NTRU-based FHE for Larger Key and Message Space

Robin Jadoul[1], Axel Mertens[1], Jeongeun Park[2⋆], and Hilder V. L. Pereira[3]

[1] COSIC, KU Leuven, Leuven, Belgium,
firstname.lastname@esat.kuleuven.be
[2] Norwegian University of Science and Technology (NTNU), Trondheim, Norway
jeongeun.park@ntnu.no
[3] Universidade Estadual de Campinas, Campinas, Brazil
hilder@unicamp.br

Abstract. The NTRU problem has proven a useful building block for efficient bootstrapping in Fully Homomorphic Encryption (FHE) schemes, and different such schemes have been proposed. FINAL (ASIACRYPT 2022) first constructed FHE using homomorphic multiplexer (CMux) gates for the blind rotation operation. Later, XZD+23 (CRYPTO 2023) gave an asymptotic optimization by changing the ciphertext format to enable ring automorphism evaluations. In this work, we examine an adaptation to FINAL to evaluate CMux gates of higher arity and the resulting tradeoff to running times and bootstrapping key sizes. In this setting, we can compare the time and space efficiency of both bootstrapping protocols with larger key space against each other and the state of the art.

1 Introduction

All current constructions for fully homomorphic encryption (FHE) schemes rely on noisy encryptions for which the noise accumulates and grows as homomorphic operations are applied. When the noise grows too big, however, correct decrypted value is no longer guaranteed. Therefore, a form of homomorphic decryption that resets the noise level back to a point where more operations can be applied is necessary, the so called *bootstrapping* operation, which is the most computationally expensive part of the scheme. Since the first introduction of FHE, there have been continous attempts [DM15,CGGI16,CH18,HS21,GIKV23] to make bootstrapping algorithms more practical and overcome this bottleneck. Initially, FHE schemes had large message spaces that could pack many messages into a single ciphertext, offering good amortized costs, since each homomorphic operation acts in parallel on all packed messages. However, the bootstrapping for these is very expensive in terms of both memory and time. One well-known scheme in this family, BGV [BGV12], performs bootstrapping with a typical running time of a few minutes on a common commercial personal computer, even for modern and optimized implementations [HS15].

A very different approach was taken by [AP14], which proposed an efficient bootstrapping for ciphertexts encrypting a single bit instead of many larger messages. That method was improved in [DM15] and finally implemented, yielding, for the first time, a bootstrapping algorithm that runs in less than one second on a common laptop. [CGGI16] improved futher upon this method, with the introduction of the TFHE scheme, for which bootstrapping runs in merely a few miliseconds.

After a hiatus of a few years, another scheme, called FINAL [BIP+22], improved upon TFHE and achieved single-bit bootstrapping that runs in about 70% of TFHE's running time, using bootstrapping keys of only about half the size. The core of this improvement lies in the introduction of NTRU-based ciphertexts for the bootstrapping algorithm. As a concurrent work, Kluczniak [Klu22]

⋆ This work was done when the author was at COSIC, KU Leuven.

introduced similar scheme, which was broken due to their choice of ciphertext space [Joy22] called cyclic NTRU ring. With the negacyclic version, still their construction has survived against existing attacks, their ciphertext form has additional larger noise than the ciphertext of FINAL, which is less efficient than FINAL in terms of noise management.

Using a slightly different ciphertext form, still based on the NTRU problem, Xiang et al. [XZD+23] introduced a new scheme, which we call by XZD+23 from now on, that allows for the usage of ring automorphisms in the bootstrapping phase, leading to asymptotically lower memory usage and bigger key sizes that could allow for smaller base ciphertexts. Their improvement to the running time when compared to TFHE is, however, overall not as pronounced as FINAL's and while the memory usage is asymptotically better, for most practical parameter sets, it can be seen to be higher. As an alternative to enable larger key sizes and smaller lattice dimensions, Joye and Paillier [JP22] suggested a version of the CMux gate with higher arity for the TFHE scheme. As FINAL's bootstrapping makes use of the same CMux construction, it could be amenable to a similar change, but no descriptions of this change have been given yet, nor has the comparison between both methods to increase the key space been made for NTRU-based FHE schemes.

Although those schemes achieve the fastest bootstrapping known to date, using them to evaluate arbitrary circuits can end up being very expensive, because they only encrypt bits. Even simple operations, such as integer addition, have to be represented by binary circuits and each gate of the circuit requires one bootstrapping. Thus, to overcome this limitation, TFHE was extended to support messages in $\mathbb{Z}_{2^k}$, that is, k-bit messages. Simultaneously, it gained the ability to perform *programmable bootstrapping* (PBS) (also sometimes referred to as *functional bootstrapping*). In this paradigm, every bootstrapping operation can not only perform a noise reduction; it can also apply an arbitrary function f, represented as a lookup table, to the underlying message of a ciphertext, at no additional cost [CJP21].

Our contributions

In this work, we study the n-ary CMUX from [JP22] and how it can be employed in FINAL. We also adapt other techniques and improvements from the literature to FINAL. For example, we notice that by carefully changing the definition of scalar NGS ciphertexts from FINAL, it is possible to use an approximate gadget decomposition (introduced first in TFHE) without increasing the final noise in a noticable way, thereby reducing the cost of the bootstrappring almost for free. We apply the same optimizations to the XZD+23 scheme, in order to compare the two main NTRU-based FHEs on equal footing.

While the bigger potential key size in XZD+23 brings a slight reduction in its memory usage and running time, we observe that it will theoretically still need approximately double the amount of significant operations (as measured by counting the number of required FFTs and FFT-domain multiplications). We also find that, depending on the employed FFT-library and the subsequent timings for an FFT-operation and a pointwise multiplication, FINAL comes out on top in efficiency-comparison with XZD+23. We further give theoretical bounds for the key space that result in higher efficiency than the ternary key setting.

We note that NTRU based PBS for binary LWE secret key was briefly introduced in [Klu22], however, as we mentioned above, their ciphertext has larger noise than FINAL, therefore, FINAL can naturally achieve the same functionality without increasing parameters. We stess that we examine PBS of FINAL, further, with larger secret key in this paper.

Roadmap

After the preliminaries in Section 2, we discuss our improvements and PBS for FINAL in Section 3. We also give evidence that bootstrapping by n-ary CMUX is not a guaranteed improvement for FINAL, and needs to be carefully evaluated per implementation or even supporting hardware. Then, in Section 4, we apply similar improvements and PBS to the scheme from [XZD+23]. Finally, in Section 5, the two schemes are compared.

2 Preliminaries

2.1 Notation and mathematical background

Vectors, polynomials, and norms We use lower-case bold letters for vectors and upper-case bold letters for matrices, e.g. $\mathbf{a}$ and $\mathbf{A}$. For any vector $\mathbf{u}$, $\|\mathbf{u}\|$ denotes the infinity norm. Throughout the paper, N is always a power of two and $\mathcal{R} := \mathbb{Z}[X]/\langle X^N + 1\rangle$ is the $(2N)$-th cyclotomic ring. Any element f of $\mathcal{R}$ can be always represented by a unique polynomial of degree smaller than N, hence, writing $f = \sum_{i=0}^{N-1} f_i \cdot X^i$ is unambiguous and we can define the coefficient vector of f as $\boldsymbol{\phi}(f) := (f_0, \ldots, f_{N-1}) \in \mathbb{Z}^N$. We define the infinity norm of f as $\|f\| := \|\boldsymbol{\phi}(f)\|$. For any $Q \in \mathbb{Z}^+$, let $\mathcal{R}_Q := \mathcal{R}/Q\mathcal{R} = \mathbb{Z}_Q[X]/\langle X^N + 1\rangle$.

Finally, we define $\mathbb{M} := \{\pm m \cdot X^k \mid m \in \mathbb{Z}_p, k \in \mathbb{N}\}$, which will be used as the plaintext space of the vector ciphertexts defined in Section 2.5. Here, $\mathbb{Z}_p$ is the overall plaintext space, which for the original FINAL scheme is $\mathbb{Z}_2$.

Discrete Gaussian distribution. We first describe the discrete Gaussian distribution where our secret elements are sampled from. Typically, a discrete Gaussian distribution is defined as a distribution over $\mathbb{Z}$, where every element in $\mathbb{Z}$ is sampled with probability proportional to its probability mass function value under a Gaussian distribution over $\mathbb{R}$. Let the Gaussian function $\rho_{\sigma,c}$ be defined as $\rho_{\sigma,c}(x) = \exp(-\frac{|x-c|^2}{2\cdot\sigma^2})$ for $\sigma, c \in \mathbb{R}_{>0}$. Then, $\rho_{\sigma,c}(\mathbb{Z}) = \sum_{i=-\infty}^{\infty} \rho_{\sigma,c}(i)$. The discrete Gaussian distribution with standard deviation σ and mean c is a distribution on $\mathbb{Z}$ with the probability of $x \in \mathbb{Z}$ given by $\rho_{\sigma,c}(x)/\rho_{\sigma,c}(\mathbb{Z})$. For $c = 0$, we denote this distribution by χ_σ.

Subgaussian distribution. For the analysis of encryption parameters, we need subgaussian random variables over $\mathbb{R}$.

Definition 2.1. *A random variable V over $\mathbb{R}$ is α-subgaussian if its moment generating function satisfies*

$$\mathbb{E}[\exp(t \cdot V)] \leq \frac{1}{2}\exp(\alpha^2 \cdot t^2)$$

for all $t \in \mathbb{R}$. The tails of such a subgaussian are dominated by the tails of a Gaussian.

We define the variance $\mathsf{Var}(a)$ (resp. $\mathsf{Var}(\mathbf{x})$) of $a \in \mathcal{R}$ (resp. $\mathbf{x} \in \mathbb{Z}^n$) to be the maximum variance over the coefficients of a (resp. the components of $\mathbf{x}$). The product of two polynomials $a, b \in \mathcal{R}$ has a variance $\mathsf{Var}(a \cdot b) = N \cdot \mathsf{Var}(a) \cdot \mathsf{Var}(b)$. Similarly, the maximum variance of each column of a matrix $\mathbf{X}$ is $\mathsf{Var}(\mathbf{X})$. From the definition of a subgaussian distriction, it can be proven that the variance of an α-subgaussian random variable V is bounded by $\mathsf{Var}(V) \leq \alpha^2$. Informally, this says that the tails of V are dominated by a Gaussian function with standard deviation α. The following lemma is from [JLP23].

Lemma 2.1. *If* $\mathbf{x}$ *is a discrete random vector over* $\mathbb{R}^n$ *such that each component* x_i *of* $\mathbf{x}$ *is* α_i*-subgaussian, then the vector* $\mathbf{x}$ *is a* β*-subgaussian vector where* $\beta = \max_{i\in[n]} \alpha_i$*.*

Subgaussian random variables possess a significant property known as Pythagorean additivity. When considering two random variables, α-subgaussian X and β-subgaussian Y, alongside integers $a, b \in \mathbb{Z}$, the random variable $a \cdot X + b \cdot Y$ is known to be $\sqrt{a^2 \cdot \alpha^2 + b^2 \cdot \beta^2}$-subgaussian. This in turn implies that

$$\mathsf{Var}(a \cdot X) + \mathsf{Var}(b \cdot Y) \leq a^2 \cdot \mathsf{Var}(X) + b^2 \cdot \mathsf{Var}(Y) \leq a^2 \cdot \alpha^2 + b^2 \cdot \beta^2.$$

2.2 Hardness assumptions

Most FHE schemes base their security on the well-known LWE and RLWE problems. FINAL also uses the NTRU problem. We define them formally as follows:

Definition 2.2 (LWE). *Let* $n, q \in \mathbb{N}$*,* $\sigma \in \mathbb{R}_{>0}$*, and* χ_{key} *a distribution over* $\mathbb{Z}^n$*.*

The computational $(n, q, \sigma, \chi_{\mathsf{key}})$*-LWE problem is to recover a secret* $\mathbf{s}$ *sampled from* χ_{key} *given arbitrary many LWE samples of the form* $(\mathbf{a}_i, b_i := \mathbf{a}_i \cdot \mathbf{s} + e_i) \in \mathbb{Z}_q^{n+1}$*, where* $e_i \leftarrow \chi_\sigma$*. The decisional* $(n, q, \sigma, \chi_{\mathsf{key}})$*-LWE problem is to distringuish the samples* $(\mathbf{a}_i, b_i)$ *from vectors* $\mathbf{u}_i$ *uniformly distributed in* $\mathbb{Z}_q^{n+1}$*.*

Definition 2.3 (power-of-two RLWE). *Let* $N \in \mathbb{N}$*, where* $N = 2^k$ *for some* $k \in \mathbb{N}$*. Define* $\mathcal{R} := \mathbb{Z}[X]/\langle X^N + 1\rangle$*. Let* $q \in \mathbb{N}$*,* $\sigma \in \mathbb{R}_{>0}$*, and* χ_{key} *a distribution over* $\mathcal{R}$*.*

The decisional $(n, q, \sigma, \chi_{\mathsf{key}})$*-RLWE problem is to distinguish many RLWE samples of form* $(a_i, b_i := a_i \cdot s + e_i) \in \mathcal{R}_q$*, from uniformly distributed vectors* $\mathbf{u}_i$ *in* $\mathcal{R}_q^2$*. In the previous,* $e_i \leftarrow \chi_\sigma$ *and* s *is sampled from* χ_{key}*. Analogously, the computational* $(n, q, \sigma, \chi_{\mathsf{key}})$*-RLWE problem asks to recover the key* s *from such samples.*

Definition 2.4 (NTRU). *Let* $N, Q \in \mathbb{N}$*, where* $N = 2^k$ *for some* k*, and* $\mathcal{R} := \mathbb{Z}[X]/\langle X^N + 1\rangle$*. Let* $\sigma \in \mathbb{R}_{>0}$*,* $g, f \leftarrow \chi_\sigma^N$ *and* f *be invertible in* $\mathcal{R}_Q$*.*

The decisional (N, Q, σ) *NTRU problem is to distinguish between* $h := g \cdot f^{-1} \bmod Q$ *and a uniformly random polynomial sampled from* R_Q*. Similarly, the computational* (N, Q, σ) *NTRU asks to recover* f *from* h*.*

2.3 Decompositions

For fixed integers q and B, we set $\ell := \lceil \log_B q \rceil$ and define the gadget vector $\mathfrak{g}_{q,B} := (B^0, \ldots, B^{\ell-1})$. When q and B are clear from the context, we write $\mathfrak{g}$. Then, for any $k \in \mathbb{Z}_q$, we represent k by an integer in $[-q/2, q/2)$ and define its signed decomposition in base B as $\mathfrak{g}^{-1}(k) = (k_0, \ldots, k_{\ell-1})$ for each integer $|k_i| \leq B/2$ for $i \in [\ell]$. It is easy to see that $\mathfrak{g}^{-1}(k) \cdot \mathfrak{g} = k$. For any $f \in R_Q$, we define $\mathfrak{g}^{-1}(f) := \sum_{i=0}^{N-1} \mathfrak{g}^{-1}(f_i) X^i$. It is clear that

$$\mathfrak{g}^{-1}(f) \cdot \mathfrak{g} = \sum_{i=0}^{N-1} \mathfrak{g}^{-1}(f_i) \cdot \mathfrak{g} \cdot X^i = \sum_{i=0}^{N-1} f_i \cdot X^i = f.$$

Alternatively, a randomized gadget decomposition [GMP19,JLP21,JLP23] can be used not to rely on Heuristic assumption for independent behaviours of each element of the output noise vector.

2.4 LWE encryption scheme

FHE schemes like FHEW [DM15], TFHE [CGGI16], and FINAL [BIP+22] are actually composed of two homomorphic schemes: a simple one, that encrypts the messages and performs the homomorphic operations, which we call the base scheme, and a second scheme, called the accumulator. The former is used to encrypt the actual data and is commonly based on the LWE problem, while the latter is used in the bootstrapping of the base scheme. In this section, we present the LWE scheme with message space $\mathbb{Z}_p$.

As first discussed in [BV11], an encryption scheme can be constructed from the LWE problem as follows:

Definition 2.5 (LWE encryption). *Let $n, q, p \in \mathbb{N}$, $\sigma \in \mathbb{R}_{>0}$, and χ_{key} a distribution over $\mathbb{Z}^n$.*

We define the scaling factor as $\Delta \coloneqq q/p$. If q and p, respectively the ciphertext modulus and the plaintext modulus, are not both powers of two, a rounding must be applied at the moment of encryption. We call the following an LWE encryption of $m \in \mathbb{Z}_p$ under the secret key $s \leftarrow \chi_{\mathsf{key}}$: $(\mathbf{a}, \mathbf{b} = \sum_{i=0}^{n-1} \langle a_i, s_i \rangle + \Delta \cdot m + e)$ where elements of $\mathbf{a}$ are sampled uniformly random from $\mathbb{Z}_q$ and e is sampled from χ_σ. other than binary.

A ciphertext of this format can be decrypted by computing $m = \left\lfloor \frac{\mathbf{b} - \sum_{i=0}^{n-1} a_i \cdot s_i}{\Delta} \right\rceil$. Decryption will be successful on condition that $|e| \leq \Delta/2$.

Intuitively, during encryption, we lift the message to a higher range by multiplying it with Δ, and add noise in the lower range. This means that m is mapped to a bucket around the "clean" value. Unless the noise becomes so big that the ciphertext enters another bucket, the message can be decrypted correctly.

Note that, originally, this encryption scheme used χ_{key} as a distribution over $\{0,1\}^n$, but we shall use other key distributions later on too.

2.5 Ciphertexts for FINAL

In this section we recall the NTRU-based, GSW-like Scheme (NGS) that was presented in FINAL [BIP+22]. It is used in the accumulator in the bootstrapping algorithm. The scheme includes two encryption functions with different ciphertext forms, the first of which ($\mathsf{NGS.EncS}$) encrypts a plaintext m as an element of $\mathcal{R}_Q$, and the second ($\mathsf{NGS.EncVec}$) encrypts m as a vector of elements in $\mathcal{R}_Q$. The message m can be any polynomial in $\mathcal{R}_p$ for $\mathsf{NGS.EncS}$, but for $\mathsf{NGS.EncVec}$ it is assumed that $m \in \mathbb{M} = \{\pm b \cdot X^k \mid b \in \mathbb{Z}_p, k \in \mathbb{N}\}$. As the NGS ciphertexts are never decrypted in the bootstrapping mechanism, we omit the corresponding description here.

NGS can be described by the following four procedures:

- $\mathsf{NGS.ParamGen}(1^\lambda)$: From the security parameter, output $\mathsf{Param} := (N, p, Q, \zeta, B, l)$, with N the dimension of $\mathcal{R}$, p and Q respectively the plaintext and ciphertext modulus, ζ the standard deviation for the key, B the decomposition base and $l = \lceil \log_B(Q) \rceil$ the dimension over $\mathcal{R}_Q$ of a vector ciphertext.
- $\mathsf{NGS.KeyGen}(\mathsf{Param})$: Sample $f' \leftarrow \chi_\zeta^N$ and set $f \coloneqq 1 + p \cdot f'$ until f^{-1} exists in $\mathcal{R}_Q$. Output $sk \coloneqq f$.
- $\mathsf{NGS.EncS}(sk, m)$: Sample $g \leftarrow \chi_\zeta^N$, set $\Delta \coloneqq Q/p$, and return $c = g/f + \Delta \cdot m \in \mathcal{R}_Q$. This c is called a *scalar encryption of m*.
- $\mathsf{NGS.EncVec}(sk, m)$: Sample $g_i \leftarrow \chi_\zeta^N$ for $0 \leq i < l$. Set $\mathbf{g} \coloneqq (g_0, \ldots, g_{l-1})$ and $\mathfrak{g} = (B^0, \ldots, B^{l-1})$. Return $\mathbf{c} = \mathbf{g}/f + \mathfrak{g} \cdot m \in \mathcal{R}_Q^l$. This $\mathbf{c}$ is called a *vector encryption of m*.

An *external product* can be defined as an operation that takes an NGS scalar ciphertext encrypting a message u and an NGS ciphertext encrypting a message v, returning an NGS scalar ciphertext encrypting the message $u \cdot v$.

Definition 2.6 (External product [BIP+22]). *From a scalar encryption $c \coloneqq g/f + \Delta \cdot u \in \mathcal{R}_Q$ of the ternary polynomial u and a vector encryption $\mathbf{c} \coloneqq \mathbf{g}/f + \mathbf{g} \cdot v \in \mathcal{R}_Q^l$ of the message v, we define the vector product to be*

$$c \boxdot \mathbf{c} \coloneqq \mathbf{g}^{-1}(c) \cdot \mathbf{c} \in \mathcal{R}_Q \tag{1}$$

As $\mathbf{g}^{-1}(c) \cdot \mathbf{g} = c$, note that $c_{mult} = c \boxdot \mathbf{c}$ is equal to

$$c_{mult} \coloneqq (\mathbf{g}^{-1}(c) \cdot \mathbf{g})/f + (\mathbf{g}^{-1}(c) \cdot \mathbf{g} \cdot v) = \underbrace{(\mathbf{g}^{-1}(c) \cdot \mathbf{g} + g \cdot v)}_{g_{mult}}/f + \Delta \cdot u \cdot v. \tag{2}$$

c_{mult} is then a valid scalar encryption of $u \cdot v$ if the noise term g_{mult} is small enough.

The noise growth in the external product can be bounded by $\mathsf{Var}(\mathsf{Err}(c_{mult})) \leq N\ell\gamma^2 \cdot \mathsf{Var}(\mathbf{g}) + ||v||_2^2 \cdot \mathsf{Var}(g) + 4\zeta^2$, as proven in lemma 4 of FINAL[BIP+22]. Here, $\mathbf{g}^{-1}(a)$ is assumed γ-subgaussian for $a \in \mathcal{R}_Q$ and some $\gamma = O(B)$. For k consecutive external products with fresh vector ciphertexts, the noise is bound by $\mathsf{Var}(\mathsf{Err}(c')) \leq N\ell\gamma^2 \cdot \sum_{i=1}^{k} \mathsf{Var}(\mathbf{g_i}) + \mathsf{Var}(g_0) + 4\zeta^2$.

In Section 3.3, we introduce a modification to this definition, to reduce the cost (both computational and memory) of bootstrapping, at the cost of only insignificant noise growth.

3 Improvements for FINAL

In this section, we suggest three modifications and additions to the original FINAL scheme: we suggest how to use a larger key space, we discuss approximate decomposition in external products, and finally we explain how to enable PBS after growing the plaintext space.

3.1 Bootstrapping FINAL

Before describing our modifications, we first recall a high-level overview of FHE bootstrapping based on a blind rotation procedure, as applied in TFHE [CGGI16], FINAL [BIP+22] and XZD+23 [XZD+23]. Starting from an LWE ciphertext $c = (\langle \mathbf{a}, \mathbf{s} \rangle, b) \mod 2N$ in the base scheme, let the *test polynomial* $T(X) = \sum_{0 \leq i < N} \lfloor i/\Delta \rceil \cdot X^i$. Observe that the negacyclic rotation of T,

$$X^{2N/p - q/2N(b - \langle \mathbf{a}, \mathbf{s} \rangle)} \cdot T(X)$$

has a decryption of c in the constant coefficient.

By computing this rotation *blindly* – that is, without revealing $\mathbf{s}$, usually by having some encryption of it as extra information: the *bootstrapping* or *evaluation key* – and performing a final key switching step that extracts this coefficient as a new LWE ciphertext under $\mathbf{s}$, we have performed a bootstrap. For a properly tuned blind rotation procedure, the noise will be sufficiently reduced to allow further homomorphic operation.

In practice, the blind rotation will accumulate rotations of X^{a_i}, rotated by a secret amount s_i, with an additive homomorphism in the exponent. These individual blind rotations are usually achieved through a multiplexing operation based on s_i, or by applying an appropriate ring automorphism to (an encryption of) X^{s_i}, potentially followed by some key-switching step to ensure all individual blind rotations are encryptions under the same accumulator key, so that the additive homomorphism applies.

3.2 Larger key space

Let us first recall the (binary) CMUX, as it is used in the blind rotation procedure of FINAL. It can be seen as the homomorphic version of the Multiplexer (MUX) gate, that has three inputs and one output. Based on the value of the first input (the *selector bit* b) one of the other two input values is returned as the output value. In the homomorphic variant, all the inputs and outputs are ciphertexts. The gate computes $a_b = (b-1) \cdot a_0 + b \cdot a_1 = a_0 + b \cdot (a_1 - a_0)$.

In FINAL, the values a_0, a_1 and a_b are scalar NGS ciphertexts, and b is a vector NGS ciphertext encrypting one bit. A general extension of the CMUX gate for ν values a_i is called an ν-ary CMUX. The selector for this is no longer just a bit, but rather an index with a value between zero and $\nu - 1$. This index can be encoded as ν indicator values b_i of which exactly one has a value of one, and every other one has value zero. Then the generalized CMUX computes

$$\sum_{i=0}^{\nu-1} b_i \cdot a_i$$

which can be rewritten to perform one multiplication fewer and have one indicator (b_0) fewer by remembering that $\sum_i b_i = 1$, namely:

$$a_0 + \sum_{i=1}^{\nu-1} b_i \cdot (a_i - a_0).$$

FINAL's base encryption scheme uses a binary LWE key. Given fixed level of security (say 128 bits of security), we can consider the tradeoff between the LWE dimension n and the size of the key space, or the key distribution χ_{key}. When we increase the key space (or increase the standard deviation for χ_{key}), the dimension n can be lowered without losing the wanted security guarantees. The advantage is that the number of required "individual" blind rotations during bootstrapping lowers, as n of those are performed, but this comes at a higher cost for each such rotation. Thus, by introducing more freedom in the choice of χ_{key} for FINAL, it could be possible to find a better tradeoff.

There are two different approaches in the literature that have introduced a similar extension for TFHE. One uses the aforementioned extension from binary to ν-ary CMux gates [JP22]. The other makes use of ring automorphisms [LMK+23]. The first approach works naturally when it is applied to FINAL. However, we have found the second approach, based on ring automorphisms, to be of limited use for FINAL, due to a seeming incompatibility with the NGS ciphertext format used, that is mostly incompatible with the required key-switching procedure to the accumulator key. [XZD+23] manages to bypass this limitation with a similar, but sufficiently different ciphertext format. We discuss this scheme later on in section 4, alongside similar enhancements as discussed in this section.

General extension of ternary CMux gate In order to deal with a larger secret key space in FINAL, we employ an ν-ary CMux gate in the bootstrapping. We use the idea proposed by Joye and Paillier in [JP22]. Other literature such as [LMK+23] combines existing similar bootstrapping methods, resulting in a scheme that has a smaller noise growth than either. However, this method is slower than the one of [JP22]. The number of polynomial multiplications required in the bootstrapping algorithm is $2nl$ for [JP22], and $3nl$ for [LMK+23]. It must be noted, however, that the latter algorithm requires less space for key storage.

We now provide a concrete definition for the ν-ary CMux gate, as applied to FINAL's NGS ciphertexts. Let the LWE key $\mathbf{s} \leftarrow [-K, K]$, and let its components s_i be the selector values. We can then define $2K$ keys $\mathtt{bsk}_{i,j}$ for $j = -K, \ldots -1, 1, \ldots, K$ such that

$$\mathtt{bsk}_{i,j} := \begin{cases} \mathsf{NGS.EncVec}(1) & \text{when } s_i = j \\ \mathsf{NGS.EncVec}(0) & \text{otherwise} \end{cases}$$

The $(2 \cdot K + 1)$-ary CMux gate that computes the blind rotation $X^{c_i \cdot s_i}$ then becomes

$$\mathtt{CMux}_i(c_i) := 1 + \sum_{j=-K, j\neq 0}^{K} (X^{c_i \cdot j} - 1) \cdot \mathtt{bsk}_{i,j} \tag{3}$$

It is clear that for a given c_i, there will never be more than one term in the summation that is nonzero, and that correctness for the blind rotation is achieved.

With this definition, we can now determine a bound on the noise output of a single CMux gate, in function of the noise $\mathsf{Err}(\mathtt{bsk})$ of a fresh vector encryption. Let $c_{\mathsf{Mux}} := \mathtt{CMux}_i(c_i)$ for any $0 \leq i \leq n-1$. Then, the following holds:

$$\begin{aligned} \mathsf{Var}(\mathsf{Err}(c_{\mathsf{Mux}})) &\leq \sum_{k=-K, k\neq 0}^{K} ||X^{c_i} - 1||_2^2 \cdot \mathsf{Var}(\mathsf{Err}(\mathtt{bsk}_{i,k})) \\ &\leq 4K \cdot \mathsf{Var}(\mathsf{Err}(\mathtt{bsk})). \end{aligned}$$

In algorithm 1, we present the bootstrapping from [BIP+22], as applied to our new context, with an updated definition of $\mathtt{CMux}$ and a generalized plaintext space. It can be verified that this algorithm corresponds to the high-level overview at the start of this section, and that therefore the constant term of $\mathsf{msg}(\mathsf{ACC})$ in line 6 is $\lfloor Q/p \rceil \cdot m$.

Algorithm 1: Bootstrapping algorithm for $m \in \mathbb{Z}_p$

Input: $\mathtt{ct} \in \mathbb{Z}_q^{n \times n}$, a base scheme ciphertext encrypting $m \in \mathbb{Z}_p$
$\{\mathtt{bsk}_{i,j}\}_{0 \leq i \leq n-1, j \in [-K,K] \setminus \{0\}}$, bootstrapping keys, where each $\mathtt{bsk}_{i,j} \in \mathcal{R}_{Q,N}^l$
$\mathtt{ksk}$, a key-switching key from the NGS secret key $f \in \mathcal{R}$ to the base scheme secret key $\mathbf{s} \in \mathbb{Z}^n$
Output: $\mathtt{ct}' \in \mathbb{Z}_q^n$, a base scheme ciphertext encrypting the same m.

1 $((a_0, \ldots, a_{n-1}), b) \leftarrow \lfloor \frac{2 \cdot N \cdot \mathtt{ct}}{q} \rceil$
2 $\mathsf{ACC} \leftarrow \lfloor \frac{Q}{p} \rceil \cdot X^{-b+N/p} \cdot \sum_{i=0}^{N-1} X^i$;
3 **for** $i = 0 \ldots n-1$ **do**
4 $\quad c_{\mathsf{Mux}} \leftarrow \mathtt{CMux}_i(a_i)$;
5 $\quad \mathsf{ACC} \leftarrow \mathsf{ACC} \boxdot c_{\mathsf{Mux}}$;
6 $\mathsf{ACC} \leftarrow \mathsf{ModSwitch}_{Q \to q}(\mathsf{ACC})$
7 $\mathtt{ct}' \leftarrow \mathsf{KeySwitch}(\mathsf{ACC}, \mathtt{ksk})$[4]
8 **return** $\mathtt{ct}'$

Noise Analysis Now we consider the noise of the output of this bootstrapping algorithm. In the first line, we switch the modulus of the input ciphertext from q to $2N$. We denote the resulting

ciphertext by (a_{2N}, b_{2N}). We then have:

$$\left|\frac{b-\langle \mathbf{a},\mathbf{s}\rangle}{\Delta} - \frac{q}{2N}\cdot\frac{b_{2N}-\langle \mathbf{a}_{2N},\mathbf{s}\rangle}{\Delta}\right| \leq \frac{q}{4\cdot N}\cdot\left|\frac{b-\langle \mathbf{a},\mathbf{s}\rangle}{\Delta}\right|$$

From line 3 to 5 of the bootstrapping algorithm, the output ACC is obtained by performing n external products with c_{Mux} whose noise variance is $\mathsf{Var}(\mathsf{Err}(c_{\mathsf{Mux}}))$. The variance of the final $\mathsf{Err}(\mathsf{ACC})$ can therefore be bounded by:

$$\begin{aligned}\mathsf{Var}(\mathsf{Err}(\mathsf{ACC})) &\leq n\cdot N\cdot l\cdot\gamma^2\cdot\mathsf{Var}(\mathsf{Err}(c_{\mathsf{Mux}})) + 4\cdot||\mathsf{msg}(\mathsf{ACC})||_2^2\cdot\zeta^2\\ &\leq n\cdot N\cdot l\cdot\gamma^2\cdot\mathsf{Var}(\mathsf{Err}(c_{\mathsf{Mux}})) + 4Np^2\cdot\zeta^2\end{aligned}$$

After this step, another modulus switch is applied. Following lemma 6 of [BIP+22], this leads to a bound of

$$\begin{aligned}\mathsf{Var}(\mathsf{Err}(\mathsf{ACC})) \leq\ & (q/Q)^2\cdot n\cdot N\cdot l\cdot\gamma^2\cdot\mathsf{Var}(\mathsf{Err}(c_{\mathsf{Mux}}))\\ &+(q/Q)^2\cdot 4\cdot Np^2\cdot\zeta + 1 + 16\cdot N\cdot\zeta^2\end{aligned}$$

After the external product with a key switching key $\mathtt{ksk}$ in line 7, the noise in the resulting $\mathtt{ct}'$ has a variance

$$\begin{aligned}\mathsf{Var}(\mathsf{Err}(\mathtt{ct})) &\leq N\cdot L\cdot\gamma^2\cdot\mathsf{Var}(\mathsf{Err}(\mathtt{ksk})) + \mathsf{Var}(\mathsf{Err}(\mathsf{ACC}))\\ &\leq N\cdot L\cdot\gamma^2\cdot\mathsf{Var}(\mathsf{Err}(\mathtt{ksk}))\\ &+ (q/Q)^2\cdot n\cdot N\cdot l\cdot\gamma^2\cdot\mathsf{Var}(\mathsf{Err}(c_{\mathsf{Mux}}))\\ &+ (q/Q)^2\cdot 4\cdot N\cdot\zeta^2 + 1 + 16\cdot N\cdot\zeta^2\end{aligned} \tag{4}$$

where L is the dimension of the key switching key.

Under the central limit heuristic, we can bound the noise of the output $\mathtt{ct}'$ of this algorithm. Given an LWE ciphertext $\mathtt{ct}$ encrypting a message m, algorithm 1 outputs an LWE ciphertext $\mathtt{ct}'$ encrypting the same message. In adition, under the central limit heuristic, the noise contained in the output behaves as a Gaussian distribution, hence, with overwhelming probability, it satisfies the following bound

$$\begin{aligned}||\mathsf{Err}(\mathtt{ct}')||_2^2 \leq 6^2\cdot(&N\cdot L\cdot\gamma^2\cdot\varepsilon_{\mathtt{ksk}} + 4\cdot K\cdot(q/Q)^2\cdot n\cdot N\cdot l\cdot\gamma^2\cdot\varepsilon_{\mathtt{bsk}}\\ &+ (q/Q)^2\cdot 4\cdot N\cdot\zeta^2 + 1 + 16\cdot N\cdot\zeta^2)\end{aligned}$$

where $\varepsilon_{\mathtt{ksk}} = O(\mathsf{Var}(\mathsf{Err}(\mathtt{ksk})))$ and $\varepsilon_{\mathtt{bsk}} = O(\mathsf{Var}(\mathsf{Err}(\mathtt{bsk})))$.

Parameters and efficiency This analysis of the ν-ary CMux was prompted by the hope that a larger key space could allow for a lower LWE dimension, and hence better efficiency without affecting the security level of the scheme. However, the efficiency does not solely depend on the LWE dimension n, but also on the computational cost of the CMUX evaluation, which grows linearly in K (when we let the secret key be sampled from $[-K, K]$). In particular, we only consider the computational cost of an FFT transformation (either forward or inverse) and that of a pointwise multiplication (in the FFT domain) as significant, and ignore all other costs. Denote the time cost for these as respectively T_{FFT} and T_{PM}. For a bootstrapping operation with LWE dimension n and secret key sampled in $[-K, K]$, we see the following two expensive parts of Algorithm 1: $2Kn\ell\cdot T_{\mathsf{PM}}$

for the CMux evaluations and $n(\ell+1)\cdot T_{\mathsf{FFT}} + n\ell \cdot T_{\mathsf{PM}}$ for the external products, leading to a total cost of

$$\mathsf{cost}_K := n\left((\ell+1)\cdot T_{\mathsf{FFT}} + (2K+1)\ell \cdot T_{\mathsf{PM}}\right).$$

When we compare this to the cost of a FINAL bootstrap with binary keys of LWE (resp. NGS vector ciphertext) dimension n' (resp. ℓ'):

$$\mathsf{cost}_{\mathsf{bin}} := n'\left((\ell'+1)\cdot T_{\mathsf{FFT}} + 2\ell' \cdot T_{\mathsf{PM}}\right).$$

It seems reasonable to assume that an FFT takes longer to compute than a pointwise multiplication of the same length, and as such that $T_{\mathsf{FFT}} > T_{\mathsf{PM}}$, but when increasing K, the corresponding value of n does not decrease as rapidly. Therefore, we can see that any decision as to which variant is preferable will depend on exactly how much faster the pointwise multiplication is. We can quantify this with the value $\alpha := \frac{T_{\mathsf{FFT}}}{T_{\mathsf{PM}}}$, which will depend on the machine used to run the bootstrapping and the chosen FFT implementation.

Here, we provide both a bound on the optimal choice of K, depending on α and the respective dimensions for the ciphertexts, and, in table 1, an overview of some values α for different FFT libraries. We use degree 2^{14} polynomials for these tests, as that is a reasonable upper bound for practical FHE parameters, as well as resulting in an expectation of a more pronounced asymptotic difference. The actual value of α will hence depend not only on implementation and hardware efficiency, but also on the choice of parameters. These measurements were all made on a `AMD Ryzen 5 PRO 3400G` CPU, in a single threaded context. Where possible, a choice for a real-to-complex variant of the FFT was made, to better match the usage in the scheme. Theoretically, the optimal choice for K would correspond to

$$K < \frac{\alpha(n'(\ell'+1) - n(\ell+1)) + 2n'\ell' - n\ell}{2nl}.$$

For the sake of intuition, if we assume the extremely generous situation where we could achieve $(n,\ell) = (n'/2,\ell')$ with $\ell' = 1$, the obtained bound is $K < \alpha + \frac{3}{2}$, implying that T_{FFT} should be rougly K times as slow as T_{PM}. As can be seen in table 1, even under this generous assumption, the gain when choosing an efficient FFT implementation is very limited.

Table 1. Observed running times for FFT and pointwise multiplication under different programming languages and for different FFT libraries, running on degree 2^{14} polynomials.

Language	FFT Library	T_{FFT} (ms)	T_{PM} (ms)	$\alpha := \frac{T_{\mathsf{FFT}}}{T_{\mathsf{PM}}}$
C++	spqlios	59	21	2.81
C++	ffnt	245		11.67
C++	fftw	46		2.19
C++	nayuki	165		7.86
Rust	concrete_fft	86	26	3.31
Rust	concrete_fftw	42		1.62

The issue with automorphisms The other approach to introduce larger keyspaces, as considered in [LMK+23], makes use of ring automorphisms on $\mathcal{R}$: $\mathtt{ct}(X) \mapsto \mathtt{ct}(X^k)$. These are cheap to

compute, as they only involve a reordering and some sign changes of the coeffients of a polynomial. However it also changes the encryption key from $f(X)$ to $f(X^k)$. To account for this, the automorphism should be followed by a key switching procedure back to $f(X)$. That is, we want some (efficient) procedure $\mathsf{NGS.EncS}(f(X), m) = \mathsf{KeySwitch}(\mathsf{NGS.EncS}(f(X^k), m), \mathtt{ksk})$ that does not change the underlying message. Additionally, the additional noise from the key switching should be limited to fit within the bootstrapping noise budget.

In other words, we start with a scalar NGS ciphertext which is a form of $c := g/f + \Delta m$, and end up with another scalar NGS ciphertext $g'/f' + \Delta m$. An insightful approach is to do an external product with an NGS vector ciphertext encrypting f under f'. If we limit ourselves to choices of $\mathtt{ksk}$ such that $\mathsf{KeySwitch}(c, \mathtt{ksk}) = c \boxdot \mathtt{ksk}$, as an arbitrary limit to mean "efficient" that simulatenously giving an easy evaluation of the noise growth, it can be seen that the result generally falls into either of two cases. It does not have the correct form to be a scalar NGS ciphertext, or the noise growth is too high.

It should be noted that [XZD+23] manages to perform a blind rotation with ring automorphisms, by choosing an alternative scalar ciphertext form that allows for key switching with an external product: $Enc_f(m) = \frac{\tau \cdot g + \Delta \cdot u}{f} \in \mathcal{R}_Q$. We extend our results for FINAL to this alternative form in section 4 and afterwards compare the resulting FHE schemes in section 5.

3.3 Approximate decomposition

Recall the definition of a scalar NGS ciphertext encrypting a message m_0. We now define a *noisy scalar NGS ciphertext*, which has an extra noise term $e \in \mathcal{R}_Q$:

$$\hat{c} = \mathsf{NGS.EncS}_e(f, m_0) = \frac{g}{f} + \Delta \cdot m_0 + e.$$

The decryption error for a noisy ciphertext then becomes $g + e \cdot f$, and in order to have the additional error due to e be comparable to the error also present in a regular ciphertext, we introduce the following condition, to enforce that the variance of the newly introduced error term does not exceed the noise already present normally:

$$\mathsf{Var}(e) \leq \frac{\mathsf{Var}(g)}{N\mathsf{Var}(f)} \tag{5}$$

The external product can be extended to cover multiplication between a noisy scalar ciphertext and a (regular) vector ciphertext encrypting m_1 without any further changes. It can be verified that the result is again a noisy scalar NGS ciphertext.

$$\begin{aligned}
\hat{c} \boxdot \mathbf{c} = \langle \mathfrak{g}_B^{-1}(\hat{c}), \mathbf{c} \rangle &= \sum_{i=0}^{\ell-1} \frac{c_i g_i}{f} + \hat{c} m_1 \\
&= \frac{\sum_{i=0}^{\ell-1} c_i g_i + m_1 g}{f} + \Delta m_0 m_1 + m_1 e \\
&= \frac{\bar{g}}{f} + \Delta m_0 m_1 + \bar{e}
\end{aligned} \tag{6}$$

We can now also define the *approximate decomposition* of a (either noisy or regular) scalar ciphertext c as the decomposition $\mathfrak{g}_B^{-1}(c)$, with the k least significant digits set to 0. Alternatively,

this can be expressed as the decomposition $\mathfrak{g}_B^{-1}(\hat{c})$ of the noisy scalar ciphertext $\hat{c} = c - [c]_{B^k}$. When we consider the external product that uses an approximate decomposition, or equivalently, the external product $\hat{c} \boxdot \mathbf{c}$,[5] we see from equation 6 that the resulting noise terms become $\bar{g} = \sum_{i=k}^{l-1} c_i g_i + m_1 g$ and $\bar{e} = m_1(e - [c]_{B^k})$. We can hence compute a bound for the variance of the resulting errors, where we assume $\mathbf{c}$ to be a fresh vector encryption.

$$\begin{aligned}
\mathsf{Var}(\bar{g}) &\leq \mathsf{Var}(m_1 g) + \mathsf{Var}\left(\sum_{i=k}^{\ell-1} c_i g_i\right) \\
&\leq \mathsf{Var}(g) + NB^2(\ell - k)\mathsf{Var}(g_i) \\
&\leq p^2\mathsf{Var}(g) + NB^2(\ell - k)\mathsf{Var}(f)
\end{aligned}$$

$$\begin{aligned}
\mathsf{Var}(\bar{e}) &\leq \mathsf{Var}(m_1 e) + \mathsf{Var}(m_1[c]_{B^k}) \\
&\leq \frac{p^2\mathsf{Var}(g)}{N\mathsf{Var}(f)} + p^2 B^k
\end{aligned}$$

Entering this into inequality 5, we get a bound on k, the number of digits that can be ignored by the approximate decomposition:

$$\begin{aligned}
\frac{p^2\mathsf{Var}(g)}{N\mathsf{Var}(f)} + p^2 B^{2k} &\leq \frac{p^2\mathsf{Var}(g) + NB^2(\ell - k)\mathsf{Var}(f)}{N\mathsf{Var}(f)} \\
B^{2k-2} &\leq \ell - k \\
k &\leq \frac{1}{2}\log_B(\ell - k) - log_B(p) + 1
\end{aligned}$$

The advantage gained from this external product with approximate decomposition is that we can omit the k least significant components of any vector ciphertext involved as well, without further effect. In practice, this amounts to a saving of k forward FFT computations and k pointwise multiplications. The computation and memory costs are now identical to having only vector dimension $d - k$, rather than d. This does come at the cost of a limited noise growth, which has no practical impact since a bootstrapping operation is performed after every homomorphic operation.

3.4 Programmable bootstrapping

After combining the bootstrapping algorithm with plaintext space $\mathbb{Z}_p$ as in algorithm 1 and enhancing it with a free choice of ν-ary CMux gates and approximate decompositions, we can introduce programmable bootstrapping (PBS) into the mix. Recall that the bootstrapping procedure makes use of a blind rotation followed by a keyswitch back to the base scheme. The goal of the blind rotation is to rotate the coefficients of a test polynomial $T(X) \in \mathcal{R}_Q$ by a decryption of an LWE ciphertext, in such a way that the constant coefficient of the underlying plaintext polynomial ends up containing the decryption of the LWE ciphertext. In the description of algorithm 1, the initial value of ACC represents (a noiseless encryption of) the polynomial $T(X)$.

[5] We allow the ciphertext being decomposed approximately to already be noisy, so the extra noise term of $\hat{c}$ becomes $e - [c]_{B^k}$.

When considering PBS, this notion is enhanced such that the constant coefficient no longer just contains the message m, but the evaluation of an arbitrary function $F(m)$, usually represented by a lookup table of size p.[6]

To encode the function or lookup table F into a polynomial $T(X)$, we observe that

$$(T \cdot X^{\langle \mathbf{a},\mathbf{s}\rangle - b})(0) = F(m) = F\left(\left\lfloor \frac{\langle \mathbf{a}, \mathbf{s}, \rangle - b}{\Delta/2} \right\rceil\right)$$

and so, when we let $T(X) = t_0 + t_1 X + \ldots + t_{N-1} X^{N-1}$, choosing

$$t_i = F\left(\left\lfloor \frac{2i}{\Delta} + \frac{1}{2} \right\rceil\right)$$

satisfies this requirement. We note here that we consider the decryption after a modulus switch to $2N$, and that as such $\Delta = 2N/p$ here. Applying this construction to the identity function $F(x) = x$, we indeed recover the previous non-programmable bootstrapping algorithm. When the blind rotation and keyswitching procedures work correctly and with appropriate noise growth, it then follows that the addition of PBS preserves correctness and adds no further noise growth to the final bootstrapped ciphertext.

Note that for LWE, here we use $\frac{\Delta}{2}$ rather than Δ. This extra bit is needed to allow programmable bootstrapping with non-negacyclic functions. Alternatively, when F is a negacyclic function, its compatibility with the negacyclic nature of the cyclotomic ring $\mathcal{R}$ permits some extra efficiency in the plaintext space within ciphertexts. It is then possible to use the regular choice of $\Delta = q/p$ (or $\Delta_{2N} = 2N/p$ after modulo switching) for the LWE ciphertexts.

4 Improvements for XZD+23

Since [XZD+23] follows a comparable blueprint to [BIP+22], with a modification to the scalar NTRU ciphertext and the introduction of ring automorphisms in the blind rotation, the natural question becomes whether the modification made for FINAL can also be applied to the XZD+23 scheme. By construction, the scheme already supports arbitrary distributions for the LWE key, so we need only consider the approximate decomposition and the programmable bootstrapping.

4.1 Approximate decomposition

Parallel to the approximate decomposition for FINAL, we first recall that a scalar ciphertext encrypting a message m_0 has the form

$$c = \frac{\tau g + \Delta m_0}{f}$$

and then define a *noisy scalar ciphertext* to have an additional noise term, such that

$$\hat{c} = \frac{\tau g + \Delta m_0}{f} + e.$$

The external product with a vector ciphertext $\mathbf{c} = \tau \frac{\mathbf{g}}{f} + \mathbf{g} m_1$ then becomes

[6] Since the memory and computational cost for FHE tend to grow rapidly as the plaintext modulus p grows, the memory requirements to represent and store this lookup table are easily met. In practice, $\mathbb{Z}_p$ will seldom be much larger than a single byte at once

$$\begin{aligned}\hat{c} \boxdot \mathbf{c} = \langle \mathfrak{g}_B^{-1}(\hat{c}), \mathbf{c} \rangle &= \sum_{i=0}^{\ell-1} \tau \frac{\hat{c}_i g_i}{f} + \hat{c} m_1 \\ &= \frac{\tau \left(\sum_{i=0}^{\ell-1} \hat{c}_i g_i + m_1 g \right) + \Delta m_0 m_1}{f} + m_1 e \\ &= \frac{\tau \bar{g} + \Delta m_0 m_1}{f} + \bar{e} \end{aligned} \tag{7}$$

To ensure the extra noise e does not grow too big or break correctness, we seek to bound its variance by that of g, and as such require that

$$\mathsf{Var}(e) \leq \frac{\tau^2 \mathsf{Var}(g)}{N \mathsf{Var}(f)} \tag{8}$$

Now let $\hat{c}$ represent an approximation of some (potentially noisy) ciphertext c, such that $\hat{c} = c - [c]_{B^k}$. That is, $\mathfrak{g}_B^{-1}(\hat{c})$ corresponds to $\mathfrak{g}^{-1}(c)$ with the k least significant digits set to zeros. Then we can additionally assume that the k corresponding elements of $\mathbf{c}$ are arbitrary, or even absent, which means we need less memory to store $\mathbf{c}$ and less time to compute $\hat{c} \boxdot \mathbf{c}$. From 7, we can determine bounds on the variance of the corresponding $\bar{g}$ and $\bar{e}$, under the assumption that $\mathbf{c}$ is a fresh vector ciphertext and that hence $\mathsf{Var}(g_i) = \mathsf{Var}(f)$.

$$\mathsf{Var}(\bar{g}) \leq p^2 \mathsf{Var}(g) + N B^2 (\ell - k) \mathsf{Var}(f) \mathsf{Var}(\bar{e}) \leq \frac{\tau^2 p^2 \mathsf{Var}(g)}{N \mathsf{Var}(f)} + p^2 B^k \tag{9}$$

If we then further constrain $\mathsf{Var}(\bar{g})$ and $\mathsf{Var}(\bar{e})$ to follow 8, we derive

$$k \leq \frac{1}{2} \log_B(\ell - k) - log_B(p) + 1 \tag{10}$$

as a limit on the number of components we can omit from any stored vector ciphertexts and from the external product calculations. It can be observed that the gain in efficiency is equivalent to that which we achieve with the optimization for FINAL. It should be noted that XZD+23 can also be easily adapted to support programmable bootstrapping following the same blueprint as described in Section 3.4.

5 Comparison

Comparing performance of different FHE schemes without implicitly comparing implementations along with their respective amount of processor-specific optimizations and implementation-defined peculiarities is all but impossible. We eschew this approach in favor of a more theoretical comparison. In particular, we examine the external products performed during a single bootstrapping operation, as that is generally the most expensive operation to be performed. We quantify the number of FFT operations (either forward or inverse) and the number of polynomial pointwise products in the FFT domain, incurred by the external product and otherwise. As discussed for the evaluation of our ν-ary CMux in section 3.2, it is possible to express the difference in timing between these two operations by some factor α, but since this factor again highly depends on choice

of implementation and hardware, we do not further combine these two terms in our comparison. We present a side-by-side comparison of these metrics for the FINAL and XZD+23 schemes in table 2. Note that in XZD+23, the authors report a clear improvement when compared to TFHE based on implementations of both schemes. The difference of this comparison is more pronounced than we would expect from our theoretical results counting the number of required FFT and pointwise multiplications.

For the memory requirements of a bootstrapping operation, we refer solely to the size of the bootstrapping or evaluation keys needed, as the keyswitching procedure from the accumulator back to the base ciphertext is similar enough between all examined schemes. We express all memory sizes in amount of complex numbers needed,[7] as everything can be precomputed and stored in the FFT domain. These numbers are also presented in table 2.

As indicated, parameter choices will differ between schemes. In table 3 we give the LWE and NTRU parameters for our adapted FINAL scheme for 2, 4 and 8 message bits respectively. The LWE parameters were chosen equal to those of TFHE-rs (for the same message spaces, respectively). FINAL will achieve (at least) the same message accuracy p, as FINAL has less noise for the same parameters. Next, we ensure a security level of 128 bits, by defining the NTRU parameters using the NTRU Fatigue estimator [Dv21]. We furthermore guarantee a maximal decryption error probability of 2^{-40}, with respect to the noise bound on a bootstrapped ciphertext (eq. 4).

Finally, in table 4, we show how increasing the key space influences the LWE dimension, and how this in turn influences the run-time of the bootstrapping algorithm. We find that α values under 10, it is never beneficial for the performance to have $K > 1$. Also, as given in table 2, larger K also increases the required memory.

It is difficulty to fairly compare our improved version of FINAL with that of XDZ+23 due to effects of the increased key space on the LWE dimension. If we use a uniform distribution over $[-3, 3]$ (that is, we set $K = 3$) for our version of FINAL, as an approximation of the Gaussian distribution with variance $4/3$ (as is given as a parameter set in [XZD+23]), we can compare our schemes both with our theoretical results and with their experimental outcome. We take the parameter set described by XZD+23 as *P128G*, which is more lenient having allowing a decryption error probability up to 2^{-34}. We then find, from the formulas in table 2, that for this setting, our algorithm requires one third of the FFTs, but 2.3 times the number of pointwise multiplications, while consuming 16.5% less memory. For α-values larger than 0.96. The instantiation of FINAL with our modifications is hence more performant than our instantiation of XZD+23.

The effects of our modifications on the two schemes are not just found in performance, as they expand the functionality as well. Still, in both schemes, using the approximate decomposition will grant a decrease of ℓ with at least 1, depending on the other parameters. Furthermore, we repeat that the ν-ary CMux provides a trade-off between pointwise multiplications and FFTs, which could be beneficial depending on the FFT library used. The performance of unaltered FINAL can be found by fixing K in table 2.

By examining the noise growth for each individual scheme, it can be observed that for binary LWE keys, the parameter choice be close to identical. When growing the key space either by the application of ν-ary CMux gates or by taking advantage of efficient ring automorphism computation being possible, the LWE dimension can be reduced without dropping below a wanted security threshold. It is however the case that this improvement does not generally weigh up against the constant factors seen in the different counts of operations. The XZD+23 scheme has the advantage

[7] This would usually be equivalent to 16 bytes each.

Table 2. Overview of the number of FFTs, pointwise multiplications (PM) and memory for the bootstrapping and evaluation keys (expressed in amount of complex numbers) per FHE scheme. A + superscript indicates a scheme incorporating our improvements.

Scheme	#FFT	#PM	Memory
FINAL$^+$	$n_F(\ell_F+1)$	$(2K+1)n_F\ell_F$	$(2K)n_F N_F \ell_F$
XZD+23$^+$	$2n_X(\ell_X+1)$	$2n_X\ell_X$	$N_X^2\ell_X$

Table 3. The parameters for FINAL with our modifications, for different plaintext size p, using ternary secret keys.

p	n	q	B	ℓ	N	Q
2^2	656	2^{64}	2^8	2	2^{10}	2^{20}
2^4	742	2^{64}	2^{23}	1	2^{10}	2^{20}
2^8	1017	2^{64}	2^{15}	2	2^{11}	2^{30}

that its memory usage is entirely independent from the choice of key space, and its running time only decreases as the LWE dimension decreases. For the tradeoffs involved in the adaptation of FINAL to larger key sizes, we refer the reader back to section 3.2.

To get an idea about the concrete memory requirements presented here, we need to be able to compare some typical values for the NTRU dimension N and the LWE dimension n, as they have a strong impact on the interpretation of table 2. For a security target level of 128 bits, a typical choice of n would be some value $2^9 \leq n \leq 2^{10}$. The NTRU dimension N needs to grow more as the plaintext space $\mathbb{Z}_p$ grows, though it must remain a power of two to preserve correctness. For $p = 2$,[8] the choice $N = 2^{10}$ is made. It then grows along a growing p, and for moderately sized p, already values such as $N = 2^{14}$ are required. We thus observe that generally $N > n$, and a term quadratic in N in the memory requirements can quickly become prohibitive for larger plaintext sizes.

6 Conclusion

In this paper, we have suggested improvements for the two leading FHE schemes with NTRU-based bootstrapping, FINAL [BIP+22] and XZD+23 [XZD+23]. For both schemes, we described an approximate decomposition technique that allows improvements in both memory requirements and the performance of external products, by decreasing the dimensions of vector ciphertexts. Additionally, we established the feasilibity of increases in the size of the plaintext space, further enabling programmable bootstrapping for arbitrary functions $\mathbb{Z}_p \to \mathbb{Z}_p$. In FINAL, we established the potential for an increase in the key space of the base scheme, based up techniques by Joye and Paillier [JP22], finding its application to provide only limited benefits.

Finally, we provided a comparison of the two main approaches to an increased key space for NTRU-based FHE. In this, we found FINAL to be the better choice in most choices, due to a constant factor difference in amount of required external products and the difference in memory

[8] That is, the case where no PBS is possible or needed, as described in [BIP+22] and [XZD+23].

Table 4. The influence of key space K on n, and the minimum value of α where run-time is better than $K = 1$.

p	2^2				2^4				2^8			
K	2	3	5	10	2	3	5	10	2	3	5	10
n	568	550	530	500	656	635	615	585	935	915	890	860
α	10.9	18.8	31.7	55.7	13.3	21.8	36.7	65.1	20.8	33.9	54.1	96.6

required for the bootstrapping keys, despite the increase in key size coming at a higher cost than XZD+23 in general.

Acknowledgements

This work has been supported in part by the FWO under an Odysseus project GOH9718N, and CyberSecurity Research Flanders with reference number VR20192203.

References

AP14. Jacob Alperin-Sheriff and Chris Peikert. Faster bootstrapping with polynomial error. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part I*, volume 8616 of *LNCS*, pages 297–314. Springer, Heidelberg, August 2014.

BGV12. Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (Leveled) fully homomorphic encryption without bootstrapping. In Shafi Goldwasser, editor, *ITCS 2012*, pages 309–325. ACM, January 2012.

BIP+22. Charlotte Bonte, Ilia Iliashenko, Jeongeun Park, Hilder V. L. Pereira, and Nigel P. Smart. FINAL: Faster FHE instantiated with NTRU and LWE. In Shweta Agrawal and Dongdai Lin, editors, *ASIACRYPT 2022, Part II*, volume 13792 of *LNCS*, pages 188–215. Springer, Heidelberg, December 2022.

BV11. Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) LWE. In Rafail Ostrovsky, editor, *52nd FOCS*, pages 97–106. IEEE Computer Society Press, October 2011.

CGGI16. Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *ASIACRYPT 2016, Part I*, volume 10031 of *LNCS*, pages 3–33. Springer, Heidelberg, December 2016.

CH18. Hao Chen and Kyoohyung Han. Homomorphic lower digits removal and improved FHE bootstrapping. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part I*, volume 10820 of *LNCS*, pages 315–337. Springer, Heidelberg, April / May 2018.

CJP21. Ilaria Chillotti, Marc Joye, and Pascal Paillier. Programmable bootstrapping enables efficient homomorphic inference of deep neural networks. In Shlomi Dolev, Oded Margalit, Benny Pinkas, and Alexander Schwarzmann, editors, *Cyber Security Cryptography and Machine Learning*, pages 1–19, Cham, 2021. Springer International Publishing.

DM15. Léo Ducas and Daniele Micciancio. FHEW: Bootstrapping homomorphic encryption in less than a second. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part I*, volume 9056 of *LNCS*, pages 617–640. Springer, Heidelberg, April 2015.

Dv21. Léo Ducas and Wessel P. J. van Woerden. NTRU fatigue: How stretched is overstretched? In Mehdi Tibouchi and Huaxiong Wang, editors, *ASIACRYPT 2021, Part IV*, volume 13093 of *LNCS*, pages 3–32. Springer, Heidelberg, December 2021.

GIKV23. Robin Geelen, Ilia Iliashenko, Jiayi Kang, and Frederik Vercauteren. On polynomial functions modulo p^e and faster bootstrapping for homomorphic encryption. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part III*, volume 14006 of *LNCS*, pages 257–286. Springer, Heidelberg, April 2023.

GMP19. Nicholas Genise, Daniele Micciancio, and Yuriy Polyakov. Building an efficient lattice gadget toolkit: Subgaussian sampling and more. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part II*, volume 11477 of *LNCS*, pages 655–684. Springer, Heidelberg, May 2019.

HS15. Shai Halevi and Victor Shoup. Bootstrapping for HElib. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part I*, volume 9056 of *LNCS*, pages 641–670. Springer, Heidelberg, April 2015.

HS21. Shai Halevi and Victor Shoup. Bootstrapping for HElib. *Journal of Cryptology*, 34(1):7, January 2021.

JLP21. Sohyun Jeon, Hyang-Sook Lee, and Jeongeun Park. Efficient lattice gadget decomposition algorithm with bounded uniform distribution. *IEEE Access*, 9:17429–17437, 2021.

JLP23. Sohyun Jeon, Hyang-Sook Lee, and Jeongeun Park. Practical randomized lattice gadget decomposition with application to fhe. In Gene Tsudik, Mauro Conti, Kaitai Liang, and Georgios Smaragdakis, editors, *Computer Security – ESORICS 2023*, pages 353–371, Cham, 2023. Springer Nature Switzerland.

Joy22. Marc Joye. On NTRU-ν-um modulo $X^N - 1$. Cryptology ePrint Archive, Report 2022/1092, 2022. https://eprint.iacr.org/2022/1092.

JP22. Marc Joye and Pascal Paillier. Blind rotation in fully homomorphic encryption with extended keys. In Shlomi Dolev, Jonathan Katz, and Amnon Meisels, editors, *Cyber Security, Cryptology, and Machine Learning*, pages 1–18, Cham, 2022. Springer International Publishing.

Klu22. Kamil Kluczniak. NTRU-v-um: Secure fully homomorphic encryption from NTRU with small modulus. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *ACM CCS 2022*, pages 1783–1797. ACM Press, November 2022.

LMK+23. Yongwoo Lee, Daniele Micciancio, Andrey Kim, Rakyong Choi, Maxim Deryabin, Jieun Eom, and Donghoon Yoo. Efficient FHEW bootstrapping with small evaluation keys, and applications to threshold homomorphic encryption. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part III*, volume 14006 of *LNCS*, pages 227–256. Springer, Heidelberg, April 2023.

XZD+23. Binwu Xiang, Jiang Zhang, Yi Deng, Yiran Dai, and Dengguo Feng. Fast blind rotation for bootstrapping FHEs. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part IV*, volume 14084 of *LNCS*, pages 3–36. Springer, Heidelberg, August 2023.