Single-Server Delegation of NTT with Application to Crystals-Kyber

Giovanni Di Crescenzo
Peraton Labs
Basking Ridge, NJ, 07920, USA
gdicrescenzo@peratonlabs.com

Matluba Khodjaeva
CUNY John Jay College of Criminal Justice
New York, NY, 10016, USA
mkhodjaeva@gradcenter.cuny.edu

Dilan D. Morales Caro
CUNY John Jay College of Criminal Justice
New York, NY, 10016, USA
dilan0803@gmail.com

Delaram Kahrobaei
CUNY Graduate Center
New York, NY, 10016, USA
dkahrobaei@gc.cuny.edu

ABSTRACT

The Number-Theory Transform (NTT) is an important operation used in many lattice-based public-key cryptosystems and, more generally, cryptographic protocols. Polynomial multiplication in quotient polynomial rings is often the most expensive computation in lattice-based public-key cryptosystems, including Crystals-Kyber, and many others among the submissions to the NIST Post-Quantum Cryptography Standardization competition. NTT-based polynomial multiplication does significantly improve runtime with respect to other techniques, and yet, it remains the most expensive operation and is in need for optimizations and improvements, especially for implementation on resource-constrained devices.

In this paper we study the problem of a resource-constrained client delegating NTT computations to a single, computationally stronger, server, possibly implemented in a cloud-based architecture. We show several protocols for a client to delegate single/batch computations of the NTT and NTT^{-1} functions, in input scenarios where the input polynomial(s) are private or public, where the client performs a smaller number of multiplications in $\mathbb{Z}_q$ with respect to non-delegated NTT or NTT^{-1} computations. The improvements are both of asymptotic nature, for a large range of the involved parameters, and are noticeable for practical parameter settings. Our implementation achieves one-order-of-magnitude improvements for NTT computations, and manage to improve the client online runtime for the algorithms in the already significantly optimized Crystals-Kyber cryptosystem, which is currently recommended for standardization.

CCS CONCEPTS

• **Security and privacy** → *Public key encryption*; *Mathematical foundations of cryptography*; *Privacy-preserving protocols.*

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
CCSW'24, October 18, 2024, Salt Lake City, UT, USA
© 2024 Association for Computing Machinery.
ACM ISBN 978-1-4503-5204-8/17/11…$15.00
https://doi.org/10.1145/3140649.3140657

KEYWORDS

Secure Outsourcing, Secure Delegation, NTT, Lattices, Post-Quantum Cryptography, Ring Theory

1 INTRODUCTION

Public key cryptography, a cornerstone of modern digital security, relies heavily on the computational difficulty of certain mathematical problems, such as factoring large numbers and computing discrete logarithms, which form the basis of encryption algorithms from the RSA family (originated with [32]) and the ECC family (originated with [21, 25]). However, the advent of quantum computers poses a significant threat to public-key cryptography and the many real-life systems that use it as a security foundation. Quantum computers, with their ability to process complex calculations at unprecedented speeds, using algorithms like Shor's algorithm [34], could efficiently solve these problems, rendering current public key cryptographic systems insecure. As a result, there is a growing focus on developing quantum-resistant public-key encryption schemes, in the research area of post-quantum cryptography, to ensure data security in a future where quantum computing is prevalent. These new encryption schemes base their security on mathematical problems that are believed to be resistant to quantum attacks, and are mostly from these five types: hash-based, code-based, lattice-based, multivariable-based, and isogeny-based schemes. Among these, lattice-based cryptography seems to be very promising for its combined properties with respect to security, communication bandwidth, time efficiency, and allowed applications. In the post-quantum cryptography standardization competition held by the US National Institute of Standards and Technology (NIST), lattice-based cryptography was the area with the largest number of submissions [26–29], which included Crystals-Kyber [2, 8], the only public-key encryption scheme currently recommended for standardization, as a result of this competition. As we are planning to use Crystals-Kyber for a wide number of applications, its performance is being carefully studied. Multiplication between large-degree polynomials in a polynomial quotient ring, and its optimized execution based on the Number Theoretic Transform (NTT), is the dominating operation in all 3 algorithms of Crystals-Kyber. The need for fast NTT execution will be critical for public-key encryption uses within recent computing trends, including the Internet of Things, computing with resource-constrained devices and cloud/edge/fog computing. The hardware community is already producing research

on accelerators for NTT-based polynomial multiplication (see, e.g., [7] and referenes therein). In this paper, we study software-only methods of delegated computation of NTT operations.

Delegation. Server-aided cryptography (starting with, e.g., [1, 14, 24], and also known as outsourcing or delegation or verified computing) is an active research area addressing the problem of resource-constrained clients, such as IoT devices, delegating or outsourcing cryptographic computations to computationally more powerful (e.g., cloud-based) servers, while preserving input privacy and satisfying result correctness and security properties. Batch cryptography (starting with, e.g., [15]) is another active research area addressing the problem of performing multiple cryptographic operations better than independently repeating each operation, with respect to some performance metric.

More generally, we require a solution to the batch delegation of a function F to be a 2-phase, 2-party, protocol between client C and server S, where in the offline phase some information is computed and stored on C's device, and in the online phase C and S have a brief message exchange (typically, a message from C to S followed by one from S to C; see Figure 1), and where the following requirements are satisfied (see also Section 2 for more formal definitions):

1. δ_c-*result correctness*: if C and S honestly run the protocol, at the end of the protocol C returns $F(x_1), \ldots, F(x_n)$ with probability at least δ_c;
2. ϵ_p-*input privacy*: except for probability ϵ_p, no new information about the input batch $(x_1, \ldots, x_n)$ is revealed to S;
3. ϵ_s-*result security*: S should not be able, except possibly with probability ϵ_s, to convince C to return a result different than $(F(x_1), \ldots, F(x_n))$ at the end of the protocol; and
4. $(t_F, t_S, t_P, t_C, cc, sc, mc)$-*efficiency*:
   - *client online runtime efficiency*: C's online phase runtime, denoted as t_C, should be significantly smaller than the runtime, denoted as t_F, of computing $(F(x_1), \ldots, F(x_n))$ without delegation;
   - small S's runtime t_S (i.e., ideally, a small constant times t_F);
   - small online phase communication complexity cc (i.e., ideally, a small constant times the length of input and output);
   - small number of online phase messages mc (i.e., ideally, ≤ 2);
   - small storage complexity sc and offline runtime t_P.

We stress that the client online runtime efficiency property is critical for delegation to be effective.

Our contribution. We show several new client-server protocols for the delegated single and batch computation of the NTT and inverse NTT functions, in input scenario where the input polynomial(s) are private or public. All of our protocols satisfy result correctness, input privacy (unless the input is public), result security, and client online runtime efficiency and are the first to satisfy this set of properties. The properties of our main protocols are summarized in Table 1. Our delegation protocols achieve input privacy by a set of time-and-space-efficient input masking techniques and result security by a set of time-and-space-efficient probabilistic tests for computation verification.

We start by recalling the definition of polynomial rings and considering the NTT methods produced in the literature for polynomial multiplication in these rings. We abstract these methods into a single class of methods that satisfy 3 main properties: correctness (i.e., they successfully allow an alternative computation of polynomial multiplication), linearity (i.e., they produce a suitable homomorphism) and efficiency (i.e., NTT-based polynomial multiplication is faster than standard polynomial multiplication). Here, recall that standard polynomial multiplication [19] takes time $O(n^2)$, improved algorithms [9, 20, 36, 38] take time superlinear in n, and the entire method of NTT-based multiplication (including conversion) takes time $O(n \log n)$, where n is the polynomial degree. This class seemingly captures many known NTT methods in the literature (see, e.g., [22]).

Our delegation protocols work for the NTT and inverse NTT functions of any NTT methods in this class. Their client online runtime is $O(nr)$, for $r = \lceil \lambda / \log q \rceil$, which is lower than the $O(n \log n)$ runtime of the non-delegated computation of the NTT function, for a very wide range of parameters λ, q, n, including desirable parameter values both for asymptotic reasoning and current pratical use. For asymptotic comparison, we note that if we target the security probability to be negligible in n, then this requires $\lambda = \Omega((\log n)^{1+\epsilon})$, for some constant $\epsilon > 0$, and then the online client runtime is $o(n \log n)$ whenever $q = \omega((\log n)^{\epsilon})$. For comparison with respect to practical parameter values, we note that, in our protocols, client online runtime is strictly lower than non-delegated computation by about 1 order of magnitude in our software implementations of these protocols, as detailed in Tables 2-5, using parameter values that are consistent with the use of NTT operations in the Crystal-Kyber public key cryptosystem.

We discuss a few approaches of using our delegation protocols to improve the efficiency of the NTT operations in the Crystal-Kyber public key cryptosystem. Our software implementation of the most basic (and thus, least optimized) of these approaches does manage to improve the runtime of all three original algorithms in the cryptosystem by an average of about 13-17%, despite the original algorithms having gone through years of improvement and optimization studies.

Our protocols also perform well with respect to the other targeted efficiency properties, including client-server communication, client storage and server computation.

Comparison with previous work. We are not aware of any research on delegation of NTT in polynomial rings. There is much other work on single-server delegation of computations different than the NTT operations studied here, for which we refer to reader to these recent surveys: [13] for other specific operations used in cryptography protocols (e.g., integer ring multiplication, group exponentiation, pairings), [33] for other computations beyond cryptography, and [4] for computation of arbitrary functions, with clients more computationally powerful than considered here. There is also research in delegation in multi-server models where servers are assumed to be not colluding or communicating (see, e.g., [10]).

Organization of the paper. In Section 2 we recall the formal definition of protocols for delegated computation and review the algebraic setup for NTT computations, including our abstraction of NTT methods in the literature. In Section 3 we present our delegation protocols for NTT operations. In Section 4 we discuss our approaches for using our delegation protocols within the Crystals-Kyber public-key cryptosystem. In Section 5 we show our numeric

results on implementing our delegation protocols and our delegated version of the Crystals-Kyber public-key cryptosystem.

2 NOTATIONS AND DEFINITIONS

In this section we recall the formal definition, based on [16, 18] of protocols for delegated computation of a batch of function values, for arbitrary functions, as well as their result correctness, input privacy, result security, and efficiency requirements. We also review the algebraic setup for NTT computations, and discuss an abstraction of NTT methods based on a list of properties that are seemingly satisfied by all NTT methods in the literature (and which we will assume in our delegation protocols).

Basic notations. By $\vec{x}$ we denote the m-tuple $(x_1, \ldots, x_m)$. The expression $z \leftarrow T$ denotes randomly and independently choosing z from set T. By $y \leftarrow A(x_1, x_2, \ldots)$ we denote running the algorithm A on input $x_1, x_2, \ldots$ and any random coins, and returning y as output. By $(y, tr) \leftarrow (A(u_1, u_2, \ldots), B(v_1, v_2, \ldots))$ we denote running the interactive protocol between A, with input $u_1, u_2, \ldots$ and any random coins, and B, with input $v_1, v_2, \ldots$ and any random coins, where tr denotes A's and B's messages in this execution, and y is A's final output.

System scenario: entities and protocol. We consider a system with a single client, denoted by C, and a single server, denoted by S, who are connected by a reliable and authenticated channel, and therefore do not consider any integrity, replay or redirection attacks on this channel. As in previous work in the area, we consider a delegation protocol to include both an *offline phase*, typically consisting of client setup and deployment, and an *online phase*, typically starting when inputs to the delegated function are available to the client. While the protocol in the online phase is designed by taking into account the parties' different computation power, in particular reducing as much as possible the client's computation, this is not the case in the offline phase, which could be run, for instance, by a trusted deployer which at the end stores information on the client device.

Let σ denote the computational security parameter (derived from hardness considerations of the underlying computational problem), and let λ denote the statistical security parameter (defined so that statistical test failure events with probability $2^{-\lambda}$ are extremely rare). Both parameters are expressed in unary notation (i.e., $1^\sigma, 1^\lambda$), and we think of σ as being larger than λ. Let F denote a function and $desc(F)$ denote F's description in some conventional representation. Assuming $1^\sigma, 1^\lambda, desc(F)$ are known to both C and S, we define a *client-server protocol for the delegated (m-instance) computation of* F as the execution: $\{st \leftarrow \text{Offline}(1^\sigma, 1^\lambda, desc(F)); (\vec{y}, tr) \leftarrow (C(st, \vec{x}), S)\}$. Step 1 is executed in the offline phase, when the input to the function F that C wants to delegate is not available yet, and could be executed either by C or a third party not colluding with S. Step 2 is executed in the online phase, when inputs $\vec{x} = (x_1, \ldots, x_m)$ to the function F are available to C, and at the end of this phase, C learns $\vec{y} = (y_1, \ldots, y_m)$, where $y_i = F(x_i)$, for $i = 1, \ldots, m$, and tr is the transcript of the communication exchanged between C and S. We will often omit $desc(F), 1^\sigma, 1^\lambda, tr$ for brevity.

A system architecture and workflow can be found in Figure 1.

Result Correctness Requirement. Informally, this requirement states that if both parties follow the protocol, C obtains some output

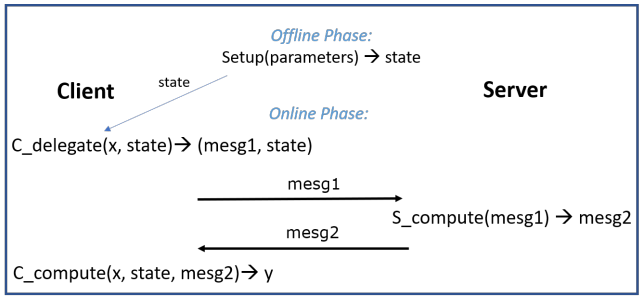


Figure 1: Delegated computation of $y = F(x)$

at the end of the protocol, and this output is, with high probability, equal to the m-tuple of values obtained by evaluating function F on C's m inputs. Formally, we say that a client-server protocol (C, S) for the delegated m-instance computation of F satisfies δ_c*-correctness* if for any $\vec{x} = (x_1, \ldots, x_m)$ in $Dom(F)$,

$$\text{Prob}\left[\, out \leftarrow \text{CorrExp}_{\text{F}} \,:\, out = 1 \,\right] \geq \delta_c,$$

for some δ_c close to 1, where experiment CorrExp is:

1. $st \leftarrow \text{Offline}(desc(F))$
2. $(\vec{y}, tr) \leftarrow (C(st, \vec{x}), S)$
3. if $y_i = F(x_i)$ for all $i = 1, \ldots, m$, then **return:** 1 else **return:** 0

Input Privacy Requirement. Informally, this requirement should guarantee the following: if C follows the protocol, a malicious adversary corrupting S cannot obtain any information about C's inputs $\vec{x}$ from a protocol execution. This is formalized by extending the indistinguishability-based approach typically used in definitions for encryption schemes. Let (C, S) be a client-server protocol for the delegated m-instance computation of F. We say that (C, S) satisfies ϵ_p*-privacy (in the sense of indistinguishability) against a malicious adversary* if for any algorithm A, it holds that

$$\text{Prob}\left[\, out \leftarrow \text{PrivExp}_{\text{F,A}} \,:\, out = 1 \,\right] \leq 1/2 + \epsilon_p,$$

for some ϵ_p close to 0, where experiment PrivExp is:

1. $st \leftarrow \text{Offline}(desc(F))$
2. $(\vec{x}_0, \vec{x}_1, aux) \leftarrow A(desc(F))$
3. $b \leftarrow \{0, 1\}$
4. $(\vec{y}, tr) \leftarrow (C(st, \vec{x}_b), A(aux))$
5. $d \leftarrow A(tr, aux)$
6. if $|x_{0i}| = |x_{1i}|$, for all $i = 1, \ldots, m$, and $b = d$ then **return:** 1 else **return:** 0.

Result Security Requirement. Informally, the security requirement states that for any efficient and malicious adversary corrupting S and even choosing C's input tuple $\vec{x}$, at the end of the protocol, C cannot be convinced to obtain some output tuple $\vec{z}$ containing a value $z_i \neq F(x_i)$ for some $i \in \{1, \ldots, m\}$. Formally, we say that the client-server protocol (C, S) for the delegated m-instance computation of F satisfies ϵ_s*-security against a malicious adversary* if for any algorithm A,

$$\text{Prob}\left[\, out \leftarrow \text{SecExp}_{\text{F,A}} \,:\, out = 1 \,\right] \leq \epsilon_s,$$

for some ϵ_s close to 0, where experiment SecExp is:

1. $st \leftarrow \text{Offline}(desc(F))$
2. $(\vec{x}, aux) \leftarrow A(desc(F))$

3. $(\vec{z}, tr) \leftarrow (C(st, \vec{x}), A(aux))$
4. if $z_i \in \{\perp, F(x_i)\}$ for all $i = 1, \ldots, m$, then **return:** 0
   else **return:** 1.

Efficiency Metrics and Requirements. Let (C, S) be a client-server protocol for the delegated computation of function F. We say that (C, S) has *efficiency parameters* $(t_F, t_P, t_C, t_S, cc, mc)$, if F can be computed (without outsourcing) using $t_F(\sigma, \lambda)$ atomic operations, C can be run in the offline phase using $t_P(\sigma, \lambda)$ atomic operations and in the online phase using $t_C(\sigma, \lambda)$ atomic operations, S can be run using $t_S(\sigma, \lambda)$ atomic operations, C and S exchange a total of at most mc messages, of total length at most cc. While we naturally try to minimize all these protocol efficiency metrics, our main goal is to design protocols where

(1) $t_C(\sigma, \lambda) < t_F(\sigma, \lambda)$, and
(2) $t_S(\sigma, \lambda)$ is not significantly larger than $t_F(\sigma, \lambda)$,

We sometimes use "*delegation improvement ratio* > 1" as a rephrased version of goal (1), where the delegation improvement ratio is defined as $t_F(\sigma, \lambda)/t_C(\sigma, \lambda)$. In our analysis, we only consider the most expensive ring operations as atomic operations (e.g., multiplications in ring $\mathbb{Z}_q$), and neglect lower-order operations (e.g., equality testing, additions and subtractions in ring $\mathbb{Z}_q$). This is based on the underlying assumption, consistent with the state of the art in cryptographic theory and practice implementations, where addition in ring $\mathbb{Z}_q$ requires significantly less computing resources than multiplication in ring $\mathbb{Z}_q$.

Function and Input Scenarios. In the rest of the paper we will present protocols for the m-instance delegated computation of function F, for $m = 1$ (i.e., the *single-computation* case) and for $m > 1$ (i.e., the *batch-computation* case).

We define 4 *input scenarios*, depending on certain features of the inputs $\vec{x}$ to function F: specifically, whether privacy of the inputs is desired or not, and whether the inputs are available in the offline phase or only known in the online phase. We say that $\vec{x}$ is

- *public online* if $\vec{x}$ is unknown in the offline phase but known by C and S in the online phase;
- *public offline* if $\vec{x}$ is known by both parties starting from the offline phase;
- *private online* if $\vec{x}$ is unknown in the offline phase but known by C in the online phase);
- *private offline* if $\vec{x}$ is known by C starting from the offline phase but unknown by S.

We note that not all input scenarios are interesting; for instance, the '$\vec{x}$ public offline' scenario is uninteresting since we assume that C is only resource-constrained in the online phase. Moreover, protocols for different scenarios can be related; for instance, a secure protocol for a scenario where an input is labeled as private is also a secure protocol for the otherwise identical scenario where that same input component is labeled as public; however it remains of interest to provide protocols for the public input scenario if these come with efficiency improvements. In this paper we will always consider online inputs, unless otherwise specified.

Algebraic setup for NTT methods. A *ring* is a set R accompanied with two 2-argument operations, commonly denoted as addition (+) and multiplication ($\cdot$), satisfying 3 axioms: (1) R is an abelian group under addition (i.e., + is associative and commutative, there exists an additive identity 0, and every $a \in R$ has an additive inverse $-a$); (2) R is a monoid under multiplication (i.e., $\cdot$ is associative and there exists a multiplicative identity 1); (3) multiplication is distributive with respect to addition.

Let n, q denote integers, and consider the following rings:

(1) $(\mathbb{Z}, +, \cdot)$, where $\mathbb{Z}$ denotes the set of integers; + denotes integer addition and $\cdot$ denotes integer multiplication.
(2) $(\mathbb{Z}_q, +_q, *_q)$, where $\mathbb{Z}_q = \{0, \ldots, q-1\}$, $+_q$ denotes addition modulo q and $*_q$ denotes multiplication modulo q.
(3) $(\mathbb{Z}[x], +, \cdot)$, where $\mathbb{Z}[x]$ is the set of polynomials with integer coefficients, + denotes polynomial addition and $\times$ denotes polynomial multiplication.
(4) $(\mathbb{Z}_q[x], +_q, *_q)$, where $\mathbb{Z}_q[x]$ is the set of polynomials with coefficients in $\mathbb{Z}_q$, $+_q$ denotes polynomial addition modulo q and $*_q$ denotes polynomial multiplication modulo q.

Let $\phi(x)$ denote an n-degree cyclotomic polynomial and let $R_q = \mathbb{Z}_q[x]/(\phi(x))$. In this paper we will mainly consider the *quotient ring* $(R_q, +_\phi, *_\phi)$, where R_q is the set of polynomials modulo $\phi(x)$ with coefficients in $\mathbb{Z}_q$, $+_\phi$ denotes addition of polynomials modulo $\phi(x)$ and $*_\phi$ denotes multiplication of polynomials modulo $\phi(x)$.

We will also use the subtraction operations in the above rings, which we denote as $-$, $-_q$ and $-_\phi$.

The literature contains several cryptosystems using such quotient rings, varying on how one sets integers q, n and polynomial $\phi(x)$; see, e.g. Table 3 in [22]. For instance, in Crystals-Kyber's round 1 NIST submission [8, 26], the authors set integer $n = 2^{e-1}$, polynomial $\phi(x) = x^n + 1$, which is thus the $2n$-th cyclotomic polynomial, and then integer $q = 1 \mod 2n$.

Typically, a computation of polynomial addition modulo $\phi(x)$ only requires n additions in $\mathbb{Z}_q$, but the computation of polynomial multiplication modulo $\phi(x)$ is significantly more expensive and is the most expensive operation in the cryptosystem. The textbook method [19] requires $O(n^2)$ multiplications in $\mathbb{Z}_q$, and even improved techniques such as Karatsuba's algorithm [20, 38] and Toom-Cook's algorithm [9, 36], do require super-linear time; specifically: $O(n^{1.58})$ and $O(n^c)$, for $c = \log_k(2k-1)$, for some integer $k > 1$. The typical method to implement multiplication in cryptography libraries is based on an NTT function, and, briefly speaking, consists of the following steps:

(1) transforming each of the two input polynomials into an NTT representation,
(2) performing the multiplication of these polynomials in NTT representation, and
(3) transforming the obtained polynomial back in the original representation.

Here, the multiplication in NTT representation (step 2) takes $O(n)$ multiplications in $\mathbb{Z}_q$ and the transformations between representations (steps 1 and 3) only require $O(n \log n)$ such multiplications, thus giving an overall quasi-linear complexity.

Different quotient rings of the above type may need different and specific NTT methods and related algorithms. The survey in [22] details such methods for many of the NIST submissions. All of these methods (to the best of our literature review and knowledge) satisfy certain basic properties, which we abstract and formally define here, as they will be used in the rest of the paper.

An *NTT method* for the quotient ring $(R_q, +_\phi, *_\phi)$ is a 4-tuple $(\text{NTT}, \text{NTT}^{-1}, +_t, \circ_t)$ of functions, defined as follows:

(1) NTT: $R_q \rightarrow (\mathbb{Z}_q)^n$
(2) $\text{NTT}^{-1} : (\mathbb{Z}_q)^n \rightarrow R_q$
(3) $+_t : (\mathbb{Z}_q)^n \times (\mathbb{Z}_q)^n \rightarrow (\mathbb{Z}_q)^n$
(4) $*_t : (\mathbb{Z}_q)^n \times (\mathbb{Z}_q)^n \rightarrow (\mathbb{Z}_q)^n$.

We remark that values in R_q are typically implemented as values in $\mathbb{Z}_q^n$, and thus in the above definition, the distinction between R_q and $\mathbb{Z}_q^n$ serves purely as a semantics reminder.

Given an NTT method for the quotient ring $(R_q, +_\phi, *_\phi)$, and given function $F \in \{\text{NTT}, \text{NTT}^{-1}\}$, we say that an NTT method satisfies $(F, *_\phi, *_t)$-*correctness* if for any two polynomials $p_1[x], p_2[x] \in R_q$, it holds that

$$p_1[x] *_\phi p_2[x] = F^{-1}(F(p_1[x]) *_t F(p_2[x]))$$

Given an NTT method for the quotient ring $(R_q, +_\phi, *_\phi)$, and given function $F \in \{\text{NTT}, \text{NTT}^{-1}\}$, we say that the NTT method satisfies *F-linearity* if for any two polynomials $p_1[x], p_2[x] \in R_q$, it holds that

$$F(p_1[x] +_\phi p_2[x]) = F(p_1[x]) +_t F(p_2[x]).$$

We say that an NTT method for the quotient ring $(R_q, +_\phi, *_\phi)$ satisfies *efficiency* if the operations $+_t$ and $*_t$ can be computed using $O(n)$ multiplications in $\mathbb{Z}_q$.

In the rest of the paper we will consider NTT methods that satisfy some or all of these properties and consider delegation protocols for the (single and many-input) NTT and NTT^{-1} functions in them. We observe that many NTT methods in the literature (e.g., all those in Table 3 of [22], and we were unable to find one that does not) satisfy the following properties:

(1) $(F, *_\phi, *_t)$-correctness, for both $F \in \{\text{NTT}, \text{NTT}^{-1}\}$
(2) F-linearity, for both $F \in \{\text{NTT}, \text{NTT}^{-1}\}$
(3) efficiency.

For instance, in Crystals-Kyber's round 3 NIST submission [2], the authors often discuss the efficiency property, the $(\text{NTT}, *_\phi, *_t)$-correctness property, and the bijective property of NTT, which implies the existence of NTT^{-1} and the $(\text{NTT}^{-1}, *_\phi, *_t)$-correctness property. As for linearity, the NTT/NTT^{-1}-linearity properties follow from [2], equations (4) and (5), page 6.

3 DELEGATING NTT COMPUTATIONS

In this section we investigate client-server protocols for the delegated single/batch computation of the NTT and NTT^{-1} functions, as defined in Section 2, in input scenario where the input polynomial(s) are private or public. We show several delegation protocols, each applicable to a different (function, input scenario) combination, where the client only performs a smaller number of multiplications in $\mathbb{Z}_q$ with respect to non-delegated NTT or NTT^{-1} computations. We summarize our main results with the following

Theorem 3.1. Let n, q denote integers, let $(R_q, +_\phi, *_\phi)$ denote the quotient ring and consider an NTT method for this ring, as defined in Section 2. Also, let σ denote the computational security parameter, and let λ be a statistical security parameter. There exists (constructively) single-server protocols $\mathcal{P}_j$, for $j \in \{1.1, 1.2, 1.3, 2.1, 3.1\}$, for the delegated computation of function F_j, in input scenario is_j, and in an NTT method with properties np_j, satisfying:

(1) δ_c-result correctness, ϵ_p-input privacy, ϵ_s-result security for values of these parameters as specified in Table 1 (top); and
(2) $(t_F, t_S, t_P, t_C, cc, sc, mc)$-efficiency, for values of these parameters as specified in Table 1 (bottom).

The main takeaway from this theorem is that C can delegate a single or batch computation of the NTT and NTT^{-1} functions, on private or public inputs, to S using a protocol that satisfies result correctness, input privacy (when required), result security, and various efficiency properties; most notably, in delegation of a single computation of the NTT function on a private input, the online client runtime is $O(nr)$, for $r = \lceil \lambda / \log q \rceil$, which is lower than the $O(n \log n)$ runtime of the non-delegated computation of the NTT function, for a very wide range of parameters λ, q, n, including desirable parameter values both for asymptotic reasoning and current pratical use. For asymptotic comparison, we note that if we target the security probability to be negligible in n, then this requires $\lambda = \Omega((\log n)^{1+\epsilon})$, for some constant $\epsilon > 0$, and then the online client runtime is $o(n \log n)$ whenever $q = \omega((\log n)^\epsilon)$. For comparison with respect to practical parameter values, we note that, in our protocols, client online runtime is strictly lower than non-delegated computation by about 1 order of magnitude in our software implementations of these protocols, as detailed in Section 5.

In the rest of the section, we describe the delegation protocols claimed in Theorem 3.1 (as well as various extensions of them), along with proofs of their properties.

3.1 NTT on a Private Input - Protocol 1.1

Informal Description of Protocol $\mathcal{P}_{1.1}$: Our two main challenges in designing a protocol for the delegated computation of the NTT function are to ensure that the protocol simultaneously satisfies the input privacy and result security properties. In both our protocols $\mathcal{P}_{1,1}$ and $\mathcal{P}_{1,2}$, first C sends to S a masked version $z_0 = x -_\phi u_0$ of polynomial x, using a random polynomial $u_0 \in R_q$; then, S sends receiving $w_0 = \text{NTT}(z_0)$ to C, and finally C computes $y = w_0 +_t v_0$, where $v_0 = \text{NTT}(u_0)$ was computed in the offline phase and stored on C's device. If C and S follow the protocol, the linearity of the NTT function implies that C returns $y = \text{NTT}(x)$, and masking implies that S learns nothing about x. We address our next challenge (i.e., achieving result security) by designing checks for C to verify that value w_0 returned by S was correctly computed. C efficiently achieves to this by asking S for an additional NTT function computation on an auxiliary (and masked as well) input z_1 and by (1) a deterministic membership test that verifies that values w_0, w_1 do belong to the co-domain $(\mathbb{Z}_q)^n$ of the NTT function; and (2) a new probabilistic test that efficiently verifies a probabilistic condition relating the NTT output values obtained by S on the masked original inputs and the masked auxiliary NTT inputs, based on the linearity property of the NTT function:

$$w_1 = b *_t w_0 +_t v,$$

where $w_0 = \text{NTT}(z_0)$, $w_1 = \text{NTT}(z_1)$, $z_1 = b *_\phi z_0 +_\phi u$, and $b \in \mathbb{Z}_q$. As it turns that this test only detects a cheating S with probability $\geq 1 - 1/q$, we consider the problem of increasing this

Table 1: (Top:) A first set of properties of our protocols: delegated function F_j, input scenario is_j, assumed NTT method properties np_j, result correctness parameter δ_c, input privacy parameter ϵ_p and result security parameter ϵ_s. (Bottom): Efficiency properties of our protocols: number of messages mc, C's storage sc at the end of the offline phase (in # of values from $\mathbb{Z}_q$), communication complexity cc (in # of values from $\mathbb{Z}_q$), C's online runtime t_C, S's runtime t_S, and offline runtime t_P (all in terms of multiplications $*_q$ in $\mathbb{Z}_q$). Here, $r = \lceil \lambda/\log q \rceil$ and t_F denotes the time to run a single NTT or NTT^{-1} computation.

Protocol	Function F_j	Input scenario is_j	Assumed NTT method properties np_j	δ_c	ϵ_p	ϵ_s
$\mathcal{P}_{1,1}$ [§3.1]	NTT	x private	NTT-linear, efficient	1	0	$2^{-\lambda}$
$\mathcal{P}_{1,2}$ [§3.2]	NTT	x private	$(*_\phi, *_t)$-correct, NTT-linear, efficient	1	0	$2^{-\lambda}$
$\mathcal{P}_{1,3}$ [§3.3]	NTT	x public	NTT-linear, efficient	1	n.a.	$2^{-\lambda}$
$\mathcal{P}_{2,1}$ [§3.4]	NTT^{-1}	x private	NTT^{-1}-linear, efficient	1	0	$2^{-\lambda}$
$\mathcal{P}_{3,1}$ [§3.5]	batch NTT	$x_1, \ldots, x_m$ private	NTT-linear, efficient	1	0	$2^{-\lambda}$

Protocol	mc	sc	cc	t_C	t_S	t_P
$\mathcal{P}_{1,1}$ [§3.1]	2	$O(nr)$	$O(nr)$	$O(nr)$	$O(rt_F)$	$O(rt_F)$
$\mathcal{P}_{1,2}$ [§3.2]	2	$O(n)$	$O(n)$	$O(nr)$	$O(t_F)$	$O(t_F)$
$\mathcal{P}_{1,3}$ [§3.3]	2	$O(nr)$	$O(nr)$	$O(nr)$	$O(rt_F)$	$O(rt_F)$
$\mathcal{P}_{2,1}$ [§3.4]	2	$O(nr)$	$O(nr)$	$O(nr)$	$O(rt_F)$	$O(rt_F)$
$\mathcal{P}_{3,1}$ [§3.5]	2	$O(nr)$	$O(n(r+m))$	$O(nrm)$	$O((r+m)t_F)$	$O((r+m)t_F)$

probability to $1 - 2^{-\lambda}$, so to satisfy the result security requirement. One immediate approach towards this goal would be to perform $r = \lceil \lambda/\log q \rceil$ parallel and independent repetitions of the above protocol. As a further optimization, we observe that it only suffices to repeat the additional z_1 value and reuse the same z_0 value. A formal description follows.

Formal Description of Protocol $\mathcal{P}_{1.1}$ (Private Input):

Input for C and S: $1^\sigma, 1^\lambda, desc(R_q)$, where $R_q = \mathbb{Z}_q[x]/(x^n + 1)$

Offline phase instructions:

1. Set $r = \lceil \lambda/\log q \rceil$
2. for $i = 0, 1, \ldots, r$
   uniformly choose $u_i \in R_q$ and set $v_i := \text{NTT}(u_i)$
3. Store $state := \{(u_0, v_0), (u_1, v_1), \ldots, (u_r, v_r)\}$ on C

Online Input to C: $x \in R_q$, $state$

Online phase instructions:

1. C sets $z_0 := x -_\phi u_0$
   for $i = 1, \ldots, r$,
   C uniformly chooses scalar $b_i \in \mathbb{Z}_q$
   C computes $z_i := b_i *_\phi z_0 +_\phi u_i$
   C sends $z_0, z_1, \ldots, z_r$ to S
2. S computes $w_i = \text{NTT}(z_i)$ for $i = 0, 1, \ldots, r$
   S sends $w_0, w_1, \ldots, w_r$ to C
3. C checks that
   (Membership Check): $w_0 \in (\mathbb{Z}_q)^n$
   (Probabilistic Check): $w_i = b_i *_t w_0 +_t v_i$, for $i = 1, \ldots, r$
   if any one of the above checks is not satisfied then
   C **returns:** $\perp$ and the protocol halts.
4. C **returns:** $y := w_0 +_t v_0$

Properties of protocol $\mathcal{P}_{1,1}$: *The efficiency properties* are verified by protocol inspection. In particular:

- *Storage Complexity:* at the end of the offline phase $r + 1$ polynomials in R_q and $r + 1$ tuples in $(\mathbb{Z}_q)^n$ are stored on C's device.
- *Round complexity:* The online phase of the protocol only requires two messages: one from C to S, followed by one from S to C.
- *Communication complexity:* During the online phase, C sends $r + 1$ polynomials in R_q, and S sends $r + 1$ tuples in $(\mathbb{Z}_q)^n$.
- *Runtime complexity:* During the online phase, C performs r $*_\phi$ operations between a scalar b_i and a polynomial z_0, r $*_t$ operations and at most $r + 1$ $+_t$ operations, all three of which requiring at most $O(nr)$ multiplications $*_q$ in $\mathbb{Z}_q$, as well as a number of lower-order operations (i.e., $r + 1$ additions or subtractions in R_q, and n checks for membership in $\mathbb{Z}_q$). S only performs $r + 1$ computation of the NTT function. The offline phase only requires $r + 1$ computation of the NTT function. See also Table 1 (bottom).

The *result correctness* property follows by showing that if C and S follow the protocol, both the membership and probabilistic checks are satisfied, and C always outputs $y = \text{NTT}(x)$.

The satisfaction of the membership check directly follows by the NTT function's definition.

The satisfaction of the probabilistic check follows by observing that for $i = 1, \ldots, r$,

$$\begin{aligned} w_i &= \text{NTT}(z_i) \\ &= \text{NTT}(b_i *_\phi z_0 +_\phi u_i) \\ &= \text{NTT}(b_i *_\phi z_0) +_t \text{NTT}(u_i) \\ &= b_i *_t \text{NTT}(z_0) +_t \text{NTT}(u_i) \\ &= b_i *_t w_0 +_t v_i, \end{aligned}$$

where the 1st equality follows by how S computes w_i, the 2nd equality follows by how C computes z_i, the 3rd and 4th equalities follows from different applications of the NTT-linearity property, and the last equality follows by how S computes w_0 and C computes v_i.

This implies that C never returns $\perp$, and thus always returns

$$y = w_0 +_t v_0 = \mathrm{NTT}(z_0) +_t \mathrm{NTT}(u_0) = \mathrm{NTT}(x -_\phi u_0) +_t \mathrm{NTT}(u_0) = \mathrm{NTT}(x),$$

where the 2nd equality follows by how S computes w_0 and how v_0 was generated in the offline phase, the 3rd equality follows by how C computes z_0 and the 4th equality follows from the NTT-linearity property.

The *input privacy* property of the protocol against any malicious S follows by observing that, C's message $(z_0, z_1, \ldots, z_r)$ to S does not leak any information about C's input x. This follows because values $u_0, u_1, \ldots, u_r$ are uniformly and independently chosen from R_q. Moreover, by similar reasoning, this message does not leak any information about b_i, a fact useful in the proof of the security property.

To prove the *result security* property against any malicious S, we need to compute an upper bound ϵ_s on the security probability that S convinces C to output a y such that $y \neq \mathrm{NTT}(x)$. We obtain that $\epsilon_s \leq 2^{-\lambda}$, as a consequence of the following three facts, which we later prove:

1. tuple $(z_0, z_1, \ldots, z_r)$ leaks no information about b to S, where $z_0 = x -_\phi u_0$ and $z_i = b_i *_\phi z_0 +_\phi u_i$, where $x, u_0, u_i \in R_q$ and $b_i \in \mathbb{Z}_q$, for $i = 1, \ldots, q$;
2. for any S's message $w = (w_0, w_1, \ldots, w_r)$ such that at least one of $w_0, w_1, \ldots, w_r$ is different than what would be returned according to the protocol instructions, the number of tuples $(b_1, \ldots, b_r) \in (\mathbb{Z}_q)^r$ for which $(w_0, w_1, \ldots, w_r)$ satisfies both the membership and the probabilistic test in the protocol is ≤ 1;
3. for any S's message $w = (w_0, w_1, \ldots, w_r)$ such that at least one of $w_0, w_1, \ldots, w_r$ is different than what would be returned according to the protocol instructions, the probability that w satisfies the probabilistic test is $\leq 2^{-\lambda}$.

To prove Fact 1, we observe that all z_i values are uniformly and independently distributed in R_q since so are the u_i values, for $i = 0, 1, \ldots, r$. We also observe that, for the same reason, the distribution of $(z_0, z_1, \ldots, z_r)$ is independent from that of b, a fact that will be useful in the proof of the result security property.

Towards proving Fact 2, let $w = (w_0, w_1, \ldots, w_r)$ be the tuple that would be returned by S according to the protocol, and assume that a malicious algorithm Adv, corrupting S, returns a different tuple $w' = (w'_0, w'_1, \ldots, w'_r)$. Note that if $w'_i \notin (\mathbb{Z}_q)^n$ for some $i \in \{0, 1, \ldots, r-1\}$, the tuple w' does not satisfy the membership test. Thus, for all $i = 0, 1, \ldots, r$. we assume that $w'_i \in (\mathbb{Z}_q)^n$ and we can then write w'_i as $(w'_{i,0}, \ldots, w'_{i,n-1})$, where $w'_{i,j} \in \mathbb{Z}_q$, for all $j = 0, \ldots, n-1$. Because $\mathbb{Z}_q$ is also a group with respect to addition $+_q$, we can write, for $i \in [0, r-1]$ and $j \in [0, n-1]$,

$$w'_{i,j} = d_{i,j} +_q w_{i,j},$$

for some $d_{i,j} \in \mathbb{Z}_q$ such that not all of the $d_{i,j}$ are equal to 1.

Now, consider the i-th equation in the probabilistic test, for any $i \in [1, r-1]$:

$$w'_i = b_i *_\phi w'_0 +_\phi v_i,$$

and note that each of these equations can be rewritten as n equations, where the (i, j)-th equation, for any $j \in [0, n-1]$, can be written as

$$w'_{i,j} = b_i *_q w'_{0,j} +_q v_{i,j}.$$

Then, consider the following equivalent rewritings of the (i, j)-th equation in the probabilistic test, obtained by variable substitutions and simplifications:

$$\begin{aligned} w'_{i,j} &= b_i *_q w'_{0,j} +_q v_{i,j} \\ d_{i,j} +_q w_{i,j} &= b_i *_q (d_{0,j} +_q w_{0,j}) +_q v_{i,j} \\ d_{i,j} +_q (b_i *_q w_{0,j} +_q v_{i,j}) &= b_i *_q d_{0,j} +_q b_i *_q w_{0,j} +_q v_{i,j} \\ d_{i,j} +_q v_{i,j} &= b_i *_q d_{0,j} +_q v_{i,j} \\ d_{i,j} &= b_i *_q d_{0,j} \end{aligned}$$

where the third equality follows from the protocol's result correctness property. Now, if for any pair $(d_{0,j}, d_{i,j})$, there exist at least two distinct $b_{i,0}$ and $b_{i,1}$ in $\mathbb{Z}_q$ satisfying this latter equation, we have that

$$d_{i,j} = b_{i,0} *_q d_{0,j}, \text{and}$$
$$d_{i,j} = b_{i,1} *_q d_{0,j}.$$

By applying the $-_q$ operation, we have that

$$0 = (b_{i,0} -_q b_{i,1}) *_q d_{0,j},$$

which implies that at least one of the following two conditions holds:

(1) $b_{i1} = b_{i0}$, for all $i = 0, 1, \ldots, r$,
(2) $d_{0,j} = 0$, for all $j = 0, 1, \ldots, n-1$.

Note that if (1) holds, the fact follows. On the other hand, if (2) holds, we also have that $d_{i,j} = 0$, which implies that $w'_{i,j} = w_{i,j}$, implying that S was honest. This proved Fact 2.

Towards proving Fact 3, first recall Fact 1 saying that C's message $(z_0, z_1, \ldots, z_r)$ does not leak any information about values $b_1, \ldots, b_r \in (\mathbb{Z}_q)^r$, and thus all values in $(\mathbb{Z}_q)^r$ are still equally likely for $(b_1, \ldots, b_r)$ even when conditioning over C's message. Then we can compute an upper bound on the probability that S's message w satisfies the i-th equation of the probabilistic test, as the ratio between the number of b_i values for which w satisfies this equation, and the number of values of b_i that are still equally likely even when conditioning over C's message. In this ratio, the numerator is at most 1 (by Fact 2), and the denominator is $|\mathbb{Z}_q| = q$ (by Fact 1). Since there are $r = \lceil \lambda / \log q \rceil$ such equations in the probabilistic test, and all of these equation use random and independent b_i values, the probability that w satisfies all r equations in the probabilistic test is

$$\leq \left(\frac{1}{q}\right)^r = \left(\frac{1}{q}\right)^{\lceil \lambda / \log q \rceil} \leq \frac{1}{2^\lambda}.$$

This proves Fact 3.

3.2 NTT on a Private Input - Protocol 1.2

Informal Description of Protocol $\mathcal{P}_{1.2}$: The design approach for this protocol starts very similarly as for protocol $\mathcal{P}_{1.1}$, but then changes in generating a slightly different probabilistic test. Specifically, the new test is based on both the linearity and the correct computation properties of the NTT functions, and is defined as follows:

$$w_1 = c *_t w_0 +_t v,$$

where $w_0 = \text{NTT}(z_0)$, $w_1 = \text{NTT}(z_1)$, $c = \text{NTT}(b)$, $z_1 = b *_\phi z_0 +_\phi u$, and b is a polynomial in R_q of degree at most $r-1$. Here, it turns out that this test detects a cheating S with probability $1 - 2^{-\lambda}$, and so the protocol satisfies the result security requirement, without need for parallel repetitions. Overall, when compared with protocol $\mathcal{P}_{1.1}$, protocol $\mathcal{P}_{1.2}$ uses a larger set of assumptions on the used NTT method, but has lower communication complexity and lower online client runtime for the smallest range values for λ. A formal description follows.

Formal Description of Protocol $\mathcal{P}_{1.2}$ (Private Input):

Input for C and S: $1^\sigma, 1^\lambda, desc(R_q)$, where $R_q = \mathbb{Z}_q[x]/(x^n+1)$

Offline phase instructions:

1. Set $r = \lceil \lambda / \log q \rceil$
2. for $i = 0, 1$:
   uniformly choose $u_i \in R_q$ and set $v_i := \text{NTT}(u_i)$
3. uniformly choose $(r-1)$-degree polynomial $b \in \mathbb{Z}_q[x]/\phi(x)$ and set $c = \text{NTT}(b)$
4. Store $state := \{1^r, (u_0, v_0), (u_1, v_1), (b, c)\}$ on C

Online Input to C: $x \in R_q$, $state$

Online phase instructions:

1. C uniformly chooses $(r-1)$-degree polynomial $b \in \mathbb{Z}_q[x]/\phi(x)$
   C computes $z_0 = x -_\phi u_0$ and $z_1 := b *_\phi z_0 +_\phi u_1$
   C sends z_0, z_1 to S
2. S computes $w_i = \text{NTT}(z_i)$ for $i = 0, 1$
   S sends w_0, w_1 to C
3. C checks that
   (Membership Check): $w_0 \in (\mathbb{Z}_q)^n$
   (Probabilistic Check): $w_1 = c *_t w_0 +_t v_1$
   if any one of the above checks is not satisfied then
   C **returns:** $\perp$ and the protocol halts.
4. C **returns:** $y := w_0 +_t v_0$

Properties of protocol $\mathcal{P}_{1,2}$: The efficiency, result correctness, input privacy and result security properties of $\mathcal{P}_{1,2}$ are similar and proved similarly as for $\mathcal{P}_{1,1}$. In what follows, we only discuss any differences in either the properties or their proofs.

The efficiency properties of $\mathcal{P}_{1,2}$ are verified by protocol inspection as for $\mathcal{P}_{1,1}$. In particular, we note the main difference: both the storage complexity and the communication complexity are a constant number (≤ 6) of ring elements, while there were $2(r+1)$ ring elements in $\mathcal{P}_{1,1}$.

The result correctness is similarly proved, the main difference being that it uses an application of the NTT-linearity property and an application of the computation correctness property (instead of 2 applications of the NTT-linearity property as in $\mathcal{P}_{1,1}$).

Details of the proofs are included for completeness in Appendix A.

3.3 NTT on a Public Input - Protocols 1.3, 1.4

In the 'x public' input scenario we can define protocol $\mathcal{P}_{1,3}$, by slight modifications to protocol $\mathcal{P}_{1,1}$; i.e., by removing the original masking to input x and a number of related computations. Specifically, we define protocol $\mathcal{P}_{2,1}$ by applying the following modifications to protocol $\mathcal{P}_{1,1}$:

(1) The offline phase does not generate u_0 or compute v_0

(2) C sets $z_0 = x$ and does not need to send it to S (since x is public and known to S); note that
   S's computation of w_0 satisfies $w_0 = \text{NTT}(z_0) = \text{NTT}(x)$
   C's computation of y only consists of setting $y = w_0$.

It is easy to verify that protocol $\mathcal{P}_{1,3}$ essentially preserves the properties of protocol $\mathcal{P}_{1,1}$: result correctness and result security, as well as efficiency, with a slight improvement on all efficiency metrics. We also observe that using the same modifications, one can similarly obtain a protocol $\mathcal{P}_{1,4}$ for the 'x public' input scenario, starting from protocol $\mathcal{P}_{1,2}$, which essentially preserves the properties of protocol $\mathcal{P}_{1,2}$, again with a slight improvement on all efficiency metrics.

3.4 Protocols 2.1 - 2.4 for Inverse NTT

We can solve the problem of delegating function NTT^{-1} using a direct adaptation of the same protocols as those described in Sections 3.2 and 3.3 to delegate the NTT function.

Specifically, for $i = 1, \ldots, 4$, we can define protocol $\mathcal{P}_{2,i}$ starting from protocol $\mathcal{P}_{1,i}$, and replacing the computations of function NTT in the offline phase and by S in the online phase, with analogue computations of function NTT^{-1}.

The proofs of the result correctness, privacy and result security properties for $\mathcal{P}_{1,i}$ continue to hold for $\mathcal{P}_{2,i}$ when we substitute the use of assumptions of $(\text{NTT}, *_\phi, *_t)$-correctness and NTT-linearity with assumptions of $(\text{NTT}^{-1}, *_\phi, *_t)$-correctness and NTT^{-1}-linearity, respectively.

3.5 Protocols 3.1 - 3.8 for Batch NTT

Informal Description of Protocol $\mathcal{P}_{3.1}$: We consider delegated computation over a batch of m private inputs $x_1, \ldots, x_m$. The main challenge in designing a protocol for delegating a batch of NTT computations is to define a protocol with improved properties (especially, efficiency properties) with respect to a direct independent repetition of the protocols from Sections 3.2, 3.3, and 3.4, for each input in the batch. In our protocol for the scenario of private inputs, C again uses masking of the original inputs in the batch, but then uses a single set of r masked auxiliary inputs, instead of such a set for each input in the batch. Since the application of the masking technique essentially repeats the analogue technique of protocol $\mathcal{P}_{1,1}$ by computing a masked value $z_{0,j}$ for each original input x_j in the batch, we analogously have that if C and S follow the protocol, the linearity of the NTT function implies that C returns $y_j = \text{NTT}(x_j)$, for $j = 0, \ldots, m-1$, and masking implies that S learns nothing about $x_0, \ldots, x_{m-1}$.

We then achieve result security by designing a single set of r checks for C to verify that the m NTT output values returned by S were correctly computed. C efficiently achieves to this by asking S for additional NTT function computations on auxiliary (and masked as well to preserve input privacy) values z_{1i} and by (1) a deterministic membership test that verifies that values $w_{0,0}, \ldots, w_{0,m-1}$ do belong to the co-domain $(\mathbb{Z}_q)^n$ of the NTT function; and (2) a new probabilistic test that efficiently verifies a probabilistic condition relating the NTT output values obtained by S on the masked original inputs and the masked auxiliary NTT inputs, based on the linearity

property of the NTT function:

$$w_{1,i} = (\sum_{t;j=0}^{m-1} b_{i,j} *_t w_{0,j}) +_t v_i, \text{ for } i = 0, \ldots, r-1,$$

where, $w_{0,j} = \text{NTT}(z_{0,j})$, $w_{1,i} = \text{NTT}(z_{1,i})$, $z_{1,i} = (\sum_{\phi;j=0}^{m-1} b_{i,j} *_\phi z_{0,j}) +_\phi \alpha_i$, and $b_{i,j} \in \mathbb{Z}_q$, for $j = 0, \ldots, m-1$ and $i = 0, \ldots, r-1$.

A formal description follows.

Formal Description of Protocol $\mathcal{P}_{3,1}$ (Private Inputs):

Input for C and S: $1^\sigma, 1^\lambda, 1^m, desc(R_q)$, where $R_q = \mathbb{Z}_q[x]/(x^n+1)$

Offline phase instructions:

1. Set $r = \lceil \lambda / \log q \rceil$ and $state = \emptyset$
2. for $j = 0, 1, \ldots, m-1$
   uniformly choose $u_j \in R_q$ and set $v_j := \text{NTT}(u_j)$
   set $state = state \cup \{(u_0, v_0), (u_1, v_1), \ldots, (u_m, v_m)\}$
3. for $i = 0, 1, \ldots, r-1$
   uniformly choose $\alpha_i \in R_q$ and set $\beta_i := \text{NTT}(\alpha_i)$
   set $state = state \cup \{(\alpha_0, \beta_0), (\alpha_1, \beta_1), \ldots, (\alpha_r, \beta_r)\}$
4. Store $state$ on C

Input to C: $x \in R_q$, $state$

Online phase instructions:

1. C uniformly chooses
   scalar $b_{i,j} \in \mathbb{Z}_q$, for $j = 0, \ldots, m-1$ and $i = 0, \ldots, r-1$
   C sets $z_{0,j} = x -_\phi u_{0,j}$, for $j = 0, \ldots, m-1$
2. C sets $z_{1,i} = (\sum_{\phi;j=0}^{m-1} b_{i,j} *_\phi z_{0,j}) +_\phi \alpha_i$, for $i = 0, \ldots, r-1$
   C sends $z_{0,0}, \ldots, z_{0,m-1}, z_{1,0}, \ldots, z_{1,r-1}$ to S
3. S computes $w_{0,j} = \text{NTT}(z_{0,j})$, for $j = 0, 1, \ldots, m-1$
   S computes $w_{1,i} = \text{NTT}(z_{1,i})$, for $i = 0, 1, \ldots, r-1$
   S sends $w_{0,0}, \ldots, w_{0,m-1}, w_{1,0}, \ldots, w_{1,r-1}$ to C
4. C checks that
   (Membership Check): $w_{0,0}, \ldots, w_{0,m-1} \in (\mathbb{Z}_q)^n$
   (Probabilistic Check): $w_{1,i} = (\sum_{t;j=0}^{m-1} b_{i,j} *_t w_{0,j}) +_t v_i$
   for $i = 0, \ldots, r-1$
   if any one of the above checks is not satisfied then
   C **returns:** $\perp$ and the protocol halts.
5. C sets $y_j = w_{0,j} +_t v_j$, for $j = 0, \ldots, m-1$
   C **returns:** $(y_0, \ldots, y_{m-1})$

Properties of protocol $\mathcal{P}_{3,1}$: The efficiency, result correctness, input privacy and result security properties of $\mathcal{P}_{3,1}$ are similar to the analogue properties for $\mathcal{P}_{1,1}$ and proved by direct extension of the proofs shown in Section 3.2.

Here, we only note an important achieved property: both the storage complexity and the communication complexity in $\mathcal{P}_{3,1}$ consist of $O(r+m)$ elements in R_q, instead of $O(rm)$ elements that would be needed by a direct repetition of protocol $\mathcal{P}_{1,1}$.

Formal proofs for efficiency, result correctness, input privacy and result security properties of protocol $\mathcal{P}_{3,1}$ are included for completeness in Appendix B.

Alternative method for Batch NTT delegation, Private Inputs. We note that the techniques behind protocol $\mathcal{P}_{1,2}$ can be directly combined with those in protocol $\mathcal{P}_{3,1}$ to produce a protocol $\mathcal{P}_{3,2}$ which can be seen as an alternative protocol for the delegation of a batch of m NTT computation over private inputs.

Protocol $\mathcal{P}_{3,2}$ gains the interesting properties of $\mathcal{P}_{1,2}$; in particular, the communication complexity consists of $O(m)$ elements in R_q instead of $O(m+r)$, as in $\mathcal{P}_{3,1}$. Analogously, protocol $\mathcal{P}_{3,2}$ makes the extra assumption of $(\text{NTT}, *_\phi, *_t)$-correctness for the considered NTT method.

Batch NTT on Public Inputs. We note that the techniques behind protocol $\mathcal{P}_{1,3}$ can be directly combined with those in protocol $\mathcal{P}_{3,1}$ (resp., $\mathcal{P}_{3,2}$) to produce a protocol $\mathcal{P}_{3,3}$ (resp., $\mathcal{P}_{3,4}$) which achieves delegation of a batch of m NTT computation over *public* inputs.

Protocols $\mathcal{P}_{3,3}$ and $\mathcal{P}_{3,4}$ gain the interesting properties of $\mathcal{P}_{1,3}$. Recall that this latter protocol did not require any masking of the input to the protocol, and some slight efficiency gains were obtained as this masked input was neither sent by C nor processed by S. When we apply this observation to batch delegation, we obtain that none of the inputs in the batch require any masking, and this is somewhat more interesting in that we can eliminate $O(m)$ ring elements from the communication from C to S and from C's storage. This results in $O(r)$ storage and communication for protocol $\mathcal{P}_{3,3}$ and $O(1)$ storage and communication for $\mathcal{P}_{3,4}$.

Batch Inverse NTT. We note that the techniques behind protocols $\mathcal{P}_{3,1}, \ldots, \mathcal{P}_{3,4}$ can be directly combined with those discussed in Section 3.4 to produce protocols $\mathcal{P}_{3,5}, \ldots, \mathcal{P}_{3,8}$ for batch delegation of NTT^{-1}, with properties essentially mirroring the properties in protocols $\mathcal{P}_{3,1}, \ldots, \mathcal{P}_{3,4}$. In particular, protocol $\mathcal{P}_{3,5}$ achieves batch delegation of NTT^{-1} on private inputs. Recall that, as discussed in Section 3.4, the computations of function NTT in the offline phase and by S in the online phase are replaced by analogue computations of function NTT^{-1}, and thus the proofs of the resulting protocols need the assumptions of $(\text{NTT}^{-1}, *_\phi, *_t)$-correctness and NTT^{-1}-linearity.

4 NTT DELEGATION IN CRYSTALS-KYBER

In this section we show applications of our NTT delegation results to the Crystals-Kyber public-key cryptosystem [2, 8]. First, we review this cryptosystem in Section 4.1 and then we show in Section 4.2 some different approaches to perform delegation of NTT operations in it, using results from Section 3.

4.1 The Crystals-Kyber cryptosystem

We review the NIST round 3 submission version of this cryptosystem, as from [2], in Algorithms 1-3 below, and point out the algorithm steps where NTT and NTT^{-1} functions are executed. We focus on the IND-CPA-secure version for simplicity, since the transformation from an IND-CPA-secure to an IND-CCA-secure cryptosystem in [2] does not further use any NTT operation. We use the same notation as in [2]; in addition, for $F \in \{\text{NTT}, \text{NTT}^{-1}\}$, we use the notation 'cwise-F' to denote the independent application of the function F to each (polynomial) component of the input vector.

Key Generation. We observe that the NTT function is used k times in step 17 and k times in step 18; i.e., each time taking as input a new component of the k-size vectors $\mathbf{s}$ and $\mathbf{e}$.

Encryption. Here, the NTT function is used k times in step 18 (i.e., once for every component of vector $\mathbf{r}$), and the NTT^{-1} function is used k times in step 19 (i.e., once for every component of vector $\hat{\mathbf{A}}^T \circ \hat{r}$) and once in step 20.

Decryption. Here, the NTT function is used k times in step 3 (i.e., once for every component of vector $\mathbf{r}$), and the NTT^{-1} function is used once in step 4.

Algorithm 1 KYBER.CPA.PKE.KeyGen(): key generation

1: $\boldsymbol{d} \leftarrow \mathbb{B}^{32}$
2: $(\rho, \sigma) \leftarrow G(\boldsymbol{d})$
3: $N \leftarrow 0$
4: **for** i from 0 to $k-1$ **do**
5: **for** j from 0 to $k-1$ **do**
6: $\boldsymbol{A}[i][j] \leftarrow \text{Parse}(XOF(\rho, j, i))$
7: **end for**
8: **end for**
9: **for** i from 0 to $k-1$ **do**
10: $\boldsymbol{s}[i] \leftarrow CBD_{\eta_1}(\text{PRF}(\sigma, N))$
11: $N \leftarrow N+1$
12: **end for**
13: **for** i from 0 to $k-1$ **do**
14: $\boldsymbol{e}[i] \leftarrow CBD_{\eta_1}(\text{PRF}(\sigma, N))$
15: $N \leftarrow N+1$
16: **end for**
17: $\hat{\boldsymbol{s}} \leftarrow \text{cwise-NTT}(\boldsymbol{s})$
18: $\hat{\boldsymbol{e}} \leftarrow \text{cwise-NTT}(\boldsymbol{e})$
19: $\hat{\boldsymbol{t}} \leftarrow \hat{\boldsymbol{A}} \circ \hat{\boldsymbol{s}} + \hat{\boldsymbol{e}}$
20: $\text{pk} \leftarrow (\text{Encode}_{12}(\hat{\boldsymbol{t}} \mod q) \parallel \rho)$
21: $\text{sk} \leftarrow \text{Encode}_{12}(\hat{\boldsymbol{s}} \mod q)$
22: **return** (pk, sk)

Algorithm 2 KYBER.CPA.PKE.Enc(pk, m, r): encryption

1: $N \leftarrow 0$
2: $\boldsymbol{t} \leftarrow \text{Decode}_{12}(\text{pk})$
3: $p \leftarrow \text{pk} + 12 \cdot k \cdot n/8$
4: **for** i from 0 to $k-1$ **do**
5: **for** j from 0 to $k-1$ **do**
6: $\hat{\boldsymbol{A}}^T[i][j] \leftarrow \text{Parse}(XOF(p, i, j))$
7: **end for**
8: **end for**
9: **for** i from 0 to $k-1$ **do**
10: $\boldsymbol{r}[i] \leftarrow CBD_{\eta_1}(\text{PRF}(r, N))$
11: $N \leftarrow N+1$
12: **end for**
13: **for** i from 0 to $k-1$ **do**
14: $\boldsymbol{e_1}[i] \leftarrow CBD_{\eta_2}(\text{PRF}(r, N))$
15: $N \leftarrow N+1$
16: **end for**
17: $e_2 \leftarrow CBD_{\eta_2}(\text{PRF}(r, N))$
18: $\hat{\boldsymbol{r}} \leftarrow \text{cwise-NTT}(\boldsymbol{r})$
19: $\boldsymbol{u} \leftarrow \text{cwise-NTT}^{-1}(\hat{\boldsymbol{A}}^T \circ \hat{\boldsymbol{r}}) + \boldsymbol{e_1}$
20: $v \leftarrow \text{NTT}^{-1}(\boldsymbol{t}^T \circ \hat{\boldsymbol{r}}) + \boldsymbol{e_2} + \text{Decompress}_q(\text{Decode}(m), 1)$
21: $\boldsymbol{c_1} \leftarrow \text{Encode}_{d_u}(\text{Compress}_q(\boldsymbol{u}, d_u))$
22: $c_2 \leftarrow \text{Encode}_{d_v}(\text{Compress}_q(v, d_v))$
23: **return** $\boldsymbol{c} \leftarrow (\boldsymbol{c_1} \parallel \boldsymbol{c_2})$

Algorithm 3 KYBER.CPA.PKE.Dec(sk, c): decryption

1: $\boldsymbol{u} \leftarrow \text{Decompress}_q(\text{Decode}_{d_u}(\boldsymbol{c}), d_u)$
2: $v \leftarrow \text{Decompress}_q(\text{Decode}_{d_v}(\boldsymbol{c} + d_u \cdot k \cdot n/8), d_v)$
3: $\boldsymbol{s} \leftarrow \text{Decode}_{12}(\text{sk})$, $\hat{\boldsymbol{u}} \leftarrow \text{cwise-NTT}(\boldsymbol{u})$
4: $m \leftarrow \text{Encode}_1(\text{Compress}_q(v - \text{NTT}^{-1}(\hat{\boldsymbol{s}}^T \circ \hat{\boldsymbol{u}}), 1))$
5: **return** m

4.2 Delegated Crystals-Kyber

We discuss some approaches on delegating any NTT operation in the Crystals-Kyber cryptosystem using our protocols from Section 3. We use the shorthand KeyGen, Enc and Dec to denote its key-generation, encryption and decryption algorithms.

Direct NTT and NTT^{-1} delegation. In this approach, every execution of the NTT or NTT^{-1} functions is replaced by an appropriate delegation protocol for that function. Specifically, we can use: (a) an execution of protocol $\mathcal{P}_{1,1}$ to delegate each of the $2k$ NTT executions in KeyGen, each of the k executions in step 18 of Enc, and each of the k executions in step 3 of Dec; and (b) an execution of protocol $\mathcal{P}_{2,1}$ to delegate each of the $k+1$ NTT^{-1} executions in steps 19 and 20 of Enc, and the NTT^{-1} execution in step 4 of Dec.

Using batch delegation. In this approach, any 2 or more executions of the NTT or NTT^{-1} functions within the same algorithm are replaced by an appropriate *batch* delegation protocol for that function. Specifically, we can use: (a) an execution of batch NTT delegation protocol $\mathcal{P}_{3,1}$ to delegate a batch of $2k$ NTT executions in KeyGen, a batch of k executions in step 18 of Enc, and a batch of k executions in step 3 of Dec; and (b) an execution of batch NTT^{-1} delegation protocol $\mathcal{P}_{3,5}$ to delegate a batch of $k+1$ NTT^{-1} executions in steps 19 and 20 of Enc and protocol $\mathcal{P}_{2,1}$ to delegate the single NTT^{-1} execution in step 4 of Dec.

Using NTT and NTT^{-1} correctness. The previous two approaches only assumed linearity and efficiency properties of the NTT method. If we also assume correctness property (which is usually possible), and are interested in reducing communication complexity, we can also replace every use of protocol $\mathcal{P}_{i,1}$ with protocol $\mathcal{P}_{i,2}$, for $i = 1, 2, 3$, and every use of protocol $\mathcal{P}_{3,5}$ with protocol $\mathcal{P}_{3,6}$, in both of the above approaches.

Using more precomputation. In all above approaches we have assumed that all inputs are only available in the online phase (a worst-case scenario). In use cases, where inputs to the NTT operations are available in the offline phase, another optimization would consist of directly computing offline the specific NTT or NTT^{-1} operation. This might be the case, for instance, for steps 18 and 19 of Enc.

5 NUMERIC PERFORMANCE ANALYSIS

We show numerical results from our software implementation of our delegation protocols as well as a delegated version of the Crystals-Kyber cryptosystem. Our software implementation was carried out on a macOS Sonoma Version 14.5 laptop with apple M1 Max chip and 64 GB of memory with 8 cores of performance and 2 of efficiency. We used the python libraries: numpy, pycryptodome and kyber-py from [31] for all protocols. We set the range for parameter k (denoting the vector dimension in Crystal-Kyber) as

follows: $k \in \{1, \ldots, 4\}$ in protocols $\mathcal{P}_{1,\cdot}$, and $\mathcal{P}_{2,\cdot}$; and $k \in \{2, \ldots, 8\}$ for protocols $\mathcal{P}_{3,\cdot}$, indicating a preference for a larger number of polynomials in batch implementations.

5.1 Single/Batch NTT, NTT^{-1} delegation results

Our first set of results is for the main protocols in Section 3, and are captured in Table 2 and 3.

We focus our numeric analysis on measuring the *delegation runtime improvement ratio* t_F/t_C as an estimate of the effectiveness of delegation (i.e., delegation results in an improvement of the online client runtime if and only if this ratio is > 1 and, whenever it is > 1, the higher the ratio, the higher the improvement.) In both tables we choose the value range for k and values for q that are consistent with typical parameter choices in Crystal-Kyber's round-1 and round-3 NIST submissions. We also note that for these choices of q, a choice of $r = 5$ already guarantees a security probability of error $\leq 2^{-59}$, showing that in important practical applications, r can be set as a small constant.

A major takeaway is that for most parameter choices, delegation improves online client runtime by about 1 order of magnitude. Other takeaways on the measured delegation improvement ratio include:

(1) it increases as k increases;
(2) it increases as r decreases;
(3) in the 'x public' scenario, it is generally higher than in the 'x private' scenario;
(4) in delegation of a batch of 2 inputs, it is generally higher than in single-input delegation.

5.2 Crystals-Kyber delegation results

Our second set of results is for the delegation of NTT operations in Crystals-Kyber, and are captured in Table 4 and 5. We implemented the 'direct NTT and NTT^{-1} delegation' approach, as discussed in Section 4.2, as a worst-case validation scenario of the delegation approach, since all other approaches perform even further optimizations.

We focus our numeric analysis on measuring the *client online runtime* t_C in the delegated version, and the algorithm runtime in the original (non-delegated) version of the cryptosystem. Thus, delegation results in an improvement of the client's runtime in running the algorithm if and only if he time measured in the delegated scenario is lower than the time measured in the original scenario. In both tables we choose the value range for k and values for q that are consistent with typical parameter choices in Crystal-Kyber's round-1 and round-3 NIST submissions.

A major takeaway is that for all parameter choices, delegation always improves online client runtime, despite the original algorithms having gone through years of improvement and optimization studies. In the case $q = 3329$, the average runtime reduction due to delegation was 15.43% on the key generation algorithm, 15.26% on the encryption algorithm, and 15.28% on the decryption algorithm, In the case $q = 7681$, we measured the following numbers: 13.25%, 17.11%, and 16.43%.

Table 2: Ratio t_F/t_C of non-delegated NTT computation to client runtime in the online phase, with respect to protocols $\mathcal{P}_{1.1}, \mathcal{P}_{1,3}, \mathcal{P}_{2,1}, \mathcal{P}_{3,1}, \mathcal{P}_{3,5}$, where $n = 256$, $q = 3329$, $r \in \{1, \ldots, 6\}$ and $k \in \{1, \ldots, 8\}$, averaged over 10,000 repetitions.

$q = 3329$ **Ratio:** t_F/t_C						
	$r = 1$ $\lambda = 12$	$r = 2$ $\lambda = 24$	$r = 3$ $\lambda = 36$	$r = 4$ $\lambda = 47$	$r = 5$ $\lambda = 59$	$r = 6$ $\lambda = 71$
$\mathcal{P}_{1,1}$**: Delegation of** $NTT(x)$**,** x **private**						
$k = 1$	5.72	4.47	3.82	3.45	3.02	2.73
$k = 2$	10.34	7.47	6.20	6.06	4.82	4.08
$k = 3$	12.28	8.90	7.90	7.72	6.64	5.13
$k = 4$	14.03	11.08	8.69	8.63	4.87	4.49
$\mathcal{P}_{1,3}$**: Delegation of** $NTT(x)$**,** x **public**						
$k = 1$	5.25	4.17	3.44	2.64	2.51	2.25
$k = 2$	9.82	6.89	5.27	5.30	5.04	3.73
$k = 3$	15.51	11.99	9.04	7.57	6.38	5.40
$k = 4$	16.60	12.84	12.05	9.98	7.99	6.05
$\mathcal{P}_{2,1}$**: Delegation of** $NTT^{-1}(x)$**,** x **private**						
$k = 1$	8.79	6.87	5.93	5.26	4.65	4.10
$k = 2$	18.30	13.25	10.60	8.78	6.56	6.38
$k = 3$	22.87	16.57	12.62	11.59	9.98	7.90
$k = 4$	25.08	19.38	15.50	12.49	10.93	9.48
$\mathcal{P}_{3,1}$**: Batch Delegation of** $NTT(x_i), i = 1, 2, x_i$ **private**						
$k = 2$	7.62	6.54	5.83	5.16	4.54	4.13
$k = 4$	14.97	12.10	8.66	8.17	7.29	7.01
$k = 6$	20.69	16.67	13.89	12.07	10.58	8.93
$k = 8$	25.21	18.62	16.89	14.57	12.78	9.78
$\mathcal{P}_{3,5}$ **: Batch Delegation of** $NTT^{-1}(x_i), i = 1, 2, x_i$ **private**						
$k = 2$	10.70	8.82	8.01	7.18	6.85	6.36
$k = 4$	23.30	18.99	16.17	13.78	11.44	10.92
$k = 6$	30.21	23.25	20.24	16.14	13.57	13.02
$k = 8$	28.40	20.57	19.26	15.30	14.01	13.16

6 CONCLUSIONS

We studied the problem of a resource-constrained client delegating NTT computations to a single, computationally stronger, server, possibly implemented in a cloud-based architecture. We showed several protocols for a client to delegate single/batch computations of the NTT and NTT^{-1} functions, in input scenarios where the input polynomial(s) are private or public, where the client performs a smaller number of multiplications in $\mathbb{Z}_q$ with respect to non-delegated NTT or NTT^{-1} computations. The improvements were both of asymptotic nature, for a large range of the involved parameters, and are noticeable for practical parameter settings. Our software implementation achieved one-order-of-magnitude improvements for NTT computations, and managed to improve the client online runtime for the algorithms in the already significantly optimized Crystals-Kyber cryptosystem, which is currently recommended for standardization. Our methods suggest that the most expensive operations in many lattice-based cryptographic protocols (specifically, Number Theoretic Transforms) can be delegated to a cloud server, with considerable savings on client computation (e.g., about 1 order of magnitude).

Table 4: Time of Kyber Algorithms of KeyGen, Enc, Dec using Delegation of each single NTT and NTT^{-1} computation (delegated) vs. time of Original Kyber512, Kyber 768, Kyber 1024 algorithms (original), $q = 3329$, averaged over 100 repetitions.

			Time		
$q = 3329$		Name	Keygen	Enc	Dec
$r = 4$	DELEGATED	Kyber512	0.266	0.371	0.597
		Kyber768	0.374	0.537	0.85
		Kyber1024	0.557	0.759	1.182
	ORIGINAL	Kyber512	0.331	0.462	0.723
		Kyber768	0.459	0.67	1.041
		Kyber1024	0.673	0.933	1.429
$r = 5$	DELEGATED	Kyber512	0.266	0.376	0.599
		Kyber768	0.385	0.551	0.873
		Kyber1024	0.584	0.785	1.219
	ORIGINAL	Kyber512	0.307	0.455	0.723
		Kyber768	0.459	0.673	1.067
		Kyber1024	0.675	0.938	1.438
$r = 6$	DELEGATED	Kyber512	0.288	0.403	0.627
		Kyber768	0.387	0.555	0.905
		Kyber1024	0.564	0.769	1.193
	ORIGINAL	Kyber512	0.311	0.461	0.736
		Kyber768	0.461	0.706	1.05
		Kyber1024	0.701	1.005	1.454

Table 5: Time of Kyber Algorithms of KeyGen, Enc, Dec using Delegation of each single NTT and NTT^{-1} computation (delegated) vs. time of Original Kyber512, Kyber 768, Kyber 1024 algorithms (original), $q = 7681$, averaged over 100 repetitions.

			Time		
$q = 7681$		Name	Keygen	Enc	Dec
$r = 4$	DELEGATED	Kyber512	0.262	0.373	0.595
		Kyber768	0.379	0.544	0.86
		Kyber1024	0.573	0.782	1.231
	ORIGINAL	Kyber512	0.302	0.45	0.712
		Kyber768	0.447	0.657	1.024
		Kyber1024	0.656	0.916	1.402
$r = 5$	DELEGATED	Kyber512	0.277	0.38	0.614
		Kyber768	0.385	0.554	0.888
		Kyber1024	0.56	0.766	1.187
	ORIGINAL	Kyber512	0.3	0.45	0.714
		Kyber768	0.448	0.659	1.027
		Kyber1024	0.657	0.914	1.4
$r = 6$	DELEGATED	Kyber512	0.27	0.379	0.607
		Kyber768	0.38	0.549	0.863
		Kyber1024	0.552	0.754	1.172
	ORIGINAL	Kyber512	0.303	0.453	0.717
		Kyber768	0.452	0.664	1.035
		Kyber1024	0.669	0.932	1.43

Table 3: Ratio t_F/t_C of non-delegated NTT computation to client runtime in the online phase, with respect to protocols $\mathcal{P}_{1.1}, \mathcal{P}_{1,3}, \mathcal{P}_{2,1}, \mathcal{P}_{3,1}, \mathcal{P}_{3,5}$, where $n = 256$, $q = 3329$, $r \in \{1, \ldots, 6\}$ and $k \in \{1, \ldots, 8\}$, averaged over 10,000 repetitions.

$q = 7681$ **Ratio:** t_F/t_C						
	$r = 1$ $\lambda = 13$	$r = 2$ $\lambda = 26$	$r = 3$ $\lambda = 39$	$r = 4$ $\lambda = 52$	$r = 5$ $\lambda = 65$	$r = 6$ $\lambda = 78$
$\mathcal{P}_{1.1}$: **Delegation of** $NTT(x)$, x **private**						
$k = 1$	5.23	4.39	3.66	3.28	2.60	2.63
$k = 2$	12.06	8.94	6.25	5.91	5.09	4.30
$k = 3$	15.35	11.29	9.08	7.83	5.70	5.09
$k = 4$	13.58	10.86	8.43	6.82	6.23	5.07
$\mathcal{P}_{1,3}$: **Delegation of** $NTT(x)$, x **public**						
$k = 1$	5.78	4.65	3.70	3.17	2.87	2.81
$k = 2$	10.17	6.28	6.59	4.53	4.64	4.09
$k = 3$	17.43	11.66	10.07	7.69	6.93	5.51
$k = 4$	19.43	15.60	12.05	9.49	8.18	5.94
$\mathcal{P}_{2,1}$: **Delegation of** $NTT^{-1}(x)$, x **private**						
$k = 1$	10.22	8.27	6.68	5.67	4.88	4.36
$k = 2$	18.80	13.62	11.18	9.16	8.03	7.15
$k = 3$	24.49	17.39	14.45	12.17	10.45	9.17
$k = 4$	29.03	21.63	17.17	14.20	12.16	10.68
$\mathcal{P}_{3,1}$: **Batch Delegation of** $NTT(x_i), i = 1, 2, x_i$ **private**						
$k = 2$	7.66	6.06	5.16	5.09	4.57	3.72
$k = 4$	11.43	10.08	8.78	7.24	6.87	5.77
$k = 6$	16.51	13.46	11.35	9.59	8.84	7.59
$k = 8$	19.84	15.51	13.01	12.21	11.04	9.00
$\mathcal{P}_{3,5}$: **Batch Delegation of** $NTT^{-1}(x_i), i = 1, 2, x_i$ **private**						
$k = 2$	11.18	11.04	9.44	8.26	7.34	6.19
$k = 4$	18.03	15.39	14.25	12.93	10.61	10.16
$k = 6$	25.73	23.04	17.53	15.75	13.23	11.90
$k = 8$	29.93	27.46	20.90	18.38	19.02	16.65

Acknowledgement. We thank Reza Azarderakhsh and Vladimir Shpilrain for interesting discussions. Work by Matluba Khodjaeva and Dilan Morales Caro was supported by a Faculty Scholarship grant from the Office for the Advancement of Research at John Jay College and PSC CUNY Cycles 53 and 54. Work by Giovanni Di Crescenzo was supported by the Defense Advanced Research Projects Agency (DARPA), contract n. HR001120C0156. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation hereon. Disclaimer: The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of DARPA, or the U.S. Government. Distribution Statement "A" (Approved for Public Release, Distribution Unlimited).

REFERENCES

[1] M. Abadi, J. Feigenbaum, J. Kilian, *On Hiding Information from an Oracle* In: J. Comput. Syst. Sci. 39(1): 21-50 (1989)
[2] R. Avanzi, J. Bos, L. Ducas, E. Kiltz, T. Lepont, V. Lyubashevsky, J. M. Schanck, P. Schwabe, G. Seiler, D. Stehle, *Crystals-Kyber: Algorithm Specifications and supporting documentation, version 3.02*, NIST PQC, 2021
[3] R. Agarwal, and S. Burrus, *Number theoretic transforms to implement fast digital convolution.* In: Proceedings of the IEEE, 63(4):550–560, 1975.
[4] H. Ahmad, L. Wang, H. Hong, J. Li, H. Dawoo, M. Ahmed, and Y. Yang, *Primitives towards verifiable computation: a survey.* In: Front. Comput. Sci. 2018, Vol. 12, Issue (3) : 451-478.
[5] A. Arbit and Y. Livne and Y. Oren and A. Wool, *Implementing public-key cryptography on passive RFID tags is practical.* In: Int. J. Inf. Sec. 14(1): 85-99, 2015.

[6] R. Avanzi, J. W. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. M. Schanck, P. Schwabe, G. Seiler, and D. Stehlé. *Supporting documentation: Crystals-kyber: Algorithm specifications and supporting documentation (version 3.0).* In: NIST PQC, 2020.
[7] Luke Beckwith, Abubakr Abdulgadir, Reza Azarderakhsh: A Flexible Shared Hardware Accelerator for NIST-Recommended Algorithms CRYSTALS-Kyber and CRYSTALS-Dilithium with SCA Protection. in Proc. of CT-RSA 2023, pp. 469-490
[8] J. W. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. M. Schanck, P. Schwabe, G. Seiler, and D. Stehlé. *CRYSTALS - kyber: A cca-secure module-lattice-based KEM.* In: EuroS&P 2018, pages 353–367, 2018
[9] S. A. Cook, and S. O. Aanderaa, *On the minimum computation time of functions.* In: Transactions of the American Mathematical Society, volume 142, pages 291–314, 1969.
[10] X. Chen, W. Susilo, J. Li, D. S. Wong, J. Ma, S. Tang, Q. Tang, *Efficient algorithms for secure outsourcing of bilinear pairings.* In Theoretical Computer Science, vol. 562: pp. 112–121, 2015.
[11] E. Chu, and A. George. *Inside the FFT black box: serial and parallel fast Fourier transform algorithms.* In: CRC press, 1999.
[12] J. W Cooley, and J. W Tukey. *An algorithm for the machine calculation of complex fourier series.* In: Mathematics of computation, 19(90):297–301, 1965.
[13] G. Di Crescenzo, M. Khodjaeva, D. Kahrobaei, and V. Shpilrain: *A Survey on Delegated Computation.* In: Proc. of DLT 2022, LNCS, Springer: pp. 33-53.
[14] J. Feigenbaum, *Encrypting Problem Instances: Or ..., Can You Take Advantage of Someone Without Having to Trust Him?*, in Proc. of CRYPTO 1985: 477-488
[15] A. Fiat, *Batch RSA.* In Journal of Cryptology, v. 10, n. 2, 1997.
[16] R. Gennaro, C. Gentry, and B. Parno, *Non-interactive verifiable computing: Outsourcing computation to untrusted workers.* In: Advances in Cryptology, CRYPTO 2010, Lecture Notes Comp. Sc. **6223**, 465-482, 2010.
[17] W. Morven Gentleman, and G. Sande. *Fast fourier transforms: for fun and profit.* In: AFIPS '66, volume 29, pages 563–578, 1966.
[18] S. Hohenberger and A. Lysyanskaya, *How to securely outsource cryptographic computations.* In: Proceedings of the Theory of Cryptography Conference 2005, pages 264-282, Springer, 2005.
[19] D. E Knuth, *Art of computer programming, volume 2: Seminumerical algorithms.* In: Addison-Wesley Professional, 2014.
[20] A. A. Karatsuba, and Y. Ofman, *Multiplication of many-digital numbers by automatic computers.* In: Doklady Akademii Nauk, 145(2):293–294.
[21] N. Koblitz, *Elliptic Curve Cryptosystems*, in Mathematics of Computation, 48(177), pp. 203-209, 1987.
[22] Z. Liang, and Y. Zhao, *Number theoretic transform and its applications in lattice-based cryptosystems: A survey.* In: arXiv preprint arXiv:2211.13546, 2022.
[23] A. Langlois and D. Stehlé. *Worst-case to average-case reductions for module lattices.* In:Des. Codes Cryptogr., 75(3):565–599, 2015.
[24] T. Matsumoto, K. Kato, H. Imai, *An improved algorithm for secure outsourcing of modular exponentiations.* In: Proc. of CRYPTO 1988, pp. 497–506, LNCS, Springer.
[25] V. S. Miller, *Use of Elliptic Curves in Cryptography*, in Proc. of CRYPTO '85, LNCS vol. 218, pp. 417-426, 1986
[26] NIST. Post-quantum cryptography, round 1 submissions. https://csrc.nist.gov/Projects/Post-Quantum-Cryptography/round-1-submissions, 2016.
[27] NIST. Post-quantum cryptography, round 2 submissions. https://csrc.nist.gov/Projects/Post-Quantum-Cryptography/round-2-submissions, 2019.
[28] NIST. Post-quantum cryptography, round 3 submissions. https://csrc.nist.gov/Projects/Post-Quantum-Cryptography/round-3-submissions, 2020.
[29] NIST. Pqc standardization process: Announcing four candidates to be standardized, plus fourth round candidates. https://csrc.nist.gov/News/2022/pqc-candidates-to-be-standardized-and-round-4, 2022.
[30] J. M Pollard. *The fast fourier transform in a finite field.* In: Mathematics of computation, 25(114):365–374, 1971.
[31] G. Pope, (n.d.). *kyber-py: A pure Python implementation of the Kyber post-quantum cryptography algorithm. GitHub repository.* Retrieved from https://github.com/GiacomoPope/kyber-py
[32] R. L. Rivest, A. Shamir, L. Adleman, *A Method for Obtaining Digital Signatures and Public-Key Cryptosystems.* Communications of the ACM, 21(2), pp. 120-126, 1978
[33] Z. Shan, K. Ren, M. Blanton, C. Wang, *Practical Secure Computation Outsourcing: A Survey.* In: ACM Comput. Surv. 51(2): 31:1-31:40, 2018.
[34] P. W. Shor. *Algorithms for Quantum Computation: Discrete Logarithms and Factoring.* in Proc. of the 35th Annual Symposium on Foundations of Computer Science, pp. 124-134, 1984
[35] Q. Sun, D. Zheng, and Z. Shen. *Fast Number Theoretic Transform.* In: China Science Press, 1980.
[36] L. Toom. *The complexity of a scheme of functional elements realizing the multiplication of integers.* In: Soviet Mathematics Doklady, 3(4):714–716, 1963.
[37] P. Tsang, S. Chow, and S. Smith *Batch pairing delegation.* In: Proceedings of International Workshop on Security, 2007, pp.74–90. Springer, Berlin.
[38] A. Weimerskirch, and C. Paar. *Generalizations of the karatsuba algorithm for efficient implementations.* In: IACR Cryptol. ePrint Arch., 2006:224, 2006.
[39] F. Winkler.*Polynomial Algorithms in Computer Algebra.* In: Springer, 1996.
[40] A. C. Yao, *Protocols for secure computations.* In: Proceedings of the 23rd Annual Symposium on Foundations of Computer Science, pp. 160-168, IEEE Computer Society, 1982.

A PROTOCOL 1.2 - PROOFS OF PROPERTIES

The efficiency properties are verified by protocol inspection. In particular:

- *Storage Complexity:* at the end of the offline phase 3 values in R_q and 3 values in $(\mathbb{Z}_q)^n$ are stored on C's device.
- *Round complexity:* The online phase of the protocol only requires two messages: one from C to S, followed by one from S to C.
- *Communication complexity:* During the online phase, C sends 2 values in R_q, and S sends 2 values in $(\mathbb{Z}_q)^n$.
- *Runtime complexity:* During the online phase, C performs one multiplication $*_\phi$ between an $(r-1)$-degree polynomial $b \in R_q$ and an $(n-1)$-degree polynomial $z_0 \in R_q$, requiring at most $O(nr)$ multiplications $*_q$ in $\mathbb{Z}_q$, and one multiplication $*_t$ between two $(n-1)$-degree polynomials in NTT representation, requiring at most $O(n)$ multiplications $*_q$ in $\mathbb{Z}_q$, as well as a number of lower-order operations (i.e., 2 additions $+_\phi$, or subtractions $-_\phi$, in R_q; one addition $+_t$ in NTT representation; and n checks for membership in $\mathbb{Z}_q$). S only performs 2 computation of the NTT function. The offline phase only requires 3 computation of the NTT function. See also Table 1 (bottom).

The *result correctness* property follows by showing that if C and S follow the protocol, both the membership and probabilistic checks are satisfied, and C always outputs $y = \mathrm{NTT}(x)$.

The satisfaction of the membership check directly follows by the NTT function's definition.

The satisfaction of the probabilistic check follows by observing that

$$\begin{aligned} w_1 &= \mathrm{NTT}(z_1) \\ &= \mathrm{NTT}(b *_\phi z_0 +_\phi u_i) \\ &= \mathrm{NTT}(b *_\phi z_0) +_t \mathrm{NTT}(u_i) \\ &= (\mathrm{NTT}(b) *_t \mathrm{NTT}(z_0)) +_t \mathrm{NTT}(u_i) \\ &= c *_t w_0 +_t v_1, \end{aligned}$$

where the 1st equality follows by how S computes w_1, the 2nd equality follows by how C computes z_1, the 3rd equality follows from the NTT-linearity of the NTT method, the 4th equality follows from the computation correctness of the NTT method, and the last equality follows by how w_0 is computed by S, c is computed by C and v_1 is computed in the offline phase.

This implies that C never returns $\perp$, and thus always returns

$$\begin{aligned} y &= w_0 +_t v_0 = \mathrm{NTT}(z_0) +_t \mathrm{NTT}(u_0) \\ &= \mathrm{NTT}(x -_\phi u_0) +_t \mathrm{NTT}(u_0) = \mathrm{NTT}(x), \end{aligned}$$

where the 2nd equality follows by how S computes w_0 and how v_0 was generated in the offline phase, the 3rd equality follows by how C computes z_0 and the 4th equality follows from the NTT-linearity property.

The *input privacy* property of the protocol against any malicious S follows by observing that, C's message (z_0, z_1) to S does not leak

any information about C's input x. This follows because values u_0, u_1 are uniformly and independently chosen from R_q. Moreover, by similar reasoning, this message does not leak any information about polynomial b, a fact useful in the proof of the security property.

To prove the *result security* property against any malicious S, we need to compute an upper bound ϵ_s on the security probability that S convinces C to output a y such that $y \neq \mathrm{NTT}(x)$. We obtain that $\epsilon_s \leq 2^{-\lambda}$, as a consequence of the following three facts, which we later prove:

1. pair (z_0, z_1) leaks no information about b to S, where $z_0 = x -_\phi u_0$ and $z_1 = b *_\phi z_0 +_\phi u_1$, where $x, u_0, u_1 \in R_q$ and $b \in \mathbb{Z}_q$;
2. for any S's message $w = (w_0, w_1)$ such that at least one of w_0, w_1 is different than what would be returned according to the protocol instructions, the number of $(r-1)$-degree polynomials $b \in R_q$ for which pair (w_0, w_1) satisfies both the membership and the probabilistic test in the protocol is ≤ 1;
3. for any S's message $w = (w_0, w_1)$ such that at least one of w_0, w_1 is different than what would be returned according to the protocol instructions, the probability that w satisfies the probabilistic test is $\leq 2^{-\lambda}$.

To prove Fact 1, we observe that z_0 and z_1 are uniformly and independently distributed in R_q since so are u_0 and u_1. We also observe that, for the same reason, the distribution of pair (z_0, z_1) is independent from that of b, a fact that will be useful in the proof of the result security property.

Towards proving Fact 2, let $w = (w_0, w_1)$ be the pair that would be returned by S according to the protocol, and assume that a malicious algorithm Adv, corrupting S, returns a different tuple $w' = (w'_0, w'_1)$. Note that if $w'_i \notin (\mathbb{Z}_q)^n$ for some $i \in \{0, 1\}$, the pair w' does not satisfy the membership test. Thus, for all $i = 0, 1$. we assume that $w'_i \in (\mathbb{Z}_q)^n$ and we can then write w'_i as $(w'_{i,0}, \ldots, w'_{i,n-1})$, where $w'_{i,j} \in \mathbb{Z}_q$, for all $j = 0, \ldots, n-1$. Because $\mathbb{Z}_q$ is also a group with respect to addition $+_q$, we can write, for $i \in \{0, 1\}$ and $j \in [0, n-1]$,

$$w'_{i,j} = d_{i,j} +_q w_{i,j},$$

for some $d_{i,j} \in \mathbb{Z}_q$ such that not all of the $d_{i,j}$ are equal to 1.

Now, consider the probabilistic test equation

$$w'_1 = b *_\phi w'_0 +_\phi v_1,$$

and note that it can be rewritten as n equations, where the j-th equation, for any $j \in [0, n-1]$, can be written as

$$w'_{1,j} = b *_q w'_{0,j} +_q v_{1,j}.$$

Then, consider the following equivalent rewritings of the (i, j)-th equation in the probabilistic test, obtained by variable substitutions and simplifications:

$$\begin{aligned} w'_1 &= b *_\phi w'_0 +_\phi v_1 \\ d_i +_q w_1 &= b *_\phi (d_0 +_q w_0) +_\phi v_1 \\ d_i +_q (b *_\phi w_0 +_\phi v_1) &= b *_\phi (d_0 +_q w_0) +_\phi v_1 \\ d_i +_q v_1 &= b *_q d_0 +_\phi v_1 \\ d_i &= b *_\phi d_0 \end{aligned}$$

where the third equality follows from the protocol's result correctness property. Now, if for any pair (d_0, d_1), there exist at least two distinct $(r-1)$-degree polynomials b_0 and b_1 in R_q satisfying this latter equation, we have that

$$\begin{aligned} d_i &= b_0 *_\phi d_0, \text{and} \\ d_i &= b_1 *_\phi d_0. \end{aligned}$$

By applying the $-_\phi$ operation, we have that

$$0 = (b_0 -_\phi b_1) *_\phi d_0,$$

which implies that at least one of the following two conditions holds:

(1) $b_1 = b_0$,
(2) $d_0 = 0$.

Note that if (1) holds, the fact follows. On the other hand, if (2) holds, we also have that $d_1 = 0$, which implies that $w'_i = w_i$, for $i = 0, 1$, implying that S was honest. This proved Fact 2.

Towards proving Fact 3, first recall Fact 1 saying that C's message (z_0, z_1) does not leak any information about the $(r-1)$-degree polynomial $b \in R_q$, and thus all possibly values in $(\mathbb{Z}_q)^r$ are still equally likely for $(b_0, \ldots, b_r)$ even when conditioning over C's message. Then we can compute an upper bound on the probability that S's message w satisfies the i-th equation of the probabilistic test, as the ratio between the number of $(r-1)$-degree polynomial $b \in R_q$ for which w satisfies this equation, and the number of $(r-1)$-degree polynomial $b \in R_q$ that are still equally likely even when conditioning over C's message. In this ratio, the numerator is at most 1 (by Fact 2), and the denominator is the number of $(r-1)$-degree polynomials with coefficients in $\mathbb{Z}_q$ (by Fact 1). Since there are q^r such polynomials, and $r = \lceil \lambda / \log q \rceil$, the probability that w satisfies the probabilistic test is

$$\leq \left(\frac{1}{q}\right)^r = \left(\frac{1}{q}\right)^{\lceil \lambda / \log q \rceil} \leq \frac{1}{2^\lambda}.$$

This proves Fact 3.

B PROTOCOL 3.1 - PROOFS FOR PROPERTIES

The efficiency properties are verified by protocol inspection. In particular:

- *Storage Complexity:* at the end of the offline phase $r + m + 2$ polynomials in R_q and $r + m + 2$ tuples in $(\mathbb{Z}_q)^n$ are stored on C's device.
- *Round complexity:* The online phase of the protocol only requires two messages: one from C to S, followed by one from S to C.
- *Communication complexity:* During the online phase, C sends $r + m$ polynomials in R_q, and S sends $r + m$ tuples in $(\mathbb{Z}_q)^n$.
- *Runtime complexity:* During the online phase, C performs mr $*_\phi$ operations between a scalar $b_{i,j}$ and a polynomial $z_{0,j}$, mr $*_t$ operations and at most mr $+_t$ operations, all three of which requiring at most $O(nmr)$ multiplications $*_q$ in $\mathbb{Z}_q$, as well as a number of lower-order operations (i.e., $m(r+1)$ additions or subtractions in R_q, and mn checks for membership in $\mathbb{Z}_q$). S only performs $r + m$ computation of the NTT function. The offline phase only requires $r + m$ computation of the NTT function. See also Table 1 (bottom).

The *result correctness* property follows by showing that if C and S follow the protocol, both the membership and probabilistic checks are satisfied, and C always outputs $y_j = \mathrm{NTT}(x_j)$, for $j = 0, \ldots, m-1$.

The satisfaction of the membership check directly follows by the NTT function's definition.

The satisfaction of the probabilistic check follows by observing that for $i = 0, \ldots, r-1$,

$$\begin{aligned} w_{1,i} &= \mathrm{NTT}(z_{1,i}) \\ &= \mathrm{NTT}\left((\sum_{\phi;j=0}^{m-1} b_{i,j} *_\phi z_{0,j}) +_\phi \alpha_i \right) \\ &= \mathrm{NTT}\left(\sum_{\phi;j=0}^{m-1} b_{i,j} *_\phi z_{0,j} \right) +_t \mathrm{NTT}(\alpha_i) \\ &= \left(\sum_{t;j=0}^{m-1} b_{i,j} *_t \mathrm{NTT}(z_{0,j}) \right) +_t \mathrm{NTT}(\alpha_i) \\ &= \left(\sum_{t;j=0}^{m-1} b_{i,j} *_t w_{0,j} \right) +_t \beta_i \end{aligned}$$

where the 1st equality follows by how S computes $w_{1,i}$, the 2nd equality follows by how C computes $z_{1,i}$, the 3rd and 4th equalities follows from different applications of the NTT-linearity, and the last equality follows by how S computes $w_{0,j}$ and C computes β_i.

This implies that C never returns $\perp$, and thus always returns, for $j = 0, 1, \ldots, m-1$,

$$\begin{aligned} y_j &= w_{0,j} +_t v_{0,j} = \mathrm{NTT}(z_{0,j}) +_t \mathrm{NTT}(u_{0,j}) \\ &= \mathrm{NTT}(x_j -_\phi u_{0,j}) +_t \mathrm{NTT}(u_{0,j}) = \mathrm{NTT}(x_j), \end{aligned}$$

where the 2nd equality follows by how S computes $w_{0,j}$ and how $v_{0,j}$ was generated in the offline phase, the 3rd equality follows by how C computes $z_{0,j}$ and the 4th equality follows from the NTT-linearity property.

The *input privacy* property of the protocol in the presence of any malicious server S follows by observing that, client C's message $(z_{0,0}, \ldots, z_{0,m-1}, z_{1,0}, \ldots, z_{1,r-1})$ to S does not leak any information about C's inputs $x_0, \ldots, x_{m-1}$. This follows because values $u_0, u_1, \ldots, u_{m-1}$ and $\alpha_0, \alpha_1, \ldots, \alpha_{r-1}$ are uniformly and independently chosen from R_q. Moreover, by similar reasoning, this message does not leak any information about b_i, a fact useful in the proof of the security property.

To prove the *result security* property against any malicious S, we need to compute an upper bound ϵ_s on the security probability that S convinces C to output $(y_0, \ldots, y_{m-1})$ such that $y_j \neq \mathrm{NTT}(x_j)$ for some $j \in [1, m]$. We obtain that $\epsilon_s \leq 2^{-\lambda}$, as a consequence of the following three facts, which we later prove:

1. tuple $(z_{0,0}, \ldots, z_{0,m-1}, z_{1,0}, \ldots, z_{1,r-1})$ leaks no information about all $b_{i,j}$'s to S;
2. for any S's message $w = (w_0, w_1)$, where $w_0 = (w_{0,0}, \ldots, w_{0,m-1})$ and $w_1 = (w_{1,0}, \ldots, w_{1,r-1})$ such that at least one of
$$w_{0,0}, \ldots, w_{0,m-1}, w_{1,0}, \ldots, w_{1,r-1}$$
is different than what would be returned according to the protocol instructions, the number of tuples $(b_{i,j} \mid i \in [0, r-1], j \in [0, m-1]) \in (\mathbb{Z}_q)^{r+m}$ for which w satisfies both the membership and the probabilistic test in the protocol is $\leq q^{r+m-1}$;
3. for any S's message w such that at least one of
$$w_{0,0}, \ldots, w_{0,m-1}, w_{1,0}, \ldots, w_{1,r-1}$$
is different than what would be returned according to the protocol instructions, the probability that w satisfies the probabilistic test is $\leq 2^{-\lambda}$.

To prove Fact 1, we observe that all $z_{0,j}$ and $z_{1,i}$ values are uniformly and independently distributed in R_q since so are the u_j and α_i values, for $i = 0, 1, \ldots, r-1$ and $j = 0, 1, \ldots, m-1$. We also observe that, for the same reason, the distribution of C's message is independent from that of the $b_{i,j}$ values, a fact that will be useful in the proof of the result security property.

Towards proving Fact 2, let

$$w = (w_{0,0}, \ldots, w_{0,m-1}, w_{1,0}, \ldots, w_{1,r-1})$$

be the tuple that would be returned by S according to the protocol, and assume that a malicious algorithm Adv, corrupting S, returns a different tuple

$$w' = (w'_{0,0}, \ldots, w'_{0,m-1}, w'_{1,0}, \ldots, w'_{1,r-1}).$$

Note that if $w'_{0,j} \notin (\mathbb{Z}_q)^n$ for some $j \in \{0, 1, \ldots, m-1\}$, or $w'_{1,i} \notin (\mathbb{Z}_q)^n$ for some $i \in \{0, 1, \ldots, r-1\}$ the tuple w' does not satisfy the membership test. Thus, for all $i = 0, 1, \ldots, m-1$. and all $j = 0, 1, \ldots, r-1$, we assume that $w'_{0,i} \in (\mathbb{Z}_q)^n$ and $w'_{1,j} \in (\mathbb{Z}_q)^n$, and we can then write $w'_{0,i}$ as $(w'_{0,i,0}, \ldots, w'_{0,i,n-1})$, and $w'_{1,j}$ as $(w'_{1,j,0}, \ldots, w'_{1,j,n-1})$, where $w'_{0,i,\ell} \in \mathbb{Z}_q$ and $w'_{1,j,\ell} \in \mathbb{Z}_q$, for all $\ell = 0, \ldots, n-1$. Because $\mathbb{Z}_q$ is also a group with respect to addition $+_q$, we can write, for $i \in [0, r-1]$, $j \in [0, m-1]$, and $\ell \in [0, n-1]$,

$$w'_{0,i,\ell} = d_{0,i,\ell} +_q w_{0,i,\ell}, \text{ and } w'_{1,j,\ell} = d_{1,j,\ell} +_q w_{1,j,\ell},$$

for some $d_{0,i,\ell}, d_{1,j,\ell} \in \mathbb{Z}_q$ such that not all of these latter values are equal to 1.

Now, consider the i-th equation in the probabilistic test, for any $i \in [0, r-1]$:

$$w'_{1,i} = (\sum_{t;j=0}^{m-1} b_{i,j} *_t w'_{0,j}) +_t v_i$$

and note that each of these equations can be rewritten as n equations, where the (i, ℓ)-th equation, for any $\ell \in [0, n-1]$, can be written as

$$w'_{1,i,\ell} = (\sum_{t;j=0}^{m-1} b_{i,j} *_q w'_{0,j,\ell}) +_q v_{i,\ell}.$$

Then, consider the following equivalent rewritings of the (i, j)-th equation in the probabilistic test, obtained by variable substitutions

and simplifications:

$$w'_{1,i,\ell} = (\sum_{t;j=0}^{m-1} b_{i,j} *_q w'_{0,j,\ell}) +_q v_{i,\ell}.$$

$$d_{1,i,\ell} +_q w_{1,i,\ell} = (\sum_{t;j=0}^{m-1} b_{i,j} *_q (d_{0,j,\ell} + w_{0,j,\ell})) +_q v_{i,\ell}$$

$$d_{1,i,\ell} +_q w_{1,i,\ell} = \sum_{t;j=0}^{m-1} b_{i,j} *_q d_{0,j,\ell} +_q \sum_{t;j=0}^{m-1} b_{i,j} *_q w_{0,j,\ell} +_q v_{i,\ell}$$

$$d_{1,i,\ell} +_q v_{i,\ell} = (\sum_{t;j=0}^{m-1} b_{i,j} *_q d_{0,j,\ell}) +_q v_{i,\ell}$$

$$d_{1,i,\ell} = \sum_{t;j=0}^{m-1} b_{i,j} *_q d_{0,j,\ell}$$

where the fourth equality follows from the protocol's result correctness property. Now, if for any $(\{d_{0,j,\ell}\}, d_{1,i,\ell})$, and any $mr-1$ among all mr $b_{i,j}$ values in $\mathbb{Z}_q$ there exist at most one value for the remaining $b_{i,j}$ value (denoted as $b^*_{i,j}$) satisfying this latter equation. Then, we obtain, that for any $mr-1$ values among the mr $b_{i,j}$ values in $\mathbb{Z}_q$, there is at most one value for the remaining b_{ij} for which both the deterministic and the probabilistic check are satisfied. Then the fact follows by observing that there are at most q^{mr-1} possible values for $mr-1$ integers in $\mathbb{Z}_q$. This proves Fact 2.

Towards proving Fact 3, first recall Fact 1 saying that C's message does not leak any information about all values $b_{i,j}$'s to S. and thus all values in $(\mathbb{Z}_q)^{rm}$ are still equally likely for all values $b_{i,j}$'s even when conditioning over C's message. Then we can compute an upper bound on the probability that S's message w satisfies the i-th equation of the probabilistic test, as the ratio between the number of b_{ij} values for which w satisfies the test's equations, and the number of values of $b_{i,j}$ that are still equally likely even when conditioning over C's message. In this ratio, the numerator is at most q^{rm-1} (by Fact 2), and the denominator is q^{rm} (by Fact 1). Since there are $r = \lceil \lambda / \log q \rceil$ such equations in the probabilistic test, and all of these equations use random and independent $b_{i,j}$ values, the probability that w satisfies all r equations in the probabilistic test is

$$\leq \left(\frac{q^{rm-1}}{q^{rm}}\right)^r = \left(\frac{1}{q}\right)^r = \left(\frac{1}{q}\right)^{\lceil \lambda/\log q \rceil} \leq \frac{1}{2^\lambda}.$$

This proves Fact 3.