A New Perspective on Key Switching for BGV-like Schemes

Johannes Mono ●
johannes.mono@rub.de
0000-0002-0839-058X

Tim Güneysu ●●
tim.gueneysu@rub.de
0000-0002-3293-4989

● Ruhr University Bochum, Bochum, Germany
● DFKI GmbH, Bremen, Germany

Fully homomorphic encryption is a promising solution for privacy-preserving computation. For BFV, BGV, and CKKS, three state-of-the-art fully homomorphic encryption schemes, the so-called key switching is one of the primary bottlenecks when evaluating homomorphic circuits. While a large body of work explores optimal selection for scheme parameters such as the polynomial degree or the ciphertext modulus, the realm of key switching parameters is relatively unexplored.

This work closes this gap, formally exploring the parameter space for BGV-like key switching. We introduce a new asymptotic bound for key switching complexity, thereby providing a new perspective on this crucial operation. We also explore the parameter space for the recently proposed double-decomposition technique by Kim et al. [24], which outperforms current state-of-the-art only in very specific circumstances. Furthermore, we revisit an idea by Gentry, Halevi, and Smart [19] switching primes in and out of the ciphertext and find novel opportunities for constant folding, speeding up key switching by up to 50 % and up to 11.6 %, respectively.

In recent years, the research community has made great progress toward making privacy-preserving computation a more realistic opportunity in everyday use. One approach is fully homomorphic encryption (FHE), in which a client encrypts its data and outsources the computation to a powerful server.

State-of-the-art schemes base security on the Learning with Errors over Rings (RLWE) assumption [28, 25], where a ciphertext has an associated error that grows during homomorphic evaluation. Once an error reaches a pre-determined threshold, it can be refreshed with bootstrapping, a process corresponding to an encrypted homomorphic decryption.

Currently, there are two strains of schemes differing in the type of underlying plaintext data: Boolean-based schemes encrypt single bits or small bit groups (FHEW [16], TFHE [11]), and arithmetic-based schemes encrypt word-sized data (BFV [5, 17], BGV [6], CKKS [10]). While the former enjoys fast bootstrapping and high computational flexibility due to the Boolean-based nature of the underlying data, the latter is more efficient for highly parallelizable arithmetic using a multi-level approach with slower bootstrapping.

Three arithmetic-based schemes are considered state-of-the-art: BFV and BGV for integer arithmetic and CKKS for approximate arithmetic. Due to their similarities, we will refer to these schemes and related concepts as BGV-like. The ring $\mathcal{R}_q = \mathbb{Z}_q[X]/\left(X^N + 1\right)$ builds the foundation of their ciphertext space, and the ciphertext modulus q is chosen large enough to accommodate the growing error of multiple homomorphic operations. These operations are split into levels, and scheme-specific error management techniques at a level boundary slow down error growth to delay the expensive bootstrapping process.

Usually, BGV-like schemes encrypt and operate on vectors of plaintext data with slot-wise additions and multiplications, additionally supporting rotations over the encrypted vectors. They provide two internal housekeeping facilities, one for error management differing in each scheme, and another called key switching. Key switching is very similar in all three schemes and requires two additional parameters: the key switching modulus P and the decomposition number ω.

BGV-like operations and housekeeping facilities are commonly implemented using two different Chinese Remainder Theorem (CRT) decompositions, also referred to as double CRT (DCRT) representation: the residue number system (RNS) decomposes the ciphertext modulus into ℓ co-prime primes q_i with $q = \prod q_i$ enabling word-sized arithmetic over each modulus q_i, and, for fast polynomial multiplication, the forward

and inverse number theoretic transform (NTT) are used. We also use the RNS representation for the key switching modulus P, decomposing it into k primes P_j.

The security level λ depends on the size of the modulus qP and the polynomial degree N. For a fixed degree N, increasing q (and thus qP) increases the available space for the error to grow, allowing for more computations before a bootstrapping becomes necessary, simultaneously decreasing the security level λ. For a fixed modulus qP, increasing N increases the security level λ, significantly increasing computation time and memory requirements.

Although there exists a relatively large body of work exploring parameter selection for the security level λ, polynomial degree N, and the ciphertext modulus q [2, 12, 1, 13, 27], the same cannot be said for the key switching parameters P and ω. While Kim, Polyakov, and Zucca [23] explore different methods of key switching and their complexity, they do not provide further exploration of the parameter space for key switching.

The lack of work on key switching parameters is even more surprising considering its heavy impact on performance; it currently is considered one of the main bottlenecks for BGV-like schemes [8, 18].[1] One exception is a recently published work at Crypto 2023 by Kim et al. [24], which proposes a new double-decomposition technique for state-of-the-art key switching. Though their work considers key switching complexity in general, they also do not explore the parameter space, drawing incorrect conclusions about the asymptotic complexity of key switching and the effectiveness of their approach.

In this work, we aim to close the current gap in analyzing key switching parameters and, more generally, improve the current state-of-the-art on key switching. More specifically, we make the following contributions:

- We provide the first formal analysis of the key switching parameters P and ω and show how to choose these parameters optimally. In the process, we generalize the analysis to arbitrary combinations of input and output domains (Subsection 3.1) and provide a new perspective on key switching with the bound $\mathcal{O}(\omega\ell)$ (Subsection 3.2).
- Using integer linear programming (ILP) techniques, we explore the parameter space for the newly proposed double-decomposition approach to key switching (Subsection 3.2). We implement the new approach with state-of-the-art techniques and show that the double-decomposition approach

1 For example, in a publicly available implementation of homomorphic matrix multiplication [26], profiling data shows that key switching is responsible for more than 50 % of execution time.

only outperforms the single-decomposition approach to key switching in specific circumstances (Subsection 4.4).

- We revisit an idea by Gentry, Halevi, and Smart [19] for mainly choosing large RNS primes, integrating the process of switching in smaller primes into key switching with low overhead (Subsection 3.3), speeding up key switching by up to 50 % (Subsection 4.1).
- We highlight new opportunities for constant folding in key switching, reducing the number of constant multiplications by up to $6\ell N + 4kN$ multiplications (Subsection 3.4) and speeding up key switching by up to 11.6 % (Subsection 4.2).

Preliminaries
Section 2

The following provides the necessary background to our work. We start by defining the notation used throughout the paper. Then, we introduce a shared context for the BGV-like schemes BFV, BGV, and CKKS and follow with short definitions for scheme-specific operations. Afterward, we describe the DCRT representation commonly used in BGV-like implementations and finalize the preliminaries with a unified view of modulus switching and key switching.

2.1 Notation

Let $q = \prod_{i=1}^{\ell} q_i$ be a product of primes with each q_i co-prime. For a subset of primes q_i, we write $q_{a,b} = \prod_{i=a}^{b} q_i$ for $a \leq b$ and $a, b \in \{1, \dots, \ell\}$. For a power-of-two N, a ciphertext consists of polynomials c_i in the ring $\mathcal{R}_q = \mathbb{Z}_q[X]/(X^N + 1)$.

With $[\cdot]_q$, we denote the centered modular reduction to $[-q/2, q/2)$, and, for polynomials, apply the operation for each coefficient. We sometimes omit explicit modular reduction if no ambiguity exists. We denote a generalized rounding operation over the integers by $\lceil a \rfloor_t$, that is, rounding to the closest integer such that $[a]_t = 0$. If not explicitly noted, we assume $t = 1$.

We denote with $s, u \leftarrow \chi_s$ and $e \leftarrow \chi_e$ sampling from the secret and error distributions, respectively, and, with a slight abuse of notation, write $a \leftarrow \mathcal{R}_q$ for a uniformly random sample from $\mathcal{R}_q$. Table 1 summarizes frequently used notation in the remainder of this work.

Polynomial norms

The infinity norm $\|a\|_\infty$ of a polynomial $a \in \mathcal{R}$ is the maximum absolute value of all its coefficients. The canonical embedding norm is defined as $\|a\|_{\mathsf{can}} = \max_{j \in \mathbb{Z}_{2N}^*} \|a(\xi^j)\|_\infty$ for all primitive $2N$-th roots of unity ξ^j with

λ	security level
N	polynomial degree
p	plaintext modulus
t	error scaling factor
$q_{1,\ell}$	ciphertext modulus
$P_{1,k}$	key switching modulus
ω	key switching decomposition number
b, β	bit size of q_i and P_j, respectively
B	upper bound on b and β
c_i	ciphertext polynomial

Table 1: Frequently used notation.

$\|a \cdot b\|_{\mathsf{can}} = \|a\|_{\mathsf{can}} \cdot \|b\|_{\mathsf{can}}$ for any $a, b \in \mathcal{R}$. For a power-of-two degree, $\|a\|_\infty \leq \|a\|_{\mathsf{can}}$ [15].

The canonical norm is bounded by $\|a\|_{\mathsf{can}} \leq D\sqrt{NV_a}$ for some D and the variance V_a. For a random polynomial $a \in \mathcal{R}_q$, $V_a \approx q^2/12$. Note that $V_{a+b} = V_a + V_b$, $V_{ab} = NV_aV_b$, and, for a constant γ, $V_{\gamma a} = \gamma^2 V_a$. For D, a common choice is $D = 6$ [12].

2.2 BGV-like schemes

We define a unified public key using the scheme-specific error scaling factor t as

$$\mathsf{pk} = ([as + te]_q, [-a]_q)$$

for $a \leftarrow \mathcal{R}_q$, $s \leftarrow \chi_s$, and $e \leftarrow \chi_e$. Using this public key, the individual encryption routines of the BGV-like schemes BFV, BGV, and CKKS output a ciphertext $(c_0, c_1) \in \mathcal{R}_q^2$.

A helpful perspective on a ciphertext is as polynomial $c(s) = c_0 + c_1 s$ in the secret key, and evaluating the ciphertext in the secret key results in $c(s) = m + te$, where m is a (possibly encoded) message and te is an (initially small) error that grows during homomorphic computation. The process of removing the error and decoding the message depends on the combination of encoding and error scaling, and we describe it in more detail in the respective scheme-specific sections.

Arithmetic operations nicely map to our polynomial representation: Addition on the underlying plaintext data of messages m and m', with corresponding ciphertexts $c = (c_0, c_1)$ and $c' = (c'_0, c'_1)$, respectively, requires a polynomial addition

$$c(s) + c'(s) = c_0 + c'_0 + (c_1 + c'_1)s = m + m' + te_{\text{add}},$$

constant multiplication with a constant scalar or polynomial γ results in

$$\gamma c(s) = \gamma c_0 + \gamma c_1 s = \gamma m + t e_{\text{const}},$$

and multiplication outputs the quadratic polynomial

$$c(s) \cdot c'(s) = c_0 c'_0 + (c_0 c'_1 + c_1 c'_0)s + c_1 c'_1 s^2 = mm' + t e_{\text{mul}}.$$

Usally, BGV-like schemes operate on multiple integers simultaneously; we encode up to N integers or approximate numbers in one message polynomial, depending on the specific parameters. Polynomial addition or multiplication corresponds to element-wise operations on the encoded plaintext data, which has a hypercube structure; applying a parameter- and dimension-specific permutation π on each ciphertext polynomial c_i rotates the hypercube along the chosen dimension.

After a rotation, the ciphertext is now encrypted under the permutation of the secret key $\pi(s)$, and we use a key switching to transform $c(s) = c_0 + c_1\pi(s)$ to $\tilde{c}(s) = c_0 + \tilde{c}_0 + \tilde{c}_1 s$, encrypting the same message for a known permutation π under the original secret key. For a multiplication, key switching enables transforming the output $c(s) = c_0 + c_1 s + c_2 s^2$ to a new polynomial $\tilde{c}(s) = c_0 + \tilde{c}_0 + (c_1 + \tilde{c}_1)s$, encrypting the same message. We define a unified approach to key switching in Subsection 2.8.

Choosing parameters

Security for BGV-like schemes is based on the RLWE assumption. It depends on the distributions χ_s and χ_e, the polynomial degree N, and the product qP of the ciphertext modulus with the key switching modulus. A great tool to estimate security is the Lattice Estimator [2], which, given the parameters above, estimates the time and memory costs of the best-known lattice attacks.

Common distribution choices are a uniform ternary distribution χ_s and a centered Gaussian distribution χ_e with variance $\sigma = 3.19$ [1]. In this work, we sometimes refer to a simplified perspective on secure parameter selection, in which only N and qP determine security, implicitly assuming fixed distributions χ_s and χ_e.

Choosing the remaining parameters for a given use case is an extensive research area [2, 12, 1, 13, 27]. The following is a simplified (and slightly inaccurrate) description; however, it suffices for our purposes: An upper bound on the error is defined for each homomorphic operation for a given input bound. After multiplication, a scheme-specific error management technique slows down error growth to delay the necessity for bootstrapping. We retrieve an upper bound on q by composing the upper bounds of each operation according to a circuit model of a target use case.

We then choose the smallest power-of-two degree N such that the security estimate for N and q is greater than λ. In most cases, the security

estimate will exceed λ; we use this security margin for the key switching modulus P, choosing ω accordingly, such that a security estimate on N and qP is still larger than λ. The above essentially delivers a power-of-two degree N and ciphertext modulus q for a given use case and security level, selecting key switching parameters afterward. Choosing optimal parameters can significantly reduce execution time and memory requirements.

2.3 The BFV scheme

The BFV scheme [5, 17, 23] is a state-of-the-art FHE scheme for integer arithmetic. Given a plaintext modulus p, the message polynomial $m \in \mathcal{R}_p$ is stored in the most significant bits of the ciphertext modulus encoded as $\lceil \frac{q}{p} m \rfloor$, while the error is stored in the least significant bits of the ciphertext modulus, that is, $t = 1$.

Encryption and decryption

Using the unified public key pk, we encrypt a message as

$$(c_0, c_1) = \left(\left[\mathsf{pk}_0 u + e_0 + \left\lceil \frac{q}{p} m \right\rfloor \right]_q, \left[\mathsf{pk}_1 u + e_1 \right]_q \right)$$

with $u \leftarrow \chi_s$, $e_0, e_1 \leftarrow \chi_e$, and decrypt with

$$m = \left[\left\lceil \frac{p}{q} \left[c_0 + c_1 s \right]_q \right\rfloor \right]_p$$

which is correct as long as the error is small enough. For a thorough analysis of error growth and security, we refer to the relevant works on the BFV scheme [5, 17, 23].

Error management

In BFV, the error management technique requires a slight modification to the unified perspective on multiplication in Subsection 2.2. Instead of operating on c and c' modulo q, we switch the modulus of the latter to another co-prime q' of approximately the same size. Then, after multiplication over the integers, a scaling and rounding $\left[\left\lceil \frac{p}{q'} cc' \right\rfloor \right]_q$ corrects the encoding factor [23].

In contrast to BGV and CKKS, we usually use the entire ciphertext modulus q throughout the complete circuit (temporarily operating modulo q' or qP); hence, BFV is also called scale-invariant, and the specific size of each q_i has negligible impact on error growth.

2.4 The BGV scheme

The BGV scheme [6, 23] is another state-of-the-art scheme for integer arithmetic. Contrary to BFV, the message polynomial $m \in \mathcal{R}_p$ is stored in the least significant bits of the ciphertext modulus, and the error sits

above the message, that is, $t = p$. BFV and BGV are, in fact, so similar that we can convert between ciphertexts with a simple scalar multiplication [3].

Encryption and decryption

We encrypt a message in BGV as

$$(c_0, c_1) = \left(\left[\mathsf{pk}_0 u + p e_0 + m \right]_q, \left[\mathsf{pk}_1 u + p e_1 \right]_q \right)$$

with random samples $u \leftarrow \chi_s$, $e_0, e_1 \leftarrow \chi_e$, and decryption computes as

$$m = \Big[\left[c_0 + c_1 s \right]_q \Big]_p .$$

As in BFV, the error needs to be small enough for correctness, and we refer to the relevant literature on the BGV scheme for details on error growth and security [6, 23, 27].

Error management

In BGV, the error after multiplication is reduced by scaling the ciphertext with a part of the ciphertext modulus itself, consuming RNS primes q_i in the process. This process, modulus switching, scales the ciphertext, including the associated error, by q_i, while keeping the message the same.

For example, consuming the ciphertext modulus prime q_ℓ scales the error by $1/q_\ell$, and afterward, ciphertexts live in the space $\mathcal{R}_{q_{1,\ell-1}}$. Therefore, the size of the individual q_i significantly impacts error growth during homomorphic evaluation and is highly relevant during parameter selection.

2.5 The CKKS scheme

Although a CKKS message $m \in \mathcal{R}$ consists of integers, we assume that an approximate result is good enough, such as with fixed-point arithmetic. Hence, we can consider the homomorphic error as part of the message itself, and both are stored in the least significant bits of the ciphertext modulus ($t = 1$). We refer to the relevant works on CKKS investigating security, error growth, and approximation error [10, 9, 22].

Encryption and decryption

A CKKS encryption is the tuple

$$(c_0, c_1) = \left(\left[\mathsf{pk}_0 u + e_0 + m \right]_q, \left[\mathsf{pk}_1 u + e_1 \right]_q \right)$$

with $u \leftarrow \chi_s$, $e_0, e_1 \leftarrow \chi_e$, and, for decryption, we compute

$$m' = \left[c_0 + c_1 s \right]_q$$

recovering the approximate message m'.

Error management

In CKKS, we use a conceptually similar approach as in BGV, serving a different purpose on the underlying plaintext data. As in BGV, we scale by the individual q_i, operating on ciphertext polynomials in $\mathcal{R}_{q_{1,i-1}}$ afterward.

There are two major differences: First, a small approximation error is acceptable as we are only interested in an approximate decryption. Second, the encrypted message moves to the more significant bits of the ciphertext modulus during multiplication, and instead of scaling the error, we move the message back to the least significant bits. As in BGV, choosing appropriate q_i is crucial to ensure correctness and a small approximation error.

2.6 The DCRT representation

The most common approach for implementing BGV-like schemes is using the double CRT (DCRT) representation. We apply two Chinese Remainder Theorem (CRT) decompositions on a polynomial: the RNS for word-sized arithmetic modulo q and the NTT for fast polynomial multiplication.

Residue number system

The RNS decomposes a number in $\mathbb{Z}_q$ toward each prime q_i. Instead of using one polynomial in $\mathcal{R}_q$, we use ℓ polynomials with $a_i = [a]_{q_i} \in \mathcal{R}_{q_i}$ for $a \in \mathcal{R}_q$. We also use the RNS for $P_{1,k}$. The CRT then reconstructs the original polynomial a based on the individual a_i.

In the RNS, we can perform additions and (constant) multiplications as before. However, a limitation is that division and the modulo operator do not natively map to the RNS space, requiring other approaches. For division, there is a notable exception: If a constant $\gamma \in \mathbb{Z}$ divides each coefficient of $a \in \mathcal{R}_q$, division by γ over $\mathcal{R}$ corresponds to multiplication with the inverse $\gamma^{-1} \bmod q$.

We can convert a polynomial $a_i \in \mathcal{R}_{E_i}$ between two arbitrary RNS bases $E_{1,n}$ and $E'_{1,n'}$ with a fast base extension, only using word-sized arithmetic. The fast base extension is defined as

$$\mathsf{BaseExt}(a, E, E') = \left(\sum_{i=0}^{n} \left[a \frac{E_i}{E_{1,n}} \right]_{E_i} \frac{E_{1,n}}{E_i} \bmod E'_j \right)_{j=1}^{n'}$$

and outputs $a + \varepsilon E$ in the base $E'_{1,n'}$ for a small ε. Under certain circumstances, we can consider εE as part of the homomorphic error. Otherwise, we remove it using an error correction method such as BEHZ [4] or HPS [20].

Number theoretic transform

The forward and inverse NTT are variants of the Fast Fourier Transform over a finite field evaluating a polynomial in the $2N$-th roots of unity ξ^j in time $\mathcal{O}(N \log N)$. For $\mathcal{R}_{q_i}$, we require $q_i = 1 \pmod{2N}$, and the NTT corresponds to the factorization $\prod(X - \xi^j) \bmod q_i$ into linear terms; another perspective on the factorization is as generalized CRT, hence the name DCRT representation.

For two polynomials in the NTT domain, we compute their product in time $\mathcal{O}(N)$ using coefficient-wise multiplication. Overall, NTT-based polynomial multiplication has an asymptotic complexity of $\mathcal{O}(N \log N)$.

Due to the linearity of the NTT, polynomial addition and constant multiplication can be performed in either domain. However, when interacting with polynomials of other primes, such as during the fast base extension, we require a polynomial to be in the coefficient domain, requiring costly inverse and forward NTT operations.

2.7 Modulus switching

We define a unified version of modulus switching for $a \in \mathcal{R}_q$ to $a' \in \mathcal{R}_{q'}$ for all BGV-like schemes based on the generalized rounding as

$$a' = \left[\left\lceil \frac{q'}{q} a \right\rfloor_t\right]_{q'} = \left[\frac{q'a + \delta}{q}\right]_{q'} = \left[\frac{q'a - t\left[t^{-1}q'a\right]_q}{q}\right]_{q'}.$$

The above is RNS-friendly since, by definition, $q \mid (q'a + \delta)$, and multiplying with the multiplicative inverse corresponds to the required division.

For $t = 1$, this scales the encrypted data in the least significant bits by roughly q'/q (the error for BFV and the approximate message for CKKS), and, for BGV with $t = p$, we have $-t\left[t^{-1}q'a\right]_q = 0 \bmod p$, only scaling the error by q'/q and keeping the encrypted message intact.

2.8 Key switching

Key switching transforms a single BGV-like ciphertext monomial $c_i s'$ to a polynomial

$$c_i s' = \tilde{c}_0 + \tilde{c}_1 s + te$$

for a small error e. The following summarizes the current state-of-the-art on key switching [23, 24].

We need to generate key switching keys, and each distinct s' requires a key switching key. For example, key switching after a multiplications needs one key switching key with $s' = s^2$, while a fixed permutation π requires another key switching key with $s' = \pi(s)$.

A BGV-like key switching key is the tuple

$$\mathsf{ksw} = \left(\left[as + te + s'\right]_q, \left[-a\right]_q\right)$$

for $a \leftarrow \mathcal{R}_q$, $e \leftarrow \chi_e$. For naïve key switching, we compute

$$\begin{aligned} c_i\mathsf{ksw}(s) &= c_i\mathsf{ksw}_0 + c_i\mathsf{ksw}_1 s \pmod q \\ &= c_i s' + t c_i e \pmod q \end{aligned}$$

Note, however, that the error tc_ie is, in fact, not small as initially claimed. Hence, we need to modify key switching to reduce this error term.

Single-decomposition technique

Current state-of-the-art, which we also refer to as the single-decomposition technique, employs two different approaches to control the error: decomposing the ciphertext with respect to the RNS primes q_i, reducing the bound to the decomposition [7] and temporarily operating on an extended modulus qP, scaling the error by P afterward [19]. Kim, Polyakov, and Zucca provide an excellent summary of both approaches in the extended version of their work, which we recommend for a more in-depth description [23].

For a decomposition $\mathcal{D}(\cdot)$ and its inverse $\mathcal{D}^{-1}(\cdot)$ and $a, b \in \mathcal{R}_{qP}$, we require $ab = \langle \mathcal{D}(a), \mathcal{D}^{-1}(b) \rangle$. In DCRT-based implementations, we decompose towards the RNS basis, dividing the primes into ω groups $q_{\mathcal{D}_j}$ with up to $\lceil \ell/\omega \rceil$ primes in each group and use the CRT as $\mathcal{D}^{-1}(\cdot)$. Computing $\mathcal{D}(a)$ corresponds to $[a]_{q_i}$ for $q_i \in q_{\mathcal{D}_j}$ and is free [20].

We define a BGV-like key switching key as

$$\mathsf{ksw} = \left(\left[a_j s + t e_j + P\mathcal{D}_j^{-1}(s') \right]_{qP}, [-a]_{qP} \right)_{j=1}^{\omega}$$

where $a \leftarrow \mathcal{R}_{qP}^{\omega}$, $e \leftarrow \chi_e^{\omega}$, and $\mathcal{D}_j^{-1}(\cdot)$ is the jth entry of $\mathcal{D}^{-1}(\cdot)$. Slightly abusing notation, we switch keys with

$$\left\lceil \frac{[\langle \mathcal{D}(c_i), \mathsf{ksw} \rangle (s)]_{qP}}{P} \right\rfloor_t = c_i s' + \left\lceil \frac{[t \langle \mathcal{D}(c_i), e \rangle]_{qP}}{P} \right\rfloor_t .$$

In practice, the resulting error is negligible for $k = \lceil \ell/\omega \rceil$ [23].

Kim, Polyakov, and Zucca [23] analyze the complexity of key switching for the number of NTT operations (the most costly building block of key switching), the number of multiplications, and the memory requirements for a key switching key in bits. However, they do not further explore the parameter space or discuss further implications for parameter selection. We summarize their results in Table 2.

For a correct implementation, we also have to consider the fast base extension as well as forward and inverse NTT operations. For completeness, we also take into account that we add the output of key switching to (parts of) an existing ciphertext. For example, consider BFV (input and output in the coefficient domain).

Metric	Scheme	Cost
`ntt`	BFV/BGV	$(\omega+2)(\ell+k)$
`mul`	BFV	$\ell(\ell+2\omega+2k+5)+2k$
`mul`	BGV	$\ell(\ell+2\omega+2k+7)+4k$
bit size	BFV/BGV	$2\omega N \log qP$

Table 2: Single-decomposition key switching complexity according to the analysis of Kim, Polyakov, and Zucca [23] in terms of forward and inverse NTT (`ntt`) as well as modular (constant) multiplications (`mul`) with input and output in the same domain, coefficient for BFV and NTT for BGV. Complexity $\mathcal{O}(N \log N)$ for `ntt` and $\mathcal{O}(N)$ for `mul` are implicit.

1 Ciphertext extension: Perform a fast base extension on the ω ciphertext decompositions modulo $q_{\mathcal{D}_j}$ as

$$c_{\mathsf{ext}} = \mathsf{BaseExt}(\mathcal{D}_j(c_i), q_{\mathcal{D}_j}, qP).$$

2 Dot product: With a key switching key in the NTT domain, perform the dot product as

$$c'_i = \langle \mathsf{NTT}_{\mathsf{fwd}}(c_{\mathsf{ext}}), \mathsf{ksw}_i \rangle \pmod{qP}.$$

3 Delta computation: Compute δ for scaling by P as

$$\delta_i = -t\,\mathsf{BaseExt}\left(\mathsf{NTT}_{\mathsf{inv}}\left(\left[t^{-1}c'_i\right]_P\right), P, q\right) \pmod{q}.$$

4 Modulus switching: Scale down $c'_i + \delta_i$ and add the result to the input as

$$\left(c_0 + \frac{\mathsf{NTT}_{\mathsf{inv}}(c'_0)+\delta_0}{P}, c_1 + \frac{\mathsf{NTT}_{\mathsf{inv}}(c'_1)+\delta_1}{P}\right).$$

When we switch for c_1, for example, after a rotation, we set $c_1 = 0$ in this final step.

Double-decomposition technique

Kim et al. recently proposed a double-decomposition technique building upon single-decomposition key switching [24]. The technique only changes the algorithmic approach to key switching and does not influence the error.

Their idea is as follows: In the second step, instead of computing the dot product $\langle \mathcal{D}(c_i), \mathsf{ksw} \rangle$ in $\mathcal{R}_{qP}$, we add a second decomposition over qP into $\tilde{\omega}$ groups such that each group has up to $\lceil (\ell+k)/\tilde{\omega} \rceil$ primes. Then, switching to a shared RNS basis $E = \{E_1, \ldots, E_r\}$ and computing the dot product in $\mathcal{R}_E$ can improve execution time. For more details, we refer to the original publication [24].

Metric	Cost
`ntt`	$(\omega + 2\tilde{\omega})r$
`mul`	$(3\ell + 2\omega\tilde{\omega} + 2k + 2)r + \ell(2k + 7) + 6k$
bit size	$2\omega\tilde{\omega}N \log E$

Table 3: Double-decomposition key switching complexity, including modulus switching costs, in terms of forward and inverse NTT (`ntt`) as well as modular (constant) multiplications (`mul`) for input and output in the coefficient domain.

In their evaluation, the authors assume CKKS with input and output in the coefficient domain and without the modulus switching costs for scaling down by P. We list their results, including modulus switching costs, in Table 3 to keep consistency with Kim, Polyakov, and Zucca [23].

For an implementation, again considering input and output in the coefficient domain, we compute

1 Ciphertext extension:

$$c_{\text{ext}} = \mathsf{BaseExt}(\mathcal{D}_j(c_i), q_{\mathcal{D}_j}, E).$$

2 Dot product:

$$\begin{aligned} c_i'' &= [\langle \mathsf{NTT}_{\mathsf{fwd}}(c_{\text{ext}}), \mathsf{ksw}_i \rangle]_E \\ c_i' &= \mathsf{BaseExt}(\mathsf{NTT}_{\mathsf{inv}}(c_i''), E, qP). \end{aligned}$$

3 Delta computation:

$$\delta_i = -t\, \mathsf{BaseExt}\left(\left[t^{-1}c_i'\right]_P, P, q\right) \pmod{q}.$$

4 Modulus switching:

$$\left(c_0 + \frac{c_0' + \delta_0}{P}, c_1 + \frac{c_1' + \delta_1}{P}\right).$$

As for the single-decomposition technique, when switching c_1, we set $c_1 = 0$ in this final step.

In contrast to the single-decomposition technique, where we can always consider the error as part of the ciphertext error [27], the fast base extension in step two of the double-decomposition technique requires error correction.

Contributions
Section 3

We describe our main contributions in the following, generalizing key switching complexity to arbitrary input and output domains. Afterward,

we formulate our new perspective on key switching, show how to use primarily large primes with small overhead, and introduce our new ideas for multiplication folding. We conclude with a short analysis of memory costs for key switching.

3.1 Generalizing key switching complexity to input and output domains

One limitation of current analyses is the coupling with a specific scheme such as BFV or BGV. Instead, we can analyze key switching for a given input and a target output domain, generalizing analysis to all BGV-like schemes.

For example, in BFV, the modulus switching before a multiplication requires input in the coefficient domain, and the scaling and rounding afterward results in output in the coefficient domain. Therefore, the most common input and output domain in BFV is the coefficient domain; however, depending on use-case-specific circumstances, one might also want to work with input or produce output in the NTT domain.

In contrast, in BGV and CKKS, multiplication requires input in the NTT domain and produces output in the NTT domain, and thus, the most common input and output domain for key switching is the NTT domain. The curious reader might wonder about rotations: For BGV-like schemes, efficient variants exist for permuting the ciphertext polynomials in either domain, and homomorphic rotations do not influence the common domain for a specific scheme.

Single-decomposition technique

Kim, Polyakov, and Zucca [23] show that ciphertext extension, dot product, and delta computation require $\omega(\ell + k) + 2k$ NTT operations independent of the input domain. Then, for modulus switching, we compute

$$\left(c_0 + \frac{c_0' + \delta_0}{P},\ c_1 + \frac{c_1' + \delta_1}{P}\right),$$

where c_i is the ciphertext input, c_i' the output of the dot product in the NTT domain, and δ_i the output of the delta computation in the coefficient domain. For rotations, we set $c_1 = 0$.

For a matching input and output domain, it is relatively straightforward to get the desired result with 2ℓ additional NTT operations and an overall complexity of $(\omega + 2)(\ell + k)$: For c_i in the coefficient domain, we apply ℓ inverse NTTs on each c_i', one for each ciphertext prime, while for c_i in the NTT domain, we apply ℓ forward NTTs on each δ_i.

For a non-matching input and output domain, we can achieve the same complexity concerning NTT operations with

$$\left(\frac{Pc_0 + c_0' + \delta_0}{P},\ \frac{Pc_1 + c_1' + \delta_1}{P}\right),$$

Operation	Domain		Cost
Rotation	`coef`	`coef`	$(\omega + 2\tilde{\omega})r$
	`coef`	`ntt`	$(\omega + 2\tilde{\omega})r + 2\ell$
	`ntt`	`coef`	$(\omega + 2\tilde{\omega})r + 2\ell$
	`ntt`	`ntt`	$(\omega + 2\tilde{\omega})r + 3\ell$
Multiplication	`coef`	`coef`	$(\omega + 2\tilde{\omega})r$
	`coef`	`ntt`	$(\omega + 2\tilde{\omega})r + 2\ell$
	`ntt`	`coef`	$(\omega + 2\tilde{\omega})r + 3\ell$
	`ntt`	`ntt`	$(\omega + 2\tilde{\omega})r + 3\ell$

Table 4: Double-decomposition key switching complexity regarding NTT operations generalized to arbitrary input and output domains.

scaling c_i by P. Depending on the input domain, we add Pc_i to c'_i or δ_i, afterward applying 2ℓ forward or inverse NTT operations as required for the desired output domain. While this increases the number of multiplications for now, we show in Subsection 3.4 how to avoid these additional costs.

Double-decomposition technique

The double-decomposition technique assumes input and output in the coefficient domain. For an input in the NTT domain, we need ℓ initial inverse NTTs for c_1 (after a rotation) or c_2 (after a multiplication) before ciphertext extension.[2]

Modulus switching also computes as

$$\left(c_0 + \frac{c'_0 + \delta_0}{P},\ c_1 + \frac{c'_1 + \delta_1}{P}\right);$$

however, in contrast to before, c'_i and δ_i are both in the coefficient domain. For output in the NTT domain, we thus require 2ℓ forward NTTs for $c'_i + \delta_i$ (possibly adding Pc_i for input in the coefficient domain).

For input in the NTT domain and output in the coefficient domain, we apply ℓ inverse NTTs for c_0 and, for multiplications, another ℓ inverse NTTs for c_1. We summarize the generalized complexity of the double-decomposition technique in Table 4.

3.2 A new perspective on key switching

In their recent work, Kim et al. [24] claim an asymptotic key switching complexity of $\mathcal{O}(\ell^2)$ for the single-decomposition technique and of $\mathcal{O}(\ell)$

2 These additional inverse NTT operations can be avoided in the single-decomposition technique; for the precise details, we refer to the in-depth description of Kim, Polyakov, and Zucca [23] and our accompanying key switching implementation.

for their technique. Their reasoning is as follows: By choosing the parameters $k, r \in \mathcal{O}(1)$, thus $\omega, \tilde{\omega} \in \mathcal{O}(\ell)$, the bounds follow accordingly. However, this implicitly limits the parameter space for k and r, which results in a skewed perspective on key switching.

Single-decomposition technique

In this work, we choose a different perspective on key switching. We consider $\omega \leq \ell$ as a parameter in the security level, which can be set individually. Assuming $b \approx \beta \approx B$ and, for simplicity, $\omega \mid \ell$, the number of primes in the key switching modulus follows as $k = \ell/\omega$ (see also Subsection 2.8).

Then, the number of NTT operations is

$$(\omega + 2)(\ell + k) = \omega\ell + 3\ell + \frac{2\ell}{\omega}.$$

For $\omega_2 = 2, \omega_1 = 1$,

$$\omega_2\ell + 3\ell + \frac{2\ell}{\omega_2} = \omega_1\ell + 3\ell + \frac{2\ell}{\omega_1},$$

and for $\omega_2 > \omega_1 > 1$,

$$\omega_2\ell + 3\ell + \frac{2\ell}{\omega_2} > \omega_1\ell + 3\ell + \frac{2\ell}{\omega_1}.$$

A simple fact follows: Increasing ω increases the computational complexity of key switching, and for better performance, we want to choose $k \in \mathcal{O}(\ell/\omega)$. This results in an asymptotic key switching complexity of

$$\omega\ell + 3\ell + \frac{2\ell}{\omega} = \mathcal{O}(\omega\ell).$$

While the above assumes a fixed polynomial degree, we can set ω as we please, choosing N securely for qP afterward; however, we then have to consider the complexity of the NTT operation itself. Although using the bound $\mathcal{O}(N \log N)$ at face value does not quite match reality due to the hidden factors, it is sufficient for the following argument.

An optimal choice of $\omega = 2$ results in 6ℓ NTT operations for key switching. If N has to be increased to the next power-of-two degree to match a given security level, the count of NTT operations increases to $12\ell N \log(N + 1) \approx 12\ell N \log N$. Therefore, increasing the polynomial degree to reduce key switching costs is approximately worth it once

$$\left(\omega\ell + 3\ell + \frac{2\ell}{\omega}\right) N \log N > 12\ell N \log N$$

and thus $\omega^2 - 9\omega + 2 > 0$, and $\omega > 8.77$ follows.

However, increasing the polynomial degree does increase the computational complexity of all other homomorphic computations, as these do not depend on the key switching parameters. We evaluate the above in Subsection 4.3 and discuss the implications for implementations holistically in Subsection 5.2.

Double-decomposition technique

Extending the new perspective to the double-decomposition technique requires estimating the number of primes r in E. In the original work, the authors use the infinity norm; we use the canonical norm for a better estimate.

Assuming $b \approx \beta \approx B$, E needs to be large enough to hold a product $ab \in R$ with $V_a \approx 2^{\ell/\omega b}$ and $V_b \approx 2^{(\ell+k)/\tilde{\omega} b}$. Then, for $N = 2^n$, we can roughly upper bound $\log E$ as

$$\log\left(D\sqrt{NV_{ab}}\right) = (n-1) + (\ell/\omega + \ell/\tilde{\omega} + \ell/(\omega\tilde{\omega}))b.$$

With n/b negligible in practice, a reasonable estimate for the number of primes in E is

$$r = \frac{\omega\ell + \tilde{\omega}\ell + \ell}{\omega\tilde{\omega}},$$

and we want to minimize the term $(\omega + 2\tilde{\omega})r$ for minimum complexity regarding the number of NTT operations.

However, minimizing this term for ω and $\tilde{\omega}$ over $\mathbb{R}$ results in values outside the desired target range, and we use integer linear programming (ILP) to find optimal parameters. For $\ell \leq 200$, a generous bound on the number of ciphertext primes, $\omega = \ell$, and hence $\tilde{\omega} = \sqrt{\ell(\ell+1)/2}$, results in the minimum number of NTT operations.

3.3 Choosing (mostly) large RNS primes

So far, our analysis assumes $b \approx \beta \approx B$, that is, RNS primes for q and P (and E) close to the maximum size B. In general, choosing the RNS primes as large as possible is beneficial for two apparent reasons:

- Each additional prime increases the number of polynomial operations during homomorphic evaluation. The larger each prime, the fewer primes we need, reducing computation time and memory costs.
- Assuming a maximum prime size of B bit, using fewer bits usually wastes computational and memory resources as operations are still performed over B-sized numbers.

Given a bound $\log q$ (the same idea extends to P and E, respectively), we would like to choose primes as follows: Compute $\ell = \lceil \log q/B \rceil$, choose b such that $\log q \approx \ell \cdot b$, and generate ℓ primes close to 2^b. However, for BGV and CKKS, the size of q_i significantly impacts the error growth, and scaling by roughly 2^b during modulus switching is not necessarily the optimal choice. Note that no such limitations exist for P and E, and the above approach always works.

To introduce the desired flexibility, we revisit an idea by Gentry, Halevi, and Smart and add a few small primes, which we constantly switch

in and out of the modulus during homomorphic evaluation [19]. Since their work only describes the high-level idea, we introduce some additional notation for the RNS setting, integrate their idea with key switching, and analyze its complexity.

Switching primes in and out

We choose $\ell - \kappa$ primes close to 2^b and $\kappa > \mu$ smaller primes with their product close to $2^{\mu b}$, such that $\log q \approx (\ell + \mu)b$. During homomorphic evaluation, we use a small prime for modulus switching. After κ modulus switchings, we replace any μ large primes with the κ small primes, restoring our capabilities for scaling with small primes.

If we want to replace the primes $\{q_{\ell-\mu}, \dots, q_\ell\}$ with $\{q_1, \dots, q_\kappa\}$, we can switch the modulus of $a \in R_{q_{\kappa+1,\ell}}$ as

$$\begin{aligned}\left\lceil \frac{q_{1,\ell-\mu}}{q_{\kappa+1,\ell}} a \right\rfloor_t &= \frac{q_{1,\ell-\mu}a - t[t^{-1}q_{1,\ell-\mu}a]_{q_{\kappa+1,\ell}}}{q_{\kappa+1,\ell}} \pmod{q_{1,\ell-\mu}} \\ &= \frac{q_{1,\kappa}a - t[t^{-1}q_{1,\kappa}a]_{q_{\ell-\mu,\ell}}}{q_{\ell-\mu,\ell}} \pmod{q_{1,\ell-\mu}}.\end{aligned}$$

Integrating prime switching with key switching

We can integrate prime switching during the modulus switching step of key switching by simply switching out the large primes $\{q_{\ell-\mu}, \dots, q_\ell\}$ in addition to the primes $\{P_1, \dots P_k\}$. Switching for $c_i' \in R_{q_{\kappa+1,\ell}P_{1,k}}$ transforms the above to

$$\frac{q_{1,\kappa}c_i' - t[t^{-1}q_{1,\kappa}c_i']_{q_{\ell-\mu,\ell}P_{1,k}}}{q_{\ell-\mu,\ell}P_{1,k}} \pmod{q_{1,\ell-\mu}}.$$

Compared to switching out only $P_{1,k}$ and switching in no primes, the number of inverse NTT operations increases from $2k$ to $2(\mu + k)$ for base extending $\delta_i = -t[t^{-1}q_{1,\kappa}c_i']_{q_{\ell-\mu,\ell}P_{1,k}}$. However, we also save 2μ NTT operations on either c_i' or δ_i since we reduce the number of primes for the modulus switching output from ℓ to $\ell - \mu$.

For output in the coefficient domain, switching primes in and out requires no additional NTT operations. For output in the NTT domain, we need to perform 2κ additional forward NTT operations (κ operations per δ_i). The former is free regarding the number of NTT operations, and, with $\kappa \in \mathcal{O}(1)$, the latter has only a small overhead.

Choosing the primes

Based on this more generic setup, we adjust the process for choosing the primes q_i as follows: Compute the number of RNS primes $\ell' = \lceil \log q/B \rceil$, and choose b such that $\log q \approx \ell' \cdot b$. Then, we generate $\ell' - \mu$ primes close to 2^b and κ primes close to $2^{\mu b/\kappa}$. Overall, this results in $\ell = \ell' - \mu + \kappa$ ciphertext primes.

Compared to the more naïve approach, that is choosing all primes of size $\mu b/\kappa$, we reduce ℓ as long as

$$\left\lceil \frac{\log q}{\mu b/\kappa} \right\rceil \approx \left\lceil \frac{\ell'}{\mu/\kappa} \right\rceil > \ell \;\Leftrightarrow\; \kappa\ell' - \mu\ell \geq \mu \;\Leftrightarrow\; \ell \geq \kappa + \frac{\mu}{\kappa - \mu}.$$

For example, with $B = 60$, we consider a use-case scaling each level with 36-bit primes; then, choosing $\mu = 2$ and $\kappa = 3$ results in $b = 54$, which reduces the overall number of primes as soon as $\ell \geq 5$ (equivalent to $\log q > 144$). Evaluating the above idea generically for all parameter settings is rather difficult due to the use-case-specific nature of parameters for BGV-like schemes; however, choosing primes as large as possible with a small number of additional scaling factors can also be adapted to more complex scenarios.

3.4 Folding multiplications in key switching

We split multiplication types in key switching into four context-based groups: (1) coefficient-wise multiplication with the key switching key during the dot product, (2) multiplications with a constant such as P^{-1} during modulus switching, (3) multiplication with $E_i/E_{1,n}$ modulo E_i during base extension, for example, for $E = P$, and (4) multiplication with $E_{1,n}/E_i$ modulo E'_j during base extension. Note that, for group (1), we can only fold fixed values such as t or P since the ciphertext modulus can change as in BGV or CKKS.

For brevity, we only outline our optimizations for the single-decomposition technique with input and output in the coefficient domain. However, these ideas also apply to other combinations of input and output domain, the double-decomposition technique, and key switching when replacing primes; we refer the interested reader to our implementation, where all folded variants are implemented and tested.

Our crucial observation is the following: After the dot product, we use $c' = (c'_0, c'_1) \bmod P$ only to compute $\delta = (\delta_0, \delta_1)$ and use $c' \bmod q$ only for switching the modulus. Thus, we can merge known multiplications (group 2–4) for $c' \bmod q$ with the key switching key $\mathsf{ksw} \bmod q$ and with $\mathsf{ksw} \bmod P$ separately (group 1). Additionally, for the delta computation, we can merge the multiplication by $-t$ over q_i (group 2) with the second part of the base extension from P to q (group 4).

Below, we list our algorithmic modifications to the key switching algorithm starting at the dot product. We exclude the required NTT operations, as constant multiplications can be performed in either domain:

$[\cdot]_q$	$[\cdot]_P$
$c'_{\text{fold}} = \langle c_{\text{ext}}, P^{-1}\mathsf{ksw}\rangle$	$\delta'_{\text{fold}} = \langle c_{\text{ext}}, (tP/P_j)^{-1}\mathsf{ksw}\rangle$
$\delta_{\text{fold}} = P^{-1}tP/P_j\ \delta'_{\text{fold}}$	
$c + c'_{\text{fold}} + \delta_{\text{fold}}$	

For the single-decomposition technique, this reduces the number of constant multiplications in key switching down to a unified bound of

$$\ell\left(\ell + 2\omega + 2\frac{\ell}{\omega} + 3\right)$$

for any t and any combination of input and output domain, each multiplication with complexity $\mathcal{O}(N)$.

For a given N and ℓ, finding the minimal number of multiplications depends only on $2\omega + 2\ell/\omega$, which is minimal for $\omega = \sqrt{\ell}$. Due to the implicit scaling by P^{-1}, we can add c without additional multiplications to either c'_{fold} in the NTT domain or to δ_{fold} in the coefficient domain. During precomputation, we trivially simplify $P^{-1}tP/P_j = t/P_j$.

Our folding ideas also apply to the fast base extension when correcting the error with the HPS method, which we use for implementation. Thus, we also optimize the double-decomposition technique in a similar fashion; for more details, we refer to our implementation.

Overall, the folded variant of the double-decomposition technique requires $(\ell + 2\omega\tilde{\omega})r + \ell(2k+3)$ multiplications, which, using the estimates on k and r, transforms to

$$\ell\left(2\omega + 2\tilde{\omega} + \frac{3\ell}{\omega} + \frac{\ell}{\tilde{\omega}} + \frac{\ell}{\omega\tilde{\omega}} + 5\right).$$

As with NTT complexity, we use ILP techniques to minimize this expression for $\ell \leq 200$, resulting in optimal choices for ω and $\tilde{\omega}$ close to $\sqrt{\ell}$ to minimize multiplication complexity.

3.5 Analyzing memory costs for key switching

In the single-decomposition technique, each key switching key requires ω polynomial pairs for the modulus qP. Thus, assuming B bits of storage for each prime, the size of one key switching key requires

$$2\omega\,(\ell + k)\,NB = 2(\omega\ell + \ell)NB$$

bits of memory. Trivially, increasing ω increases storage requirements. Considering $\omega = 2$, which usually is the better-performing choice than $\omega = 1$ (see also Section 4), increasing the polynomial degree to $2N$ reduces the memory requirements for each key once $2(\omega\ell + \ell)NB > 12\ell NB$, hence $\omega > 5$.

For the double-decomposition technique, one key switching key requires

$$2\omega\tilde{\omega}rNB = 2(\omega\ell + \tilde{\omega}\ell + \ell)NB$$

bits, and increasing ω or $\tilde{\omega}$ increases the size of the key switching key. In contrast to the single-decomposition technique, where reducing ω benefits NTT and memory complexity, we now have a trade-off between memory size and NTT performance. However, larger keys also have a performance penalty, as performing the dot product during key switching can be memory-bound [8].

Evaluation
Section 4

We evaluate our contributions using a comprehensive set of benchmarks. For the polynomial degrees $N \in \{2^{14}, \dots 2^{17}\}$, the closed formula by Mono et al. [27] outputs the upper bounds $\{433, 867, 1735, 3470\}$ for $\log qP$, respectively[3]. Our implementation[4] uses the open-source BGV library fhelib [14] with a HEXL-based polynomial layer [21].

We run all benchmarks on Ubuntu 20.04.5. The Central Processing Unit (CPU) is an Intel Core i9-7900X CPU at 3.3 GHz with 20 cores, and the system features 64 GiB of available memory. We disable Intel turbo boost, pin the benchmarking execution to a single CPU core. For a given degree, we generate three different types of parameter sets, which we describe below.

Type (1) parameter sets. Targeting single-decomposition key switching, we set the ciphertext modulus to 50 %, 65 %, 75 %, 80 %, 85 %, 90 %, and 95 % of the available modulus space, using the remaining space for P, choosing ω minimal. If $\omega = 1$, we generate an additional set with $\omega = 2$, as the latter reduces multiplication complexity. If $\omega \geq 9$, we generate an additional set with $2N$ and $\omega = 2$. We skip a parameter set if it requires $\omega > \ell$ for correctness, listing the results in Table 5.

Type (2) parameter sets. For the double-decomposition technique, we use the same percentages of the modulus space as for the type (1) parameter sets; however, choosing $\log P = b$, $\omega = \ell$, and $\tilde{\omega} = \sqrt{\ell(\ell+1)/2}$. As before, we skip a parameter set if it requires $\omega > \ell$ for correctness and

3 These bounds are slightly tighter than in the Homomorphic Encryption Standard [1], which only provides bounds for $N \in \{2^{10}, \dots, 2^{15}\}$. Since the standard's initial release, many publications require $N > 2^{15}$, which we account for in our evaluation by moving to larger degrees.

4 https://github.com/Chair-for-Security-Engineering/owl

measure execution time with the entire ciphertext modulus. We list the results in Table 6.

Type (3) parameter sets. Finally, we generate sets using $1/2$, $2/3$, and $3/4$ of the available modulus space, divided into 36-bit chunks, targeting the single-decomposition technique. We choose ω as small as possible (including $\omega = 1$) and generate a matching parameter set in which we replace $\kappa = 3$ primes of size 36 bit with $\mu = 2$ primes of size 54 bit. In contrast to type (1) and type (2) sets, we remove κ primes before measuring execution time for key switching for folded implementations without and with replacing primes, listing the results in Table 7.

4.1 Small versus large primes

We use the results for type (3) parameter sets in Table 7 to evaluate the performance gain for choosing mostly large primes and make several interesting observations:

- Switching in primes using the folded variant performs equally or better than only using small primes in all evaluated parameter sets, even in the smallest setting. The results also match our expectations from the example in Subsection 3.3, where key switching is faster for parameter sets with $\log q > 144$.
- In absolute terms, the overhead is, as expected, roughly constant for a given polynomial degree (for $\log N = 14$, for example, the overhead is 0.7 ms). In relative terms, the overhead will be more noticeable for a small number of primes, no matter the degree. Note that we have to replace primes during key switching only every κ levels and can even wait until the last key switching on a given level to work with fewer primes until then.

Overall, using fewer primes speeds up key switching significantly by up to 50 %.

4.2 Folding optimizations

We evaluate our folding optimizations with the type (1) parameter sets in Table 5. We improve execution time across the board by up to 11.6 %. For small ω, the speed-ups in execution time are more significant compared to parameter sets with large ω.

4.3 Increased polynomial degree

For type (1) parameter sets in Table 5, we include alternative sets with increased polynomial degree and $\omega = 2$. For example, for $\log N = 15$

and $\log q = 780$, increasing the polynomial degree leads to a speed-up of $1.2\times$. We receive the same speed-up for $\log N = 16$ and $\log q = 1624$, while for $\log N = 17$ and $\log q = 3300$, we only speed up key switching by a factor of $1.02\times$.

However, not all parameter sets crossing our theoretical threshold outperform the parameter sets with a smaller degree. For example, key switching with $\log N = 16$, $\log q = 1560$, and $\omega = 10$ performs better than the matching set with increased $\log N = 17$ and $\omega = 2$.

4.4 Comparison of decomposition techniques

By combining the results for the parameter sets of type (1) in Table 5 and type (2) in Table 6, we can compare the single- and double-decomposition techniques, each with optimally chosen parameters regarding the NTT complexity for a given degree and ciphertext modulus. The latter outperforms the former only for $\log N = 16$ and $\log q = 1624$.

We believe the reasons for this to be two-fold:

- For the double-decomposition technique, only considering NTT complexity results in huge key switching keys, and retrieving these keys during the dot product of key switching has large performance costs.
- While the fast base extension itself is relatively fast, we require an error correction when extending from the shared base E to the extended modulus qP. The error correction has noticeable costs, slowing down key switching for the double-decomposition technique.

Most likely, due to the above reasons, there is room for further improvement when choosing parameters in the double-decomposition technique, which we discuss in more detail in Subsection 5.3.

Our results significantly differ compared to the original results by Kim et al. [24], mainly because we choose key switching parameters considering each approach in isolation for a given degree and ciphertext modulus (matching the approach to parameter selection for actual use cases), instead of using a shared set of key switching parameters.

Discussion
Section 5

In the following, we discuss multiple aspects of our work, starting with the relevance of key switching for homomorphic use cases. We also discuss the implications of increasing the polynomial degree holistically, highlight some limitations of our work, and explore opportunities for future work.

5.1 Performance impact of key switching in homomorphic encryption

In the FHE community, it is common knowledge that key switching is one of the most expensive parts of homomorphic evaluation for BGV-like schemes. First, for most use cases, we perform many key switchings, as performing a desired computation often involves many rotations, even for low-level circuits with only a handful of modulus switching. One such example is homomorphic matrix multiplication [26], where a profiling run on our benchmarking setup shows that more than 50 % of execution time is spent in key switching.

Second, on the server side, key switching and modulus switching are the only homomorphic operations requiring forward and inverse NTT operations, the main computational bottleneck for homomorphic encryption, with respective asymptotic complexities of $\mathcal{O}(\omega\ell)$ and $\mathcal{O}(\ell)$. For modulus switching, however, we often can minimize the costs by merging it with key switching.

Third, key switching requires reading large keys from memory, which are only used for a single modular multiplication. This is especially relevant in hardware accelerators, as once the computational bottlenecks are accelerated, memory becomes the main problem even for custom-designed memory architectures [8, 18].

Thus, improving key switching boosts performance significantly in most homomorphic use cases. When optimizing use cases, another result of our work comes in handy: Sometimes, a use case benefits from key switching output with mixed output domains (output for some RNS primes in the NTT domain and some in the coefficient domain), which we show how to achieve for free using our folding optimizations.

5.2 Increasing the polynomial degree

Contrary to current folklore, increasing the polynomial degree can increase performance, namely for otherwise very large ω. However, although execution time decreases for key switching, the computational complexity of other operations increases with larger N.

The same holds for memory: we only reduce costs for key switching (such as the key switching keys or the temporary storage for the extended polynomials during key switching), increasing all other ciphertext storage, permanent or temporary.

We believe that, in addition to a large ω, the following conditions should be fulfilled before considering an increase of the polynomial degree: (1) the main bottleneck of the use case is the key switching operation; (2) the use case requires a multitude of rotations, each with their own unique key switching key; and (3) the key switching operation is memory-bound such as in hardware.

The double-decomposition technique can also perform better than in-

creasing the polynomial degree for large ω. However, key switching keys for the double-decomposition technique are by default larger than for the single-decomposition technique with the same parameters or require additional computations for every use [24]; therefore, an increased polynomial degree with a more straightforward key switching implementation and smaller keys might be more beneficial, especially for hardware implementations.

5.3 Limitations and future work

One limitation of our work is the somewhat simplified approach for selecting key switching parameters ω and $\tilde{\omega}$ for the double-decomposition technique. In the single-decomposition technique, choosing $\omega \geq 2$ as small as possible is beneficial regarding NTT and memory complexity. In contrast, the double-decomposition technique requires a trade-off between NTT complexity ($\omega = \ell$) and memory requirements ($\omega = 1$), which also has performance implications.

For optimal parameters, the double-decomposition approach probably requires hyper-parameter optimization using platform-specific information such as memory bandwidth, execution time for a forward and inverse NTT, as well as execution time for constant and polynomial multiplication; this is an excellent opportunity for future work. Doing so could include finding closed formulas for multiplication complexity instead of a rough estimate.

We also did not explore whether our folding optimizations positively impact the error correction after a fast base extension with the BEHZ method. Selecting the best correction method for a given platform could also be part of the previously mentioned hyper-parameter optimization for the double-decomposition technique.

Conclusion
Section 6

This work provides the first formal analysis of key switching parameters for BGV-like schemes. We consider the single-decomposition technique, that is, current state-of-the-art, and the recently proposed double-decomposition technique, providing improved theoretical complexities, generalized to arbitrary input and output domains.

In the process, we provide a new perspective on key switching, resulting in a new asymptotic bound on key switching complexity, and show that for parameters with minimal computational complexity, the single-decomposition technique mostly outperforms the double-decomposition technique.

Additionally, we formalize the process of switching primes in and out

of a ciphertext, integrate it with key switching, and analyze the resulting complexity; we also highlight novel opportunities for folding constants during key switching. These improvements speed up key switching by up to 50 % and 11.6 %, respectively.

References

[1] Martin R. Albrecht, Melissa Chase, Hao Chen, Jintai Ding, Shafi Goldwasser, Sergey Gorbunov, Shai Halevi, Jeffrey Hoffstein, Kim Laine, Kristin E. Lauter, Satya Lokam, Daniele Micciancio, Dustin Moody, Travis Morrison, Amit Sahai, and Vinod Vaikuntanathan. "Homomorphic Encryption Standard". In: IACR Cryptol. ePrint Arch. 2019.939 (2019). URL: https://eprint.iacr.org/2019/939.

[2] Martin R. Albrecht, Rachel Player, and Sam Scott. "On the concrete hardness of Learning with Errors". In: J. Math. Cryptol. 9.3 (2015), pp. 169–203. URL: http://www.degruyter.com/view/j/jmc.2015.9.issue-3/jmc-2015-0016/jmc-2015-0016.xml.

[3] Jacob Alperin-Sheriff and Chris Peikert. *Practical Bootstrapping in Quasilinear Time*. Oct. 2013. URL: https://eprint.iacr.org/2013/372.

[4] Jean-Claude Bajard, Julien Eynard, M. Anwar Hasan, and Vincent Zucca. "A Full RNS Variant of FV Like Somewhat Homomorphic Encryption Schemes". In: Selected Areas in Cryptography - SAC 2016 - 23rd International Conference, St. John's, NL, Canada, August 10-12, 2016, Revised Selected Papers. Ed. by Roberto Avanzi and Howard M. Heys. Vol. 10532. Lecture Notes in Computer Science. Springer, 2016, pp. 423–442. DOI: 10.1007/978-3-319-69453-5_23. URL: https://doi.org/10.1007/978-3-319-69453-5_23.

[5] Zvika Brakerski. "Fully Homomorphic Encryption without Modulus Switching from Classical GapSVP". In: Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings. Ed. by Reihaneh Safavi-Naini and Ran Canetti. Vol. 7417. Lecture Notes in Computer Science. Springer, 2012, pp. 868–886. DOI: 10.1007/978-3-642-32009-5_50. URL: https://doi.org/10.1007/978-3-642-32009-5_50.

[6] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. "(Leveled) Fully Homomorphic Encryption without Bootstrapping". In: ACM Trans. Comput. Theory 6.3 (2014), 13:1–13:36. DOI: 10.1145/2633600. URL: https://doi.org/10.1145/2633600.

[7] Zvika Brakerski and Vinod Vaikuntanathan. "Fully Homomorphic Encryption from Ring-LWE and Security for Key Dependent Messages". In: Advances in Cryptology - CRYPTO 2011 - 31st Annual Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2011. Proceedings. Ed. by Phillip Rogaway. Vol. 6841. Lecture Notes in Computer Science. Springer, 2011, pp. 505–524. DOI: 10.1007/978-3-642-22792-9_29. URL: https://doi.org/10.1007/978-3-642-22792-9%5C_29.

[8] Leo de Castro, Rashmi Agrawal, Rabia Tugce Yazicigil, Anantha P. Chandrakasan, Vinod Vaikuntanathan, Chiraag Juvekar, and Ajay Joshi. “Does Fully Homomorphic Encryption Need Compute Acceleration?” In: IACR Cryptol. ePrint Arch. 2021.1636 (2021). URL: https://eprint.iacr.org/2021/1636.

[9] Jung Hee Cheon, Kyoohyung Han, Andrey Kim, Miran Kim, and Yongsoo Song. “A Full RNS Variant of Approximate Homomorphic Encryption”. In: Selected Areas in Cryptography - SAC 2018 - 25th International Conference, Calgary, AB, Canada, August 15-17, 2018, Revised Selected Papers. Ed. by Carlos Cid and Michael J. Jacobson Jr. Vol. 11349. Lecture Notes in Computer Science. Springer, 2018, pp. 347–368. DOI: 10.1007/978-3-030-10970-7_16. URL: https://doi.org/10.1007/978-3-030-10970-7_16.

[10] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yong Soo Song. “Homomorphic Encryption for Arithmetic of Approximate Numbers”. In: Advances in Cryptology - ASIACRYPT 2017 - 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part I. Ed. by Tsuyoshi Takagi and Thomas Peyrin. Vol. 10624. Lecture Notes in Computer Science. Springer, 2017, pp. 409–437. DOI: 10.1007/978-3-319-70694-8_15. URL: https://doi.org/10.1007/978-3-319-70694-8_15.

[11] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. “TFHE: Fast Fully Homomorphic Encryption Over the Torus”. In: J. Cryptol. 33.1 (2020), pp. 34–91. DOI: 10.1007/s00145-019-09319-x. URL: https://doi.org/10.1007/s00145-019-09319-x.

[12] Ana Costache and Nigel P. Smart. “Which Ring Based Somewhat Homomorphic Encryption Scheme is Best?” In: Topics in Cryptology - CT-RSA 2016 - The Cryptographers’ Track at the RSA Conference 2016, San Francisco, CA, USA, February 29 - March 4, 2016, Proceedings. Ed. by Kazue Sako. Vol. 9610. Lecture Notes in Computer Science. Springer, 2016, pp. 325–340. DOI: 10.1007/978-3-319-29485-8_19. URL: https://doi.org/10.1007/978-3-319-29485-8_19.

[13] Anamaria Costache, Kim Laine, and Rachel Player. “Evaluating the Effectiveness of Heuristic Worst-Case Noise Analysis in FHE”. In: Computer Security - ESORICS 2020 - 25th European Symposium on Research in Computer Security, ESORICS 2020, Guildford, UK, September 14-18, 2020, Proceedings, Part II. Ed. by Liqun Chen, Ninghui Li, Kaitai Liang, and Steve A. Schneider. Vol. 12309. Lecture Notes in Computer Science. Springer, 2020, pp. 546–565. DOI: 10.1007/978-3-030-59013-0_27. URL: https://doi.org/10.1007/978-3-030-59013-0_27.

[14] Cryptography Research Centre. *fhelib*. https://github.com/Crypto-TII/fhelib. 2023.

[15] Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. “Multiparty Computation from Somewhat Homomorphic Encryption”. In: Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings. Ed. by Reihaneh Safavi-Naini and Ran Canetti. Vol. 7417. Lecture Notes in Computer Science. Springer, 2012, pp. 643–662. DOI: 10.1007/978-3-642-32009-5_38. URL: https://doi.org/10.1007/978-3-642-32009-5_38.

[16] Léo Ducas and Daniele Micciancio. “FHEW: Bootstrapping Homomorphic Encryption in Less Than a Second”. In: Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I. Ed. by Elisabeth Oswald and Marc Fischlin. Vol. 9056. Lecture Notes in Computer Science. Springer, 2015, pp. 617–640. DOI: 10.1007/978-3-662-46800-5_24. URL: https://doi.org/10.1007/978-3-662-46800-5_24.

[17] Junfeng Fan and Frederik Vercauteren. “Somewhat Practical Fully Homomorphic Encryption”. In: IACR Cryptol. ePrint Arch. (2012), p. 144. URL: http://eprint.iacr.org/2012/144.

[18] Robin Geelen, Michiel Van Beirendonck, Hilder V. L. Pereira, Brian Huffman, Tynan McAuley, Ben Selfridge, Daniel Wagner, Georgios Dimou, Ingrid Verbauwhede, Frederik Vercauteren, and David W. Archer. “BASALISC: Flexible Asynchronous Hardware Accelerator for Fully Homomorphic Encryption”. In: IACR Cryptol. ePrint Arch. 2022.657 (2022). URL: https://eprint.iacr.org/2022/657.

[19] Craig Gentry, Shai Halevi, and Nigel P. Smart. “Homomorphic Evaluation of the AES Circuit”. In: Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings. Ed. by Reihaneh Safavi-Naini and Ran Canetti. Vol. 7417. Lecture Notes in Computer Science. Springer, 2012, pp. 850–867. DOI: 10.1007/978-3-642-32009-5_49. URL: https://doi.org/10.1007/978-3-642-32009-5_49.

[20] Shai Halevi, Yuriy Polyakov, and Victor Shoup. “An Improved RNS Variant of the BFV Homomorphic Encryption Scheme”. In: Topics in Cryptology - CT-RSA 2019 - The Cryptographers’ Track at the RSA Conference 2019, San Francisco, CA, USA, March 4-8, 2019, Proceedings. Ed. by Mitsuru Matsui. Vol. 11405. Lecture Notes in Computer Science. Springer, 2019, pp. 83–105. DOI: 10.1007/978-3-030-12612-4_5. URL: https://doi.org/10.1007/978-3-030-12612-4_5.

[21] Fabian Boemer, Sejun Kim, Gelila Seifu, Fillipe DM de Souza, Vinodh Gopal, et al. *Intel HEXL (release 1.2)*. https://github.com/intel/hexl. Sept. 2021.

[22] Andrey Kim, Antonis Papadimitriou, and Yuriy Polyakov. “Approximate Homomorphic Encryption with Reduced Approximation Error”. In: Topics in Cryptology - CT-RSA 2022 - Cryptographers’ Track at the RSA Conference 2022, Virtual Event, March 1-2, 2022, Proceedings. Ed. by Steven D. Galbraith. Vol. 13161. Lecture Notes in Computer Science. Springer, 2022, pp. 120–144. DOI: 10.1007/978-3-030-95312-6_6. URL: https://doi.org/10.1007/978-3-030-95312-6%5C_6.

[23] Andrey Kim, Yuriy Polyakov, and Vincent Zucca. “Revisiting Homomorphic Encryption Schemes for Finite Fields”. In: Advances in Cryptology - ASIACRYPT 2021 - 27th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 6-10, 2021, Proceedings, Part III. Ed. by Mehdi Tibouchi and Huaxiong Wang. Vol. 13092. Lecture Notes in Computer Science. Springer, 2021, pp. 608–639. DOI: 10.1007/978-3-030-92078-4_21. URL: https://doi.org/10.1007/978-3-030-92078-4_21.

[24] Miran Kim, Dongwon Lee, Jinyeong Seo, and Yongsoo Song. *Accelerating HE Operations from Key Decomposition Technique*. June 2023. URL: https://eprint.iacr.org/2023/413.

[25] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. "On Ideal Lattices and Learning with Errors over Rings". In: J. ACM 60.6 (2013), 43:1–43:35. DOI: 10.1145/2535925. URL: https://doi.org/10.1145/2535925.

[26] Johannes Mono and Tim Güneysu. "Implementing and Optimizing Matrix Triples with Homomorphic Encryption". In: Proceedings of the 2023 ACM Asia Conference on Computer and Communications Security, ASIA CCS 2023, Melbourne, VIC, Australia, July 10-14, 2023. Ed. by Joseph K. Liu, Yang Xiang, Surya Nepal, and Gene Tsudik. ACM, 2023, pp. 29–40. DOI: 10.1145/3579856.3590344. URL: https://doi.org/10.1145/3579856.3590344.

[27] Johannes Mono, Chiara Marcolla, Georg Land, Tim Güneysu, and Najwa Aaraj. "Finding and Evaluating Parameters for BGV". In: Progress in Cryptology - AFRICACRYPT 2023 - 14th International Conference on Cryptology in Africa, Sousse, Tunisia, July 19-21, 2023, Proceedings. Ed. by Nadia El Mrabet, Luca De Feo, and Sylvain Duquesne. Vol. 14064. Lecture Notes in Computer Science. Springer, 2023, pp. 370–394. DOI: 10.1007/978-3-031-37679-5_16. URL: https://doi.org/10.1007/978-3-031-37679-5_16.

[28] Oded Regev. "On lattices, learning with errors, random linear codes, and cryptography". In: J. ACM 56.6 (2009), 34:1–34:40. DOI: 10.1145/1568318.1568324. URL: http://doi.acm.org/10.1145/1568318.1568324.

Parameters			Time (*ms*)		
$\log N$	$\log q$	ω	naïve	folded	speedup
14	216	1	3.9	3.5	11.6 %
14	216	2	3.8	3.6	7.6 %
14	280	2	5.3	4.9	8.3 %
14	324	3	6.8	6.4	6.6 %
14	342	5	9.3	9.0	3.5 %
14	364	6	12.5	12.0	4.3 %
15	432	1	20.9	19.2	8.8 %
15	432	2	20.0	19.1	4.9 %
15	560	2	26.6	24.8	7.4 %
15	649	3	33.0	31.5	4.8 %
15	684	5	45.2	43.1	4.8 %
15	728	6	54.5	53.2	2.5 %
15	780	10	100.4	98.6	1.9 %
16	780	2	84.9	80.7	5.2 %
16	855	1	125.5	118.0	6.3 %
16	855	2	117.6	113.3	3.8 %
16	1121	2	159.2	152.3	4.5 %
16	1298	3	227.0	220.1	3.1 %
16	1380	5	275.6	269.0	2.4 %
16	1475	6	335.6	330.1	1.7 %
16	1560	10	465.6	457.9	1.7 %
17	1560	2	584.6	564.6	3.5 %
16	1624	19	793.0	785.3	1.0 %
17	1624	2	650.3	626.5	3.8 %
17	1711	1	801.5	762.1	5.2 %
17	1711	2	690.5	671.0	2.9 %
17	2242	2	996.4	970.5	2.7 %
17	2580	3	1207.5	1176.4	2.7 %
17	2760	5	1444.6	1411.2	2.4 %
17	2940	6	1665.0	1647.7	1.0 %
17	3120	9	2099.9	2072.6	1.3 %
18	3120	2	3299.0	3172.6	4.0 %
17	3300	19	3566.8	3532.2	1.0 %
18	3300	2	3592.8	3462.4	3.8 %

Table 5: Results for parameter sets of type (1), measuring execution times of a naïve implementation without and with folding optimizations for the single-decomposition technique. For each parameter set, we also list the speed-up of the folded compared to the naïve implementation.

Parameters				Time (ms)
$\log N$	$\log q$	ω	$\tilde{\omega}$	
14	216	4	3	12.5
14	280	5	4	16.9
14	324	6	5	19.6
14	342	6	5	19.6
14	364	7	5	21.5
15	432	8	6	54.1
15	560	10	7	71.6
15	649	11	8	85.4
15	684	12	9	93.1
15	728	13	10	92.7
15	780	13	10	104.8
16	855	15	11	285.2
16	1121	19	14	410.2
16	1298	22	16	535.3
16	1380	23	17	564.9
16	1475	25	18	622.8
16	1560	26	19	663.5
16	1624	28	20	734.5
17	1711	29	21	1768.6
17	2242	38	27	2745.9
17	2580	43	31	3649.8
17	2760	46	33	3933.2
17	2940	49	35	4220.1
17	3120	52	37	4658.4
17	3300	55	39	5140.2

Table 6: Results for parameter sets of type (2), measuring execution time for the folded double-decomposition implementation.

$\log N$	$\log q$	$\log q_i$		Time (ms)		
		36	54	folded	switch	overhead
14	252	7		4.7		
		4	2	4.0	4.7	18.0 %
14	288	8		5.9		
		5	2	5.0	5.7	14.5 %
15	396	11		18.8		
		5	4	15.9	18.1	14.1 %
15	540	15		29.3		
		3	8	21.9	24.3	10.7 %
15	612	17		38.3		
		5	8	29.4	32.1	9.1 %
16	828	23		156.4		
		5	12	95.7	104.9	9.6 %
16	1116	31		241.1		
		4	18	170.1	178.2	4.7 %
16	1260	35		301.9		
		5	20	209.8	216.5	3.2 %
17	1692	47		1065.6		
		5	28	783.0	818.0	4.5 %
17	2268	63		1529.5		
		3	40	1013.1	1046.0	3.2 %
17	2556	71		1917.1		
		5	44	1262.1	1302.6	3.2 %

Table 7: Results for parameter sets of type (3), measuring execution time for the folded implementation without replacing primes, and for a folded implementation replacing $\mu = 2$ 54-bit primes with $\kappa = 3$ 36-bit primes. For the relevant parameter sets, we also list the relative overhead for switching in and out primes compared to the folded implementation without switching primes.