EFFICIENT IMPLEMENTATION OF LATTICE-BASED SCHEMES

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF APPLIED MATHEMATICS
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

YUSUF ALPER BILGIN

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF DOCTOR OF PHILOSOPHY
IN
CRYPTOGRAPHY

SEPTEMBER 2020

Approval of the thesis:

EFFICIENT IMPLEMENTATION OF LATTICE-BASED SCHEMES

submitted by **YUSUF ALPER BILGIN** in partial fulfillment of the requirements for the degree of **Doctor of Philosophy in Cryptography Department, Middle East Technical University** by,

Prof. Dr. Ömür Uğur
Director, Graduate School of **Applied Mathematics**

Prof. Dr. Ferruh Özbudak
Head of Department, **Cryptography**

Assoc. Prof. Dr. Murat Cenk
Supervisor, **Cryptography, METU**

Examining Committee Members:

Prof. Dr. Ferruh Özbudak
Mathematics, METU

Assoc. Prof. Dr. Murat Cenk
Cryptography, METU

Assoc. Prof. Dr. Ali Doğanaksoy
Mathematics, METU

Assoc. Prof. Dr. Fatih Sulak
Mathematics, Atılım University

Assist. Prof. Dr. Erdem Alkım
Computer Engineering, Ondokuz Mayıs University

Date:

ABSTRACT

EFFICIENT IMPLEMENTATION OF LATTICE-BASED SCHEMES

Bilgin, Yusuf Alper
Ph.D., Department of Cryptography
Supervisor : Assoc. Prof. Dr. Murat Cenk

September 2020, 72 pages

Quantum computing and quantum computers have been discussed for almost three decades. However, they remain mainly in theory. Almost all big companies like Google, IBM, and Microsoft have put their effort to build the most scalable quantum computers in recent years. These computers can change the game in cryptography since the known hard problems such as integer factorization and discrete logarithms can be broken with a large-scale quantum computer. These computers would seriously jeopardize the confidentiality and integrity of all digital communications. The question of when large-scale quantum computers will be built is still an open question. There are some educated predictions that sufficiently large quantum computers will be built to break all public-key schemes currently in use within the next ten or so years. That is why the National Institute of Standards and Technology (NIST) has announced a standardization process in 2016 to prepare our information security systems to be able to resist in quantum computing. The first two rounds of this process have been completed, and there are seven on-going candidates and eight alternate candidates. Among these 15 schemes, seven of them are based on lattices.

In this thesis, we study the efficient implementation of lattice-based schemes. We first propose an efficient and compact variant of NewHope, one of the most efficient second-round candidates of the NIST post-quantum standardization project. The proposed algorithm NewHope-Compact heavily uses recent advances on Number

Theoretic Transform (NTT), so that transformation from one polynomial to another is easy. To make it possible, we changed the definition of a component in component-wise multiplication during polynomial multiplication and show that changing the security level only requires to change the size of the polynomial and the definition of a component. Then, we present various optimizations for lattice-based KEMs using the NTT on the popular ARM Cortex-M4 microcontroller. Improvements come in the form of a faster code using more efficient modular reductions, optimized small-degree polynomial multiplications, and more aggressive layer merging in the NTT, but also in the form of reduced stack usage. We test our optimizations in software implementations of KYBER, one of the round three candidates in the NIST post-quantum project, NEWHOPE, and NEWHOPE-COMPACT.

Keywords: Post-quantum cryptography, lattice-based cryptography, key encapsulation mechanism, number theoretic transform, NTT, KEM, RLWE, MLWE, ARM Cortex-M4, KYBER, NEWHOPE, NEWHOPE-COMPACT

ÖZ

KAFES TABANLI ALGORİTMALARIN VERİMLİ GERÇEKLENMESİ

Bilgin, Yusuf Alper
Doktora, Kriptografi Bölümü
Tez Yöneticisi : Doç. Dr. Murat Cenk

Eylül 2020, 72 sayfa

Kuantum bilgisayarlar neredeyse otuz yıldır tartışılmasına rağmen bu konuda araştırmalar büyük ölçüde teoride kalmıştır. Son yıllarda, Google, IBM ve Microsoft gibi tanınmış şirketler büyük ölçekli bir kuantum bilgisayar yapabilmek için çaba sarf etmektedir. Bu bilgisayarlar ile çarpanlarına ayırma ve ayrık logaritmalar gibi zor bilinen problemler büyük ölçekli bir kuantum bilgisayar tarafından kırılabilecektir. Bu yüzden, dijital iletişimin gizliliği ve bütünlüğü ciddi biçimde tehlikeye girecektir. Büyük ölçekli kuantum bilgisayarların ne zaman yapılacağı sorusu hala açık bir sorudur. Önümüzdeki 10 yıl içinde, hâlihazırda kullanımda olan tüm açık anahtar algoritmalarını kırmak için yeterince büyük kuantum bilgisayarların inşa edileceği konusunda bazı tahminler vardır. Bu nedenle, NIST, bilgi güvenlik sistemlerimizi kuantum bilgisayarlarına karşı koruyabilmek için 2016 yılında bir standartlaştırma süreci başlatmıştır. Bu sürecin ilk iki turu tamamlanmıştır ve yedi tane aday algoritma, sekiz tane de alternatif algoritma seçilmiştir. Bu 15 algoritma içerisinden sekiz tanesi kafes tabanlıdır.

Bu tez kapsamında kafes tabanlı algoritmaların verimli bir şekilde gerçeklenmesi üzerine çalışılmıştır. İlk olarak, NIST kuantum sonrası standartlaştırma süreci ikinci tur adayı olan NEWHOPE algoritmasının hızlı ve kompakt bir varyasyonu olan NEWHOPE-COMPACT algoritması önerilmiştir. Önerilen bu algoritma sayı teorik dönüşümünde (NTT) olan son gelişmeleri yoğun bir şekilde kullanmaktadır. Bunun için elemanlar arası çarpmada kullanılan her bir elemanın tanımı değiştirilmiştir. Güvenlik seviyesini değiştirmek için sadece polinom boyutunu ve eleman tanımını değiştirmenin yeterli olduğu gösterilmiştir. Daha sonra, NTT'yi kullanan kafes tabanlı algoritmalar için ARM Cortex-M4 üzerinde çeşitli optimizasyonlar sunulmuştur. Bu optimizasyonlar ile daha verimili modüler indirgeme, optimize edilmiş küçük terimli polinom çarpımı ve daha agresif NTT katmanı birleşimi kullanılarak daha hızlı bir uygulama sunulmuştur. Gerçekleştirilen bu performans optimizasyonun yanında yığın bellek kullanımı da azaltılmıştır. Bu optimizasyonlar, NIST kuantum sonrası standartlaşma sürecinde üçüntü tur adayı olan KYBER, ikinci tur adayı olan ve üçüncü turda elenen NEWHOPE ve kendi önerdiğimiz NEWHOPE-COMPACT üzerinde test edilmiştir.

Anahtar Kelimeler: Kuantum sonrası kriptografi, kafes tabanlı kriptografi, anahtar kapsülleme mekanizması, sayı teorik dönüşüm

Per la mia bella principessa
To my mother and father

ACKNOWLEDGMENTS

I would first like to thank my supervisor Assoc. Dr. Murat Cenk for his valuable supervision during this work. He was always there when I asked for, spared a lot of time to listen to my ideas, and gave useful suggestions.

I am also grateful to work with Dr. Erdem Alkım. His vision showed me the way when I struggled. It is a great honor to work with him.

I thank TÜBİTAK for their one-year financial support with Graduate Scholarship no. 2211.

I would also like to thank my colleagues Sarp Mertol, İbrahim Arslan, and Batuhan Karataş for listening to my talks about this thesis even they tried to avoid it as much as possible.

I would like to express my deepest gratitude to my mother and my father for their invaluable support to me in my entire life. They always support me during my long educational life and believe in me.

Finally, I thank the most important person in my life, my fiancé Selin Oylan. None of this would have been possible without her. She always enlightens my way with her great shine. Looking at her beautiful eyes and listening to her calming voice always motivates me. I feel so lucky to have her in my life. I am curious and excited about our long journey together.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

AES	Advanced Encryption Standard
CRT	Chinese Remainder Theorem
DSA	Digital Signature Algorithm
DSP	Digital Signal Processing
ECC	Elliptic Curve Cryptography
FFT	Fast Fourier Transform
IND-CPA	Indistinguishability under chosen plaintext attack
IND-CPA	Indistinguishability under chosen ciphertext attack
KEM	Key encapsulation mechanism
LWE	Learning with errors
MLWE	Module-learning with errors
MLWE	Module-learning with rounding
NIST	National Institute of Standards and Technology
NTT	Number Theoretic Transform
PKE	Public key encryption
PQM4	Post-quantum crypto library for the ARM Cortex-M4
RLWE	Ring-learning with errors
RLWE	Ring-learning with rounding
RSA	Rivest–Shamir–Adleman
SIMD	Single Input Multiple Data

CHAPTER 1

INTRODUCTION

Cryptography is the study or technique of securing all sorts of communications from third party adversaries. The prefix "crypt" means hidden, and the postfix "graphy" means writing. Cryptography is almost as old as the invention of writing, and it has been in our life since then. The first known usage of cryptography dated back to the ancient Egyptians. They communicate through hieroglyphs, which are somewhat encrypted.

One of the most famous ancient cryptographic algorithms is Caeser Cipher. The Roman empire Julius Caeser, who did not trust his messengers, used it to communicate with his governers, officers, and generals. This cipher relies on shifting the letters of a message by three-position ahead. This method was performed by using a pen and paper. However, the technology is evolving, so does the cryptography. The invention of complex mechanical and electromechanical machines in the early 20^{th} century enables more advanced encryption. The first modern example of this improvement is the Enigma machine, which was used by Germans during World War II. It is followed by Bombe, the British cryptanalysis machine to decipher Enigma.

Cryptography is mostly used to ensure the secrecy of communications for military or politics up to the 70s. However, the groundbreaking discovery of Diffie and Hellman [18] in 1976 changes this by introducing public-key cryptography, which broadens the applications of cryptography. They solved the key distribution problem, which is one of the fundamental and most challenging ones. One year later, Rivest, Shamir, and Adleman introduced RSA [45] by solving the open problem left by [18], which was finding a suitable one-way function. After these works, cryptographic algorithms are built based on mathematical problems that are easy to state but hard to solve. Today, most of the public-key cryptosystems that we use rely on the difficulty of integer factorization and discrete logarithm.

Although the field of cryptography is mostly dominated by mathematicians and computer scientists, a physicist Richard Feynman changed this by publishing his game-changer work [19] in 1982. He founded the field of quantum computing and also the term quantum computers with this work. In digital computing, we have bits, which are 0 or 1. On the other hand, quantum computing works on qubits that are 0, 1, or in the state of quantum superposition [43,48]. In other words, a quantum computer with n qubits can be in 2^n different states at the same time, while a classical computer with the same number of bits can be one of 2^n different states at a time. Hence, we can utilize a quantum computer to solve large scale problems such as analysis of chemical interactions, speeding up searches of a large database, challenging optimizations problems, and cryptanalysis. Especially the last one attracts the attention of governments and big companies. We do not have large enough quantum computers yet to solve such large scale problems, but the field is snowballing. Different parties, such as government agencies, big companies like Google, Intel, IBM, Microsoft, Alibaba, and researches, are racing each other to build the quantum computer with more qubits and more power. Lately, Google announced that they reached the quantum supremacy in [11]. They stated that the Sycamore quantum processor takes 200 seconds to sample one instance of a quantum circuit. Their estimation shows that an equivalent task would take 10000 years for a classical supercomputer. However, [44] states that this computation takes only two and a half days instead of 10000 years.

One of the applications that we can take advantage of quantum computers is cryptanalysis. The security of the current public-key cryptosystems rests on the difficulty of integer factorization and discrete logarithm. [48] introduced efficient randomized algorithms for these two problems on a hypothetical quantum computer. By using these algorithms, one can break all currently used public-key cryptographic systems such as RSA, Digital Signature Algorithm (DSA) or Elliptic Curve Cryptography (ECC) on the existence of a large scale quantum computer. The situation is also similar to symmetric cryptography. The security of a symmetric algorithm such as the Advanced Encryption System (AES) will be halved on a logarithmic scale by using

Grover's algorithm [23] on a powerful enough quantum computer. We are not even close to such quantum computers, and some are still suspicious about establishing them. However, this does not mean that we are not able to build them. According to Moore's Law [41], the number of transistors in a dense integrated circuit doubles about every two years. If this rule is valid for quantum computers, we might have powerful enough quantum computers to decipher public-key schemes in about ten years. In 2015, Michele Mosca said [42] that "I estimate a 1/7 chance of breaking RSA-2048 by 2026 and a 1/2 chance by 2031". Therefore, a significant amount of work has been devoted to the replacement of currently used public-key cryptography with the algorithms that are resistant to attacks done by both classical and quantum computers, called post-quantum cryptography. There are five types of problem families. These are lattice-based, multivariate, hash-based, code-based, and supersingular elliptic curve isogeny-based cryptography. There are plenty of public-key proposals based on those systems. Some of them, such as lattice-based, have been studied for more than 20 years, but some of them, like supersingular elliptic curve isogeny-based, are relatively newer than the others.

The interest in post-quantum cryptography, i.e., cryptography resisting adversaries equipped with both classical and quantum computers, has grown significantly among the research community in the last few years. This growth is partially driven by the National Institute of Standards and Technology (NIST) post-quantum standardization project aiming to create a formal environment in which concrete instantiations of several post-quantum techniques for signatures and key encapsulation can be analyzed and compared to each other concerning several metrics. This standardization project was started in 2016 [20, 39]. The first round of the project took place mainly during 2018 and assessed different possible quantum-safe algorithms. As stated in [2, 20, 40], *"performance considerations will NOT play a major role in the early portion of the evaluation process."*. Therefore, the performance was not the primary consideration, but instead, NIST considered the security and cost as primary factors in its decision. In early 2019, 26 out of the 69 initial algorithms advanced to the second round of the project [2]. In this second round, as [38] stated that the candidates' practical performance played an essential role in the selection of a hypothetical future standardization. The second round of evaluation was recently completed, and

NIST announced the third round candidates [1], which contain four key encapsulation mechanisms (KEM) and three digital signature schemes. In addition, five KEMs and three digital signature schemes were also advanced to the next round as alternate candidate algorithms.

This thesis is organized as follows: Following this introduction chapter, Chapter 2 gives the notation and preliminaries information required to follow up with the upcoming chapters. Chapter 3 presents a compact, simple, and efficient KEM whose security relies on Ring-learning with errors (RLWE) problem. This scheme is a variant of NEWHOPE, one of the second round candidates of the NIST post-quantum standardization project, and named as NEWHOPE-COMPACT. Chapter 4 proposes various optimizations for lattice-based KEMs using the number theoretic transform (NTT) on the popular ARM Cortex-M4 microcontroller. Chapter 5 concludes the work.

CHAPTER 2

PRELIMINARIES

In this chapter, we provide the necessary background and notation for understanding the technical content of this thesis. We start with the notation. Then, we briefly describe KEM and how one can use NTT to speed up polynomial multiplications. We also describe a different NTT-based polynomial multiplication approach given by [36], and recently used by [4, 12, 35]. We end this chapter by giving algorithm descriptions for NewHope and Kyber, two post-quantum standardization candidates.

2.1 Notation

Let q be a prime number, n be a power of two. The quotient ring $\mathbb{Z}/q\mathbb{Z}$ is denoted as $\mathbb{Z}_q$. $\mathcal{R} = \mathbb{Z}[X]/(X^n + 1)$ be the ring of integer polynomials modulo $X^n + 1$. Then, we define $\mathcal{R}_q = \mathbb{Z}_q[X]/(X^n + 1)$ as a special case of $\mathcal{R}$ such that every coefficient is reduced modulo q. Note that n and q are selected in such a way that performing an efficient NTT of an element in $\mathcal{R}_q$ is possible. For that reason, n is selected as a power of two such that $X^n + 1$ is the $2n$-th cyclotomic polynomial, and q is selected as a prime allowing an efficient NTT implementation. We represent an element $a \in \mathcal{R}_q$ as $a = \sum_{i=0}^{n-1} a_i X^i$, where a_i is in $\mathbb{Z}_q$. Moreover, bold lower-case letters such as $\mathbf{v}$ denote a column vector, and bold upper-case letters such as $\mathbf{A}$ denote a matrix with entries in $\mathcal{R}_q$. The representations of polynomials, vectors, and matrices in the NTT domain are denoted as $\hat{a}$, $\hat{\mathbf{v}}$, and $\hat{\mathbf{A}}$, respectively.

2.2 Key Encapsulation Mechanism

A KEM is similar to the Diffie-Hellman key exchange scheme ([18]) in context. They are both used to generate a shared secret. However, while both parties contribute to the generation of the shared secret in the Diffie-Hellman scheme, only one party generates the secret and shares it with the other in a KEM. This generated key is encrypted before being shared. Hence, a public key encryption (PKE) scheme is needed. We define these two primitives below:

Definition 2.1. *A public key encryption scheme consists of three algorithms defined as follows:*

1. *The key generation (*KeyGen*) is a probabilistic algorithm that outputs a public/secret key pair (pk, sk).*
2. *The encryption (*Enc*) is a probabilistic algorithm that takes as input a message (m) and a public key (pk) and returns a ciphertext (c).*
3. *The decryption (*Dec*) is a deterministic algorithm that takes as input a ciphertext (c) and a secret key (sk) and returns a message (m) or failure.*

Definition 2.2. *A key encapsulation mechanism consists of three algorithms defined as follows:*

1. *The key generation (*KeyGen*) is a probabilistic algorithm that outputs a public/secret key pair (pk, sk).*
2. *The encapsulation (*Encaps*) is a probabilistic algorithm that takes as input a public key (pk) and returns a shared secret (ss) and a ciphertext (c).*
3. *The decapsulation (*Decaps*) is a deterministic algorithm that takes as input a ciphertext (c) and a secret key (sk) and returns a shared secret (ss$'$) or failure.*

The decapsulation algorithm does not always output the shared secret successfully but rather an incorrect shared secret or false for some constructions of key encapsulation mechanisms. This error case is referred to as decryption or decapsulation failures. A KEM is correct if all (pk, sk) $\leftarrow$KeyGen() and (c, ss) $\leftarrow$Encaps(pk), ss = ss$'$ for ss$'$ $\leftarrow$Decaps(c, sk). A decapsulation failure is the case where Decaps function returns failure.

The security of a KEM is usually defined with indistinguishability under chosen plaintext attack (IND-CPA) and chosen ciphertext attack (IND-CCA), which are defined below:

Definition 2.3. *IND-CPA is defined between a challenger and an adversary for a KEM as follows:*

1. *The challenger generates a key pair (pk, sk) by running* KeyGen.
2. *The challenger runs (c, ss$_0$)* $\leftarrow$Encaps, *samples another shared secret ss$_1$, selects a bit $b \in \{0, 1\}$ and publishes (pk, c, ss$_b$) to the adversary.*
3. *The adversary may call the polynomial number of* Encaps *operation.*
4. *The adversary finally selects a bit $b' \in \{0, 1\}$. If $b = b'$, the attack is successful.*

A KEM is IND-CPA secure when every polynomial-time adversary has only a negligible advantage over random guessing.

Definition 2.4. *IND-CCA is defined between a challenger and an adversary for a KEM as follows:*

1. *The challenger generates a key pair (pk, sk) by running* KeyGen.
2. *The challenger runs (c, ss$_0$)* $\leftarrow$Encaps, *samples another shared secret ss$_1$, selects a bit $b \in \{0, 1\}$ and publishes (pk, c, ss$_b$) to the adversary.*
3. *The adversary may call the polynomial number of* Encaps *or* Decaps *operation. However, it may not submit c to the* Decaps.
4. *The adversary finally selects a bit $b' \in \{0, 1\}$. If $b = b'$, the attack is successful.*

A KEM is IND-CCA secure when every polynomial-time adversary has only a negligible advantage over random guessing.

2.3 Utilizing NTT for Polynomial Multiplication

Let $a, b,$ and $c \in \mathcal{R}_q$. If the parameters n and q are selected such that $q \equiv 1 \mod 2n$, the multiplication of $c = a \cdot b \mod (X^n + 1)$ can be calculated efficiently by utilizing NTT. The reason for the condition $q \equiv 1 \mod 2n$ is that a primitive n-th root of unity ω exists. This multiplication can be written as $c = \mathsf{NTT}^{-1}(\mathsf{NTT}(a) \circ \mathsf{NTT}(b))$ where NTT^{-1} denotes inverse NTT, $\circ$ denotes coefficient-wise multiplication. NTT and NTT^{-1} formulae are written as follows:

$$\mathsf{NTT}(a) = \hat{a} = \sum_{i=0}^{n-1} \hat{a}_i X^i, \text{ where } \hat{a}_i = \sum_{j=0}^{n-1} a_j \omega^{ij} \mod q,$$

$$\mathsf{NTT}^{-1}(\hat{a}) = a = \sum_{i=0}^{n-1} a_i X^i, \text{ where } a_i = \left(n^{-1} \sum_{j=0}^{n-1} \hat{a}_j \omega^{-ij}\right) \mod q.$$

The multiplication of $c = a \cdot b$ includes 2 forward NTTs, one inverse NTT, and n coefficient-wise multiplications. There are two different approaches to calculate forward NTT and inverse NTT efficiently. These are the Fast Fourier Transform (FFT) Trick and the Twisted FFT Trick, which are given below and well explained in [27].

2.3.1 The FFT Trick

This trick leads to the Cooley-Tukey butterfly ([17]) in forward NTT and Gentleman-Sande butterfly ([22]) in inverse NTT.

Note that we are working on $\mathcal{R}_q$ and the parameters n and q are chosen as $q \equiv 1 \mod 2n$, so that a primitive n-th root of unity ω and its square root $\gamma = \sqrt{\omega} \mod q$ exists. The observation of $X^n + 1 = X^n - \gamma^n$ leads to the following Chinese Remainder Theorem (CRT) map:

$$\mathcal{R}_q = \mathbb{Z}_q[X] / (X^n - \gamma^n) \to \mathbb{Z}_q[X] / (X^{n/2} - \gamma^{n/2}) \times \mathbb{Z}_q[X] / (X^{n/2} + \gamma^{n/2}).$$

Let $f \in \mathcal{R}_q$. Then, in order to get the coefficients after the above CRT map, we need to calculate

$$g = f \mod (X^{n/2} - \gamma^{n/2}),$$
$$h = f \mod (X^{n/2} + \gamma^{n/2}).$$

Therefore, we have two separate polynomials $g \in \mathbb{Z}_q[X] / (X^{n/2} - \gamma^{n/2})$ and $h \in \mathbb{Z}_q[X] / (X^{n/2} + \gamma^{n/2})$ now. The calculations of any parameters of g and h are given below:

$$g_i = f_i + \gamma^{n/2} \cdot f_{i+n/2},$$
$$h_i = f_i - \gamma^{n/2} \cdot f_{i+n/2},$$

where $0 \leq i < n/2$. These operations are called as Cooley-Tukey butterfly. As can be seen, these butterflies take $n/2$ additions, $n/2$ subtractions, and $n/2$ multiplications by $\gamma^{n/2}$ in total.

In order to get the NTT representation of a ring of polynomial, the above CRT map should be fully split. The split of g is easy. However, in order to split h, the first trick should be applied again by observing that $X^{n/2} + \gamma^{n/2} = X^{n/2} - \gamma^{n+n/2}$. Finally, we get n coefficients the form

$$a_{2i} = c_i \mod (X - \gamma^{brv(n/2+i)}),$$
$$a_{2i+1} = c_i \mod (X + \gamma^{brv(n/2+i)}),$$

where c_i's are the polynomials calculated from the previous CRT map for $0 \leq i < n/2$ and brv is the bit-reversal operation that is defined as

$$brv(v) = \sum_{i=0}^{\log_2(n)-1} (((v >> i)\&1) << (\log_2(n) - 1 - i)).$$

Note that the CRT map splits $k = \log_2 n$ times in order to get NTT representation of a polynomial in $\mathcal{R}_q$. From now on, let us call each of these splits as level. Therefore, an n-point NTT has $k = \log_2 n$ levels, and it is called as k level NTT.

The inverse of this NTT transformation is computed by using Gentleman-Sande butterflies. Let us illustrate the final level.

$$f_i = (g_i + h_i)/2,$$
$$f_{i+n/2} = (g_i - h_i)/(2 \times \gamma^{n/2}),$$

where $0 \leq i < n/2$. As it can be seen, these butterflies take $n/2$ additions, $n/2$ subtractions, and $n/2$ multiplications as the Cooley-Tukey butterflies. Note that dividing by two can be postponed to the end instead of doing it on each butterfly. This yields a multiplication with every coefficient by n^{-1} at the end. In fact, to reduce the number of multiplications at the end, half of these multiplications can be performed together with the multiplications by $\gamma^{-n/2}$ at the last level.

2.3.2 The Twisted FFT Trick

This trick leads to Gentleman-Sande butterfly in both forward and inverse NTTs.

There exist a primitive n-th root of unity ω and its square root $\gamma = \sqrt{\omega} \mod q$ as in the FFT trick. The twisted FFT trick starts with the following isomorphism

$$X \to \gamma X : R_q = \mathbb{Z}_q[X]/(X^n + 1) \to \mathbb{Z}_q[X]/(X^n - 1).$$

Then, the CRT map at the first level can be computed as

$$\mathbb{Z}_q[X]/(X^n - 1) \to \mathbb{Z}_q[X]/(X^{n/2} - 1) \times \mathbb{Z}_q[X]/(X^{n/2} + 1).$$

Let $f \in \mathcal{R}_q$ be the polynomial after the isomorphism of $X \to \gamma X$. Then, in order to get the coefficients after the first level, we need to calculate

$$\begin{aligned} g &= f \mod (X^{n/2} - 1), \\ h &= f \mod (X^{n/2} + 1). \end{aligned}$$

Therefore, we have two separate polynomials $g \in \mathbb{Z}_q[X]/(X^{n/2} - 1)$ and $h \in \mathbb{Z}_q[X]/(X^{n/2} + 1)$ now. The calculations of any parameters of g and h are given below.

$$\begin{aligned} g_i &= f_i + f_{i+n/2}, \\ h_i &= (f_i - f_{i+n/2}) \times \gamma^0, \end{aligned}$$

where $0 \leq i < n/2$. These operations are called as Gentleman-Sande butterfly. As it can be seen, these butterflies take $n/2$ additions, $n/2$ subtractions, and $n/2$ multiplications by γ^0. After applying these butterflies iteratively for k levels, we can get the representation of our polynomial in the NTT domain. During these computations, the corresponding bit-reversed power of γ is used. For the first level, we only need one γ value for our butterflies which is $\gamma^{brv[0]} = \gamma^0$ where brv corresponds to bit-reversal. At the i^{th} level, we have 2^i polynomials requiring 2^{i-1} γ values, which are $\gamma^{brv[j]}$

where $0 \leq j < 2^{i-1}$.

The inverse NTT can be computed with the same algorithm by following a little trick. Gentleman-Sande butterflies accept inputs in bit-reversed order, and it gives the result in standard order. Therefore, by applying the bit-reversal permutation before forward and inverse NTTs, we can utilize the same algorithm for both operations. Note that the powers of γ will be different. The inverse of the first isomorphism should also be taken at the end by multiplying each coefficient with the corresponding power of γ^{-1}. These multiplications can be hidden by multiplications with n^{-1}.

As mentioned before, the FFT Trick leads to Cooley-Tukey butterfly in forward NTT and Gentleman-Sande butterfly in NTT^{-1}. On the other hand, the Twisted FFT Trick leads to Gentleman-Sande butterfly in both NTT and NTT^{-1}. This work focuses on the former choice due to the use of signed integers, which leads to faster implementation. The pseudocode for NTT based on the Cooley-Tukey butterfly and NTT^{-1} based on the Gentleman-Sande butterfly are given in Algorithm 1 and Algorithm 2, respectively.

2.3.3 A Different Polynomial Multiplication Approach

The standard polynomial multiplication utilizing NTT is $c = \mathsf{NTT}^{-1}(\mathsf{NTT}(a) \circ \mathsf{NTT}(b))$ where $a, b,$ and $c \in \mathcal{R}_q$. This NTTs contain $k = \log_2 n$ levels where $n-1$ is the degree of polynomials. Although this polynomial multiplication is very efficient, it is interesting to see that if one stops at the $(k-1)$-th level or even the $(k-2)$-th or $(k-3)$-th levels instead of applying all k levels, the multiplication of two polynomials might be faster. This is an important observation firstly made by [36].

KYBER: The round 2 submission of the KYBER scheme includes an update in the definition of NTT. This update corresponds to stop NTT at 7-th level instead of applying a full NTT, which is 8-level. Algorithm 1 and Algorithm 2 can still be used with only minor modifications. The modification in Algorithm 1 is that $(\ell \geq 1)$ condition at line 3 is replaced with $(\ell \geq 2)$ while in Algorithm 2 $(\ell \leftarrow 1)$ at line 3 is replaced with $(\ell \leftarrow 2)$, and n at lines 14 and 15 is replaced with $n/2$. These changes also affect the coefficient-wise multiplications while multiplying two polynomials in

Algorithm 1 NTT based on the Cooley-Tukey butterfly

Input: A polynomial $a \in \mathcal{R}_q$ where i-th coefficient of a is denoted as $a[i]$, $0 \leq i < n$, and a precomputed table of Γ which stores $\Gamma[i] = \gamma^{brv(i)}$ where $brv(i) = \sum_{j=0}^{\log_2(n)-1}(((i >> j)\&1) << (\log_2(n) - 1 - i)), 0 \leq i < n$.

Output: $\hat{a} \leftarrow \mathsf{NTT}(a)$ in bit-reversed ordering.

```
1:  function NTT($a$)
2:      $i \leftarrow 1$
3:      for $\ell \leftarrow n/2; \ell \geq 1; \ell \leftarrow \ell/2$ do
4:          for $s \leftarrow 0; s < n; s \leftarrow s + \ell$ do
5:              $g \leftarrow \Gamma[i]$
6:              for $j \leftarrow s; j < s + \ell; j \leftarrow j + 1$ do
7:                  $t \leftarrow g \cdot a[j + \ell] \pmod q$
8:                  $a[j + \ell] \leftarrow a[j] - t \pmod q$
9:                  $a[j] \leftarrow a[j] + t \pmod q$
10:             end for
11:             $i \leftarrow i + 1$
12:         end for
13:     end for
14:     $\hat{a} \leftarrow a$
15:     return $\hat{a}$
16: end function
```

Algorithm 2 NTT^{-1} based on the Gentleman-Sande butterfly

Input: A polynomial $\hat{c} \in \mathcal{R}_q$ in bit-reversed ordering where i-th coefficient of $\hat{c}$ is denoted as $\hat{c}[i]$, $0 \leq i < n$, and a precomputed table of Γ^{-1} which stores $\Gamma^{-1}[i] = \gamma^{-(brv(i)+1)}$ where $brv(i) = \sum_{l=0}^{\log_2(n)-1}(((i >> l)\&1) << (\log_2(n) - 1 - i))$, $0 \leq i < n$.

Output: $c \leftarrow \mathsf{NTT}^{-1}(\hat{c})$ in standard ordering.

```
1: function NTT^{-1}($\hat{c}$)
2:     $i \leftarrow 0$
3:     for $\ell \leftarrow 1; \ell < n; \ell \leftarrow 2 \cdot \ell$ do
4:         for $s \leftarrow 0; s < n; s \leftarrow j + \ell$ do
5:             $g \leftarrow \Gamma^{-1}[i]$
6:             for $j \leftarrow s; j < s + \ell; j \leftarrow j + 1$ do
7:                 $t \leftarrow \hat{c}[j]$
8:                 $\hat{c}[j] \leftarrow t + \hat{c}[j + \ell] \pmod q$
9:                 $\hat{c}[j + \ell] \leftarrow g \cdot (t - \hat{c}[j + \ell]) \pmod q$
10:            end for
11:            $i \leftarrow i + 1$
12:        end for
13:    end for
14:    for $j \leftarrow 0; j < n; j \leftarrow j + 1$ do
15:        $c[j] \leftarrow \hat{c}[j]/n$
16:    end for
17:    return $c$
18: end function
```

$\mathcal{R}_q$ such that they are performed on small polynomials in $\mathbb{Z}_q/(X^2 - r)$ instead of integer coefficients. The schoolbook multiplication method to multiply two elements in $\mathbb{Z}_q/(X^2 - r)$ is given in Algorithm 3.

Algorithm 3 Multiplication of polynomials in $\mathbb{Z}_q/(X^2 - r)$

Input: a and $b \in \mathbb{Z}_q/(X^2 - r)$ where r is a power of γ. i-th coefficient of a is denoted as $a[i]$ where $0 \leq i < 2$.

Output: $c \in \mathbb{Z}_q/(X^2 - r)$.

```
1: function basemul($a$, $b$)
2:     $c[0] \leftarrow a[1] \cdot b[1]$
3:     $c[0] \leftarrow c[0] \cdot r$                    ▷ · for modular reduction
4:     $c[0] \leftarrow c[0] + (a[0] \cdot b[0])$        ▷ + for modular reduction
5:     $c[1] \leftarrow a[0] \cdot b[1]$
6:     $c[1] \leftarrow c[1] + (a[1] \cdot b[0])$
7:     return $c$
8: end function
```

A NTT-based polynomial multiplication consists of two forward NTT and one inverse NTT. By using the new approach, one can save three levels of NTT in total, two forward, and one inverse. The total cost of each NTT level is $n/2$ additions, $n/2$ subtractions, and $n/2$ multiplications. Therefore, the total saving is $3n/2$ additions, $3n/2$ subtractions, and $3n/2$ multiplications. However, apart from NTT calculations, there is also coefficient-wise multiplications or polynomial multiplications according to the approach used. The standard approach has n coefficient-wise multiplications, while the new approach has polynomial multiplications which contain $n/2$ additions, $n/2$ subtractions, and $5n/2$ multiplications. Although the multiplication counts are the same in total ($5n/2$ for both), the new approach has n additions, and n subtractions less than the standard approach. Besides this advantage, it has a more important advantage hidden in the definition of NTT. In order to have a ring allowing fast multiplication using NTT, schemes based on the RLWE problem usually select n as a power of two such that $n = 2^k$. In order to have a primitive n-th root of unity, the prime q is selected, such that $q \equiv 1 \mod 2n$. Therefore, it can be fully split, and k level NTT can be calculated efficiently. However, the new approach does not require k level NTT. Instead, $k - 1$ level is enough for polynomial multiplication in $\mathcal{R}_q$.

Therefore, $n/2$-th root of unity is sufficient for the calculation, and q can be selected, such that $q \equiv 1 \mod n$. If one decides to stop at another level ℓ instead of $k-1$, then the selection of q should satisfy $q \equiv 1 \mod (n/2^{k-\ell-1})$. The degrees of the resulting polynomials after NTT will be $2^{k-\ell}$. Therefore, these polynomials will be represented in $\mathbb{Z}_q[X]/(X^{2^{k-\ell}} \pm \gamma^{brv[i]+1})$ where brv is defined as below:

$$brv(v) = \sum_{i=0}^{\ell-1}(((v >> i)\&1) << (\ell - 1 - i)).$$

Then, the polynomial multiplication is performed in $\mathbb{Z}_q[X]/(X^{2^{k-\ell}} \pm \gamma^{brv[i]+1})$. Finally, by taking ℓ level NTT^{-1}, one can get the result of the multiplication of two polynomials in $\mathcal{R}_q$.

Two recent studies use this polynomial multiplication approach apart from KYBER [12]. One of the similar approaches is [51]. It first represents the input degree n polynomial as two degree $n/2$ polynomials. Then, it applies $k-1$ level NTT to both of them. The multiplication of two polynomials in their NTT domain representation is similar to the method described above. Finally, the inverse NTTs of two degree $n/2$ polynomials are taken separately. Although the approach is similar, their results are not as fast as KYBER. Their performance is even slower than the standard NTT-based multiplication approach. Another scheme that uses a similar approach is [35]. Their ring structure, $\mathbb{Z}_{7681}[X]/(X^{768} - X^{384} + 1)$, is different. It is observed that this ring can be split up to $X^3 \pm r$. Therefore, 8 level NTT can be applied to the polynomials in this ring. Then, the multiplication can be performed in $\mathbb{Z}_{7681}[X]/(X^3 \pm r)$. Finally, by taking 8 level inverse NTT, one can get the result. Note that all of these approaches are given by [36] as "The Mixed Basis FFT Multiplication Algorithm" for $n = 2^k \cdot \ell$, where the cases $\ell = 1, 3, 5,$ or 7 are considered. However, if ℓ is chosen a small multiple of 2, it will be faster than the standard NTT-based multiplication approach according to their complexity analysis.

2.4 NEWHOPE

NEWHOPE [3, 7] was one of the NIST post-quantum standardization candidates whose security is based on the hardness of solving the RLWE problem [33, 34]. This

Algorithm 4 NEWHOPE-CPA-PKE key generation

```
Output: public key $pk = (\hat{b}', \rho)$
Output: secret key $sk = \hat{s}$
1: $seed \xleftarrow{\$} \{0, \cdots, 255\}^{32}$
2: $\rho, \sigma \leftarrow \mathsf{SHAKE256}(64, seed)$
3: $\hat{a} \leftarrow \mathsf{GenA}(\rho)$
4: $s \leftarrow \mathsf{Sample}(\sigma, 0)$
5: $e \leftarrow \mathsf{Sample}(\sigma, 1)$
6: $\hat{b} \leftarrow \hat{a} \circ \mathsf{NTT}(s) + \mathsf{NTT}(e)$
7: return $pk = (\hat{b}, \rho), sk = \hat{s}$
```

Algorithm 5 NEWHOPE-CPA-PKE encryption

```
Input: public key $pk = (\hat{b}, \rho)$
Input: message $\mu$ encoded in $\mathcal{R}_q$
Input: seed $coin \in \{0, \cdots, 255\}^{32}$
Output: ciphertext $(\hat{u}', h)$
1: $\hat{a} \leftarrow \mathsf{GenA}(\rho)$
2: $s' \leftarrow \mathsf{Sample}(coin, 0)$
3: $e' \leftarrow \mathsf{Sample}(coin, 1)$
4: $e'' \leftarrow \mathsf{Sample}(coin, 2)$
5: $\hat{t} \leftarrow \mathsf{NTT}(s')$
6: $\hat{u} \leftarrow \hat{a} \circ \hat{t} + \mathsf{NTT}(e')$
7: $v' \leftarrow \mathsf{NTT}^{-1}(\hat{b} \circ \hat{t}) + e'' + \mu$
8: return $c = (\hat{u}, \mathsf{Compress}(v'))$
```

Algorithm 6 NEWHOPE-CPA-PKE decryption

```
Input: ciphertext $c = (\hat{u}, h)$
Input: secret key $sk = \hat{s}$
Output: message $\mu \in \{0, \cdots, 255\}^{32}$
1: $v' \leftarrow \mathsf{Decompress}(h)$
2: return $\mu = \mathsf{Decode}(v' - \mathsf{NTT}^{-1}(\hat{u} \circ \hat{s}))$
```

cryptosystem includes both an adaptive CPA-secure KEM, referred to as NEWHOPE-CPA-KEM, and an adaptive CCA-secure KEM, referred to as NEWHOPE-CCA-KEM. Both versions are based on the previously proposed NEWHOPE-SIMPLE [6] scheme, which was designed as a semantically secure PKE scheme and referred to as NEWHOPE-CPA-PKE. Key generation, encryption, and decryption functions of NEWHOPE-CPA-PKE are presented in Algorithm 4, Algorithm 5, and Algorithm 6. The constructions of NEWHOPE-CPA-KEM and NEWHOPE-CCA-KEM using NEWHOPE-CPA-PKE are out of the scope of this work; we refer to [3, Alg. 16-21] for more information.

Typically, the most time-consuming part of all Learning With Errors (LWE) variant cryptosystems is the hashing used for the randomness generation. This randomness is required inside the GenA and the Sample functions. Moreover, both CPA and CCA secure versions of KEM constructions also require hashing. Note that the encode, decode, compress, and decompress functions perform bit/byte-level manipulation, thus

Table 2.1: Parameters of NEWHOPE512 and NEWHOPE1024 and derived high-level properties [3].

Parameter Set	NEWHOPE512	NEWHOPE1024
Dimension n	512	1024
Modulus q	12289	12289
Noise Parameter k	8	8
NTT parameter γ	10968	7
Decryption error probability	2^{-213}	2^{-216}
Claimed post-quantum bit security	101	233
NIST Security Strength Category	1	5

making them relatively inexpensive. Apart from hashing, the main cost of NEWHOPE is multiplication in $\mathcal{R}_q$. NEWHOPE selects its parameters n and q to enable fast polynomial multiplication in $\mathcal{R}_q$ by utilizing the NTT. The parameter sets are provided in Table 2.1. The modulus q is selected such that $q \equiv 1 \pmod{2n}$, ensuring the existence of the n-th root of unity ω and the $2n$-th root of unity $\gamma = \sqrt{\omega}$, which is a prerequisite for the NTT.

Polynomial multiplication utilizing the NTT: A forward NTT is performed to transform all of the coefficients to the NTT domain, and the inverse of this operation, NTT^{-1}, is performed to carry all coefficients to the normal domain again. The formulae for these two operations are given as follows:

$$\mathsf{NTT}(a) = \hat{a} = \sum_{i=0}^{n-1} \hat{a}_i X^i, \text{ where } \hat{a}_i = \sum_{j=0}^{n-1} a_j \omega^{ij} \mod q,$$
$$\mathsf{NTT}^{-1}(\hat{a}) = a = \sum_{i=0}^{n-1} a_i X^i, \text{ where } a_i = \big(n^{-1} \sum_{j=0}^{n-1} \hat{a}_j \omega^{-ij}\big) \mod q.$$

The multiplication of $a, b \in \mathcal{R}_q$ can be computed as $ab = \mathsf{NTT}^{-1}(\mathsf{NTT}(a) \circ \mathsf{NTT}(b))$, where $\circ$ denotes the coefficient-wise multiplication. To perform the NTT and the NTT^{-1} operations faster, the KYBER and NEWHOPE reference implementations used two different approaches. KYBER makes use of the Cooley-Tukey butterfly [17] in the NTT and the Gentleman-Sande butterfly [22] in the NTT^{-1}, while NEWHOPE utilizes the Gentleman-Sande butterfly in both cases. The main reason that NEWHOPE selects the second one is that it allows more aggressive lazy reductions when unsigned

Algorithm 7 KYBER.CPAPKE key generation

```
Output: public key $pk = (\hat{\mathbf{b}}, \rho)$
Output: secret key $sk = \hat{\mathbf{s}}$
1: $seed \xleftarrow{\$} \{0, \cdots, 255\}^{32}$
2: $\rho, \sigma \leftarrow \mathsf{SHAKE256}(64, seed)$
3: $\hat{\mathbf{A}} \leftarrow \mathsf{GenMatrixA}(\rho)$
4: $\mathbf{s} \leftarrow \mathsf{SampleVec}(\sigma, 0)$
5: $\mathbf{e} \leftarrow \mathsf{SampleVec}(\sigma, 1)$
6: $\hat{\mathbf{b}} \leftarrow \hat{\mathbf{A}} \circ \mathsf{NTT}(\mathbf{s}) + \mathsf{NTT}(\mathbf{e})$
7: return $pk = (\hat{\mathbf{b}}, \rho), sk = \hat{\mathbf{s}}$
```

Algorithm 8 KYBER.CPAPKE encryption

```
Input: public key $pk = (\hat{\mathbf{b}}, \rho)$
Input: message $\mu \in \mathcal{R}_q$
Input: seed $coin \in \{0, \cdots, 255\}^{32}$
Output: ciphertext $(\mathbf{u}', h)$
1: $\hat{\mathbf{A}} \leftarrow \mathsf{GenMatrixA}(\rho)$
2: $\mathbf{s}' \leftarrow \mathsf{SampleVec}(coin, 0)$
3: $\mathbf{e}' \leftarrow \mathsf{SampleVec}(coin, 1)$
4: $e'' \leftarrow \mathsf{SampleVec}(coin, 2)$
5: $\hat{\mathbf{t}} \leftarrow \mathsf{NTT}(\mathbf{s}')$
6: $\mathbf{u} \leftarrow \mathsf{NTT}^{-1}(\hat{\mathbf{A}}^T \circ \hat{\mathbf{t}}) + \mathbf{e}'$
7: $v' \leftarrow \mathsf{NTT}^{-1}((\hat{\mathbf{b}}^T \circ \hat{\mathbf{t}}) + e'' + \mu$
8: return $(\mathsf{Compress}(\mathbf{u}), \mathsf{Compress}(v'))$
```

Algorithm 9 KYBER.CPAPKE decryption

```
Input: ciphertext $c = (\mathbf{u}', h)$
Input: secret key $sk = \hat{\mathbf{s}}$
Output: message $\mu \in \mathcal{R}_q$
1: $\mathbf{u} \leftarrow \mathsf{Decompress}(\mathbf{u}')$
2: $v' \leftarrow \mathsf{Decompress}(h)$
3: return $\mu = v' - \mathsf{NTT}^{-1}(\hat{\mathbf{s}}^T \circ \mathsf{NTT}(\mathbf{u}))$
```

integers are used in the implementation. However, the NEWHOPE reference implementation requires both input and output of the NTT^{-1} to be in the normal order, hence requiring an extra bitreversal.

2.5 KYBER

KYBER [12, 14] is a post-quantum KEM whose security relies on the hardness of the Module Learning With Errors (MLWE) problem [31]. KYBER constructs a CCA-secure KEM KYBER.CCAKEM by using a CPA-secure PKE, which is referred to as KYBER.CPAPKE, using a variant of the Fujisaki-Okamoto transform [21]. We refer to [12, Alg. 7-9] for a description of KYBER.CCAKEM. The key generation, encryption, and decryption functions of KYBER.CPAPKE are presented in Algorithm 7, Algorithm 8, and Algorithm 9, respectively.

Similar to other LWE-based systems, such as NEWHOPE, the most time-consuming part of KYBER is the hashing used for randomness generation. Aside from that, the most costly operation is the multiplication in $\mathcal{R}_q$. KYBER also utilizes the NTT to speed up this operation. The modulus q and the dimension n are selected as $q = 3329$ and $n = 256$ for all parameter sets of KYBER. This enables a 7-level NTT with the parameter $\gamma = 17$. After performing the 7-level NTT, there are 128 degree-one polynomials. Therefore, coefficient-wise multiplications are performed on these degree-one polynomials modulo $(X^2 - r)$, where r is a power of γ, by using the schoolbook method.

CHAPTER 3

COMPACT AND SIMPLE RLWE KEY EXCHANGE MECHANISM

This chapter is based on the publication "Compact and Simple RLWE Key Exchange Mechanism" [4].

This chapter starts with an introduction. Section 3.2 discusses the complexity of the revisited NTT-based polynomial multiplication algorithm that is given in Section 2.3.3. Then, Section 3.3 provides our implementation details and parameter sets for the proposed KEM, a variant of NEWHOPE and named as NEWHOPE-COMPACT. Finally, our performance results for NEWHOPE-COMPACT and a comparison with reference (non-optimized) implementations of NEWHOPE and KYBER are presented in Section 3.4.

3.1 Introduction

Among all second round candidates, 8 out of 17 KEMs and 3 out of 9 digital signature schemes are based on structured lattices. Therefore, their performances depend heavily on polynomial arithmetic in a ring of integer polynomials modulo an irreducible polynomial over the rationals denoted as $\mathbb{Z}_q/(f(x))$. In other words, polynomial arithmetic is one of the most important parts of structured lattice-based post-quantum safe algorithms since the main mathematical elements of such algorithms are polynomials. The most complex and time-consuming polynomial arithmetic is polynomial multiplication for such schemes. Although standard polynomial multiplication is very trivial, it consumes a lot of CPU cycles due to their quadratic complexity. There are many efficient algorithms to handle polynomial multiplication efficiently such as NTT, Karatsuba or Tom-Cook multiplications. NTT, which is a special case of FFT over finite fields, has been shown to be a powerful algorithm in order to perform the polynomial multiplication over finite fields for some type of lattice-based cryptography [3, 12, 24, 32, 47]. NTT transform does not require any extra memory space, and it is highly vectorizable. That is why some of the lattice-based cryptographic primitives prefer to use it. Another reason for choosing NTT multiplication over other multiplication methods is that the random polynomials can be directly sampled in the NTT domain so that one can save the cost of some forward NTTs. NEWHOPE and KYBER are two important NIST post-quantum standardization project candidates utilizing NTT to handle polynomial arithmetic efficiently.

Implementing lattice-based schemes by using Single Input Multiple Data (SIMD) instructions are quite popular since NTT is easily vectorizable in this setting. NEWHOPE is one of the fastest NTT implementations by using floating-point instructions. However, these instructions work on 64 bit double values, although the coefficients of NEWHOPE are only 14 bits. In [47], Seiler introduced the use of integer instructions with a modification of the original Montgomery reduction algorithm [37]. The integer instructions can work on 16-bit values. This is much more efficient than the floating-point instructions because one 256-bit AVX2 register can hold 16 coefficients instead of just four coefficients. This approach can significantly speed up AVX2 implementation of NEWHOPE. Moreover, the latest results of [12,35,51] show that with a little modification in NTT transformation during polynomial multiplication, one can reduce the size of the prime. This modification is to not carry out the full NTT, but rather stop it before reaching the level of base field arithmetic. Hence, the congruence condition on the prime modulus is relaxed, and a smaller prime is possible. This also means that the coefficient-wise multiplications in NTT domain are done on polynomials instead of integer coefficients. These polynomial multiplications are performed with schoolbook or Karatsuba methods.

Availability of the software: All of the software described in this chapter available online at `https://github.com/erdemalkim/NewHopeCompact` and `https://github.com/alperbilgin/NewHopeCompact`.

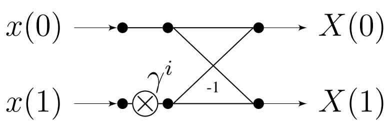


Figure 3.1: Cooley-Tukey Butterfly

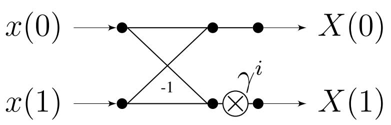


Figure 3.2: Gentleman-Sande Butterfly

3.2 A Complexity Analysis of Polynomial Multiplication Methods

3.2.1 Standard Multiplication Utilizing NTT

n-term polynomial multiplication utilizing NTT consists of two forward NTT, one inverse NTT, and n coefficient-wise multiplications. Although this is not always the case, we assume that Cooley-Tukey, Figure 3.1, and Gentleman Sande, Figure 3.2, butterflies are used for forward and inverse NTT, respectively.

Let us start with the computation cost of the Cooley-Tukey butterfly. As can be seen in Figure 3.1, a 2-term butterfly costs one multiplication, one addition, and one subtraction. Let's calculate subtraction as addition for simplicity, since $a - b$ can be written as $a + (-b)$. Therefore, the cost of 2-term 1-level NTT is one multiplication and two additions. The cost of 4-term 2-level NTT is two times of 2-term 1-level NTT, two multiplications, and four additions.

Proposition 3.1. *The cost of n-term NTT can be computed with the following recurrence relation:*

$$T(n) = 2 \cdot T\left(\frac{n}{2}\right) + \frac{n}{2} \cdot M + n \cdot A,$$

where $T(1) = 0$, M and A stand for multiplication and addition and n is the number of coefficients, which is a power of two.

Proof. Let $n = 2^k$. The computation cost of each NTT level is $n/2$ multiplications and n additions. If we divide this polynomial as two $n/2$-term polynomials, $k-1$-level NTT is possible and each NTT levels of these polynomials costs $n/4$ multiplications and $n/2$ additions since there are $n/2$ terms. Therefore, the sum of the computation costs for each NTT levels of two $n/2$-term polynomials is equal to the cost of each NTT levels of n-term polynomial. Consequently, k-level NTT is constituted of two $(n/2)$-term $(k-1)$-level NTTs plus the last level which includes $n/2$

multiplications and n additions like all other levels. $\square$

The recursion given in Proposition 3.1 yields

$$T(n) = \frac{k}{2} \cdot n \cdot M + k \cdot n \cdot A,$$

where $k = \log_2 n$.

The cost of the Gentleman-Sande butterfly is equal to the cost of the Cooley-Tukey butterfly since the same number of operations are required for both of them, as it can be seen from Figure 3.1 and Figure 3.2. Therefore, the total computation cost of multiplying two n-term polynomials in $\mathcal{R}_q$ is

$$\begin{aligned} PM(n) &= 3 \cdot T(n) + n \cdot M, \\ &= \frac{3k+2}{2} \cdot n \cdot M + 3k \cdot n \cdot A. \end{aligned}$$

3.2.2 An Analysis of $k-1$ Level NTT then Schoolbook Multiplication

This method is described in Section 2.3.3. It requires two forward and one inverse NTTs as in the previous method. However, this time NTTs have $k-1$ levels. Note that $2 \cdot T(n/2)$ operations are already performed up to the last level. This method stops there and leaves the calculations required for the last level, which are $n/2 \cdot M + n \cdot A$. Therefore, we need to use $2 \cdot T(n/2)$ instead of $T(n)$ in our calculations for the cost of NTT.

Proposition 3.2. *The multiplication of two polynomials in modulo $\mathbb{Z}_q[X]/(X^2 - r)$ costs five multiplications and two additions including modular reduction in modulo $(X^2 - r)$.*

Proof. This multiplication algorithm is given in Algorithm 18. One multiplies two polynomials in modulo $\mathbb{Z}_q[X]/(X^2 - r)$ with $(2^2 = 4)$ multiplications and one addition (computed with the formula given by [26]). The modular reduction also needs one more multiplication with r and one addition. Hence, in total, it costs five multiplications and two additions. $\square$

The result of the multiplication of two small polynomials in modulo $\mathbb{Z}_q[X]/(X^2 - r)$ gives us two coefficients, as can be seen in Algorithm 18. Therefore, the total cost can be written as

$$\begin{aligned} Hybrid\text{-}SM_1(n) &= 3 \cdot 2 \cdot T\big(\frac{n}{2}\big) + \frac{1}{2}\big(5n \cdot M + 2n \cdot A\big), \\ &= \frac{3k-3}{2} \cdot n \cdot M + 3 \cdot n \cdot (k-1) \cdot A + \frac{5n}{2} \cdot M + n \cdot A, \\ &= \frac{3k+2}{2} \cdot n \cdot M + (3k-2) \cdot n \cdot A. \end{aligned}$$

3.2.3 An Analysis of $k-2$ Level NTT then Schoolbook Multiplication

This method stops at the $k-2$ level of the NTT. Then, we have small polynomials in modulo $(X^4 - r)$. The schoolbook method is used for the multiplication of these small polynomials.

Proposition 3.3. *To multiply two polynomials in modulo $\mathbb{Z}_q[X]/(X^4 - r)$ using schoolbook multiplication, we need to perform 19 multiplications and 12 additions, including modular reductions in modulo $(X^4 - r)$.*

Proof. This multiplication algorithm is given in Algorithm 10. The multiplication of two 4 coefficients polynomial requires $4^2 = 16$ multiplications and $3^2 = 9$ additions. Moreover, the modular reduction in modulo $(X^4 - r)$ costs three multiplications and three additions as can be seen from Algorithm 10. Therefore, the multiplication of two small polynomials in modulo $(X^4 - r)$ requires 19 multiplications and 12 additions. □

After the multiplication in modulo $(X^4 - r)$, we end up with four coefficients. Therefore, the total cost can be written as

$$\begin{aligned} Hybrid\text{-}SM_2(n) &= 3 \cdot 4 \cdot T\big(\frac{n}{4}\big) + \frac{1}{4}\big(19n \cdot M + 12n \cdot A\big), \\ &= \frac{3k-6}{2} \cdot n \cdot M + 3 \cdot n \cdot (k-2) \cdot A + \frac{19n}{4} \cdot M + 3 \cdot n \cdot A, \\ &= \frac{6k+7}{4} \cdot n \cdot M + (3k-3) \cdot n \cdot A. \end{aligned}$$

Algorithm 10 Multiplication of polynomials in $\mathbb{Z}_q/(X^4 - r)$

Input: a and $b \in \mathbb{Z}_q/(X^4 - r)$ where r is a power of γ. i-th coefficient of a is denoted as $a[i]$ where $0 \leq i < 4$.

Output: $c \in \mathbb{Z}_q/(X^4 - r)$.

```
1:  function basemul($a$, $b$)
2:      $c[0] \leftarrow (a[1] \cdot b[3]) + (a[2] \cdot b[2]) + (a[3] \cdot b[1])$
3:      $c[0] \leftarrow c[0] \bullet r$                                   ▷ $\bullet$ for modular reduction
4:      $c[0] \leftarrow c[0] \boldsymbol{+} (a[0] \cdot b[0])$                      ▷ $\boldsymbol{+}$ for modular reduction
5:      $c[1] \leftarrow (a[2] \cdot b[3]) + (a[3] \cdot b[2])$
6:      $c[1] \leftarrow c[1] \bullet r$                                   ▷ $\bullet$ for modular reduction
7:      $c[1] \leftarrow c[1] \boldsymbol{+} (a[0] \cdot b[1]) + (a[1] \cdot b[0])$        ▷ $\boldsymbol{+}$ for modular reduction
8:      $c[2] \leftarrow a[3] \cdot b[3]$
9:      $c[2] \leftarrow c[2] \bullet r$                                   ▷ $\bullet$ for modular reduction
10:     $c[2] \leftarrow c[2] \boldsymbol{+} (a[0] \cdot b[2]) + (a[1] \cdot b[1]) + (a[2] \cdot b[0])$   ▷ $\boldsymbol{+}$ for modular
    reduction
11:     $c[3] \leftarrow (a[0] \cdot b[3]) + (a[1] \cdot b[2]) + (a[2] \cdot b[1]) + (a[3] \cdot b[0])$
12:     return $c$
13: end function
```

3.2.4 An Analysis of $k-2$ Level NTT then One-Iteration Karatsuba Multiplication

It is also possible to multiply two polynomials in modulo $\mathbb{Z}_q[X]/(X^4-r)$ by using one-iteration Karatsuba multiplication given by [50]. It requires ten multiplications and 27 additions without modular reduction. The multiplication of two polynomials in $\mathbb{Z}_q[X]/(X^4-r)$ is given in Algorithm 11.

Algorithm 11 Multiplication of polynomials in $\mathbb{Z}_q/(X^4-r)$

Input: a and $b \in \mathbb{Z}_q/(X^4-r)$ where r is a power of γ. i-th coefficient of a is denoted as $a[i]$ where $0 \le i < 2$.

Output: $c \in \mathbb{Z}_q/(X^4-r)$.

```
1: function basemul($a$, $b$)
2:     $d \leftarrow$ Apply Algorithm 2 of [50] to get $d = a \cdot b$ where $d$ is a degree $2n-2$ polynomial
3:     $c[0] \leftarrow d[0] + d[4] \cdot r$          ▷ $+$ and $\cdot$ for modular reduction
4:     $c[1] \leftarrow d[1] + d[5] \cdot r$          ▷ $+$ and $\cdot$ for modular reduction
5:     $c[2] \leftarrow d[2] + d[6] \cdot r$          ▷ $+$ and $\cdot$ for modular reduction
6:     $c[3] \leftarrow d[3]$
7:     return $c$
8: end function
```

Modular reduction requires three multiplications and three additions, as in the previous method and as shown in Algorithm 11. Hence, it requires 13 multiplications and 30 additions, including modular reductions in modulo (X^4-r). Then, the total costs of this approach can be written as

$$\begin{aligned} \mathit{Hybrid\text{-}KM}_2(n) &= 3 \cdot 4 \cdot T\big(\frac{n}{4}\big) + \frac{1}{4}\big(13n \cdot M + 30n \cdot A\big), \\ &= \frac{3k-6}{2} \cdot n \cdot M + 3 \cdot n \cdot (k-2) \cdot A + \frac{13n}{4} \cdot M + \frac{30n}{4} \cdot A, \\ &= \frac{6k+1}{4} \cdot n \cdot M + \big(3k + \frac{3}{2}\big) \cdot n \cdot A. \end{aligned}$$

3.2.5 An Analysis of $k-3$ Level NTT then Schoolbook Multiplication

Proposition 3.4. *The multiplication in modulo $\mathbb{Z}_q[X]/(X^8-r)$ is performed with 71 multiplications and 56 additions including modular reductions in modulo (X^8-r).*

Proof. The multiplication of two degree seven polynomials require $8^2 = 64$ multiplications and $7^2 = 49$ additions. The modular reduction in modulo (X^8-r) takes seven multiplications with r and seven additions. Therefore, in total, this multiplication is performed with 71 multiplications and 56 additions. $\square$

After the multiplication in modulo $\mathbb{Z}_q[X]/(X^8-r)$, we get eight coefficients. Therefore, the total cost is

$$\begin{aligned}
\textit{Hybrid-SM}_3(n) &= 3\cdot 8\cdot T\big(\frac{n}{8}\big) + \frac{1}{8}\big(71n\cdot M + 56n\cdot A\big),\\
&= \frac{3k-9}{2}\cdot n\cdot M + 3\cdot n\cdot(k-3)\cdot A + \frac{71n}{8}\cdot M + 7\cdot n\cdot A,\\
&= \frac{12k+35}{8}\cdot n\cdot M + (3k-2)\cdot n\cdot A.
\end{aligned}$$

3.2.6 An Analysis of $k-3$ Level NTT then One-Iteration Karatsuba Multiplication

The multiplication of two degree seven polynomials costs 36 multiplications and 133 additions without modular reductions by using one-iteration Karatsuba multiplication of [50]. The modular reduction in modulo (X^8-r) of the multiplication of two degree seven polynomials requires seven multiplications and seven additions as in the previous method. Altogether it costs 43 multiplications and 140 additions with the modular reduction in modulo (X^8-r). After the multiplication, we get eight coefficients. Therefore, the total cost for this implementation is

$$\begin{aligned}
\textit{Hybrid-KM}_3(n) &= 3\cdot 8\cdot T\big(\frac{n}{8}\big) + \frac{1}{8}\big(43n\cdot M + 140n\cdot A\big),\\
&= \frac{3k-9}{2}\cdot n\cdot M + 3\cdot n\cdot(k-3)\cdot A + \frac{43n}{8}\cdot M + \frac{140n}{8}\cdot A,\\
&= \big(\frac{12k+7}{8}\big)\cdot n\cdot M + \big(3k+\frac{17}{2}\big)\cdot n\cdot A.
\end{aligned}$$

The summary of all methods can be found on Table 3.1. Although this is not directly seen from Table 3.1, multiplication counts are more important than addition. Because if one uses integer words in implementation with a Montgomery reduction is needed after every multiplication, the Barrett reduction might be omitted after an addition. These omitted reductions are called as lazy reductions. The number of possible joint lazy reductions increases as q decreases from the next computer word. As mentioned before, the bounds on q are looser when fewer NTT levels are applied. On the other hand, the operation counts increase when the applied NTT level number decreases. Therefore, there is a trade-off between the two. We decided to move on with $k-2$ level for $(n = 512)$ and $k-3$ level for $(n = 1024)$ since this enables the same q selection for both of them. Moreover, Karatsuba multiplication is preferred for NEWHOPE-COMPACT.

Table 3.1: The computation costs of different polynomial multiplication algorithms. The total number of multiplication and addition numbers to multiply two polynomials in $\mathcal{R}_q$.

Operations / Multiplication Methods	Multiplications	Additions
NTT-based Multiplication	$\frac{12k+8}{8} \cdot n$	$3k \cdot n$
Hybrid Schoolbook-1 Multiplication	$\frac{12k+8}{8} \cdot n$	$(3k-2) \cdot n$
Hybrid Schoolbook-2 Multiplication	$\frac{12k+14}{8} \cdot n$	$(3k-3) \cdot n$
Hybrid Karatsuba-2 Multiplication	$\frac{12k+2}{8} \cdot n$	$\left(3k+\frac{3}{2}\right) \cdot n$
Hybrid Schoolbook-3 Multiplication	$\frac{12k+35}{8} \cdot n$	$(3k-2) \cdot n$
Hybrid Karatsuba-3 Multiplication	$\frac{12k+7}{8} \cdot n$	$\left(3k+\frac{17}{2}\right) \cdot n$

Note that applying these multiplication methods to KYBER give different computation costs, since the elements are represented as vectors or matrices of polynomials instead of just polynomials. Their polynomials are smaller. However, vector or matrix representation removes this advantage and adds extra complexity.

3.3 Implementation Details

Let us recall some important parts of the NEWHOPE algorithm first. Refer to [3] for full explanations.

Sampling: The secret and error terms are sampled by using the centered binomial distribution ψ_8. Sampling from ψ_8 is computed with the formula $\sum_{i=0}^{7} b_i - b_i'$ where the $b_i, b_i' \in \{0, 1\}$ are uniform independent bits. These bits are generated with SHAKE256, which generates 128 bytes of random on each call. These 128 bytes are used to construct 64 coefficients since each of them needs 16 random bits (2 bytes) to be computed. In order to generate all coefficients of a polynomial, SHAKE256 should be called 8 or 16 times for NEWHOPE512 or NEWHOPE1024 respectively.

NTT and NTT^{-1}: NEWHOPE follows the Twisted FFT Trick. Meaning that it uses Gentleman-Sande butterflies for both forward and inverse NTTs. After a full level forward NTT, which is 9 for NEWHOPE512 and 10 for NEWHOPE1024, coefficient-wise multiplication is performed. This is the standard NTT-based polynomial multiplication, and it is described in Section 2.3.

GenA: The public parameter **a** is generated by using this function. It takes 32 bytes random seed and generates **a** by expanding this seed with SHAKE128. This is one of the most time-consuming parts of the algorithm.

Encoding and decoding of the secret and public key: The coefficients of secret and public key polynomials are encoded into a byte array. Each coefficient consists of 14 bits due to the size of the selected prime. Therefore, all coefficients can be mapped to 896 bytes (NEWHOPE512) or 1792 bytes (NEWHOPE1024). The public key also includes 32 bytes seed to generate the public parameter **a**, for a total of 928 bytes (NEWHOPE512) or 1824 bytes (NEWHOPE1024). Decoding is the reverse of this operation, mapping a byte array to a polynomial.

Encoding and decoding of the ciphertext: The 32-byte message is encoded into a polynomial in $\mathcal{R}_q$. Each bit of the message is encoded into $\lfloor n/256 \rfloor$ coefficients to allow robustness against errors. Then, this polynomial is encrypted. The encryption result has two polynomials in $\mathcal{R}_q$, the public key polynomial u, and another polynomial v'. u is encoded in the same way with the public key **a**. On the other hand, v' is compressed by performing a modulus switching between modulus q and modulus 8. The total size of the ciphertext, containing u and v', after these encoding and compression is 1088 bytes (NEWHOPE512) or 2176 bytes (NEWHOPE1024).

The parameter sets of NEWHOPE are given on Table 3.2.

Table 3.2: Parameters of NEWHOPE512 and NEWHOPE1024 and derived high level properties [3]

Parameter Set	NEWHOPE512	NEWHOPE1024
Dimension n	512	1024
Modulus q	12289	12289
Noise Parameter k	8	8
NTT parameter γ	10968	7
Decryption error probability	2^{-213}	2^{-216}
Claimed post-quantum bit security	101	233
NIST Security Strength Category	1	5

Our first design decision is to use the FFT Trick so that the Cooley-Tukey butterfly in forward NTT and Gentleman Sande butterfly in reverse NTT are used and keep the level number of NTTs at 7 and the same for all n values so that the parameters modulus q, noise parameter k, and NTT parameter γ remain unchanged. This provides simplicity, and less functions are affected when the ring is changed. This makes switching from one parameter set to another and changing the security level easy. The only changing part is the multiplication of coefficients in NTT representation. NTT and NTT^{-1} functions are exactly the same for both $(n = 512)$ and $(n = 1024)$. In consequence, the precomputed powers of γ and γ^{-1} are the same for both and only 256 bytes in total, unlike the original NTT implementation for NEWHOPE ring which requires at least 1536 bytes for $(n = 512)$ and totally different 3072 bytes for $(n = 1024)$. There is also no need for a bit-reversal table which costs 1024 bytes $(n = 512)$ or 2048 bytes $(n = 1024)$ unlike the original NEWHOPE. In other words, the new implementation requires at least 2560 bytes memory less in total, which can be very important for constrained devices. This also enables having both security levels together at the same device without a memory penalty. Our level choice for NTT is 7, as mentioned before. This implies $k-2$ level NTT approach $(n = 512)$ or $k-3$ level NTT approach $(n = 1024)$, since the total level number is 9 or 10 respectively. Therefore, the coefficients after 7 level NTT is, in fact, a term of a polynomial in $\mathbb{Z}_q[X]/(X^4 \pm r)$ for $(n = 512)$ and $\mathbb{Z}_q[X]/(X^8 \pm r)$ for $(n = 1024)$ where r is a power of γ. Moreover, in order to keep NTT and NTT^{-1} functions the same for both $(n = 512)$ and $(n = 1024)$ we reorder the coefficients of input polynomials in such a way that 128-point NTT can be performed. Implementing 512-point or 1024-point NTT and stopping at 7-th level requires different implementations for $(n = 512)$ and $(n = 1024)$ since it requires two coefficients with different lengths at each butterflies. In other words, at the first level u_0 and u_{256} are needed for the first butterfly for $(n = 512)$, on the other hand u_0 and u_{512} are required for $(n = 1024)$ where $u \in \mathcal{R}_q$. The reordering starts with dividing input polynomial in 4 $(n = 512)$ or 8 $(n = 1024)$ equivalent parts. Each parts contains exactly 128 coefficients. Let $u \in \mathbb{Z}_q/(X^{512}+1)$ and $v \in \mathbb{Z}_q/(X^{1024}+1)$. They are divided as

$$\begin{aligned}
u_a &= (u_0, u_4, u_8, \cdots, u_{512-4}), & v_a &= (v_0, v_8, v_{16}, \cdots, v_{1024-8}),\\
u_b &= (u_1, u_5, u_9, \cdots, u_{512-3}), & v_b &= (v_1, v_9, v_{17}, \cdots, v_{1024-7}),\\
u_c &= (u_2, u_6, u_{10}, \cdots, u_{512-2}), & v_c &= (v_2, v_{10}, v_{18}, \cdots, v_{1024-6}),\\
u_d &= (u_3, u_7, u_{11}, \cdots, u_{512-1}), & v_d &= (v_3, v_{11}, v_{19}, \cdots, v_{1024-5}),\\
& & v_e &= (v_4, v_{12}, v_{20}, \cdots, v_{1024-4}),\\
& & v_f &= (v_5, v_{13}, v_{21}, \cdots, v_{1024-3}),\\
& & v_g &= (v_6, v_{14}, v_{22}, \cdots, v_{1024-2}),\\
& & v_h &= (v_7, v_{15}, v_{23}, \cdots, v_{1024-1}).
\end{aligned}$$

After this reordering process, one can see that applying 7 level 512-point NTT to u is equivalent to applying 7 level 128-point NTT to u_a, u_b, u_c and u_d. However, the result of the latter is not equivalent to the result of the former. Instead, it is reordered, just like the input. As a result, the small polynomials in the NTT domain are constructed as follows:

$$\begin{aligned}\hat{x} = \big(\hat{x}_0 &= u_a(0) + u_b(0) \cdot X + u_c(0) \cdot X^2 + u_d(0) \cdot X^3, \\ \hat{x}_1 &= u_a(1) + u_b(1) \cdot X + u_c(1) \cdot X^2 + u_d(1) \cdot X^3 \\ &. \\ &. \\ &. \\ \hat{x}_{126} &= u_a(126) + u_b(126) \cdot X + u_c(126) \cdot X^2 + u_d(126) \cdot X^3 \\ \hat{x}_{127} &= u_a(127) + u_b(127) \cdot X + u_c(127) \cdot X^2 + u_d(127) \cdot X^3\big),\end{aligned}$$

where $x_i \in \mathbb{Z}_q[X]/(X^4 \pm r)$ for $0 \leq i < 128$. A similar strategy is followed for $(n = 1024)$. The reordering process does not cost anything because all inputs are randomly generated or distributed so one can assume that they are already reordered. Furthermore, a better vectorization is possible for AVX2 implementation by applying reordering.

Moreover, as discussed in Section 2.3.3, we can now select our q such that it satisfies $q \equiv 1 \mod (512/2^{k-(k-2)-1})$ $(n = 1024)$ or $q \equiv 1 \mod (1024/2^{k-(k-3)-1})$ $(n = 512)$ which both are equal to $q \equiv 1 \mod 256$. The smallest q that satisfies this equality is 3329. In result, we have selected our parameters as shown on Table 3.3. To analyze the failure probability for our parameters, we follow the approach from [12]. Moreover, the post-quantum bit security of the NEWHOPE-COMPACT parameter set is estimated by using the "PQSecurity.py" script provided by NEWHOPE. The result of the script is given in Table 3.4.

The new parameter sets speed up the sampling of secret and noise polynomials. They are sampled from ψ_2 now. Sampling from ψ_2 is computed with the formula $\sum_{i=0}^{1} b_i - b'_i$. As mentioned before, the random bits are generated by using SHAKE256 which generates 128 bytes of random on each call. Different from the NEWHOPE sampling function, which samples 64 coefficients on each call, we can now sample 256 coefficients at a time.

The new coefficients are composed of 12 bits instead of 14 bits, since q is reduced to 3329 from 12289. Therefore, all coefficients of the secret key can now be encoded as

Table 3.3: Parameters of NEWHOPE-COMPACT512 and NEWHOPE-COMPACT1024 and derived high level properties

Parameter Set	NH-COMPACT512	NH-COMPACT1024
Dimension n	512	1024
Modulus q	3329	3329
Noise Parameter k	2	2
NTT parameter γ	17	17
Decryption error probability	2^{-256}	2^{-181}
Claimed post-quantum bit security	100	230
NIST Security Strength Category	1	5

Table 3.4: Core hardness of NEWHOPE-COMPACT512 and NEWHOPE-COMPACT1024

Attack	m	b	Known Classical	Known Quantum	Best Plausible
NEWHOPE-COMPACT512: $q = 3329$, $n = 512$, $\varsigma = \sqrt{2}$					
Primal	462	381	111	101	79
Dual	465	380	111	100	78
NEWHOPE-COMPACT1024: $q = 3329$, $n = 1024$, $\varsigma = \sqrt{2}$					
Primal	841	873	255	231	181
Dual	862	868	253	230	180

1632 bytes $(n = 512)$ or 3168 bytes $(n = 1024)$. The public key which also includes 32 bytes seed is represented with 800 bytes $(n = 512)$ or 1568 bytes $(n = 1024)$. Moreover, the total size of the ciphertext decreases to 992 bytes $(n = 512)$ or 2080 bytes $(n = 1024)$. All of these mentioned sizes are summarized on Table 3.5.

Table 3.5: Sizes of public keys, secret keys and ciphertexts of NEWHOPE (denoted as [3]) and NEWHOPE-COMPACT (denoted as Ours) in bytes

Parameter Set	$\|pk\|$		$\|sk\|$		$\|ciphertext\|$	
	[3]	Ours	[3]	Ours	[3]	Ours
512-CCA-KEM	928	800	1888	1632	1120	992
1024-CCA-KEM	1824	1568	3680	3168	2208	2080

3.4 Results and Comparision

Benchmark tests for all implementations are performed on an Intel Core i7-6500U Skylake processor running at 2500 MHz with Turbo Boost and Hyperthreading disabled. The operating system is Ubuntu 18.04.2 LTS with Linux Kernel 4.15.0, and all software are compiled with gcc-7.3.0. For comparisons, we have taken C reference (non-optimized) implementations of NEWHOPE from `https://github.com/newhopecrypto/newhope` and KYBER from `https://github.com/pq-cystals/kyber`. We report the median of 10000 executions of the corresponding function for cycle counts.

The benchmark results for reference (non-optimized) C implementations of KYBER, NEWHOPE and NEWHOPE-COMPACT for security level 1 and 5 are given on Table 3.6. Our CCA-KEM implementations are 1.21 times ($n = 512$) or 1.13 times ($n = 1024$) faster, and 1.17 times ($n = 512$) or 1.26 times ($n = 1024$) faster than the corresponding implementations of [3] and [12] respectively. We have used one-iteration Karatsuba multiplication, which is mentioned in Section 3.2 as the base multiplication method in our reference C implementations. The main reason for getting a faster implementation than NEWHOPE is the size of q. Since the size of q is reduced by two bits, we can get more benefits from lazy reductions. In fact, for forward NTT, we do not need any modular reduction after addition or subtraction, while NEWHOPE requires a modular reduction after addition at each odd level. On the other hand, KYBER uses the same prime. Therefore, we do not get any benefit from lazy reductions. The difference in the cycle counts between this work and KYBER is due to the generation of the public parameter a and the difference in the constructions of the schemes.

Note that an optimized avx2 implementation of NEWHOPE-COMPACT would be more precise to compare performance results with other schemes.

Thanks to its structure, KYBER is offering a parameter set for NIST security level 3. However, the NEWHOPE ring needs to be changed for a similar security level.

Table 3.6: Cycle counts of KYBER, NEWHOPE and NEWHOPE-COMPACT C reference (non-optimized) implementations.

Operations	CCA-KEM-512			CCA-KEM-1024		
	KYBER [12]	NEWHOPE [3]	This work	KYBER [12]	NEWHOPE [3]	This work
GEN	121 586	119 163	89 286	324 614	237 791	186 438
ENCAPS	163 996	180 193	147 045	381 360	365 240	320 787
DECAPS	197 532	203 440	176 098	431 408	417 448	394 910
Total	483 114	502 796	412 429	1 137 382	1 020 479	902 135

Proposed NTT implementation can be easily adopted such rings, see Section 3.5.

3.5 An Implementation for n=768

NEWHOPE does not offer a parameter set for NIST security level 3. However, a recent observation by [35] made it possible by offering a new ring structure $\mathbb{Z}_q/(X^{768} - X^{384} + 1)$. They start with splitting this polynomial into two polynomials of the form $X^{n/2} - \zeta_1$ and $X^{n/2} - \zeta_2$ such that ζ_1 and ζ_2 are two primitive sixth root of unity and $\zeta_1 + \zeta_2 = 1$, $\zeta_1 \cdot \zeta_2 = 1$. The CRT map of this ring is as follows:

$$\begin{aligned}\mathcal{R}_q = \mathbb{Z}_q[X]/(X^{768} - X^{384} + 1) \rightarrow \\ \mathbb{Z}_q[X]/(X^{384} - \zeta_1) \times \mathbb{Z}_q[X]/(X^{384} - \zeta_2)\end{aligned}$$

Let $f \in \mathcal{R}_q$. Then, in order to get the coefficients after the first level, we need to compute:

$$g = f \mod (X^{n/2} - \zeta_1), h = f \mod (X^{n/2} - \zeta_2)$$

We know that $\zeta_2 = 1 - \zeta_1$. Therefore, instead of computing $h = f \mod (X^{n/2} - \zeta_2)$ we can compute $h = f \mod (X^{n/2} + 1 - \zeta_1)$. Therefore, the burden of multiplication with ζ_2 to perform the modular reduction is removed. We can benefit from the already computed product with ζ_1. After this trick is applied, it turns out that a standard NTT can be performed. We have 7-level NTT left. We know that $\zeta_1^6 \equiv 1 \mod q$. To be able to perform 7-level NTT by using ζ_1, $\gamma^{128} \equiv \zeta_1 \mod q$ is needed. Then, q needs to satisfy $q \equiv 1 \mod 768$. The smallest q that satisfies this condition is 7681. That's why, [35] selects q as 7681 and γ such that it satisfies $\gamma^{128} \equiv \zeta_1 \mod 7681$.

This also implies that $\gamma^{640} \equiv \zeta_2 \mod 7681$, since $\zeta_1^5 = \zeta_2$. After NTT, there are 256 polynomials of degree 3 in $\mathbb{Z}_q[X]/(X^3 \pm r)$.

In order to keep our total NTT level the same for all of our implementations, we changed the parameters of [35]. Our algorithm for $n = 512$ and $n = 1024$ has 7-level NTT. Therefore, in order to use 7-level NTT q needs to satisfy $q \equiv 1 \mod 384$ so that $\gamma^{64} \equiv \zeta_1 \mod q$ exits. We cannot use 3329, since $3329 \neq 1 \mod 384$. The smallest such q is 3457. However, 3457 cannot be used for other rings $\mathbb{Z}_q/(X^{512}+1)$ and $\mathbb{Z}_q/(X^{1024}+1)$, because 256-th root of unity does not exits ($3457 \neq 1 \mod 256$) so there is no γ such that $\gamma^{256} = 1$ and 7-level NTT is not possible. Therefore, we have selected our parameters for $n = 768$ as $q = 3457$, noise parameter $k = 2$. By using the "PQSecurity.py" script provided by NEWHOPE, the post-quantum bit-security is estimated as 163. The result of this script is given in Table 3.7. Therefore, it achieves NIST security level 3.

Table 3.7: Core hardness of NEWHOPE-COMPACT768

Attack	m	b	Known Classical	Known Quantum	Best Plausible
NEWHOPE-COMPACT768: $q = 3457$, $n = 768$, $\varsigma = \sqrt{2}$					
Primal	655	620	181	164	128
Dual	667	617	180	163	128

Although there are different approaches like [25] to analyze the failure probability for our parameters, we follow the approach from [12]. But because the underlying ring has a trinomial quotient, each coefficient of the result of multiplication becomes a sum of $\frac{n}{2}$ multiplication of elements in the form of $ab + b'(a + a')$, where a, a', b, b' from ψ_2. Although some coefficients of the result are of the form $ab + ab'$ in their sum, we use the first form for simplicity, which is also suggested in [35]. Thus, the result is a sum of $\frac{n}{2}$ multiplication of the form of $ab + b'(a + a')$. This computation gave us 2^{-170} failure probability for $n = 768$.

The sizes of the public key, the secret key, and the ciphertext when q is selected as 3457 are 1184 bytes, 2400 bytes, and 1568 bytes, respectively.

Benchmark results for our C implementation can be found on Table 3.8. It is 1.18 times faster than non-optimized reference implementation of KYBER.

Table 3.8: Cycle counts of KYBER and NEWHOPE C reference (non-optimized) implementation

Operations	CCA-KEM-768		
	KYBER [12]	NEWHOPE [3]	This work
GEN	208 826	-	137 960
ENCAPS	254 838	-	228 976
DECAPS	294 748	-	277 814
Total	758 412	-	644 750

CHAPTER 4

CORTEX-M4 OPTIMIZATIONS FOR {R,M}LWE SCHEMES

This chapter is based on the publication "Cortex-M4 optimizations for {R,M}LWE schemes" [5].

Section 4.2 describes algorithms most commonly used for efficient and constant-time implementations of modular reductions inside the NTT. Section 4.3 provides details of our implementation and optimizations to achieve a better performance while using an as small as possible stack space. It also gives a trade-off between secret-key size and speed. Our performance results for NEWHOPE, NEWHOPE-COMPACT, and KYBER as well as a comparison with previous implementations of those schemes are presented in Section 4.4. Finally, in Section 4.5, we conclude this chapter.

4.1 Introduction

While most of the candidates already have optimized versions of their code targeting large CPUs, which often feature vector extensions such as AVX2, implementations for such architectures usually do not consider the memory usage or code size as a bottleneck. Instead, they usually are solely optimized for computation time. However, small embedded devices can be greatly impacted by the switch toward a post-quantum paradigm. They often offer less memory, low computing power, and even have the drawback to be more susceptible to side-channel attacks. Hence, in recent papers, researchers focused more and more on small devices, e.g., based on the popular ARM Cortex-M4 processor, to assess the performance on embedded platforms. This microcontroller has the advantage of having large enough memory to support public-key algorithms while being still reasonably small and cheap in the grand scheme of computing. Its popularity led to the development of `pqm4` [29], a library aiming to offer a common framework for benchmarking implementations of post-quantum algorithms on this platform.

Availability of the software: All source code is available at `https://github.com/erdemalkim/NewHope-Compact-M4`. The source code of KYBER and NEWHOPE have already been pushed to `pqm4`.

4.2 Preliminaries

In this section, we provide the required background that is necessary for understanding the remainder of this chapter.

NewHope-Compact: Alkim, Bilgin, and Cenk proposed a compact and fast instantiation of NEWHOPE called NEWHOPE-COMPACT [4]. They presented three new parameter sets, which are shown in Table 4.1. As can be seen from the new parameter sets, they reduced the modulus q from 12289 to 3329 while preserving the same security level by the adjustment of the noise parameter k. Due to the change in q, the $2n$-th root of unity γ does not exist anymore, i.e., $q \not\equiv 1 \pmod{2n}$. However, even in this situation, recent results [4,12,35,51] show that a fast NTT is still possible. The use of the NTT in this situation is achieved in [4] by selecting γ as the 256-th root of unity so that a 7-level NTT is possible. Then, at the end, instead of having $n = 512$ or 1024 integer coefficients, i.e., degree zero polynomials, NEWHOPE-COMPACT has 128 degree three or degree seven polynomials, respectively. Then, coefficient-wise multiplications in the NTT domain are performed on small-degree polynomials modulo $(X^4 - r)$ for $n = 512$ or $(X^8 - r)$ for $n = 1024$, where r is a power of γ, instead of multiplication of integer coefficients. They used a one-iteration Karatsuba method [50] for the multiplication of small-degree polynomials. The pseudocode of this step can be found in [4, Alg. 5]. Note that the only parts changed from NEWHOPE to NEWHOPE-COMPACT are the definition of the NTT and the coefficient-wise multiplication. Therefore, one can still refer to Algorithm 4, Algorithm 5, and Algorithm 6 for the definition of key generation, encryption, and decryption, respectively, since

Table 4.1: Parameters of NEWHOPE-COMPACT512, NEWHOPE-COMPACT768 and NEWHOPE-COMPACT1024 and derived high-level properties [4].

Parameter Set	NH-COMPACT512	NH-COMPACT768	NH-COMPACT1024
Dimension n	512	768	1024
Modulus q	3329	3457	3329
Noise Parameter k	2	2	2
NTT parameter γ	17	55	17
Decryption error probability	2^{-256}	2^{-170}	2^{-181}
Claimed post-quantum bit security	100	163	230
NIST Security Strength Category	1	3	5

these algorithms are written from a high-level perspective and the mentioned changes are hidden inside internal functions.

Another contribution of [4], as can be seen in Table 4.1, is the proposal of a new security level for NEWHOPE, which is referred to as NEWHOPE-COMPACT768. This new security level is made possible by using a different ring structure $\mathbb{Z}_q[X]/(X^{768} - x^{384} + 1)$, first proposed by [35]. NEWHOPE-COMPACT selects q as 3457, which allows a similar implementation with other parameter sets. They applied a trick at the first level of the NTT to switch the regular ring structure $\mathcal{R}_q$ and a similar one at the last level of the NTT^{-1}. This trick uses the factorization of $(X^2 - X + 1)$ as $(X - \zeta_1)$ and $(X - \zeta_2)$, where ζ_1 and ζ_2 are both sixth roots of unity. We refer to [4, Appendix A] and [35, Sec. 4.1] for more details. In the end, the one-iteration Karatsuba method is applied to perform coefficient-wise multiplications for the small-degree polynomials modulo $(X^6 - r)$, where r is a power of γ.

4.2.1 FFT Trick

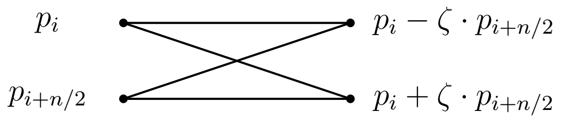
In this work, we used what is known in the recent literature as the FFT trick [47]. The idea is to map the ring

$$\mathbb{Z}_q[X]/\langle X^n - \gamma^n \rangle$$

to

$$\mathbb{Z}_q[X]/\langle X^{n/2} - \gamma^{n/2} \rangle \times \mathbb{Z}_q[X]/\langle X^{n/2} + \gamma^{n/2} \rangle$$

Figure 4.1: Cooley-Tukey Butterfly



by computing the straightforward CRT map

$$p \mapsto (p \bmod X^{n/2} - \gamma^{n/2}, p \bmod X^{n/2} + \gamma^{n/2}).$$

Since $X^{n/2} + \gamma^{n/2} = X^{n/2} - \gamma^{n+n/2}$, the same map can be computed again on both components until reaching a product of rings in which the multiplication is cheap.

The core operation is to reduce a polynomial p modulo both $X^{n/2} \pm \zeta$ (where ζ is an arbitrary power of γ) is called a butterfly and is depicted in Figure 4.1. The computation of the NTT consists of applying $n/2$ butterflies to pairs of coefficients of the whole polynomial iteratively between 1 and $\log_2 n$ times, each iteration being referred to as a layer. Figure 4.2 depicts a full NTT consisting of 9 layers mapping the ring $\mathbb{Z}_q[X]/\langle X^{512} + 1\rangle$ to a product of rings of the form $\mathbb{Z}_q[X]/\langle X - \zeta\rangle$. In NEWHOPE-COMPACT and KYBER, the NTT is stopped earlier because $\mathbb{Z}_q$ does not offer enough high-order roots of unity to compute all the layers. This means that the NTT itself is less expensive, but the base operation in the product of rings is more costly. This base operation is a polynomial multiplication in rings of the form $\mathbb{Z}_q[X]/\langle X^a - r\rangle$ for a in $\{2, 4, 6, 8\}$ depending on the algorithm and parameter set used.

4.2.2 Montgomery and Barrett Reductions

Montgomery [37] and Barrett [13] reductions are beneficial for efficient and constant-time implementations of reductions in $\mathcal{R}_q$. Efficient versions for signed integers of these reduction algorithms were presented in [47, Alg. 3, 5], and they are recalled in Algorithm 12 and Algorithm 13. Moreover, assembly implementations of these reduction algorithms on the Cortex-M4 are provided in Algorithm 14, [16, Alg. 7], and Algorithm 15. There are two important things to consider when using these efficient reduction algorithms: Firstly, while the Montgomery reduction gives outputs

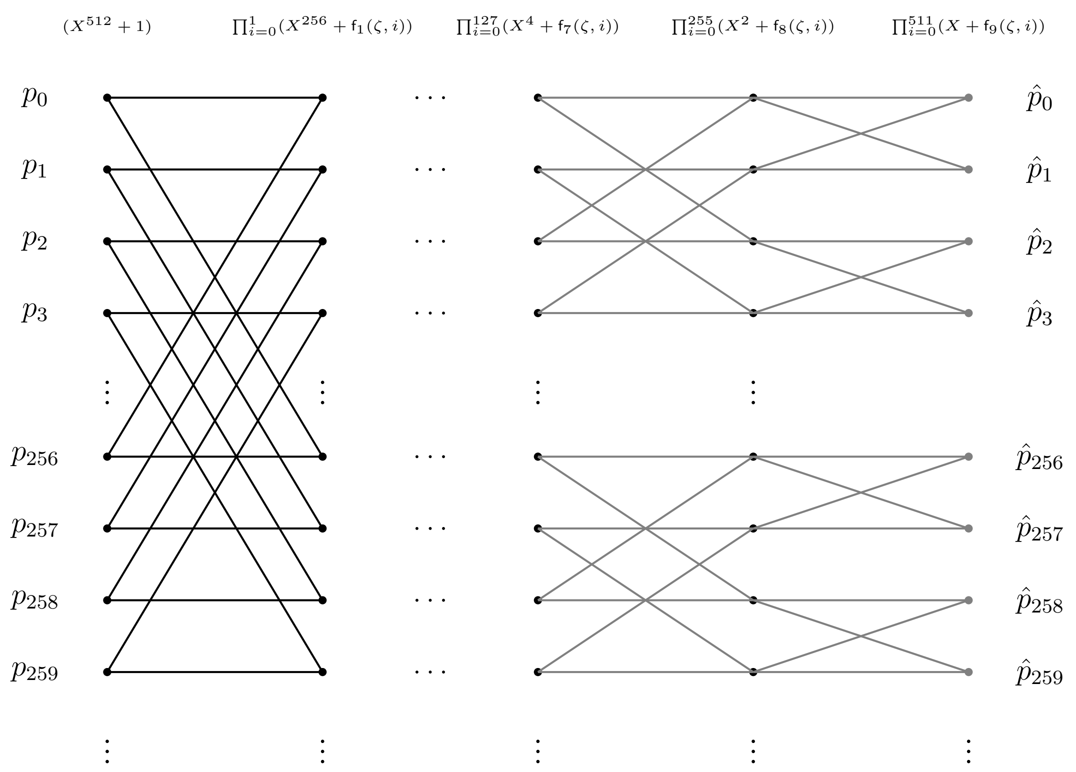


Figure 4.2: Full NTT on a degree 511 polynomial. The function $f_j(\zeta, i) = \zeta^{\mathsf{brv}(2^j-1+i)}$ selects the correct root to compute the isomorphism. All those roots are usually pre-computed and correctly ordered in a table. Techniques to reduce q skip some levels: for example, using $q = 3329$ as in NEWHOPE-COMPACT requires to skip the last two layers (grey).

between $-q$ and q, the Barrett reduction gives outputs between 0 and q. Secondly, the Montgomery reduction cannot handle all signed words. It accepts inputs in the range of $-\frac{\beta}{2}q$ to $\frac{\beta}{2}q$, where β is selected as 2^{16} for efficient implementations.

4.2.3 ARM Cortex-M4

Our target platform is the STM32F4DISCOVERY [49] development board featuring a 32-bit ARM Cortex-M4 [10], which is the selected platform by NIST to evaluate post-quantum candidates on microcontrollers. The Cortex-M4 implements the ARMv7E-M instruction set and provides special Digital Signal Processing (DSP) instructions. These DSP extensions offer SIMD instructions that can perform arithmetic operations on two halfwords or four bytes in parallel. These instructions have been shown to be very beneficial to speed up post-quantum algorithms [9, 15, 16, 28–30, 46]. This architecture comes with the restriction of a limited number of registers, which is 16

Algorithm 12 Signed Montgomery reduction [47] using Montgomery factor $\beta = 2^{16}$.

Input: odd q where $0 < q < \frac{\beta}{2}$, and a where $-\frac{\beta}{2}q \leq a = a_1\beta + a_0 < \frac{\beta}{2}q$ and $0 < a_0 < \beta$

Output: r' where $r' = \beta^{-1}a \pmod{q}$, and $-q < r' < q$

```
1: $m \leftarrow a_0 q^{-1} \pmod{^{\pm} \beta}$        ▷ signed low product, $q^{-1}$ precomputed
2: $t_1 \leftarrow \left\lfloor \frac{mq}{\beta} \right\rfloor$        ▷ signed high product
3: $r' \leftarrow a_1 - t_1$
```

Algorithm 13 Signed Barrett reduction [47] using $\beta = 2^{16}$.

Input: odd q where $0 < q < \frac{\beta}{2}$, and a where $-\frac{\beta}{2} \leq a < \frac{\beta}{2}$

Output: r where $r = a \pmod{q}$, and $0 \leq r \leq q$

```
1: $v \leftarrow \left\lfloor \frac{2^{\log(q)-1} \cdot \beta}{q} \right\rceil$        ▷ precomputed
2: $t \leftarrow \left\lfloor \frac{av}{2^{\log(q)-1} \cdot \beta} \right\rfloor$        ▷ signed high product and arithmetic right shift
3: $t \leftarrow tq \mod \beta$        ▷ signed low product
4: $r \leftarrow a - t$
```

Algorithm 14 Signed Montgomery reduction [16] using Montgomery factor $\beta = 2^{16}$.

Input: a where $-\frac{\beta}{2}q \leq a < \frac{\beta}{2}q$

Output: reduced $a \rightarrow r'$ where $r' = \beta^{-1}a \pmod{q}$, and $-q < r' < q$

```
1: smulbb $t, a, q^{-1}$        ▷ $t \leftarrow (a \mod \beta) \cdot q^{-1}$
2: smulbb $t, t, q$        ▷ $t \leftarrow (t \mod \beta) \cdot q$
3: usub16 $a, a, t$        ▷ $a_{top} \leftarrow \left\lfloor \frac{a}{2^{16}} \right\rfloor - \left\lfloor \frac{t}{2^{16}} \right\rfloor$
```

general-purpose 32-bit registers, out of which only 14 are available for the developer. The Cortex-M4 is also used by the benchmarking and testing framework `pqm4` [29].

4.3 Implementation Details

This section first describes our optimizations to speed-up the computation of the polynomial multiplication in $\mathcal{R}_q$. This includes both the computation of the NTT and the NTT^{-1} as well as coefficient-wise multiplication of polynomials modulo $(X^d - r)$ where $d = 1$ in NEWHOPE, $d = 2$ in KYBER, and $d = 4$, 6, and 8 in NEWHOPE-COMPACT for $n = 512$, 768, and 1024, respectively. Then, we present our implementation techniques that decrease the stack usage of NEWHOPE and NEWHOPE-COMPACT. We use the on-the-fly generation of $\hat{a}$ during arithmetic operations in the NTT domain proposed by [16] for KYBER. We also introduce a new method for key generation, which adds the error polynomial in the normal domain instead of in the NTT domain. Finally, we provide a trade-off between the size of the secret key and the performance.

4.3.1 Optimization of Polynomial Multiplication for Speed

Polynomial multiplication in $\mathcal{R}_q$ is one of the most time-consuming parts of KEMs whose security relies on RLWE/Ring Learning With Rounding (RLWR) or MLWE/-Module Learning With Rounding (MLWR) problems. Some of them, specifically NEWHOPE and KYBER, utilize the NTT for ring arithmetic in order to have a fast and efficient implementation. We present an optimized assembly implementation of polynomial multiplication on the Cortex-M4, which can be used by all RLWE/RLWR-MLWE/MLWR schemes utilizing the NTT for the ring arithmetic and have a modulus smaller than 2^{15}. Although some of the techniques described in this section are specific to the rings used by NEWHOPE, NEWHOPE-COMPACT, or KYBER, adapting an implementation of one to another with some minor changes is possible. We indicate such points when appropriate.

Representation of polynomials and packing: Similar to [16], we represent polynomials in $\mathcal{R}_q$ as an array composed of signed 16-bit integers. This representation was first introduced in an AVX2 implementation of KYBER by [47]. In their reference implementations, KYBER and NEWHOPE-COMPACT are already implemented by using signed halfword integers. However, [9] preferred using unsigned values for NEWHOPE on the Cortex-M4. As discussed previously in Section 2.4, NEWHOPE utilizes the Gentleman-Sande butterfly in both the NTT and the NTT^{-1}, while the other two schemes use the Cooley-Tukey butterfly in the NTT and the Gentleman-Sande butterfly in the NTT^{-1}. We decided to follow the approach used by KYBER and NEWHOPE-COMPACT since it has been shown in [47] that using this approach is more efficient when the coefficients of the polynomials are represented as signed halfwords. Moreover, the Cortex-M4 is a 32-bit architecture, while the polynomial coefficients are below 16 bits. Therefore, in order to fully utilize the properties of the Cortex-M4 platform, we packed two coefficients into one register. Thereby, we can utilize SIMD instructions and perform addition/subtraction on two halfwords in parallel by using `uadd16` or `usub16`. Moreover, similar to [16], we implement a double butterfly, which takes a packed register as input and returns a packed butterfly result.

Montgomery, Barrett, and lazy reductions: The Montgomery reduction, which is given in Algorithm 14, is implemented by [16] in three clock cycles. In this work, we optimize the implementation of the Montgomery reduction such that it can be performed in only two clock cycles. We achieve this by storing $-q^{-1}$ instead of q^{-1} and using the `smlabb` instruction, which multiplies two halfwords and adds the 32-bit result to another 32-bit value in one clock cycle. This implementation is given in Algorithm 16. The KYBER implementation of [16] performs 3200 Montgomery reductions (1792 in the two NTT, 896 in NTT^{-1}, 512 in base multiplication) in a full polynomial multiplication ($\mathsf{NTT}^{-1}(\mathsf{NTT}(a) \circ \mathsf{NTT}(b))$). Therefore, this change saves 3200 clock cycles for a full polynomial multiplication in $\mathcal{R}_q$.

Similarly to [16], we used the Barrett reduction given by [47, Alg. 5]. The assembly implementation of this reduction on a packed argument is given in Algorithm 15. It requires nine clock cycles on our Cortex-M4 microcontroller. We also implemented a Montgomery reduction on a packed argument (Algorithm 17). It needs eight clock cycles, which is slightly faster than the Barrett reduction. Each halfword of the input of Algorithm 17 is multiplied by β to be able to get the same result as the Barrett

Algorithm 15 Barrett reduction on packed argument using $\beta = 2^{16}$.

Input: a (32 bit signed integer where a_{top} and a_{bottom} contains two different coefficients)

Output: $r = r_{top} \mid r_{bottom}$ where $r_{top} \equiv a_{top} \pmod q$, $r_{bottom} \equiv a_{bottom} \pmod q$, $0 \leq r_{top}, r_{bottom} \leq q$ and $0 \leq q < 2^{15}$

```
1: movw v, const            ▷ v ← const where const=$\lfloor \frac{2^{\lfloor \log(q) \rfloor - 1} . 2^{\beta}}{q} \rceil$ and precomputed
2: smulbb $t_1, a, v$                          ▷ $t_1 \leftarrow a_{bottom} \cdot v$
3: smultb $t_2, a, v$                          ▷ $t_2 \leftarrow a_{top} \cdot v$
4: asr $t_1, t_1$, #$(\log(\beta) + \lfloor \log(q) \rfloor - 1)$    ▷ $t_1 \leftarrow t_1 >> (\log(\beta) + \lfloor \log(q) \rfloor - 1)$
5: asr $t_2, t_2$, #$(\log(\beta) + \lfloor \log(q) \rfloor - 1)$    ▷ $t_2 \leftarrow t_2 >> (\log(\beta) + \lfloor \log(q) \rfloor - 1)$
6: smulbb $t_1, t_1, q$                        ▷ $t_1 \leftarrow t_1 \cdot q$
7: smulbb $t_2, t_2, q$                        ▷ $t_2 \leftarrow t_2 \cdot q$
8: pkhbt $t, t_1, t_2$, lsl#16                 ▷ $t \leftarrow (t_1 \& 0xFFFFu) | (t_2 << 16)$
9: usub16 $r, a, t$              ▷ $r_{top} \leftarrow a_{top} - t_{top}$ and $r_{bottom} \leftarrow a_{bottom} - t_{bottom}$
```

Algorithm 16 Signed Montgomery reduction using Montgomery factor $\beta = 2^{16}$.

Input: a where $-\frac{\beta}{2} q \leq a < \frac{\beta}{2} q$

Output: reduced $a \rightarrow r'$ where $r' = \beta^{-1} a \pmod q$, and $-q < r' < q$

```
1: smulbb $t, a, -q^{-1}$          ▷ $t \leftarrow (a \mod \beta) \cdot (-q^{-1})$
2: smlabb $a, t, q, a$             ▷ $a_{top} \leftarrow \lfloor \frac{(t \mod \beta) \cdot q + a}{2^{16}} \rfloor$
```

reduction. Note that their outputs are not in the same range. Hence, after the last layer of the NTT and the NTT^{-1}, where we need our output to be between 0 and q, we use the Barrett reduction. In other cases, we use the faster Montgomery reduction.

Algorithm 17 Signed Montgomery reduction on packed argument using Montgomery factor $\beta = 2^{16}$.

Input: a (32 bit signed integer where a_{top} and a_{bottom} contains two different coefficients)

Output: $r = r_{top} \mid r_{bottom}$ where $r_{top} \equiv a_{top} \pmod q$, $r_{bottom} \equiv a_{bottom} \pmod q$

```
1: movw $v$, const          ▷ $v \leftarrow$ const where const=$\beta \pmod q$ and precomputed
2: smulbb $t_1, a, v$
3: smulbb $r_1, t_1, -q^{-1}$          ▷ $r_1 \leftarrow (t_1 \mod \beta) \cdot (-q^{-1})$
4: smlabb $r_1, r_1, q, t_1$          ▷ $r_{1_{top}} \leftarrow \left\lfloor \frac{(r_1 \mod \beta) \cdot q + t_1}{2^{16}} \right\rfloor$
5: smultb $t_2, a, v$
6: smulbb $r_2, t_2, -q^{-1}$          ▷ $r_2 \leftarrow (t_2 \mod \beta) \cdot (-q^{-1})$
7: smlabb $r_2, r_2, q, t_2$          ▷ $r_{2_{top}} \leftarrow \left\lfloor \frac{(r_2 \mod \beta) \cdot q + t_2}{2^{16}} \right\rfloor$
8: pkhtb $r, r_2, r_1$, asr #16          ▷ $r \leftarrow (r_{2_{top}} | (r_{1_{top}} >> 16))$
```

Depending on the size of the modulus and the register size of the underlying architecture, it is not always necessary to reduce the results after an addition or subtraction. Skipping unnecessary reductions is called lazy reduction. It is common that optimized NTT implementations heavily use this technique to speed-up the code. However, those lazy reductions are mostly performed after an addition or subtraction, as stated before. In this work, we also perform lazy reductions during component-wise multiplications, also referred to as base multiplications, for moduli 3329 and 3457 used in KYBER and NEWHOPE-COMPACT.

The base multiplication for KYBER is given in Algorithm 18. Each $\pmod q$ in Algorithm 18 corresponds to a Montgomery reduction. As we can see, five Montgomery reductions are needed in each base multiplication. We noticed that if both of the coefficients that are multiplied are already reduced modulo q, then the result is much lower than the value that the Montgomery reduction can handle, i.e., $2^{15} \cdot q$ (see Proposition 4.1). Thus, we can sum up the results of several multiplications before performing a Montgomery reduction, e.g., we can compute

($c[0] \leftarrow ((a[0] \cdot b[0]) + ((a[1] \cdot b[1]) \bmod q) \cdot r) \bmod q$) instead of applying line 2 of Algorithm 18. Therefore, we save one Montgomery reduction per coefficient, i.e., two per base multiplication. In total, there are 128 base multiplications in a polynomial multiplication in $\mathcal{R}_q$. Thus, we save 256 Montgomery reductions, i.e., 768 clock cycles for the implementation of [16] and 512 clock cycles for our implementation for a polynomial multiplication of KYBER.

Moreover, we can use this lazy-reduction technique for NEWHOPE-COMPACT. The base multiplications for NEWHOPE-COMPACT512, NEWHOPE-COMPACT768, and NEWHOPE-COMPACT1024 require 4, 6, and 8 sequential additions of the multiplication results, which can be handled as shown in Proposition 4.1. Therefore, we can skip 3, 5, or 7 Montgomery reductions per coefficient, which sums up to 1536, 3840, or 7168 Montgomery reductions in total for NEWHOPE-COMPACT512, NEWHOPE-COMPACT768, or NEWHOPE-COMPACT1024, respectively. Note that we can also omit the Montgomery reductions after the multiplications in the first layer of the Cooley-Tukey butterflies, where the inputs are polynomials with small coefficients that are sampled from the centered binomial distribution. The results of the first multiplications can only grow up to $\pm 2q$ for KYBER and NEWHOPE-COMPACT and up to $\pm q$ for NEWHOPE. However, this technique cannot be used if the input polynomial is not a polynomial having small coefficients, i.e., not sampled from the centered binomial distribution. We have such cases in KYBER by design because of the ciphertext compression. We also cause this case in NEWHOPE and NEWHOPE-COMPACT with the stack-usage optimization explained in Section 4.3.2.

Algorithm 18 Multiplication of polynomials in $\mathbb{Z}_q[X]/(X^2 - r)$ for KYBER.

Input: a and $b \in \mathbb{Z}_q[X]/(X^2 - r)$ where r is a power of γ.

Output: $c \in \mathbb{Z}_q[X]/(X^2 - r)$.

```
1: function basemul($a$, $b$)
2:     $c[0] \leftarrow (a[0] \cdot b[0]) \bmod q + ((a[1] \cdot b[1]) \bmod q) \cdot r) \bmod q$
3:     $c[1] \leftarrow (a[0] \cdot b[1]) \bmod q + (a[1] \cdot b[0]) \bmod q$
4:     return $c$
5: end function
```

Proposition 4.1. *Let* $-3329 < a_i, b_i < 3329$ *where* $0 \leq i \leq 8$, *and* $c = \sum_{i=0}^{8} a_i \cdot b_i$.

c is in the range of $(-2^{15} \cdot 3329)$ *to* $(2^{15} \cdot 3329)$

Proof. Let $a_i = 3328$ and $b_i = 3328$ for $0 \leq i \leq 8$ be the maximum allowed values. Then, $\sum_{i=0}^{8} a_i \cdot b_i = 99680256$. This is very close to $2^{15} \cdot 3329$ which is the maximum value for the input of the Montgomery reduction. In fact, adding another multiplication of coefficients to the sum, i.e., $3328^2 + 99680256 = 110755840$, will exceed $2^{15} \cdot 3329 = 109084672$. □

Merging NTT layers: Merging multiple NTT layers reduces the number of load and store instructions and gives a noticeable performance improvement on the Cortex-M4 [9, 16]. While [9] uses eight registers for eight coefficients and performs three layers of the NTT, [16] uses only four registers for eight coefficients and performs two layers of the NTT without storing and reloading. Both use the remaining registers to store the constants required in the Montgomery and Barrett reductions as well as the loop counter. In this work, we use eight registers to keep 16 coefficients and perform three or four layers of the NTT, depending on the distance between the coefficients being used in the same butterfly on the next layer. In other words, we load coefficients into these eight registers in such a way that a maximum of NTT layers can be performed before storing the results. Thanks to the structure of the NTT used in NEWHOPE and NEWHOPE-COMPACT, we can merge four layers, since at some point, we need coefficients with distance one, and loading consecutive coefficients from memory is free with the `ldr` instruction.

Moreover, KYBER uses seven layers of a 256-point NTT, which is different from NEWHOPE-COMPACT, having seven layers of a 128-point NTT. Consequently, while NEWHOPE-COMPACT has distance one coefficients on the last layer, KYBER requires distance two coefficients. Thus, the last four layers cannot be merged. Therefore, we merged seven layers as 3+3+1 for KYBER.

Although eight registers are used to store coefficients, we can still spare some registers to store the constants required for the Montgomery reductions, specifically q and $-q^{-1}$. However, we have to reload the Barrett constant (line 1 of Algorithm 15) or the Montgomery constant (line 1 of Algorithm 17) at every use, but note that we do not need them heavily thanks to lazy reductions. We follow a different approach for the loop counter, which will be described in the loop unrolling paragraph. Hence, we save more than we lose, i.e., loading more coefficients and performing more layers in every loop is better than loading the Barrett constant only once and keeping it in the register to be used for all Barrett reductions.

Precomputation of twiddle factors: The powers of the NTT constant γ are often referred to as twiddle factors. It is a common approach to precompute all of these twiddle factors in the Montgomery domain and store them in flash memory. In this work, we use the Montgomery factor $\beta = 2^{16}$, similar to [16, 47]. We reorder these constants before storing them in the flash memory to have them appearing in memory in the same order as they are used during the computation. Hence, we can easily load the next one without computing its address. The load instruction on the Cortex-M4 has the ability to move the pointer to the next twiddle factor while fetching the current value from memory. Thus, moving to the next factor has no extra cost. Since the first Montgomery reduction for NEWHOPE and NEWHOPE-COMPACT can be skipped, as mentioned before, the first twiddle factor for these two schemes should not be stored in the Montgomery domain when this optimization is used. Moreover, since we use the Gentleman-Sande butterfly for the NTT^{-1}, we perform the division with n on half of the coefficients during the last butterfly by just multiplying the twiddle factor(s) for the last layer with n^{-1}.

Unrolling: We unroll the outer loop of the NTT and iterate over the layers as usual. [9, 16] spare one register for the loop counter. While [9] uses this loop counter both to check the number of iterations remaining in the loop and to decide which precomputed twiddle factor to use, [16] uses it only to detect when to end the loop. We decided not to spare a register for this loop counter and instead use this freed register to load more coefficients in every loop and merge more NTT layers. Then, we can naively use the `.rept` precompiler directive instead of this loop counter. However, since the `.rept` only repeats the same code, it increases the code size dramatically. Hence, we went back to the loop counter idea again. However, instead of keeping it in a register, we spill it to the stack. Consequently, the code size stays reasonable while we can still load more coefficients in every loop. As an obvious observation, using the `.rept` directive is slightly faster. Hence, it might be useful for some applications where there is plenty of empty memory for storing the code.

Link-time optimization: Link-time optimization is an important option to control optimization, although it may increase the code size. The usual approach without link-time optimization is that each source file is compiled with some optimization level to generate different object files. Then, these optimized object files are linked together to compose an executable file. Although this approach does a good job optimizing source code, it turns out that the linker can perform even better optimization when link-time optimization (`-flto`) is enabled. This option can give a performance boost of around 10%. The critical performance gain is achieved from cross-module function inlining, which is not directly possible without `-flto`. This will tend to increase the code size since inlining functions across source files introduces code duplication. However, it should also be noted that link-time optimization is more effective at identifying unused code or code that has no impact on the output.

The `pqm4` platform does not use `-flto` as a default option since it increases stack consumption or results in a slower computation of some schemes while improving the performance of KYBER [16]. Therefore, we have also tested the effect of `-flto` on performance for our implementations of KYBER, NEWHOPE, and NEWHOPE-COMPACT and realized that they all benefit from it and have a performance boost. However, since assembly-optimized polynomial multiplication was already implemented carefully by inlining all necessary functions such as modular reductions, adding `-flto` has no effect on its performance.

4.3.2 Optimization of NEWHOPE and NEWHOPE-COMPACT for Stack Usage

On embedded devices, RAM usage is often a significant bottleneck. Outside of real-time systems, one can always wait for a slow algorithm, but if the algorithm needs more memory than available on the device, it cannot be used. While the Cortex-M4 on our board offers quite a large amount of memory, we decided to optimize for stack usage as well.

Our goal was to reduce the minimum amount of stack space required to compute the cipher while keeping performance mostly unaffected. While it is possible to follow a more aggressive approach to reduce stack usage, such implementations would be considerably slower than ours. The three main metrics regarding implementation are speed, stack usage, and code size. The one to optimize depends on the context in which the cipher will be used. In our work, we tried to optimize the first two while keeping the last one reasonable.

Key generation: The core of the key generation (Algorithm 4) is the computation of $\hat{b} \leftarrow \hat{a} \circ \mathsf{NTT}(s) + \mathsf{NTT}(e)$. Since each coefficient of the output of the NTT depends on *all* coefficients of the input, all coefficients of s and e must have been generated before proceeding to the addition. Hence, at least two full polynomials should be stored in memory. To reduce the memory usage, we used the observation that polynomial multiplication can be performed on-the-fly in the NTT domain and, likewise, adding an error to a polynomial can be performed on-the-fly in the *normal* domain. Indeed, the operation $\circ$ works sequentially on parts of its inputs (one coefficient at the time for NEWHOPE and four, six, or eight for NEWHOPE-COMPACT depending on the parameter set used) and does not need all of them in memory at the same time. Similarly, each coefficient of the error polynomial can be computed and added separately, but only if the addition considered is in the normal domain. This is why instead of computing

$$\hat{b} \leftarrow \hat{a} \circ \mathsf{NTT}(s) + \mathsf{NTT}(e),$$

we compute

$$\hat{b} \leftarrow \mathsf{NTT}(\mathsf{NTT}^{-1}(\hat{a} \circ \mathsf{NTT}(s)) + e)$$

and perform the multiplication and the addition on-the-fly. This requires one more NTT^{-1} but allows to store only *one* polynomial in memory, containing s and $\hat{b}$ subsequently. This approach reduces stack usage significantly and since our benchmarks show that computing one extra optimized NTT^{-1} only increases the key generation time by around 5%, we believe that this is a good trade-off. This trick can be similarly applied to KYBER. Note that the small *relative* cost of this technique is specific to our context and is mainly due to the fact that hashing is the main performance bottleneck. This issue will be discussed in more detail in Section 4.4. If a faster hash function is used, the decrease in performance will be higher than 5%. That being said, the *absolute* cost of the trick is always the same as one NTT^{-1}.

Encryption: The encryption procedure (Algorithm 5) is mainly driven by the following computations:

1. $\hat{t} \leftarrow \mathsf{NTT}(s')$
2. $\hat{u} \leftarrow \hat{a} \circ \hat{t} + \mathsf{NTT}(e')$
3. $v' \leftarrow \mathsf{NTT}^{-1}(\mathsf{DecodePoly}(\hat{b}') \circ \hat{t}) + e'' + \mathsf{Encode}(\mu)$

The first two yield a situation similar to the key generation but unfortunately require two polynomials on the stack frame. Indeed, since $\hat{t}$ appears in the second and the last computation, the result of $\hat{a} \circ \hat{t}$ *cannot* be stored in the same memory space as $\hat{t}$ (and since it would need to go through a NTT^{-1}, it does need to be fully stored). Once $\hat{u}$ is computed, it can be packed in the ciphertext and free one of the two polynomials. The last computation is quite friendly for stack usage. Since both the base multiplication and the addition operate on small portions of the polynomial, $e'' + \mathsf{Encode}(\mu)$ can be computed coefficient-wise and $\hat{b}$ can be partially unpacked from the inputs. Thus, Line 3 could technically be computed with one polynomial plus a small overhead in the stack frame. Since two polynomials were already allocated previously and only maximal stack usage is relevant, we actually fully unpack $\hat{b}$. Finally, the stack usage is bigger than the one of the key generation because of the extra polynomial stored.

Decryption: The decryption of NEWHOPE (Algorithm 6) is quite lightweight in terms of stack usage. Unfortunately, the algorithms introduced in the preliminaries are the CPA version of the cipher. Since the CCA transform runs the encryption procedure during decryption, the stack usage is essentially the same as for encryption.

4.3.3 Trade-offs Between Secret Key Size and Speed

There are different trade-offs between the secret key size and the performance of the scheme. [12, 14] proposed to store the seed used for all randomness in the key generation if the size of the secret key is critical. However, this requires to perform key generation again during decapsulation and gives a significant performance penalty. As also stated by [12, 14], another optimization could be storing the secret key in the normal domain instead of the NTT domain. Hence, each coefficient can be compressed to 3 bits, since their possible values are in between -2 and 2. Note that our NTT implementation is fast on the Cortex-M4, so we decided that such optimizations are good trade-offs for this platform. We also observed that sampling the secret polynomial s is fast enough, although it is one of the most time-consuming parts of such algorithms is the hashing used for the randomness generation. Note that sampling the secret key from the centered binomial distribution is lightweight in comparison to the generation of the public parameter a since we can extract two coefficients from only one byte by using the centered binomial distribution, while uniform sampling needs two bytes to extract only one coefficient. Hence, we decided to store only the 32-byte secret-key seed. Then, the secret key is sampled and transformed into the NTT domain again during decapsulation. These operations reduce the secret key size by 736 bytes for KYBER512 and NEWHOPE-COMPACT512, 864 bytes for NEWHOPE512, 1120 bytes for KYBER768 and NEWHOPE-COMPACT768, 1504 bytes for KYBER1024 and NEWHOPE-COMPACT1024, and 1760 bytes for NEWHOPE1024. However, they increase the decapsulation time by around 7% for KYBER, 9% for NEWHOPE-COMPACT, and 18% for NEWHOPE, while decreasing the key generation time slightly.

4.4 Results and Comparison

Our optimizations were implemented in the three sibling schemes NEWHOPE, NEWHOPE-COMPACT, and KYBER. Comparing different schemes across parameter sets is often complicated because performance is always strongly correlated with the targeted security level. Most of the implemented schemes propose parameter sets for NIST security levels 1, 3, and 5, which correspond to 128, 192, and 256 bits of security. Fortunately, since all the schemes involved in our tests are similar and based on {R,M}LWE, the dimension of the underlying lattice problem can be roughly translated into NIST security levels. Hence, we compare them for dimensions 512, 768 (if available), and 1024, which correspond to the three aforementioned security levels.

4.4.1 Speed Comparison

The results of our benchmarks in terms of speed can be found in Table 4.2. The code was compiled and run in the same conditions as the schemes benchmarked in `pqm4` [29]. We use `arm-none-eabi-gcc` release (version 9.2.1). We compare the two candidates NEWHOPE and KYBER against their previous Cortex-M4 optimized implementations available in `pqm4` and also add the newcomer NEWHOPE-COMPACT. One can see that NEWHOPE and KYBER perform around 10% better with our optimizations while using less stack space (see Table 4.3). Furthermore, NEWHOPE-COMPACT is more than 40% faster compared to NEWHOPE and more than 25% faster compared to KYBER for all security levels. This is explained by the two following observations:

- NEWHOPE-COMPACT is a variant of NEWHOPE using a smaller modulus and distribution, which means an increased performance during polynomial multiplication because of lazy reductions and less hashing needed to sample from the error distribution.
- NEWHOPE-COMPACT is based on RLWE, while KYBER is based on MLWE. Hence, even though they share similar parameter sets, the inherent performance penalty of using the less structured version of LWE hurts KYBER.

Moreover, some design decisions affect the performance or the stack usage of the scheme. These design decisions are unrolling NTT loops by using the `.rept` directive instead of the loop counter (Section 4.3.1), optimizing the stack usage (Section 4.3.2), and the trade-off between the size of the secret key and performance (Section 4.3.3). These three decisions can be easily enabled or disabled. The effects of the last two choices on the performance and the stack usage are given in Table 4.2. Using the `.rept` directive instead of the loop counter decreases the cycle count by n per NTT call, where n is the degree of the polynomial for the selected parameters. However, the code size increases by a factor of 10 to 50. Optimizing the stack usage decreases the memory used by key generation while increasing the cycle count of the same function. Finally, applying the method described in Section 4.3.3 to reduce the size of the secret key increases the cycle count of the decapsulation while decreasing it for the key generation.

4.4.2 Dominance of Hashing

The speed difference shown in Table 4.2 might look slim at first sight. This is due to the fact that, as pointed out by previous works, those schemes have been optimized so much that the bottleneck is now the generation of random numbers through hashing instead of the polynomial multiplication procedure. Table 4.4 shows the time spent hashing for all algorithms and parameter sets. As we can see, with a minimum of 66% for the decapsulation of NEWHOPE-COMPACT, all the algorithms are severely dominated by hashing. Even if polynomial multiplications were somehow instantaneous, the results of Table 4.2 would be somewhat similar.

4.4.3 Comparing Polynomial Multiplications

The reader might wonder why to bother optimizing polynomial multiplications further if it is not the bottleneck anymore. The reason is twofold: First, `Keccak` (`SHA-3`) is used to expand the seed in every scheme implemented. However, the choice of the seed expansion algorithm is somewhat orthogonal to the scheme and does not affect post-quantum assumptions. Hence, using a faster hash function would reduce the impact of hashing on the performance. Furthermore, it might be unnecessary to use a cryptographic hash function to generate the public parameters. For instance, [15] uses a faster, non-cryptographic RNG to speed-up a scheme based on LWE. Second, even if `Keccak` is used since its usage will likely grow in all future cryptographic applications, we might eventually see hardware acceleration for it on a lot of architectures. This would naturally drastically decrease the time spent hashing in our schemes and make the polynomial multiplication the most important optimization target again. Recall that, as stated in Section 4.3.2, this would increase the relative cost of the reduced stack usage trick used in the key generation. Nevertheless, we think that outside of unrealistically fast polynomial generation, the trade-off can still be useful.

Since our work is the first Cortex-M4 implementation of NEWHOPE-COMPACT, we do not have any point of comparison for our technique for this scheme. Table 4.5 shows the speed-up for the dimension of the NTT used in all parameter sets of NEWHOPE and KYBER and the cycle count of all subroutines of the polynomial multiplication for each algorithm and dimension. The total cost of multiplication operations for each scheme is presented in Table 4.6. This table was obtained by summing all the time spent in the three multiplication subroutines: NTT, NTT^{-1}, and $\circ$. Note that the stack optimized version of our implementation is used in this table to show its actual impact on the performance. It can be seen that KYBER and NEWHOPE-COMPACT have similar performance, while NEWHOPE is slower. This is mainly due to the extra layers of the NTT and the increased number of reductions caused by the larger modulus. Note that our NTT^{-1} cycles do not include any bitreversal operation because we need NTT^{-1} to output a bitreversed order for the stack optimization (Section 4.3.2). To be able to verify test vectors with the reference implementation of NEWHOPE, we have implemented a separate bitreversal operation that takes roughly $4n$-cycle for the selected parameter set, which is not included in the NTT^{-1}. It can be seen that our implementation of the NEWHOPE NTT is slightly slower compared to the implementation from [29], while we have noticeably better performance for the NTT^{-1}. Table 4.6 shows that even though we have a slower NTT, the total number of cycles spent in polynomial operations is reduced compared to [29].

4.5 Conclusion

In this chapter, we proposed several optimizations for {R,M}LWE schemes on the Cortex-M4. Among them, some are direct improvements over the current literature, while others are trade-offs that are up to the user of the scheme to deem needed or not. The core speed optimizations are due to a more aggressive layer-merging strategy and a minimization of the number of reductions in the base multiplication. Our implementation has already been integrated into the `pqm4` library. We also provide a comparison of the proposed trade-offs. The code is written in a modular fashion that allows the user to switch between versions easily.

Our results show that all optimization techniques have advantages and disadvantages and might be useful for different applications. Note that our default option includes only the stack usage optimization since our goal is having a fast implementation while using a stack space as small as possible. One interesting point is that our NTT implementation for NEWHOPE has slightly slower performance than the one reported in [29], suggesting that there is still room for improvement.

The time spent during polynomial operations has two main bottlenecks, namely modular reduction and memory access operations among layers of the NTT. This chapter solves these bottlenecks by proposing a 2-cycle implementation of the Montgomery modular reduction and by increasing the number of merged layers. While this chapter focuses on this strategy, keeping coefficients in 16-bit signed integers requires performing modular reduction regularly. It would be interesting to see if using 32-bit signed integers with proper modular reduction can improve the performance. This approach would require to store one coefficient per register, which means that fewer layers can be merged. However, the number of modular reductions might be reduced since lazy reduction can be applied more aggressively. Such an implementation was recently proposed for the RISC-V architecture [8]. The authors propose an implementation which uses the Barrett reduction after both addition/subtraction and multiplication. Their reduction implementation can reduce 32-bit numbers, so it allows more aggressive use of lazy reduction at the cost of using more registers. The Cortex-M4 has a 32-bit multiplier. Hence, it might be interesting to evaluate the performance of that implementation on the Cortex-M4.

Table 4.2: Cycle count comparison for the {R,M}LWE schemes improved by our work. **G**: key generation, **E**: encapsulation, **D**: decapsulation.

Scheme		**Previous work**	**This work Speed**	**This work Stack Opt.**	**This work Short sk**
NEWHOPE	512	**G**: 588 683 [a] **E**: 918 558 [a] **D**: 904 800 [a]	**G**: 561 161 **E**: 865 243 **D**: 820 130	**G**: 578 850 **E**: 865 856 **D**: 820 742	**G**: 555 625 **E**: 865 018 **D**: 968 961
	1024	**G**: 1 161 112 [a] **E**: 1 777 918 [a] **D**: 1 760 470 [a]	**G**: 1 117 398 **E**: 1 687 272 **D**: 1 612 960	**G**: 1 157 331 **E**: 1 689 375 **D**: 1 614 804	**G**: 1 106 350 **E**: 1 686 964 **D**: 1 918 747
NH-CMPCT	512	-	**G**: 335 991 **E**: 531 453 **D**: 484 416	**G**: 349 692 **E**: 532 423 **D**: 484 945	**G**: 330 826 **E**: 531 282 **D**: 526 805
	768	-	**G**: 501 885 **E**: 782 315 **D**: 717 250	**G**: 524 181 **E**: 784 117 **D**: 718 950	**G**: 494 364 **E**: 782 471 **D**: 786 664
	1024	-	**G**: 658 581 **E**: 1 022 903 **D**: 940 023	**G**: 686 225 **E**: 1 025 503 **D**: 941 076	**G**: 648 206 **E**: 1 022 773 **D**: 1 030 809
KYBER	512	**G**: 514 291 [b] **E**: 652 769 [b] **D**: 621 245 [b]	**G**: 452 919 **E**: 586 380 **D**: 542 576	**G**: 461 693 **E**: 586 754 **D**: 543 332	**G**: 446 876 **E**: 586 403 **D**: 579 594
	768	**G**: 976 757 [b] **E**: 1 146 556 [b] **D**: 1 094 849 [b]	**G**: 860 227 **E**: 1 031 603 **D**: 967 124	**G**: 872 140 **E**: 1 030 764 **D**: 966 848	**G**: 850 228 **E**: 1 030 679 **D**: 1 021 439
	1024	**G**: 1 575 052 [b] **E**: 1 779 848 [b] **D**: 1 709 348 [b]	**G**: 1 394 148 **E**: 1 603 776 **D**: 1 522 900	**G**: 1 410 591 **E**: 1 603 988 **D**: 1 523 175	**G**: 1 381 514 **E**: 1 603 772 **D**: 1 595 530

[a] [29], `https://github.com/mupq/pqm4/`, commit `be0c421`.
[b] [16]

Table 4.3: Stack usage comparison for the {R,M}LWE schemes improved by our work. **G**: key generation, **E**: encapsulation, **D**: decapsulation.

Scheme	NEWHOPE **(This work)**	NEWHOPE **[29][a]**	NH-CMPCT **(This work)**	KYBER **(This work)**	KYBER **[16]**
512	**G**: 2 056 **E**: 2 864 **D**: 2 880	**G**: 5 960 **E**: 9 168 **D**: 10 296	**G**: 2 160 **E**: 2 984 **D**: 2 984	**G**: 2 392 **E**: 2 344 **D**: 2 360	**G**: 2 952 **E**: 2 552 **D**: 2 560
768	-	-	**G**: 2 600 **E**: 3 936 **D**: 3 936	**G**: 3 240 **E**: 2 856 **D**: 2 864	**G**: 3 848 **E**: 3 128 **D**: 3 072
1024	**G**: 3 072 **E**: 4 904 **D**: 4 920	**G**: 11 080 **E**: 17 360 **D**: 19 576	**G**: 3 176 **E**: 5 024 **D**: 5 024	**G**: 3 776 **E**: 3 744 **D**: 3 760	**G**: 4 360 **E**: 3 584 **D**: 3 592

[a] `https://github.com/mupq/pqm4/`, commit `be0c421`.

Table 4.4: Time spent hashing. **G**: key generation, **E**: encapsulation, **D**: decapsulation.

Scheme	Dimension 512	Dimension 768	Dimension 1024
NEWHOPE	**G**: 75% **E**: 80% **D**: 72%	-	**G**: 73% **E**: 78% **D**: 71%
NEWHOPE-COMPACT	**G**: 75% **E**: 78% **D**: 67%	**G**: 72% **E**: 77% **D**: 66%	**G**: 73% **E**: 77% **D**: 66%
KYBER	**G**: 76% **E**: 80% **D**: 69%	**G**: 77% **E**: 80% **D**: 72%	**G**: 78% **E**: 80% **D**: 73%

Table 4.5: Comparison of the polynomial multiplication functions of all the schemes. Kyber actually uses the exact same NTT code for all dimensions.

Scheme	Dimension	NTT	NTT^{-1}	$\circ$
NEWHOPE	512	28662	22836	4736
	512 ([29][a])	29767	35813	5459
	1024	63387	49880	9396
	1024 ([29][a])	59752	71942	10836
	1024 ([9])	86769	97340	14977
NEWHOPE-COMPACT	512	12799	13052	7052
	768	19647	21226	12749
	1024	25536	26039	18510
KYBER	256	6847	6975	2317
	256 ([16])	7754	9377	3076

[a] `https://github.com/mupq/pqm4/`, commit `be0c421`.

Table 4.6: Total time spent in polynomial multiplication subroutines (NTT, NTT^{-1}, and $\circ$).

Scheme	Dimension	KeyGen	Encaps	Decaps
NEWHOPE	512	84896	89632	117204
	512 ([29][a])	64993	106265	147537
	1024	186050	195446	254722
	1024 ([29][a])	130340	213118	295896
NEWHOPE-COMPACT	512	45702	52754	72858
	768	73269	86018	119993
	1024	95621	114131	158680
KYBER	512	50606	48521	73824
	512 ([16])	43320	62095	93132
	768	82860	76245	110712
	768 ([16])	74208	97682	139549
	1024	119748	108603	152234
	1024 ([16])	111248	139421	192118

[a] `https://github.com/mupq/pqm4/`, commit `be0c421`.

CHAPTER 5

CONCLUSION

NIST initiated a post-quantum standardization project in 2017 in order to protect our information against adversaries equipped with quantum computers. This standardization includes both KEM and digital signatures. In this thesis, we have focused on the efficient implementation of post-quantum secure lattice-based KEMs, specifically NEWHOPE and KYBER.

The contributions of this thesis can be summarized as follows:

Chapter 3 presents a fast, compact and simple variant of NEWHOPE called as NEWHOPE-COMPACT. It makes use of the advantages of both NEWHOPE and KYBER. The prime of NEWHOPE is reduced to 3329 from 12289 by following the techniques used by [12, 35, 51]. Therefore, while the sizes of secret and public keys are almost as small as KYBER's, the security is based on the hardness of solving the RLWE problem instead of the MLWE problem. Consequently, instead of working with matrices or vectors of polynomials, we only deal with polynomials. Moreover, NEWHOPE has 1280 bytes ($n = 512$) or 2560 bytes ($n = 1024$) precomputed values, and these values are different while NEWHOPE-COMPACT has only 256 bytes in total and they are the same for both ($n = 512$) and ($n = 1024$). There is also no need for a bit-reversal table as in NEWHOPE whose size is 512 bytes ($n = 512$) or 1024 bytes ($n = 1024$). Therefore, even for constrained devices switching the security level is easy.

Chapter 4 describes an optimized Cortex-M4 implementation of NEWHOPE, KYBER, and a recently proposed variant of those schemes called NEWHOPE-COMPACT. We present various optimizations, mainly in terms of speed and stack usage on the ARM microcontroller. Since those schemes share structural similarities, general improvements are applicable to all of them. Our implementation outperforms the current state-of-the-art for KYBER and NEWHOPE while giving a unified framework to compare the three schemes, as they use the same level of optimization, which was not the case in previous works [29]. Chapter 4 contributions are listed as follows:

- We propose a 2-cycle modular reduction implementation for the Montgomery arithmetic, which translates subtraction to addition to allow the use of special instructions.
- We show that small-degree polynomial multiplications can be implemented efficiently by using lazy-reduction techniques. Hence, we show that early termination of the NTT can be implemented more efficiently when the base multiplication has a degree higher than 2.
- We show that even though the target architecture has only 14 usable registers, 16 coefficients can be used during the butterfly layers. This allowed us to merge up to four layers of the NTT and reduce the number of load and store instructions.
- We give several trade-offs between speed and other metrics. We show that the stack usage of key generation can be reduced by adding the error vector on-the-fly at the cost of an extra NTT computation. We also implement a well-known idea to store the secret key as a seed and re-expand it during decapsulation.

REFERENCES

[1] G. Alagic, J. Alperin-Sheriff, D. Apon, D. Cooper, Q. Dang, J. Kelsey, Y.-K. Liu, C. Miller, D. Moody, R. Peralta, R. Perlner, A. Robinson, and D. Smith-Tone, Status Report on the Second Round of the NIST Post-Quantum Cryptography Standardization Process, National Institute for Standards and Technology Internal Report 8240, 2019.

[2] G. Alagic, J. Alperin-Sheriff, D. Apon, D. Cooper, Q. Dang, Y.-K. Liu, C. Miller, D. Moody, R. Peralta, R. Perlner, A. Robinson, and D. Smith-Tone, Status Report on the First Round of the NIST Post-Quantum Cryptography Standardization Process, National Institute for Standards and Technology Internal Report 8240, 2019.

[3] E. Alkim, R. Avanzi, J. Bos, L. Ducas, A. de la Piedra, T. Pöppelmann, P. Schwabe, D. Stebila, M. R. Albrecht, E. Orsini, V. Osheter, K. G. Paterson, G. Peer, and N. P. Smart, NewHope - Algorithm Specifications And Supporting Documentation (version 1.03), Technical report, NIST Post-Quantum Cryptography Standardization Project, 2019.

[4] E. Alkim, Y. A. Bilgin, and M. Cenk, Compact and simple RLWE based key encapsulation mechanism, in P. Schwabe and N. Thériault, editors, *Progress in Cryptology – LATINCRYPT 2019: 6th International Conference on Cryptology and Information Security in Latin America*, volume 11774 of *Lecture Notes in Computer Science*, pp. 237–256, Springer, Heidelberg, Germany, Santiago de Chile, Chile, October 2–4 2019.

[5] E. Alkim, Y. A. Bilgin, M. Cenk, and F. Gérard, Cortex-M4 optimizations for {R,M}LWE schemes, IACR Transactions on Cryptographic Hardware and Embedded Systems, 2020(3), pp. 336–357, 2020.

[6] E. Alkim, L. Ducas, T. Pöppelmann, and P. Schwabe, NewHope without reconciliation, Cryptology ePrint Archive, Report 2016/1157, 2016, `http://eprint.iacr.org/2016/1157`.

[7] E. Alkim, L. Ducas, T. Pöppelmann, and P. Schwabe, Post-quantum key exchange - A new hope, in T. Holz and S. Savage, editors, *USENIX Security 2016: 25th USENIX Security Symposium*, pp. 327–343, USENIX Association, Austin, TX, USA, August 10–12 2016.

[8] E. Alkim, H. Evkan, N. Lahr, R. Niederhagen, and R. Petri, ISA extensions for finite field arithmetic - accelerating Kyber and NewHope on RISC-V, IACR Transactions on Cryptographic Hardware and Embedded Systems, 2020(3), pp. 219–242, 2020.

[9] E. Alkim, P. Jakubeit, and P. Schwabe, Newhope on ARM cortex-m, in C. Carlet, M. A. Hasan, and V. Saraswat, editors, *Security, Privacy, and Applied Cryptography Engineering - 6th International Conference, SPACE 2016, Hyderabad, India, December 14-18, 2016, Proceedings*, volume 10076 of *Lecture Notes in Computer Science*, pp. 332–349, Springer, 2016.

[10] ARM, Arm cortex-m4, `https://www.arm.com/products/silicon-ip-cpu/cortex-m/cortex-m4`.

[11] F. Arute, K. Arya, R. Babbush, D. Bacon, J. C. Bardin, R. Barends, R. Biswas, S. Boixo, F. G. S. L. Brandao, D. A. Buell, B. Burkett, Y. Chen, Z. Chen, B. Chiaro, R. Collins, W. Courtney, A. Dunsworth, E. Farhi, B. Foxen, A. Fowler, C. Gidney, M. Giustina, R. Graff, K. Guerin, S. Habegger, M. P. Harrigan, M. J. Hartmann, A. Ho, M. Hoffmann, T. Huang, T. S. Humble, S. V. Isakov, E. Jeffrey, Z. Jiang, D. Kafri, K. Kechedzhi, J. Kelly, P. V. Klimov, S. Knysh, A. Korotkov, F. Kostritsa, D. Landhuis, M. Lindmark, E. Lucero, D. Lyakh, S. Mandrà, J. R. McClean, M. McEwen, A. Megrant, X. Mi, K. Michielsen, M. Mohseni, J. Mutus, O. Naaman, M. Neeley, C. Neill, M. Y. Niu, E. Ostby, A. Petukhov, J. C. Platt, C. Quintana, E. G. Rieffel, P. Roushan, N. C. Rubin, D. Sank, K. J. Satzinger, V. Smelyanskiy, K. J. Sung, M. D. Trevithick, A. Vainsencher, B. Villalonga, T. White, Z. J. Yao, P. Yeh, A. Zalcman, H. Neven, and J. M. Martinis, Quantum supremacy using a programmable superconducting processor, Nature, 574(7779), pp. 505–510, Oct 2019.

[12] R. Avanzi, J. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. M. Schanck, P. Schwabe, G. Seiler, and D. Stehlé, CRYSTALS-KYBER - Algorithm Specifications And Supporting Documentation (version 2.0), Technical report, NIST Post-Quantum Cryptography Standardization Project, 2019.

[13] P. Barrett, Implementing the rivest shamir and adleman public key encryption algorithm on a standard digital signal processor, in A. M. Odlyzko, editor, *Advances in Cryptology - CRYPTO '86, Santa Barbara, California, USA, 1986, Proceedings*, volume 263 of *Lecture Notes in Computer Science*, pp. 311–323, Springer, 1986.

[14] 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 *2018 IEEE European Symposium on Security and Privacy, EuroS&P 2018, London, United Kingdom, April 24-26, 2018*, pp. 353–367, IEEE, 2018.

[15] J. W. Bos, S. Friedberger, M. Martinoli, E. Oswald, and M. Stam, Fly, you fool! faster frodo for the arm cortex-m4, Cryptology ePrint Archive, Report 2018/1116, 2018, `https://eprint.iacr.org/2018/1116`.

[16] L. Botros, M. J. Kannwischer, and P. Schwabe, Memory-efficient high-speed implementation of Kyber on cortex-M4, in J. Buchmann, A. Nitaj, and T. eddine Rachidi, editors, *AFRICACRYPT 19: 11th International Conference on Cryptology in Africa*, volume 11627 of *Lecture Notes in Computer Science*, pp. 209–228, Springer, Heidelberg, Germany, Rabat, Morocco, July 9–11 2019.

[17] J. W. Cooley and J. W. Tukey, An algorithm for the machine calculation of complex fourier series, Mathematics of Computation, 19(90), pp. 297–301, 1965.

[18] W. Diffie and M. E. Hellman, New directions in cryptography, IEEE Trans. Inf. Theory, 22(6), pp. 644–654, 1976.

[19] R. Feynman, Simulating physics with computers, International Journal of Theoretical Physics, 21, pp. 467–488, 1982.

[20] N. I. for Standards and Technology, Submission requirements and evaluation criteria for the postquantum cryptography standardization process. official call for proposals, NIST Post-Quantum Cryptography Standardization Process, December, 2016, `http://csrc.nist.gov/groups/ST/post-quantum-crypto/documents/call-for-proposals-final-dec-2016.pdf`.

[21] E. Fujisaki and T. Okamoto, Secure integration of asymmetric and symmetric encryption schemes, in M. J. Wiener, editor, *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, volume 1666 of *Lecture Notes in Computer Science*, pp. 537–554, Springer, 1999.

[22] W. M. Gentleman and G. Sande, Fast fourier transforms: For fun and profit, in *Proceedings of the November 7-10, 1966, Fall Joint Computer Conference*, AFIPS '66 (Fall), pp. 563–578, ACM, New York, NY, USA, 1966.

[23] L. K. Grover, A fast quantum mechanical algorithm for database search, in G. L. Miller, editor, *Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing, Philadelphia, Pennsylvania, USA, May 22-24, 1996*, pp. 212–219, ACM, 1996.

[24] T. Güneysu, T. Oder, T. Pöppelmann, and P. Schwabe, Software speed records for lattice-based signatures, in P. Gaborit, editor, *Post-Quantum Cryptography - 5th International Workshop, PQCrypto 2013, Limoges, France, June 4-7, 2013. Proceedings*, volume 7932 of *Lecture Notes in Computer Science*, pp. 67–82, Springer, 2013.

[25] M. Hamburg, Three Bears, Technical report, National Institute of Standards and Technology, 2019, available at `https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions`.

[26] M. B. İlter and M. Cenk, Efficient big integer multiplication in cryptography, International Journal of Information Security Science, 6(4), pp. 70–78, 2017.

[27] D. J. Bernstein, Multidigit multiplication for mathematicians, Advances in Applied Mathematics, pp. 1–19, 09 2001.

[28] M. J. Kannwischer, J. Rijneveld, and P. Schwabe, Faster multiplication in $\mathbb{Z}_{2^m}[x]$ on cortex-M4 to speed up NIST PQC candidates, in R. H. Deng, V. Gauthier-Umaña, M. Ochoa, and M. Yung, editors, *ACNS 19: 17th International Conference on Applied Cryptography and Network Security*, volume 11464 of *Lecture Notes in Computer Science*, pp. 281–301, Springer, Heidelberg, Germany, Bogota, Colombia, June 5–7 2019.

[29] M. J. Kannwischer, J. Rijneveld, P. Schwabe, and K. Stoffelen, PQM4: Post-quantum crypto library for the ARM Cortex-M4, `https://github.com/mupq/pqm4`.

[30] A. Karmakar, J. M. B. Mera, S. S. Roy, and I. Verbauwhede, Saber on ARM cca-secure module lattice-based key encapsulation on ARM, IACR Transactions on Cryptographic Hardware and Embedded Systems, 2018(3), pp. 243–266, 2018.

[31] A. Langlois and D. Stehlé, Worst-case to average-case reductions for module lattices, Des. Codes Cryptogr., 75(3), pp. 565–599, 2015.

[32] P. Longa and M. Naehrig, Speeding up the number theoretic transform for faster ideal lattice-based cryptography, in S. Foresti and G. Persiano, editors, *Cryptology and Network Security - 15th International Conference, CANS 2016, Milan, Italy, November 14-16, 2016, Proceedings*, volume 10052 of *Lecture Notes in Computer Science*, pp. 124–139, 2016.

[33] V. Lyubashevsky, C. Peikert, and O. Regev, On ideal lattices and learning with errors over rings, in H. Gilbert, editor, *Advances in Cryptology - EUROCRYPT 2010, 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Monaco / French Riviera, May 30 - June 3, 2010. Proceedings*, volume 6110 of *Lecture Notes in Computer Science*, pp. 1–23, Springer, 2010.

[34] V. Lyubashevsky, C. Peikert, and O. Regev, On ideal lattices and learning with errors over rings, J. ACM, 60(6), pp. 43:1–43:35, 2013.

[35] V. Lyubashevsky and G. Seiler, NTTRU: Truly fast ntru using NTT, IACR Transactions on Cryptographic Hardware and Embedded Systems, 2019(3), pp. 180–201, 2019.

[36] R. T. Moenck, Practical fast polynomial multiplication, in *Proceedings of the Third ACM Symposium on Symbolic and Algebraic Computation*, SYMSAC '76, pp. 136–148, 1976.

[37] P. L. Montgomery, Modular multiplication without trial division, Mathematics of Computation, 44(170), pp. 519–521, 1985.

[38] D. Moody, The 2^{nd} round of the NIST PQC standardization process, The 2^{nd} NIST PQC Standardization Workshop, August 2019, `https://csrc.nist.gov/CSRC/media/Presentations/the-2nd-round-of-the-nist-pqc-standardization-proc/images-media/moody-opening-remarks.pdf`.

[39] D. Moody, Post-quantum cryptography: NIST's plan for the future, PQCrypto 2016 Conference, 23-26 February 2016, Fukuoka, Japan, February 2016, `https://pqcrypto2016.jp/data/pqc2016_nist_announcement.pdf`.

[40] D. Moody, Round 2 of the NIST PQC "competition" what was NIST thinking?, PQCrypto 2019 Conference, 8-10 May 2019, Chongqing, China, May 2019, `https://csrc.nist.gov/CSRC/media/Presentations/Round-2-of-the-NIST-PQC-Competition-What-was-NIST/images-media/pqcrypto-may2019-moody.pdf`.

[41] G. E. Moore, Progress in digital integrated electronics [technical literaiture, copyright 1975 ieee. reprinted, with permission. technical digest. international electron devices meeting, ieee, 1975, pp. 11-13.], IEEE Solid-State Circuits Society Newsletter, 11(3), pp. 36–37, 2006.

[42] M. Mosca, Cybersecurity in an era with quantum computers: Will we be ready?, IEEE Security Privacy, 16(5), pp. 38–41, 2018.

[43] M. A. Nielsen and I. L. Chuang, *Introduction and overview*, p. 13–13, Cambridge University Press, 2010.

[44] E. Pednault, J. A. Gunnels, G. Nannicini, L. Horesh, and R. Wisnieff, Leveraging secondary storage to simulate deep 54-qubit sycamore circuits, 2019.

[45] R. L. Rivest, A. Shamir, and L. M. Adleman, A method for obtaining digital signatures and public-key cryptosystems, Commun. ACM, 21(2), pp. 120–126, 1978.

[46] M. O. Saarinen, S. Bhattacharya, Ó. García-Morchón, R. Rietman, L. Tolhuizen, and Z. Zhang, Shorter messages and faster post-quantum encryption with round5 on cortex M, in B. Bilgin and J. Fischer, editors, *Smart Card Research and Advanced Applications, 17th International Conference, CARDIS 2018, Montpellier, France, November 12-14, 2018, Revised Selected Papers.*,

volume 11389 of *Lecture Notes in Computer Science*, pp. 95–110, Springer, 2018.

[47] G. Seiler, Faster AVX2 optimized NTT multiplication for ring-LWE lattice cryptography, Cryptology ePrint Archive, Report 2018/039, 2018, `https://eprint.iacr.org/2018/039`.

[48] P. W. Shor, Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer, SIAM J. Comput., 26(5), pp. 1484–1509, 1997.

[49] STMicroelectronics, STM32F4DISCOVERY kit, `https://www.st.com/en/evaluation-tools/stm32f4discovery.html`.

[50] A. Weimerskirch and C. Paar, Generalizations of the karatsuba algorithm for efficient implementations, IACR Cryptology ePrint Archive, 2006, p. 224, 2006, `http://eprint.iacr.org/2006/224`.

[51] S. Zhou, H. Xue, D. Zhang, K. Wang, X. Lu, B. Li, and J. He, Preprocess-then-ntt technique and its applications to kyber and newhope, in F. Guo, X. Huang, and M. Yung, editors, *Information Security and Cryptology - 14th International Conference, Inscrypt 2018, Fuzhou, China, December 14-17, 2018, Revised Selected Papers*, volume 11449 of *Lecture Notes in Computer Science*, pp. 117–137, Springer, 2018.

CURRICULUM VITAE

PERSONAL INFORMATION

Surname, Name: Bilgin, Yusuf Alper
Nationality: Turkish
Date of Birth: 30.01.1990
e-mail: y.alperbilgin@gmail.com

EDUCATION

Degree	Institution	Year of Graduation
M.S.	METU - Electrical & Electronics Engineering	2016
B.S.	METU - Electrical & Electronics Engineering	2012

PROFESSIONAL EXPERIENCE

Year	Place	Enrollment
06.2015-	Aselsan Inc.	Software Engineer
10.2012-06.2015	Anketek Inc.	R&D Engineer

PUBLICATIONS

- Erdem Alkim, Yusuf Alper Bilgin, Murat Cenk, and François Gérard. Cortex-M4 optimizations for {R,M}LWE schemes. *IACR Transactions on Cryptographic Hardware and Embedded Systems — TCHES*, 2020(3):336–357, 2020. `https://doi.org/10.13154/tches.v2020.i3.336-357`.

- Erdem Alkim, Yusuf Alper Bilgin, and Murat Cenk. Compact and simple RLWE based key encapsulation mechanism. In Peter Schwabe and Nicolas Thériault, editors, *Progress in Cryptology – LATINCRYPT 2019: 6th International Conference on Cryptology and Information Security in Latin America, volume 11774 of Lecture Notes in Computer Science*, pages 237–256, Santiago de Chile, Chile, 2019. Springer, Heidelberg, Germany. `https://doi.org/10.1007/978-3-030-30530-7_12`.

- Berkin Aksoy, Yusuf Alper Bilgin, Murat Cenk, Murat Burhan İlter, Neşe Koçak, Yunus Emre Yılmaz. Analyzing NIST 2nd-round Lattice-based Post-quantum KEM Algorithms, *Information Security and Cryptology Conference*, Ankara, Turkey, 2019

- Yusuf Alper Bilgin and Ali Özgür Yılmaz. Neighbor discovery in network with directional antennas, *2013 21st Signal Processing and Communications Applications Conference (SIU)*, Haspolat, 2013, pp. 1-4, `https://doi.org/10.1109/SIU.2013.6531286`.