IACR Transactions on Cryptographic Hardware and Embedded Systems
ISSN 2569-2925, Vol. 2026, No. 1, pp. 82–104. DOI:10.46586/tches.v2026.i1.82-104

Don't be mean: Reducing Approximation Noise in TFHE through Mean Compensation

Thomas de Ruijter, Jan-Pieter D'Anvers and Ingrid Verbauwhede

COSIC KU Leuven, Leuven, Belgium, {firstname.lastname}@esat.kuleuven.be

Abstract. Fully Homomorphic Encryption (FHE) allows computations on encrypted data without revealing any information about the data itself. However, FHE ciphertexts include noise for security reasons, which increases during operations and can lead to decryption errors. This paper addresses the noise introduced during bootstrapping in Torus Fully Homomorphic Encryption (TFHE), particularly focusing on approximation errors during modulus switching and gadget decomposition. We propose a mean compensation technique that removes the mean term from the noise equations, achieving up to a twofold reduction in noise variance. This method can be combined with bootstrap key unrolling for further noise reduction. Mean compensation can reduce the error probability of a standard parameter set from $2^{-73.77}$ to $2^{-130.21}$, or allows the selection of more efficient parameters leading to a speedup of bootstrapping up to a factor 2.06×. Compared to the state-of-the-art rerandomization technique, mean compensation requires no additional key material, and offers a fixed workflow. Furthermore, it removes the dependency between the noise variance and the Hamming weight of the secret key.

Keywords: Fully homomorphic encryption · Noise analysis · Modulus switching · Gadget decomposition · Bootstrap key unrolling

1 Introduction

Fully Homomorphic Encryption (FHE) allows computations on encrypted data without revealing any information about the data. A prominent application scenario is cloud computing, where clients outsource workloads to potentially untrusted servers. Here, the client encrypts data before sending it to the server, which performs computations directly on the encrypted data and returns an encrypted result. Only the client can decrypt the final output, ensuring data security throughout the computation. Notable FHE schemes include BFV, BGV, CKKS, GBFV, TFHE, and FINAL [FV12, BGV12, CKKS17, GV25, CGGI16, BIP+22].

FHE ciphertexts include some noise for security reasons. This noise increases throughout the operations and eventually leads to decryption errors if it grows too large. To prevent these errors, the noise can be reset to a low value through a process called bootstrapping. Bootstrapping is a maintenance operation that refreshes ciphertexts and typically dominates the computation time.

Torus Fully Homomorphic Encryption (TFHE) [CGGI16] is a popular FHE scheme that specializes in fast bootstrapping of small ciphertexts and supports the homomorphic evaluation of non-linear functions using programmable bootstrapping. Its security is based on the Learning With Errors (LWE), Ring-LWE, and Module-LWE problems [Reg05]. Its parameters, such as the noise variance on input ciphertexts and the LWE dimension, determine the trade-off between security, computational cost, and the error probability of TFHE.

While some noise is introduced for security reasons, ciphertext approximations during bootstrapping increase the noise variance significantly. This unwanted noise is introduced

Licensed under Creative Commons License CC-BY 4.0.
Received: 2025-07-15 Accepted: 2025-09-15 Published: 2026-01-16

by approximation errors during modulus switching and gadget decomposition in both key switching and blind rotation, all necessary subroutines of bootstrapping. For instance, for a standard parameter set[1] in the TFHE-rs library [Zam22], these sources account for 89.5% of the total noise variance. Reducing the noise leads to a lower error probability and/or more efficient parameters.

Previous works have studied and reduced the noise on ciphertexts in TFHE. Chillotti et al. [CLOT21] developed a detailed noise analysis to describe the exact contribution of all individual noise sources. Noise reduction techniques were introduced by Ligier et al. [LOT25] and Bernard et al. [BJSW25b]. Ligier et al. suggest an alternative to bootstrapping that subtracts noise from a ciphertext. They extract a chosen number of bits of the noise by shifting it into the message space and bootstrapping. The resulting encryption of the noise is then subtracted from the original ciphertext to lower its noise. This approach can only evaluate the identity function and no longer allows programmable bootstrapping. Bernard et al. reduce the noise variance during modulus switching by rerandomizing the ciphertexts. Encryptions of zero are repeatedly added to the ciphertext until an encryption with a small enough approximation error is found. This procedure requires access to and the storage of additional ciphertexts and increases the running time due to memory fetches.

Our contributions: Several FHE operations introduce an approximation noise into the ciphertext during bootstrapping. This noise is not needed for security in the LWE problem but still accumulates to the total noise term, increasing the chance for decryption errors. As a consequence, we want to minimize these noise terms.

Looking at the noise contributions of these operations, we see that these include the mean of the secret distribution, μ_s. In this paper, we demonstrate how to remove these terms from the noise equations, achieving up to a twofold reduction in noise variance.

For example, for an LWE ciphertext $(a, b = a \cdot s + m + e)$, one can calculate the error by decrypting: $b - a \cdot s = m + e$. When we make an approximation error on the coefficients of a, this results in an approximated $\hat{a}$ and error u, so that $a = \hat{a} + u$. Decryption of the ciphertext $(\hat{a}, b)$ then gives:

$$\begin{aligned} b - \hat{a} \cdot s &= b - (a - u) \cdot s \\ &= m + e + u \cdot s. \end{aligned} \tag{1}$$

This results in an additional error $u \cdot s$, with variance $\sigma_u^2 \cdot (\sigma_s^2 + \mu_s^2)$ assuming that the approximation error has a mean value of zero.

Our compensation technique removes the contribution of the mean term μ_s, thereby halving the approximation error during modulus switching and gadget decomposition for both key switching and blind rotations [2]. This method comes with a negligible computational cost, reduces the error probability for a standard parameter set from $2^{-73.77}$ to $2^{-130.21}$, and removes the dependency between the Hamming weight of the secret key and the noise variance of the bootstrapped ciphertext, as it no longer depends on μ_s. Alternatively, by generating new parameter sets, we can exchange this decrease in error probability into performance improvements up to a factor 2.06×, for identical security and error probability requirements.

Furthermore, we look into combining mean compensation with bootstrap key unrolling, which also reduces the noise introduced during modulus switching. However, we show that a naive combination of both approaches yields worse results compared to mean compensation, and introduce a new combination technique to optimize the noise reduction. This allows

[1] PS1 in Appendix A

[2] We add a correction term to the body to offset the contribution of the mean term. When using boolean secrets, this halves the noise variance.

us to further reduce the error probability to $2^{-134.26}$ or $2^{-143.19}$ for an unrolling factor of two or three respectively.

Finally, we compare mean compensation with the rerandomization technique of Bernard et al. [BJSW25b], as they both remove the mean component of the modulus switching noise. Consequently, both approaches find similar noise reductions. However, our method does not require additional key material and offers a fixed workflow. Moreover, we combine them into a hybrid approach that offers the same noise reduction using fewer rerandomizations.

2 Preliminaries

2.1 Notations

For the rest of this work, we use the following notations. We use elements from the real torus $\mathbb{T} := \mathbb{R}/\mathbb{Z}$, the set $[0,1)$ of real numbers modulo 1. They are discretized onto $\mathbb{T}_q := q^{-1}\mathbb{Z}/\mathbb{Z}$, with q an integer and typically a power of two. The representatives of $\mathbb{T}_q$ are $\{0, \frac{1}{q}, ..., \frac{q-1}{q}\}$ and operate modulo 1. However, implementations typically multiply by q, to work over $\mathbb{Z}_q := \mathbb{Z}/q\mathbb{Z}$, the set of integers modulo q. Similarly, we define the plaintext space as $\mathbb{T}_p := p^{-1}\mathbb{Z}/\mathbb{Z}$ with representatives $\{0, \frac{1}{p}, ..., \frac{p-1}{p}\}$, and where p divides q. $\mathcal{R}_q := \mathbb{T}_q[X]/(X^N+1)$ and $\mathcal{R}_p := \mathbb{T}_p[X]/(X^N+1)$ denote polynomials modulo X^N+1 with N a power of two and where the coefficients are elements of $\mathbb{T}_q$ and $\mathbb{T}_p$ respectively. The set $\mathbb{B}$ is the integer subset $\{0,1\}$. Superscripts are used to denote vectors. For example, $\mathcal{R}_q^k$ denotes a vector with k entries from $\mathcal{R}_q$.

$\mathbb{E}[x]$ and $\mathrm{Var}(x)$ respectively denote the expected value and variance of a variable x. When possible, we abbreviate them to μ_x and σ_x^2. For a set S, $a \xleftarrow{\$} S$ denotes that a is sampled uniformly at random from S. Alternatively, for a distribution χ, $a \leftarrow \chi$ denotes that a is sampled according to the distribution of χ. In this work, we use $\hat{\chi}$, which is the gaussian distribution with mean $\mu = 0$ and variance $\sigma^2 = \sigma^2_{\mathrm{lwe}}$, discretized onto $\mathbb{T}_q$, and $\hat{\chi}_N$, which is the distribution of polynomials of degree N with coefficients from the gaussian distribution with mean $\mu = 0$ and variance $\sigma^2 = \sigma^2_{\mathrm{glwe}}$, discretized onto $\mathcal{R}_q$. The distribution $\mathcal{U}(\llbracket x,y \llbracket)$ denotes the discretized uniform distribution between x (inclusive) and y (exclusive), of elements from $\mathbb{T}_q$. For a real value x, $\lfloor x \rceil$ denotes the nearest integer to x, rounding down in case of a tie. For a value $x \in \mathbb{B}$, $P[x]$ denotes the probability that $x = 1$.

2.2 TFHE encryption

A Torus LWE (TLWE) ciphertext is constructed by combining three components: a secret vector $s \xleftarrow{\$} \mathbb{B}^n$, a mask $a \xleftarrow{\$} \mathbb{T}_q^n$, and a small error $e \leftarrow \hat{\chi}$. A message m from the plaintext space $\mathbb{T}_p$ can be encrypted as a tuple of the mask a and body b: $(a, b = a \cdot s + m + e) \in \mathbb{T}_q^n \times \mathbb{T}_q$. A Torus Generalized LWE (TGLWE) ciphertext is a generalization where all elements are polynomials. It uses $s \xleftarrow{\$} \left(\mathbb{B}[X]/(X^N+1)\right)^k$, $a \xleftarrow{\$} \mathcal{R}_q^k$, and $e \leftarrow \hat{\chi}_N$ to encrypt a message $m \in \mathcal{R}_p$ as $(a, b = a \cdot s + m + e) \in \mathcal{R}_q^k \times \mathcal{R}_q$.

Using the secret vector s, one can decrypt a TLWE (TGLWE) ciphertext back into a noisy version of the message m by computing $b - a \cdot s = m + e$. Rounding this result to the nearest element of $\mathbb{T}_p$ ($\mathcal{R}_p$) recovers the message. However, if the error e grows too large, we round to an incorrect element and find decryption errors. We are interested in the variance of the error term, σ_e^2, as this directly determines the error probability for a given plaintext space. We can compute the error on a ciphertext as $e = b - a \cdot s - m$.

T(G)LWE ciphertexts are additively homomorphic. When we encrypt m_1 and m_2 using the same secret vector s, we can add the mask and body element-wise to find an encryption of $m_1 + m_2$. Similarly, multiplication by a constant can be computed as a series of additions.

Moreover, TFHE defines an external product using a so-called TGGSW ciphertext, $\boxdot$: TGGSW $\times$ TGLWE $\rightarrow$ TGLWE. This external product computes a multiplication between a TGLWE ciphertext encrypting a message m_1, and a TGGSW ciphertext encrypting a message m_2, and results in a new TGLWE ciphertext encrypting the message $m_1 \cdot m_2$.

2.3 Bootstrapping in TFHE

The bootstrapping cycle in TFHE is build up of key switching, modulus switching, blind rotations, and sample extraction, as illustrated in Figure 1. Below, we focus on the first three steps, as they introduce approximation errors and are therefore relevant for this work. In Section 3, we will derive the noise equations for the approximation errors in detail for each of these steps. For a comprehensive overview of bootstrapping, we refer to the work by Chillotti et at. [CGGI18] and the work by Joye [Joy22]. For an overview of all noise contributions, we refer to the work by Chillotti et al. [CLOT21].

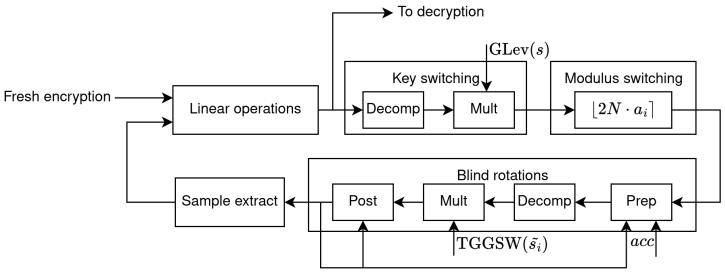

Figure 1: Overview of the subroutines making up bootstrapping

2.3.1 Key switching

Key switching modifies an input TLWE ciphertext $(a, b = \sum a \cdot s + m + e) \in \mathbb{T}_q^{kN} \times \mathbb{T}_q$ under encryption key s, to an output TLWE ciphertext $(\tilde{a}, \tilde{b} = \sum \tilde{a} \cdot \tilde{s} + m + \tilde{e}) \in \mathbb{T}_q^{n} \times \mathbb{T}_q$, encrypting the same message m using a different encryption key $\tilde{s}$. The main idea is to calculate $b - \sum a \cdot s$ homomorphically under the secret key $\tilde{s}$. This homomorphic multiplication $a \cdot s$ is performed in two steps: a gadget decomposition of a_i and a multiplication. Gadget decomposition approximates every element of the mask with l_{ks} integers of $\log_2(\beta_{\text{ks}})$ bits. The remaining term u_i is the approximation error:

$$a_i = a_i^{(1)} \cdot \frac{1}{\beta_{\text{ks}}} + a_i^{(2)} \cdot \frac{1}{\beta_{\text{ks}}^2} + \cdots + a_i^{(l_{\text{ks}})} \cdot \frac{1}{\beta_{\text{ks}}^{l_{\text{ks}}}} + u_i = \sum_{j=1}^{l_{\text{ks}}} (a_i^{(j)} \cdot \frac{1}{\beta_{\text{ks}}^{j}}) + u_i. \tag{2}$$

Afterwards, we multiply every decomposed mask element with its respective GLev(s_i), a list of leveled encryptions of the secret bit s_i under the target key $\tilde{s}$: LWE($\frac{s_i}{\beta_{\text{ks}}^{j}}$).

Both steps introduce noise in the output ciphertext. The underlying sources are the approximation errors u_i and the encryption noise of GLev(s) with variance σ_{lwe}^2.

2.3.2 Modulus switching

Modulus switching changes the modulus of the TLWE ciphertext from q_1 to q_2 by computing $\left\lfloor \frac{q_2}{q_1} \cdot \right\rceil$. For TFHE, we typically use $q_1 = 1$ and $q_2 = 2N$. Consequently, we compute $\tilde{a_i} = \lfloor 2N \cdot a_i \rceil$ and $\tilde{b} = \lfloor 2N \cdot b \rceil$ to represent all elements of the ciphertext with integers modulo $2N$. Implementations of TFHE typically multiply all torus elements with q to work over $\mathbb{Z}_q$ instead of $\mathbb{T}_q$. In that case, $q_1 = q$, resulting in $\tilde{a_i} = \left\lfloor \frac{2N \cdot a_i}{q} \right\rceil$ and $\tilde{b} = \left\lfloor \frac{2N \cdot b}{q} \right\rceil$.

The rounding operator introduces new approximation errors, $u_i = 2N \cdot a_i - \lfloor 2N \cdot a_i \rceil$ for each mask element and $u_b = 2N \cdot b - \lfloor 2N \cdot b \rceil$, increasing the noise variance of the ciphertext.

2.3.3 Blind rotation

The goal of the blind rotations is to homomorphically multiply a TGLWE ciphertext $acc_0 = (0,...,0,\sum_{i=0}^{N-1} f(i) \cdot X^i)$, the so-called accumulator, by $X^{-(b-a \cdot s)}$. This means that we rotate the accumulator with $-b+a \cdot s = -m-e$ slots. Therefore, this multiplication results in an encryption of a polynomial with $f(b-a \cdot s) = f(m+e)$ as its constant term. By choosing the coefficients of the body of acc_0 cleverly, we can simultaneously compute any function $g(x)$ on the encrypted message, as well as reset the noise. Specifically, we set the coefficients $f(i)$ to $g(m)$ where m is the closest element in the plaintext space to i. As a result, the encrypted constant term after the homomorphic multiplication, $f(m+e)$, equals an encryption of $g(m)$ as long as the error e is sufficiently small.

To multiply the accumulator with $X^{-(b-a \cdot s)}$, we iteratively compute

$$\begin{aligned} acc_i &= \mathrm{TGGSW}(s_i) \boxdot (acc_{i-1} \cdot X^{a_i} - acc_{i-1}) + acc_{i-1} \\ &= \begin{cases} acc_{i-1}, & \text{if } s_i = 0 \\ acc_{i-1} \cdot X^{a_i}, & \text{if } s_i = 1 \end{cases} \\ &= acc_{i-1} \cdot X^{a_i \cdot s_i}. \end{aligned} \tag{3}$$

Every iteration is called a blind rotation as we "rotate" (multiply) the accumulator by $a_i \cdot s_i$ slots "blindly" (without learning the value of s_i). Finally, we update the accumulator to $acc_n \cdot X^{-b}$ to complete the computation of $acc_0 \cdot X^{-(b-a \cdot s)}$.

In blind rotation, similarly to key switching, gadget decomposition is computed before multiplying with the encrypted key, $\mathrm{TGGSW}(s_i)$. Consequently, we find the same two types of errors: the first error arises from the approximation during gadget decomposition, resulting in an error polynomial U_i with the variance of its coefficients defined as σ_U^2. The second error originates from the encryption noise on the TGGSW encryptions, with variance σ_{glwe}^2.

2.3.4 Multiple function evaluation

It is possible to evaluate multiple functions at the same time using manyLUT [CLOT21]. When we want to compute two functions of one ciphertext, manyLUT computes the blind rotations (the costly part) once and extracts both results out of it. It exploits some of the redundancy in the accumulator, *acc*, to implement multiple functions. For example, to extract two functions, g_1 and g_2, they initialize the even coefficients of the accumulator with function evaluations of g_1, and the odd coefficients with function evaluations of g_2.

We can use manyLUT when the least significant bit of every element of the TLWE ciphertext after modulus switching is equal to zero. When this is the case, the blind rotations will always rotate the accumulator by an even number. This way, after computing all blind rotations, we have rotated the accumulator by an even number of slots. Therefore, the constant coefficient will be an encoding of $g_1(m)$, while the first order coefficient will be an encoding of $g_2(m)$. When we extract a sample of each, we evaluate two functions at approximately the cost of one regular programmable bootstrap.

The downside of the approach is the increased modulus switching noise. As we require the least significant bit of every element of the TLWE ciphertext to equal zero, we can compute modulus switching as $\tilde{a_i} = 2 \cdot \lfloor N \cdot a_i \rceil$ and $\tilde{b} = 2 \cdot \lfloor N \cdot b \rceil$. This means that the rounding error, u_i, can be twice as large. Therefore, we find that the variance of this error, σ_u^2, will be four times larger.

3 Mean compensation

Gadget decomposition and modulus switching introduce approximation errors into the ciphertext during bootstrapping as they compress the representation of each of the elements of the ciphertext. The formulas for the resulting noise variances include the mean of the secret, μ_s. The main idea of this paper is to remove this mean term, thereby halving the noise variance.

First, we illustrate how the mean of the secret appears in the noise contributions. Before approximation, the error on a ciphertext $(a,b=\sum_{i=1}^{n} a_i \cdot s_i + m + e)$ equals $e = b - \sum_{i=1}^{n} a_i \cdot s_i - m$. When we approximate the elements of the mask a_i with $\hat{a}_i$, we make an approximation error $u_i = a_i - \hat{a}_i$. The error on the approximated ciphertext $(\hat{a},b)$ equals:

$$\begin{aligned} b - \sum_{i=1}^{n} \hat{a}_i \cdot s_i - m &= b - \sum_{i=1}^{n} (a_i - u_i) \cdot s_i - m \\ &= e + \sum_{i=1}^{n} u_i \cdot s_i. \end{aligned} \tag{4}$$

The additional error term, $\sum_{i=1}^{n} u_i \cdot s_i$, increases the noise variance by $n \cdot (\sigma_u^2 \cdot (\sigma_s^2 + \mu_s^2) + \mu_u^2 \cdot \sigma_s^2)$, where σ_u^2 is the variance of the approximation error, and σ_s^2 and μ_s are the variance and mean of the distribution used to sample the secret s_i. Throughout this paper, for simplicity, we will ignore the mean term of the approximation errors[3]: $\mu_u \approx 0$.

The mean of the secret is not zero, as TFHE typically uses $s \xleftarrow{\$} \mathbb{B}^n$ with mean $\mu_s = \frac{1}{2}$ and standard deviation $\sigma_s = \frac{1}{2}$ for all ciphertexts. Whereas the input ciphertext for the blind rotations requires boolean secret keys, other key distributions may be used for the resulting TGLWE ciphertext. In that case, key switching will return to a ciphertext with a boolean secret key.

Since the approximation errors are known to the party computing the bootstrap, the error $u_i \cdot s_i$ can be compensated for by subtracting a correction term $u_i \cdot \mu_s$. One such correction term is too small to be represented with the compressed elements of the ciphertext, as this was the origin of the approximation error in the first place. However, it is possible to represent the sum of these correction terms, $\mu_s \cdot \sum u_i$.

Note that μ_s refers to the mean of the distribution from which the secret was sampled, and not the mean of the secret sample itself. Sharing the latter with the party computing the bootstrap would reveal the Hamming weight of the secret and therefore reduce security.

Now, we compensate for the approximation errors by adjusting the body b, into $b_{\text{comp}} = b - \mu_s \cdot \sum_{i=1}^{n} u_i$. Then, the error on the compensated ciphertext $(\hat{a},b_{\text{comp}})$ equals:

$$\begin{aligned} b_{\text{comp}} - \sum_{i=1}^{n} \hat{a}_i \cdot s_i - m &= b - \mu_s \cdot \sum_{i=1}^{n} u_i - \sum_{i=1}^{n} (a_i - u_i) \cdot s_i - m \\ &= e - \mu_s \cdot \sum_{i=1}^{n} u_i + \sum_{i=1}^{n} u_i \cdot s_i \\ &= e + \sum_{i=1}^{n} u_i \cdot (s_i - \mu_s). \end{aligned} \tag{5}$$

In this case, the additional error term, $\sum_{i=1}^{n} u_i \cdot (s_i - \mu_s)$, has a noise variance of $n \cdot \sigma_u^2 \cdot \sigma_s^2$ as the mean value of $s_i - \mu_s$ is zero. Moreover, the noise variance no longer depends on the

[3] In some implementations, $\mu_u = 0$ by performing techniques such as balanced rounding, but even if this is not the case, σ_u^2 is typically in the order of magnitude $N^{-2}/12 \approx 2^{-27}$ while μ_u^2 is approximately $(2q)^{-2} \approx 2^{-130}$ and is thus negligible. It is trivial to adapt the results of this paper to non-zero μ_u if one wishes.

Hamming weight of the secret. For example, for a uniform distributed $s_i \in \{0,1\}$, the error was originally increased by u_i or was unchanged depending on the secret bit, resulting in a noise variance that depends on the Hamming weight of the secret. Using mean compensation, the error increases by $u_i \cdot \frac{1}{2}$ or $u_i \cdot \frac{-1}{2}$ depending on the secret bit. As both possible error terms are now identically distributed, the dependency on the Hamming weight is avoided.

In the remainder of this section, we apply this general idea to the approximation errors during modulus switching and gadget decomposition.

3.1 Key switching decomposition

Gadget decomposition during key switching introduces an approximation error for each mask element a_i. First, we derive the noise variance of this error without compensation as a baseline. Then, we compensate the mean term and highlight the improvement in noise variance.

To find the baseline noise variance, we study the error made during decomposition. Gadget decomposition operates on a TLWE encryption of m, $(a,b=\sum_{i=1}^{kN} a_i s_i + m + e_{\text{in}}) \in \mathbb{T}_q^{kN} \times \mathbb{T}_q$. It splits every a_i into l_{ks} terms of $\log_2(\beta_{\text{ks}})$ bits:

$$a_i = \sum_{j=1}^{l_{\text{ks}}} (a_i^{(j)} \cdot \frac{1}{\beta_{\text{ks}}^j}) + u_i. \tag{6}$$

An approximation error, u_i, is made as we now represent each mask element with fewer bits. This error is distributed following:

$$\mathcal{U}\left(\left[\!\left[\frac{-1}{2\beta_{\text{ks}}^{l_{\text{ks}}}}, \frac{1}{2\beta_{\text{ks}}^{l_{\text{ks}}}} \right]\!\right] \right). \tag{7}$$

This uniform distribution has a variance of $\sigma_u^2 \approx \frac{1}{12 \cdot \beta_{\text{ks}}^{2 \cdot l_{\text{ks}}}}$ and approximate mean of $\mu_u \approx 0$.

To quantify the total increase in noise variance, we can recompose the mask after approximation and compute decryption. When we remove the message m and the input noise e_{in}, we find the new error term:

$$\begin{aligned} e_{\text{ks-decomp}} &= b - \sum_{i=1}^{kN} \Big(\sum_{j=1}^{l_{\text{ks}}} (a_i^{(j)} \cdot \frac{1}{\beta_{\text{ks}}^j}) \Big) \cdot s_i - m - e_{\text{in}} \\ &= b - \sum_{i=1}^{kN} (a_i - u_i) \cdot s_i - m - e_{\text{in}} \\ &= \sum_{i=1}^{kN} u_i \cdot s_i. \end{aligned} \tag{8}$$

with the noise variance $\sigma^2_{\text{ks-decomp}} = kN \cdot \sigma_u^2 \cdot (\sigma_s^2 + \mu_s^2)$.

Now, we compensate the body b to remove the term μ_s^2 from this equation. More specifically, we use $b_{\text{comp}} = b - \left\lfloor (\mu_s \cdot \sum_{i=1}^{kN} u_i) \cdot q \right\rceil / q$. In that case, the error equals:

$$\begin{aligned} e_{\text{ks-decomp}} &= b_{\text{comp}} - \sum_{i=1}^{kN} \Big(\sum_{j=1}^{l_{\text{ks}}} (a_i^{(j)} \cdot \frac{1}{\beta_{\text{ks}}^j}) \Big) \cdot s_i - m - e_{\text{in}} \\ &\approx b - \mu_s \cdot \sum_{i=1}^{kN} u_i - \sum_{i=1}^{kN} (a_i - u_i) \cdot s_i - m - e_{\text{in}} \\ &\approx \sum_{i=1}^{kN} u_i \cdot (s_i - \mu_s). \end{aligned} \tag{9}$$

with the variance $\sigma^2_{\text{ks-decomp}} = kN \cdot \sigma_u^2 \cdot \sigma_s^2$.

The compensation is described in algorithm 1. We adjust the input body of key switching, but do not alter the key switching procedure itself. As we have to compute the rounding anyway, we compute kN more additions and $kN+1$ more subtractions compared to the naive method, which is negligible compared to the bootstrapping cost. These operations accumulate the rounding errors into a variable c and compensate the body b based on the value of c. Therefore, the noise variance due to the rounding operation is halved at negligible computational cost.

Algorithm 1: Mean compensation for key switching decomposition.

Input: Input ciphertext: $(a,b) \in \mathbb{T}_q^{kN} \times \mathbb{T}_q$
Output: Output ciphertext: $(a,b_{\text{comp}}) \in \mathbb{T}_q^{kN} \times \mathbb{T}_q$

$c = 0$;
for $i = 1$ **to** kN **do**
  $A_i = \left\lfloor \beta_{\text{ks}}^{l_{\text{ks}}} \cdot a_i \right\rceil$;
  $c \mathrel{+}= a_i - \frac{A_i}{\beta_{\text{ks}}^{l_{\text{ks}}}}$;
end
$b_{\text{comp}} = b - \lfloor (c \cdot \mu_s) \cdot q \rceil / q$;
return (a, b_{comp})

3.2 Modulus switching

We can apply the same technique to the rounding error during modulus switching. Once again, we first compute the baseline noise variance increase and then compensate the mean term. This time, we also approximate the body b, which introduces an error $u_b = b - \frac{\lfloor 2N \cdot b \rceil}{2N}$, in addition to $u_i = a_i - \frac{\lfloor 2N \cdot a_i \rceil}{2N}$. As both terms originate from rounding, we know that they are distributed following:

$$\mathcal{U}\left(\left[\!\left[\frac{-1}{4N}, \frac{1}{4N} \right]\!\right] \right). \tag{10}$$

This uniform distribution has a variance of $\sigma_u^2 \approx \frac{1}{12 \cdot (2N)^2}$ and approximate mean of $\mu_u \approx 0$.

To find the baseline error, we compute decryption on the rounded ciphertext and remove the message m and input noise e_{in}:

$$\begin{aligned} e_{\text{ms}} &= \frac{\lfloor 2N \cdot b \rceil}{2N} - \sum_{i=1}^{n} \frac{\lfloor 2N \cdot a_i \rceil}{2N} \cdot s_i - m - e_{\text{in}} \\ &= (b - u_b) - \sum_{i=1}^{n} (a_i - u_i) \cdot s_i - m - e_{\text{in}} \\ &= -u_b + \sum_{i=1}^{n} u_i \cdot s_i. \end{aligned} \tag{11}$$

This increases the noise variance by $\sigma^2_{\text{ms}} = \sigma_u^2 \cdot (n \cdot (\sigma_s^2 + \mu_s^2) + 1)$.

Now, we compensate the mean term by adjusting the body b to $b_{\text{comp}} = b - \mu_s \cdot \sum_{i=1}^{n} u_i$. Note that rounding this compensated body b_{comp} results in a new approximation error $u_{\hat{b}}$, distributed similarly to u_b. After this compensation, we can recompute the introduced error

during modulus switching as:

$$\begin{aligned} e_{\text{ms}} &= \frac{\lfloor 2N \cdot b_{\text{comp}} \rceil}{2N} - \sum_{i=1}^{n} \frac{\lfloor 2N \cdot a_i \rceil}{2N} \cdot s_i - m - e_{\text{in}} \\ &= (b - \mu_s \cdot \sum_{i=1}^{n} u_i - u_{\hat{b}}) - \sum_{i=1}^{n} (a_i - u_i) \cdot s_i - m - e_{\text{in}} \\ &= -u_{\hat{b}} + \sum_{i=1}^{n} u_i \cdot (s_i - \mu_s) \end{aligned} \tag{12}$$

with the noise variance $\sigma^2_{\text{ms}} = \sigma^2_u \cdot (n \cdot \sigma^2_s + 1)$.

Our new modulus switching method is described in algorithm 2. It computes the modulus switch for each coefficient of the mask and accumulates the total rounding error in c. This c is then compensated before rounding the body. The mean compensated modulus switching method takes n more additions and $n+1$ more subtractions compared to the naive method.

Algorithm 2: Mean Compensated Modulus Switching.

Input: Input ciphertext: $(a,b) \in \mathbb{T}_q^n \times \mathbb{T}_q$
Output: Output ciphertext: $(\tilde{a},\tilde{b}) \in \mathbb{Z}_{2N}^n \times \mathbb{Z}_{2N}$

```
c = 0 ;
for i = 1 to n do
    ã_i = ⌊2N · a_i⌉ ;
    c += a_i − ã_i / 2N ;
end
b_comp = b − c · μ_s ;
b̃ = ⌊2N · b_comp⌉ ;
return (ã, b̃)
```

3.3 Blind rotation decomposition

The gadget decomposition during blind rotations can also exploit the same compensation technique. However, we now compute on polynomials, TGLWE ciphertexts, in contrast to the TLWE ciphertexts from the previous applications. We are interested in the noise variance on the coefficients of the decrypted polynomial separately, as sample extraction selects a single coefficient. Once more, we first study the baseline noise variance increase and then compensate the mean term.

Every blind rotation computes $acc_i = \text{TGGSW}(s_i) \boxdot (acc_{i-1} \cdot X^{a_i} - acc_{i-1}) + acc_{i-1} = acc_{i-1} \cdot X^{a_i s_i}$ by using the gadget decomposition of $(A,B) = (acc_{i-1} \cdot X^{a_i} - acc_{i-1}) \in \mathcal{R}_q^k \times \mathcal{R}_q$. The decomposition is defined as:

$$A_i = \sum_{j=1}^{l_{\text{br}}} \left(A_i^{(j)} \cdot \frac{1}{\beta_{\text{br}}^j} \right) + U_i \tag{13}$$

$$B = \sum_{j=1}^{l_{\text{br}}} \left(B^{(j)} \cdot \frac{1}{\beta_{\text{br}}^j} \right) + U_B. \tag{14}$$

where every coefficient, h, of every polynomial of rounding errors, U_i and U_B, is distributed following:

$$\mathcal{U}\left(\left[\!\left[\frac{-1}{2\beta_{\text{br}}^{l_{\text{br}}}}, \frac{1}{2\beta_{\text{br}}^{l_{\text{br}}}} \right]\!\right] \right). \tag{15}$$

This uniform distribution has a variance of $\sigma_U^2 \approx \frac{1}{12 \cdot \beta_{\text{br}}^{2 \cdot l_{\text{br}}}}$ and approximate mean of $\mu_U \approx 0$.

Similarly to before, we quantify the baseline error, $E_{\text{br-decomp}}$, as:

$$\begin{aligned} E_{\text{br-decomp}} &= \sum_{j=1}^{l_{\text{br}}} (B^{(j)} \cdot \frac{1}{\beta_{\text{br}}^j}) - \sum_{i=1}^{k} \left(\sum_{j=1}^{l_{\text{br}}} (A_i^{(j)} \cdot \frac{1}{\beta_{\text{br}}^j}) \right) \cdot S_i - M - E_{\text{in}} \\ &= -U_B + \sum_{i=1}^{k} U_i \cdot S_i. \end{aligned} \tag{16}$$

with the variance on each of the elements of the polynomial equal to $\sigma_U^2 \cdot (kN \cdot (\sigma_S^2 + \mu_S^2) + 1)$.

This time, the compensation is more expensive to compute because we work with polynomials. Again, we adjust the body B based on rounding errors in the gadget decomposition of A_i. More specifically, we use $B_{\text{comp}} = B - \mu_S I^N \cdot \sum_{i=1}^{k} U_i$, with $I^N = 1 + X + X^2 + \cdots + X^{N-1}$. Similarly to the compensated modulus switching, Equation 12, the introduced error becomes:

$$E_{\text{br-decomp}} = -U_B + \sum_{i=1}^{k} U_i \cdot (S_i - \mu_S I^N). \tag{17}$$

with the variance on each of the elements of the polynomial equal to $\sigma_U^2 \cdot (kN \cdot \sigma_S^2 + 1)$.

The calculation of the compensated body polynomial B_{comp} uses the result of the multiplication between the polynomials $\mu_S I^N$ and $\sum_{i=1}^{k} U_i$. Naively, this is computed using a Fast Fourier Transformation of size N, requiring $\mathcal{O}(N \cdot \log(N))$ operations. However, due to the structure of $\mu_S I^N$, we found that we can calculate this product more efficiently. Specifically, note that we can compute the coefficients of $C = \mu_S I^N \cdot \sum_{i=1}^{k} U_i$ as

$$\begin{bmatrix} C[0] \\ C[1] \\ \vdots \\ C[N-1] \end{bmatrix} = \mu_S \cdot \begin{bmatrix} 1 & -1 & -1 & \dots & -1 \\ 1 & 1 & -1 & \dots & -1 \\ 1 & 1 & 1 & \dots & -1 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & 1 & \dots & 1 \end{bmatrix} \cdot \begin{bmatrix} \sum_{i=1}^{k} U_i[0] \\ \sum_{i=1}^{k} U_i[1] \\ \vdots \\ \sum_{i=1}^{k} U_i[N-1] \end{bmatrix}. \tag{18}$$

We observe that the only difference between every pair of adjacent rows of the matrix is the sign of one of the elements. Therefore, we can easily compute every next output coefficient using the previous one. This structure allows us to calculate the polynomial multiplication in $\mathcal{O}(N)$ operations.

The full computation of the compensated body is described in algorithm 3. First, we compute $U = \sum_{i=1}^{k} U_i$ in kN additions and kN subtractions (lines 2-4). Then, the last coefficient of the compensation polynomial ($C[N-1]$) can be found by adding all coefficients of U together in $N-1$ additions (line 5). When we subtract $U[N-1]$ twice from this result, we find $C[N-2]$. Next, we subtract $U[N-2]$ twice to find $C[N-3]$. Continuing this process, we can compute all other coefficients of C with a total of $N-1$ subtractions (line 8). Subtracting this polynomial from the body B requires an additional N operations (lines 7 and 10).

Algorithm 3: Mean compensation for blind rotation decomposition.

Input: Input ciphertext: $(A,B) \in \mathcal{R}_q^k \times \mathcal{R}_q$
Output: Output ciphertext: $(A,B_{\text{comp}}) \in \mathcal{R}_q^k \times \mathcal{R}_q$

$U = 0 \in \mathcal{R}_q$;
for $i = 1$ **to** k **do**
  $U \mathrel{+}= A_i - \frac{\left\lfloor \beta_{\text{br}}^{l_{\text{br}}} \cdot A_i \right\rceil}{\beta_{\text{br}}^{l_{\text{br}}}}$;
end
$C[N-1] = \mu_S \cdot \sum_{h=0}^{N-1} U[h]$;
for $h = N-1$ **to** 1 **do**
  $B_{\text{comp}}[h] = B[h] - C[h]$;
  $C[h-1] = C[h] - 2\mu_S \cdot U[h]$;
end
$B_{\text{comp}}[0] = B[0] - C[0]$;
return (A, B_{comp})

Mean compensation for blind rotation decomposition is computationally significantly more expensive than mean compensation in modulus switching or key switching decomposition as we compute on polynomials. Specifically, we use approximately $(2k+3) \cdot N$ operations during each of the n blind rotates.

We lowered the gadget decomposition noise during a single blind rotation. However, this gadget decomposition is used to compute $\text{TGGSW}(s_i) \boxdot (acc_{i-1} \cdot X^{a_i} - acc_{i-1}) + acc_{i-1}$, with a total increase in noise variance due to gadget decomposition of

$$\sigma^2_{\text{br-decomp}} = \sigma^2_U \cdot (kN \cdot \sigma^2_S + 1) \cdot (\sigma^2_s + \mu^2_s) \cdot n \tag{19}$$

over the n rotations. σ^2_s and μ_s are respectively the variance and mean of the encrypted secret s_i.

Naively, one would expect that this mean term can also be removed using our mean compensation technique. However, in this case, the error term (an encryption of $-U_B + \sum_{i=1}^{k} U_i \cdot S_i$) is homomorphically multiplied by the secret bit (an encryption of s_i). As the error term is encrypted, we cannot compensate its mean term as it is unknown to the party computing the bootstrap.

4 Bootstrap key unrolling

Bootstrap Key Unrolling (BKU) executes multiple classical blind rotations with one external product. BKU was introduced by Zhou et al. [ZYL+18] and improved by Bourse et al. [BMMP18]. This technique also impacts the noise variance introduced during modulus switching by postponing the rounding. In this section, we demonstrate that while this technique does not trivially combine with mean compensation, careful design can further reduce the noise variance.

First, we introduce BKU as discussed in [BMMP18]. Instead of sequentially computing

$$acc_i = \text{TGGSW}(s_i) \boxdot (acc_{i-1} \cdot X^{\lfloor 2N \cdot a_i \rceil} - acc_{i-1}) + acc_{i-1} \tag{20}$$

$$acc_{i+1} = \text{TGGSW}(s_{i+1}) \boxdot (acc_i \cdot X^{\lfloor 2N \cdot a_{i+1} \rceil} - acc_i) + acc_i, \tag{21}$$

one would compute

$$acc_{i+1} = \begin{bmatrix} & (X^{\lfloor 2N\cdot a_i\rceil+\lfloor 2N\cdot a_{i+1})\rceil}-1)\cdot \mathrm{TGGSW}(s_i\cdot s_{i+1}) \\ + & (X^{\lfloor 2N\cdot a_i\rceil}-1)\cdot \mathrm{TGGSW}(s_i\cdot(1-s_{i+1})) \\ + & (X^{\lfloor 2N\cdot a_{i+1}\rceil}-1)\cdot \mathrm{TGGSW}((1-s_i)\cdot s_{i+1}) \\ + & 1 \end{bmatrix} \boxdot acc_{i-1} \tag{22}$$

using encryptions of $s_i\cdot s_{i+1}$, $s_i\cdot(1-s_{i+1})$, and $(1-s_i)\cdot s_{i+1}$.

The primary benefit of this approach is the reduction in the number of external products required. In the previous example, we demonstrated BKU with a factor two as we merged two original blind rotations. Consequently, we compute only one external product instead of two.

However, the disadvantage is the exponential increase in usage of bootstrapping keys. For each pair of blind rotations, we originally used two bootstrapping keys. With BKU with a factor of two, we require three bootstrapping keys. BKU with a factor of three already requires seven keys. Therefore, Jiang et al. [JLJ22] and Ye et al. [YKP22] indicate to keep the unrolling factor limited to two.

Finally, the noise on the output ciphertext due to modulus switching (the rounding in the exponents) can be reduced when using BKU. The remainder of this section quantifies this reduction and combines it with our new compensation technique.

4.1 Noise implications

We compare the noise variance increase during modulus switching for different conditions. First, we look at BKU with normal rounding and show that this results in the usual noise increase without BKU. Then, we combine this with mean compensation to find the expected halving of the noise. Afterwards, we introduce a known technique that we will call late rounding, which reduces the noise variance [ZYL+18, BMMP18, Zam22]. Next, we combine this technique naively with mean compensation and find worse results compared to the combination between normal rounding and mean compensation. Finally, we optimize the combination to find an even lower noise variance.

Normal rounding When using BKU, the modulus switching error will be identical to the one made with classical modulus switching. We update the accumulator acc iteratively as:

$$acc_{i+1} = \begin{cases} acc_{i-1}X^{\lfloor 2N\cdot a_i\rceil+\lfloor 2N\cdot a_{i+1}\rceil}, & \text{if } s_i=1 \;\&\; s_{i+1}=1 \\ acc_{i-1}X^{\lfloor 2N\cdot a_i\rceil}, & \text{if } s_i=1 \;\&\; s_{i+1}=0 \\ acc_{i-1}X^{\lfloor 2N\cdot a_{i+1}\rceil}, & \text{if } s_i=0 \;\&\; s_{i+1}=1 \\ acc_{i-1}, & \text{if } s_i=0 \;\&\; s_{i+1}=0. \end{cases} \tag{23}$$

During each iteration, we introduce either both rounding errors, one of the two, or neither of them, depending on the values of s_i and s_{i+1}:

$$\begin{aligned} e_{\mathrm{ms},i} &= (u_i+u_{i+1})\cdot s_i\cdot s_{i+1}+u_i\cdot s_i\cdot(1-s_{i+1})+u_{i+1}\cdot(1-s_i)\cdot s_{i+1} \\ &= u_i\cdot s_i+u_{i+1}\cdot s_{i+1} \end{aligned} \tag{24}$$

with a variance of $\sigma^2_{\mathrm{ms},i} = 2\cdot\sigma_u^2\cdot(\sigma_s^2+\mu_s^2)$.

This noise term is added for every blind rotation. As we handle two elements of the secret in every blind rotation and the secret contains n elements, this noise term is added $n/2$ times. The rounding of the body also introduces an additional error, u_b. Therefore, we find a total noise variance during modulus switching of:

$$\begin{aligned} \sigma^2_{\mathrm{ms}} &= \frac{n}{2}\cdot\sigma^2_{ms,i}+\sigma_u^2 \\ &= n\cdot\sigma_u^2\cdot(\sigma_s^2+\mu_s^2)+\sigma_u^2. \end{aligned} \tag{25}$$

We find that both the variance equation and the underlying error term are identical to classical modulus switching without BKU. As a consequence, we can combine BKU with mean compensation to halve the noise variance down to:

$$\sigma^2_{\text{ms}} = n \cdot \sigma_u^2 \cdot \sigma_s^2 + \sigma_u^2. \tag{26}$$

Late rounding A simple and commonly used approach to reduce the modulus switching noise is what we will call late rounding. Previously, we rounded $\lfloor 2N \cdot a_i \rceil$ and $\lfloor 2N \cdot a_{i+1} \rceil$ separately, resulting in two new rounding errors on acc_{i+1}. Instead, we can postpone rounding and multiply TGGSW$(s_i \cdot s_{i+1})$ by $X^{\lfloor 2N \cdot a_i + 2N \cdot a_{i+1} \rceil} - 1$ in Equation 22 to compute:

$$acc_{i+1} = \begin{cases} acc_{i-1} X^{\lfloor 2N \cdot a_i + 2N \cdot a_{i+1} \rceil}, & \text{if } s_i = 1 \ \& \ s_{i+1} = 1 \\ acc_{i-1} X^{\lfloor 2N \cdot a_i \rceil}, & \text{if } s_i = 1 \ \& \ s_{i+1} = 0 \\ acc_{i-1} X^{\lfloor 2N \cdot a_{i+1} \rceil}, & \text{if } s_i = 0 \ \& \ s_{i+1} = 1 \\ acc_{i-1}, & \text{if } s_i = 0 \ \& \ s_{i+1} = 0. \end{cases} \tag{27}$$

In this case, we introduce a single rounding error when at least one of the secret bits is set to one. When they are both set to zero, we introduce no rounding error. As a result, the error equals

$$e_{\text{ms},i} = u_{i;i+1} \cdot s_i \cdot s_{i+1} + u_i \cdot s_i \cdot (1 - s_{i+1}) + u_{i+1} \cdot (1 - s_i) \cdot s_{i+1} \tag{28}$$

with $u_{i;i+1} = 2N \cdot a_i + 2N \cdot a_{i+1} - \lfloor 2N \cdot a_i + 2N \cdot a_{i+1} \rceil$ a uniform rounding error similar to u_i and u_{i+1}. This leads to an increase in noise variance during each iteration of:

$$\sigma^2_{\text{ms},i} = \sigma_u^2 \cdot (\sigma_s^2 + \mu_s^2)^2 + 2 \cdot \sigma_u \cdot (\sigma_s^2 + \mu_s^2) \cdot (\sigma_s^2 + (1 - \mu_s)^2). \tag{29}$$

Combined over all iterations and taking the body into account, this becomes:

$$\sigma^2_{\text{ms}} = n \cdot \sigma_u^2 \cdot \frac{(\sigma_s^2 + \mu_s^2)^2 + 2 \cdot (\sigma_s^2 + \mu_s^2) \cdot (\sigma_s^2 + (1 - \mu_s)^2)}{2} + \sigma_u^2. \tag{30}$$

For $s \xleftarrow{\$} \mathbb{B}^n$, this corresponds to a 25% reduction in modulus switching noise compared to normal rounding. This reduction in noise variance is smaller than what we can achieve when combining normal rounding with mean compensation (50% reduction).

Mean compensation and late rounding Next, we analyze what happens to the noise when we naively combine mean compensation and late rounding. In this scenario, we use the compensated body $b_{\text{comp}} = b - \mu_s \cdot \sum_{i=1}^{n} u_i$ as well as the late rounding strategy. As a result, the compensation term now appears in the error of a single iteration:

$$e_{\text{ms},i} = u_{i;i+1} \cdot s_i \cdot s_{i+1} + u_i \cdot s_i \cdot (1 - s_{i+1}) + u_{i+1} \cdot (1 - s_i) \cdot s_{i+1} - (u_i + u_{i+1}) \cdot \mu_s. \tag{31}$$

At this point, the dependency between u_i or u_{i+1}, and $u_{i;i+1}$ complicates the noise variance calculation. When both secret bits are set to one, $u_{i;i+1}$ contributes to the error, in addition to u_i and u_{i+1} through the compensation term. However, we cannot simply add the noise variances separately as they are dependent. To simplify our analysis, we define a new variable $T_i = u_i + u_{i+1} - u_{i;i+1}$, which shows whether late rounding changes the rounding. When $|u_i + u_{i+1}| > 1/2$, late rounding will round to a different value and $u_{i;i+1} \neq u_i + u_{i+1}$. The difference is captured by T_i:

$$T_i = \begin{cases} 1 & \text{if } u_i + u_{i+1} > 1/2 \\ -1 & \text{if } u_i + u_{i+1} < -1/2 \\ 0 & \text{otherwise.} \end{cases} \tag{32}$$

Using this piecewise function, the resulting error is written as:

$$\begin{aligned} e_{\mathrm{ms},i} = &(u_i + u_{i+1} - T_i)\cdot s_i \cdot s_{i+1} + u_i \cdot s_i \cdot (1 - s_{i+1}) \\ &+ u_{i+1}\cdot(1-s_i)\cdot s_{i+1} - (u_i + u_{i+1})\cdot \mu_s \end{aligned} \tag{33}$$

which simplifies to:

$$e_{\mathrm{ms},i} = u_i \cdot (s_i - \mu_s) + u_{i+1}\cdot(s_{i+1} - \mu_s) - T_i \cdot s_i \cdot s_{i+1}. \tag{34}$$

We compute the variance of this error as:

$$\sigma^2_{\mathrm{ms},i} = \sum_{s_i} P[s_i] \cdot \sum_{s_{i+1}} P[s_{i+1}] \cdot \int_{u_i=-1/2}^{1/2} \int_{u_{i+1}=-1/2}^{1/2} e^2_{\mathrm{ms},i} \cdot du_i \cdot du_{i+1}. \tag{35}$$

When assuming $s \xleftarrow{\$} \mathbb{B}^n$, this equals:

$$\sigma^2_{\mathrm{ms},i} = \sigma^2_u \cdot \frac{3}{4}. \tag{36}$$

This is a larger variance than normal rounding combined with mean compensation. To pinpoint the problem, we focus on the case where $|u_i + u_{i+1}| > 1/2$. In that case, we find a lower rounding error through late rounding. However, note that the sign of this rounding error, $u_{i;i+1}$, is opposite to the sign of $u_i + u_{i+1}$. Therefore, when we compensate the expected rounding errors u_i and u_{i+1} by subtracting $(u_i + u_{i+1})\cdot\mu_s$ from the body, we increase the absolute value of the rounding error. This is the underlying reason why mean compensation should not be combined naively with late rounding.

Optimal solution We step away from the naive combination between late rounding and mean compensation, and instead optimize mean compensation while taking late rounding into account. In this section, we first parametrize the relevant equations with $\boldsymbol{c_u}$, $\boldsymbol{c_T}$, and $\boldsymbol{\tau}$, and then solve for minimal noise variance. We use the following compensation term:

$$b_{\mathrm{comp}} = b - \boldsymbol{c_u} \cdot \sum_{i=1}^{n} u_i + \boldsymbol{c_T} \cdot \sum_{i=1}^{n/2} T_{2i-1} \tag{37}$$

with T_i defined using a parametrized threshold $\boldsymbol{\tau}$:

$$T_i = \begin{cases} 1 & \text{if } u_i + u_{i+1} > \boldsymbol{\tau} \\ -1 & \text{if } u_i + u_{i+1} < -\boldsymbol{\tau} \\ 0 & \text{otherwise.} \end{cases} \tag{38}$$

When combined with late rounding, we update the accumulator as:

$$\begin{aligned} acc_{i+1} = &\left(\begin{bmatrix} & (X^{\lfloor 2N\cdot a_i \rceil + \lfloor 2N \cdot a_{i+1}) \rceil + T_i} - 1)\cdot \mathrm{TGGSW}(s_i \cdot s_{i+1}) \\ + & (X^{\lfloor 2N\cdot a_i \rceil} - 1)\cdot \mathrm{TGGSW}(s_i \cdot (1 - s_{i+1})) \\ + & (X^{\lfloor 2N\cdot a_{i+1} \rceil} - 1)\cdot \mathrm{TGGSW}((1-s_i)\cdot s_{i+1}) \\ + & 1 \end{bmatrix} \boxdot acc_{i-1} \right) \\ &\cdot X^{\boldsymbol{c_u}\cdot u_i + \boldsymbol{c_u}\cdot u_{i+1} - \boldsymbol{c_T}\cdot T_i}. \end{aligned} \tag{39}$$

This results in an error term due to modulus switching of:

$$e_{\mathrm{ms},i} = u_i \cdot (s_i - \boldsymbol{c_u}) + u_{i+1}\cdot(s_{i+1} - \boldsymbol{c_u}) - T_i \cdot (s_i \cdot s_{i+1} - \boldsymbol{c_T}). \tag{40}$$

Table 1: Variance of modulus switching noise using bootstrap key unrolling and the suggested mean compensation technique. The results are normalized to the normal modulus switching procedure.

Bootstrap key unrolling factor	1	2	3
Normal	1	1	1
Mean compensation [ours]	0.5	0.5	0.5
Late rounding	/	0.75	0.58
Late rounding + mean compensation	/	0.75	0.84
Optimal [ours]	/	0.48	0.44

Next, we compute the variance based on this error as in Equation 35 and minimize the noise variance $\sigma^2_{\text{ms},i}$ to find the optimal values of the parameters $\boldsymbol{c_u}$,$\boldsymbol{c_T}$, and $\boldsymbol{\tau}$. For $s \xleftarrow{\$} \mathbb{B}^n$, we find:

$$\boldsymbol{c_u} = 1/2 (= \mu_s) \tag{41}$$
$$\boldsymbol{c_T} = 1/4 (= \mu_s^2) \tag{42}$$
$$\boldsymbol{\tau} = 3/4. \tag{43}$$

These values align with expectations, as they minimize the mean component in the secret terms. That is, these are the terms that reduce the error term to

$$e_{\text{ms},i} = u_i \cdot (s_i - \mu_s) + u_{i+1} \cdot (s_{i+1} - \mu_s) - T_i \cdot (s_i \cdot s_{i+1} - \mu_s^2). \tag{44}$$

Summary All previous combinations between BKU and mean compensation are summarized in Table 1. This table also includes the noise implications in the case of a BKU factor of three, that were found with similar calculations.

5 Theoretical analysis

In this section, we compare our work with the rerandomization technique by Bernard et al. [BJSW25b] and develop a hybrid approach. Afterwards, we answer a set of open research questions formulated during a recent presentation by Joye [BJSW25a].

5.1 Comparison with [BJSW25b]

Similarly to this work, Bernard et al. [BJSW25b] suggest an approach to reduce modulus switching noise. Similar to our approach, they also exploit the fact that the approximation errors are known at runtime. This allows computing the expected noise variance based on the approximation errors. By repeatedly adding encryptions of zero, they rerandomize the coefficients of the ciphertext until the expected modulus switching noise is lower than a predetermined threshold.

We simulated both approaches in Python[4] to measure the remaining noise variances. To analyze the nature of the noise reduction, we also split the total modulus switching error $-u_b + \sum u_i \cdot s_i$ into an mean component, a variance component, and a body rounding

[4]Scripts and implementation available at https://github.com/KULeuven-COSIC/Dont_be_mean

Table 2: Measurement results of the remaining modulus switching noise variance after applying mean compensation and/or rerandomizing [BJSW25b]. The variance and mean components are reported separately to illustrate that both techniques mainly reduce the mean component. The hybrid approach requires fewer rerandomizations compared to the approach from [BJSW25b] to achieve the same noise reduction.

		Variance compon. + body rounding		Mean compon.		Total noise	
	# rerand.	Measured	Rel.	Measured	Rel.	Measured	Rel.
Baseline	/	$1.08\cdot10^{-6}$	100%	$1.07\cdot10^{-6}$	100%	$2.14\cdot10^{-6}$	100%
[BJSW25b]	15.3	$1.04\cdot10^{-6}$	96%	$2.72\cdot10^{-8}$	3%	$1.06\cdot10^{-6}$	49%
Mean comp.	/	$1.08\cdot10^{-6}$	100%	0	0%	$1.06\cdot10^{-6}$	50%
Hybrid	1.5	$1.05\cdot10^{-6}$	97%	0	0%	$1.05\cdot10^{-6}$	49%

component based on the law of total variance and assuming $\mathbb{E}[u_i]=0$:

$$\begin{aligned}\mathrm{Var}\left(-u_b+\sum u_i\cdot s_i\right) &= \mathrm{Var}\left(\mathbb{E}\left[\sum u_i\cdot s_i|u_i\right]\right)+\mathbb{E}\left[\mathrm{Var}\left(\sum u_i\cdot s_i|u_i\right)\right]+\mathrm{Var}(u_b)\\ &= \mathrm{Var}\left(\sum u_i\cdot\mathbb{E}[s_i]\right)+\mathbb{E}\left[\sum u_i^2\cdot\mathrm{Var}(s_i)\right]+\mathrm{Var}(u_b)\\ &= \underbrace{\mathbb{E}\left[\left(\sum u_i\cdot\mathbb{E}[s_i]\right)^2\right]}_{\text{Mean compon.}}+\underbrace{\mathbb{E}\left[\sum u_i^2\cdot\mathrm{Var}(s_i)\right]}_{\text{Variance compon.}}+\underbrace{\mathrm{Var}(u_b)}_{\text{Body rounding}}. \end{aligned} \quad (45)$$

Our simulations, reported in Table 2, show that our mean compensation technique eliminates the mean component of the mask, while the variance component is not changed. While the rerandomization technique targets a reduction of the full modulus switch noise, it is interesting to see that this technique also mainly reduces the mean component, and the variance component is only reduced slightly.

Although rerandomizations can reduce the variance component, Table 3 shows that this effect gives diminishing returns in function of the number of rerandomizations. Note that, for this table, we adjusted Bernard et al.'s technique to choose the best rerandomization within the allowed number of randomizations instead of the first one with an expected noise variance below the chosen threshold. This adjustment allows evaluating their noise reduction in function of the number of rerandomizations.

The overall result for the rerandomisation technique is a total average modulus switching noise reduction by 51% after an average of 15.3 rerandomizations for a standard parameter set[5] in the TFHE-rs library [Zam22], of which the vast majority is due to the reduction of the mean component, and further rerandomizations give diminishing returns.

Table 2 and Table 3 also include results of combining mean compensation and rerandomizations into a hybrid approach. Like standard rerandomization, our approach applies rerandomizations to the ciphertext. However, it also performs mean compensation before computing the expected total noise variance, taking that compensation into account when comparing against the threshold. Since mean compensation already reduces the mean component across all rerandomizations, the expected variance component becomes the dominant factor in selecting the best candidate. As a result, our hybrid method achieves similar reductions in total noise with fewer rerandomizations.

From an implementation perspective, mean compensation is preferred over rerandomization. It offers a fixed workflow, in contrast to rerandomizing, where the required number of

[5] PS1 in Appendix A

Table 3: Noise variance due to modulus switching in function of the number of alternative encodings. The rerandomization with the lowest expected noise variance out of all available options was chosen.

		Variance compon. + body rounding		Mean compon.		Total noise	
	# rerand.	Measured	Rel.	Measured	Rel.	Measured	Rel.
Baseline	/	$1.08\cdot10^{-6}$	100%	$1.07\cdot10^{-6}$	100%	$2.14\cdot10^{-6}$	100%
[BJSW25b]	8	$1.06\cdot10^{-6}$	98%	$7.18\cdot10^{-8}$	6.7%	$1.12\cdot10^{-6}$	52%
	64	$1.03\cdot10^{-6}$	95%	$2.14\cdot10^{-8}$	2.0%	$1.04\cdot10^{-6}$	49%
	512	$1.00\cdot10^{-6}$	93%	$1.32\cdot10^{-8}$	1.2%	$1.00\cdot10^{-6}$	47%
	4096	$9.82\cdot10^{-7}$	91%	$1.07\cdot10^{-8}$	1.0%	$9.73\cdot10^{-7}$	46%
Mean comp.	/	$1.08\cdot10^{-6}$	100%	0	0%	$1.06\cdot10^{-6}$	50%
Hybrid: Ours + [BJSW25b]	8	$1.03\cdot10^{-6}$	96%	0	0%	$1.04\cdot10^{-6}$	49%
	64	$1.00\cdot10^{-6}$	93%	0	0%	$1.01\cdot10^{-6}$	47%
	512	$9.81\cdot10^{-7}$	91%	0	0%	$9.67\cdot10^{-7}$	45%
	4096	$9.63\cdot10^{-7}$	89%	0	0%	$9.68\cdot10^{-7}$	45%

rerandomizations is found only at runtime. Moreover, the rerandomization implementation in the TFHE-rs library [Zam22] introduces an average slowdown of 11% for a single bootstrap. This performance degradation may be attributed to the lack of parallelism and the additional memory accesses for the encryptions of zero, as this slowdown disappears when executing in a single-threaded context or when parallelizing many different bootstraps.

5.2 Open research questions [BJSW25a]

Based on the previous analysis, we answer the open research questions raised during a recent presentation by Joye [BJSW25a] on the work by Bernard et al. [BJSW25b]:

1. "Beyond adding random encryptions of zero, are there alternative transformations?"
2. "Can we establish lower bounds on drift minimization?"
3. "Can drift-aware modulus switching be applied beyond FHE?"

To address the first question: yes, there are alternative transformations. Notably, each of these transformations reduces the mean component, which can be efficiently and elegantly compensated as demonstrated in this paper. However, none of the existing transformations fundamentally target the variance component $\sigma_u^2 \cdot \sigma_s^2$.

Going to the second question on a lower bound, we have clearly demonstrated that the lower bound on the mean component is the rounding error of the (compensated) body, while none of the techniques fundamentally change the variance component. As shown in Table 3, combining mean compensation with the technique of Bernard et al., the remaining variance component is only lowered slightly, with rapidly diminishing returns for higher number of ciphertexts tested.

One can even construct a theoretical lower bound for this reduction of the variance component. Under similar n, the minimal noise variance for security increases with decreasing q. Using the LWE estimator by Albrecht et al. [APS15], we find that for

$$n = 866 \text{ and } q = 2^{64}\text{: } \sigma^2_{min,q=2^{64}} = 2^{-39.30}$$
$$n = 866 \text{ and } q = 2^{12}\text{: } \sigma^2_{min,q=2^{12}} = 2^{-32.14}.$$

As modulus switching reduces q, this means that it should always increase the noise variance by a minimum of $\sigma^2_{min,q=2^{12}} - \sigma^2_{min,q=2^{64}} = 2^{-32.15} = 2.10\cdot10^{-10}$. Otherwise, we could use

Table 4: Expected and measured noise variance for modulus switching (MS), key switching decomposition (KS), and blind rotation decomposition (BR).

	Optimizations	Formula	Expected	Measured	Relative
MS	Standard	$\sigma_u^2 \cdot (n \cdot (\sigma_s^2 + \mu_s^2) + 1)$	$2.16 \cdot 10^{-6}$	$2.15 \cdot 10^{-6}$	1
	Mean comp.	$\sigma_u^2 \cdot (n \cdot \sigma_s^2 + 1)$	$1.08 \cdot 10^{-6}$	$1.07 \cdot 10^{-6}$	0.498
	Late rounding	$\sigma_u^2 \cdot (\frac{n}{2} \cdot 3 \cdot (\sigma_s^2 + \mu_s^2)^2 + 1)$	$1.62 \cdot 10^{-6}$	$1.61 \cdot 10^{-6}$	0.749
	Optimal BKU[1]	/	$1.05 \cdot 10^{-6}$	$1.03 \cdot 10^{-6}$	0.479
KS-decomp	Standard	$\sigma_u^2 \cdot kN \cdot (\sigma_s^2 + \mu_s^2)$	$7.95 \cdot 10^{-8}$	$7.84 \cdot 10^{-8}$	1
	Mean comp.	$\sigma_u^2 \cdot kN \cdot \sigma_s^2$	$3.97 \cdot 10^{-8}$	$3.88 \cdot 10^{-8}$	0.495
BR-decomp	Standard	$\sigma_U^2 \cdot (kN \cdot (\sigma_S^2 + \mu_S^2) + 1) \cdot \frac{n}{2}$	$5.26 \cdot 10^{-10}$	$5.26 \cdot 10^{-10}$	1
	Mean comp.	$\sigma_U^2 \cdot (kN \cdot \sigma_S^2 + 1) \cdot \frac{n}{2}$	$2.63 \cdot 10^{-10}$	$2.65 \cdot 10^{-10}$	0.504

[1] Optimal bootstrap key unrolling with factor 2

it to generate a lower security LWE sample and we would have found a new attack on LWE. Thus, a technique that would fundamentally reduce the variance component could be used to break LWE, which makes it unlikely to exist in the first place.

As for the third question, knowing the properties of the compensation we can clearly define when mean compensation is useful. Beyond the modulus switch noise reduction proposed by Bernard et al., one can additionally use the compensation technique for approximation errors from gadget decompositions. More generally, the technique can be used in any LWE instance where an approximation noise is made, and the secret has a non-zero mean. The gain of mean compensation is a noise reduction by a factor $\frac{\sigma_s^2 + \mu_s^2}{\sigma_s^2}$. Therefore, the technique is most pressing for a secret where $\frac{\mu_s^2}{\sigma_s^2}$ is not too small.

6 Experimental results

In this section, we validate and quantify the improvements of our proposed methods. First, we measure the noise due to modulus switching and gadget decomposition. We compare a scenario with and without mean compensation to confirm our theoretical findings. Second, we measure the computational overhead of our mean compensation techniques. Finally, we report the runtime improvements of using mean compensation during bootstrapping with optimized parameter sets for the new noise variance equations.

6.1 Noise variance

To measure the effect of mean compensation, we apply the different methods to a single parameter set. We opt for a standard parameter set[6] used in the TFHE-rs library [Zam22]. We created a Python script[7] that simulates the various approximations and measures the error with and without compensation. We calculated the variance of the errors from 2^{15} independent executions and show an overview in Table 4, confirming the expected noise reductions. Integration of mean compensation into the TFHE-rs library also confirms these results.

To compute the error probability for the parameter set, we have to measure the full accumulated noise variance. For example, we also have to include the noise variance introduced during encryption. The standard version of all operations accumulates a noise variance of

[6] PS1 in Appendix A

[7] Scripts and implementation available at https://github.com/KULeuven-COSIC/Dont_be_mean

Table 5: Error probability and relative execution time of bootstrap with different mean compensation techniques implemented. For modulus switching (MS), and key switching decomposition (KS), the variation in execution time between runs dominates the increase in computations. Blind rotation decomposition (BR) is notably more expensive.

Mean compensation			Error probability	Relative execution time
MS	KS-decomp	BR-decomp		
✗	✗	✗	$2^{-73.77}$	1
✓	✗	✗	$2^{-126.70}$	1.002
✓	✓	✗	$2^{-130.21}$	0.998
✓	✓	✓	$2^{-130.81}$	1.295

$2.51 \cdot 10^{-6}$ and results in an error probability of $2^{-73.77}$. By computing mean compensation for modulus switching and both gadget decompositions, we reduce the noise variance to $1.39 \cdot 10^{-6}$ and the error probability to $2^{-130.81}$.

6.2 Computational overhead of mean compensation

Next, we measure the computational overhead of implementing the three suggested mean compensation techniques. To this end, we implemented them in the TFHE-rs library [Zam22] and computed bootstraps using a standard parameter set[8]. The results are summarized in Table 5. Note that the main reductions in noise variance are found by using mean compensation for modulus switching and key switching decomposition. The overhead of these optimizations is negligible and is smaller than the measurement error of the timing measurements. This explains why some executions appear faster when we enable compensation. In contrast, we remark that the global reduction in noise variance is negligible for blind rotation decomposition, while the overhead is noticeably higher. For this reason, we will not use blind rotation decomposition in the following section.

6.3 Runtime improvements

The noise reduction of mean compensation can also be exchanged for performance improvements for identical security and error probability through two approaches. First, we optimized the parameter sets using our parameter estimation tool[9], based on the LWE estimator by Albrecht et al. [APS15], while the error probability is found using the noise formulas in Section 3 and the noise analysis by Chillotti et al. [CLOT21]. Second, we can combine mean compensation with the manyLUT approach by Chillotti et al., that increases the modulus switching noise, but evaluates two functions in one bootstrap.

Table 6 reports the computation time for parameter sets[10] with 128-bit security, and a maximum error probability of 2^{-40}, 2^{-64}, or 2^{-128}. We measure the time to compute two different functions for 64 different ciphertexts with four or five encrypted bits, excluding the padding bit. This is a realistic usecase, ideal for manyLUT, that computes both functions simultaneously. The results show that mean compensation allows to reduce computation time up to 2.06× in about half of the scenarios, as the speedup strongly depends on whether the lower noise variance allows a lower polynomial size. Moreover, we note that the combination between mean compensation and manyLUT allows a second function evaluation with low additional cost for some scenarios. As bootstrapping is typically the primary computational bottleneck in TFHE [CGGI16], this speedup should directly translate to improved performance across applications.

[8] PS1 in Appendix A
[9] Scripts and implementation available at https://github.com/KULeuven-COSIC/Dont_be_mean
[10] PS2-19 in Appendix A

Table 6: Computation time to evaluate two different functions for 64 different ciphertexts using programmable bootstrapping. All benchmarks were run using eight Ryzen 7 pro 8840hs cores.

Security	Error prob.	# bits	Standard	Mean comp.		Mean comp. + manyLUT [CLOT21]	
			[ms]	[ms]	speedup	[ms]	speedup
$\geq$ 128-bit	$\leq 2^{-40}$	4	194	183	1.06×	104	1.86×
		5	494	329	1.50×	368	1.34×
	$\leq 2^{-64}$	4	202	198	1.02×	206	0.98×
		5	704	668	1.05×	746	0.94×
	$\leq 2^{-128}$	4	434	210	2.06×	228	1.91×
		5	1463	732	2.00×	775	1.89×

7 Conclusion

In this paper, we suggested a mean compensation technique that halves the approximation error during modulus switching and gadget decomposition for both key switching and blind rotations. Based on our findings, we recommend enabling mean compensation for modulus switching and key switching by default, while disabling it for blind rotations, as the deploy-by-default configuration. This method comes with a negligible computational cost and reduces the error probability for a standard parameter set from $2^{-73.77}$ to $2^{-130.21}$. It also offers a fixed workflow and removes the dependency between the noise variance and the Hamming weight of the secret key. Furthermore, we combined mean compensation with bootstrap key unrolling, to further reduce the noise increase during modulus switching. This reduces the error probability to $2^{-134.26}$ or $2^{-143.19}$ for an unrolling factor of two or three respectively. Moreover, new parameter sets allow to exchange this decrease in error probability into performance improvements up to a factor 2.06×, for identical security and error probability requirements.

Acknowledgements

This work was supported in part by the Horizon 2020 ERC Advanced Grant (101020005 Belfort) and the CyberSecurity Research Flanders with reference number VOEWICS02.



References

[APS15] Martin R. Albrecht, Rachel Player, and Sam Scott. On the concrete hardness of learning with errors. Cryptology ePrint Archive, Report 2015/046, 2015.

[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, Cham, December 2022.

[BJSW25a] Olivier Bernard, Marc Joye, Nigel Smart, and Michael Walter. Strong IND-CPAD security for FHE (essentially) for free, March 2025. Presented at the FHE.org Conference.

[BJSW25b] Olivier Bernard, Marc Joye, Nigel P. Smart, and Michael Walter. Drifting towards better error probabilities in fully homomorphic encryption schemes. In *Advances in Cryptology – EUROCRYPT 2025: 44th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Madrid, Spain, May 4–8, 2025, Proceedings, Part VIII*, page 181–211, Berlin, Heidelberg, 2025. Springer-Verlag.

[BMMP18] Florian Bourse, Michele Minelli, Matthias Minihold, and Pascal Paillier. Fast homomorphic evaluation of deep discretized neural networks. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part III*, volume 10993 of *LNCS*, pages 483–512. Springer, Cham, August 2018.

[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, Berlin, Heidelberg, December 2016.

[CGGI18] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. TFHE: Fast fully homomorphic encryption over the torus. Cryptology ePrint Archive, Report 2018/421, 2018.

[CKKS17] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yong Soo Song. Homomorphic encryption for arithmetic of approximate numbers. In Tsuyoshi Takagi and Thomas Peyrin, editors, *ASIACRYPT 2017, Part I*, volume 10624 of *LNCS*, pages 409–437. Springer, Cham, December 2017.

[CLOT21] Ilaria Chillotti, Damien Ligier, Jean-Baptiste Orfila, and Samuel Tap. Improved programmable bootstrapping with larger precision and efficient arithmetic circuits for TFHE. In Mehdi Tibouchi and Huaxiong Wang, editors, *ASIACRYPT 2021, Part III*, volume 13092 of *LNCS*, pages 670–699. Springer, Cham, December 2021.

[FV12] Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive, Report 2012/144, 2012.

[GV25] Robin Geelen and Frederik Vercauteren. Fully homomorphic encryption for cyclotomic prime moduli. In *Advances in Cryptology – EUROCRYPT 2025*, pages 366–397, 2025.

[JLJ22] Lei Jiang, Qian Lou, and Nrushad Joshi. MATCHA: a fast and energy-efficient accelerator for fully homomorphic encryption over the torus. In *Proceedings of the 59th ACM/IEEE Design Automation Conference*, DAC '22, page 235–240, New York, NY, USA, 2022. Association for Computing Machinery.

[Joy22] Marc Joye. SoK: Fully homomorphic encryption over the [discretized] torus. *IACR TCHES*, 2022(4):661–692, 2022.

[LOT25] Damien Ligier, Jean-Baptiste Orfila, and Samuel Tap. Bootstrapping homomorphic ciphertext, January 2025. European Patent Application.

[Reg05] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In Harold N. Gabow and Ronald Fagin, editors, *37th ACM STOC*, pages 84–93. ACM Press, May 2005.

[YKP22] Tian Ye, Rajgopal Kannan, and Viktor K. Prasanna. FPGA acceleration of fully homomorphic encryption over the torus. In *2022 IEEE High Performance Extreme Computing Conference (HPEC)*, pages 1–7, 2022.

[Zam22] Zama. TFHE-rs: A Pure Rust Implementation of the TFHE Scheme for Boolean and Integer Arithmetics Over Encrypted Data, 2022. https://github.com/zama-ai/tfhe-rs.

[ZYL+18] Tanping Zhou, Xiaoyuan Yang, Longfei Liu, Wei Zhang, and Ningbo Li. Faster bootstrapping with multiple addends. *IEEE Access*, 6:49868–49876, 2018.

A Parameter sets

Table 7: Summary of all parameter sets used throughout this work, generated using [12]. Note that the plaintext space includes the padding bit. All parameter sets have 128-bit security according to the LWE estimator by Albrecht et al. [APS15]. PS1, PS2-7, PS8-13, and PS14-19 have respectively an error probability of $2^{-73.77}$, 2^{-40}, 2^{-64}, and 2^{-128}.

	p	q	N	n	k	β_{br}	l_{br}	β_{ks}	l_{ks}	e_{lwe}	e_{glwe}	MC	manyLUT
PS1	2^5	2^{64}	2^{11}	866	1	2^{23}	1	2^3	5	$2.046 \cdot 10^{-6}$	$2.845 \cdot 10^{-15}$	✗	✗
PS2	2^5	2^{64}	2^{11}	842	1	2^{23}	1	2^5	3	$1.907 \cdot 10^{-6}$	$8.882 \cdot 10^{-16}$	✗	✗
PS3	2^6	2^{64}	2^{12}	919	1	2^{22}	1	2^3	6	$4.768 \cdot 10^{-7}$	$5.421 \cdot 10^{-20}$	✗	✗
PS4	2^5	2^{64}	2^{10}	803	2	2^{23}	1	2^2	7	$3.815 \cdot 10^{-6}$	$8.882 \cdot 10^{-16}$	✓	✗
PS5	2^6	2^{64}	2^{11}	880	1	2^{15}	2	2^3	5	$9.537 \cdot 10^{-7}$	$8.882 \cdot 10^{-16}$	✓	✗
PS6	2^5	2^{64}	2^{11}	803	1	2^{23}	1	2^2	7	$3.815 \cdot 10^{-6}$	$8.882 \cdot 10^{-16}$	✓	✓
PS7	2^6	2^{64}	2^{12}	919	1	2^{15}	2	2^3	6	$4.768 \cdot 10^{-7}$	$5.421 \cdot 10^{-20}$	✓	✓
PS8	2^5	2^{64}	2^{11}	880	1	2^{23}	1	2^5	3	$9.537 \cdot 10^{-7}$	$8.882 \cdot 10^{-16}$	✗	✗
PS9	2^6	2^{64}	2^{12}	919	1	2^{15}	2	2^4	4	$4.768 \cdot 10^{-7}$	$5.421 \cdot 10^{-20}$	✗	✗
PS10	2^5	2^{64}	2^{11}	763	1	2^{23}	1	2^2	7	$7.629 \cdot 10^{-6}$	$8.882 \cdot 10^{-16}$	✓	✗
PS11	2^6	2^{64}	2^{12}	880	1	2^{15}	2	2^4	4	$9.537 \cdot 10^{-7}$	$5.421 \cdot 10^{-20}$	✓	✗
PS12	2^5	2^{64}	2^{12}	880	1	2^{22}	1	2^5	3	$9.537 \cdot 10^{-7}$	$5.421 \cdot 10^{-20}$	✓	✓
PS13	2^6	2^{64}	2^{13}	880	1	2^{15}	2	2^3	5	$9.537 \cdot 10^{-7}$	$5.421 \cdot 10^{-20}$	✓	✓
PS14	2^5	2^{64}	2^{11}	842	1	2^{22}	1	2^3	5	$1.907 \cdot 10^{-6}$	$5.421 \cdot 10^{-20}$	✗	✗
PS15	2^6	2^{64}	2^{14}	919	1	2^{15}	2	2^3	6	$4.768 \cdot 10^{-7}$	$5.421 \cdot 10^{-20}$	✗	✗
PS16	2^5	2^{64}	2^{11}	842	1	2^{23}	1	2^3	5	$1.907 \cdot 10^{-6}$	$8.882 \cdot 10^{-16}$	✓	✗
PS17	2^6	2^{64}	2^{12}	958	1	2^{15}	2	2^4	4	$2.384 \cdot 10^{-7}$	$5.421 \cdot 10^{-20}$	✓	✗
PS18	2^5	2^{64}	2^{12}	880	1	2^{22}	1	2^3	5	$9.537 \cdot 10^{-7}$	$5.421 \cdot 10^{-20}$	✓	✓
PS19	2^6	2^{64}	2^{13}	982	1	2^{15}	2	2^5	4	$1.192 \cdot 10^{-7}$	$5.421 \cdot 10^{-20}$	✓	✓

[12] Scripts and implementation available at https://github.com/KULeuven-COSIC/Dont_be_mean