Incompleteness in Number-Theoretic Transforms: New Tradeoffs and Faster Lattice-Based Cryptographic Applications

Syed Mahbub Hafiz*, Bahattin Yildiz*, Marcos A. Simplicio Jr*†, Thales B. Paiva*,
Henrique Ogawa*, Gabrielle De Micheli*, and Eduardo L. Cominetti*
*Future Security Team, LG Electronics USA, Inc.
{syedmahbub.hafiz, bahattin.yildiz, thales.paiva, henrique1.ogawa, gabrielle.demicheli, eduardo.cominetti}@lge.com
†Universidade de São Paulo, Brazil
msimplicio@larc.usp.br

***Abstract*—Lattices are the basis of most NIST-recommended post-quantum cryptography (PQC) schemes, required to thwart the threat posed by the eventual construction of large-scale quantum computers. At the same time, lattices enable more advanced cryptographic constructions, such as fully homomorphic encryption (FHE), which is increasingly used for privacy-preserving applications like machine learning. This work delves into the efficiency and trade-off assessment of polynomial multiplication algorithms and their applications to PQC, FHE, and other schemes. Such algorithms are at the core of lattice-based cryptography and may become a critical bottleneck when deploying PQC- and FHE-based solutions on resource-constrained devices. We propose a formal analysis of so-called *incompleteness* in the Number Theoretic Transform (NTT). Although this concept is not new, our systematization shows how to optimize polynomial multiplication in quotient rings, considering factors such as the degree of incompleteness, the associated prime moduli, constraints of the target platform, and target security level. Besides efficiency, we formally show that the systematized family of *incomplete* NTT variants supports a larger set of prime moduli. This property enables new trade-offs for algorithms like the FIPS-approved module-lattice-based key encapsulation mechanism (ML-KEM) and faster amortized bootstrapping in FHE schemes. Our results include shorter ciphertexts in ML-KEM with only a modest hit in performance and a 6 – 42% performance boost in the NTT computation of a state-of-the-art FHE solution.**

***Index Terms*—Polynomial multiplication, post-quantum cryptography (PQC), lattices, fully homomorphic encryption (FHE), optimization, number-theoretic transform (NTT).**

1. Introduction

Lattice-based cryptography is the basis of many quantum-safe key encapsulation mechanisms (KEM) [1], digital signature algorithms (DSA) [2], and fully homomorphic encryption (FHE) schemes [3]. Post-quantum cryptography (PQC) is increasingly relevant, as classical cryptographic algorithms (e.g., based on integer factorization [4] and discrete logarithms [5]) may become vulnerable to a large-scale, cryptographically relevant quantum computer (CRQC) [6] running Shor's algorithm [7]. In 2016, NIST began standardizing quantum-safe KEM and DSA schemes [8]. Most schemes chosen for standardization are lattice-based [1], [2], [9], leveraging computationally hard problems like learning with errors (LWE) [10].

Lattices are also the basis of many modern FHE solutions [11]–[15]. By using efficient bootstrapping mechanisms, those schemes enable privacy-preserving computation over encrypted data, an increasingly necessary requirement for machine learning solutions that need to cope with government regulations.

Besides quantum security and better privacy, lattices also facilitate parallelism and scalability across different security levels [16]–[18]. However, implementing PQC and FHE on constrained devices, like microcontrollers in Internet of Things (IoT) applications, requires computing, bandwidth, and memory efficiency. This often demands trade-offs and optimizations of polynomial multiplications in quotient rings, operations that account for 30-50% of computation in lattice-based PQC and FHE [19]. For polynomials with n coefficients, the traditional Schoolbook multiplication algorithm has a complexity of $O(n^2)$, while the 2-way Toom-Cook (Karatsuba) algorithm achieves $O(n^{1.58})$. A more efficient method is the number-theoretic transform (NTT), the finite-field analog of the fast Fourier transform (FFT). Specifically, NTT enables polynomial multiplications in a quotient ring such as $\mathbb{Z}_q[x]/(\Phi(x))$ with a complexity of $O(n \lg n)$.

The NTT method essentially converts polynomials $\mathbf{a}(x)$ and $\mathbf{b}(x)$ from the coefficient domain to the value domain, where they can be multiplied more easily. Although NTT can be implemented differently, one common approach is using an iterative algorithm. In this case, the execution sequence follows a tree-like structure with $k = \lg n$ layers (iterations) – see Figure 1, which shows the execution tree for a complete, (k,k)-layer NTT. Due to the properties of cyclotomic polynomials and the Chinese remainder theorem (CRT), the degree-δ polynomial $\Phi(x)$ in one layer can be factored into two smaller, degree-($\delta/2$) polynomials in the next layer, defining a residue number system. Thus, $\mathbf{a}(x)$ can be represented in this lower layer as two polynomials, each with half the number of coefficients. After k layers, $\mathbf{a}(x)$ and $\mathbf{b}(x)$ can be represented using n constants each, so their product in this layer becomes simply a point-wise multiplication. Applying the inverse NTT can then convert this product back to the coefficient domain.

There are many algorithm-, software-, and hardware-level optimization tricks in the literature [20]–[27] to improve the performance of NTT-based polynomial mul-

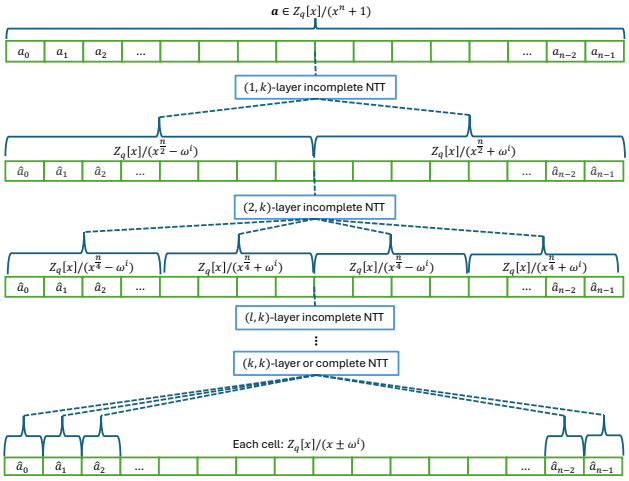


Figure 1: The execution tree of an NTT, showing the layer-by-layer CRT-based splitting of the polynomial quotient rings.

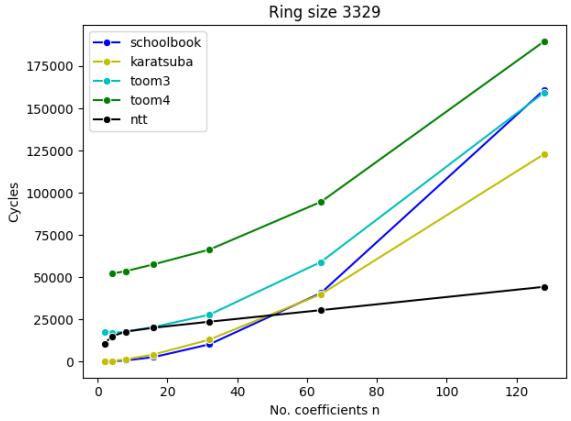


Figure 2: CPU cycles required to multiply two n-coefficient polynomials: when $n \leq 50$, Schoolbook/Karatsuba is more efficient than NTT. We used an x86_64 server with an Intel(R) Xeon(R) Gold 5118 CPU@2.30GHz as a testbed, and measured the mean over 100 runs of each algorithm implemented in C.

tiplication. Most of them are grounded on the complete NTT construction, comprising all layers. However, further performance gains can be obtained by stopping at an earlier layer of the NTT. Suppose that we stop the NTT execution at layer $\ell < k$, which we call an (ℓ,k)-layer incomplete NTT; the inverse NTT will also be incomplete, starting at layer ℓ. In this case, although the polynomials in layer ℓ are no longer constant, their degree may be small enough to enable efficient computations using a regular polynomial multiplication algorithm. Indeed, as shown in Figure 2, our experimental evaluations indicated that algorithms like Schoolbook and Karatsuba could be faster than NTT, depending on the size of the input polynomials.

Our literature review indicates that there are a few works [28]–[33] where more efficiency is achieved using an incomplete NTT. One example is Kyber [1], a key encapsulation mechanism recently standardized by NIST under the name ML-KEM: while its original version adopted a complete NTT, the current standard uses one less layer for improved performance. To the best of our knowledge, though, this choice was essentially empirical rather than based on a formal and comprehensive analysis of possible alternatives. In particular, the correlation between various potential alternatives and performance, ciphertext size, and other metrics remain largely unexplored.

More generally, we could not find any study that thoroughly explores and formally defines the entire range and benefits of NTT incompleteness. To bridge this gap, we hereby propose a systematic approach to identify optimal settings for performing polynomial multiplications using a family of incomplete NTTs. This enables, for example, the execution of Schoolbook or Karatsuba algorithms for the small-degree polynomials appearing at the incomplete layers of the NTT, leading to faster execution time. Moreover, this approach enables interesting tradeoffs. In particular, adopting NTT often restricts relevant parameters, such as the degree of the modulus polynomial n and the modulus q. Those, in turn, affect other metrics like security and overall performance. We show that such restrictions can also be relaxed by incorporating incomplete NTTs.

Contributions. Our main outcomes are described below.

I) We formalize and generalize the full spectrum of incompleteness in NTT-based polynomial multiplications – applicable in both cyclic and nega-cyclic convolutions – and investigate their main benefits. For instance, we formally show that an incomplete NTT enables a broader, more relaxed set of compatible moduli. This creates a range of options for fast polynomial multiplications, enabling performance gains of up to 40% in applications like FHE amortized bootstrapping. At the same time, this family enables different combinations of algorithms for the base polynomial multiplications done after the incomplete NTT calls, opening a "*mixing-and-matching*" door to choosing various options (e.g., like Karatsuba, Toom-Cook, and Schoolbook) depending on the context. This study complements the few works in the literature that discuss incomplete NTTs in an ad-hoc and empirical manner and incorporates them as special cases of the analyzed family of incomplete NTTs.

II) We propose a framework for finding the optimal configuration of incomplete NTT-based polynomial multiplications. Specifically, for a target computing platform, application, and security level, our approach optimizes parameters like the number of layers to be skipped and the corresponding finite field prime modulus. The resulting configuration may not only benefit from optimizations available in the literature but also benefit from a wider choice of values for the underlying modulus.

III) We consider some concrete scenarios to evaluate the practical benefits of the proposed systematization. Namely, we show an in-depth study of how NTT incompleteness affects Kyber [1] in a resource-constrained setting, using an ARM Cortex M4 microcontroller, and we demonstrate the impact of our approach on the overall performance of a state-of-the-art amortized bootstrapping algorithm for FHE [34]. The source code and scripts used to produce the experimental results are publicly available at: https://github.com/smhafiz/incompleteness_in_ntts_new_tradeoffs_lattice_crypto.git.

Among our evaluation results, we examine the standalone task of multiplying two polynomials under Kyber's cryptographic settings using an isochronous implementa-

tion (i.e., protected against timing side-channel attacks). Our results indicate that a 2-incomplete NTT is 5% faster than the 1-incomplete NTT adopted in Kyber. Since Kyber skips NTT calls by directly sampling in the NTT domain, though, those gains end up being diluted when we consider the whole task of keypair generation, encapsulation, and decapsulation. We also show new parameter sets that enable interesting tradeoffs between decryption failure rate (DFR), ciphertext compression, and performance, leveraging unique properties of incomplete NTTs (e.g., a wider modulus choice). For example, for its highest security level, Kyber1024, we present a new parameter set that achieves a DFR lower than 2^{-165} and 4% shorter ciphertexts at the expense of a modest performance hit (around 9%) compared to the reference implementation [1]. This compression is relevant because, compared to classical schemes (e.g., RSA and ECDH), Kyber's main drawback is substantially larger ciphertexts [35].

For concreteness, we also evaluate how different incomplete NTT settings affect the TLS 1.3 protocol in a hybrid key exchange scenario, combining ECDH and Kyber, as done in some real-world deployments [36]–[38]. As expected, the overall computation time of the initial key exchange is dominated by the slower ECDH, while the bandwidth usage is governed by Kyber's larger ciphertexts. However, with NTT incompleteness, both client- and server-side computation costs are increased only by 0.5%, 2%, and 5% for 128-, 192-, and 256-bit security levels, respectively. Meanwhile, Kyber's ciphertext compression leads to communication costs 4 – 8% lower, compared to the 6% gain obtained in [35].

Finally, in our FHE case study, we explore optimal choices of incompleteness in NTT and prime modulus for different security levels of a state-of-the-art amortized bootstrapping algorithm [34]. This ultimately results in a 42%, 30%, 6%, and 33% improvement over regular (complete) NTT-based option for 128-, 192-, 256-, and 512-bit security, respectively. We envision that this broad exploration of polynomial multiplication strategies, as well as our experimental findings, can be applied to optimize any application involving such operations, including lattice-based cryptography.

Paper outline. Section 2 lists some useful notations used in this study. Section 3 formalizes and generalizes NTT incompleteness, leading to our proposal and thorough evaluation of the theoretical benefits of an (ℓ,k)-layer incomplete NTT in Section 4. To evaluate the proposed mechanisms, Section 5 experimentally demonstrates how to obtain optimal incompleteness parameters in practice. Sections 6 and 7 then assess two real-world scenarios, showing novel tradeoffs for ML-KEM Kyber, and better performance in a state-of-the-art amortized bootstrapping algorithm for FHEW-like schemes. Section 8 reviews related works and Section 9 concludes the discussion.

2. Notations

$\mathbb{Z}_q$ is the finite field of integers modulo prime q. Polynomials are represented by bold and small letters. $\mathbb{Z}_q[x]/(\Phi(x))$ is a polynomial quotient ring of polynomials $\mathbf{a}(x) \in \mathbb{Z}_q[x]$ reduced by $\Phi(x)$. Due to the reduction, each polynomial of the ring has a degree less than the degree of $\Phi(x)$ and the same number of coefficients. The degree of the modulus polynomial $\Phi(x)$ is denoted by n and is a power of 2 throughout this work. If $\Phi(x)$ is x^n-1, the primitive nth root of unity is ω_n, and they are relevant to cyclic convolution. If $\Phi(x)$ is the $2n$th cyclotomic polynomial, x^n+1, the primitive $2n$th root of unity is ζ_{2n}, and they are relevant to negacyclic convolution. We use ζ_{2n}^{ρ} to denote that ζ_{2n} is raised to the power of ρ. Integers ℓ and k are the number of iterations (layers) used in the incomplete and complete NTT, respectively. When presenting a cryptographic application, we use λ to denote its security level. For simplicity, we use the term "NTT" when referring to both the linear transformation and the fast algorithm used to compute it. Also, when clear from the context, we use it to denote an NTT-based polynomial multiplication. Finally, lg stands for the base-2 logarithm.

3. NTT incompleteness: introduction

We start by revising the regular NTT, a linear transformation that enables fast multiplication of two polynomials $\mathbf{a}$(x) and $\mathbf{b}$(x) in the quotient ring $\mathbb{Z}_q[x]/(\Phi(x))$. Let $\Phi(x)$ be the $2n$th cyclotomic polynomial (x^n+1) for the negacyclic (a.k.a. negative-wrapped) convolution-based multiplication, where $n=2^k$. Following the properties of Cyclotomics discussed in Appendix C.1, one root of $\Phi(x)$ is a primitive $2n$th root of unity, hereby denoted ζ_{2n}, with $q \equiv 1 \bmod 2n$ and $\zeta_{2n}^{2n} \equiv 1 \bmod q$. More than that, all of its n roots (a.k.a. twiddle factors) are given by $\zeta_{2n}^1, \zeta_{2n}^3, \zeta_{2n}^5, \ldots, \zeta_{2n}^{2n-1}$. Using these roots, we can factorize x^n+1 into n linear polynomials (a.k.a. ideals in the ring): $x^n+1 \equiv (x-\zeta_{2n}^1)(x-\zeta_{2n}^3)(x-\zeta_{2n}^5)\ldots(x-\zeta_{2n}^{2n-1}) \in \mathbb{Z}_q[x]$. Due to remainder arithmetic [39] and Chinese remainder theorem (CRT) (reviewed in Appendix C.2), we have a ring isomorphism:

$$\mathbb{Z}_q[x]/(x^n+1) \cong \mathbb{Z}_q[x]/(x-\zeta_{2n}^1) \times \ldots \times \mathbb{Z}_q[x]/(x-\zeta_{2n}^{2n-1}).$$

Intuitively, reducing any given polynomial $\mathbf{a}$(x) by x^n+1 is equivalent to reducing it by n linear factors in $\mathbb{Z}_q[x]$. The remainder $\mathbf{a}(x) \bmod (x-\zeta_{2n}^{2i-1})$ can then be computed by evaluating the polynomial at the root, ζ_{2n}^{2i-1}. Hence, the reduction consists in evaluating $\mathbf{a}$(x) on the n roots, i.e., we need to compute $\mathbf{a}(\zeta_{2n}^1), \mathbf{a}(\zeta_{2n}^3), \ldots, \mathbf{a}(\zeta_{2n}^{2n-1})$. Thus, we map $\mathbf{a}$(x) in the ring to this tuple in $\mathbb{Z}_q^n$.

With this preamble, we can now discuss how the NTT converts a polynomial from the coefficient domain to the value domain. One of the most efficient NTT algorithms, described in [40], iteratively splits the modulus polynomial x^n+1 into two factors per iteration, forming the layered execution tree illustrated in Figure 1. To reach the linear factors, this divide-and-conquer approach takes $k = \lg n$ iterations. Since the in-place updates of the input vector involve $O(n)$ operations, the asymptotic cost of the algorithm is $O(n \lg n)$.

Now, due to the mapping technique [39] in the ring isomorphism, we can evaluate the product polynomial $\mathbf{c}(x) = \mathbf{a}(x) \cdot \mathbf{b}(x) \in \mathbb{Z}_q[x]/(x^n+1)$ by using the NTT, i.e., by making $\mathbf{c}(x) = \text{invNTT}(\text{NTT}(\mathbf{a}(x)) \odot \text{NTT}(\mathbf{b}(x)))$. The point-wise multiplication $\odot$ can be applied once the input polynomials $\mathbf{a}$(x) and $\mathbf{b}$(x) are

separately converted to the value domain, at the end of the NTT operation. We then have the n-tuple in $\mathbb{Z}_q^n$:

$$\mathbf{a}(\zeta_{2n}^1)\cdot\mathbf{b}(\zeta_{2n}^1), \mathbf{a}(\zeta_{2n}^3)\cdot\mathbf{b}(\zeta_{2n}^3), \ldots, \mathbf{a}(\zeta_{2n}^{2n-1})\cdot\mathbf{b}(\zeta_{2n}^{2n-1}),$$

This gives the n point values of the product polynomial $\mathbf{c}(\mathrm{x})$ in the value domain. We can then convert $\mathbf{c}(\mathrm{x})$ to the coefficient domain using the inverse NTT operation (or invNTT, for short), which uses the CRT to interpolate those n points and, hence, reconstruct the polynomial $\mathbf{c}(\mathrm{x})$ of degree-$(n-1)$. The point-wise multiplication has n multiplications and the most efficient in-place iterative invNTT algorithm [41], similarly to the forward NTT algorithm, uses $O(n \lg n)$ operations. Overall, the complexity of multiplying two polynomials in $\mathbb{Z}_q[x]/(x^n+1)$ becomes, thus, $O(n \lg n + n + n \lg n) = O(n \lg n)$.

One interesting property of this iterative NTT algorithm is that it is possible to stop its execution before the final layer of the execution tree, due to the mathematical properties of Cyclotomics and CRT. Doing so leads to the implementation of what is called an incomplete NTT method [30]. By stopping at the $\ell < k$ layer when processing the cyclotomic ring[1] element $\mathbf{a}(x)$ as input, it no longer factors into linear polynomials, but rather into small-degree polynomials. Hence, if $\mathbf{a}(x)$ and $\mathbf{b}(x)$ are submitted to this incomplete NTT execution, they cannot be multiplied via a point-wise operation anymore. Instead, that requires a complete polynomial multiplication algorithm (e.g., Schoolbook or Karatsuba). We call this latter operation *base multiplication*, whose output is a tuple of similarly small-degree polynomials. Finally, to obtain $\mathbf{c}(x) = \mathbf{a}(x)\cdot\mathbf{b}(x) \in \mathbb{Z}_q[x]/(x^n+1)$ in the coefficient domain, the output of the base multiplication must be processed with an incomplete invNTT, running only ℓ iterations. Therefore, analogously to the complete NTT, we can write $\mathbf{c}(x) = \mathrm{invNTT}_{(\ell,k)}(\mathrm{NTT}_{(\ell,k)}(\mathbf{a}(x)) \star \mathrm{NTT}_{(\ell,k)}(\mathbf{b}(x)))$, where $\star$ is the position-wise base multiplication.

Formalizing NTT incompleteness. We can now formally define the family of incomplete NTT and its properties.

Definition 1 (Incomplete and complete NTT). An (ℓ,k)-layer NTT is an algorithm that stops at ℓ out of k layers, where $k = \log n$ and $\ell \in [1,k]$. We call an (ℓ,k)-layer NTT a complete NTT when $\ell = k$, and an incomplete NTT when $\ell \in [1, k-1]$.

Definition 2 (Incompleteness property). An (ℓ,k)-layer NTT has $(k-\ell)$-incompleteness.

Corollary 1 (Completeness property). Analogously, an (ℓ,k)-layer NTT has ℓ-completeness.

Next, we define a convenient operation to describe the powers of the twiddle factors.

Definition 3 (Bit reversal). Let n be a non-negative integer such that $0 \le n < 2^k$ for some k. Let $\mathsf{BitRev}_k(n)$ denote the integer whose bit representation is the same as n when written in reverse. Formally, if $n = 2^{k-1}n_{k-1} + \ldots + 2n_1 + n_0$, then $\mathsf{BitRev}_k(n) = 2^{k-1}n_0 + \ldots + 2n_{k-2} + n_{k-1}$.

The following three statements are various outcomes of iterative splitting of the polynomial quotient rings due to incompleteness.

1. Cyclotomic ring is a polynomial quotient ring where the modulus polynomial is fixed as a cyclotomic polynomial.

Observation 1. After an incomplete (ℓ,k)-layer NTT, the 2^ℓ elements or polynomials are in the $\mathbb{Z}_q[x]/(x^{2^{k-\ell}} - \zeta_{2^{\ell+1}}^{2\cdot\mathsf{BitRev}_\ell(i)+1})$ polynomial quotient ring, where $i \in [0, 2^\ell - 1]$ and $\zeta_{2^{\ell+1}}$ is the primitive $2^{\ell+1}$-th root of unity.

Corollary 2. After an incomplete (ℓ,k)-layer NTT, each of the reduced polynomials has a degree at most $2^{k-\ell}-1$ and, thus, has $2^{k-\ell}$ coefficients.

Theorem 1. An incomplete (ℓ,k)-layer NTT factors the quotient polynomial x^n+1 into $\prod_{i=0}^{i=2^\ell-1}(x^{2^{k-\ell}} - \zeta_{2^{\ell+1}}^{2\cdot\mathsf{BitRev}_\ell(i)+1})$.

Figure 1 can be seen as a graphical depiction of this theorem. After a complete (k,k)-layer NTT, the elements or polynomials are in $\mathbb{Z}_q[x]/(x^{2^0} - \zeta_{\frac{2n}{2^0}}^{\rho}) = \mathbb{Z}_q[x]/(x^{2^0} - \zeta_{\frac{2\cdot 2^k}{2^0}}^{\rho}) = \mathbb{Z}_q[x]/(x^{2^0} - \zeta_{\frac{2^{k+1}}{2^0}}^{\rho}) = \mathbb{Z}_q[x]/(x^{2^0} - \zeta_{2^{k+1-0}}^{\rho})$ — the 2^k smaller polynomials are degree-(2^0-1), or degree-0, or constant. After an incomplete $(k-1,k)$-layer NTT, the elements or polynomials are in $\mathbb{Z}_q[x]/(x^{2^1} - \zeta_{\frac{2n}{2}}^{\rho}) = \mathbb{Z}_q[x]/(x^{2^1} - \zeta_{2^{k+1-1}}^{\rho}) = \mathbb{Z}_q[x]/(x^{2^1} - \zeta_{2^k}^{\rho})$ — in this case, the 2^{k-1} smaller polynomials are degree-(2^1-1) or degree-1. Likewise, after an incomplete (ℓ,k)-layer NTT, the 2^ℓ elements or polynomials are in $\mathbb{Z}_q[x]/(x^{2^{k-\ell}} - \zeta_{\frac{2n}{2^{k-\ell}}}^{\rho}) = \mathbb{Z}_q[x]/(x^{2^{k-\ell}} - \zeta_{2^{k+1-k+\ell}}^{\rho}) = \mathbb{Z}_q[x]/(x^{2^{k-\ell}} - \zeta_{2^{\ell+1}}^{\rho})$ — in this case, the smaller polynomials are degree-$(2^{k-\ell}-1)$.

Observation 2. For the $(k-\ell)$-incompleteness, we only require a primitive $2^{\ell+1}$-th root of unity, i.e., $\zeta_{2^{\ell+1}}$.

Corollary 3. For the $(k-\ell)$-incompleteness, we require a prime modulus, $q \equiv 1 \bmod 2^{\ell+1}$.

For $(k-\ell)$-incompleteness, we do not need a primitive $2n$-th root of unity, but rather a primitive $\frac{2n}{2^{k-\ell}} = 2^{\ell+1}$-th root of unity. This can be achieved using a smaller q, as shown in Theorem 2.

Theorem 2. An incomplete (ℓ,k)-layer NTT can use a smaller q than the complete, (k,k)-layer NTT counterpart.

Proof. A complete, (k,k)-layer NTT needs a modulus $q \equiv 1 \bmod 2^{k+1}$ and, from Definition 1, we have $\ell < k$. □

Illustrative example. We showcase the incomplete NTT method with real examples in what follows.

Example 1. Say we want to multiply two polynomials with $n = 256 = 2^8$ coefficients so that $k = 8$. If we employ a $(5,8)$-layer incomplete NTT-based multiplication, the NTT algorithm stops at layer $\ell = 5$ out of $k = 8$ layers. We need a primitive $2^{5+1} = 64$-th root of unity instead of a 512-th one. One immediate implication is that to obtain the 512-th primitive root of unity, we need a prime modulus of at least 7681 – conversely, for the 64-th primitive root of unity, a much smaller modulus is available, e.g., as small as 257. After the 5-th layer, there are 2^5 smaller polynomials, all in $\mathbb{Z}_q[x]/(x^{2^3} - \zeta_{2^6}^{\rho}) = \mathbb{Z}_q[x]/(x^8 - \zeta_{64}^{\rho})$ — in this case, those polynomials are degree-(2^3-1) = degree-7, and have 8 coefficients. Furthermore, the $(5,8)$-layer incomplete NTT splits $(x^{256}+1)$ into $\prod_{i=0}^{i=31}(x^8 - \zeta_{64}^{2\cdot\mathsf{BitRev}_5(i)+1}) = (x^8-\zeta_{64}^{1})(x^8-\zeta_{64}^{33})\ldots(x^8-\zeta_{64}^{31})(x^8-\zeta_{64}^{63})$.

Corollary 4. For a given k-completeness, there is a minimum modulus q_{min} that satisfies $q \equiv 1 \bmod 2^{\ell+1}$, irrespective of the original number of coefficients, n.

Example 2. For 1-completeness, $q_{min} = 5$. Likewise for 2-completeness, $q_{min} = 17$; for 3-completeness, $q_{min} = 17$; for 4-completeness, $q_{min} = 97$; for 5-completeness, $q_{min} = 193$; for 6-completeness, $q_{min} = 257$; for 7-completeness, $q_{min} = 257$; for 8-completeness, $q_{min} = 7681$; and so on.

We observe that the minimum prime modulus also gets larger as we increase the completeness.

Algorithm 1 Incomplete (ℓ, k)-layer NTT

1: **procedure** $\text{NTT}_{(\ell,k)}(\mathbf{a})$
2: **for** $i \leftarrow 0$ to $\ell - 1$ **do**
3: $d \leftarrow 2^{k-1-i}$
4: **for** $j \leftarrow 0$ to $2^i - 1$ **do**
5: $\omega \leftarrow \zeta_{2^{\ell+1}}^{\text{BitRev}_\ell(2^i+j)}$
6: **for** $u \leftarrow 0$ to $d - 1$ **do**
7: $\texttt{idx} \leftarrow 2dj + u$
8: $t_0 \leftarrow \mathbf{a}[\texttt{idx}]$
9: $t_1 \leftarrow \omega \cdot \mathbf{a}[\texttt{idx} + d]$
10: $\mathbf{a}[\texttt{idx}] \leftarrow t_0 + t_1$
11: $\mathbf{a}[\texttt{idx} + d] \leftarrow t_0 - t_1$
12: **return** $\hat{\mathbf{a}}$

4. A flexible, incomplete (ℓ, k)-layer NTT

In this section, we discuss how to construct an adaptable and efficient (ℓ, k)-layer incomplete NTT algorithm for multiplying polynomials using negacyclic convolution (also transferable to cyclic convolution) in a quotient ring. Essentially, this is shown in Algorithm 3, which described the steps required for multiplying two polynomials in a suitable negacyclic ring by leveraging the arbitrarily incomplete forward and inverse NTT (Algorithms 1 and 2, respectively). We can see that, after executing the (ℓ, k)-layer NTTs for input polynomials $\mathbf{a}$ and $\mathbf{b}$, Algorithm 3 is left with a total of 2^ℓ small-degree polynomials for each input. Those polynomials are correspondingly multiplied using traditional Schoolbook or Karatsuba in Step 5 of Algorithm 3. We now define this particular polynomial multiplication, hereby called base multiplication.

Definition 4. $\text{PolyMul}_n^{\mathbb{Z}_q}(\mathbf{a}, \mathbf{b})$ is a negacyclic convolution-based algorithm to multiply two degree-$(n-1)$ polynomials $\mathbf{a}, \mathbf{b} \in \mathbb{Z}_q[x]/(x^n + 1)$. The product, $\mathbf{c} := \text{PolyMul}_n^{\mathbb{Z}_q}(\mathbf{a}, \mathbf{b})$ is also in $\mathbb{Z}_q[x]/(x^n + 1)$.

Algorithm 2 Incomplete (ℓ, k)-layer invNTT

1: **procedure** $\text{INVNTT}_{(\ell,k)}(\hat{\mathbf{c}})$
2: **for** $i \leftarrow \ell - 1$ down to 0 **do**
3: $d \leftarrow 2^{k-1-i}$
4: **for** $j \leftarrow 0$ to $2^i - 1$ **do**
5: $\omega \leftarrow \zeta_{2^{\ell+1}}^{-\text{BitRev}_\ell(2^i+j)}$
6: **for** $u \leftarrow 0$ to $d - 1$ **do**
7: $\texttt{idx} \leftarrow 2dj + u$
8: $t_0 \leftarrow \hat{\mathbf{c}}[\texttt{idx}] + \hat{\mathbf{c}}[\texttt{idx} + d]$
9: $t_1 \leftarrow \hat{\mathbf{c}}[\texttt{idx}] - \hat{\mathbf{c}}[\texttt{idx} + d]$
10: $\hat{\mathbf{c}}[\texttt{idx}] \leftarrow t_0$
11: $\hat{\mathbf{c}}[\texttt{idx} + d] \leftarrow \omega \cdot t_1$
12: **for** $i \leftarrow 0$ to $n - 1$ **do**
13: $\hat{\mathbf{c}}[i] \leftarrow \hat{\mathbf{c}}[i] \cdot 2^{-\ell} \bmod q$
14: **return** $\mathbf{c}$

Definition 5. $\text{BaseMul}_{2^{k-\ell}}^{\mathbb{Z}_q}(\mathbf{a}, \mathbf{b})$ is a $\text{PolyMul}_{2^{k-\ell}}^{\mathbb{Z}_q}(\mathbf{a}, \mathbf{b})$ and occurs after an incomplete (ℓ, k)-layer NTT.

To facilitate the comparison between polynomial multiplication algorithms using incomplete and complete NTTs, we also define in Algorithm 4 the negacyclic polynomial multiplication using the latter. This algorithm uses the following definition.

Definition 6. $\text{PointWiseMul}_{\mathbb{Z}_q}(a, b)$ is a constant multiplication $\text{PolyMul}_1^{\mathbb{Z}_q}(a, b)$, and occurs after a complete (k, k)-layer NTT, where $a, b \in \mathbb{Z}_q^n$.

Algorithm 3 Incomplete (ℓ, k)-layer NTT-based polynomial multiplication

1: **procedure** $\text{PolyMul}_n^{\mathbb{Z}_q}(\mathbf{a}, \mathbf{b})$
2: $\hat{\mathbf{a}} \leftarrow \text{NTT}_{(\ell,k)}(\mathbf{a})$
3: $\hat{\mathbf{b}} \leftarrow \text{NTT}_{(\ell,k)}(\mathbf{b})$
4: **for** $i = 1$ to 2^ℓ chunks **do**
5: $\hat{\mathbf{c}}[i] \leftarrow \text{BaseMul}_{2^{k-\ell}}^{\mathbb{Z}_q}(\hat{\mathbf{a}}[i], \hat{\mathbf{b}}[i])$ ▷ $2i$th $2^{k-\ell}$-length chunk-wise multiplication
6: $\mathbf{c} \leftarrow \text{invNTT}_{(\ell,k)}(\hat{\mathbf{c}})$

Algorithm 4 (k, k)-layer complete NTT-based polynomial multiplication

1: **procedure** $\text{PolyMul}_n^{\mathbb{Z}_q}(\mathbf{a}, \mathbf{b})$
2: $\hat{\mathbf{a}} \leftarrow \text{NTT}_{(k,k)}(\mathbf{a})$
3: $\hat{\mathbf{b}} \leftarrow \text{NTT}_{(k,k)}(\mathbf{b})$
4: **for** $i = 1$ to 2^k elements **do**
5: $\hat{\mathbf{c}}[i] \leftarrow \text{PointWiseMul}_{\mathbb{Z}_q}(\hat{\mathbf{a}}[i], \hat{\mathbf{b}}[i])$ ▷ ith element-wise multiplication
6: $\mathbf{c} \leftarrow \text{invNTT}_{(k,k)}(\hat{\mathbf{c}})$

With those base algorithms defined, we can proceed with the potential benefits of using incomplete NTTs, as well as finding configurations that better explore those benefits.

4.1. Incompleteness for relaxation on moduli

NTT imposes restrictions on moduli choice for a given n. This may be undesirable from a designer's perspective because smaller moduli may enable a range of practical optimizations, including lazy modular reductions: with smaller inputs, a single register can accommodate the output of a larger number of algebraic operations before a modular reduction is needed to avoid overflows. The effect of such optimizations can be observed in the design of CRYSTALS-Kyber, for example, which adopted $q = 3329$ (12 bits) in the final round of the NIST standardization procedure, replacing the initial choice of $q = 7681$ (13 bits) for increased efficiency. Other examples are discussed in Section 6 (also for CRYSTALS-Kyber) and in Section 7 (for amortized bootstrapping for FHEW/TFHE-like schemes).

We hereby show that NTT incompleteness can ease the constraints, offering more choices for moduli selection for a given n. For instance, while a complete NTT may not have any supported modulus in a given range of values,

our proposed family of incomplete NTTs may allow a much broader set of moduli options. In what follows, we formally prove this flexibility added by NTT incompleteness. Before that, however, we show some lemmas that will help us build our proof.

Lemma 1. For $\ell \geq 1$, there is a prime q such that $q \equiv 1 \bmod 2^{\ell}$, but $q \not\equiv 1 \bmod 2^{\ell+1}$.

Proof. Dirichlet's theorem states that if $a, b \in \mathbb{N}$ are relatively prime (co-prime), there are infinitely many primes of the form $q = a + mb$ with $m \in \mathbb{Z}$. For our case, we assume $a = 1 + 2^{\ell}$ and $b = 2^{\ell+1}$. Since $a = 1 + 2^{\ell}$ is an odd number and $b = 2^{\ell+1}$ is a power of 2, they are co-prime. Therefore, we can apply Dirichlet's theorem to find infinitely many primes of the form $(1 + 2^{\ell}) + m \cdot 2^{\ell+1}$. But if we take any one of these primes $q = 1 + 2^{\ell} + m \cdot 2^{\ell+1}$, we see that $q \equiv 1 \bmod 2^{\ell}$ and $q \equiv 1 + 2^{\ell} \bmod 2^{\ell+1} \not\equiv 1 \bmod 2^{\ell+1}$. Consequently, we can find a prime q (or, in fact, infinitely many) such that $q \equiv 1 \bmod 2^{\ell}$, but $q \not\equiv 1 \bmod 2^{\ell+1}$. □

We can extend this result to prove a more general case when $\ell_1 < \ell_2$.

Lemma 2. For $\ell_1 < \ell_2$, there is a prime q such that $q \equiv 1 \bmod 2^{\ell_1}$, but $q \not\equiv 1 \bmod 2^{\ell_2}$.

Proof. Assume $\ell_2 = \ell_1 + m$ where $m \geq 1$ is a positive integer. According to Lemma 1, there is a prime q such that $q \equiv 1 \bmod 2^{\ell_1+m-1}$, but $q \not\equiv 1 \bmod 2^{\ell_1+m} \not\equiv 1 \bmod 2^{\ell_2}$. Since, $m \geq 1$, we have $q \equiv 1 \bmod 2^{\ell_1}$. Thus, we can find a prime q such that $q \equiv 1 \bmod 2^{\ell_1}$, but $q \not\equiv 1 \bmod 2^{\ell_2}$. □

Theorem 3. Let S_{ℓ_1}, S_{ℓ_2} be two sets of moduli that support (ℓ_1, k)-layer and (ℓ_2, k)-layer incomplete NTT, respectively. If $\ell_1 < \ell_2$ then $S_{\ell_2} \subsetneq S_{\ell_1}$.

Proof. First we prove that $S_{\ell_2} \subset S_{\ell_1}$, then we show that $S_{\ell_2} \neq S_{\ell_1}$. From Corollary 3, we know that $S_{\ell_1} = \{q_1 : q_1 \equiv 1 \bmod 2^{\ell_1+1}\}$ and $S_{\ell_2} = \{q_2 : q_2 \equiv 1 \bmod 2^{\ell_2+1}\}$.

Take an element q_2 from set S_{ℓ_2}. Then, for some integer $m \in \mathbb{N}$, we have

$$q_2 = 1 + m2^{\ell_2+1} = 1 + m2^{\ell_2-\ell_1}2^{\ell_1+1}.$$

Since $\ell_2 - \ell_1 > 0$, then $q_2 \equiv 1 \bmod 2^{\ell_1+1}$, meaning that q_2 is also an element of S_{ℓ_1}. Thus, we prove $S_{\ell_2} \subset S_{\ell_1}$.

From Lemma 2, we know that there exists a prime q such that $q \in S_{\ell_1}$ but $q \notin S_{\ell_2}$ when $\ell_1 < \ell_2$. Therefore, we prove $S_{\ell_2} \neq S_{\ell_1}$. We conclude, thus, that $S_{\ell_2} \subsetneq S_{\ell_1}$. □

4.2. Achieving optimal incompleteness

From the discussion so far, we have a new arbitrarily flexible incomplete NTT algorithm along with its unique advantage of enabling a broader set of moduli. The question is how to choose the best magnitude of incompleteness in a given setup. Next, to inspect the optimization equation, we gradually study some important definitions and observations.

Definition 7 (Computation needed). For a given platform, let resources consumed for executing operation A be denoted by $\mathcal{W}(A)$ (read as "computation in A"). Such resources can be measured by common metrics, such as the number of CPU cycles, elapsed wall-clock time, etc.

Here, we are more concerned with the computational costs of the actual implementation than with asymptotic complexity. Therefore, we can write the following definitions for capturing the computational costs of polynomial multiplication using complete and incomplete NTT implementations, as well as the differences between them:

Observation 3 (Computation needed in complete NTT-based $PolyMul$).

$$\begin{aligned}\mathcal{W}(\text{Alg. } 4) = \mathcal{W}\big(\text{NTT}_{(k,k)}(\mathbf{a})\big) + \mathcal{W}\big(\text{NTT}_{(k,k)}(\mathbf{b})\big) \\ + 2^k \cdot \mathcal{W}\big(\text{PointWiseMul}_{\mathbb{Z}_q}\big) \\ + \mathcal{W}\big(\text{invNTT}_{(k,k)}(\hat{\mathbf{c}})\big)\,. \end{aligned} \tag{1}$$

Observation 4 (Computation needed in incomplete NTT-based $PolyMul$).

$$\begin{aligned}\mathcal{W}(\text{Alg. } 3) = \mathcal{W}\big(\text{NTT}_{(\ell,k)}(\mathbf{a})\big) + \mathcal{W}\big(\text{NTT}_{(\ell,k)}(\mathbf{b})\big) \\ + 2^{\ell} \cdot \mathcal{W}\Big(\text{BaseMul}_{2^{k-\ell}}^{\mathbb{Z}_q}\Big) \\ + \mathcal{W}\big(\text{invNTT}_{(\ell,k)}(\hat{\mathbf{c}})\big)\,. \end{aligned} \tag{2}$$

Definition 8 (Savings in incomplete NTT). For a given n, let $\mathcal{S}\big(\text{NTT}_{(\ell,k)}\big)$ (read as "savings in $\text{NTT}_{(\ell,k)}$") be the difference between $\mathcal{W}\big(\text{NTT}_{(\ell,k)}\big)$ and $\mathcal{W}\big(\text{NTT}_{(k,k)}\big)$, i.e.,

$$\mathcal{S}\big(\text{NTT}_{(\ell,k)}\big) = \mathcal{W}\big(\text{NTT}_{(k,k)}\big) - \mathcal{W}\big(\text{NTT}_{(\ell,k)}\big)\,.$$

Definition 9 (Overhead in incomplete NTT-based $PolyMul$). For a given $n = 2^k$, let $\mathcal{C}(\text{BaseMul})$ (read as "overhead in BaseMul") be the difference between $\mathcal{W}\Big(\text{BaseMul}_{2^{k-\ell}}^{\mathbb{Z}_q}\Big)$ and $\mathcal{W}\big(\text{PointWiseMul}_{\mathbb{Z}_q}\big)$, i.e.,

$$\begin{aligned}\mathcal{C}(\text{BaseMul}) = 2^{\ell} \cdot \mathcal{W}\Big(\text{BaseMul}_{2^{k-\ell}}^{\mathbb{Z}_q}\Big) \\ - 2^k \cdot \mathcal{W}\big(\text{PointWiseMul}_{\mathbb{Z}_q}\big)\,. \end{aligned}$$

Algorithm 3 can save a noticeable amount of computation during its two calls of forward NTT and one call of the inverse NTT, as it avoids some layers of regular computation. On the other hand, it incurs some overhead to multiply smaller-degree polynomials by BaseMul, compared to point-wise constant multiplication. There are performance gains to be obtained when the savings outweigh the incurred overhead, i.e., the incomplete NTT is better if the following condition is met:

Definition 10 (Incompleteness is better). For a given $n = 2^k$, we say that $(k - \ell)$-incompleteness is better if $\mathcal{W}(\text{Algorithm } 4) > \mathcal{W}(\text{Algorithm } 3)$. Alternatively, if $\mathcal{S}\big(2 \cdot \text{NTT}_{(\ell,k)} + \text{invNTT}_{(\ell,k)}\big) > \mathcal{C}(\text{BaseMul})$.

Also, according to Theorem 3, and Corollary 3 and Theorem 2, one significant benefit of incomplete NTT is its ability to operate with smaller moduli than those usually required for a complete NTT. Hence, in principle we have two degrees of liberty for optimization — the values of ℓ and q — to choose according to any target computing environment. Nevertheless, in many cryptographic applications, there is a relation between the chosen modulus and the associated security strength of a given cryptographic protocol, which may limit the choice of q. It is important, thus, to consider two scenarios when looking for optimal performance gains: when q can be chosen and when it is fixed by the application itself. Those scenarios are formalized as follows.

TABLE 1: Theoretical polynomial multiplication costs when $\mu/\alpha = 5$, according to Equations 3 and 4.

n	Complete NTT	Incomplete NTT			Schoolbook
	$(\ell = k)$	$(\ell = k-1)$	$(\ell = k-2)$	$(\ell = k-3)$	$(\ell = 0)$
64	4672	4512	4608	5472	24,640
128	10,688	10,368	10,560	12,288	98,432
256	24,064	23,424	23,808	27,264	393,472
512	53,504	52,224	52,992	59,904	1,573,376

4.2.1. Optimal incompleteness when q can be chosen.

Definition 11 (SecurityAchieved(q)). For a given lattice cryptography-based application, the security strength that can be obtained by having a given modulus q for the underlying cryptographic protocol.

Definition 12 (Optimal $(k-\ell)$-incompleteness and q). For a given n and λ, we define the optimal $(k-\ell)$-incompleteness as the pair of (cryptographically-relevant) ℓ and q that maximize the gain function defined as

$$\begin{aligned}\mathcal{G}(\ell, q) &= \mathcal{W}(\text{Alg. } 4) - \mathcal{W}(\text{Alg. } 3) \\ &= \mathcal{S}\big(2 \cdot \text{NTT}_{(\ell,k)} + \text{invNTT}_{(\ell,k)}\big) - \mathcal{C}(\text{BaseMul}),\end{aligned}$$

where ℓ and q satisfy

1) $1 \leq \ell < k$,
2) $q \equiv 1 \bmod 2^{\ell+1}$, and
3) $\text{SecurityAchieved}(q) \geq \lambda$.

4.2.2. Optimal incompleteness when q is fixed.

Definition 13 (Optimal $(k-\ell)$-incompleteness). For a given n and q, we define the optimal $(k-\ell)$-incompleteness as the value of ℓ that maximizes the gain function $\mathcal{G}(\ell) = \mathcal{W}(\text{Alg. } 4) - \mathcal{W}(\text{Alg. } 3)$, for $1 \leq \ell < k$.

4.3. Analysis

Modular addition and multiplication consume varying numbers of CPU cycles and memory across different computing platforms. Hence, it is imperative to consider their actual costs, measured in the target platform, as part of the optimization process. Then, we can plug those costs in our proposed optimization algorithms.

Definition 14. For a given finite field, let α and μ be the costs associated, respectively, with (modular) addition and multiplication in a given platform.

Equation 2 becomes, then:

$$\begin{aligned}\mathcal{W}(\text{Alg. } 3) &= \mathcal{W}\big(\text{NTT}_{(\ell,k)}(\mathbf{a})\big) + \mathcal{W}\big(\text{NTT}_{(\ell,k)}(\mathbf{b})\big) \\ &\quad + 2^{\ell} \cdot \mathcal{W}\Big(\text{BaseMul}^{\mathbb{Z}_q}_{2^{k-\ell}}\Big) \\ &\quad + \mathcal{W}\big(\text{invNTT}_{(\ell,k)}(\hat{\mathbf{c}})\big) \\ &= 3\ell \cdot 2^{k-1}(2\alpha + \mu) \\ &\quad + \big(2^{2k-\ell} + 2^k\big)(\alpha + \mu). \end{aligned} \tag{3}$$

Similarly, Equation 1 becomes:

$$\begin{aligned}\mathcal{W}(\text{Alg. } 4) &= \mathcal{W}\big(\text{NTT}_{(k,k)}(\mathbf{a})\big) + \mathcal{W}\big(\text{NTT}_{(k,k)}(\mathbf{b})\big) \\ &\quad + 2^{k} \cdot \mathcal{W}\big(\text{PointWiseMul}_{\mathbb{Z}_q}\big) \\ &\quad + \mathcal{W}\big(\text{invNTT}_{(k,k)}(\hat{\mathbf{c}})\big) \\ &= 3 \cdot k \cdot 2^{k-1}(2\alpha + \mu) + \mu \cdot 2^{k+1}. \end{aligned} \tag{4}$$

Multiplication typically takes more CPU cycles than addition in hardware. However, the actual cost ratio varies depending on the target machine, from general-purpose to embedded systems. Hence, to illustrate our optimization problem, we hereby discuss a few arbitrary ratios: we use $\mu/\alpha = 5$ in Table 1,[2] and some other ratios in the Appendix (see Tables 5 and 6). We also consider a range of coefficients $n = [64, 512]$. We can see that, for incompleteness parameters $\ell = k-1$ or $\ell = k-2$, the computation cost of polynomial multiplication using the incomplete NTT is lower than what is needed with the complete NTT ($\ell = k = \log_2(n)$). This indicates that, for a reasonable value of μ/α, there are a few options to choose from the incompleteness pool that perform better than with a complete NTT. We remark that the computation needed when using the incomplete NTT observed for $\mu/\alpha = 5$ when $1 \leq \ell \leq k-3$ is lower than that of the Schoolbook algorithm ($\ell = 0$), but higher than the computation needed with the complete NTT. Following Definition 8, we can write:

Observation 5. For a given n, the $\mathcal{S}\big(\text{NTT}_{(\ell,k)}\big)$ is the difference between $\mathcal{W}\big(\text{NTT}_{(\ell,k)}\big)$ and $\mathcal{W}\big(\text{NTT}_{(k,k)}\big)$, i.e.,

$$\begin{aligned}\mathcal{S}\big(\text{NTT}_{(\ell,k)}\big) &= \mathcal{W}\big(\text{NTT}_{(k,k)}\big) - \mathcal{W}\big(\text{NTT}_{(\ell,k)}\big) \\ &= 2^{k-1}(k-\ell)(2\alpha + \mu).\end{aligned}$$

Likewise,

$$\begin{aligned}\mathcal{S}\big(\text{invNTT}_{(\ell,k)}\big) &= \mathcal{W}\big(\text{invNTT}_{(k,k)}\big) - \mathcal{W}\big(\text{invNTT}_{(\ell,k)}\big) \\ &= 2^{k-1}(k-\ell)(2\alpha + \mu).\end{aligned} \tag{5}$$

Also, following Definition 9:

Observation 6. For a given n, the $\mathcal{C}(\text{BaseMul})$ is the difference between $\mathcal{W}\Big(\text{BaseMul}^{\mathbb{Z}_q}_{2^{k-\ell}}\Big)$ and $\mathcal{W}\big(\text{PointWiseMul}_{\mathbb{Z}_q}\big)$, i.e.,

$$\begin{aligned}\mathcal{C}(\text{BaseMul}) &= 2^{\ell} \cdot \mathcal{W}\Big(\text{BaseMul}^{\mathbb{Z}_q}_{2^{k-\ell}}\Big) \\ &\quad - 2^{k} \cdot \mathcal{W}\big(\text{PointWiseMul}_{\mathbb{Z}_q}\big) \\ &= 2^{2k-\ell}(\alpha + \mu) + 2^{k} \cdot (\alpha - \mu).\end{aligned} \tag{6}$$

Finally, given the values of n and λ from the underlying application, the platform-specific modular arithmetic costs of α and μ, as well as the set of constraints applicable to ℓ and q, we solve the optimization problem below to maximize function $\mathcal{G}(\ell, q)$ (see Definition 12 and Equations 5, 6):

$$\begin{aligned}\max_{\ell,q} \quad & \mathcal{G}(\ell, q) = 3 \cdot 2^{k-1}(k-\ell)(2\alpha + \mu) \\ & \qquad - 2^{2k-\ell}(\alpha + \mu) - 2^{k} \cdot (\alpha - \mu) \\ \text{satisfying} \quad & 1 \leq \ell < k, \\ & q \equiv 1 \bmod 2^{\ell+1}, \\ & \text{SecurityAchieved}(q) \geq \lambda.\end{aligned} \tag{7}$$

To determine a suitable modulus for an (ℓ, k)-layer incomplete NTT, we solve the above optimization problem, which is also illustrated in Appendix D as a bird's-eye view. Since efficient lattice-based schemes typically require smaller moduli, this can be achieved through an exhaustive search. In many cases, application security requirements allow a range of prime moduli. Once such

2. This choice of $\mu/\alpha = 5$ should be somewhat compatible with the Cortex-M3 processor, where additions take one cycle and long unsigned multiplication (UMULL) takes 3 to 5 cycles [42].

TABLE 2: Optimal ℓ when one input polynomial is already in the NTT domain considering $\mu/\alpha = 5$. Computation costs are theoretical but exact.

n	Complete NTT	Incomplete NTT			Schoolbook
	$(\ell = k)$	$(\ell = k-1)$	$(\ell = k-2)$	$(\ell = k-3)$	$(\ell = 0)$
64	3328	3392	3712	4800	24,640
128	7552	7680	8320	10,496	98,432
256	16,896	17,152	18,432	22,784	393,472
512	37,376	37,888	40,448	49,152	1,573,376

a range is identified, we iteratively check the next prime (starting from the lower bound) against the given constraints. Specifically, the prime must be congruent to 1 modulo $2^{\ell+1}$ and must support both primitive $2^{\ell+1}$th and 2^{ℓ}th roots of unity for negacyclic convolution-based polynomial multiplication in a quotient ring using an incomplete NTT. The complexity of this brute-force approach depends on the number of primes within that range.

4.4. Optimal ℓ if inputs are in NTT domain

For some cryptographic schemes, we can directly sample input polynomials in the NTT domain, which saves the cost associated with executing NTT-related operations. If one of the given polynomials is already in the NTT domain, then Definition 10 becomes the following corollary.

Corollary 5. For a given n, when one polynomial is already in NTT domain, incompleteness is better if $\mathcal{S}\left(\text{NTT}_{(\ell,k)} + \text{invNTT}_{(\ell,k)}\right) > \mathcal{C}(\mathsf{BaseMul})$.

The gain function in the optimization Equation 7 then changes to:

$$\mathcal{G}(\ell) = 2 \cdot 2^{k-1} \cdot (k-\ell)(2\alpha+\mu) - 2^{2k-\ell} \cdot (\alpha+\mu) - 2^{k} \cdot (\alpha-\mu). \tag{8}$$

Similarly, if both input polynomials are already in the NTT domain, Definition 10 becomes:

Corollary 6. For a given n and q, when both polynomials are in NTT domain, incompleteness is better if $\mathcal{S}\left(\text{invNTT}_{(\ell,k)}\right) > \mathcal{C}(\mathsf{BaseMul})$.

Meanwhile, the gain function in the optimization Equation 7 narrows down to:

$$\mathcal{G}(\ell) = 2^{k-1} \cdot (k-\ell)(2\alpha+\mu) - 2^{2k-\ell} \cdot (\alpha+\mu) - 2^{k} \cdot (\alpha-\mu). \tag{9}$$

TABLE 3: Optimal ℓ when both input polynomials are already in the NTT domain considering $\mu/\alpha = 5$. Computation costs are theoretical but exact.

n	Complete NTT	Incomplete NTT			Schoolbook
	$(\ell = k)$	$(\ell = k-1)$	$(\ell = k-2)$	$(\ell = k-3)$	$(\ell = 0)$
64	1984	2272	2816	4128	24,640
128	4416	4992	6080	8704	98,432
256	9728	10,880	13,056	18,304	393,472
512	21,248	23,552	27,904	38,400	1,573,376

We demonstrate the implication of having polynomials directly sampled in the NTT domain in Table 2 considering $\mu/\alpha = 5$. Like Table 1, we have various $n = [64, 512]$ values of the theoretical computation cost of Schoolbook, complete NTT, and all selected incomplete NTT members for a given n. We find that a few incomplete NTT options, e.g., $\ell = k-1$ and $\ell = k-2$, that were better than complete NTT (in Table 1) are either a bit (Table 2) or more (Table 3) costly than complete NTT. Especially, when one or both input polynomials can be sampled directly in the NTT domain, the saving margin disappears (as also noticed from the above corollaries), and complete NTT becomes the better choice. However, 1-incomplete ($\ell = k-1$) NTTs are still comparable to the complete NTT performance, and since incompleteness enables a relaxed set of moduli, some contexts may benefit from it by boosting performance or new tradeoff opportunities offered by this choice.

5. Benchmarking optimal incompleteness

Using our C testbed on an x86_64 server equipped with an Intel(R) Xeon(R) Gold 5118 CPU@2.30GHz (introduced in Figure 2), we analyzed the performance of traditional polynomial multiplication strategies, motivating the use of Schoolbook or Karatsuba algorithms for small-degree polynomials in the incomplete NTT setting. While, in theory, any polynomial multiplication algorithm can serve as the base multiplication, we initially maintained a general discussion and considered three widely used algorithms as potential candidates. However, in practice, our results indicate that Schoolbook multiplication outperforms other approaches on both our server and ARM Cortex-M4 embedded device for the base multiplication.

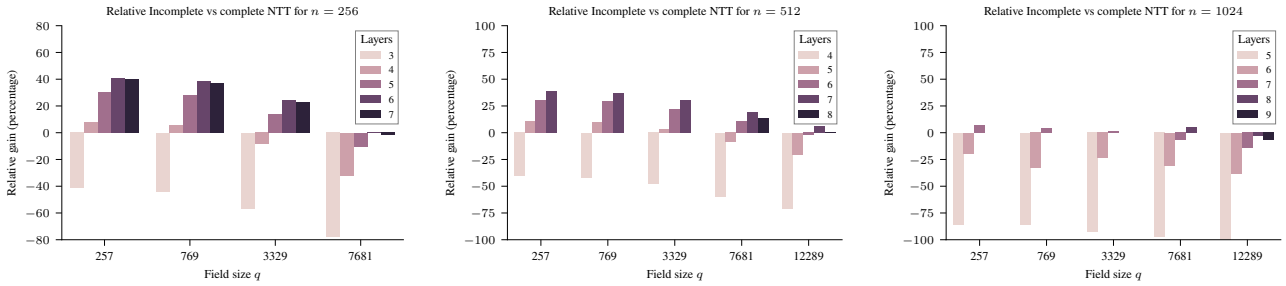
Now, we evaluate our proposed methodology for $n = \{256, 512, 1024\}$, considering the scenarios where q is fixed and can be changed. In PQC, smaller values of n are more relevant, whereas in FHE, larger values (≥ 1024) are used. Our results are illustrated in Figure 3. Along the Y-axis, we plot the gain[3] as a percentage of incomplete NTT's cost relative to its complete counterpart, as defined below:

Definition 15 (Relative gain). Let $\mathcal{G}_{\text{relative}}$ be the relative gain of incomplete NTT-based $PolyMul$ against complete NTT-based $PolyMul$. It is evaluated as

$$\mathcal{G}_{\text{relative}} = \frac{\mathcal{W}\left(\text{NTT}_{(k,k)}\right) - \mathcal{W}\left(\text{NTT}_{(\ell,k)}\right)}{\mathcal{W}\left(\text{NTT}_{(k,k)}\right)} = 1 - \frac{\mathcal{W}\left(\text{NTT}_{(\ell,k)}\right)}{\mathcal{W}\left(\text{NTT}_{(k,k)}\right)}.$$

As shown in Figures 3a, 3b, and 3c, for each value of n, multiple advantageous options – characterized by a relative gain above 0% – exist for selecting ℓ and q. However, the number of instances where incomplete NTT configurations yield better performance, indicated by the number of bars above the zero line, decreases as n increases. Figures 3a and 3c illustrate that if a given application requires a minimum modulus of 7681 for $n = 256$ or 12289 for $n = 1024$ due to security or other constraints, the optimal choices are the $(8, 8)$-layer complete NTT and the $(10, 10)$-layer complete NTT, respectively. However, Figure 3b shows that when the application does not permit modulus changes, the $(9, 9)$-layer complete NTT is not necessarily the optimal choice.

3. Note that when relative gain becomes negative, we call it a relative overhead.



(a) Optimal $\ell = 6$ (b) Optimal $\ell = 7$ (c) Optimal $\ell = 7$

Figure 3: Optimal ℓ and q in practice, measured by the relative gain in CPU cycles on our C testbed, for multiplying two arbitrary polynomials with $n = 256$ (3a), 512 (3b), and 1024 (3c). Relative gain 0% represents the complete NTT instances.

Specifically, when $n = 512$ and q is fixed at 12289, the $(7, 9)$-layer incomplete NTT outperforms the $(9, 9)$-layer complete NTT. These findings indicate that even when an application imposes constraints on modulus selection, introducing incompleteness in the NTT computation may still lead to performance improvements.

If an application allows modulus variation, reducing the modulus for each value of n leads to an increased number of advantageous options through incompleteness. Specifically, when $n = 256, 512$, the performance of incomplete NTT configurations with $\ell = 8, 7, 6, 5$ improves – the corresponding bars become taller – as the modulus decreases. Furthermore, for any given reduced q, fixing n (e.g., $n = 256$) reveals that the $(6, 8)$-layer incomplete NTT outperforms all other configurations. This allows us to determine the optimal value of ℓ for each n, which we identify as $6, 7$, and 7 for $n = 256, 512$, and 1024, respectively.

As a concrete example, the $(6, 8)$-layer incomplete NTT with $q = 257$ is 41% faster than the $(8, 8)$-layer complete NTT, which requires $q \geq 7681$. Similarly, for $n = \{512, 1024\}$, using a smaller modulus enables the incomplete NTT to outperform complete NTT alternatives. These findings demonstrate that certain incomplete NTT configurations provide better performance than their complete NTT counterparts. However, the most substantial performance gains are observed when the target application supports using a smaller modulus – a condition that may not always be viable in practice.

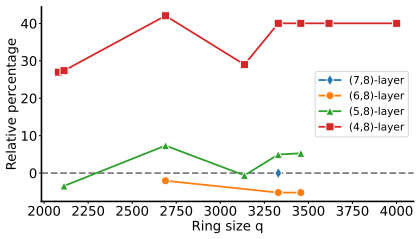
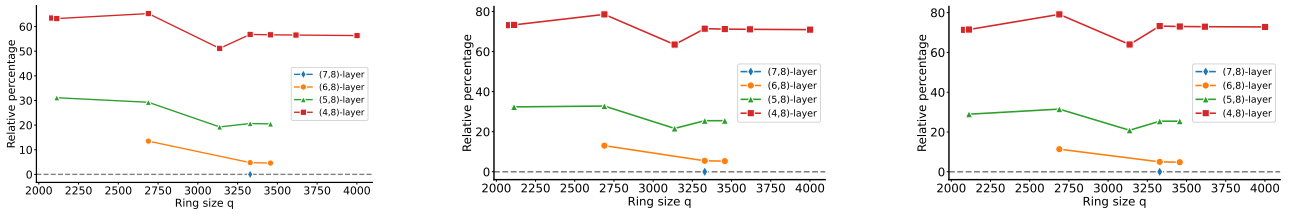


Figure 4: CPU cycles taken by the incomplete NTT-based $PolyMul$ (Alg. 3) in the modified PQClean reference implementation of Kyber on an ARM Cortex-M4 for varying incompleteness and selected moduli. The y-axis represents the relative overhead (in %) compared to Kyber's default (7,8)-layer NTT with $q = 3329$, marked as a blue diamond with a 0% overhead.

6. Case study: ML-KEM Crystals-Kyber

Kyber's original design used $q = 7681$ as the underlying prime modulus, and a complete (8,8)-layer NTT for polynomial multiplication. However, its most recent version provides faster polynomial multiplication by adopting 1-incompleteness in NTT and changing the modulus to $q = 3329$, which is a particular case of our proposed family of incomplete NTTs. In both versions, though, the set of parameters employed is such that: (1) the underlying MLWE problem is capable of effectively protecting the scheme's private key and ciphertext; and (2) the resulting decryption failure rate (DFR) is negligible – namely, 2^{-120} for security level 1, 2^{-136} for security level 2, and 2^{-165} for security level 3 [43]. In this section, we assess Kyber's choice of parameters, as well as alternatives, by seeking the optimal value of incompleteness under varying practical constraints. The goal is to explore trade-offs between those parameters while still preserving Kyber's underlying MLWE security and DFR. For comparison with our proposed scenarios, we mark the current Kyber configuration as the default setup.

Experimental setup. We use Kyber's constant-time C reference implementation from PQClean [44], targeting an ARM Cortex-M4 (STM32 Nucleo-F439ZI) due to its practical relevance. Cycle counts are measured via the Cortex-M4's Data Watchpoint and Trace (DWT) registers, with clock prescaler settings configured to 28 MHz for minimizing memory latency [20]. Performance is reported in CPU cycles, making it independent of the particular clock frequency configured in the device. This approach offers a standardized, precise, and configuration-agnostic metric that facilitates comparisons between different studies and device settings. Nevertheless, the actual time in seconds for each experiment can be calculated by multiplying the cycle count by the inverse of the CPU clock frequency. Software compilation was conducted using `arm-none-eabi-gcc` (v9.2.1) with `-O3` optimizations for Cortex-M4. Peripheral configurations were managed via STM32 CubeMX. The reported results corresponds to an average of 100 runs of each experiment, yielding a negligible standard deviation (below 0.3%). We also verified correctness across all procedures (keypair generation, encapsulation, and decapsulation) to ensure our modifications did not introduce implementation errors.



(a) Kyber key pair generation. (b) Kyber key encapsulation. (c) Kyber key decapsulation.

Figure 5: CPU cycles spent executing modified PQClean reference implementation of Kyber512 on an ARM Cortex-M4 device to enable various incompleteness in NTT/invNTT and associated and selected moduli. Along the Y-axis, we present the relative overhead in percentage of each case compared to the default case, i.e., (7,8)-layer NTT with q=3329, marked by a diamond-shaped blue point.

6.1. Optimizing performance and modulus

Initially, we evaluate the performance of the following operations in Kyber: incomplete NTT (Appendix Figure 13a); incomplete invNTT (Appendix Figure 13b); and full polynomial multiplication (Figure 4). As expected, we observe that more incompleteness leads to NTT and invNTT procedures requiring less computation. For instance, a call to the (4,8)-layer incomplete NTT function is faster than any ($\ell > 4$,8)-layer options. However, higher incompleteness incurs overhead in the base multiplication (shown in Table 4, since it requires the multiplication of higher-degree constituent polynomials as discussed in Algorithm 3. Consequently, the overall polynomial multiplication performance exhibits different results. More precisely, as discussed in Section 4.2 and Definition 10, we observe better performance if two calls to incomplete NTT and one call to incomplete invNTT save more cycles than the overhead added to the comparatively higher-degree polynomial-based base multiplication. In particular, as shown in Figure 4, a (6,8)-layer incomplete NTT-based polynomial multiplication is 5% faster than the one observed in Kyber's default, (7,8)-layer specification.

BaseMul	Time (cycles)
$\mathsf{BaseMul}_2^{\mathbb{Z}_{3329}}$	134
$\mathsf{BaseMul}_4^{\mathbb{Z}_{3329}}$	450
$\mathsf{BaseMul}_8^{\mathbb{Z}_{3329}}$	1766
$\mathsf{BaseMul}_{16}^{\mathbb{Z}_{3329}}$	7112

TABLE 4: CPU cycles consumed executing BaseMul for (7,8)-layer to (4,8)-layer incomplete NTT-based polynomial multiplication options (Algorithm 3).

As discussed in Section 4.1, NTT incompleteness also allows a more relaxed set of moduli. For example, for a given range, say $2048 < q < 4096$ (i.e., the prime moduli that need 12 bits to represent), (5,8)-layer incomplete NTT supports more moduli (e.g., $2113, 2689, 2753, 3137, 3329, 3457$) than that of (6,8)-layer incomplete NTT (with only $2689, 3329, 3457$). Whereas, the default (7,8)-layer permits only one prime modulus (3329) in this range, being restrictive. Therefore, for each level of incompleteness, we measure the performance under different q from the union set $\{2081, 2113, 2689, 3137, 3329, 3457, 3617, 4001\}$ obtaining the results shown in Figure 13.

By analyzing the results, we can see that a given level of incompleteness leads to similar performance for the

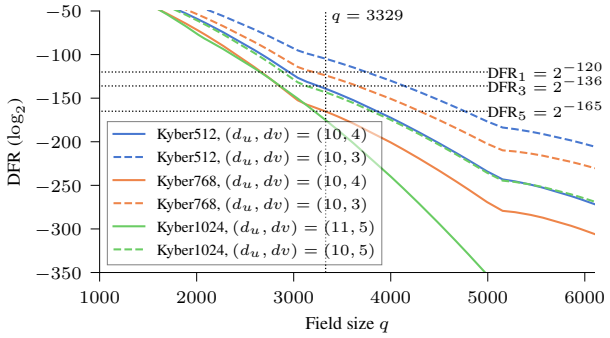


Figure 6: DFRs for Kyber with all three security levels for varying moduli leveraged by incompleteness in NTT with current and proposed values of (d_u, d_v) depicted by solid and dashed lines, respectively. The horizontal lines show the DFR targets for security levels 1, 3 and 5.

multiplication between two arbitrary polynomials, except for a few moduli (e.g., 2689 and 3457).[4] Overall, the (4,8)- and (5,8)-layers are respectively 40% and 5% slower than the default (7,8)-layer incomplete NTT chosen by Kyber. However, (6,8)-layer incomplete NTT-based polynomial multiplication is 5% faster than the (7,8)-layers. Therefore, for polynomial multiplication, the (6,8)-layer with either $q = 3329$ or 3457 would be a better choice.

However, since Kyber allows sampling one or both of the input polynomial(s) directly in the NTT domain (as in Section 4.4), those gains in the standalone polynomial multiplication are not reflected in Kyber's keypair generation, encapsulation, and decapsulation operations. More precisely, we observed that saving at most one call to NTT and one call to invNTT often could not supersede the overhead in schoolbook-based base multiplication. This difference accumulated over all polynomial multiplications involved in Kyber's operations, resulting in a performance hit of 5%, 8%, and 10% for (6,8)-layer incomplete NTT-based Kyber512 (Figure 5), Kyber768 (Figure 17), and Kyber1024 (Figure 18).

4. Investigating the reason, we isolated the compiler-generated Assembly code for the (6,8)-layer incomplete NTT with q = 2689 and q = 3457. Both Assembly snippets look identical except for the zeta table (twiddle factor) values. In addition, we have also inserted some instruction counters to check how many times each Assembly label block was being accessed in both cases. Since the observed counter values were the same for both cases, we concluded that the instruction execution counts are also equal for both q = 2689 and q = 3457 cases. Hence, this difference is not being caused by divergences in the compiler outputs. Instead, it is happening during the code execution.

6.2. Optimizing DFR and ciphertext size

In this and in the following sections, we use the security estimate script[5] from the Kyber team to evaluate the impact and new trade-offs that can leverage incompleteness. Figure 6 shows how the value of (d_u, d_v) pair[6] affects the decryption failure rate (DFR) when the modulus q (in the X-axis) is changed. Regular values of (d_u, d_v) are shown in solid lines, while modified values use dashed lines. We observe that, by decreasing either d_u or d_v while increasing q, the DFR also decreases. This is relevant because a low DFR is a common security requirement for KEMs, which otherwise may be vulnerable to chosen-ciphertext attacks exploiting decryption failures [45], [46]. Therefore, the lower DFR obtained with an increased modulus, as enabled through incompleteness, can be beneficial in the design of such algorithms.

Remarkably, there is no clear guidance on how the DFR target for each security level should be defined. To allow for some tradeoff margin in our analysis while keeping the DFR as low as expected by state-of-the-art lattice schemes, we define DFR targets compatible with those adopted by Saber [43], another scheme considered promising by NIST . Concretely, we adopt the DFR targets $\text{DFR}_1 = 2^{-120}$, $\text{DFR}_3 = 2^{-136}$, and $\text{DFR}_5 = 2^{-165}$, for the corresponding NIST security levels 1, 3, and 5.

Figure 6 shows that, when we decrease the value of d_v or d_u by 1 for a given q, the DFR gets higher, following the dashed lines for all security levels of Kyber. From the definition, lower d_u or d_v facilitates shorter ciphertext sizes. This flexibility on the value of q, enabled by our family of incomplete NTTs, is useful when one desires to achieve a particular DFR for a specific application context. For instance, this feature comes in handy if we need a compressed ciphertext size. We have to lower the value d_u or d_v. However, as we observed, lowering d_u or d_v increases the DFR. We can increase the modulus from 3329 to compensate for that change in DFR and still support the minimum value of it.

6.3. Proposed parameter sets with new tradeoffs

Figure 7 shows that the proposed family of incomplete NTT members contributes new options for the balance between DFR and ciphertext compression for all security levels. The key factor of this contribution is enabling a broader and relaxed set of prime moduli $q = \{4001, 3137, 3457\}$ by introducing more incompleteness $\ell = \{4, 5, 6\}$, respectively, than the default selection with $q = 3329$ and $\ell = 7$. The corresponding new points are marked in yellow, red, and magenta, respectively, and the current point is marked in black.

We observe that, for each security level, there are several new points that fall under the corresponding DFR target. For instance, in level 1, there are three points of interest: two providing lower DFR than that of Kyber512 and one providing 4% ciphertext compression. Likewise, level 3 also has three additional points of interest, with one of them providing 3% smaller ciphertexts. In contrast,

5. https://github.com/pq-crystals/security-estimates

6. The (d_u, d_v) parameter gives the number of bits into which the coefficients from the two parts of the ciphertext are compressed.

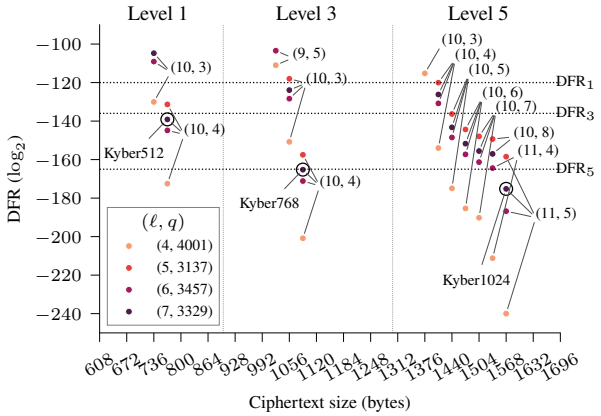


Figure 7: New tradeoffs concerning DFRs, ciphertext compressions for Kyber via NTT incompleteness. The plot shows DFR and ciphertext sizes for pairs of compression parameters (d_u, d_v) when using different moduli. (ℓ, q) pairs are illustrated in solid circles with different colors and (d_u, d_v) pairs are annotated in groups by arrows. The current Kyber setups for all security levels are marked by second circles. Two vertical dashed lines separate three regions, and three horizontal dashed lines showcase the target DFR values of three security levels.

there are multiple interesting possibilities in level 5 below DFR_5 that enable up to 8% shorter ciphertexts when using $\ell = 4$ out of 8 NTT levels. According to the underlying application requirements (e.g. shorter ciphertexts due to network latency or very low DFR for security reasons), a PQC practitioner can choose one of them.

In the specific case of Kyber1024, an interesting tradeoff exists for $(d_u = 10, d_v = 7, q = 3457, \ell = 6)$: the DFR remains below the target of 2^{-165}, while ciphertexts become 4% shorter than the standardized choice. In terms of performance, this choice of $q = 3457$ leads to an overhead of approximately 9% for keypair generation, encapsulation, and decapsulation (see Figure 18).

6.4. Impact on hybrid key exchange protocols

Transport Layer Security (TLS) is one of the most widely deployed security protocols for providing authentication, integrity, and confidentiality between communicating peers [47]. A key component of the TLS process is the key exchange, during which the client and server work to establish a shared secret that enables subsequent encrypted data exchanges [48]. In TLS 1.3, the key exchange process typically relies on the Elliptic-Curve Diffie Hellman Ephemeral (ECDHE) scheme [48]. Furthermore, a recommended strategy by Cloudflare, Google, etc. [36]–[38] for facilitating a gradual transition to quantum-resistant systems is to begin adopting hybrid schemes [49]. For instance, in hybrid key exchange, the classical and post-quantum components are computed separately, and the final shared secret is obtained by concatenating them [49]. This way, an adversary would need to break both the classical and post-quantum components to obtain the shared secret [47]. In particular, the draft standard "*draft-kwiatkowski-tls-ecdhe-mlkem-02*" proposes using ECDHE (either X25519 or secp256r1) in conjunction with the Kyber768 KEM for the key exchange process in TLS

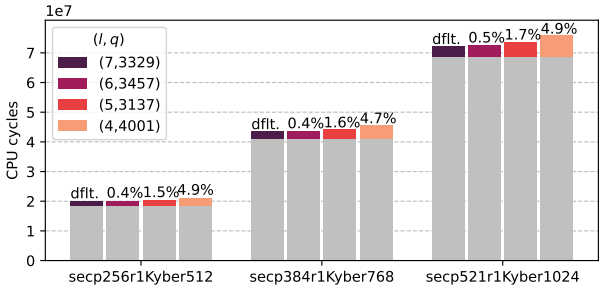


Figure 8: Total CPU cycle counts for hybrid key exchange operations on the TLS 1.3 client side using various NTT (ℓ, q) pairs in Kyber. The bottom and top portions of each bar represent the proportion of the total cycle count attributed to elliptic curve and Kyber operations, respectively. 'Dflt.' denotes the current standard, and the other percentages indicate additional overhead compared to the default.

1.3 [50]. The hybrid key exchange process in TLS 1.3 is illustrated in the call flow diagram in Appendix Figure 14.

To assess the impact of Kyber with varying NTT incompleteness settings in the context of TLS 1.3 with hybrid key exchange, we selected the (ℓ, q) pairs from Figure 7. Kyber instances with different (ℓ, q) pairs were evaluated in conjunction with the corresponding NIST curves at the same security level. For the actual experiments, we extended the Cortex-M4 experimental setup from Section 6, incorporating the NIST curves implementation from Mbed TLS[7]. Then, for both the client and server, we measured the cycle counts of the elliptic curve and Kyber operations involved in the key exchange process illustrated in Figure 14. Figure 8, as well as Figure 15 in Appendix H, show the cycle counts for each hybrid key exchange scheme, illustrating the effects of various (ℓ, q) combinations for Kyber on the client and server sides.

Our experiments indicate that the $(\ell, q) = (6, 3457)$ combination leads to a 0.4% average increase in CPU cycles on the client side and a 0.3% on the server side, compared to the default (7,3329) setting. For the $(5, 3137)$ pair, we observed an average increase of 1.5% in CPU cycles on the client side and 1.1% on the server side. Lastly, for $(4, 4001)$, the increase is around 4.9% and 3% for the client and server sides, respectively.

The results indicate that more incompleteness in NTT causes minimal computation impacts when we build a hybrid key exchange protocol with a classic counterpart. In expense of that, the download cost (of ciphertext of the shared secret) for the TLS 1.3 client is reduced by 4%, 3%, and 8% with the $(\ell = 4, q = 4001)$ pair (recap Figure 7) for 128-, 192-, and 256-bit security, respectively, of the hybrid scheme. Note that prior work [35] compresses Kyber ciphertext by only 6%, using a different method (namely, a 2-dimensional encoding).

More concretely, with $(\ell = 4, q = 4001)$, and $(d_u = 10, d_v = 5)$, for the kyber1024-based TLS 1.3 hybrid key exchange, we save 1024 bits, at the cost of an increase in total time (i.e., both client- and server-side) of 29.43 milliseconds (setting the clock speed of the STM32F4 board of ARM Cortex-M4 at 180MHz). In that case, the handshake should be faster if the bandwidth of the network system is lower than $1024/29.43 = 34.8$ kbps. Otherwise, the handshake will theoretically be slower, although the break-even may actually appear at larger bandwidths due to other practical factors: in networks with limited maximum transmission unit (MTU), larger payloads may require a larger number of packets, so our size reduction would be beneficial; also, noisy networks usually benefit from smaller payloads, since the probability of errors in packets grows with the packet size.

7. https://github.com/Mbed-TLS/mbedtls/releases/tag/mbedtls-3.4.0

7. Case Study: Amortized Bootstrapping

We analyze the impact of incomplete NTT in a practical scenario: an FHE amortized bootstrapping algorithm. This choice is motivated by the growing relevance of such algorithms in the scientific literature, especially in recent years [34], [51]–[54]. State-of-the-art schemes, such as FHEW/TFHE [12], [14], refresh a single (encrypted) message per bootstrapping operation. While the resulting bootstrapping algorithm is relatively fast compared to precedent schemes, other FHE schemes, such as BGV/BFV [15], [55], support batching techniques. This means they can amortize the cost of bootstrapping over multiple packed messages within a single (larger) ciphertext, achieving higher efficiency. In light of this observation, some studies have proposed amortized bootstrapping algorithms for FHEW/TFHE-like schemes aiming to improve their bootstrapping costs.

The algorithm introduced by De Micheli et al. [34] is an example of such an amortized bootstrapping algorithm. Their proposal packs many messages into a single ciphertext using a ring structure. Bootstrapping this ring-based ciphertext then relies on polynomial multiplications, which can leverage NTT-based algorithms. In fact, [34] does apply an incomplete NTT to enhance performance, although their concrete parameter analysis is somewhat limited. Thus, it becomes natural to consider their algorithm as a case study and systematically explore different configurations to achieve optimal results. Specifically, we hereby examine new values for the incompleteness degree ℓ and modulus q, considering security levels from 128 to 512 bits, and provide detailed performance figures.

Bootstrapping with packed LWE ciphertexts. To better evaluate our results, it is useful to detail the amortized bootstrapping algorithm further proposed in [34]. Essentially, it can be divided into three main steps:

1) A packing step: it takes some LWE ciphertexts as input and "packs" them into a single RLWE ciphertext, denoted as $(\mathbf{a}, \mathbf{b}) \in \mathcal{R}_q \times \mathcal{R}_q$, where $\mathcal{R}_q$ is the dth cyclotomic ring modulo some prime q.
2) A homomorphic decryption step: at the core of the bootstrapping procedure, it computes the encryption of the ring element $(\mathbf{a} \cdot \mathbf{z} + \mathbf{b}) \in \mathcal{R}_q$, where a (gadget) RLWE encryption of the secret key $\mathbf{z}$ is known.
3) An extraction step: the LWE ciphertexts are recovered with reduced noise.

We focus on the second step, the homomorphic decryption step, as we are interested in the homomorphic computation of the polynomial multiplication $\mathbf{a} \cdot \mathrm{Enc}(\mathbf{z})$. As mentioned above, the proposed algorithm uses the notion of incomplete NTT (called "partial FFT" in [34]) for better

performance. The polynomial multiplication $\mathbf{a}\cdot\mathrm{Enc}(\mathbf{z})$ can then be expressed as

$$\mathrm{invNTT}^{\mathrm{H}}_{(\ell,k)}(\mathrm{NTT}_{(\ell,k)}(\mathbf{a}) \overset{\mathrm{H}}{\star} \mathrm{Enc}\left(\mathrm{NTT}_{(\ell,k)}(\mathbf{z})\right)),$$

where $\overset{\mathrm{H}}{\star}$ denotes the homomorphic multiplication of elements in the NTT domain and $\mathrm{invNTT}^{\mathrm{H}}_{(\ell,k)}$ is a homomorphic implementation of the $\mathrm{invNTT}_{(\ell,k)}$ operation.

Besides working with the $\mathcal{R}_q$ ring, the algorithm also involves the qth cyclotomic ring modulo some $Q > 0$, denoted by $\mathcal{R}_{\mathrm{reg}}$ in [34]. This is the ring used by the so-called cryptographic registers in the FHEW framework [14]. A register corresponds to a gadget RLWE (denoted RLWE$'$) or RGSW encryption of a monomial $X^m \in \mathcal{R}_{\mathrm{reg}}$ for $m \in \mathbb{Z}_q$. Essentially, using these registers means that all computations are performed "in the exponent." By encrypting values in the exponent, FHEW leverages the algebraic structure of cyclotomic rings to perform fast monomial multiplications. This allows for the efficient implementation of the blind rotation in bootstrapping.

Going back to the polynomial multiplication in step 2, we notice that it uses an incomplete NTT (in the clear) to obtain $\mathrm{NTT}_{(\ell,k)}(\mathbf{a})$, as well as an inverse homomorphic incomplete NTT to ultimately obtain $\mathbf{a}\cdot\mathrm{Enc}(\mathbf{z})$. For the last component, the bootstrapping key will contain RGSW registers of $\mathrm{NTT}_{(\ell,k)}(\mathbf{z})$. The resulting output are registers encoding the coefficients of $\mathbf{a}\cdot\mathbf{z}+\mathbf{b}$, i.e., RLWE ciphertexts encrypting $X^{(\mathbf{a}\cdot\mathbf{z}+\mathbf{b})_i}$ where $(\mathbf{a}\cdot\mathbf{z}+\mathbf{b})_i$ denotes the ith coefficient. We refer to [34] for additional details, in particular for the homomorphic multiplication which needs to be adapted to the use of an incomplete NTT (see [34, Algorithm 2]).

Better bootstrapping performance with alternative NTT incompleteness settings. To analyze the performance of amortized bootstrapping under different settings, we first recall the core operation used in [34] to estimate its cost: the scalar multiplication by arbitrary ring elements, which is used as a building block for other procedures. This operation, denoted as $\mathcal{R}_{\mathrm{reg}} \odot \mathrm{RLWE}'$ when performed in the registers, multiplies an RLWE$'$ ciphertext by a ring element $\mathbf{r} \in \mathcal{R}_{\mathrm{reg}}$. More formally, $\odot : \mathcal{R}_{\mathrm{reg}} \times \mathrm{RLWE}' \to \mathrm{RLWE}$ is defined as

$$\mathbf{r} \odot \mathrm{RLWE}'(\mathbf{m}) = \sum_{i=0}^{d_B-1} \mathbf{r}_i \cdot \mathrm{RLWE}(v_i \cdot \mathbf{m}) = \mathrm{RLWE}(\mathbf{r}\cdot\mathbf{m}),$$

RLWE$'$, in turn, is defined as $\mathrm{RLWE}'(\mathbf{m}) = (\mathrm{RLWE}(v_0\cdot\mathbf{m}), \cdots, \mathrm{RLWE}(v_{d_B-1}\cdot\mathbf{m}))$ for a gadget vector $v = (v_0, \cdots, v_{d_B-1})$ of length d_B and the ring element $\mathbf{r} = \sum_{i=0}^{d_B-1} v_i \cdot \mathbf{r}_i$, which can also be decomposed using the same gadget vector.

All other operations in the registers used in the algorithm, namely ciphertext-ciphertext multiplication, ring automorphisms, and scheme-switching, can be described using the $\odot$ multiplication [34, Table 2]. The homomorphic invNTT itself relies on such operations and, thus, can also be described using the ring scalar multiplication $\odot$.

Given these considerations, the cost of the algorithm can be estimated by the number of $\odot$ operations performed per register (e.g., as reported in [34, Table 3]). Once we have the total $\odot$ count for the entire algorithm, its overall cost (our y-axis in Figure 9) is estimated as follows.

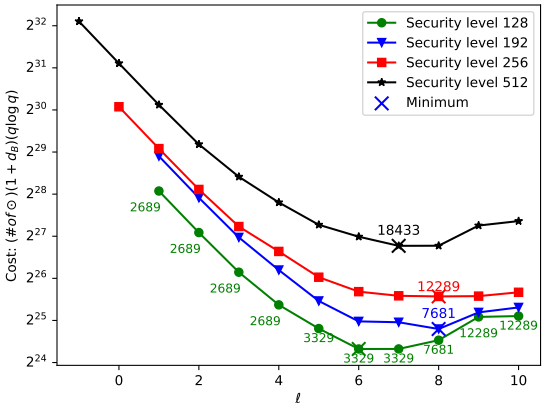


Figure 9: Theoretical performance – in the order of (number of $\odot$)$(1+d_B)(q\lg q)$ – evaluation for the amortized bootstrapping algorithm from [34]. For each security level $\lambda = 128, 192, 256, 512$, the best performance is achieved with an incomplete-NTT where $\ell = 6, 8, 8, 7$, respectively. We report each case's optimal modulus q by the $\times$ mark.

First, note that each $\odot$ product requires exactly (d_B+1) NTT operations, where d_B is the number of elements in the gadget vector. Moreover, each NTT operates on a q-dimensional vector, as we are looking at elements of $\mathcal{R}_{\mathrm{reg}}$, leading to a total computational complexity of $q\lg q$. Combining these costs, the overall performance of the algorithm can be estimated as (number of $\odot$)$(1+d_B)(q\lg q)$. This equation reveals the impact of q on the performance of the bootstrapping algorithm: it shows that adopting a smaller modulus, as allowed by some incomplete NTT settings (see Section 4.1), can reduce the costs of this operation. This is exactly what motivates our analyses conducted in this section.

When optimizing the overall cost of the bootstrapping algorithm, some security constraints must be taken into account. Specifically, for a target security level, the amount of LWE ciphertexts $\phi(d)$ that can be securely packed into an RLWE ciphertext depends on the dth cyclotomic ring $\mathcal{R}_q$ modulo q. Suitable parameters satisfying these security constraints are determined using an LWE estimator [56]. Considering these constraints, parameters such as the incompleteness degree ℓ, the length of the gadget d_B, and the radix decomposition of the NTT can then be chosen to optimize operation count and error growth.

Figure 9 shows the overall costs of bootstrapping for different incompleteness degrees ℓ and security levels — 192, 256, and 512 are new with respect to [34]. We can observe that the best performance is always achieved when an incomplete NTT is used with parameter $\ell = 6, 8, 8, 7$ for the security levels $\lambda = 128, 192, 256, 512$, respectively. The smallest (possible) value of q is chosen for each corresponding parameter set. We note that a complete NTT (*i.e.,* $\ell = 10$) forces the modulus $q = 12,289$ for security levels 128, 192, and 256, and up to $q = 40,961$ for security level 512. This leads to quite expensive computations. For $\lambda = 128, 192, 512$, the optimal values of q are $3329, 7681, 18433$, respectively. This ability to work with smaller moduli, as well as the significant size gap between the moduli optimized by incompleteness and those constrained by complete-NTT, results in a performance gain

of $33\%, 29\%$ and 42% for security levels $\lambda = 128, 192$ and 512. In contrast, for $\lambda = 256$, the optimal modulus remains $q = 12{,}289$ even in the incomplete NTT scenario. Consequently, the performance gain from incompleteness is less significant, although still at 6%.

For $\lambda = 128$, we report in Figure 9 the value of q for each ℓ to illustrate the best available choice of the modulus. De Micheli et al. only reported the results for $q = 7681$ in [34, Table 6] which matches our number for this specific value of q. We note that more incompleteness (e.g., $\ell = 1$ to 4) leads to an even smaller modulus $q = 2689$. However, in these cases, the base multiplication becomes too expensive to counter-balance the smaller modulus. Appendix I shows additional plots for the tradeoffs between the size of q and the number of scalar multiplications.

Finally, our experimental results shown in Figure 3c suggest that adopting larger values of n in FHE schemes (e.g., 1024) may limit the performance improvement obtained from NTT incompleteness. However, this study demonstrates that an incomplete NTT not only applies to cleartext computation, which may offer a minor speedup, but also provides significant advantages in algorithms where NTT is performed homomorphically.

8. Related work

Many studies in the literature evaluate and optimize the performance of NTT in resource-constrained devices, commonly through assembly tricks and software-hardware co-design [20]–[27]. The PQClean project [44], for example, is a repository of highly tested, reliable implementations in C. The pqm4 project [20] goes even further, providing various implementations tailored for the Cortex-M4, including reference, clean, and highly optimized implementations leveraging assembly instructions and floating-point registers. Abdulrahman et al. [21] further improve efficiency by using a smaller modulus in Dilithium to enable Fermat transforms, and by applying Cooley–Tukey butterflies for inverse NTT in both Kyber and Dilithium.

In the software-hardware co-design front, Abdulrahman et al. [21] leveraged floating-point registers to cache values in the NTT and used asymmetric multiplication to reduce redundant computations in Kyber. Güneysu et al. [57] focused on optimizing GLP, BLISS, and Dilithium by incorporating unrolling and inlining coding strategies in NTT calls to accelerate the signing procedure. Greconici et al. [58] presented constant-time implementations of Dilithium, using techniques such as lazy reduction and signed polynomial representation to prevent underflows and optimize modular arithmetic. Botros et al. [23] and Alkim et al. [24] introduced optimizations for Kyber, such as exploiting DSP instructions, reducing RAM usage without sacrificing performance, and applying link-time optimization. They also introduced signed Montgomery reduction and precomputation of twiddle factors in Montgomery representation to streamline NTT computation.

A few works [28]–[33], similarly to ours, state the benefits of stopping a few layers earlier than a regular NTT. However, we could not find any study exploring and formally defining the whole spectrum of NTT incompleteness or the potential performance gains from adopting a much smaller modulus in an incomplete setting.

At the same time, one of the main contributions of our work consists in a method that enables finding the optimal order of incompleteness, something that was not explored in the realm of post-quantum cryptography until very recently [30], [59]. For example, the thesis of Lips [30] explores optimizations of polynomial multiplication for embedded platforms, comparing state-of-the-art options with a focus on regular addition and multiplication costs as efficiency metrics. The thesis discusses that regular NTT outperforms these options and states that incomplete NTT may offer even greater efficiency and adaptability. However, exploring the full potential of incompleteness, such as applying Schoolbook at an earlier stage of the NTT, is left as future work. In contrast, our work explores the full spectrum of incompleteness following a formal representation while also including ciphertext compression as an additional metric in our analysis.

Compared to related studies, our systematic method is flexible and quite general, so it can be adapted to optimize polynomial multiplication in a wide range of applications and platforms. Furthermore, since our proposal is more fundamental and broad in nature, it can be combined with most of the prior instances of optimizing assembly-level and software-hardware co-designs.

9. Concluding remarks

We formally analyzed incompleteness in the number-theoretic transform (NTT) and demonstrated that (ℓ, k)-layer incomplete NTTs outperform complete NTT for polynomial multiplication in a quotient ring, a key operation in lattice-based cryptography. This enhances the efficiency of PQC and FHE implementations. While prior works have explored NTT incompleteness, our study provides the first systematic evaluation across the full parameter spectrum, identifying optimal values for various application constraints.

Besides performance enhancement, the introduced incompleteness in NTT enables a broader and more relaxed set of prime moduli than complete NTT. With smaller prime moduli, we observed a significant performance boost. We thoroughly examined NIST-approved module lattice-based key encapsulation mechanism (ML-KEM) Crystals-Kyber to showcase the impact of incompleteness on performance and proposed new parameter sets, achieving smaller ciphertext sizes and lower decryption failure rates (DFR) than the current standard. This comes with a modest performance hit but becomes negligible compared to the gains in compressing the size of the secret share when we assessed the TLS 1.3 hybrid key exchange – a combination of classical and post-quantum algorithms deployed by industries.

We also showcased the optimal performance of one of the state-of-the-art amortized bootstrapping algorithms for FHEW-style schemes achieved by incompleteness in NTT, leveraging a smaller modulus that is varied by the overall security strength of the application. Both of the case studies will facilitate theoreticians and practitioners alike to improve performance and explore new tradeoffs of lattice-based cryptographic applications across different computing platforms.

Acknowledgment

The authors would like to thank the anonymous reviewers and, in particular, the shepherd for their valuable feedback, which helped improve the exposition of this paper.

References

[1] NIST, "Module-lattice-based key-encapsulation mechanism standard," tech. rep., National Institute of Standards and Technology, Department of Commerce, Washington, D.C., 2024. Federal Information Processing Standards Publication (FIPS 203).

[2] NIST, "Module-lattice-based digital signature standard," tech. rep., National Institute of Standards and Technology, Department of Commerce, Washington, D.C., 2024. Federal Information Processing Standards Publication (FIPS 204).

[3] C. Gentry, "Fully homomorphic encryption using ideal lattices," in *Proc. of the 41st annual ACM symposium on Theory of computing*, pp. 169–178, 2009.

[4] R. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Communications of the ACM*, vol. 21, no. 2, pp. 120–126, 1978.

[5] W. Diffie and M. Hellman, "New directions in cryptography," in *Democratizing Cryptography: The Work of Whitfield Diffie and Martin Hellman*, pp. 365–390, 2022.

[6] NIST, "Quantum-resistant cryptography technology interoperability and performance report (preliminary draft)," tech. rep., National Institute of Standards and Technology, Department of Commerce, Washington, D.C., 2023. Special Publication (SP 1800-38C).

[7] P. Shor, "Algorithms for quantum computation: discrete logarithms and factoring," in *Proc. 35th annual symposium on foundations of computer science*, pp. 124–134, IEEE, 1994.

[8] NIST, "Post-quantum cryptography - workshops and timeline." https://csrc.nist.gov/projects/post-quantum-cryptography/workshops-and-timeline, 2016.

[9] P. Fouque, J. Hoffstein, P. Kirchner, V. Lyubashevsky, T. Pornin, T. Prest, *et al.*, "Falcon: Fast-fourier lattice-based compact signatures over NTRU," *Submission to the NIST's post-quantum cryptography standardization process*, vol. 36, no. 5, pp. 1–75, 2018.

[10] D. Micciancio and O. Regev, *Lattice-based Cryptography*, pp. 147–191. Berlin, Heidelberg: Springer, 2009.

[11] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "(Leveled) fully homomorphic encryption without bootstrapping," *ACM Transactions on Computation Theory*, vol. 6, no. 3, pp. 1–36, 2014.

[12] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène, "TFHE: fast fully homomorphic encryption over the torus," *Journal of Cryptology*, vol. 33, no. 1, pp. 34–91, 2020.

[13] J. Cheon, A. Kim, M. Kim, and Y. Song, "Homomorphic encryption for arithmetic of approximate numbers," in *Advances in Cryptology – ASIACRYPT 2017*, pp. 409–437, Springer, 2017.

[14] L. Ducas and D. Micciancio, "FHEW: bootstrapping homomorphic encryption in less than a second," in *Advances in Cryptology – EUROCRYPT 2015*, pp. 617–640, Springer, 2015.

[15] J. Fan and F. Vercauteren, "Somewhat practical fully homomorphic encryption," *Cryptology ePrint Archive*, 2012.

[16] A. Lau, P. Zinterhof, and M. Feldbacher, "Parallel implementation of fast algorithms for good lattice points," *PACT Deliverable D5Z-1a*, 1994.

[17] J. Lee, P. Duong, and H. Lee, "Configurable encryption and decryption architectures for CKKS-based homomorphic encryption," *Sensors*, vol. 23, no. 17, p. 7389, 2023.

[18] J. Yan, S. Huo, T. Yu, C. Zhang, X. Chai, D. Hu, and K. Yan, "Multiscale analysis for 3d lattice structures based on parallel computing," *International Journal for Numerical Methods in Engineering*, vol. 122, no. 22, pp. 6756–6776, 2021.

[19] S. Kim, W. Jung, J. Park, and J. Ahn, "Accelerating number theoretic transformations for bootstrappable homomorphic encryption on GPUs," in *2020 IEEE International Symposium on Workload Characterization (IISWC)*, pp. 264–275, IEEE, 2020.

[20] M. Kannwischer, J. Rijneveld, P. Schwabe, and K. Stoffelen, "pqm4: Testing and benchmarking NIST PQC on ARM Cortex-M4." https://github.com/mupq/pqm4.

[21] A. Abdulrahman, V. Hwang, M. Kannwischer, and A. Sprenkels, "Faster Kyber and Dilithium on the Cortex-M4," in *Proc. of the 20th International Conference Applied Cryptography and Network Security (ACNS'22)*, pp. 853–871, Springer, 2022.

[22] P. Ravi, S. Gupta, A. Chattopadhyay, and S. Bhasin, "Improving speed of Dilithium's signing procedure," in *Smart Card Research and Advanced Applications (CARDIS)*, pp. 57–73, Springer, 2020.

[23] L. Botros, M. Kannwischer, and P. Schwabe, "Memory-efficient high-speed implementation of Kyber on Cortex-M4," in *Progress in Cryptology–AFRICACRYPT 2019*, pp. 209–228, Springer, 2019.

[24] E. Alkim, Y. Bilgin, M. Cenk, and F. Gérard, "Cortex-M4 optimizations for {R, M} LWE schemes," *IACR Trans. on Cryptographic Hardware and Embedded Systems*, pp. 336–357, 2020.

[25] B. Hanno, H. Vincent, M. Kannwischer, Y. Bo-Yin, and Y. Shang-Yi, "Neon NTT: faster Dilithium, Kyber, and Saber on Cortex-A72 and Apple M1," *IACR Trans. on Cryptographic Hardware and Embedded Systems*, pp. 221–244, 2021.

[26] P. Sanal, E. Karagoz, H. Seo, R. Azarderakhsh, and M. Mozaffari-Kermani, "Kyber on ARM64: Compact implementations of Kyber on 64-bit ARM Cortex-A processors," in *Security and Privacy in Communication Networks*, pp. 424–440, Springer, 2021.

[27] D. Nguyen and K. Gaj, "Optimized software implementations of CRYSTALS-Kyber, NTRU, and Saber using NEON-based special instructions of ARMv8," in *Proc. of the NIST 3rd PQC Standardization Conference (NIST PQC)*, 2021.

[28] R. Avanzi, J. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. Schanck, P. Schwabe, G. Seiler, and D. Stehlé, "CRYSTALS-Kyber algorithm specifications and supporting documentation," *NIST PQC Round*, vol. 2, no. 4, pp. 1–43, 2019.

[29] C. Chung, V. Hwang, M. Kannwischer, G. Seiler, C. Shih, and B. Yang, "NTT multiplication for NTT-unfriendly rings: New speed records for Saber and NTRU on Cortex-M4 and AVX2," *IACR Trans. on Cryptographic Hardware and Embedded Systems*, pp. 159–188, 2021.

[30] B. Lips, "The efficiency of polynomial multiplication methods for ring-based PQC algorithms of round 3 of the NIST PQC competition," Master's thesis, Eindhoven Univ. of Technology, 2021.

[31] E. Alkım, Y. Bilgin, and M. Cenk, "Compact and simple rlwe based key encapsulation mechanism," in *Progress in Cryptology–LATINCRYPT 2019*, pp. 237–256, Springer, 2019.

[32] R. Crandall and C. Pomerance, *Prime numbers: a computational perspective*, vol. 2. Springer, 2005.

[33] R. Moenck, "Practical fast polynomial multiplication," in *Proc. of the 3rd ACM symposium on Symbolic and algebraic computation*, pp. 136–148, ACM, 1976.

[34] G. de Micheli, D. Kim, D. Micciancio, and A. Suhl, "Faster amortized FHEW bootstrapping using ring automorphisms," in *IACR Int. Conf. on Public-Key Cryptography*, pp. 322–353, Springer, 2024.

[35] T. Paiva, M. Simplicio, S. Hafiz, B. Yildiz, and E. Cominetti, "Tailoring two-dimensional codes for structured lattice-based kems and applications to kyber," *Cryptology ePrint Archive*, 2024.

[36] Cloudflare, "Experiment with post-quantum cryptography today." https://blog.cloudflare.com/experiment-with-pq/, 2022.

[37] Google, "Advancing our amazing bet on asymmetric cryptography." https://blog.chromium.org/2024/, 2024.

[38] B. Westerbaan and D. Stebila, "X25519Kyber768Draft00 hybrid post-quantum key agreement," internet-draft, RFC Editor, 2023.

[39] D. J. Bernstein, "Multidigit multiplication for mathematicians," *Advances in Applied Mathematics*, vol. 12, pp. 1–19, 2001.

[40] J. Cooley and J. Tukey, "An algorithm for the machine calculation of complex fourier series," *Mathematics of computation*, vol. 19, no. 90, pp. 297–301, 1965.

[41] W. M. Gentleman and G. Sande, "Fast fourier transforms: for fun and profit," in *Proceedings of the November 7-10, 1966, fall joint computer conference*, pp. 563–578, 1966.

[42] Arm Developer, "Arm processor instructions." https://developer.arm.com/documentation/100165/0201/Programmers-Model/Instruction-set-summary/Processor-instructions, 2024.

[43] J. D'Anvers, A. Karmakar, S. Sinha Roy, and F. Vercauteren, "SABER: Mod-LWR based KEM (Round 3 Submission)," *Submission to the NIST post-quantum project*, 2020.

[44] M. Kannwischer, P. Schwabe, D. Stebila, and T. Wiggers, "Improving software quality in cryptography standardization projects," in *IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, pp. 19–30, IEEE, 2022.

[45] Q. Guo, T. Johansson, and J. Yang, "A novel CCA attack using decryption errors against LAC," in *Advances in Cryptology – ASIACRYPT 2019*, pp. 82–111, Springer, 2019.

[46] J. D'Anvers, Q. Guo, T. Johansson, A. Nilsson, F. Vercauteren, and I. Verbauwhede, "Decryption failure attacks on IND-CCA secure lattice-based schemes," in *Public-Key Cryptography–PKC 2019*, pp. 565–598, Springer, 2019.

[47] N. Alnahawi, J. Müller, J. Oupický, and A. Wiesmaier, "A comprehensive survey on post-quantum TLS," *IACR Communications in Cryptology*, vol. 1, no. 2, 2024.

[48] E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.3." RFC 8446, Aug. 2018.

[49] D. Stebila, S. Fluhrer, and S. Gueron, "Hybrid key exchange in TLS 1.3," internet-draft, RFC Editor, 2024. Work in Progress.

[50] K. Kwiatkowski, P. Kampanakis, B. Westerbaan, and D. Stebila, "Post-quantum hybrid ECDHE-MLKEM Key Agreement for TLSv1.3," internet-draft, RFC Editor, 2024. Work in Progress.

[51] A. Guimarães, H. Pereira, and B. Van Leeuwen, "Amortized bootstrapping revisited: Simpler, asymptotically-faster, implemented," in *Advances in Cryptology – ASIACRYPT 2023*, pp. 3–35, Springer, 2023.

[52] F. Liu and H. Wang, "Batch bootstrapping I: A new framework for SIMD bootstrapping in polynomial modulus," in *Advances in Cryptology – EUROCRYPT 2023*, pp. 321–352, Springer, 2023.

[53] F. Liu and H. Wang, "Batch bootstrapping II: bootstrapping in polynomial modulus only requires O(1) FHE multiplications in amortization," in *Advances in Cryptology – EUROCRYPT 2023*, pp. 353–384, Springer, 2023.

[54] Z. Liu and Y. Wang, "Amortized functional bootstrapping in less than 7 ms, with O(1) polynomial multiplications," in *Advances in Cryptology – ASIACRYPT 2023*, pp. 101–132, Springer, 2023.

[55] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "Fully homomorphic encryption without bootstrapping." Cryptology ePrint Archive, Paper 2011/277, 2011.

[56] M. R. Albrecht, R. Player, and S. Scott, "On the concrete hardness of learning with errors," *Journal of Mathematical Cryptology*, vol. 9, no. 3, pp. 169–203, 2015.

[57] T. Güneysu, M. Krausz, T. Oder, and J. Speith, "Evaluation of lattice-based signature schemes in embedded systems," in *Int. Conf. on Electronics, Circuits and Systems*, pp. 385–388, IEEE, 2018.

[58] D. Greconici, M. Kannwischer, and D. Sprenkels, "Compact Dilithium implementations on Cortex-M3 and Cortex-M4," *IACR Trans. on Cryptographic Hardware and Embedded Systems*, pp. 1–24, 2020.

[59] M. Kannwischer, *Polynomial multiplication for post-quantum cryptography*. PhD thesis, Radboud Universiteit Nijmegen, 2022.

[60] S. Bhasin, J. D'Anvers, D. Heinz, T. Pöppelmann, and M. Van Beirendonck, "Attacking and defending masked polynomial comparison for lattice-based cryptography," *IACR Trans. on Cryptographic Hardware and Embedded Systems*, pp. 334–359, 2021.

[61] O. Bronchain and G. Cassiers, "Bitslicing arithmetic/boolean masking conversions for fun and profit: with application to lattice-based kems," *IACR Trans. on Cryptographic Hardware and Embedded Systems*, pp. 553–588, 2022.

[62] T. Tosun and E. Savas, "Zero-value filtering for accelerating non-profiled side-channel attack on incomplete ntt based implementations of lattice-based cryptography," *IEEE Trans. on Information Forensics and Security*, 2024.

Appendix A. Data availability

We provide all necessary artifacts – including Python scripts, C++ source code, and evaluation results for ARM Cortex-M4 microcontrollers – to reproduce all figures and tables presented in the main text and appendices. These resources are available at: https://github.com/smhafiz/incompleteness_in_ntts_new_tradeoffs_lattice_crypto.git.

Appendix B. Baseline comparison

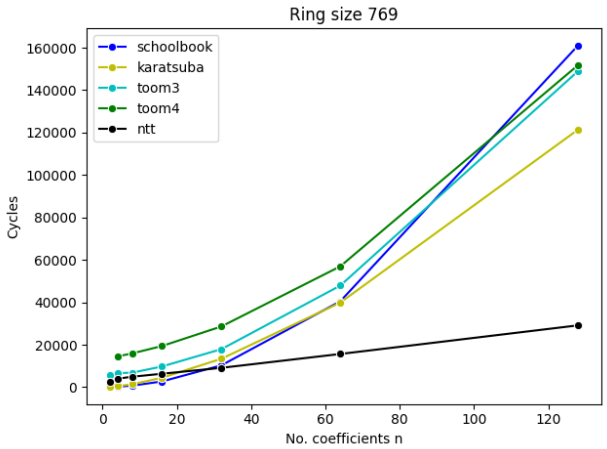


Figure 10: Consumed CPU cycles in our C testbed, averaged over 100 runs, for baseline polynomial multiplication in a quotient ring, $\mathsf{PolyMul}_2^{\mathbb{Z}_{769}}$ to $\mathsf{PolyMul}_{128}^{\mathbb{Z}_{769}}$, for $2 \leq n \leq 128$. PolyMul is defined in Definition 4. We observe that when $n \leq 30$, Schoolbook/Karatsuba is more efficient than NTT.

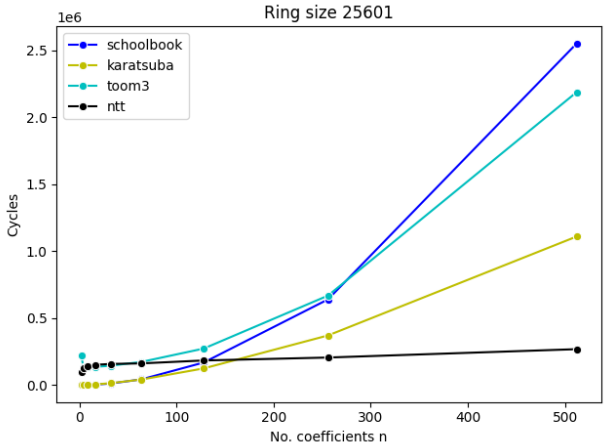


Figure 11: Consumed CPU cycles in our C testbed, averaged over 100 runs, for baseline polynomial multiplication in a quotient ring, $\mathsf{PolyMul}_2^{\mathbb{Z}_{25601}}$ to $\mathsf{PolyMul}_{512}^{\mathbb{Z}_{25601}}$, for $2 \leq n \leq 512$. PolyMul is defined in Definition 4. We observe that when $n \leq 30$, Schoolbook/Karatsuba is more efficient than NTT.

Appendix C. Preliminaries

This section reviews pertinent properties of cyclotomic polynomials and how they impact techniques like the Chi-

nese remainder theorem (CRT) and the number-theoretic transform (NTT).

C.1. Cyclotomics

For $m \geq 1$, let $\omega_m \in \mathbb{C}$ be a primitive m-th root of unity. We can take for example $\omega_m = e^{\frac{2\pi i}{m}}$. It is clear that ω_m is a generator for the multiplicative group of all mth roots of unity.

From Number Theory, we know that if ω_m is a primitive root of unity, then ω_m^j is a primitive root of unity if and only if $GCD(j, m) = 1$ or equivalently, $j \in \mathbb{Z}_m^*$, i.e., the multiplicative group of reduced residue classes modulo m. Thus, the set of all primitive mth roots of the unit is given by $\{\omega_m^j | j \in \mathbb{Z}_m^*\}$.

We now take the polynomial $\Phi_m(x) = \prod_{j \in \mathbb{Z}_m^*}(x - \omega_m^j)$, which is a monic polynomial of degree $\phi(m)$, where ϕ is the Euler Totient function. This is called the mth *cyclotomic polynomial*. Some interesting facts about the cyclotomic polynomials are that (i) $\Phi_m(x) \in \mathbb{Z}[x]$, (ii) $\Phi_m(x) | x^m - 1$ in $\mathbb{Z}[x]$, (iii) $\Phi_m(x) \nmid x^k - 1$ in $\mathbb{Z}[x]$ for any $k < n$, and (iv) $\Phi_m(x)$ is irreducible over $\mathbb{Z}$.

As a special case, if $m = 2^{k+1}$ for some integer $k \geq 0$, then $\phi(m) = m/2 = 2^k$ and so $deg(\Phi_m(x)) = 2^k$. Since $\Phi_m(x) | x^m - 1 = (x^{2^k} - 1)(x^{2^k} + 1)$ and $deg(\Phi_m(x)) = k$ and $\Phi_m(x)$ is irreducible over $\mathbb{Z}$, we must have $\Phi_m(x) = x^{2^k} + 1$ in this case. The following theorem on cyclotomics and the subsequent corollary provides the basis for why they are useful in NTT:

Theorem 4. Let p be a prime number such that $p \nmid m$. Then $\Phi_m(x) = f_1(x) f_2(x) \dots f_\ell(x)$ in $\mathbb{Z}_p[x]$, where $f_i(x)$ are irreducible and pairwise relatively prime polynomials in $\mathbb{Z}_p[x]$. Moreover, each $f_i(x)$ is of degree d, where $d = \frac{\phi(m)}{\ell}$ and d is the order of p mod m. That is d is the smallest positive integer such that $p^d \equiv 1 \bmod m$.

An immediate corollary is the following:

Corollary 7. If $p \equiv 1 \bmod m$, then $\Phi_m(x)$ splits into distinct linear factors modulo p. In this case we will have $\Phi_m(x) = \prod_{i \in \mathbb{Z}_m^*}(x - a^i)$, where a is a primitive mth root of unity modulo p.

Example 3. Consider $p(x) = x^{16} + 1$, which is the 32nd cyclotomic polynomial ($m = 32$). If we take $q = 3$, then since $3^8 \equiv 1 \bmod 32$, we expect $x^{16} + 1$ to split into two factors of degree 8 each. Indeed we have

$$x^{16} + 1 = (x^8 + x^4 + 2)(x^8 + 2x^4 + 2)$$

in $\mathbb{Z}_3[x]$.

To find a prime q so that $x^{16} + 1$ splits completely in $\mathbb{Z}_q[x]$, we need to find a prime p such that $p \equiv 1 \bmod 32$. Such a p is 97. Indeed we have

$$\begin{aligned} x^{16} + 1 = &(x + 19)(x + 20)(x + 28)(x + 30)(x + 34) \\ &(x + 42)(x + 45)(x + 46)(x + 51)(x + 52) \\ &(x + 55)(x + 63)(x + 67) \\ &(x + 69)(x + 77)(x + 78) \end{aligned}$$

in $\mathbb{Z}_{97}[x]$.

The different ways that cyclotomics can split will come in handy to determine primes that will lead to a spectrum of cases of incomplete NTT.

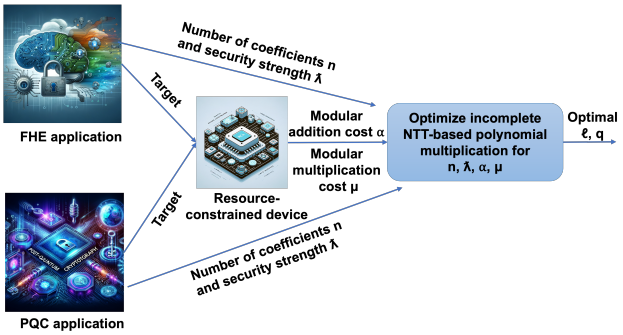


Figure 12: A bird's-eye view of the proposed method for optimizing NTT-based polynomial multiplication through incompleteness and a relaxed moduli set, considering constraints imposed by the target application and computing platform.

C.2. Chinese remainder theorem

We recall how the Chinese remainder theorem (CRT) works for polynomials and how this leads to the NTT. The following theorem is the main description of the CRT for polynomials over a finite field:

Theorem 5. Assume that F is a field and let $f(x) \in F[x]$ be a polynomial such that $f(x) = f_1(x) f_2(x) \cdots f_k(x)$ in $F[x]$, where $f_1, f_2, \dots, f_k$ are pairwise coprime polynomials. Then $F[x]/(f) \simeq F[x]/(f_1) \times F[x]/(f_2) \cdots \times F[x]/(f_k)$.

The forward direction in the isomorphism given above is the NTT algorithm, while the reverse direction in the isomorphism is the inverse NTT. The complete NTT occurs when all quotient polynomials f_i are degree 1 polynomials, i.e., linear polynomials.

Appendix D. Bird's eye view

Figure 12 gives an overview of the proposed optimization procedure. Our goal is to optimize polynomial multiplication in quotient rings based on the target application, such as PQC or FHE, which may impose constraints on parameters such as the number of coefficients (n) and minimum security strength (λ). Furthermore, different computing platforms have varying computational costs for modular additions (α) and multiplications (μ). To address these constraints, we seek a suitable procedure to determine the best incompleteness level (ℓ) and modulus (q). This is accomplished by solving Equation 7 in Section 4.3, which accounts for the constraints imposed by the target application and computing platform.

Appendix E. Impact of computing platform costs

TABLE 5: Theoretical polynomial multiplication costs when $\mu/\alpha = 1$, according to Equations 3 and 4.

n	Complete NTT	Incomplete NTT			Schoolbook
	$(\ell = k)$	$(\ell = k-1)$	$(\ell = k-2)$	$(\ell = k-3)$	$(\ell = 0)$
64	1856	1824	1792	2016	8256
128	4288	4224	4160	4608	32,896
256	9728	9600	9472	10,368	131,328
512	21,760	21,504	21,248	23,040	524,800

TABLE 6: Theoretical polynomial multiplication costs when $\mu/\alpha = 1$, according to Equations 3 and 4.

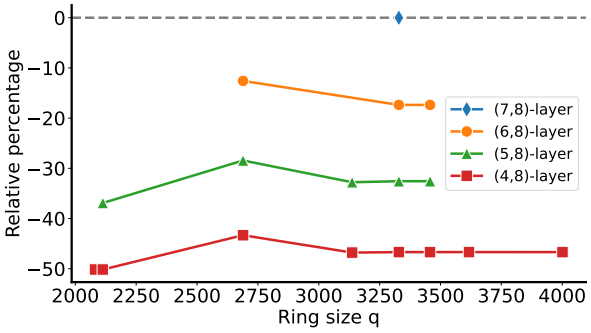
n	Complete NTT	Incomplete NTT			Schoolbook
	$(\ell = k)$	$(\ell = k-1)$	$(\ell = k-2)$	$(\ell = k-3)$	$(\ell = 0)$
64	8192	7872	8128	9792	45,120
128	18,688	18,048	18,560	21,888	180,352
256	41,984	40,704	41,728	48,384	721,152
512	93,184	90,624	92,672	105,984	2,884,096

Appendix F.
Side-channel attacks.

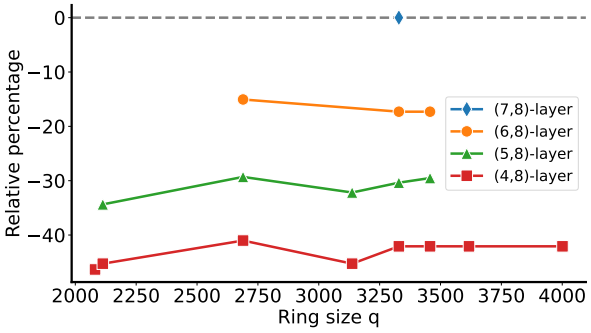
Side-channel attacks exploit leakage from cryptographic implementations, primarily via timing or physical channels (e.g., power, electromagnetic radiation). NIST mandates that PQC schemes be resistant to timing-based side-channel attacks. Therefore, the reference implementation of ML-KEM (Crystals-Kyber) is designed to be isochronous (constant-time) to prevent timing-based attacks. Similarly, our implementations are also isochronous, offering protection against timing attacks. However, Kyber's reference implementation does not address power or electromagnetic radiation-based physical side-channel attacks [60], [61], making it susceptible. This was not required by NIST, as typical defenses like shuffling and masking significantly slow down performance. Our incomplete NTT-based implementations also do not provide countermeasures against power or EM-based physical side-channel attacks [62].

Appendix G.
Optimizing performance and modulus: Kyber

Figure 13 showcases CPU cycles spent executing the incomplete NTT and invNTT in the modified PQClean reference implementation of Kyber on an ARM Cortex-M4 for different levels of incompleteness and selected modulus values. The Y-axis represents the relative overhead (in percentage) compared to Kyber's default (7,8)-layer NTT with $q = 3329$, marked as a diamond-shaped blue point with a 0% overhead.



(a) Incomplete NTT on Kyber testbed.



(b) Incomplete invNTT on Kyber testbed.

Figure 13: Optimizing performance and modulus for Kyber.

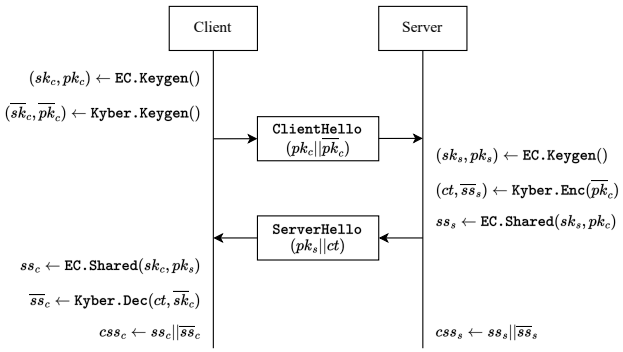


Figure 14: Hybrid key exchange process in TLS 1.3: the client and server's shared secrets (css_c, css_s) are obtained by concatenating the corresponding ECDHE and Kyber768 computed secrets.

Appendix H.
TLS 1.3 hybrid key exchange protocol

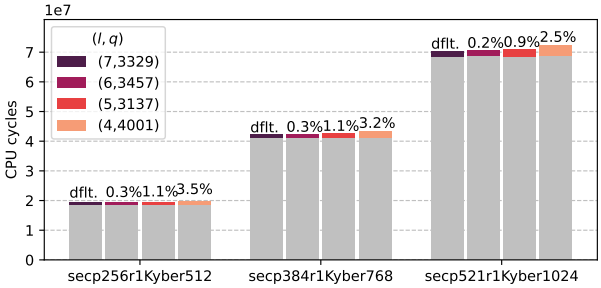


Figure 15: Total CPU cycle counts for hybrid key exchange operations on the TLS 1.3 server side using various NTT (ℓ, q) pairs in Kyber. The bottom portion of each bar represents the proportion of the total cycle count attributed to EC operations, while the top portion indicates the proportion of the total cycle count attributed to Kyber operations. 'Dflt.' denotes the current standard, and the other percentages illustrate additional overhead than the default.

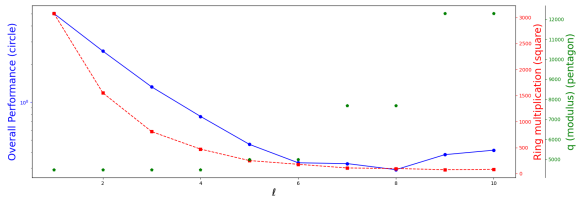
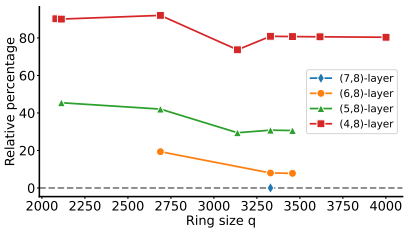


Figure 16: Overall theoretical performance, number of ring multiplication, and optimized choice of modulus q as a function of the incompleteness parameter ℓ for $\lambda = 192$.

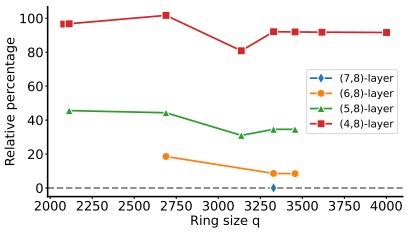
Appendix I.
Parameter choices: amortized bootstrapping

In Figure 16, we illustrate the overall performance of the amortized bootstrapping algorithm from [34] as a function of the incompleteness degree ℓ along with the number of scalar ring multiplications and moduli q for security level $\lambda = 192$ (similar figures can be generated for $\lambda = 128, 256$ and $\lambda = 512$). The overall performance is a function of both the modulus q and the number of ring multiplication. More precisely, the performance is evaluated as $\odot(1 + d_B)q \lg q$ where $\odot$ is the number of $\mathcal{R} \times$ RLWE multiplications and d_B is the gadget size. As expected, we can see that the more incompleteness one considers, the smaller the moduli q gets. Indeed, for $\ell = 10$, which corresponds to a complete-NTT, the choice of q is maximal, *i.e.,* $q = 12289$. However, as ℓ decreases, the choice of q can be optimized, and smaller values for the modulus are possible (for example, $q = 7681$ for $\ell = 7, 8$ down to $q = 4481$ for $\ell = 1, 2, 3, 4$). Moreover, the number of ring scalar multiplications $\mathcal{R} \times$ RLWE also varies with the degree of incompleteness. We refer to [34] for the precise analysis of the number of operations in each step of the algorithm and simply note that the number of ring multiplications tends to increase as incompleteness also increases since more operations are needed for the base multiplication. As already commented in the main body of the paper, the optimal performance is given for $\ell = 8$, which corresponds to $q = 7681$ and 98 $\mathcal{R} \times$ RLWE multiplications.

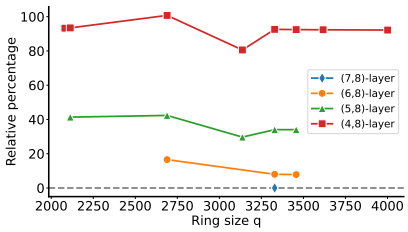
Appendix J.
Benchmarking Kyber768 and Kyber1024



(a) Kyber768 key pair generation.

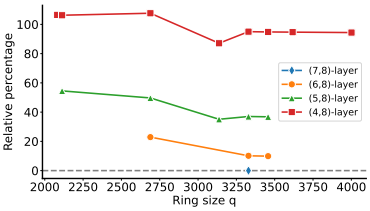


(b) Kyber768 key encapsulation.

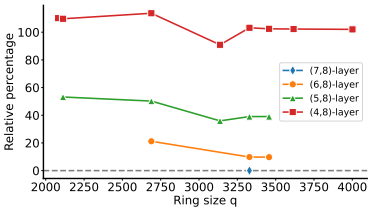


(c) Kyber768 key decapsulation.

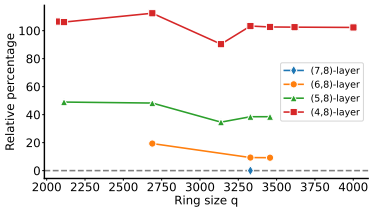
Figure 17: CPU cycles spent executing modified PQClean reference implementation of Kyber768 on an ARM Cortex-M4 device to enable various incompleteness in NTT/invNTT and associated and selected moduli. Along the Y-axis, we present the relative overhead in percentage of each case compared to the default case, i.e., (7,8)-layer NTT with $q = 3329$, marked by a diamond-shaped blue point with a 0% relative overhead.



(a) Kyber1024 key pair generation.



(b) Kyber1024 key encapsulation.



(c) Kyber1024 key decapsulation.

Figure 18: CPU cycles spent executing modified PQClean reference implementation of Kyber1024 on an ARM Cortex-M4 device to enable various incompleteness in NTT/invNTT and associated and selected moduli. Along the Y-axis, we present the relative overhead in percentage of each case compared to the default case, i.e., (7,8)-layer NTT with $q = 3329$, marked by a diamond-shaped blue point with a 0% relative overhead.