Composite Scaling in CKKS

Eric Crockett

March 25, 2025

1 Introduction

This article explains what *composite scaling* is, how it works, and how it fits in with development at Cornami.

The original CKKS paper [Che+16] used a single massive ciphertext modulus Q. This is inefficient because it requires big-integer arithmetic. A much more efficient version called RNS-CKKS [Che+18] was created to solve this problem. In short, it breaks the ciphertext modulus into a product of primes where each prime (hopefully) fits into a native machine word. The Chinese Remainder Theorem defines a mapping between $x \in \mathbb{Z}_Q$ and $(x \in Z_{q_1}, x \in Z_{q_2}, ...)$. With this representation, we can induce arithmetic $\mod Q$ by doing native-word operations on each of the components. In RNS-CKKS, we require $\log q_i \approx \log \Delta$ where Δ is the scaling factor parameter for CKKS.

In practice, we typically pick $\log \Delta \approx \log q_i \approx 64$, as this results in relatively manageable noise growth and fast arithmetic on CPUs. If we chose smaller primes (say 32 bits), we would either have to use a dramatically smaller scaling factor (so small that we might quickly overpower our message with noise) or drop multiple moduli after each multiplication. The latter option is essentially the idea of "composite scaling": we break each "level" of the tower into a few smaller pieces. So instead of using a single 64-bit prime per level, we could use 2 32-bit primes or 3 21-bit primes. This can enable more efficient arithmetic on hardware where small-word arithmetic is (substantially) faster than large-word arithmetic.

2 Using Composite Scaling in Practice

In the real world, a client uses a FHE client to select parameters and encrypt data. They send the encrypted data (and associated evaluation keys) to a server, which uses a compatible library to parse the serialized ciphertexts and apply homomorphic operations, and sends the (encrypted) result back to the client for decryption. The client doesn't necessarily know what hardware will be used for evaluation on the server, so it doesn't have any particular reason to pick (say) 32-bit moduli over 64-bit moduli.

An easy solution is to require coordination between the client and the server. The client library either needs to be hard-coded to use composite scaling (with hard-coded composite scaling parameters), or must be configurable. We are also considering another option which would allow the client to encrypt with arbitrary parameters, and then the server could switch the modulus to use composite scaling after the fact. It's not entirely clear how and if this would work.

3 Composite Scaling Details

One of the major takeaways of [Agr+23] is that there is basically no difference in terms of precision loss when using composite scaling vs regular RNS-CKKS.

Broadly, for some operations

4 Hardware Modular Arithmetic

A *completely orthogonal* computational tool used in many FHE libraries is a mechanism for computing modular reductions for a machine-word modulus.

First, we note how the hardware $x \bmod q$ operation works: it computes $x - \lfloor x/q \rfloor \cdot q$ The division in particular can be very expensive on any hardware, and it's not possible in a fractal core. As a result, we explore better options below.

4.1 Montgomery Arithmetic

In Montgomery arithmetic, we choose $R = 2^k$ which is coprime to q, and represent a value $a \bmod q$ as $aR \bmod q$. Clearly we can add numbers directly in Montgomery form. To multiply a and b in Montgomery form, we compute $aR \cdot bR = abR^2$, and then apply Montgomery reduction, an algorithm which simultaneously divides by R and mods by q (almost). The output is actually $abR \bmod 2q$. This explains why $2q$ pops up a lot in the guts of FHE implementations; it's not relevant for the FHE math.

References

[Agr+23] Rashmi Agrawal et al. *High-precision RNS-CKKS on fixed but smaller word-size architectures: theory and application.* Cryptology ePrint Archive, Paper 2023/1462. 2023. URL: https://eprint.iacr.org/2023/1462.

[Che+16] Jung Hee Cheon et al. *Homomorphic Encryption for Arithmetic of Approximate Numbers*. Cryptology ePrint Archive, Paper 2016/421. 2016. URL: `https://eprint.iacr.org/2016/421`.

[Che+18] Jung Hee Cheon et al. *A Full RNS Variant of Approximate Homomorphic Encryption*. Cryptology ePrint Archive, Paper 2018/931. 2018. URL: `https://eprint.iacr.org/2018/931`.