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

Morgane Vollmer¹, Karim Bigou¹, Arnaud Tisserand²

¹ Université de Bretagne Occidentale / Lab-STICC, UMR CNRS 6285
² CNRS / Lab-STICC, UMR CNRS 6285

ARITH-2023, 4–6 September 2023, Portland, Oregon, USA
Contents

1 Context and State-of-the-Art
2 Proposed Fast Hierarchical Base Extension
3 Implementation, Results and Comparisons
4 Conclusion
Contents

1. Context and State-of-the-Art

2. Proposed Fast Hierarchical Base Extension

3. Implementation, Results and Comparisons

4. Conclusion
Homomorphic Encryption [Gen09]

Homomorphic encryption (HE) allows to perform computations over encrypted data without the knowledge of the secret key.

Solution for privacy-concerned applications: medical/machine learning

\[
x + y = x + y \\
x \times y = x \times y
\]

Huge cost: requires numerous arithmetic operations on very large intermediate data.

⇒ Arithmetic is key for HE since it requires operations on very large data.
The FV scheme handles huge polynomials:

- degree $n \in [2^{11}, 2^{17}]$
- with large coefficients in $\mathbb{Z}/q\mathbb{Z}$ with $\log_2(q) \in [55, 2090]$ bits

\[ c = c_0 + c_1 x + c_2 x^2 + \ldots + c_n x^n \]
Chosen HE solution: FV scheme \([\text{FV12}]\)

The FV scheme handles huge polynomials:

- degree \(n \in [2^{11}, 2^{17}]\)
- with large coefficients in \(\mathbb{Z}/q\mathbb{Z}\) with \(\log_2(q) \in [55, 2090]\) bits

\[ c = c_0 + c_1 x + c_2 x^2 + \ldots + c_n x^n \]

\textbf{RNS} is used for the \textbf{arithmetic on coefficients}
Residue Number System (RNS) [Gar59] [SV55]

RNS is a non-positional representation system based on the Chinese Remainder Theorem (CRT)

RNS Base: \( \mathcal{A} = (a_1, \ldots, a_k) \) tuple of coprime integers of size \( w \) bits called moduli, \( \prod_{i=1}^{k} a_i = A \)

Representation of an integer \( X \) in the base \( \mathcal{A} \):
\( X_{\mathcal{A}} = (|X|_{a_1}, \ldots, |X|_{a_k}) = (x_{a_1}, \ldots, x_{a_k}) \) where \( |X|_{a_i} = X \mod a_i \)
Pros and cons

Pros: Operations $+, -, \times$ computed independently over the residues $\sim$ only $k$ small operations over the residues $\Rightarrow$ fast and parallel

Cons: Division, rounding and modular reduction are difficult to compute in RNS $\Rightarrow$ intermediate operation: Base Extension
A base extension (BE) converts $X$ in base $A$ into $X$ in base $B$. BE costs $\sim k^2$ operations.

Most popular technique: computing the CRT in each moduli of the second base.

**CRT formula:**

$$|X|_{b_j} = \left| \sum_{i=1}^{k} \frac{x_{a_i} \cdot a_i}{A} \cdot \frac{A}{a_i} \right|_{b_j} = \left| \left( \sum_{i=1}^{k} \frac{x_{a_i} \cdot a_i}{A} \cdot \frac{A}{a_i} \right) - hA \right|_{b_j}$$

**Popular BE algorithms:**
- Kawamura and al. [KKSS00] (approximate computation of $h$)
- Shenoy and Kumaresan [SK89] (requires specific conditions)
FV in RNS

Adaptations of FV in RNS:
- first adaptation of FV fully in RNS was proposed in [BEHZ16]
  - speed-up from 5 to 20 for the decryption
  - speed-up from 2 to 4 for the homomorphic multiplication
- variants in [HPS19] [ABVMA18] [ABPA+21] [KPZ21]

Difficult operations:
- roundings
- modular reduction

In the state-of-the-art, half of the homomorphic multiplication is spent doing base extension
Fast Base Extension (FBE)

[BEHZ16] proposed to compute these BE with a Fast Base Extension algorithm (FBE)

\[
\text{FBE}(X, A, B) = \left| \left( \sum_{i=1}^{k} |x_{a_i} \cdot \frac{a_i}{A} | a_i \cdot \frac{A}{a_i} \right) \right|_b
\]

- Efficient because it removes the costly modular reduction by \( A \) in the CRT formula but gives an approximate result.
- [BEHZ16] manages the approximation with some correction steps outside the BE.
Hierarchical Base Extension (HBE)

[DBT19] proposed the hierarchical base extension: a hierarchical version of Kawamura Base Extension to speed-up ECC on FPGA.

New notation: \( A = (a_1, \ldots, a_k) \longrightarrow A = \begin{pmatrix} a_{1,1} & \ldots & a_{1,c} \\ \vdots & \ddots & \vdots \\ a_{r,1} & \ldots & a_{r,c} \end{pmatrix} \)

with \( k = r \times c \)

Idea:

- split the base of \( k \) moduli in \( r \) sub-bases of \( c \) moduli
- compute the CRT in each sub-base to create super-residues
- compute the CRT of the super-residues of base \( A \) in base \( B \)

In [DBT19], \( c = 2 \)
Kawamura Base Extension:

\[ x_{a_1} \quad x_{a_2} \quad x_{a_3} \quad \vdots \quad x_{a_k} \quad x_{b_1} \quad x_{b_2} \quad \ldots \quad x_{b_k} \]

\[ k^2 \]

**HBE:** \( k = r \times c \) with \( c = 2 \)
Contents

1. Context and State-of-the-Art

2. Proposed Fast Hierarchical Base Extension

3. Implementation, Results and Comparisons

4. Conclusion
Proposed Fast Hierarchical Base Extension (FHBE)

We adapt the hierarchical approach to FBE for software implementation of FV

We study the impact of the value of $c$ on the computation cost using prime moduli
Algorithm 1: FBE [BEHZ16]

Input: $X_A$
Precomp.: $T_{a_i,j}$, $\forall i \in [1, r]$ and $\forall j \in [1, c]$, $A_{b_{l,j}}^{a_i}$, $\forall i \in [1, r], \forall l \in [1, r]$ and $\forall j \in [1, c]$
Output: $X + \alpha A$ in base $B$

for $i$ from 1 to $r$ parallel do
  for $j$ from 1 to $c$ parallel do
    $\hat{x}_{a_{i,j}} \leftarrow |x_{a_{i,j}}| \times T_{a_{i,j}}|_{a_{i,j}}$
  for $i$ from 1 to $k$ do
    for $l$ from 1 to $r$ parallel do
      for $j$ from 1 to $c$ parallel do
        $x_{b_{l,j}} \leftarrow |x_{b_{l,j}}| \times A_{b_{l,j}}^{a_i}$
        $x_{b_{l,j}} \leftarrow x_{b_{l,j}} + \hat{x}_{a_{i,j}} \times A_{b_{l,j}}^{a_i}$

Algorithm 2: Proposed Fast Hierarchical BE (FHBE)

Input: $X_A$
Precomp.: $T_{a_{i,j}}$, $A_{b_{l,j}}^{a_i}$, $\forall i \in [1, r]$ and $\forall j \in [1, c]$
Output: $X + \alpha A$ in base $B$

for $i$ from 1 to $r$ parallel do
  for $j$ from 1 to $c$ parallel do
    $\hat{x}_{a_{i,j}} \leftarrow |x_{a_{i,j}}| \times T_{a_{i,j}}|_{a_{i,j}}$
  for $i$ from 1 to $r$ parallel do
    $\hat{X}_A_i \leftarrow 0$
    for $j$ from 1 to $c$ do
      $\hat{X}_A_i \leftarrow \hat{X}_A_i + \hat{x}_{a_{i,j}} \times A_{b_{l,j}}^{a_i}$ (no reduction)
  for $i$ from 1 to $r$ do
    for $l$ from 1 to $r$ parallel do
      for $j$ from 1 to $c$ parallel do
        $\hat{x}_{b_{l,j,i}} \leftarrow \hat{X}_A_i|_{b_{l,j}}$
        $x_{b_{l,j}} \leftarrow |x_{b_{l,j}}| \times A_{b_{l,j}}^{a_i}$
        $x_{b_{l,j}} \leftarrow x_{b_{l,j}} + \hat{x}_{b_{l,j,i}} \times A_{b_{l,j}}^{a_i}$
Cost Comparisons

FBE:

Number of operations:
- \( k^2 + k \) modular multiplications on \( w \) bits

Stored precomputations:
- \( k^2 + k \) stored precomputations of size \( w \) bits

FHBE:

Number of operations:
- \( \frac{k^2}{c} + \frac{k}{c} \) modular multiplications on \( w \) bits
- \( w \times (c - 1)w \) bits multiplications without reduction
- modular reductions from \( cw + \lceil \log_2(c) \rceil \) to \( w \) bits

Stored precomputations:
- \( \frac{k^2}{c} + k \) stored precomputations of size \( w \) bits
Contents

1. Context and State-of-the-Art
2. Proposed Fast Hierarchical Base Extension
3. Implementation, Results and Comparisons
4. Conclusion
Implementation

Implementation of our FHBE and the FBE algorithms:

- in C language using the GMP multiple-precision arithmetic library version 6.2.0
- GCC compiler version 9.4.0
- Linux Kernel 5.15 from Ubuntu distribution

Performance and memory cost evaluations performed on an Intel Core i7-9850H processor at 2.60GHz

We analyzed:

- the computation cost using the execution time on a single thread
- the pre-computation storage requirements
Evaluation Cases

To use FHBE, \( k \) must be divisible \( (k = r \times c) \)

This leads to two cases:

- \( k \) from [BEHZ16] is **divisible**: we use \( k \) for both FBE and FHBE
- \( k \) from [BEHZ16] is **prime**
  - we use \( k \) for FBE
  - we choose a close \( k \) with many divisors for FHBE (with the same security in number of bits)
Results when $k$ is divisible

$k$: number of moduli  
$w$: size of the moduli in bits  
$log_2(q) = k \times w$

The parameters $(k, w)$ come from [BEHZ16]

<table>
<thead>
<tr>
<th>$k \times w$</th>
<th>FBE</th>
<th>FHBE with $c$ on line below</th>
<th>best gain</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>time [µs]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12 × 62</td>
<td>2.74</td>
<td>3.47</td>
<td>2.72</td>
</tr>
<tr>
<td>26 × 30</td>
<td>10.21</td>
<td>12.67</td>
<td>-</td>
</tr>
<tr>
<td>25 × 62</td>
<td>9.53</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>size [kb]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12 × 62</td>
<td>9.53</td>
<td>5.86</td>
<td>5.14</td>
</tr>
<tr>
<td>26 × 30</td>
<td>20.32</td>
<td>11.38</td>
<td>-</td>
</tr>
<tr>
<td>25 × 62</td>
<td>39.65</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
Results when $k$ is prime

In [BEHZ16], $(k, w) = (53, 30)$ is proposed.

53 is prime, then we use:
- $(54, 30)$ with $54 = 2 \times 3^3$
- $(56, 29)$ with $56 = 2^3 \times 7$
## Contents

1. Context and State-of-the-Art
2. Proposed Fast Hierarchical Base Extension
3. Implementation, Results and Comparisons
4. Conclusion
Conclusion

FHBE, a hierarchical variant of the FBE algorithm has been proposed and implemented for software RNS implementations of the FV scheme.

It reduces:

- up to 58% the computation cost
- up to 71% the storage requirement

Future prospect:

- optimized multi-core implementation
- complete homomorphic library
Thank you for your attention
[ABPA⁺21] Ahmad Al Badawi, Yuriy Polyakov, Khin Mi Mi Aung, Bharadwaj Veeravalli et Kurt Rohloff.
Implementation and performance evaluation of RNS variants of the BFV homomorphic encryption scheme.

[ABVMA18] Ahmad Al Badawi, Bharadwaj Veeravalli, Chan Fook Mun et Khin Mi Mi Aung.
High-performance FV somewhat homomorphic encryption on GPUs: An implementation using CUDA.

A full RNS variant of FV like somewhat homomorphic encryption schemes.


Operátorové obvody (operator circuits in czech).
One popular HE solution: FV scheme [FV12]

FV: HE scheme based on the ring learning with error problem [LPR10]

Plain data: 1 polynomial of degree $n \in [2^{11}, 2^{17}]$ with coefficients in $\mathbb{Z}/t\mathbb{Z}$ with $t \geq 2$

Cipher data: 2 polynomials of degree $n \in [2^{11}, 2^{17}]$ with coefficients in $\mathbb{Z}/q\mathbb{Z}$ with $\log_2(q) \in [55, 2090]$ and $t < q$

For: $256 \, \text{B} < \text{message} < 16 \, \text{kB} \Rightarrow 28 \, \text{kB} < \text{cipher} < 68 \, \text{MB}$
One popular HE solution: FV scheme [FV12]

FV: HE scheme based on the ring learning with error problem [LPR10]

Plain data: 1 polynomial of degree \( n \in [2^{11}, 2^{17}] \) with coefficients in \( \mathbb{Z}/t\mathbb{Z} \) with \( t \geq 2 \)

Cipher data: 2 polynomials of degree \( n \in [2^{11}, 2^{17}] \) with coefficients in \( \mathbb{Z}/q\mathbb{Z} \) with \( \log_2(q) \in [55, 2090] \) and \( t < q \)

For: 256 B < message < 16 kB \( \Rightarrow \) 28 kB < cipher < 68 MB
Fast Base Extension (FBE)

$$
\text{FBE}(X, A, B) = \left| \sum_{i=1}^{k} \left| x_{a_i} \times \frac{a_i}{A} \times A \right| \right|_{b_j}
$$

- Efficient because it removes the costly modular reduction by $A$ in the CRT formula but gives an approximate result.
- [BEHZ16] manages the approximation with some correction steps outside the BE.

Cost of the BE for an homomorphic multiplication in FV RNS
(implementation results from [HPS19])