R}_q$ to $\mathcal{R}$ on the basis extension proposed in [11] to fix this problem.

6 Complexity Analysis

In this section, a comparison between the complexity of the proposed HPR-based FV scheme and [5,11] is performed. Herein, K_{RNS} corresponds to the size of the RNS bases of [5,11]. Both [5,11] make use of secondary bases of size $K_{RNS} + 1$. With the proposed HPR-based scheme, each polynomial coefficient is associated with d digits, each of them represented in an RNS basis of size k, and a secondary RNS basis with $k+1$ moduli. The value of k is about d times smaller than K_{RNS}.

6.1 Decryption

A comparison of the costs of the decryption operation for the related art and the proposed scheme can be found in Table 1. For simplicity, we associate both forward and inverse NTTs with the NTTs column. In all schemes, the computational cost is dominated by the amount of NTTs. A radix-2 NTT requires $\frac{n}{2} \log_2 n$ `EMMs` [13]. In practice, [5,11] require K_{RNS} forward NTTs and K_{RNS}

inverse NTTs; whereas k forward NTTs and k inverse NTTs are required for the HPR-based decryption. As a consequence, the cost of decryption in HPR is nearly d times smaller than that of [5,11]. Since for security reasons K_{RNS} and $K \in \mathcal{O}(n)$ and that NTTs have a quasi-linear complexity, the asymptotic complexity for the decryption procedure is reduced from $\mathcal{C}(\texttt{Dec}_{\texttt{RNS}})$ for [5,11] to $\mathcal{C}(\texttt{Dec}_{\texttt{HPR}})$ for the proposed scheme:

$$\mathcal{C}(\texttt{Dec}_{\texttt{RNS}}) \in \mathcal{O}(n^2 \log n) \rightarrow \mathcal{C}(\texttt{Dec}_{\texttt{HPR}}) \in \mathcal{O}(n^{1+\varepsilon} \log n).$$

where $k = K^{\varepsilon}$, with $\varepsilon \in (0, 1)$. As a consequence, when the number of digits d increases, k and ε decrease, which results in a more efficient algorithm. In particular for $d = K$, and thus $k = 1$, we obtain a quasi-linear complexity.

6.2 Homomorphic Multiplication

Similarly to Section 6.1, a comparison of the number of operations required for the homomorphic multiplication for the three variants of the scheme is given in Table 2. As previously, since K_{RNS} and $K \in \mathcal{O}(n)$, the asymptotic complexity of [5,11] is:

$$\mathcal{C}(\texttt{FV-Mult}_{\texttt{RNS}}) \in \mathcal{O}(n^3 \log n),$$

while, assuming $k = K^{\varepsilon}$ and $d = K^{1-\varepsilon}$, for our HPR variant it is reduced to:

$$\mathcal{C}(\texttt{FV-Mult}_{\texttt{HPR}}) \in \mathcal{O}(n^{2+\varepsilon} \log n + n^3) = \mathcal{O}(n^3).$$

While asymptotically the choice of ε does not impact the complexity, in practice there is an optimal k_n, or equivalently d_n, that minimises the computational cost presented in Table 2.

Remark 6. By considering Remark 5, the asymptotic complexity of relinearisation when applying a digit-wise decomposition can be reduced to

$$\mathcal{O}(dkn \log(n) + nd^2k) = \mathcal{O}(n^2 \log(n) + n^{3-\varepsilon}) = \mathcal{O}(n^{3-\varepsilon}) \ \texttt{EMM}.$$

Scheme	NTTs	EMMs	FPOs
[11]	$2K_{RNS}$	nK_{RNS}	$n(1 + K_{RNS})$
[5]	$2K_{RNS}$	$3nK_{RNS}$	-
HPR	$2k$	$3nk$	-

Table 1: Time complexity of decryption in HPR and in [5,11] in terms of number of Number Theoretic Transforms (NTTs), Elementary Modular Multiplications (EMMs) and Floating Point Operations (FPOs). K_{RNS} denotes the number of moduli in the RNS bases of [5,11], while k is the number of moduli used to represent the digits in the HPR representation ($k \simeq K_{RNS}/d$)

Scheme	NTTs	EMMs	FPOs
[11]	$K_{RNS}^2 + 16K_{RNS} + 7$	$n(12K_{RNS}^2 + 63K_{RNS} + 49)$	$n(10K_{RNS} + 13)$
[5]	$K_{RNS}^2 + 16K_{RNS} + 7$	$n(10K_{RNS}^2 + 25K_{RNS} + 7)$	-
HPR	$(2k+1)(d(7+k)+2)$	$n(2k+1)\left(1 + 3d\frac{d+3}{2} + dk(d+1)\right) + nk(6dk + k + d + 4) + 6dn$	$n(k+1)$

Table 2: Time complexity of the homomorphic multiplication operation for the HPR-based scheme and those proposed in [11,5] in terms of the amounts of Number Theoretic Transforms (NTTs), Elementary Modular Multiplications (EMMs) and Floating Point Operations (FPOs). K_{RNS} corresponds to the number of moduli in the RNS bases of [11] and [5], respectively. The complexity of [11] was computed from [11, Table 2] by setting the size of the secondary RNS basis to $K_{RNS} + 1$. The complexity of [5] was replicated from [5, Appendix B3].

7 Experimental Evaluation

The NTT-based Fast Lattice (NFL) library [2] was adapted to handle moduli with the bitwidths featured in Table 3. The proposed HPR-based scheme and [5,11] were implemented in C++, using the modified NFL library [2][5]. They were compiled with gcc 8.3.0 and executed on a single thread on an Intel Core i7-6700K processor running at 4.0GHz with 32GB of main memory, operated by CentOS 7.3. Experimental results regarding the execution time of decryption and homomorphic multiplication for the proposed HPR-based scheme and [5,11], for the cryptographic parameters presented in Table 3, are depicted in Figure 2.

n	$\log_2 q$	K_{RNS}	$\log_2 q_i$ (RNS)	$K = k \times d$	$\log_2 p_i$ (HPR)
2^{13}	275	5	55	$5 = 1 \times 5$	55
2^{14}	549	9	61	$9 = 3 \times 3$	61
2^{15}	1098	18	61	$18 = 3 \times 6$	61
2^{16}	2196	36	61	$36 = 3 \times 12$	61

Table 3: Parameters achieving 100 bits of security according to [3] with $\sigma_{err} = 8.0$. [11,5] have RNS bases of size K_{RNS}; while for the HPR-based approach d digits are represented in bases of size k, such that $K = k \times d$.

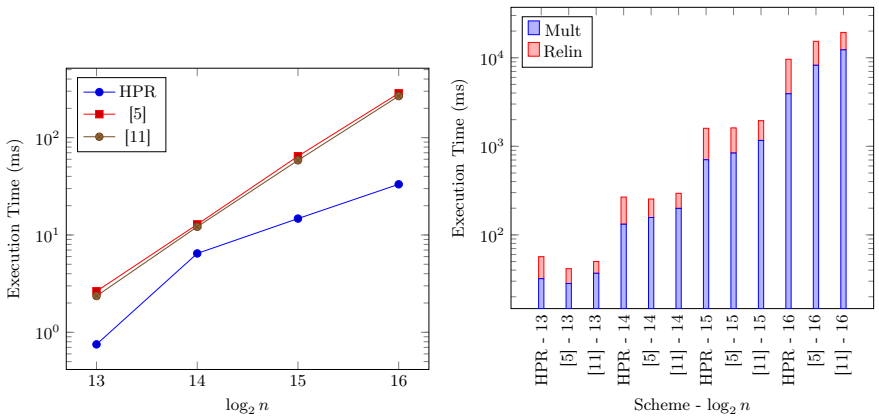
Asymptotically, decryption has a speed-up that is linear with n. This behaviour is more noticeable for $n \geq 2^{14}$ in Figure 2.a because therein $k = 1$ is featured for $n = 2^{13}$ and $k = 3$ for $n \geq 2^{14}$, since k was chosen to minimise the complexity of homomorphic multiplication. Maximum speed-ups of 8.6 and 8.0 are achieved when the proposed HPR-based scheme is compared with [5] and

[5] The code has been made available at `https://gitlab.com/fvhpr/hare`

[11], respectively, for $n = 2^{16}$. In certain situations, for instance when decryption is executed in constrained devices, it might be beneficial to choose a smaller, suboptimal, k to further reduce the execution time of decryption, at the cost of less efficient homomorphic multiplications.

Figure 2.b suggests that the theoretically predicted $\mathcal{O}(\log n)$ speed-up is only achieved in practice for large parameters. While the novel HPR-based representation brings performance improvements to the multiplication part of the procedure (i.e. without relinearisation), that are noticeable for most parameters, the complexity improvements pertaining relinearisation only take effect for $n \geq 2^{16}$. This suggests that the proposed HPR-based scheme is preferable to [5,11] for applications relying on sums of products, wherein relinearisation is only applied once, after computing the homomorphic sum of the optimised products, or when large parameters ($n \geq 2^{16}$) are considered. Maximum speed-ups regarding the whole procedure of 2.0 and 1.6 are achieved when the proposed scheme is compared with [11] and [5], respectively.

Finally, since both [11] and the proposed scheme lift polynomials from $\mathcal{R}_q$ to $\mathcal{R}$ using the floating-point based RNS basis extension, similar multiplicative depths are achieved for both schemes [4]. More concretely, we have measured the depth reached in practice by squaring several ciphertexts until they did not decrypt correctly and kept the smallest value as the practical multiplicative depth. Experimental results show that, for the parameters in Table 3, the proposed HPR-based scheme supports circuits with multiplicative depths of 15, 32, 64, and 125, while [11] supports depths of 16, 32, 64, and 125, for $n = 2^{13}$, $n = 2^{14}$, $n = 2^{15}$ and $n = 2^{16}$, respectively. In contrast, [5] achieves multiplicative depths of 15, 31, 64, and 124, for $n = 2^{13}$, $n = 2^{14}$, $n = 2^{15}$ and $n = 2^{16}$, respectively.



(a) Decryption (b) Homomorphic Multiplication

Fig. 2: Execution time (ms) of decryption and homomorphic multiplication for the HPR-based scheme and [5,11] on an i7-6700K for the cryptographic parameters in Table 3. The y axes are logarithmic

8 Conclusions

Whereas state-of-the-art implementations of FV make use of the RNS, the FV scheme requires operations that are inefficient to implement in this representation. In particular, division and rounding requires large basis extensions, and the representation of small values is ample. Herein, an alternative approach to accelerate FV was proposed, supported on the HPR. The HPR is an hybrid representation system that inherits characteristics of positional systems, while representing its digits in RNS to benefit from its arithmetic parallelism. Divisions are approximated by considering only the most significant digits of a value. Since decryption requires a division operation, the HPR-based system significantly reduces the amount of NTTs one has to compute when compared with a pure RNS approach, reducing its asymptotic complexity from $\mathcal{O}(n^2 \log n)$ to $\mathcal{O}(n \log n)$. Since the homomorphic multiplication operation also depends on division and rounding, it also benefits from the use of the HPR. Moreover, during the relinearisation step of homomorphic multiplication, small values are operated on. With the HPR, small values can be represented as a single digit, leading to a significant reduction in the amount of NTTs one has to compute to operate on them. As a result, the asymptotic complexity of homomorphic multiplication is also reduced, namely from $\mathcal{O}(n^3 \log n)$ to $\mathcal{O}(n^3)$, when comparing to RNS-based approaches. When compared with two state-of-the-art schemes, experimental speedups of up to 8.0 and 8.6 for the decryption operation and of up to 2.0 and 1.6 for the homomorphic multiplication are achieved. The experimental results also confirm that the speed-up increases as n grows.

Acknowledgement

This work was partially supported by Portuguese funds through Fundação para a Ciência e a Tecnologia (FCT) with reference UID/CEC/50021/2019 and by the Ph.D. grant with reference SFRH/BD/103791/2014; through the Pessoa/Hubert Curien programme with reference 4335 (FCT)/40832XC (CAMPUSFRANCE); and by EU's Horizon 2020 research and innovation programme under grant agreement No. 779391 (FutureTPM).

References

1. Adida, B.: Helios: trust the vote. `https://vote.heliosvoting.org/` (2015)
2. Aguilar-Melchor, C., Barrier, J., Guelton, S., Guinet, A., Killijian, M.O., Lepoint, T.: Topics in Cryptology - CT-RSA 2016: The Cryptographers' Track at the RSA Conference 2016, San Francisco, CA, USA, February 29 - March 4, 2016, Proceedings, chap. NFLlib: NTT-Based Fast Lattice Library, pp. 341–356. Springer International Publishing, Cham (2016). https://doi.org/10.1007/978-3-319-29485-8, `http://dx.doi.org/10.1007/978-3-319-29485-8_20`
3. Albrecht, M.R., Player, R., Scott, S.: On the concrete hardness of Learning with Errors. Journal of Mathematical Cryptology **9**, 169–203 (October 2015)

4. Badawi, A.A., Polyakov, Y., Aung, K.M.M., Veeravalli, B., Rohloff, K.: Implementation and performance evaluation of rns variants of the bfv homomorphic encryption scheme. Cryptology ePrint Archive, Report 2018/589 (2018), https://eprint.iacr.org/2018/589
5. Bajard, J.C., Eynard, J., Hasan, M.A., Zucca, V.: A Full RNS Variant of FV Like Somewhat Homomorphic Encryption Schemes. In: Avanzi, R., Heys, H. (eds.) Selected Areas in Cryptography – SAC 2016. pp. 423–442. Springer International Publishing, Cham (2017)
6. Bigou, K., Tisserand, A.: Hybrid Position-Residues Number System. In: Hormigo, J., Oberman, S., Revol, N. (eds.) ARITH: 23rd Symposium on Computer Arithmetic. IEEE, Santa Clara, CA, United States (Jul 2016), https://hal.inria.fr/hal-01314232
7. Bos, J., Lauter, K., Loftus, J., Naehrig, M.: Improved Security for a Ring-Based Fully Homomorphic Encryption Scheme. In: Stam, M. (ed.) Cryptography and Coding, Lecture Notes in Computer Science, vol. 8308, pp. 45–64. Springer Berlin Heidelberg (2013). https://doi.org/10.1007/978-3-642-45239-0, http://dx.doi.org/10.1007/978-3-642-45239-0_4
8. Damgard, I., Pastro, V., Smart, N.P., Zakarias, S.: Multiparty computation from somewhat homomorphic encryption. Cryptology ePrint Archive, Report 2011/535 (2011), http://eprint.iacr.org/
9. Fan, J., Vercauteren, F.: Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive, Report 2012/144 (2012), http://eprint.iacr.org/
10. Gennaro, R., Gentry, C., Parno, B.: Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In: Rabin, T. (ed.) Advances in Cryptology – CRYPTO 2010, Lecture Notes in Computer Science, vol. 6223, pp. 465–482. Springer Berlin Heidelberg (2010). https://doi.org/10.1007/978-3-642-14623-7, http://dx.doi.org/10.1007/978-3-642-14623-7_25
11. Halevi, S., Polyakov, Y., Shoup, V.: An Improved RNS Variant of the BFV Homomorphic Encryption Scheme. Cryptology ePrint Archive, Report 2018/117 (2018), https://eprint.iacr.org/2018/117
12. Martins, P., Sousa, L.: Enhancing data parallelism of fully homomorphic encryption. In: Hong, S., Park, J.H. (eds.) Information Security and Cryptology – ICISC 2016. pp. 194–207. Springer International Publishing, Cham (2017)
13. Pöppelmann, T., Güneysu, T.: Towards efficient arithmetic for lattice-based cryptography on reconfigurable hardware. In: Hevia, A., Neven, G. (eds.) Progress in Cryptology – LATINCRYPT 2012. pp. 139–158. Springer Berlin Heidelberg, Berlin, Heidelberg (2012)
14. Shenoy, A.P., Kumaresan, R.: Fast Base Extension Using a Redundant Modulus in RNS. IEEE Trans. Comput. **38**(2), 292–297 (Feb 1989). https://doi.org/10.1109/12.16508, http://dx.doi.org/10.1109/12.16508
15. Sousa, L., Antao, S., Martins, P.: Combining residue arithmetic to design efficient cryptographic circuits and systems. IEEE Circuits and Systems Magazine **16**(4), 6–32 (Fourthquarter 2016). https://doi.org/10.1109/MCAS.2016.2614714

A Proof of Lemma 1

A valid encryption of $[\boldsymbol{m}]_t$ satisfies $\boldsymbol{c}_0 + \boldsymbol{c}_1 \cdot \boldsymbol{s} = \Delta[\boldsymbol{m}]_t + \boldsymbol{v} + \boldsymbol{u}q$. Thus we have that the approximation in (7) satisfies:

$$[\boldsymbol{c}_{0,d-1} + \boldsymbol{c}_{1,d-1}\boldsymbol{s}]_p = \frac{\Delta[\boldsymbol{m}]_t + \boldsymbol{v} + q\boldsymbol{u}}{p^{d-1}} - \sum_{i=0}^{d-2}(\boldsymbol{c}_{0,i} + \boldsymbol{c}_{1,i} \cdot \boldsymbol{s})p^{i-d+1} + \boldsymbol{\epsilon}p$$

with $\boldsymbol{e} = -\sum_{i=0}^{d-2}(\boldsymbol{c}_{0,i} + \boldsymbol{c}_{1,i} \cdot \boldsymbol{s})p^{i-d+1}$. The norm of $\boldsymbol{e}$ can thus be bounded as

$$\|\boldsymbol{e}\|_\infty \leq \sum_{i=0}^{d-2}(\beta p + \delta_{\mathcal{R}}\beta p B_{\text{key}})p^{i-d+1} \leq \frac{p}{p-1}\beta(1 + \delta_{\mathcal{R}}B_{\text{key}}).$$

□

B Proof of Lemma 2

First, we show that if

$$\|\tilde{\boldsymbol{e}}\|_\infty < \frac{1}{2}\left(1 - \frac{k}{\gamma}\right) \tag{23}$$

is satisfied then $\lfloor \tilde{\boldsymbol{e}} \rceil = 0$ and $\left\| \left\lfloor \gamma \left[p\tilde{\boldsymbol{e}} \right]_p / p \right\rceil - \boldsymbol{\alpha} \right\|_\infty < \gamma/2$. In particular, $\lfloor \tilde{\boldsymbol{e}} \rceil = 0$ since $k, \gamma > 0$ and hence $\|\tilde{\boldsymbol{e}}\|_\infty < 1/2$. In addition, $[p\tilde{\boldsymbol{e}}]_p = p\tilde{\boldsymbol{e}}$ since $\|p\tilde{\boldsymbol{e}}\|_\infty < p/2$. Thus,

$$\begin{aligned}\left\| \left\lfloor \gamma \left[p\tilde{\boldsymbol{e}} \right]_p / p \right\rceil - \boldsymbol{\alpha} \right\|_\infty &= \left\| \frac{\gamma p\tilde{\boldsymbol{e}} - [\gamma p\tilde{\boldsymbol{e}}]_p}{p} - \boldsymbol{\alpha} \right\|_\infty \\ &< \frac{\gamma p(1 - k/\gamma) + p}{2p} + \frac{k-1}{2} \leq \gamma/2\end{aligned}$$

Finally, it will be shown that (11) implies (23). This is achieved by exploiting (9) to bound $\|\tilde{\boldsymbol{e}}\|_\infty$, and as consequence $\|\boldsymbol{v}\|_\infty$:

$$\|\tilde{\boldsymbol{e}}\|_\infty \leqslant \frac{t}{q}\|\boldsymbol{v}\|_\infty + \frac{t|q|_t}{2q} + \frac{t}{p-1}\beta(1 + \delta_{\mathcal{R}}B_{\text{key}}) < \frac{1}{2}\left(1 - \frac{k}{\gamma}\right).$$

Through a manipulation of the previous expression, (11) is finally reached:

$$\begin{aligned}&\frac{t}{q}\|\boldsymbol{v}\|_\infty + \frac{t|q|_t}{2q} + \frac{t}{p-1}\beta(1 + \delta_{\mathcal{R}}B_{\text{key}}) < \frac{1}{2}\left(1 - \frac{k}{\gamma}\right) \\ \Leftrightarrow &\|\boldsymbol{v}\|_\infty < \frac{q}{2t}\left(1 - \frac{k}{\gamma} - \frac{2t\beta}{p-1}(1 + \delta_{\mathcal{R}}B_{key})\right) - \frac{|q|_t}{2}\end{aligned}$$

□

C Proof of Lemma 3

For $i \in \{0,1,2\}$ we have by definition: $\widetilde{\boldsymbol{c}}_i = \sum_{j=0}^{2d-2} \widetilde{\boldsymbol{c}}_{i,j} \cdot p^j$. Moreover, given that the digits of $\boldsymbol{c}_l$ and $\boldsymbol{c}'_l$, for $l = 0, 1$, have their norm smaller than βp then: for $i \in \{0,2\}$ and for $j \in \{0, \ldots, d-2\}$, $\|\widetilde{\boldsymbol{c}}_{i,j}\|_\infty \leqslant \delta_{\mathcal{R}} t(j+1)\beta^2 p^2$; and for $j \in \{0, \ldots, d-2\}, \|\widetilde{\boldsymbol{c}}_{1,j}\|_\infty \leqslant 6\delta_{\mathcal{R}} t(j+1)\beta^2 p^2$. Now by using the second bound for every $\widetilde{\boldsymbol{c}}_{i,j}$ we can write:

$$\begin{aligned}
&\|\widetilde{\boldsymbol{c}}_{i,0} + \cdots + \widetilde{\boldsymbol{c}}_{i,d-2} \cdot p^{d-2} + ([\widetilde{\boldsymbol{c}}_{i,d-1}]_p + p\boldsymbol{\alpha}_{i,d-1}) \cdot p^{d-1}\|_\infty \\
&\leqslant 6\delta_{\mathcal{R}} t\beta^2 p^2 (1 + 2p + \cdots + (d-1)p^{d-2}) + \left(\frac{1}{2} + \frac{k-1}{2}\right) p^d \\
&\leqslant 6\delta_{\mathcal{R}} t\beta^2 p^2 \left(\frac{(d-1)p^{d-1}}{p-1} - \frac{p^d - 1}{p(p-1)^2}\right) + \frac{k}{2} p^d.
\end{aligned}$$

Recalling that $p^d = q$, we have:

$$\begin{aligned}
&\left(\left\lfloor \frac{\widetilde{\boldsymbol{c}}_{i,d-1}}{p} \right\rceil - \boldsymbol{\alpha}_{i,d-1} + \widetilde{\boldsymbol{c}}_{i,d} + \widetilde{\boldsymbol{c}}_{i,d+1} \cdot p + \cdots + \widetilde{\boldsymbol{c}}_{i,2d-2} \cdot p^{d-2}\right) \\
&= \frac{\widetilde{\boldsymbol{c}}_i}{q} - \frac{\widetilde{\boldsymbol{c}}_{i,0} + \cdots + \widetilde{\boldsymbol{c}}_{i,d-2} \cdot p^{d-2} + [\widetilde{\boldsymbol{c}}_{i,d-1}]_p \cdot p^{d-1} + \boldsymbol{\alpha}_{i,d-1} \cdot p^d}{p^d}
\end{aligned}$$

and then we deduce (14) with:

$$\boldsymbol{e}_i = -\frac{\widetilde{\boldsymbol{c}}_{0,0} + \cdots + \widetilde{\boldsymbol{c}}_{0,d-2} \cdot p^{d-2} + [\widetilde{\boldsymbol{c}}_{0,d-1}]_p \cdot p^{d-1} + \boldsymbol{\alpha}_{0,d-1} \cdot p^d}{p^d}.$$

In particular, we obtain (15): $\|\boldsymbol{e}_i\|_\infty \leqslant 6\delta_{\mathcal{R}} t\beta^2 p \left(\frac{d-1}{p-1} - \frac{1-p^{-d}}{(p-1)^2}\right) + \frac{k}{2}$. □

D Proof of Proposition 1

Knowing that $\Delta t = q - |q|_t$, $[\boldsymbol{m}]_t \cdot [\boldsymbol{m}']_t = [\boldsymbol{m} \cdot \boldsymbol{m}']_t + t\boldsymbol{r}_m$ and $\boldsymbol{v} \cdot \boldsymbol{v}' = \Delta \boldsymbol{r}_v + [\boldsymbol{v} \cdot \boldsymbol{v}']_\Delta$, we can write:

$$\begin{aligned}
&\frac{t}{q}(\boldsymbol{c}_0 + \boldsymbol{c}_1 \cdot \boldsymbol{s}) \cdot (\boldsymbol{c}'_0 + \boldsymbol{c}'_1 \cdot \boldsymbol{s}) = \frac{t}{q}(\Delta \cdot [\boldsymbol{m}]_t + \boldsymbol{v} + \boldsymbol{r}q) \cdot (\Delta \cdot [\boldsymbol{m}']_t + \boldsymbol{v}' + \boldsymbol{r}'q) \\
=&\frac{t}{q}\left(\Delta^2 [\boldsymbol{m}]_t \cdot [\boldsymbol{m}']_t + \Delta([\boldsymbol{m}]_t \cdot \boldsymbol{v}' + [\boldsymbol{m}']_t \cdot \boldsymbol{v}) + \Delta q([\boldsymbol{m}]_t \cdot \boldsymbol{r}' + [\boldsymbol{m}']_t \cdot \boldsymbol{r})\right. \\
&\left. + \boldsymbol{v} \cdot \boldsymbol{v}' + q(\boldsymbol{v} \cdot \boldsymbol{r}' + \boldsymbol{v}' \cdot \boldsymbol{r}) + q^2 \boldsymbol{r} \cdot \boldsymbol{r}'\right) \\
=&\frac{q - |q|_t}{q}\left(\Delta\left([\boldsymbol{m} \cdot \boldsymbol{m}']_t + t\boldsymbol{r}_m\right) + [\boldsymbol{m}]_t \cdot \boldsymbol{v}' + [\boldsymbol{m}']_t \cdot \boldsymbol{v} + \boldsymbol{r}_v\right) + t(\boldsymbol{v} \cdot \boldsymbol{r}' + \boldsymbol{v}' \cdot \boldsymbol{r}) \\
&+ (q - |q|_t)([\boldsymbol{m}]_t \cdot \boldsymbol{r}' + [\boldsymbol{m}']_t \cdot \boldsymbol{r}) + \frac{t}{q}[\boldsymbol{v} \cdot \boldsymbol{v}']_\Delta + tq\boldsymbol{r} \cdot \boldsymbol{r}' \\
=&\Delta[\boldsymbol{m} \cdot \boldsymbol{m}']_t + \boldsymbol{v}_{\text{mult}} + q([\boldsymbol{m}]_t \cdot \boldsymbol{r}' + [\boldsymbol{m}']_t \cdot \boldsymbol{r} + \boldsymbol{r}_m + t\boldsymbol{r} \cdot \boldsymbol{r}'),
\end{aligned}$$

$$\begin{aligned}\text{with: } \boldsymbol{v}_{\text{mult}} =& ([\boldsymbol{m}]_t \cdot \boldsymbol{v}' + [\boldsymbol{m}']_t \cdot \boldsymbol{v} + \boldsymbol{r}_v)\left(1 - \frac{|q|_t}{q}\right) + t(\boldsymbol{v}\cdot\boldsymbol{r}' + \boldsymbol{v}'\cdot\boldsymbol{r}) \\ &+ \frac{t}{q}[\boldsymbol{v}\cdot\boldsymbol{v}']_\Delta - \frac{|q|_t}{q}\Delta[\boldsymbol{m}\cdot\boldsymbol{m}']_t - |q|_t\left([\boldsymbol{m}]_t\cdot\boldsymbol{r}' + [\boldsymbol{m}']_t\cdot\boldsymbol{r}\right) \\ &- \boldsymbol{r}_m|q|_t\left(2 - \frac{|q|_t}{q}\right)\end{aligned}$$

As shown in [7], we have: $\|\boldsymbol{r}_m\|_\infty \leqslant \frac{\delta_{\mathcal{R}} t}{2}$ and $\|\boldsymbol{r}_v\|_\infty \leqslant \frac{\delta_{\mathcal{R}}}{2}\min(\|\boldsymbol{v}\|_\infty, \|\boldsymbol{v}'\|_\infty)$. Therefore:

$$\begin{aligned}\|\boldsymbol{v}_{\text{mult}}\|_\infty \leqslant& \frac{t}{2}\delta_{\mathcal{R}}(\|\boldsymbol{v}'\|_\infty + \|\boldsymbol{v}\|_\infty) + \frac{\delta_{\mathcal{R}}}{2}\min(\|\boldsymbol{v}\|_\infty, \|\boldsymbol{v}'\|_\infty) + t\delta_{\mathcal{R}} r_\infty(\|\boldsymbol{v}'\|_\infty + \|\boldsymbol{v}\|_\infty) \\ &+ \frac{t}{q}\frac{\Delta}{2} + \frac{t^2}{2q}\Delta + t^2\delta_{\mathcal{R}} r_\infty + t^2\delta_{\mathcal{R}} \\ \leqslant& \delta_{\mathcal{R}} t\left(r_\infty + \frac{1}{2}\right)(\|\boldsymbol{v}\|_\infty + \|\boldsymbol{v}'\|_\infty) + \frac{\delta_{\mathcal{R}}}{2}\min(\|\boldsymbol{v}\|_\infty, \|\boldsymbol{v}'\|_\infty) \\ &+ \frac{1}{2} + t\left(\frac{1}{2} + t\delta_{\mathcal{R}}(1 + r_\infty)\right)\end{aligned}$$

However in the conditions herein considered, we need also to take into account the errors $\boldsymbol{e}_i$ coming from (14), thus:

$$\begin{aligned}&\hat{\boldsymbol{c}}_0 + \hat{\boldsymbol{c}}_1\cdot\boldsymbol{s} + \hat{\boldsymbol{c}}_2\cdot\boldsymbol{s}^2 \\ =& \frac{t}{q}\boldsymbol{c}_0\cdot\boldsymbol{c}_0' + \boldsymbol{e}_0 + \left(\frac{t}{q}(\boldsymbol{c}_0\cdot\boldsymbol{c}_1' + \boldsymbol{c}_1\cdot\boldsymbol{c}_0) + \boldsymbol{e}_1\right)\cdot\boldsymbol{s} + \left(\frac{t}{q}\boldsymbol{c}_1\cdot\boldsymbol{c}_1' + \boldsymbol{e}_2\right)\cdot\boldsymbol{s}^2 \\ =& \frac{t}{q}(\boldsymbol{c}_0 + \boldsymbol{c}_1\cdot\boldsymbol{s})\cdot(\boldsymbol{c}_0' + \boldsymbol{c}_1'\cdot\boldsymbol{s}) + \boldsymbol{e}_0 + \boldsymbol{e}_1\cdot\boldsymbol{s} + \boldsymbol{e}_2\cdot\boldsymbol{s}^2.\end{aligned}$$

Therefore, $\hat{\boldsymbol{v}} = \boldsymbol{v}_{\text{mult}} + \boldsymbol{e}_0 + \boldsymbol{e}_1\cdot\boldsymbol{s} + \boldsymbol{e}_2\cdot\boldsymbol{s}^2$, with $\|\boldsymbol{e}_0 + \boldsymbol{e}_1\cdot\boldsymbol{s} + \boldsymbol{e}_2\cdot\boldsymbol{s}^2\|_\infty \leqslant B_e\left(1 + \delta_{\mathcal{R}} B_{key} + \delta_{\mathcal{R}}^2 B_{key}^2\right)$. We obtain (16) by summing the last two bounds. □