Using Hierarchical Approach to Speed-up RNS Base Extensions in Homomorphic Encryption Context

Morgane VOLLMER and Karim BIGOU
Université de Bretagne Occidentale
Lab-STICC, UMR 6285
F-29200 Brest, France
firstname.lastname@univ-brest.fr

Arnaud TISSERAND
CNRS
Lab-STICC, UMR 6285
F-29806 Brest, France
firstname.lastname@cnrs.fr

***Abstract*—The numerous and huge operations involved in homomorphic encryption applications require fast arithmetic. RNS arithmetic is popular in their software implementations. In this context, we proposed a hierarchical approach for RNS base extension. It leads to 50–60 % reduction of both computation time and constant storage requirements for large homomorphic parameters in our experimental setup. When state-of-the-art parameters are not suitable for our approach, we propose to use equivalent parameters leading to similar reductions.**

***Index Terms*—residue number system; base extension; homomorphic encryption**

I. Introduction

In *homomorphic encryption* (HE), computations are directly performed over encrypted operands [1]. They can be delegated to cloud servers without compromising privacy of sensitive data. HE is an emerging solution in privacy-concerned applications such as medicine [2] and machine learning [3].

Among efficient HE schemes based on the ring learning with error problem [4], BFV [5], the Fan-Vercauteren variant of the Brakeski's scheme [6], is popular. It requires *numerous arithmetic operations* on huge polynomials with thousands of *large coefficients*. Typical coefficients are integers with *hundreds to thousands* bits.

Recent HE implementations use the *residue number system* (RNS) to speed up internal computations: OpenFHE [7], PALISADE [8], SEAL [9]. RNS relies on the Chinese remainder theorem (CRT) to split large computations into parallel smaller ones. In RNS-based HE implementations, the *base extension* (BE) is the key operation on coefficients [10].

A *hierarchical* BE algorithm has been proposed in [11] for elliptic curve cryptography (ECC) in hardware. This algorithm uses RNS bases with $k = r \times c$ moduli to reduce the BE complexity from $O(k^2)$ operations to only $O(r^2 \times c)$.

Below, we propose a similar hierarchical approach to speed-up BE in BFV context. Background is recalled in Sec. II. Our approach is presented in Sec. III. Sec. IV reports preliminary software implementation results and comparisons to state-of-the-art solutions. Sec. V discusses ongoing works.

II. Notations and State of the Art

RNS for homomorphic encryption has been introduced in [12] with an adaptation of BFV [5]. Our notations are:

- $|x|_m = x \bmod m$;
- $\mathbf{Q} = (q_1, \ldots, q_k)$ an RNS *base* with k prime *moduli*;
- $Q = \prod_{i=1}^{k} q_i$ the product of all moduli in the base;
- w the width of all moduli expressed in bits, in practice w fits into a machine word.

We represent an operand $x \in \mathbb{Z}_Q$ by its vector of *residues* $(x_{q_1}, \ldots, x_{q_k}) = (|x|_{q_1}, \ldots, |x|_{q_k})$ in the RNS base $\mathbf{Q}$. Thanks to the CRT, RNS addition, subtraction and multiplication are very efficient operations which can be performed in parallel over each residue. But RNS is non positional thus comparison, division and rounding are costly. They require BEs. A BE converts x represented in the RNS base $\mathbf{Q}$ into another base $\mathbf{P}$, with Q and P coprime. Similar notations apply for base $\mathbf{P}$, for instance $P = \prod_{i=1}^{k} p_i$.

In [12], the BFV scheme from [5] is adapted to RNS. The polynomials *coefficients*, which are $\mathbb{Z}_Q$ elements with $\log_2(Q) \in [90, 1600]$ bits, are *represented in RNS*. In [12], RNS divides the computation time by 4 to 20 depending on the application. To perform operations such as rounding, a *fast base extension* (FBE) algorithm is introduced. It computes:

$$\mathrm{FBE}(x, \mathbf{Q}, \mathbf{P}) = \left(\left| \sum_{i=1}^{k} \left| x_{q_i} \times \frac{q_i}{Q} \right|_{q_i} \times \frac{Q}{q_i} \right|_{p_j} \right)_{j \in [1,k]}$$

FBE is efficient because it removes one costly modular reduction by Q from the CRT formula. It only uses *integer* computations and costs $k^2 + k$ modular multiplications on w bits. But its result is approximated: FBE returns $x' = x + \alpha q$ with $\alpha \in [0, k-1]$ instead of x in base $\mathbf{P}$. In practice for the RNS implementation of BFV proposed in [12], the approximation factor α is managed thanks to some correction steps outside of FBE during other computations.

Paper [13] proposes to include a new correction step for α inside the BE algorithm using floating-point approximations. Comparisons of this BE solution to the previous one can be

found in [14], [15]. In practice, they lead to quite similar performances in HE. Today, both solutions are implemented in HE libraries [7], [8], [9].

A *hierarchical* base extension (HBE) approach has been introduced in [11] for ECC in FPGA implementations. The k moduli are seen as a matrix of r rows and c columns (i.e., $k = r \times c$). HBE consists in computing c intermediate CRTs over the residues, creating r *super-residues*. The BE is then performed over the r super-residues, leading to a reduction of the BE complexity. The correction step for α presented in [11] (derived from [16]) has been managed directly in the HBE algorithm but it requires pseudo-Mersenne moduli. Paper [11] only provides algorithms and results for $c = 2$.

III. Fast Hierarchical Base Extension

Below, we propose a new BE algorithm with a reduced computation cost by adapting FBE from [12] with the hierarchical organization idea from [11] for a HE software context. The following notations are introduced to support the decomposition of RNS bases into matrices of moduli (instead of vectors):

- $\mathbf{Q}$: an RNS base of $k = r \times c$ moduli with
$$\mathbf{Q} = \begin{pmatrix} q_{1,1} & \cdots & q_{1,c} \\ \vdots & \cdots & \vdots \\ q_{r,1} & \cdots & q_{r,c} \end{pmatrix};$$
- $Q_i = \prod_{j=1}^{c} q_{i,j}$: product of moduli in the i-th row;
- $Q = \prod_{i=1}^{r} Q_i = \prod_{i=1}^{r} \prod_{j=1}^{c} q_{i,j}$: product of all moduli;
- $\overline{Q_i} = Q/Q_i$: product of all moduli except the i-th row;
- $\overline{q_{i,j}} = Q_i/q_{i,j}$: product of all moduli in the i-th row expect the j-th one;
- $T_{q_{i,j}} = \left|\frac{q_{i,j}}{Q}\right|_{q_{i,j}}$: constant used in BEs;
- $\mathbf{P} = (p_{1,1}, \ldots, p_{r,c})$: another RNS base coprime with $\mathbf{Q}$.

Our algorithm, denoted *fast hierarchical BE* (FHBE), is presented in Alg. 1. It mostly has the same operations as in HBE from [11], except that the α approximation is not calculated using the same trick as the one proposed for FBE in [12]. Lines 1–3 are strictly identical between FBE and FHBE. Lines 4–7 compute a partial CRT on each row of the representation to obtain r super-residues $\widehat{X}_{Q_i}$ in the base $(Q_1, \ldots, Q_r)$. The width of Q_i is bounded by $cw + \lceil \log_2(c) \rceil$ bits. Finally lines 8–12 directly compute the CRT from the super-residues and reduce it modulo each element of base $\mathbf{P}$.

HBE algorithm has been validated in [11]. Using the same validation method, one can immediately show that FHBE leads to the same result than FBE.

FBE algorithm requires $k^2 + k$ elementary modular multiplications (EMMs) over w bits moduli and $k^2 + k$ stored pre-computations of w bits. EMM metric is used in [11] to evaluate the theoretical cost of RNS algorithms. FHBE requires $k = r \times c$ EMMs at line 3 and $r^2 \times c = \frac{k^2}{c}$ EMMs at line 12, leading to a total of $\frac{k^2}{c} + k$ EMMs. However, FHBE also requires some operations with larger operands than FBE:

- $w \times (c-1)w$ bits multiplications (without reduction) at line 7;
- modular reductions from $cw + \lceil \log_2(c) \rceil$ to w bits at line 11.

Algorithm 1: Proposed Fast Hierarchical BE (FHBE)

Input: X_Q: X in base $\mathbf{Q}$
Precomp.: $T_{q_{i,j}}$, $\overline{q_{i,j}}$ $\forall\ i \in [1,r]$ and $\forall\ j \in [1,c]$, $\left|\overline{Q_i}\right|_{p_{l,j}}$ $\forall\ i \in [1,r]$, $\forall\ l \in [1,r]$ and $\forall\ j \in [1,c]$
Output: X_P: $X + \alpha Q$ in base $\mathbf{P}$

1 **for** i **from** 1 **to** r **parallel do**
2 **for** j **from** 1 **to** c **parallel do**
3 $\widehat{x}_{q_{i,j}} \leftarrow \left|x_{q_{i,j}} \times T_{q_{i,j}}\right|_{q_{i,j}}$
4 **for** i **from** 1 **to** r **parallel do**
5 $\widehat{X}_{Q_i} \leftarrow 0$
6 **for** j **from** *1* **to** c **do**
7 $\widehat{X}_{Q_i} \leftarrow \widehat{X}_{Q_i} + \widehat{x}_{q_{i,j}} \times \overline{q_{i,j}}$ *(no reduction)*
8 **for** i **from** 1 **to** r **do**
9 **for** l **from** 1 **to** r **parallel do**
10 **for** j **from** 1 **to** c **parallel do**
11 $\widehat{x}_{p_{l,j,i}} \leftarrow \left|\widehat{X}_{Q_i}\right|_{p_{l,j}}$
12 $x_{p_{l,j}} \leftarrow \left|x_{p_{l,j}} + \widehat{x}_{p_{l,j,i}} \times \left|\overline{Q_i}\right|_{p_{l,j}}\right|_{p_{l,j}}$

Paper [11] does not analyze the size of the pre-computations storage required in HBE because it always fits into the FPGA memories for ECC. The pre-computations required for our FHBE (Alg. 1) are:

- $T_{q_{i,j}}$: k residues of w bits;
- $\overline{q_{i,j}}$: k integers of $(c-1)w$ bits;
- $\left|\overline{Q_i}\right|_{p_{l,j}}$: $r \times k$ residues of w bits.

Their total size is therefore $k(c+r)w = \left(\frac{k^2}{c} + kc\right) w$ bits. FHBE leads to a smaller pre-computations storage. For a fixed k, the minimum is reached when $c = \sqrt{k}$. In the ideal case $k = c^2$, FHBE divides by $\frac{\sqrt{k}}{2}$ the pre-computation storage compared to FBE.

Regarding the reduction of the computation cost, a high level comparison of the two algorithms is not straightforward. The iteration in the deepest nested loops of FBE just computes line 12 (with another w bits constant instead of $\left|\overline{Q_i}\right|_{p_{l,j}}$). The cost of these nested loops is k^2 operations on w bits values. In FHBE, there are only $r^2 \times c = \frac{k^2}{c}$ such iterations, but there is a new operation in the iteration at line 11. FHBE also requires a preliminary loop at lines 4–7, but it only contains $r \times c$ iterations. Most of new operations perform on larger data, line 7 computes values of size up to $cw + \lceil \log_2(c) \rceil$ bits. To conclude, the optimal computation cost depends on several implementation parameters.

One can observe that Alg. 1 requires k to have divisors. The next section evaluates the impact of k integer factorization on performances.

IV. Implementation and Comparisons

We implemented our FHBE algorithm, as well as the FBE algorithm from [12], in C/C++ language using the

GMP multiple-precision arithmetic library version 6.2.0, GCC compiler version 9.4.0 and Linux kernel 5.15 from Ubuntu distribution. We used the same optimization efforts in the codes and during the compilation process. All the performance and memory cost evaluations have been performed on an Intel Core i7-9850H processor at 2.60GHz. We will study the impact of other processor architectures in the future. We plan to distribute our codes as an open source library when polynomial and HE operations will be implemented, parallelized and optimized (see Sec. V).

We analyzed and compared two performance/cost aspects of our BE algorithm and the state-of-the-art one:

- the computation cost evaluated using the execution time on a single thread (as in [14]);
- the pre-computations storage requirements measured by carefully monitoring all values allocated in GMP.

Regarding the computation time evaluation, we performed numerous BEs on random independent operands (like the 2^{11} to 2^{15} coefficients of polynomials used in HE applications). The same operands were applied to both BE algorithms to compare them with the same memory mapping and similar cache behavior for the operands (but each BE algorithm requires different pre-computations).

We used the two largest $\mathbb{Z}_Q$ from [12] (780 and 1590 bits) and the corresponding k and w values. In the future, we will perform more extensive evaluations for other sizes.

In FHBE, k the number of moduli in the RNS bases must be divisible ($k = r \times c$). This leads to two cases:

- k from state-of-the-art values is divisible, then we use it for both FBE and FHBE;
- or k from state-of-the-art values is not divisible, then we use it for FBE but we choose a close k for FHBE and we adapt w to provide the same size $\log_2(Q)$ (we do not want to impact the security level).

Our evaluation results when k is divisible are reported in Tab. I for 3 pairs (k, w) from [12]. As c must divide k, some cells of the table are empty (denoted by a dash).

FHBE leads to 28 % to 58 % faster BEs except in the case $c = 2$. This confirms that the reduction of the computation cost from $O(k^2)$ to $O(r^2 \times c)$ offered by FHBE is effective in practice. When $c > 2$, a speed-up is always observed and the largest c leads to the fastest BE. This suggest that the reduction in the number of iterations for line 12 of Alg. 1 is more important than the penalty of dealing with larger operations and values at lines 4–7 and 11. On the contrary, when $c = 2$, this penalty is more important than the reductions of the number of EMMs and FHBE is slower than FBE.

TABLE I
COMPUTATION TIME AND PRE-COMPUTATION STORAGE SIZE COMPARISONS BETWEEN FBE [12] AND FHBE WITH VARIOUS BASE DECOMPOSITIONS FOR STATE OF THE ART $k \times w$ VALUES.

	$k \times w$	FBE	FHBE with c on line below						best
			2	3	4	5	6	13	gain
time [μs]	12×62	2.74	3.47	2.72	2.35	-	**1.96**	-	28%
	26×30	10.21	12.67	-	-	-	-	**4.26**	58%
	25×62	9.53	-	-	-	**6.61**	-	-	30%
size [kb]	12×62	9.53	5.86	**5.14**	5.15	-	5.92		46%
	26×30	20.32	**11.38**	-	-	-	-	11.6	43%
	25×62	39.65	-	-	-	**15.33**	-	-	61%

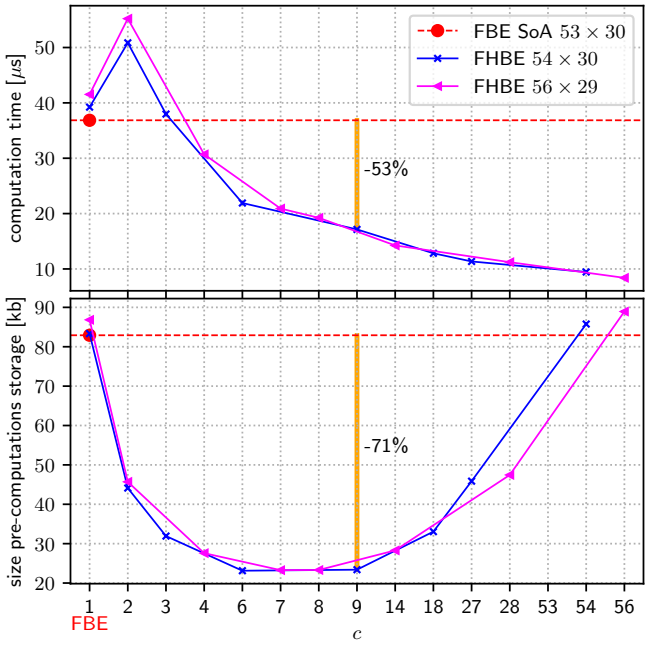

Fig. 1. Computation time and size of pre-computations storage comparisons of FBE [12] and FHBE with several $k \times w$ decompositions and c values.

Regarding the cost of the pre-computations storage in the bottom part of Tab. I, FHBE always leads to smaller memory requirements. The best storage reduction ranges from 46 to 61 %. We think this is a nice property for HE applications where data are huge. FHBE should help to reduce problems related to cache impact (but we need more experiments to analyze details). As one can guess from our analysis in Sec. III, the smaller memory usage is obtained for c close to $\sqrt{k}$. As an example, with $k = 12$ the smallest total of pre-computations are for $c = 3$ and $c = 4$.

In Fig. 1, we report results when k is not divisible. We used $k = 53$ and $w = 30$ from [12] for FBE. For FHBE, we used similar RNS base decompositions (leading to very close $\log_2(Q)$) to evaluate the impact of many choices for c:

- $k = 54 = 2 \times 3^3$ and $w = 30$;
- $k = 56 = 2^3 \times 7$ and $w = 29$;

The top curves in Fig. 1 are the computation times for both BE algorithms (with two (k, w) pairs for FHBE). Except for small c values, FHBE is faster than FBE. For the largest c, up to 80 % time reduction is achieved using FHBE instead of FBE. For small values of c, especially $c = 2$, FHBE is slower than FBE (as in Tab. I).

The bottom curves in Fig. 1 clearly show smaller pre-computations storage requirements for FHBE with up to 73 %

reduction when c is close to $\sqrt{k}$ (experimentally in our setup, for $k = 56$ the best c is 7 which is the close to $\sqrt{56} \approx 7.48$).

The orange lines highlight a good trade-off between both metrics where FHBE (with $k = 54$, $c = 9$ and $w = 30$) is 53 % faster and requires 71 % smaller pre-computations storage than FBE (with $k = 53$ and $w = 30$).

The computation time and pre-computations storage reductions are more important in Fig. 1 than the ones from Tab. I. One reason is that using k with many divisors, there are more RNS base decomposition choices. Another reason is that when k increases, FHBE asymptotic computation cost is the one of FBE divided by c, thus FHBE is faster for higher k.

To summarize, FHBE significantly reduces the computation cost and the pre-computations storage over FBE when $c > 2$.

V. Discussion

Even if FHBE leads to faster operations and smaller pre-computations for BE, this is a work in progress. We now have to adapt, implement and optimize solutions for polynomial arithmetic and HE operations (*e.g.*, addition, multiplication, relinearization) using FHBE. We also have to select good parameters for all computations layers in complete applications.

Our current implementation of FHBE is a single-thread one to demonstrate the reductions in computation time and pre-computations storage. We now also have to parallelize it on multi-core processors. But this is a complex task since parallelism is available, at least, inside each coefficient in $\mathbb{Z}_Q$ represented in RNS (over the residues) and inside each polynomial (over its independent coefficients).

RNS provides natural parallelism over the residues. For instance, [13] reports a 2.5× BE speed-up when the number of threads is k. FHBE iterations at lines 1, 2, 4, 9 and 10 are parallel loops over the rows and the columns of the RNS base, this allows many parallelization solutions. Furthermore, the smaller pre-computations in FHBE should help parallel optimizations with fewer cache misses over shared storage in multi-core processors.

HE ciphertexts are polynomials with large degrees (*e.g.*, 2^{11} to 2^{15}). As their coefficients are independent, multiple BEs can be performed in parallel. For instance in [13], this leads to 6× speed-up for homomorphic multiplications on a 32-core processor compared to single-thread one.

Finally, mixing parallelism at both levels, RNS and polynomials, simultaneously is also possible. Then, the exploration space is huge. We want to evaluate many solutions for operations algorithms, RNS parameters, compiler level optimizations and processor architectures. We are very excited to work on these different aspects.

VI. Conclusion

We proposed FHBE a hierarchical variant of the FBE algorithm for software RNS implementations of HE. FHBE reduces about 50–60 % both the computation cost and the pre-computations storage for most (k, w) pairs compared to state-of-the-art solutions.

In future works, we will implement BFV operations in RNS using FHBE. We will also evaluate various parallelization strategies (parallelism in RNS operations and/or in polynomial operations) on multi-core processors. Finally, we will investigate the impact of parameters such as c, k and w on the performances of homomorphic operations.

Acknowledgments

This work has been partially supported by a PhD grant from DGA and PEC/CREACH-LABS. The authors are very grateful to Prof. Laurent Nana for his help and feedback.

References

[1] C. Gentry, "Fully homomorphic encryption using ideal lattices," in *Proc. Symposium on Theory of Computing (STOC)*. ACM, May 2009, pp. 169–178.

[2] A. Wood, K. Najarian, and D. Kahrobaei, "Homomorphic encryption for machine learning in medicine and bioinformatics," *ACM Computing Surveys*, vol. 53, no. 4, pp. 70:1–35, Jul. 2021.

[3] F. Bourse, M. Minelli, M. Minihold, and P. Paillier, "Fast homomorphic evaluation of deep discretized neural networks," in *Proc. Annual Cryptology Conference (CRYPTO)*, ser. LNCS, vol. 10993. Springer, Aug. 2018, pp. 483–512.

[4] V. Lyubashevsky, C. Peikert, and O. Regev, "On ideal lattices and learning with errors over rings," in *Proc. Annual International Conference on Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, ser. LNCS, vol. 6110. Springer, 2010, pp. 1–23.

[5] J. Fan and F. Vercauteren, "Somewhat practical fully homomorphic encryption," 2012. [Online]. Available: https://eprint.iacr.org/2012/144

[6] Z. Brakerski, "Fully homomorphic encryption without modulus switching from classical GapSVP," in *Proc. Annual Cryptology Conference (CRYPTO)*. Springer, Aug. 2012, pp. 868–886.

[7] A. Al Badawi, J. Bates, F. Bergamaschi, D. B. Cousins, S. Erabelli, N. Genise, S. Halevi, H. Hunt, A. Kim, Y. Lee, Z. Liu, D. Micciancio, I. Quah, Y. Polyakov, S. R.V, K. Rohloff, J. Saylor, D. Suponitsky, M. Triplett, V. Vaikuntanathan, and V. Zucca, "OpenFHE: Open-source fully homomorphic encryption library," in *Proc. Workshop on Encrypted Computing & Applied Homomorphic Cryptography (WAHC)*, Nov. 2022, pp. 53–63.

[8] Y. Polyakov, K. Rohloff, G. W. Ryan, and D. Cousins, "PALISADE homomorphic encryption software library," 2021. [Online]. Available: https://palisade-crypto.org/

[9] "Microsoft SEAL," https://github.com/Microsoft/SEAL, Jan. 2023, microsoft Research, Redmond, WA.

[10] A. Al Badawi, B. Veeravalli, C. F. Mun, and K. M. M. Aung, "High-performance FV somewhat homomorphic encryption on GPUs: An implementation using CUDA," *Transactions on CHES*, no. 2, pp. 70–95, Jul. 2018.

[11] L. Djath, K. Bigou, and A. Tisserand, "Hierarchical approach in RNS base extension for asymmetric cryptography," in *Proc. International Symposium on Computer Arithmetic (ARITH)*. IEEE, Jun. 2019, pp. 46–53.

[12] J.-C. Bajard, J. Eynard, M. A. Hasan, and V. Zucca, "A full RNS variant of FV like somewhat homomorphic encryption schemes," in *Proc. International Conference on Selected Areas in Cryptography (SAC)*, ser. LNCS, vol. 10532. Springer, Aug. 2016, pp. 423–442.

[13] S. Halevi, Y. Polyakov, and V. Shoup, "An improved RNS variant of the BFV homomorphic encryption scheme," in *Proc. Topics in Cryptology – CT-RSA*. Springer, Mar. 2019, pp. 83–105.

[14] A. Al Badawi, Y. Polyakov, K. M. M. Aung, B. Veeravalli, and K. Rohloff, "Implementation and performance evaluation of RNS variants of the BFV homomorphic encryption scheme," *IEEE Transactions on Computers*, vol. 9, no. 2, pp. 941–956, Apr. 2021.

[15] J.-C. Bajard, J. Eynard, P. Martins, L. Sousa, and V. Zucca, "Note on the noise growth of the RNS variants of the BFV scheme," 2019. [Online]. Available: https://eprint.iacr.org/2019/1266

[16] S. Kawamura, M. Koike, F. Sano, and A. Shimbo, "Cox-Rower architecture for fast parallel Montgomery multiplication," in *Proc. Annual International Conference on Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, ser. LNCS, vol. 1807. Springer, May 2000, pp. 523–538.