CKKS Key Switching and its Implementation in Liberate-FHE

Eric Crockett

May 7, 2025

1 Notation

We represent $x \in \mathbb{Z}_q$ by its *canonical representative* $[x]_q \in [\lceil -q/2 \rceil, \lfloor q/2 \rfloor) \subset \mathbb{Z}$. We also use the *standard representative* $|x|_q \in [0, q) \subset \mathbb{Z}$.

Most homomorphic encryption implementation utilize the Chinese Remainder Theorem (CRT), also called the Residue Number System (RNS) to avoid confusion with other uses of the CRT in lattice cryptography. Let $q_0, \ldots, q_{k-1}$ be co-prime moduli, and let $Q_j = \prod_{i=0}^{j} q_i$. Define $Q := Q_{k-1}$ and $Q_{-1} := 1$. We denote $Q_i^* = Q/q_i \in \mathbb{Z}$ and $\tilde{Q}_i = (Q_i^*)^{-1} \in \mathbb{Z}_{q_i}$. In CRT/RNS, we identify $x \bmod Q$ with $(x_0, \ldots, x_{k-1})$ where $x_i \in \mathbb{Z}_{q_i}$. We can reconstruct $x = |x|_Q$ as $\sum_{i=0}^{k-1} |x_i|_{q_i} \cdot \tilde{Q}_i \cdot Q_i^* \bmod Q$. We call $\{q_0, q_1, \ldots, q_{k-1}\}$ the *RNS basis* of x. By "an element x in basis $\mathcal{B}$", we mean x in RNS form with respect to the moduli contained in $\mathcal{B}$. By the Chinese Remainder Theorem (CRT), there is precisely one integer x such that x in basis $\mathcal{B}$ is $(x_0, x_1, \ldots, x_{k-1})$.

2 Basis Conversion

One of the fundamental operations required for key-switching is basis conversion. Let $\mathcal{B} = \{q_i\}_{0 \leq i < k}$ be a basis, and $x \in \mathbb{Z}_Q$ be represented in basis $\mathcal{B}$. By "basis conversion", we mean that we wish to find $[x]_Q$ in basis $\mathcal{C}$ (which we can assume is disjoint from $\mathcal{B}$). We denote this as $\mathrm{Conv}_{\mathcal{B} \to \mathcal{C}}(\cdot)$. One simple way to do this is to use CRT reconstruction to compute $[x]_Q$ over the integers, and then reduce it mod p_i for each $p_i \in \mathcal{C}$. However computing the CRT reconstruction over the integers may require BigInteger arithmetic, so we seek to convert x in basis $\mathcal{B}$ to $[x]_Q$ in basis $\mathcal{C}$ *directly*, without doing a full CRT reconstruction of x. Note that we can obtain basis *extension* by concatenating the CRT components of disjoint bases $\mathcal{B}$ (i.e., the input) and $\mathcal{C}$ (i.e., the output); we generally use these terms interchangeably.

2.1 Garner's Algorithm

Liberate-FHE uses a tweak of Garner's algorithm [GCL92, Section 5.6] for basis conversion, which is based on the mixed-radix CRT reconstruction algorithm:

$$x = \sum_{i=0}^{k-1} [c_i]_{q_i} \cdot Q_{i-1} \in \mathbb{Z}.$$

where

$$c_i = \left(x_i - \sum_{j=0}^{i-1} [c_j]_{q_j} \cdot [Q_{j-1}]_{q_i} \right) \cdot Q_{i-1}^{-1} \in \mathbb{Z}_{q_i}.$$

Note that the c_i can be computed without the use of BigInteger arithmetic since all individual terms are reduced mod q_i, and all of the arithmetic is in $\mathbb{Z}_{q_i}$.

For the purposes of basis extension, we are interested in computing $[x]_p$ for some $p \in \mathcal{C}$. Since the sum for x is over the integers,

$$\begin{aligned} [x]_p &= \left[\sum_{i=0}^{k-1} [c_i]_{q_i} \cdot Q_{i-1} \right]_p \\ &\equiv \sum_{i=0}^{k-1} [c_i]_{q_i} \cdot [Q_{i-1}]_p \bmod p, \end{aligned}$$

where $[Q_{i-1}]_p$ can be pre-computed. This gives us a way to compute $[x]_p$ without using BigInteger arithmetic since each term is mod a small prime, the product is mod p, and the sum is mod p.

This algorithm has two steps:

1. Compute the c_i
2. Compute $[x]_p$ for each $p \in \mathcal{C}$

The first step is inherently quadratic in $|\mathcal{B}|$. The second step is linear for each $p \in \mathcal{C}$, giving an overall cost of $\mathcal{O}(|\mathcal{B}|^2 + |\mathcal{B}| \cdot |\mathcal{C}|)$.

The algorithm is described over the integers, but it trivially extends component-wise to ring elements. Also note that this algorithm is described using the canonical representative, but it works equally well with standard representatives.

2.1.1 Implementation

The following refers to the DeSilo C++ code provided to Cornami. Liberate builds the c_i one term of the sum at a time. Specifically, `pre_extend` *should* maintain the following invariants at the start of iteration i (named `index` in the code):

- pre_extended_j holds $[c_j]_{q_j}$ for $j \leq i$
- pre_extended_j holds $\sum_{k=0}^{i} [c_k]_{q_j} \cdot [Q_{k-1}]_{q_j}$ for $j > i$

Then in iteration i, `pre_extend` computes

$$\begin{aligned}
\text{pre_extended}_{i+1} &= \left[(\text{partition}_{i+1} - \text{pre_extended}_{i+1}) \cdot Q_i^{-1}\right]_{q_i} \\
&= \left[(\text{partition}_{i+1} - \sum_{k=0}^{i} [c_k]_{q_{i+1}} \cdot [Q_{k-1}]_{q_{i+1}}) \cdot Q_i^{-1}\right]_{q_i} \\
&= [c_i]_{q_i}
\end{aligned}$$

This maintains the invariant that state_{i+1} holds c_{i+1}. The inner `for` loop maintains the second invariant. However, there is a problem: `pre_extend` sets pre_extended_i to the output of `mont_enter`, which outputs a value in $[0, 2q_i)$. It should output a value in $[0, q_i]$, since that is the range of $[c_i]_{q_i}$.

`extend` uses the $[c_i]_{q_i}$ to compute $x \bmod p_j$ for each $p_j \in \mathcal{C}$.

References

[GCL92] K.O. Geddes, S.R. Czapor, and G. Labahn. *Algorithms for Computer Algebra*. Springer US, 1992. ISBN: 9780792392590. URL: https://books.google.com/books?id=B9tC7DOX_oUC.