# 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 $2^{-64.30}$ to $2^{-100.47}$, or allows the selection of more efficient parameters leading to a speedup of bootstrapping up to a factor 2.14×.

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

## 1 Introduction

Machine Learning (ML) has rapidly advanced in recent years, driven by an abundance of data. ML applications range from autonomous driving to medical diagnosis and often use sensitive data. However, many ML models are deployed through cloud computing, posing significant privacy risks when users upload unprotected data to untrusted cloud servers. To leverage sensitive data securely, robust data protection is essential.

A solution can be found in Fully Homomorphic Encryption (FHE), which allows computations on encrypted data without revealing any information about the data. Using FHE, data is first encrypted by the client and sent to the cloud. Then, the cloud performs computations on the encrypted data and returns the encrypted result to the client. Only the client can decrypt this to obtain the final result. This process ensures data security throughout the computation. Notable 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) problem [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 by approximation errors during modulus switching, and during gadget decomposition in both key switching and blind rotation, which are all necessary subroutines of bootstrapping. For a standard parameter set in the TFHE-rs library [Zam24], these sources account for 74.8% 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 Legier et al. [LOT25] and Bernard et al. [BJSW24]. Legier et al. suggest an alternative to bootstrapping that subtracts noise from a ciphertext. They measure the noise by shifting it into the message space and extracting a chosen number of bits into a new ciphertext using a programmable bootstrap. In this way, they can trade off the computational cost of the programmable bootstrap against the reduction of noise. This approach can only evaluate the identity function and no longer allows programmable bootstrapping. Bernard et al. reduce the increase in noise variance during modulus switching by adding encryptions of zero to the ciphertext and searching an encryption of the ciphertext with a low approximation error. In TFHE-rs, they add up to 50 different encryptions of zero to the ciphertext and select the result that generates the least modulus switching noise. This procedure extends the running time by around 10% and requires the storage of 50 additional ciphertexts.

During a recent presentation by Joye [BJSW25] on the work by Bernard et al., three open research questions were raised:

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?"

Throughout this work, we will answer these questions.

## 1.1 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 provides efficiency. It 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, $\mu_s$. 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 $\sigma_u^2 \cdot (\sigma_s^2 + \mu_s^2)$.

Our compensation technique removes the contribution of the mean term $\mu_s$, thereby halving the approximation error during modulus switching and gadget decomposition for both key switching and blind rotations. This method comes with a negligible computational cost and reduces the error probability for a standard parameter set from $2^{-64.30}$ to $2^{-100.47}$. Alternatively, by generating new parameter sets, we can exchange this decrease in error probability into performance improvements up to a factor 2.14×, for identical security and error probability requirements.

Furthermore, we look into the combination of our mean compensation technique 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 us to further reduce the error probability to $2^{-102.73}$ or $2^{-107.59}$ for an unrolling factor of two or three respectively.

### 1.2 Organization of the paper

After the preliminaries in Section 2, we will explain our new mean compensation technique to halve the noise variance due to approximation errors in Section 3. Then, Section 4 combines our technique with bootstrap key unrolling to achieve a further noise reduction, while Section 5 analyzes our technique to answer the open research questions [BJSW25]. Section 6 subsequently explains how to exchange our noise reduction for improved performance. Finally, in Section 7, we verify our noise reduction in the TFHE-rs library [Zam24], show that the computational overhead to reduce the noise is negligible, and illustrate possible performance improvements.

## 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 $\{0, \frac{1}{q}, \ldots, \frac{q-1}{q}\}$ 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 $\{0, \frac{1}{p}, \ldots, \frac{p-1}{p}\}$, 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 coefficients in $\mathbb{T}_q$ 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 $\mathcal{R}_q$.

For a set $S$, $a \xleftarrow{\$} S$ denotes that $a$ is sampled uniformly at random from $S$. Alternatively, for a distribution $\chi$, $a \leftarrow \chi$ denotes that $a$ is sampled according to the distribution of $\chi$. In this work, we use $\hat{\chi}$, which is the gaussian distribution with mean $\mu = 0$ and variance $\sigma^2 = \sigma^2_{lwe}$, 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 $\sigma^2 = \sigma^2_{glwe}$, discretized onto $\mathcal{R}_q$. The distribution $\mathcal{U}(\llbracket x,y \rrbracket)$ 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 $s\xleftarrow{\$}\mathbb{B}[X]/(X^N+1)^k$, $a\xleftarrow{\$}\mathcal{R}_q^k$, 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, $\sigma_e^2$, 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, $\boxdot$: $\text{TGGSW}\times\text{TGLWE}\to\text{TGLWE}$. 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].

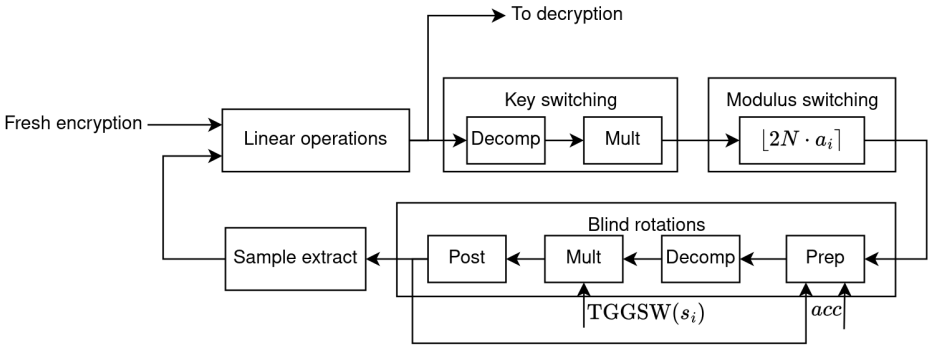



**Figure 1:** Overview of the subroutines making up bootstrapping

### 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 $\tilde{s}$, informally $b-\sum a_i\cdot\text{LWE}_{\tilde{s}}(s_i)$.

The multiplication between $a_i$ and $\text{LWE}_{\tilde{s}}(s_i)$ 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 $\log_2(\beta_{ks})$ bits. The remaining term $u_i$ is the approximation error:

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

Afterwards, we multiply every decomposed mask element with its respective $\text{LWE}_{\tilde{s}}(s_i)$, a matrix holding a special encryption of $s_i$ under the target key $\tilde{s}$.

Both steps introduce noise in the output ciphertext. The underlying sources are the approximation error $u_i$ and the encryption noise of $\text{LWE}_{\tilde{s}}(s_i)$ with variance $\sigma_{lwe}^2$.

### 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 a new approximation error, $u_i = 2N \cdot a_i - \lfloor 2N \cdot a_i \rceil$, increasing the noise variance of the ciphertext.

### 2.3.3 Blind rotation

The goal of the blind rotations is to multiply a TGLWE ciphertext $acc_0 = \sum_{i=0}^{N-1} f(i) \cdot X^i$, 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 a new polynomial with $f(b - a \cdot s) = f(m + e)$ as its constant term. By choosing the coefficients 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 constant term after the multiplication, $f(m+e)$, equals $g(m)$ as long as the error $e$ is sufficiently small.

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

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

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, $\text{TGGSW}(s_i)$. 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 $\sigma_U^2$. The second error originates from the encryption noise on the TGGSW encryptions, with variance $\sigma_{glwe}^2$.

### 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, $\sigma_u^2$, 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, $\mu_s$. 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 $n \cdot \sigma_u^2 \cdot (\sigma_s^2 + \mu_s^2)$, where $\sigma_u^2$ is the variance of the approximation error, and $\sigma_s^2$ and $\mu_s$ are the variance and mean of the secret $s_i$.

The mean of the secret is not zero, as TFHE uses $s \xleftarrow{\$} \mathbb{B}^n$ with mean $\mu_s = \frac{1}{2}$ and standard deviation $\sigma_s = \frac{1}{2}$. To remove the mean term, we would need a symmetric distribution for the secret. However, we cannot simply use a symmetric distribution, such as random uniform samples from $\{-0.5, 0.5\}$, as blind rotations would no longer work.

Nevertheless, as the approximation errors are publicly known, we could compensate for the error $u_i \cdot s_i$ 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, $\mu_s \cdot \sum u_i$.

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

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

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.

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 $m$, $(a, b = \sum_{i=1}^{kN} a_i s_i + m + e_{in}) \in \mathbb{T}_q^{kN} \times \mathbb{T}_q$. It splits every $a_i$ into $l_{ks}$ terms of $\log_2(\beta_{ks})$ bits:

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

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

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

This uniform distribution has a variance of $\sigma_u^2 = \frac{1}{12 \cdot \beta_{ks}^{2 \cdot l_{ks}}}$.

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:

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

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

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

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

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

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: $(a,b_{comp}) \in \mathbb{T}_q^{kN} \times \mathbb{T}_q$

$c = 0$ ;
**for** $i = 1$ **to** $kN$ **do**
  $A_i = \left\lfloor \beta_{ks}^{l_{ks}} \cdot a_i \right\rceil$ ;
  $c \mathrel{+}= a_i - \frac{A_i}{\beta_{ks}^{l_{ks}}}$ ;
**end**
$b_{comp} = b - c \cdot \mu_s$ ;
**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 $\sigma_u^2 = \frac{1}{12 \cdot (2N)^2}$.

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

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

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

Now, we compensate the mean term by adjusting the body $b$ to $b_{comp} = b - \mu_s \cdot \sum_{i=1}^{n} u_i$. Note that rounding this compensated body $b_{comp}$ results in the same error distribution for $u_b$. After this compensation, we can recompute the introduced error during modulus switching as:

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

with the noise variance $\sigma_{ms}^2 = \sigma_u^2 \cdot (n \cdot \sigma_s^2 + 1)$.

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 $acc_i = \text{TGGSW}(s_i) \boxdot (acc_{i-1} \cdot X^{a_i} - acc_{i-1}) + acc_{i-1} = acc_{i-1} \cdot X^{a_i s_i}$ and therefor uses 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:

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

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

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

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

This uniform distribution has a variance of $\sigma_U^2 = \frac{1}{12 \cdot \beta_{br}^{2 \cdot l_{br}}}$.

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

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

with the variance on each of the elements of the polynomial equal to

$$\sigma_U^2 \cdot (kN \cdot (\sigma_S^2 + \mu_S^2) + 1). \tag{17}$$

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 $B_{comp} = B - \mu_S I^N \cdot \sum_{i=1}^{k} U_i$, with $I^N = 1 + X + X^2 + \cdots + X^{N-1}$. Similar to the compensated modulus switching, Equation 12, the introduced error becomes:

$$E_{br} = -U_B + \sum_{i=1}^{k} U_i \cdot (S_i - \mu_S I^N) \tag{18}$$

with the variance on each of the elements of the polynomial equal to

$$\sigma_U^2 \cdot (kN \cdot \sigma_S^2 + 1). \tag{19}$$

The calculation of the compensated body polynomial $B_{comp}$ uses the result of the multiplication between the polynomials $\mu_S I^N$ and $\sum_{i=i}^{k} U_i$. 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

$$\begin{bmatrix} C[0] \\ C[1] \\ \vdots \\ C[N-1] \end{bmatrix} = \mu_S \cdot \begin{bmatrix} 1 & -1 & -1 & \dots & -1 \\ 1 & 1 & -1 & \dots & -1 \\ 1 & 1 & 1 & \dots & -1 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & 1 & \dots & 1 \end{bmatrix} \cdot \begin{bmatrix} \sum_{i=1}^{k} U_i[0] \\ \sum_{i=1}^{k} U_i[1] \\ \vdots \\ \sum_{i=1}^{k} U_i[N-1] \end{bmatrix}. \tag{20}$$

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 (line 7).

**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: $(A,B_{comp}) \in \mathcal{R}_q^k \times \mathcal{R}_q$

1 $U = 0 \in \mathcal{R}_q$ ;
2 **for** $i = 1$ **to** $k$ **do**
3 $\quad U \mathrel{+}= A_i - \frac{\left\lfloor \beta_{br}^{l_{br}} \cdot A_i \right\rceil}{\beta_{br}^{l_{br}}}$ ;
4 **end**
5 $C[N-1] = \mu_S \cdot \sum_{h=0}^{N-1} U[h]$ ;
6 **for** $h = N-1$ **to** $0$ **do**
7 $\quad B_{comp}[h] = B[h] - C[h]$ ;
8 $\quad C[h-1] = C[h] - 2\mu_S \cdot U[h]$ ;
9 **end**
10 **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 of

$$\sigma^2_{br-decomp} = \sigma^2_U \cdot (kN \cdot \sigma^2_S + 1) \cdot (\sigma^2_s + \mu^2_s) \cdot n \tag{21}$$

over the $n$ rotations. $\sigma^2_s$ and $\mu_s$ are respectively the variance and mean of the encrypted secret $s_i$.

## 3.4 Message compensation

In the previous section, we reduced the total increase in noise variance due to approximation errors during blind rotations down to $\sigma^2_{br-decomp} = \sigma^2_U \cdot (kN \cdot \sigma^2_S + 1) \cdot (\sigma^2_s + \mu^2_s) \cdot n$. In this section, we explain why we cannot remove the term $\mu^2_s$, to halve the noise variance a second time.

In contrast to previous sections, the error term that is multiplied with a secret bit, $s_i \xleftarrow{\$} \mathbb{B}$, is not a plaintext value. In this case, the error is already part of an encryption. Therefore, the sign and magnitude of the error are unknown, stopping us from compensating the mean term as before.

Instead, we can make the key distribution symmetric and accept that we will have to change the formula for blind rotations. For example, we could try to use

$$\mathrm{TGGSW}(s_i - \frac{1}{2}) \boxdot (acc_{i-1} \cdot X^{a_i} - acc_{i-1}) + (acc_{i-1} \cdot X^{a_i} + acc_{i-1})/2. \tag{22}$$

However, we cannot calculate $(acc_{i-1} \cdot X^{a_i} + acc_{i-1})/2$, as we cannot retrieve its most significant bit when dividing by two. Consequently, this mean term cannot be removed from the variance equation.

# 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

$$acc_i = \mathrm{TGGSW}(s_i) \boxdot (acc_{i-1} \cdot X^{\lfloor 2N \cdot a_i \rceil} - acc_{i-1}) + acc_{i-1} \tag{23}$$

$$acc_{i+1} = \mathrm{TGGSW}(s_{i+1}) \boxdot (acc_i \cdot X^{\lfloor 2N \cdot a_{i+1} \rceil} - acc_i) + acc_i, \tag{24}$$

one would compute

$$acc_{i+1} = \begin{bmatrix} & (X^{\lfloor 2N \cdot a_i \rceil + \lfloor 2N \cdot a_{i+1} \rceil} - 1) \cdot \mathrm{TGGSW}(s_i \cdot s_{i+1}) \\ + & (X^{\lfloor 2N \cdot a_i \rceil} - 1) \cdot \mathrm{TGGSW}(s_i \cdot (1 - s_{i+1})) \\ + & (X^{\lfloor 2N \cdot a_{i+1} \rceil} - 1) \cdot \mathrm{TGGSW}((1 - s_i) \cdot s_{i+1}) \\ + & 1 \end{bmatrix} \boxdot acc_{i-1} \tag{25}$$

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 [Zam24]. 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:

$$acc_{i+1} = \begin{cases} acc_{i-1}X^{\lfloor 2N\cdot a_i\rceil+\lfloor 2N\cdot a_{i+1}\rceil}, & \text{if } s_i=1 \;\&\; s_{i+1}=1 \\ acc_{i-1}X^{\lfloor 2N\cdot a_i\rceil}, & \text{if } s_i=1 \;\&\; s_{i+1}=0 \\ acc_{i-1}X^{\lfloor 2N\cdot a_{i+1}\rceil}, & \text{if } s_i=0 \;\&\; s_{i+1}=1 \\ acc_{i-1}, & \text{if } s_i=0 \;\&\; s_{i+1}=0. \end{cases} \tag{26}$$

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 $s_{i+1}$:

$$\begin{aligned} e_{ms,i} &= (u_i+u_{i+1})\cdot s_i\cdot s_{i+1}+u_i\cdot s_i\cdot(1-s_{i+1})+u_{i+1}\cdot(1-s_i)\cdot s_{i+1} \\ &= u_i\cdot s_i+u_{i+1}\cdot s_{i+1} \end{aligned} \tag{27}$$

with a variance of

$$\sigma^2_{ms,i} = 2\cdot\sigma^2_u\cdot(\sigma^2_s+\mu^2_s). \tag{28}$$

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:

$$\begin{aligned} \sigma^2_{ms} &= \frac{n}{2}\cdot\sigma^2_{ms,i}+\sigma^2_u \\ &= n\cdot\sigma^2_u\cdot(\sigma^2_s+\mu^2_s)+\sigma^2_u. \end{aligned} \tag{29}$$

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:

$$\sigma^2_{ms} = n\cdot\sigma^2_u\cdot\sigma^2_s+\sigma^2_u. \tag{30}$$

**Late rounding** A simple and commonly used approach to reduce the modulus switching noise is what we will call late rounding. When both $s_i$ and $s_{i+1}$ equal one, we can avoid the two separate roundings. Instead, we can postpone rounding and use $\lfloor 2N\cdot a_i+2N\cdot a_{i+1}\rceil$ to compute:

$$acc_{i+1} = \begin{cases} acc_{i-1}X^{\lfloor 2N\cdot a_i+2N\cdot a_{i+1}\rceil}, & \text{if } s_i=1 \;\&\; s_{i+1}=1 \\ acc_{i-1}X^{\lfloor 2N\cdot a_i\rceil}, & \text{if } s_i=1 \;\&\; s_{i+1}=0 \\ acc_{i-1}X^{\lfloor 2N\cdot a_{i+1}\rceil}, & \text{if } s_i=0 \;\&\; s_{i+1}=1 \\ acc_{i-1}, & \text{if } s_i=0 \;\&\; s_{i+1}=0. \end{cases} \tag{31}$$

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

$$e_{ms,i} = u_{i;i+1}\cdot s_i\cdot s_{i+1}+u_i\cdot s_i\cdot(1-s_{i+1})+u_{i+1}\cdot(1-s_i)\cdot s_{i+1} \tag{32}$$

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:

$$\sigma^2_{ms,i} = \sigma^2_u\cdot(\sigma^2_s+\mu^2_s)^2+2\cdot\sigma_u\cdot(\sigma^2_s+\mu^2_s)\cdot(\sigma^2_s+(1-\mu_s)^2). \tag{33}$$

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

$$\sigma^2_{ms} = n\cdot\sigma^2_u\cdot\frac{(\sigma^2_s+\mu^2_s)^2+2\cdot(\sigma^2_s+\mu^2_s)\cdot(\sigma^2_s+(1-\mu_s)^2)}{2}+\sigma^2_u. \tag{34}$$

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 $b_{comp} = b - \mu_s \cdot \sum_{i=1}^{n} u_i$ as well as the late rounding strategy. As a result, the compensation term now appears in the error of a single iteration:

$$e_{ms,i} = u_{i;i+1} \cdot s_i \cdot s_{i+1} + u_i \cdot s_i \cdot (1 - s_{i+1}) + u_{i+1} \cdot (1 - s_i) \cdot s_{i+1} - (u_i + u_{i+1}) \cdot \mu_s. \tag{35}$$

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 both $u_i$ and $u_{i+1}$ are large and have the same sign, late rounding will round to a different value and $u_{i;i+1} \neq u_i + u_{i+1}$. The difference is captured by $T_i$:

$$T_i = \begin{cases} 1 & \text{if } u_i + u_{i+1} > 1/2 \\ -1 & \text{if } u_i + u_{i+1} < -1/2 \\ 0 & \text{otherwise.} \end{cases} \tag{36}$$

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

$$\begin{aligned} e_{ms,i} = &(u_i + u_{i+1} - T_i) \cdot s_i \cdot s_{i+1} + u_i \cdot s_i \cdot (1 - s_{i+1}) \\ &+ u_{i+1} \cdot (1 - s_i) \cdot s_{i+1} - (u_i + u_{i+1}) \cdot \mu_s \end{aligned} \tag{37}$$

which simplifies to:

$$e_{ms,i} = u_i \cdot (s_i - \mu_s) + u_{i+1} \cdot (s_{i+1} - \mu_s) - T_i \cdot s_i \cdot s_{i+1}. \tag{38}$$

We compute the variance of this error as:

$$\sigma^2_{ms,i} = \sum_{s_i} P[s_i] \cdot \sum_{s_{i+1}} P[s_{i+1}] \cdot \int_{u_i=-1/2}^{1/2} \int_{u_{i+1}=-1/2}^{1/2} e^2_{ms,i} \cdot du_i \cdot du_{i+1}. \tag{39}$$

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

$$\sigma^2_{ms,i} = \sigma^2_u \cdot \frac{3}{4}. \tag{40}$$

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:

$$b_{comp} = b - \boldsymbol{c_u} \cdot \sum_{i=1}^{n} u_i + \boldsymbol{c_T} \cdot \sum_{i=1}^{n/2} T_{2i-1} \tag{41}$$

**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 |

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

$$T_i = \begin{cases} 1 & \text{if } u_i + u_{i+1} > \boldsymbol{\tau} \\ -1 & \text{if } u_i + u_{i+1} < -\boldsymbol{\tau} \\ 0 & \text{otherwise.} \end{cases} \tag{42}$$

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

$$\begin{aligned} acc_{i+1} = \left( \begin{bmatrix} & (X^{\lfloor 2N \cdot a_i \rceil + \lfloor 2N \cdot a_{i+1}) \rceil + T_i} - 1) \cdot \mathrm{TGGSW}(s_i \cdot s_{i+1}) \\ + & (X^{\lfloor 2N \cdot a_i \rceil} - 1) \cdot \mathrm{TGGSW}(s_i \cdot (1 - s_{i+1})) \\ + & (X^{\lfloor 2N \cdot a_{i+1} \rceil} - 1) \cdot \mathrm{TGGSW}((1 - s_i) \cdot s_{i+1}) \\ + & 1 \end{bmatrix} \boxdot acc_{i-1} \right) \\ \cdot X^{\boldsymbol{c_u} \cdot u_i + \boldsymbol{c_u} \cdot u_{i+1} - \boldsymbol{c_T} \cdot T_i}. \end{aligned} \tag{43}$$

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

$$e_{ms,i} = u_i \cdot (s_i - \boldsymbol{c_u}) + u_{i+1} \cdot (s_{i+1} - \boldsymbol{c_u}) - T_i \cdot (s_i \cdot s_{i+1} - \boldsymbol{c_T}). \tag{44}$$

Next, we compute the variance based on this error as in Equation 39 and minimize the noise variance $\sigma^2_{ms,i}$ 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:

$$\boldsymbol{c_u} = 1/2 (= \mu_s) \tag{45}$$

$$\boldsymbol{c_T} = 1/4 (= \mu_s^2) \tag{46}$$

$$\boldsymbol{\tau} = 3/4. \tag{47}$$

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

$$e_{ms,i} = u_i \cdot (s_i - \mu_s) + u_{i+1} \cdot (s_{i+1} - \mu_s) - T_i \cdot (s_i \cdot s_{i+1} - \mu_s^2). \tag{48}$$

**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 revisit and answer the open research questions [BJSW25]:

**Table 2:** Noise variance due to modulus switching using the technique by Bernard et al. [BJSW24] in function of the number of alternative encodings.

| | | Variance comp. | | Mean comp. | | Total noise | |
|---|---|---|---|---|---|---|---|
| | # altern. | $\sigma_u^2 \cdot \sigma_s^2$ | Rel. | $\sigma_u^2 \cdot \mu_s^2$ | Rel. | $\sigma_u^2 \cdot (\sigma_s^2 + \mu_s^2)$ | Rel. |
| [BJSW24] | 1 | $1.03 \cdot 10^{-6}$ | 100% | $1.03 \cdot 10^{-6}$ | 100% | $2.07 \cdot 10^{-6}$ | 100% |
| | 8 | $1.01 \cdot 10^{-6}$ | 98% | $6.88 \cdot 10^{-8}$ | 7% | $1.08 \cdot 10^{-6}$ | 52% |
| | 64 | $9.82 \cdot 10^{-7}$ | 95% | $2.01 \cdot 10^{-8}$ | 2% | $1.00 \cdot 10^{-6}$ | 48% |
| | 512 | $9.58 \cdot 10^{-7}$ | 93% | $1.05 \cdot 10^{-8}$ | 1% | $9.68 \cdot 10^{-7}$ | 47% |
| | 4096 | $9.38 \cdot 10^{-7}$ | 91% | $7.91 \cdot 10^{-9}$ | 1% | $9.46 \cdot 10^{-7}$ | 46% |
| Ours | / | $1.03 \cdot 10^{-6}$ | 100% | 0 | 0% | $1.03 \cdot 10^{-6}$ | 50% |
| Ours + [BJSW24] | 8 | $9.90 \cdot 10^{-7}$ | 96% | 0 | 0% | $9.90 \cdot 10^{-7}$ | 48% |
| | 64 | $9.61 \cdot 10^{-7}$ | 93% | 0 | 0% | $9.61 \cdot 10^{-7}$ | 46% |
| | 512 | $9.39 \cdot 10^{-7}$ | 91% | 0 | 0% | $9.39 \cdot 10^{-7}$ | 45% |
| | 4096 | $9.21 \cdot 10^{-7}$ | 89% | 0 | 0% | $9.21 \cdot 10^{-7}$ | 44% |

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?"

Similarly to this work, Bernard et al. [BJSW24] suggest an approach to reduce modulus switching noise. By adding encryptions of zero to the ciphertext, they search for an alternative encryption that results in lower rounding errors during modulus switching. We implemented their approach in Python to measures the variance over many independent executions. By comparing 64 alternatives and selecting the one with the lowest expected approximation error, their technique reduces modulus switching noise by around 52%. However, this approach takes variable time as it is unknown when a good alternative encoding will be found. Moreover, its implementation in TFHE-rs is on average 11% slower compared to our suggested mean compensation.

Our compensation technique reduces the noise by a similar amount of 50%. The similarity between the noise reduction results is not a coincidence. While our technique focuses on the mean term in $\sigma_u^2 \cdot (\sigma_s^2 + \mu_s^2)$, their technique reduces the overall noise term through an optimization of the full $\sigma_u^2 \cdot (\sigma_s^2 + \mu_s^2)$ term. However, upon closer inspection, one can see that their optimization problem also mostly reduces the $\sigma_u^2 \cdot \mu_s^2$ term. The corresponding term in the objective function dominates the term related to $\sigma_u^2 \cdot \sigma_s^2$, as it is on average 117× larger. This indicates that both techniques mainly focus on mean compensation, ours by subtracting a compensation term $\mu_s \cdot \sum u_i$, theirs by minimizing this compensation term through repetitive addition of encryptions of zero.

Table 2 shows the resulting variance introduced during modulus switching in function of the number of alternatives generated. We separated the contributions related to the mean and the variance of the secret, to highlight that the term related to the variance resists reduction.

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 is zero, while none of the techniques fundamentally change the variance. As shown in Table 2, combining mean compensation with the technique of Bernard et al. the remaining variance 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

$$n = 834 \text{ and } q = 2^{64}\text{: } \sigma^2_{min,q=2^{64}} = 2^{-37.60}$$
$$n = 834 \text{ and } q = 2^{12}\text{: } \sigma^2_{min,q=2^{12}} = 2^{-31.44}.$$

As modulus switching reduces $q$, this means that it should always increase the noise variance by a minimum of $\sigma^2_{min,q=2^{12}} - \sigma^2_{min,q=2^{64}} = 2^{-31.46} = 3.38 \cdot 10^{-10}$. Otherwise, we could use 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 Translating noise reduction in better parameters

The noise reduction techniques described above reduce the variance and thus the error probability. The ultimate goal of these techniques is to find more efficient parameter sets that allow faster FHE computations, under the same security and error probability demands. To estimate the improvements possible with our new techniques, we developed a tool to find optimal parameters for a given security and error probability target. To estimate the security we use the LWE estimator by Albrecht et al. [APS15], while the noise is found using the noise formulas in Section 3 and the noise analysis by Chillotti et al. [CLOT21].

To find optimal parameters, we loop over a sensible range for each parameter (polynomial size, LWE size, GLWE size, decomposition levels and bases, and standard deviations for both the blind rotation key and key switching key), and keep the most promising parameter sets that fulfill the security and error probability conditions. This final set of promising parameters is then benchmarked in the TFHE-rs library [Zam24] to find the optimal parameter set. To speed up this process, we performed early stopping for parameter sets that did not fulfill the requirements (for example, when for a given LWE size, the standard deviation becomes too small for security) or for parameter sets that were strictly worse than other sets. The parameter estimation script will be made available.

We did not have access to the parameter estimation scripts used for the TFHE-rs library. While not identical, the parameters we found are close to the parameters used in TFHE-rs for the same security and error probability targets.

## 6.1 Analysis of the effects of mean reduction

In TFHE, the overall noise can be subdivided into three principal components: key switching noise (both cryptographic and approximation errors), modulus switching noise, and blind ro-

tation noise (both cryptographic and approximation errors). The dominant component, modulus switching noise, is intrinsically linked to the polynomial degree $N$: $\sigma_{ms}^2 = \frac{1}{12 \cdot (2N)^2} \cdot (n \cdot \sigma_s^2 + 1)$. To achieve a small modulus switching noise, $N$ must be chosen sufficiently large. However, increasing $N$ not only raises the computational cost of polynomial multiplications in the external product (which scales with $\mathcal{O}(N\log(N))$ for the FFT operations), but it also amplifies the other noise sources. Furthermore, $N$ is typically a power of two, reducing the available options. This is the main reason why TFHE schemes become significantly more costly beyond the 4-bit (plus one padding bit) message space boundary. At that point, $N$ typically has to be doubled, which leads to an increased noise for the other sources and thus an overall increased cost.

A reduction of the modulus switch variance with a factor 4, would allow a reduction of the value of $N$ with a factor 2 (due to the quadratic $N$ in the noise formula). This would result in a reduction of the overall computing cost of more than a factor 2, due to both a reduction in the polynomial multiplication cost, and a halving of the other noise sources which can be exploited to relax other parameters.

Note that reducing the value of $N$ has its limits, as security requires sufficiently large GLWE parameters. In practice, this means that the value of $k \cdot N$ should always be larger than 512 or even 1024. As a result, for smaller message spaces, which typically have lower requirements for the modulus switching noise, our noise reduction techniques are expected to have a smaller impact on $N$ and thus on the efficiency. For the larger parameter sets (3 bits or more), our method does not suffer from this restriction. In that case, we can theoretically halve $N$ in half of the parameter cases, enabling more than a factor $2\times$ efficiency increase.

Moreover, our mean compensation technique is especially interesting in combination with multiple function evaluations through manyLUT. Using this technique, only the modulus switch noise is increased (more specifically, the variance is quadrupled). Our mean compensation technique can half the variance, thus halving the effect of this increase. In some instances, this leads to a second function evaluation with no additional cost, as will be detailed in the results section. Furthermore, this approach is not limited due to minimal requirements for $k \cdot N$ for security, as they are not reduced.

# 7 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 improvements when using mean compensation and optimal parameter sets.

## 7.1 Noise variance

To measure the effect of mean compensation, we apply the different methods to a single parameter set. We opt for the default parameters used in the TFHE-rs library [Zam24], displayed in Table 3. We created a Python script that simulates the various approximating operations 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 the 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 $2.90 \cdot 10^{-6}$ and results in an error probability of $2^{-64.30}$. By computing mean compensation

**Table 3:** Default parameter set used in TFHE-rs v0.10.0

| $p$ | $q$ | $N$ | $n$ | $k$ | $\beta_{br}$ | $l_{br}$ | $\beta_{ks}$ | $l_{ks}$ | $e_{lwe}$ | $e_{glwe}$ |
|---|---|---|---|---|---|---|---|---|---|---|
| $2^4$ | $2^{64}$ | $2^{11}$ | 834 | 1 | $2^{23}$ | 1 | $2^3$ | 5 | $3.554 \cdot 10^{-6}$ | $2.845 \cdot 10^{-15}$ |

**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 | $\sigma_u^2 \cdot (n \cdot (\sigma_s^2 + \mu_s^2) + 1)$ | $2.08 \cdot 10^{-6}$ | $2.07 \cdot 10^{-6}$ | 1 |
| | Mean comp. | $\sigma_u^2 \cdot (n \cdot \sigma_s^2 + 1)$ | $1.04 \cdot 10^{-6}$ | $1.05 \cdot 10^{-6}$ | 0.507 |
| | Late rounding | $\sigma_u^2 \cdot (\frac{n}{2} \cdot 3 \cdot (\sigma_s^2 + \mu_s^2)^2 + 1)$ | $1.56 \cdot 10^{-6}$ | $1.56 \cdot 10^{-6}$ | 0.754 |
| | Optimal BKU[1] | / | $1.01 \cdot 10^{-6}$ | $0.995 \cdot 10^{-6}$ | 0.481 |
| KS | Standard | $\sigma_u^2 \cdot kN \cdot (\sigma_s^2 + \mu_s^2)$ | $7.95 \cdot 10^{-8}$ | $7.87 \cdot 10^{-8}$ | 1 |
| | Mean comp. | $\sigma_u^2 \cdot kN \cdot \sigma_s^2$ | $3.97 \cdot 10^{-8}$ | $3.99 \cdot 10^{-8}$ | 0.507 |
| BR | Standard | $\sigma_U^2 \cdot (kN \cdot (\sigma_S^2 + \mu_S^2) + 1) \cdot \frac{n}{2}$ | $5.06 \cdot 10^{-10}$ | $5.02 \cdot 10^{-10}$ | 1 |
| | Mean comp. | $\sigma_U^2 \cdot (kN \cdot \sigma_S^2 + 1) \cdot \frac{n}{2}$ | $2.53 \cdot 10^{-10}$ | $2.53 \cdot 10^{-10}$ | 0.504 |

[1] Optimal bootstrap key unrolling with factor 2

**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 | BR | | |
| ✗ | ✗ | ✗ | $2^{-64.30}$ | 1 |
| ✓ | ✗ | ✗ | $2^{-98.39}$ | 1.002 |
| ✓ | ✓ | ✗ | $2^{-100.47}$ | 0.998 |
| ✓ | ✓ | ✓ | $2^{-100.81}$ | 1.295 |

for modulus switching and both gadget decompositions, we reduce the noise variance to $1.82 \cdot 10^{-6}$ and the error probability to $2^{-100.81}$.

## 7.2 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 [Zam24] and computed bootstraps using the default parameter set from Table 3. 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.

**Table 6:** Computation time to evaluate two different functions for 64 different ciphertexts using programmable bootstrapping. The parameter sets were optimized for computation time, given the requirements for security and error probability. 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 |
| $\geq$ 80-bit | $\leq 2^{-40}$ | 4 | 117 | 59 | 2.00× | 59 | 2.00× |
| | | 5 | 267 | 129 | 2.07× | 142 | 1.89× |
| | $\leq 2^{-64}$ | 4 | 123 | 89 | 1.39× | 42 | 1.88× |
| | | 5 | 310 | 211 | 1.47× | 227 | 1.37× |
| | $\leq 2^{-128}$ | 4 | 144 | 124 | 1.16× | 133 | 1.08× |
| | | 5 | 899 | 420 | 2.14× | 457 | 1.97× |
| $\geq$ 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× |

### 7.3 Runtime improvements

Finally, we measure the improvements in execution time by combining mean compensation with optimal parameters. We report parameter sets with different restrictions. Table 6 reports the computation time for parameter sets with 80-bit or 128-bit security, and a maximum error probability of $2^{-40}$, $2^{-64}$, or $2^{-128}$. We report results for four or five encrypted bits, excluding the padding bit. We measure the time to compute two different functions for 64 different ciphertexts. This is a realistic usecase, ideal for manyLUT, that computes both functions simultaneously. A full overview of the results for one up to five encrypted bits can be found in Figure 2.

The results show that mean compensation allows to reduce computation time up to 2.14× in about half of the scenarios where the plaintext modulus is sufficiently high. As explained in Subsection 6.1, these results depend strongly on whether the reduction in noise variance reached a threshold value that allows us to significantly reduce the computational complexity. Moreover, we note that the combination between mean compensation and manyLUT often allows a second function evaluation with no additional cost.

## 8 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. This method comes with a negligible computational cost and reduces the error probability for a standard parameter set from $2^{-64.30}$ to $2^{-100.47}$. Furthermore, we combined mean compensation with bootstrap key unrolling, to further reduce the noise increase during modulus switching. This reduces the error probability to $2^{-102.73}$ or $2^{-107.59}$ 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 2.14×, for identical security and error probability requirements. Alternatively, for most requirements, we can evaluate two functions of one ciphertext with no additional cost by using manyLUT [CLOT21].

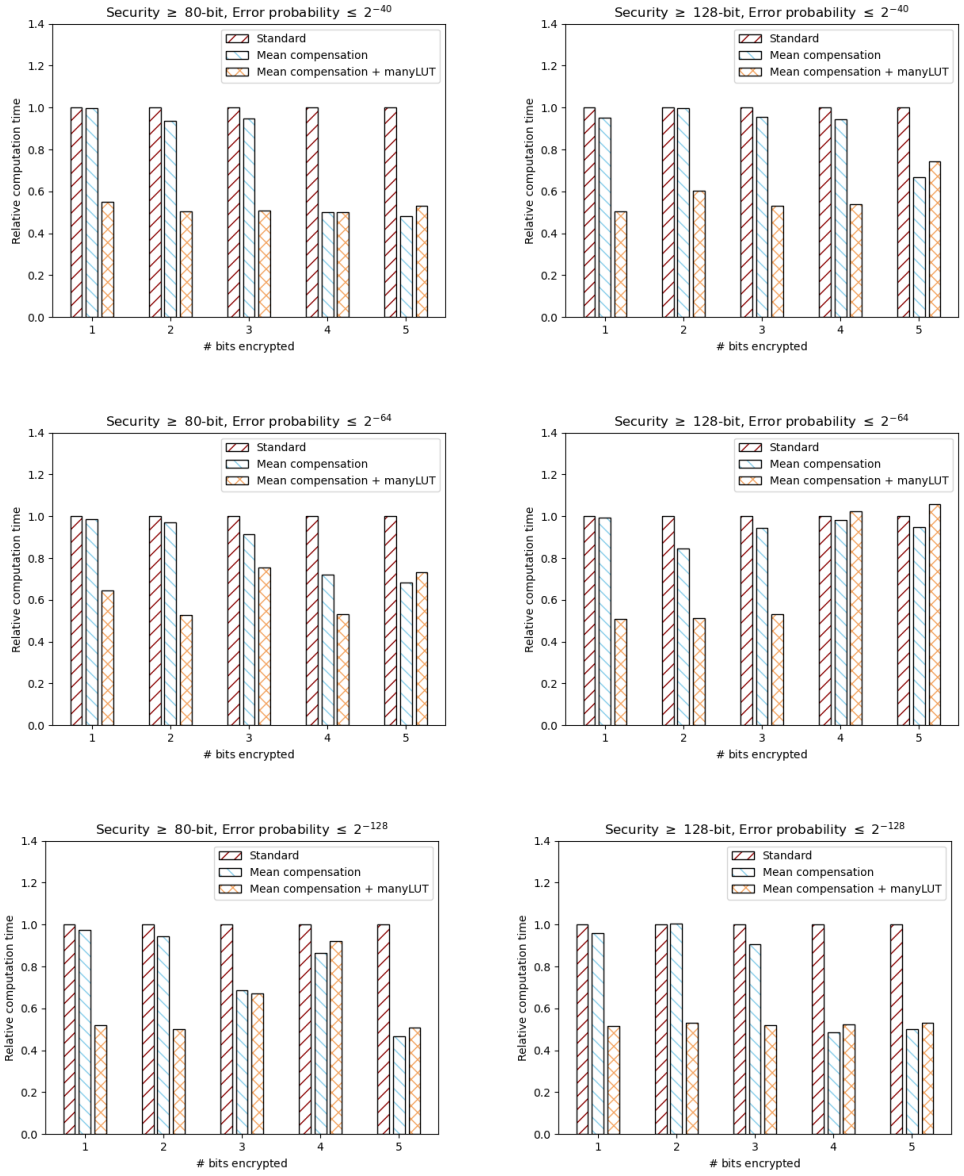
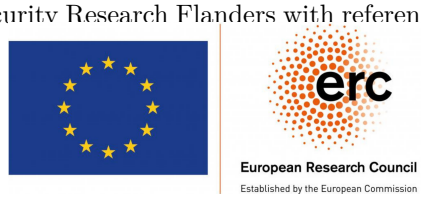



**Figure 2:** Relative computation time to evaluate two different functions for 64 different ciphertexts using programmable bootstrapping. The parameter sets were optimized for computation time, given the requirements for security and error probability. All benchmarks were run using eight Ryzen 7 pro 8840hs cores.

## 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. *J. Math. Cryptol.*, 9(3):169–203, 2015.

[BGV12] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, pages 309–325. ACM, 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 *ASIACRYPT 2022, Part II*, volume 13792 of *Lecture Notes in Computer Science*, pages 188–215. Springer, Cham, 2022.

[BJSW24] Olivier Bernard, Marc Joye, Nigel P. Smart, and Michael Walter. Drifting towards better error probabilities in fully homomorphic encryption schemes. Cryptology ePrint Archive, Paper 2024/1718, 2024.

[BJSW25] 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.

[BMMP18] Florian Bourse, Michele Minelli, Matthias Minihold, and Pascal Paillier. *Fast Homomorphic Evaluation of Deep Discretized Neural Networks: 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19–23, 2018, Proceedings, Part III*, pages 483–512. 07 2018.

[CGGI16] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 3–33. Springer, 2016.

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

[CKKS17] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. Homomorphic encryption for arithmetic of approximate numbers. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 409–437. Springer, 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, *Advances in Cryptology – ASIACRYPT 2021*, pages 670–699, Cham, 2021. Springer International Publishing.

[FV12] Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. *IACR Cryptol. ePrint Arch.*, 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 Transactions on Cryptographic Hardware and Embedded Systems*, 2022(4):661–692, Aug. 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 *Proceedings of the Thirty-Seventh Annual ACM Symposium on Theory of Computing*, pages 84–93. ACM, 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.

[Zam24] Zama. Zama tfhe-rs library, 2024. Available: 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.