2023년 2월
박사학위논문

NIST 표준 양자내성암호의 NTT 기반 효율적 다항식 알고리즘 구현 연구

조 선 대 학 교 대 학 원

정보통신공학과

김 광 식

NIST 표준 양자내성암호의 NTT 기반 효율적 다항식 알고리즘 구현 연구

Research on NTT-based Efficient Polynomial Algorithm Implementation of NIST Standard Quantum Resistance Cryptography

2023년 2월 24일

조선대학교 대학원

정보통신공학과

김 광 식

NIST 표준 양자내성암호의 NTT 기반 효율적 다항식 알고리즘 구현 연구

지도교수 김 영 식

이 논문을 공학박사학위신청 논문으로 제출함.

2022년 10월

조선대학교 대학원

정보통신공학과

김 광 식

김광식의 박사학위 논문을 인준함

위원장 조선대학교 교수 권구락 (인)

위 원 조선대학교 교수 이범식 (인)

위 원 조선대학교 교수 김찬기 (인)

위 원 조선대학교 교수 김영식 (인)

위 원 전남대학교 교수 박호성 (인)

2023년 1월

조선대학교 대학원

목 차

그 림 목 차

표 목 차

ABSTRACT

Research on NTT-based Efficient Polynomial Algorithm Implementation of NIST Standard Quantum Resistance Cryptography

Gwang-Sik Kim
Advisor : Prof. Young-Sik Kim, Ph.D.
Department of Information and Communication Engineering
Graduate School of Chosun University

Today, research on quantum computers is being actively carried out by world-renowned conglomerates such as Intel, IBM, Microsoft, and Google. However, due to the development of quantum computers, public key encryption algorithms such as RSA and elliptic curve cryptography (ECC), which are currently used internationally, need urgent replacement due to the well-known polynomial-time cryptanalysis on quantum computers.

For this purpose, the National Institute of Standards and Technology (NIST) of the United States has managed a competition to select standard post-quantum algorithms after examining and evaluating over 69 candidates for the last five years.

In this thesis proposed an efficient finite field operation method of the Rainbow algorithm, which was adopted in the NIST Round3 candidate, and a method to reduce the Montgomery multiplication operation of the Number Theoretic Transform used for high-speed operation in lattice-based operation.

I. 서 론

A. 연구 배경

오늘 날 컴퓨팅 기술의 발전으로 초고속 연산이 가능한 양자컴퓨터가 날이 갈수록 발전하고 있다. 이를 위해 세계적인 대기업인 인텔과 구글 등에서 양자컴퓨터에 대한 연구가 아주 활발하게 이루어지고 있으며 세계적인 관심을 받고 있다. 2019년 10월 구글에선 과학 저널 네이처에 양자 우월성 달성을 증명한 논문을 발표하게 되면서, 근 미래 양자컴퓨터의 시대가 도래 할 것을 알렸다[1].

위에서 언급한 논문에 따르면 난수성 증명이라는 특정한 문제를 대상으로 양자 프로세서인 시카모어(Sycamore)의 성능 실험을 진행했었는데, 그 결과 현재 사용하고 있는 슈퍼컴퓨터에서 10,000년이 걸릴 연산과정을 양자 컴퓨터를 이용해 약 200초에 해결하는 것을 보였다. 2021년 11월 IBM에서는 127Qubits (Quantum bits. 양자컴퓨터의 기본 계산 단위)의 양자 컴퓨터 개발에 성공하였으며 2022년 11월 현재 433Qubits의 양자 컴퓨터 개발이 완료되었다. 현재 우리나라 역시 50Qubits 의 양자컴퓨터를 2026년까지 개발을 목표로 연구가 활발하게 진행되고 있다.

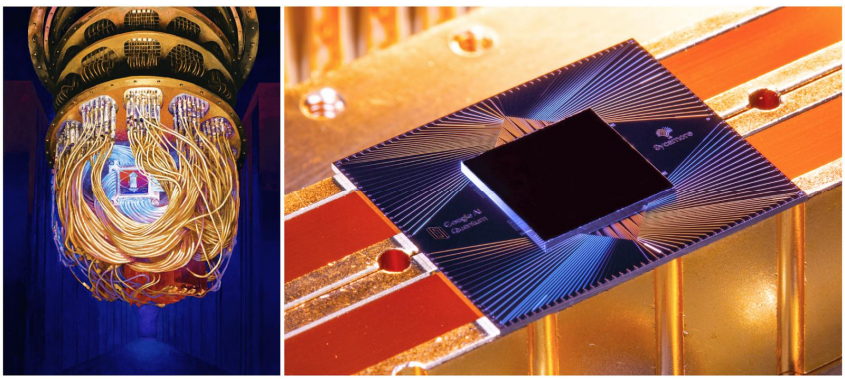

그림 1. Google의 양자 프로세서 시카모어(Sycamore)[1]

그러나 양자컴퓨터의 발전이 가속화 될수록 현재 전 세계적으로 광범위하게 사용 중인 보안 알고리즘들이 위협받고 있는 상황이다. 양자컴퓨터의 발전이 일정 수준 이상으로 도달 할 경우 현재 사용 중인 보안 알고리즘에 엄청난 위협이 될 예정이다. 미국의 정보 기술 연구 및 자문 기업인 Gartner는 양자 컴퓨터 환경의 발전 속도를 고려하였을 때 약 2030년이면 현재 사용하고 있는 RSA, ECC와 같은 공개키 시스템으로 이용 중인 보안 알고리즘들을 사용할 수 없을 것으로 예측되어지고 있다. 예를 들어 '소수끼리 곱하는 것은 쉽지만 이것을 역으로 소인수 분해를 하는 것은 오랜 시간이 소요된다.' 라는 것을 기반으로 만들어진 알고리즘인 RSA (Rivest–Shamir–Adleman) 2048 bits 알고리즘을 해독하려면 슈퍼컴퓨터로는 엄청난 예산과 수천 년이 걸릴 수 있지만 이를 양자컴퓨터를 이용하여 해독하면 슈퍼컴퓨터에 비해 아주 약간의 예산과 수 시간이면 가능하다는 연구결과가 나왔다. 이외에도 양자컴퓨터 전문가들은 15년에서 30년 사이에 양자 컴퓨터가 상용화 될 것이라고 예측하고 있으며, 암호 시스템의 변경에는 코스트와 시간과 같은 상당한 자원이 들어가기 때문에 양자 컴퓨팅 환경과 기존 컴퓨팅 환경에서의 안전성을 보장할 수 있는 우수한 성능의 표준화 암호의 필요성이 제기되었다.

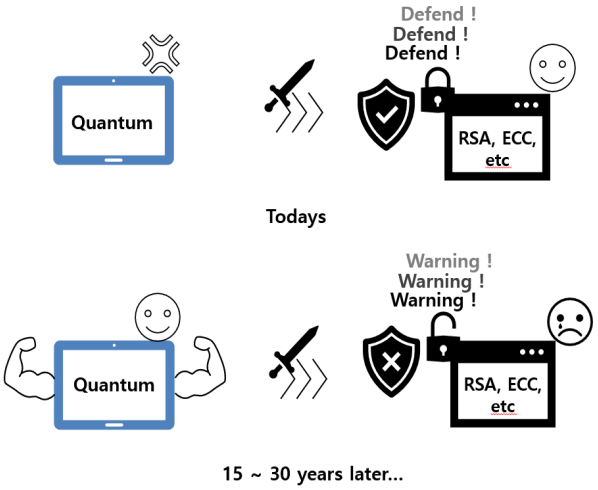

그림 2. 양자컴퓨터의 발전에 따른 위협

이에 대비하여 미국 NIST (National Institute of Standard and Technology, 미국 국립 표준기술연구소)에서 2016년 12월부터 NIST에서는 현재 사용되고 있는 컴퓨터 및 발전하고 있는 양자컴퓨터의 연산에 안전한 PQC 표준 공모를 발표했고 전자서명과 KEM (Key Encapsulation Mechanism)/PKE (Public Key Encryption) 두 분야에 대하여 큰 분류로 부호 기반(Code-based), 격자 기반(Lattice-based), 아이소제니 기반(Isogeny-based), 해시 기반(Hash-based), 다변수 기반(Multivariable-based)등의 총 82개의 알고리즘이 제안되었었으며 현재 표준화 진행의 막바지 단계에 들어섰다.

NIST가 후보 알고리즘들에게 제시한 기준으로 기존의 컴퓨팅 환경, 양자 컴퓨팅 환경으로부터 AES 수준의 안정성을 가지는 것과 더불어 제한된 자원을 가진 경량 임베디드 환경에서의 적합성, 부채널 분석에 저항을 가질 것을 권고하였다.

2022년 7월경 현재 표준으로 선택된 알고리즘으로 PKE/KEM (Public-Key Encrypti-on/Key Encapsulation Mechanism) 분야에서는 CRYSTAL-KYBER, 전자서명 분야에서는 FALCON, SPHINCS+, CRYSTAL-DILITHIUM이 채택되어 있으며 추후 추가 라운드를 통해 표준 알고리즘들을 보강할 예정이다.

본 논문에서는 NIST (National Institute of Standard and Technology, 미국 국립 표준기술연구소)에서 Round 3 후보에 채택되었던 Rainbow의 효율적 유산체 연산 구현을 통한 최적화 및 격자 기반 암호 알고리즘에서 널리 사용되고 있는 고속 연산을 위한 알고리즘인 NTT (Number Theoretic Transformation)의 연산 횟수를 줄이는 효율적 연산 방법 제안한다.

B. 논문의 목적과 연구 방법

양자 내성 암호 알고리즘은 양자 컴퓨터의 연산방식에 내성을 갖고 현재 사용되고 있는 보안 알고리즘을 대체하기 위해 고안되었다. 그러나 임베디드 환경에서의 여러 가지 코스트를 생각하였을 때 보안 요건을 충족하기 위해 사용되는 파라미터들의 값이 크며, 그에 따른 연산량의 증가로 효율적인 구현을 통한 속도 증가 및 코스트를 낮추는 것 역시 중요한 과제로 활발하게 연구가 진행되고 있다.

논문에서 대상으로 하는 알고리즘은 Rainbow와 많은 격자기반 알고리즘 및 NTT이다. Rainbow는 NIST PQC Round3 Finalist 후보에 채택되었던 알고리즘으로서

NIST PQC에서 유일한 다변수이차방정식기반(multivariate quadratic equation-based)의 전자서명 알고리즘이다. Rainbow는 공개키(Public Key) 및 비밀키(Secret Key) 쌍을 생성하는 Key Generation, 서명 생성(Signature), 서명 검증(Verification)으로 나누어진다.

그리고 NTT 알고리즘은 양자내성암호에서 격자 기반의 알고리즘에서 효율적인 연산을 위해 필수적으로 사용되며 현재 NIST에서 표준으로 선발된 알고리즘 및 진행 중인 Round의 다수의 알고리즘이 사용하고 있다.

본 논문에선 Rainbow의 효율적 유한체 연산 구현을 위한 최적화 방식과 양자내성암호에서 격자 기반의 알고리즘에서 효율적인 연산을 위해 필수적으로 사용되며 현재 NIST에서 표준으로 선발된 알고리즘 및 진행 중인 Round의 다수의 알고리즘이 사용하는 NTT (Number Theoretic Transfrom)에서 Montgomery 곱셈연산을 줄여 최적화하는 방법 대해 서술하려고 한다.

이를 위해서 2장에서는 전반적인 배경 지식, NIST PQC Rainbow와 연산체 구조 및 NTT 등에 대하여 소개하며, 제 3장에서는 Rainbow의 효율적인 유한체 연산 구현 및 격자 암호에서 주로 사용되며 현재 표준으로 채택된 알고리즘들이 많이 사용하는 NTT에 대한 효율적 연산 구현에 관하여 서술하고 제 4장에서 앞서 제시한 제안에 대한 시뮬레이션 결과를 보이며 마지막으로 5장에서 결론을 제시한다.

II. 배경지식

오늘 날 4차 산업혁명으로 인해 우리는 정보의 홍수가 일어났다고 해도 무방할 정도로 정보의 바다에서 살고 있다. 이러한 이유로 여러 기술혁신이 일어났으며 자율 주행, 사물 인터넷(IoT), 인공지능(AI)와 같은 기술로 이루어진 정보화 사회에서 살고 있다. 이에 따라 정보의 중요성이 상승했으며 정보에 대한 변조, 훼손, 유출등과 같은 위협에 정보보안이 중요해지는 시대이다. 이런 기술의 발전에 따라 등장하게 된 기술 중 하나로 양자컴퓨터가 있다.

양자 컴퓨터는 1980년대 Richard Feynman, Yuri Manin이 제안한 개념으로서 기존의 컴퓨터와 다르게 비트(bits)가 아닌 큐비트(Qubits)를 이용하며 입자의 양자적 특성을 이용해 자료를 구조화할 수 있다는 점과 양자학적 메커니즘을 이용하여 자료에 대한 연산을 수행할 수 있다는 것에 기인한다. 이러한 양자 컴퓨터는 과학기술의 한계를 극복의 길로도 평가되는 기술이다[2].

A. NIST(미국 국립 표준기술연구소) 양자 내성 암호 알고리즘 표준화

양자 컴퓨터가 날로 발전하고 있지만 발전함에 따라 현재 사용 중인 암호 알고리즘들에 위협이 되어가고 있다. 아직까진 위협적이진 않지만 Qubits의 연산이 늘어날수록 현재 전 세계적으로 사용되고 있는 RSA나 ECC 등과 같은 이산 로그 문제나 인수분해를 이용하여 설계한 공개키 암호 알고리즘들이 Shor의 알고리즘에 의해 적은 시간을 들여 풀리게 되는데 예를 들어 RSA-2048의 경우 다항시간 내에 풀릴 거라는 전망이다[3].

때문에 NIST에서는 2016년 12월 양자내성 암호 알고리즘 공모전을 개최하였으며 표준화를 목표로 삼고 있다. 2017년 11월 총 82개의 알고리즘이 제안되었으며 Round 1, Round 2 를 거쳐 현재 Round 3 를 끝내고 2022 년 격자기반의 CRYSTAL-KYBER, CRYSTAL-DILITHIUM, falcon, 해시기반의 SPHINCS+ 4개의 알고리즘이 채택되었으며 Round 4 평가에 부호기반의 BIKE, Classic-McEliece, HQC, Supersingular Isogeny

기반의 SIKE 및 새로운 알고리즘의 공모를 하고 있다[4].

표 1. NIST PQC 표준 공모전 현황

	Base	Schemes	
		PKE/KEM	Signature
Selected Algorithm 2022	Lattice-based (Module)	CRYSTAL-KYBER	CRYSTAL-DILITHIUM
	Lattice-based (Ring)		FALCON
	Hash-based		SPHINCS+
ROUND4 SUBMISSION	Code-based	BIKE, Classic-McEliece, HQC	
	SuperSingular Isogeny	SIKE	

이 중에서 현재 SIKE와 같은 경우 다양하고 많은 공격에 많은 취약점이 드러나 Round 4 후보 중에서 채택되기 힘든 것으로 전망되고 있는 추세이다. NIST에서는 현재 2024년 표준화를 목표로 표준화를 진행 중이다.

B. 수학적 이론

1. 군(Group)

Definition 1. 집합 $\mathbb{G}(=\emptyset)$ 에서 이항 연산인 *이 정의되어 있으며, 따라서

$$a, b \in \mathbb{G} \Rightarrow a * b \in \mathbb{G} \tag{1.1}$$

아래의 조건이 성립할 때 $(\mathbb{G}, *)$ 를 군$(group)$이라 정의한다.

- * 연산은 결합법칙이 성립한다.
  $(a,b,c \in \mathbb{G}$ 에 대하여, $a * (b * c) = (a * b) * c)$

- 특정 원소 $e \in \mathbb{G}$ 가 존재, 모든 원소 $a \in \mathbb{G}$ 에 대하여 $a*e = e*a = a$ 가 성립한다.

- 각 원소 $a \in \mathbb{G}$ 에 대해 $a * x = x * a = e$, 인 원소 $x \in \mathbb{G}$ 가 존재하며, 원소 a^{-1}를 a의(연산 $*$에 해당) 역원(Inverse)라 칭하며 이 원소는 a^{-1}로 나타낸다.

$$a * a^{-1} = a^{-1} * a = e \tag{1.2}$$

그리하여 $(\mathbb{G}, *)$가 군일 경우, "$\mathbb{G}$ 는 연산 $*$에 관하여 군을 이룬다."라고 말할 수 있다. 군 $(\mathbb{G}, *)$에서 항등원은 e 로 유일하며 원소의 역원 역시 하나만 존재한다.

Definition 2. 군 $(\mathbb{G}, *)$에서, $\mathbb{G}$ 가 무한적 집합일 때 이 군을 무한군(infinite group)라 칭하고 $\mathbb{G}$ 가 유한적 집합일 때 이 군을 유한군(finite group)이라고 칭한다. 특히 $|\mathbb{G}| = n$ 일 경우에 n을 이 군의 위수(Order)라 칭한다.

Definition 3. 군 $(\mathbb{G}, *)$ 에 관하여 다음의 정의가 성립될 때, 이러한 군을 아벨군(Abelian group) 또는 가환군(commutative group)이라고 한다.

교환법칙 : 모든 $a, b \in \mathbb{G}$ 에 대하여 $a * b = b * a$ 이다.

만약 그렇지 않을 경우 군 $(\mathbb{G}, *)$ 을 비 아벨군(nonabelian group) 또는 비가환군(noncommutative group)이라 칭한다. 그리고 연산 기호가 $\bullet$ 인 군을 곱셈연산군(multiplicative group)이라 하고 그 항등원은 e 또는 1이 된다. 또, 연산기호가 $+$인 아벨군을 특히 덧셈 군(additive group)이라 칭하고 그 항등원을 0, 각 원소 a의 덧셈에 관한 역원을 $-a$ 로 칭한다[5].

2. 환과 체(Ring and Field)

Definition 4. 집합 $\mathbb{R}(\neq \emptyset)$ 위에 덧셈연산과 곱셈연산이 정의되어 있는 상태에서

$$a, b \in \ a+b \in \mathbb{R},\ a \bullet b \in \mathbb{R} \tag{1.3}$$

이며 아래의 조건이 성립할 때, $(\mathbb{R}, +, \bullet)$ 을 환(ring)이라 칭한다.

- $(\mathbb{R}, +)$는 아벨군이다.

- $\bullet$에 대하여 결합법칙이 성립한다. 이는 모든 $a,b,c \in \mathbb{R}$ 에 대해, $(a \bullet b) \bullet c = a \bullet (b \bullet c)$.

- 분배법칙이 성립한다. 이는 모든 $a,b,c \in R$ 에 대해, $a \bullet (b+c) = a \bullet b + a \bullet c = (b+c) \bullet a = b \bullet a + c \bullet a$

Definition 5. 환 $\mathbb{R}$이 아래의 조건을 만족 시킬 때, 이러한 환을 가환환(commutative ring)이라 칭한다.

$$a \bullet b = b \bullet a \tag{1.4}$$

Definition 6. 환 $\mathbb{R}$이 다음과 같은 조건을 만족시킬 때, 이러한 환을 단위원 1을 가진 환(ring with identity)라고 칭한다.

- 특정한 원소 $1 \in \mathbb{R}(1 \neq 0)$이 존재하여, 모든 원소 $a \in \mathbb{R}$ 에 대하여 등식 $a \cdot 1 = 1 \cdot a = a$ 가 성립한다.

Definition 7. 단위원 1을 가진 환 $\mathbb{R}$이 아래의 조건을 만족시킬 때, 이 환을 나눗셈 환(division ring)이라 칭한다.

- 각 $a, b \in \mathbb{R}, b \neq 0$에 대하여 $aa^{-1} = a^{-1}a = 1$ 인 원소 $a^{-1} \in \mathbb{R}$ 이 존재한다.

- 환 $\mathbb{R}$ 에서 덧셈을 이용하여 뺄셈을 다음과 같이 정의한다.

$$a, b \in \mathbb{R} \text{ 일 때, } a - b = a + (-b) \tag{1.5}$$

- 가환환인 나눗셈 환을 체(field)라고 칭한다.

- $\mathbb{R}$ 이 체일 때, 곱셈연산을 사용하여 나눗셈을 다음과 같이 정의한다.

$$a, b \in \mathbb{R} \text{ 일 때, } a \div b = ab^{-1} \tag{1.6}$$

Definition 8. 환 $\mathbb{R}$이 단위원 1을 가진 가환환으로서 아래의 조건을 만족할 때, $\mathbb{R}$을 정역(integral domain)이라고 칭한다.

$$ab = 0 \Rightarrow a = 0 \text{ or } b = 0 \tag{1.7}$$

Definition 9. 환 $(\mathbb{R}, +, \cdot)$에서 $\mathbb{R}$이 무한집합인 경우, 무한환(infinite ring)이라 칭하고, $\mathbb{R}$ 이 유한집합인 경우 이 환을 유한환(finite ring) 칭한다. 또한 유한개의 원소로 이루어진 체를 유한체(finite field)라 칭하며, 특히 $|\mathbb{R}| = n$일 때, n 을 $\mathbb{R}$의 위수(order)라 칭한다.

Definition 10. 환 $\mathbb{R}$이 단위원 1을 가진 환이라고 하고, 적당한 양의 정수 k 에 대하여 $k \cdot 1 = 0$ 인 경우에, 가장 작은 양의 정수 m 을 $\mathbb{R}$의 표수

(characteristic) 라고 칭한다.

$$m \bullet 1 = 1 + 1 + \cdots + 1 = 0 \tag{2.8}$$

Definition 11. 환 $(\mathbb{R}, +, \bullet)$에서 $\mathbb{R}$의 부분집합 $\mathbb{S}(\neq \emptyset)$가 $\mathbb{R}$의 두 연산과 동일한 연산에 대하여 환을 이룰 때, $\mathbb{S}$ 를 $\mathbb{R}$ 의 부분환이라고 칭한다. 특히 체 $(\mathbb{F}, +, \bullet)$에서 $\mathbb{F}$ 의 부분집합 $S(\neq \emptyset)$가 $\mathbb{F}$ 에서 두 연산과 동일한 연산에 관하여 체를 이룰 때 $\mathbb{S}$ 를 $\mathbb{F}$ 의 부분체라고 칭한다.

Definition 12. 체 $\mathbb{F}$ 의 모든 부분체들의 교집합은 $\mathbb{F}$ 의 부분체이다. 이 부분체를 $\mathbb{F}$ 의 소체(Prime field)라고 한다.

Definition 13. 체 $\mathbb{F}$ 가 체 $\mathbb{G}$ 의 부분체일 때, $\mathbb{G}$ 를 $\mathbb{F}$ 의 확대체(Extension field)라 하며, $\mathbb{F} < \mathbb{E}$라고 표현한다.

Definition 14. 체 $\mathbb{F}$ 의 확대체 $\mathbb{G}$ 에 대하여 $\mathbb{G}$ 에 속하는 모든 원소들이 $\mathbb{F}$ 위에서 대수적일 경우 $\mathbb{E}$ 를 $\mathbb{F}$ 의 대수적확대체(Algebraic Extension)라 칭한다.

Definition 15. 체 $\mathbb{F}$ 의 확대체 $\mathbb{G}$ 가 $\mathbb{F}$ 위에서 벡터공간으로 유한차원 n일 경우, $\mathbb{E}$ 는 $\mathbb{F}$ 위에서 차수 n인 유한확대체(finite extension)라 칭하고 $[\mathbb{E}:\mathbb{F}]$로 표현한다.

Definition 16. 체 $\mathbb{F}$ 에 대해 $\mathbb{F}[x]$에 속하는 모든 상수가 아닌 다항식들이 $\mathbb{F}$ 에서 근을 가질 경우 $\mathbb{F}$ 를 대수적으로 닫혀있다(algebraically closed)라고 한다.

3. 다항식(Polynomial)과 다항식 환(Polynomial ring)

Definition 17. 환 $\mathbb{R}$을 단위원 1을 가진 가환환이라 가정하고 x 를 미지수라고 가정했을 때, 아래와 같은 형태의 형식적인 무한 합을 환 $\mathbb{R}$ 위의 다항식(polynomial)이라고 한다.

$$f(x) = a_0 + a_1 x + \cdots + a_n x^n + \cdots, a_i \in R \tag{1.9}$$

$a_0, a_1, \cdots, a_n, \cdots$을 다항식 $f(x)$ 의 계수(coefficient)라고 칭하며, 위의 다항식 $f(x)$에서, 모든 $i \geq n$ 에 대하여 $a_i = 0$ 일 경우, 이 다항식을 아래와 같이 나타낸다.

$$f(x) = a_0 + a_1 x + \cdots + a_n x^n \tag{1.10}$$

$a_0 + a_1 x + \cdots + a_n x^n$는 다항식 $f(x)$의 항(term)이라 칭하고 a_0은 상수항(constant term)이라 칭한다. 만약 $a_1 = \cdots = a_n = 0$일 경우 $f(x) = a_0$를 상수 다항식(Constant Polynomial)이라 칭하며 $f(x) = 0$일 경우 영다항식(Zero Polynomial)이라고 칭한다.

Definition 18. 상수가 아닌 다항식 $f(x) \in \mathbb{F}[x]$가 $\mathbb{F}$ 위에서 기약(irreducible) 또는 $\mathbb{F}[x]$에서 기약다항식(irreducible polynomial in $\mathbb{F}[x]$) 이라 한다는 것은 $f(x)$가 $f(x)$의 차수보다 낮은 차수의 $\mathbb{F}(x)$에 속하는 두 다항식 원소 $g(x)$와 $h(x)$의 곱 $g(x)h(x)$로 나타나지 않을 때를 의미한다.

Definition 19. 상수가 아닌 다항식 $f(x) \in \mathbb{F}[x]$가 $\mathbb{F}$ 위에서 기약(irreducible) 또는 $\mathbb{F}[x]$에서 기약다항식(irreducible polynomial in $\mathbb{F}[x]$) 이라 한다는 것은 $f(x)$가 $f(x)$의 차수보다 낮은 차수의 $\mathbb{F}[x]$에 속하는 두 다항식 원소 $g(x)$와 $h(x)$의 곱 $g(x)h(x)$로 나타나지 않을 때를 의미한다.

Definition 20. $p(x)$**가 체** $\mathbb{F}$의 원소를 계수로 갖는 n차 다항식일 경우, 방정식 $p(x) = 0$는 $\mathbb{F}$에서 서로 다른 n개를 해를 갖는다.

C. 다변수이차방정식 기반의 양자 내성 서명 알고리즘 Rainbow

Rainbow는 Ding과 Schmidt에 의해 2004년에 처음 제안된 알고리즘으로, 다중 세

그먼트 UOV (Unbalanced Oil and Vinegar) 구조로 이루어진 키 교환 알고리즘이다. 다변수이차방정식 기반의 서명 알고리즘은 다변수이차식 및 다항식 확장의 확장 동형(EIP : Extended Isomorphism of Polynomials)문제의 어려움에 기반한다. 이 중 대표적으로 NIST PQC 표준 공모의 Round 3 Finalist 후보로 채택되었던 Rainbow 서명 알고리즘이다. Rainbow의 파라미터는 공개키 크기가 보안 수준에 따라 258kB 에서 1,885kB의 크기를 가지며 이때 세부적으로 $\mathbb{F}_{16}$ 상에서의 연산이 주를 이루며 이때 $\mathbb{F}_{16}$과 $\mathbb{F}_{256}$의 경우 혼동이 없을 경우 $\mathbb{F}$ 로 축약해 표현한다[6].

다변수이차다항식 공개키 암호는 $K = \mathbb{F}_0$ 에서 주요 연산이 이뤄지고 이때 이것을 기저체(Base Field)라고 한다. Rainbow의 Level Ⅰ에서 이용하는 기저체는 $\mathbb{F}_{16}$이다. $n > m$일 때 공개키는 이와 같이 $\mathbb{F}^{n\times m} \rightarrow \mathbb{F}^{n\times m}$로 아래의 식 (1.11)로 사상이 정의된다.

$$P = \mathcal{S} \circ \mathcal{F} \circ \mathcal{T} \tag{1.11}$$

그리고 $Q : x \rightarrow y$는 중심사상(central map)이라 칭하며, 이차 다항식으로 구성되어 있고 역행렬 계산이 가능하도록 설계되어 있다. 대부분의 다변수이차다항식 기반의 암호는 Q 를 어떻게 만들어내는지에 따라 특성이 구별된다. 보안성을 높이기 위해서는 공개키 P 에서부터 개인키에 해당하는 비밀요소 행렬을 구분해내는 것이 어려워야한다[7][8].

Rainbow에선 네 가지의 정수 (q, v_1, o_1, o_2)가 중요한 파라미터로 사용되는데 중심사상 Q에선 “Oil”에 해당하는 변수들과 “Vinegar”에 해당하는 변수로 나누어진다.

첫 번째 세그먼트에서 Vinegar 변수들은 $i \in V_1 = \{1,...,v_1\}$에 대응하는 x_i이고, oil 변수들은 $i \in O_1 = \{v_1+1,...,v_2 := v_1+o_1\}$에 대응하는 x_i이다.

두 번째 세그먼트에서 Vinegar 변수들은 인덱스 집합 $V_2 = \{1,...,v_2 = v_1+o_1\}$이고 Oil 변수들은 다음 식 (1.12)와 같은 인덱스 집합으로 정의된다.

$$O_2 = \{v_2+1,...,n = v_3 = v_2+o_2 = v_1+o_1+o_2\} \tag{1.12}$$

이때 중심사상 Q는 $m = o_1+o_2$ 개의 이차 방정식 $y = (y_{v_1+1},...,y_n) = (q_{v_1+1}(x), ... , q_n(x))$를 갖고 있다. 여기서 각 이차방정식은 다음 식 (1.13)과 같이 정의된다.

$$y_k = q_k(x) = \sum_{i=1}^{v_1}\sum_{j=1}^{v_2}\alpha_{i,j}^{(k)}x_i x_j,\ k = Q_1 \tag{1.13}$$

$$y_k = q_k(x) = \sum_{i=1}^{v_2}\sum_{j=1}^{n}\alpha_{i,j}^{(k)}x_i x_j,\ k = Q_2$$

k가 Q_1에 있을 때 모든 이차 방정식 q_k에는 i와 j가 모두 Q_1에 있는 교차 항 $x_i x_j$ 가 없다. 그래서 $v_1 < i \le v_2$인 첫 번째 단계에서 모든 y_i에 대해 그리고 모든 Vinegar 변수 $j \le v_1$인 x_j에 대해, 선형 방정식을 풀어서 대응되는 Oil 변수들을 쉽게 계산할 수 있다. 마찬가지로 모든 k가 O_2에 있는 모든 이차 방정식 q_k에 대해 i와 j가 모두 O_2에 있는 교차 항 $x_i x_j$가 없다. 그리하여 $v_2 < i \le n$인 두 번째 단계에서 모든 y_i에서 $j \le v_2$인 모든 Vinegar 변수 x_j가 주어져 있을 때 선형방정식을 풀어 $x_{v_{2+1}},...,x_{v_n}$을 쉽게 계산할 수 있다. Q 에 대한 x의 역상 y는 다음과 같이 구할 수 있다[9].

1) 랜덤 추측을 통해 초기 Vinegar 변수 $\tilde{x}=(x_1,..,x_{v_1})$를 만든 후 이 값과 $(y_{v_1+1},...,y_{v_2})$로부터 Gauss 소거법을 통해 $(x_{v_{1+1}},...,x_{v_2})$를 구한다. 만일 해가 없다면 처음부터 다시 한다.

2) $\tilde{x}=(x_1,...,x_{v_2})$와 $(y_{v_2+1},...,y_n)$을 이용해서 재차 Gauss 소거법을 통해서 $(x_{v_2+1},...,x_n)$을 구한다. 만일 해가 없다면 처음부터 다시 한다.

n을 변수 $x=(x_1,\cdots,x_n)$ 의 개수, m을 방정식의 개수라 칭할 때, $\mathbb{F}_q$ 의 다변수 이차식 함수 $MQ(n,m,\mathbb{F}_q)$는 아래 식 (1.14), (1.15)와 같이 나타낼 수 있다.

$$MQ(n,m,\mathbb{F}) = \mathbb{F}(x) = (f_1(x),\ \cdots,f_m(x)) \tag{1.14}$$

$$f_s(x) = \sum_{i,j}a_{i,j}^{(s)}x_i x_j + \sum_i \beta_i^{(s)}x_i \tag{1.15}$$

주어진 $v\in\mathbb{F}_q^m$에 대해, $\mathbb{F}(x)\equiv v$ 를 n 개의 변수에 대한 m 개의 이차 방정식이

라 하며, 이를 만족하는 변수 벡터 x 를 찾아내는 것을 다변수이차식 문제라고 칭한다. 다변수이차식 문제는 NP-Complete로 잘 알려져 있으며, 양자 컴퓨터 연산에서도 풀어내기 힘들다고 알려져 있다[6][9].

다항식의 확장 동형 문제에 대해 설명하자면 공개키 pk 가 $P = \mathcal{S} \circ \mathcal{F} \circ \mathcal{T}$ 를 만족하는 P 의 계수의 집합이면서 공개키가 pk주어졌을 경우, $P = \mathcal{S} \circ \mathcal{F} \circ \mathcal{T}$ 로부터 중앙 맵 $\mathcal{F}$ 와 가역행렬 $\mathcal{S}, \mathcal{T}$ 를 각각 찾아내는 문제를 뜻한다. 다항식의 확장 동형 문제는 NP-Complete로 증명되진 않았으나, 현재까지 풀기 어려운 문제이며, 이 문제의 어려움의 정도는 다변수이차다항식 $\mathbb{F}$ 의 구조에 의존한다[10][11][12].

Rainbow에서는 NIST PQC 보안 수준 5개를 3개로 묶어서 만들었으며, 그 파라미터는 아래의 표 2와 다. 아래의 파라미터 중 n과 m은 변수 및 방정식을 뜻한다.

표 2. 레인보우 파라미터

Security	NIST Round	Round 2	Round 3
128 bits	Field	$\mathbb{F}_{16}$	$\mathbb{F}_{16}$
	(v_1, o_1, o_2)	32, 32, 32	36, 32, 32
	$n \rightarrow m$	96 → 64	100 → 64
192 bits	Field	$\mathbb{F}_{256}$	$\mathbb{F}_{256}$
	(v_1, o_1, o_2)	68, 36, 36	68, 32, 48
	$n \rightarrow m$	140 → 72	148 → 80
256 bits	Field	$\mathbb{F}_{256}$	$\mathbb{F}_{256}$
	(v_1, o_1, o_2)	92, 48, 48	96, 36, 64
	$n \rightarrow m$	188 → 96	196 → 100

I $(q, v_1, o_1, o_2) = GF(16), (v_1, o_1, o_2) = (36,32,32)$ (NIST PQC 보안 수준 1, 2)

III $(q, v_1, o_1, o_2) = GF(256), (v_1, o_1, o_2) = (68,32,48)$ (NIST PQC 보안 수준 3, 4)

V $(q, v_1, o_1, o_2) = GF(256), (v_1, o_1, o_2) = (96,36,64)$ (NIST PQC 보안 수준 5)

Rainbow에서 사용하는 총 변수의 개수가 n일 경우, $n = v_1 + o_1 + o_2$ 인데, v_1 은 첫 번째 레이어의 Vinegar 변수의 개수이고, o_1, o_2는 각각 첫 번째, 두 번째 레이어

의 오일 변수의 개수를 나타낸다[13][14].

1. Rainbow Key Generation Algorithm

Rainbow의 개인키는 역행렬이 존재하는 가역 맵 $\mathcal{S} : \mathbb{F}^{n\times m} \rightarrow \mathbb{F}^{n\times m}$, $\mathcal{T} : \mathbb{F}^{n\times m} \rightarrow \mathbb{F}^{n\times m}$ 과 중앙 맵 $\mathcal{F} : \mathbb{F}^{n\times m} \rightarrow \mathbb{F}^{n\times m}$ 으로 구성된다. 중앙 맵 $\mathcal{F}$ 에서는 m 개$(m=n-v_1)$의 다변수 다항식 $f^{(v_1+1)}, \cdots, f^{(o_1)}$ 로 구성된다. $k \in v_1,\cdots,n$ 일 경우, $f^{(k)}$ 는 아래 식 (1.16)과 같다[15][16].

$$f^{(k)}(x_1, \cdots, x_n) = \sum_{i,j\in V_l}^{i\le j} a_{ij}^{(k)} x_i x_j + \sum_{i\in V_l, j=O_l} \beta_{ij}^{(k)} x_i x_j + \sum_{i\in V_l \cup O_l} \gamma_i^{(k)} x_i + \delta^{(k)} \tag{1.16}$$

공개키의 경우 아래 식 (1.17)과 같이 나타낼 수 있다. 즉, $\mathcal{S}$, $\mathcal{F}$, $\mathcal{T}$를 행렬 곱하면 된다.

$$P = \mathcal{S} \circ \mathcal{F} \circ \mathcal{T} : \mathbb{F}^{n\times m} \rightarrow \mathbb{F}^{n\times m} \tag{1.17}$$

아래의 알고리즘 1은 Rainbow의 키 생성 과정을 나타낸다.

알고리즘 1. Rainbow의 Key Generation

입력 : Rainbow parameter (q, v_1, o_1, o_2)

출력 : Rainbow key pair (sk, pk)

처리과정

1: $m \leftarrow o_1, o_2$

2: $n \leftarrow m + v_1$

3: **repeat**

4: $M_S \leftarrow Matrix(q,m,m)$

5: **until** IsInvertible(M_S) == TRUE
6: $c_s \leftarrow {}_R F^m$
7: $\mathcal{S} \leftarrow Aff(M_S, c_S)$
8: $Inv\mathcal{S} \leftarrow M_S^{-1}$
9: **repeat**
10: $M_T \leftarrow Matrix(q,n,n)$
11: **until** IsInvertible(M_T) == TRUE
12: $c_T \leftarrow {}_R \mathbb{F}^n$
13: $\mathcal{T} \leftarrow$ **Aff**(M_T, c_T)
14: $InvT \leftarrow M_T^{-1}$
15: $F \leftarrow rainbowmap(q, v_1, o_1, o_2)$
16: $P = \mathcal{S} \circ \mathcal{F} \circ \mathcal{T}$
17: $sk \leftarrow (InvS,\ c_S,\ \mathcal{F}\ InvT, c_T)$
18: $pk \leftarrow P$
19: **return** (sk, pk)

2. Rainbow Signature Generation Algorithm

서명할 문서를 d, 해시 함수는 $H: 0,1 \rightarrow \mathbb{F}^{n \times m}$, 서명을 $z \in \mathbb{F}^{n \times m}$ 일 때, 서명 생성 방법은 아래와 같다.

- x = $\mathcal{S}^{-1}(h) \in \mathbb{F}^{n \times m}$을 계산한다.

- 중심 맵 $\mathcal{F}$ 아래에서 x 의 사전 이미지 $y \in \mathbb{F}^{n \times m}$을 계산한다. 아래의 알고리즘 2는 이러한 계산과정을 담은 의사코드이다.

알고리즘 2. Rainbow Central Map's Inverse

입력 : Rainbow Central Map $\mathbb{F} = (f^{(v_1+1)}, \cdots, f^{(n)})$**, Vector x** $\ni \mathbb{F}^{n \times m}$

출력 : vector $y \in \mathbb{F}^n$ **with** $\mathcal{F}(y) = \mathbf{x}$

처리과정

1: Choose random values for the variables $y_1, \cdots, y_{v_1}$ and substitute these values into the polynomials $f^{(i)} (i = v_1 + 1, \cdots, n)$.

2: **for** $l = 1$ to u **do**

3: Perform Gaussian Elimination on the polynomial $f^{(i)} (i \in O_l)$

4: to get the values of the variables $y_i (i \in O_l)$.

5: Substitute the values of $y_i (i \in O_l)$ into the polynomials $f^{(i)}$

6: $(i = v_{l+1} + 1, \cdots, n)$.

7: **end for**

8: **return** $y = (y_1, \cdots, y_n)$

- 서명 $z \in \mathbb{F}^{n \times m}$ / $\boldsymbol{z} = \boldsymbol{\mathcal{T}}^{-1}(\boldsymbol{y})$를 계산한다.

아래의 알고리즘 3의 의사코드는 Rainbow의 서명 과정을 나타낸다[17].

알고리즘 3. Rainbow의 Signature Generation Process

입력 : Rainbow private key $(InvS,\ c_S,\ \mathcal{F}\ InvT, c_T)$ **document** d

출력 : 서명 $z \in \mathbb{F}^n$ **such that** $P(\boldsymbol{z}) = H(d)$

처리과정

1: $\boldsymbol{h} \leftarrow H(d)$

2: $x \leftarrow InvS \bullet (\boldsymbol{h} - c_S)$

3: $y \leftarrow InvF(\mathcal{F},\ \mathrm{x})$

4: $z \leftarrow InvT \bullet (\boldsymbol{y} - c_T)$

5: **return** z

3. Rainbow Signature Verification Algorithm

메시지 d 와 서명 $\boldsymbol{z}$ 가 주어졌을 시, 서명에 대한 검증은 아래와 같이 이루어진다[18].

- 해시 함수 H 를 이용하여 해시 값 $h = H(d) \in \mathbb{F}^{n \times m}$
- 공개키 P 를 이용하여 $h' = P(z) \in \mathbb{F}^{n \times m}$을 계산한다.
- 서명 $\boldsymbol{h} = h'$ 의 조건이 만족하는 경우 서명 $\boldsymbol{z}$ 가 검증된다.

알고리즘 4. Rainbow의 Signature Verification Algorithm

입력 : document d, signature $\boldsymbol{z} \in \mathbb{F}^{n \times m}$

출력 : TRUE or FALSE

처리과정

```
1: h ← H(d)
2: h' ← P(z)
3: if h' == h
4:     return TRUE
5: else
6:     return FALSE
7: end if
```

D. 격자 기반 암호 알고리즘

격자 기반 암호 알고리즘은 격자 상에서 SVP (Shortest Vector Problem)과 CVP (Closet Vector Problem)의 수학적 기반 어려움에 기반하고 있으며, 1996년 Ajtai에 의해 암호 분석 도구로만 사용되던 격자를 암호의 기본 요소를 구성하는데 사용될 수 있다는 것이 제안했었다. Ajtai는 [7]에서 Short Integer Solution (SIS)이라 알려진 “특정 Average-case 격자 문제가 worst-case 격자 문제처럼 풀기 어렵다”라고 서술하고 있다[19][20][21].

격자란 R^n 에서 정의되는 이산 하위 그룹으로서 조금 더 자세히는 R^n에서 정의된 k 개의 기저 $B = \{b_1, b_2, \cdots, b_k\}$ 의 정수 선형 조합으로 정의되며 다음 식 (1.18)

와 같이 표시할 수 있다[22][23].

$$L = L(B) = \{B \bullet Z^n = \sum_{i=1}^{n} c_i b_i : c_i \in Z\} \tag{1.18}$$

격자는 하나의 기저 벡터에 의해 정의되었을지라도 동일 격자를 표현할 수 있는 많은 수의 다른 기저 벡터가 존재하는 특징을 가지는데, 벡터를 구성하는 각각의 기저에 해당하는 사이즈가 작고 서로 거의 수직을 이루는 기저 벡터를 '좋은(good)' 기저 벡터라 칭하며 반대로 기저들의 사이즈가 크고, 수직을 이루지 않는 기저 벡터를 '나쁜(bad)' 기저 벡터라 칭한다. 좋은 기저 벡터는 격자 위에서 정의된 어려운 문제를 쉽게 해결할 수 있어 격자 기반 암호 구성에서 개인키로 이용되며 나쁜 기저 벡터는 공개키로 사용된다[24][25][26].

격자 기반 암호에 이용되는 격자 기반의 문제는 표. 3와 같은 종류들이 있으며 암호를 구성할 때 이러한 어려움을 해결하는 것으로 설계된다. 대부분의 격자 기반의 양자내성 암호 알고리즘들은 LWE 문제를 사용하며 본 논문에서 다루는 CRYSTAL-KYBER의 경우 Ring-LWE (Learning with Error) 문제를 기반으로 하는 Module-LWE를 사용한다.

1. Ring-LWE 암호 알고리즘

위에서 언급하였듯 Module-LWE는 Ring-LWE의 문제를 기반으로 한다. RING-LWE는 Lyubashevsky 등에 의해 제안되었으며 기존의 공개키 알고리즘과 같이 키 생성, 암호화, 복호화 총 세 단계로 이루어져 있다. 아래의 표 3은 격자 기반에서 주로 다루는 암호문제의 종류를 보여준다[27][28].

표 3. 격자 기반 암호문제의 종류

문제 종류	문제 내용
CVP (Closest Vector	임의의 기저 벡터 BV로 구성된 격자 구조 $L = L(BV)$ 및 임의의 벡터 값 v가 주어졌을 때, $L(BV)$ 격자 구조 위에서

Problem)	v와 가장 가까운 벡터를 찾는 문제
SVP (Shortest Vector Problem)	임의의 기저 벡터 BV로 구성된 임의의 격자 구조 $L = L(BV)$ 가 주어졌을 경우, $L(BV)$ 격자 구조 위의 0 이 아닌 가장 짧은 벡터를 찾는 문제
SIS (Shortest Integer Solution)	랜덤한 $s \in Z_q^k$로 이루어진 임의의 행렬 $Arr = (m_1, m_2, \cdots, m_k)$에서 특정 조건 $A_v \equiv 0 (\mathrm{mod}\, q)$ 을 만족하는 가장 짧은 non-trivial 벡터를 찾는 문제
LWE (Learning With Error)	LWE는 여러 가지 문제를 가지고 있는데 정규 분포를 따르는 오류 분포 χ를 이용해 비밀 벡터 $s \in Z_q^k$ 에 대하여 (a, b)를 출력으로 갖는 LWE 분포 $A_{s,\chi} \in Z_q^n \times Z_q$ 가 있는데 a는 $a \in Z_q^k$, χ 는 랜덤하게 선택된 값이며, 이를 이용해 $b = \langle a, s \rangle + e \,(\mathrm{mod}\, q) \in Z_q$ 를 계산한다. 크게 LWE 분포 $A_{s,\chi}$에서 수집된 k개의 독립된 표본을 가지고 s를 찾아내는 Search LWE, k개의 독립된 표본이 LWE 분포 $A_{s,\chi}$로부터 추출되었는지 랜덤한 분포 내에서 추출되었는지 구별하는 문제인 Decision LWE가 있다.

일반적으로 Ring-LWE 기반의 암호는 128비트 또는 256비트 보안 레벨을 충족시키기 위해서 파라미터 (n, q, σ) (n : 차수 / q: $Modulus$ / σ: 오류 다항식을 선택하는 가우시안 분포의 표준 편차)를 $(256, 7681, \frac{11.31}{\sqrt{2\pi}})$ 혹은 $(512, 12289, \frac{12.18}{\sqrt{2\pi}})$ 을 사용한다[28][29].

지금 소개할 Ring-LWE는 Lyubashevsky의 암호 알고리즘이다. 암호 알고리즘의 설명에 앞서 Ring-LWE에서 $\mathbb{R}_q = Z_q / \langle x^n + 1 \rangle$ 안에서 정의되고, $Ring(\mathbb{R}_q)$ 안에서 동작하며, $\mathbb{R}_q$ 에 대한 산술연산으로는 모듈러 연산, 다항식 곱셈연산, 다항식 덧셈이 사용되며, $poly(R_i, C_i, a)$ 는 각각 다항식을 의미하고 $polyA$ 와 같은 경우 임의의 다항식으로 공개키 생성에 사용되며 소문자 m은 메시지다[30][31].

첫 단계로 키 생성 단계인 $KeyGen(polyA)$ 는 두 개의 오류 다항식

$polyR_1, polyR_2$ 를 이상 정규분포에서 임의로 추출 한 후 공개키로 사용되는 다항식 $polyP = polyR_1 - polyA \times polyR_2$ 를 계산한다. (여기서 이산 정규 분포는 표준편차 σ와 평균 0을 따른다.) 개인 키는 $polyR_2$, 다항식 쌍 $(polyA, polyP)$ 를 공개키로 이용한다[32][33].

다음으로 암호화 단계인 $Enc((polyA, polyP), m)$으로 n 비트 문자열 메시지 m 의 암호화는 처음으로 메시지 m을 r_q 의 원소 $(=\overline{m})$ 로 인코딩 하는 것인데 메시지의 비트 각각 $\left\lfloor \frac{q}{2} \right\rfloor$ 를 곱하는 방법이다. 여기서 $\left\lfloor \frac{q}{2} \right\rfloor$ 는 $\frac{q}{2}$ 를 넘지 않는 최대 정수임을 뜻하며, r_q 의 원소로 인코딩 되어진 메시지는 밑의 식 (1.19), (1.20)로 계산된다.

$$polyC_1 = polyA_1 \times polyE_1 + polyE_2 \tag{1.19}$$

$$polyC_2 = polyP \times polyE_1 + polyE_3 + \overline{m} \tag{1.20}$$

세 번째로 복호화 단계로서 개인키 $polyR_2$ 를 이용하여 메시지를 아래의 식 (1.21)로 계산하게 된다.

$$m = Decode(polyC_1 \times polyR_2 + polyC_2) \tag{1.21}$$

E. NTT(Number Theoretic Transformation)

현재 NIST 양자내성 암호 표준화에 채택되어 있는 격자기반의 알고리즘 3가지, CRYSTAL-KYBER, CRYSTAL-DILITHIUM, FALCON 그리고 4라운드 후보에 있는 코드기반의 알고리즘 2가지 BIKE, HQC는 다항식 곱셈연산의 효율적인 연산 수행을 위해 필수적으로 고속연산 알고리즘인 NTT를 사용하고 있다. NTT는 유한체에서 동작하는 고속 퓨리에 변환(Fast Fourier Transform)의 일종으로서 다항식 곱셈연산을 합성 곱(Convolution)에서 Point-wise multiplication으로 바꾸어 복잡도를 낮추어 주는 동작을 수행한다. 아래의 표 4는 현재 NTT를 사용하는 NIST 양자내성 암호 표준화에 채택된 알고리즘 및 현재 진행 중인 Round에 속한 알고리즘을 보여준다.

RLWE (Ring Learning with Error), MLWE (Module Learning with Error) 등의 문제를 적용하는 다수의 격자 기반 암호 알고리즘은 다항식 환에 대한 다항식 곱셈연산, 다항식 덧셈, Modulo Reduction으로 구성된다[34].

이때 다항식 환 $\mathbb{R}_q$는 $\mathbb{Z}_q[x]/f(x)$로서 $q \equiv 1 \mod 2n$ 은 충분히 큰 공용 소수로서, 이러한 다항식은 보안 매개변수 n으로 표현되는 x^n+1의 불변성 다항식이다.

표 4. NTT를 사용하고 있는 NIST PQC 표준화 알고리즘 및 후보 알고리즘

Using NTT for PQC			
Selected Algorithm	Lattice-based CRYSTAL-KYBER	Lattice-based CRYSTAL-DILITHIM	Lattice-based FALCON
Round 4 Submission	Code-based BIKE	Code-based HQC	

이러한 다항식 환 $\mathbb{R}_q$ 위의 임의의 두 다항식 $a(x)$, $b(x)$는 식 (1.22)와 같이 표현이 가능하다[35].

$$a(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{n-1} x^{n-1}$$
$$b(x) = b_0 + b_1 x + b_2 x^2 + \cdots + b_{n-1} x^{n-1} \quad (1.22)$$

위의 식 (1. 22)처럼 표현되는 다항식을 이용하는 암호화 과정 중, 가장 많은 연산시간을 소요하는 다항식 곱셈연산 $\mathbb{R}_q$ 에 대해 닫혀 있는 NTT를 이용하여 효율적인 연산이 가능하다.

그리하여 다수의 LWE 기반의 알고리즘들은 $\mathbb{Z}_q[x]/(x^n+1)$(n은 2의 제곱)의 Cyclotomic Ring을 사용하는데 이러한 환은 복잡도가 $O(n\log_2 n)$ 인 NTT를 사용하여 곱셈연산을 수행할 수 있다. NTT로 변환 없이 다항식 환의 두 원소의 곱셈연산은 $O(n^2)$ 단위 곱을 이용하나, NTT에서 곱셈연산은 $O(n)$ 단위의 곱이 필요하다. 이전에 서술하였듯이 다항식 곱셈연산은 연산처리시간이 상당히 많은 시간을 필요하기 때문에 이러한 곱셈연산을 효율적으로 연산 수행을 위해 많은 LWE Scheme이 NTT에서의 곱셈연산을 활용하기 위해 중간값을 NTT로 연산한다.

NTT 기반의 다항식 곱셈연산을 위해서는 두 가지 요소, 1의 원시적인 n 번째 제

곱근인 w 와 $\Psi \equiv w \bmod q$ 가 고려되어야 한다. w 가 n 차 원시 근이라 가정하였을 때 임의의 다항식 $polyA(x)$ 의 각 계수는 아래 식 (1.23), (1.24)과 같이 NTT 및 INTT로 표현된다.

$$A_i = \sum_{j=0}^{n-1} a_j w_n^{ij} \bmod q \ \text{(NTT)} \tag{1.23}$$

$$a_i = n^{-1} \sum_{j=0}^{n-1} A_j w_n^{-ij} \bmod q \ \text{(INTT)} \tag{1.24}$$

임의의 다항식 $polyA(x)$와 $polyB(x)$를 NTT로 변환을 수행한 후 Point-wise Multiplication을 수행, 다시 INTT로 변환과정을 수행함으로서 두 다항식 $polyA(x)$와 $polyB(x)$의 곱셈연산 결과를 얻을 수 있는데 이러한 과정 Time Domain이 아닌 Frequency Domain에서 다항식 곱셈연산을 통해 연산 수행 시간을 $O(n^2)$ 에서 $O(n)$으로 감소시킬 수 있게 된다. 아래의 식 (1.25), (1.26)은 임의의 두 다항식 $polyA(x)$와 $polyB(x)$의 곱셈연산 결과 $c(x)$를 NTT와 INTT로 표현한 식이다 [36][37].

$$c(x) = polyA(x) \times polyB(x) = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} a_i b_j x^{i+j} \tag{1.25}$$

$$c' = \mathrm{I} \qquad \mathrm{NTT}(\mathrm{NTT}(polyA') \times \mathrm{NTT}((polyB')) \tag{1.26}$$

이러한 NTT를 수행함에 있어서 중요한 연산과정이 Modulo Multiplication 혹은 Modulo Reduction이라고 불리는 연산과정이다. Modulo 연산은 어떠한 숫자를 다른 숫자로 나눈 나머지를 구하는 연산으로, 사칙연산 등 여러 가지 연산을 다수 수해할 경우 다항식의 계수가 유한체의 범위를 벗어나는 경우 유한체 내로 줄여주는 역할을 한다.

아래의 서술할 알고리즘들은 NTT를 계산할 경우에 주로 사용되는 알고리즘들 중 Montgomery Reduction과 Barrett Reduction이다.

1. Montgomery Multiplication

기본적으로 modulo 곱셈연산은 곱셈연산과 modulo 나눗셈 연산으로 나뉜다. ab 를 modulo R 에 modulo 나눗셈 연산한다는 것은 $ab = QR + q,\ 0 \leq q < R$ 을 만족하는 나머지 M 을 구하는 것을 의미하며 보통 $ab \bmod q$ 으로 표현한다. modulo 나눗셈의 방법에는 $2n$ 자리의 수를 n 자리 modulo R 으로 나누어 modulo 연산을 수행하는 고전적인 방법, 한 번에 몫을 추측하여 modulo 연산을 수행하는 barrett Reduction, modulo R 의 배수를 512개 선택하여 테이블에 저장하고 이를 이용하는 방식인 셀비 방법, 쉬프트 연산과 덧셈연산으로 나눗셈 연산을 대신하는 Montgomery Multiplication 등이 있다[38].

modulo 다항식의 NTT를 계산할 때, 최종 결과는 항상 modulo q 가 되어야 한다. 즉, q 보다 큰 정수에 대해서는 q 로 나눈 나머지 값을 계산해야 하는데, 두 정수의 합이나 차에 대해서는 입력 값에 상관없이 일정한 시간에 modulo q 한 값을 계산하는 것이 쉬우나, 곱셈연산의 결과에 대해서는 실수 연산을 사용하지 않고 일정한 시간에 modulo q 한 값을 얻는 것이 간단하지 않다. 이 때, 사용하는 방법이 Montgomery 곱셈연산이다[39][40].

NTT를 계산하기 위해 사용되는 곱셈연산은 모두 Montgomery 곱셈연산을 사용한다. $a \bmod q$ 와 $b \bmod q$ 의 곱셈연산을 결과 $ab \bmod q$ 를 얻고자 할 때, $a \bmod q$ 와 $b \bmod q$ 를 Montgomery 상의 레지듀 도메인 값인 $aR \bmod q$ 과 $bR \bmod q$ 으로 바꾸게 되면, $abR \bmod q$ 의 값을 입력 값에 상관없이 일정한 시간에 실수 연산을 하지 않고 얻을 수 있게 해주는 것이 Montgomery 곱셈연산 방법이다.

여기서, R 은 2^k 형태의 값을 사용하여 적절한 때에 곱셈연산을 bitwise shift 연산으로 대신하는 것이 가능하도록 하며, $\gcd(q,R) = 1$ 과 $R \geq q$ 의 조건을 만족시켜야한다. 서로 소인 두 자연수 q 와 R 에 대해서 Extended Euclidean 알고리즘을 적용하면 $R \bullet R' + q \bullet q' = 1$ 을 만족하는 정수 R' 와 q' 가 존재한다. 여기서 R' 와 q' 는 R 과 q 의 modulo q 와 modulo R 에 대한 곱셈연산에 대한 역원이 됨을 알 수 있다.

즉, $R \bullet R' = 1 \bmod q$ 이므로 $R' = R^{-1} \bmod q$ 이고, $q \bullet q' = 1 \bmod R$ 이므로 $q' = q^{-1} \bmod R$ 이다. $z = a \bullet b \bullet R^2$ 이라고 할 때, $z \bullet R^{-1} \bmod q$ 의 값을 다음 식 (1.27)과 같이 구할 수 있다.

$$z \cdot R^{-1} \bmod q = z \cdot R \cdot R^{-1} \cdot R^{-1} \bmod q \tag{1.27}$$

$$= z \cdot (1 - q \cdot q^{-1}) \cdot R^{-1} \bmod q$$

$$= (z - z \cdot q^{-1} \cdot q) \cdot R^{-1} \bmod q$$

$$= (z - (((z \bmod R) \cdot q^{-1}) \bmod R) \cdot q) \cdot R^{-1} \bmod q$$

위와 같이 계산된 값은 항상 $-q$ 와 q 사이에 있는 값이라는 것을 보일 수 있기 때문에, 이런 방식으로 두 수의 곱셈연산에 대해 modulo q 된 값을 구할 수 있다. 여기서 R이 2^k 형태이기 때문에 주어진 수에 대해서 modulo R을 취하거나, R을 나눈 값을 구하는 것은 bitwise and 연산과 shift 연산으로 간단하게 구현할 수 있다 [41].

알고리즘 5. Montgomery Reduction

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

and $0 < a_0 < \beta$

출력 : r' **where** $r' = \beta^{-1}a(\bmod q)$, **and** $-q < r' < q$

처리과정

1: $m \leftarrow a_0 q^{-1} (\bmod^{\pm} \beta)$

2: $t_1 \leftarrow \left\lfloor \frac{mq}{\beta} \right\rfloor$

3: $r' \leftarrow a_1 - t_1$

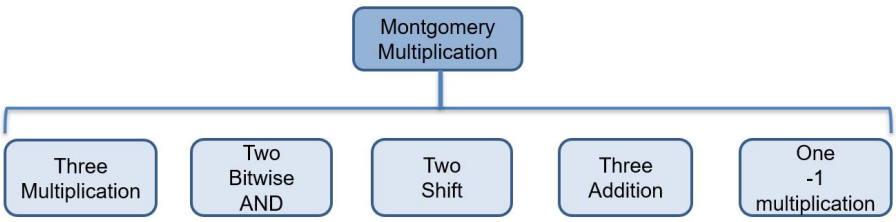

그림 3. Montgomery 곱셈연산 코스트

2. Barrett Reduction

Barrett Reduction 알고리즘은 modulo q 를 효율적으로 연산하는 방법으로 유명한 알고리즘이다. Reduction의 가장 직관적인 방법으로 알려진 연산은 나눗셈이 들어가는데, 이는 코스트가 많이 들어가기 때문에 Barrett은 $1/q$ 를 $m/2^k$ 의 근사치로 정하고 나눗셈을 오른쪽 쉬프트로 교체하여 수행하는 것을 제안했다. Barrett Reduction을 계산하는 첫 번째 방법으로는 k 를 선택하고 m 을 $2^k/q$ 보다 작거나 가장 근접한 정수로 설정하는 것이다. 아래의 알고리즘 6는 Barrett Reduction 알고리즘의 의사코드이다[42][43].

알고리즘 6. Barrett Reduction

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

출력 : r' **where** $r = a \pmod q$**, and** $0 \le r \le q$

1: $v \leftarrow \left\lfloor \frac{2^{\log(q)-1}.\beta}{q} \right\rfloor$

2: $t \leftarrow \left\lfloor \frac{av}{2^{\log(q)-1}.\beta} \right\rfloor$

3: $t \leftarrow tq \bmod \beta$

4: $r \leftarrow a - t$

F. 격자 기반의 양자 내성 키 교환 CRYSTAL-KYBER

본 절에서는 미국 NIST 양자내성암호 표준화 작업에서 채택된 우수한 격자기반 암호 알고리즘인 CRYSTAL-KYBER에 대해 설명한다. CRYSTAL-KYBER는 미국 국립표준기술연구원에서 진행한 NIST PQC Round3에서 Sujoy 등의 SABER팀과 함께 경쟁하였으나 CRYSTAL-KYBER가 채택되었다[44].

CRYSTAL-KYBER는 격자 기반의 알고리즘으로 Module-LWE를 사용하며 기본적으로 매개변수 n = 256, q = 3329를 고정으로 보안 레벨 1, 3, 5를 충족시키기 위

해 파라미터 k = 2, 3, 4를 사용하는 알고리즘을 제안하고 있다 미국 국립표준기술연구원에서는 보안 수준에 따른 보안 등급을 나누었는데 CRYSTAL-KYBER의 보안 등급은 아래의 표 5와 같다[44].

CRYSTAL-KYBER는 Ring-LWE의 다항식 $\mathbb{R}_q = \mathbb{Z}_q/\langle x^n+1\rangle$ 에서 파라미터 k를 추가하여 행렬 $\hat{\boldsymbol{A}} \in R_q^{k\times k}$, 다항식 벡터 $s \in R_q^k$, $e \in R_q^k$ 를 NTT (Number Theoretic Transform) 상에서 연산을 수행하는 암호 알고리즘이다. 크게 분류하자면 공개키(Public Key)와 비밀키(Secret Key)를 생성하는 Key Generation, 생성된 공개키와 사용자로부터 받은 평문(Plaintext)을 암호문(Ciphertext)으로 바꿔주는 Encryption 과정이 있으며, 생성된 비밀키와 암호문을 이용하여 원래의 평문으로 복원시키는 Decryption 과정으로 이루어진다[45].

표 5. NIST PQC CRYSTAL-KYBER의 파라미터

CRYSTAL-KYBER	KYBER-512	KYBER-768	KYBER-1024
NIST Security level	1	3	5
Parameter k	2	3	4
Dimension n	256	256	256
Modulus q	3329	3329	3329
Derived parameter δ	2^{-139}	2^{-164}	2^{-174}

1. CRYSTAL-KYBER에서 사용하는 알고리즘

CRYSTAL-KYBER의 CPAPKE는 Lyubashevsky 등이 Encrocrypt 2010에서 도입한 (Ring-LWE용) LPR 암호화 체계와 유사하다. 구체적인 설명은 논문의 전체 버전에 있다. 이러한 체계의 뿌리는 최초 LWE 기반 암호화로 거슬러 올라가는데 Regev가 [46]에서 제시한 방식으로 주요한 차이점은 기본 링이 $\mathbb{Z}_q$ 가 아니며 비밀 벡터와

오류 벡터 모두 작은 계수를 갖는다는 점이다[46][47].

다항식 링($\mathbb{Z}_q$ 대신에)을 사용하는 이 아이디어는 [48]에서 Hoffstein 등이 제시한 NTRU 암호 시스템으로 거슬러 올라가야 하며 비밀과 오류 사이의 대칭은 [48]에서 매우 유사한 암호 체계[49, 50]에서 [51]의 보안 정당성과 함께 이미 사용되어왔다.

LPR 암호화 방식과의 주요 차이점은 Ring-LWE 대신 Module-LWE를 사용한다는 점에 있다. 또한 CRYSTAL-KYBER의 공개 행렬 $\boldsymbol{A}$ 를 생성하기 위해 [52]에서 Alkim 등이 취한 접근방식을 채택한다. 또한 Learning With Roungding 기반의 방식[53]은 LWE 기방 방식의 암호문에서도 크기를 줄일 수 있는 일반적인 기술이다[54][55].

이러한 키 생성 과정은 seed 값을 받은 해시 함수를 이용하여 생성하는 의사난수(pseudorandom number)를 Rejection Sampling 과정을 통해 생성된 계수 값들을 NTT를 이용해 $\hat{s}, \hat{e}$ 를 생성하고 sampling을 한 후 생성된 값을 $\hat{\boldsymbol{A}} \cdot \hat{s} + \hat{e}$ 연산을 수행 후 값들을 생성하는데 $\hat{t}$ 를 인코딩을 할 경우 공개키를 생성하고 $\hat{s}$ 를 인코딩할 경우 비밀키를 생성하게 되는 과정으로 수행된다[56].

1) CRYSTAL-KYBER의 CPAPKE Key generation

알고리즘 7. CRYSTAL-KYBER.CPAPKE.KeyGen(): Key Generation

출력 : 비밀키 $sk \in B^{12 \cdot k \cdot n/8}$

출력 : 공개키 $pk \in B^{12 \cdot k \cdot n/8 + 32}$

1: $d \leftarrow B^{32}$
2: $(\rho, \sigma) := G(d)$
3: $N := 0$
4: **for** i from 0 to $k-1$ **do** > **NTT 도메인 내에서**
5: **for** j from 0 to $k-1$ **do** **행렬** $\hat{\boldsymbol{A}} \in R_q^{k \times k}$ **생성**
6: $\hat{\boldsymbol{A}}[i][j] := Parse(XOF(p,j,i))$
7: **end for**
8: **end for**
9: **for** i from 0 to $k-1$ **do**

10: $s[i] := CBD_{n_1}(PRF(\sigma, N))$ > 샘플 $s \in R_q^k$ **from** B_{η_1}

11: $N := 0$

12: **end for**

13: **for** i from 0 to $k-1$ **do**

14: $e[i] := CBD_{n_1}(PRF(\sigma, N))$ > 샘플 $e \in R_q^k$ **from** B_{η_1}

15: $N := 0$

16: **end for**

17: $\hat{s} := NTT(s)$

18: $\hat{e} := NTT(e)$

19: $\hat{t} := NTT(t)$

20: $pk := (Encode_{12}(\hat{t} \bmod^+ q) \| p)$ > $pk := \boldsymbol{A}s + e$

21: $sk := Encode_{12}(\hat{s} \bmod^+ q)$ > $sk := s$

22: return (pk, sk)

2) CRYSTAL-KYBER의 CPAPKE Encryption Algorithm

CRYSTAL-KYBER의 Encryption 알고리즘은 앞의 Key Generation 알고리즘에서 생성한 공개키와 임의의 값 coins, 암호화 할 평문 Plaintext를 입력으로 받고 공개키를 디코딩을 통해 publicseed와 $\hat{t}$ 로 분리, Key Generation 알고리즘과 동일하게 공개키에서 분리된 publicseed를 이용하여 Rejection Sampling 단계를 거쳐 행렬 값 $\hat{\boldsymbol{A}}$ 를 생성한다. coins는 noiseseed와 함께 r, e_1, e_2 를 생성하는데 사용되고 이렇게 생성한 값들을 NTT를 이용해서 $(\hat{\boldsymbol{A}}^T \circ \hat{r}) + e_1$ 과 $(\hat{t}^T \circ \hat{r}) + e_2$ 연산을 수행, INTT 과정을 거쳐 u, v 를 얻은 후 이를 인코딩하여 암호문을 출력하는 과정으로 수행된다 [57].

알고리즘 8. CRYSTAL-KYBER.CPAPKE.Enc(pk, m, r): Encryption

입력 : 공개키 $pk \in B^{12 \cdot k \cdot n/8+32}$

입력 : 메시지 $m \in B^{32}$

입력 : 임의의 coins $r \in B^{32}$

출력 : 암호문 $c \in B^{d_u \cdot k \cdot n/8 + d_v \cdot n/8}$

$N := 0$

$\hat{t} := Decode_{12}(pk)$

$\rho := pk + 12 \cdot k \cdot n/8$

for i from 0 to $k-1$ **do** > **NTT 도메인 내에서**

  for j from 0 to $k-1$ **do** **행렬** $\hat{\boldsymbol{A}} \in R_q^{k \times k}$ **생성**

    $\hat{\boldsymbol{A}}^T[i][j] := Parse(XOF(p,i,j))$

  end for

end for

for i from 0 to $k-1$ **do**

  $r[i] := CBD_{n_1}(PRF(r,N))$ > **샘플** $r \in R_q^k$ **from** B_{η_1}

  $N := N+1$

end for

for i from 0 to $k-1$ **do**

  $e_1[i] := CBD_{n_2}(PRF(r,N))$ > **샘플** $e_1 \in R_q^k$ **from** $B_{\eta 2}$

  $N := N+1$

end for

$e_2[i] := CBD_{n_2}(PRF(r,N))$ > **샘플** $e_2 \in R_q^k$ **from** $B_{\eta 2}$

$\hat{r} := NTT(r)$

$u := NTT^{-1}(\hat{\boldsymbol{A}}^{\boldsymbol{T}} \circ \hat{r}) + e_1$ > $u := \boldsymbol{A}^{\boldsymbol{T}} r + e_1$

$v := NTT^{-1}(\hat{t}^T \circ \hat{r}) + e_2 + Decompress_q(Decode_1(m),1)$

$c_1 := Encode_{d_u}(Compress_q(u,d_u))$

$c_2 := Encode_{d_v}(Compress_q(v,d_v))$

return $c = (c_1 \| c_2)$

3) CRYSTAL-KYBER의 CPAPKE Decryption Algorithm

CRYSTAL-KYBER의 Decryption 알고리즘은 앞의 Key Generation 과정에서 생성한 비밀키를 디코딩 한 $\hat{s}$와 Encryption 알고리즘에서 생성한 암호문을 디코딩을 통해 분리한 u 를 NTT에서 곱셈연산 한 후 INTT한 값을 v 로부터 빼고 난 후 인코딩 과정을 진행하여 원래의 평문을 출력하게 되는 알고리즘이다[58].

알고리즘 9. CRYSTAL-KYBER.CPAPKE.Dec(sk, c): decryption

입력 : 비밀키 $sk \in B^{12 \cdot k \cdot n/8}$

입력 : 암호문 $c \in B^{d_u \cdot k \cdot n/8 + d_v \cdot n/8}$

출력 : 메시지 $m \in B^{32}$

1: $u := Decompress_q(Decode_{d_u}(c), d_u)$

2: $v := Decompress_q(Decode_{d_v}(c + d_u \cdot k \cdot n/8), d_v)$

3: $\hat{s} := Decode_{12}(sk)$

4: $m := Encode_1(Compress_q(v - NTT^{-1}(\hat{s}^T \circ NTT(u)),1))$

5: return m

4) CRYSTAL-KYBER의 CCAKEM Algorithms

CRYSTAL-KYBER에서 사용되는 약간 수정된 Fujisaki-Okamoto Transformation을 통해 IND-CPA 보안 공개키 암호화 방식에서 Kyber의 CCAKEM-INDCCA2 보안 KEM을 구성하였는데 아래의 알고리즘 10, 11, 12는 의사코드는 Kyber.CCAKEM의 키 생성, 캡슐화 및 캡슐화 해제에 대해 서술한다[59].

알고리즘 10. CRYSTAL-KYBER.CCAKEM.KeyGen()

출력 : 공개키 $pk \in B^{12 \cdot k \cdot n/8 + 32}$

출력 : 비밀키 $sk \in B^{24 \cdot k \cdot n/8 + 96}$

1: $z \in B^{32}$

2: $(pk, sk') :=$ KYBER.CPAPKE.KeyGen()

3: $sk := (sk' || pk || H(pk) || z)$

4: return (pk, sk)

알고리즘 11. CRYSTAL-KYBER.CCAKEM.Enc(pk)

입력 : 공개키 $pk \in B^{12 \cdot k \cdot n/8+32}$

출력 : 암호문 $c \in B^{d_u \cdot k \cdot n/8 + d_v \cdot n/8}$

출력 : 공유된 키 $K \in B^*$

1: $m \leftarrow B^{32}$

2: $m \leftarrow H(m)$

3: $(\overline{K}, r) := G(m || H(pk))$

4: $c :=$ KYBER.CPAPKE.Enc(pk, m, r)

5: $K := KDF(\overline{K} || H(c))$

6: return (c, K)

알고리즘 12. CRYSTAL-KYBER.CCAKEM.Dec(c, sk)

입력 : 암호문 $c \in B^{d_u \cdot k \cdot n/8 + d_v \cdot n/8}$

입력 : 비밀키 $sk \in B^{24 \cdot k \cdot n/8 + 96}$

출력 : 공유된 키 $K \in B^*$

1: $pk := sk + 12 \cdot k \cdot n/8$

2: $h := sk + 24 \cdot k \cdot n/8 + 32 \in B^{32}$

3: $z := sk + 24 \cdot k \cdot n/8 + 64$

4: $m' :=$ KYBER.CPAPKE.Dec$(s, (u, v))$

5: $(\overline{K}', r') := G(m' || h)$

6.: $c' :=$ KYBER.CPAPKE.Enc(pk', m', r')

7: **if** $c = c'$ **then**

8: **return** $K := KDF(\overline{K}' || H(c))$

9: **else**
10: **return** $K := KDF(z \| H(c))$
11: **end if**
12: **return** K

2. CRYSTAL-KYBER에서 사용하는 세부 알고리즘

1) Uniform Sampling in R_q

CRYSTAL-KYBER는 통계적으로 균일한 무작위 분포에 가까운 $\mathbb{R}_q$ 의 샘플 요소에 결정론적 방법을 사용하는데 이러한 샘플링을 위해 KYBER 팀은 바이트 스트림 $B = b_0, b_1, b_2 ...$를 입력으로 수신하고 NTT-Representation $\hat{a} = \hat{a}_0 + \hat{a}_1 X + \cdots + \hat{a}_{n-1} X^{n-1} \in \mathbb{R}_q$ of $a \in \mathbb{R}_q$ 를 계산하는 Parse : $B^* \rightarrow \mathbb{R}_q$함수를 사용합니다. 이러한 과정은 SHA3-512로 생성한 publicseed 32바이트를 SHAKE128의 초기 입력 값을 받아 사용하는데 SHAKE128에서 absorb 함수를 거쳐 패딩된 입력을 총 25개의 state에 저장, Squeeze 과정을 통해 168바이트의 버퍼를 생성한다. 168 바이트 버퍼는 16비트 단위로 Rejection-Uniform을 수행한다. 아래의 알고리즘 13는 Rejection Sampling 과정에 대한 알고리즘이다.

알고리즘 13. Rejection Sampling Parse : $B^* \rightarrow \mathbb{R}_q$ (q = 3329)

입력 : 바이트 스트림 $B = b_0, b_1, b_2 ... \in B^*$

출력 : NTT Representation $\bar{a} \in \mathbb{R}_q$ **of** $a \in \mathbb{R}_q$

1: $i := 0$
2: $j := 0$
3: while $j < n$ do
4: $d_1 := b_i + 256 \cdot (b_{i+1} \bmod^+ 16)$
5: $d_2 := \lfloor b_{i+1}/16 \rfloor + 16 \cdot b_{i+2}$

6: if $d_1 < q$ then
7: $\hat{a}_j := d_1$
8: $j := j+1$
9: end if
10: if $d_2 < q$ and $j < n$ then
11: $\hat{a}_j := d_2$
12: $j := j+1$
13: end if
14: $i := i+3$
15: end while
16: return $\hat{a}_0 + \hat{a}_1 X + \cdots + \hat{a}_{n-1} X^{n-1}$

2) Binomial Sampling 알고리즘

CRYSTAL-KYBER는 의사 난수 함수의 출력 64η 바이트에서 결정론적으로 Bη에 따라 다항식 $f \in r_q$ 가 샘플링 되는 방법을 정의하는데 아래의 알고리즘 14에 서술되어 있는 중심 이항 분포(Centered Binomial Distribution)을 이용하여 수행한다 [60].

알고리즘 14. CBDη : $B^{64\eta} \rightarrow \mathbb{R}_q$ (n coefficient = 256)

입력 : 바이트 배열 $B = (b_0, b_1, \ldots b_{64\eta-1} \in B^{64\eta}$

출력 : 다항식 $f \in \mathbb{R}_q$

1: $(\beta_0, \ldots, \beta_{512\eta-1}) :=$BytesToBits(B)
2: **for** i **from** 0 to 255 **do**
3: $a := \sum_{j=0}^{\eta-1} \beta_{2i\eta+j}$
4: $b := \sum_{j=0}^{\eta-1} \beta_{2i\eta+\eta+j}$

5: $f_i := a - b$

6: **end for**

7: **return** $f_0 + f_1X + f_2X^2 + \cdots + f_{255}X^{255}$

3) Decode 알고리즘

CRYSTAL-KYBER가 바이트 배열로 직렬화 해야하는 두가지 데이터의 유형이 있는데 바이트 배열과 벡터 다항식이다. 바이트 배열은 ID를 통해 간단하게 직렬화 되므로 다항식을 직렬화 및 역직렬화 하는 방법을 정의해야하는데 알고리즘 15에서 이러한 Decode 알고리즘에 대한 의사코드를 제공하는데 이것은 $32l$ 바이트의 배열 $\{0, \ldots, 2^l - 1\}$을 각 계수 f_i 와 함께 다항식 $f_0 + f_1X + f_2X^2 + \cdots + f_{255}X^{255}$ 로 역직렬화한다. 또한 CRYSTAL-KYBER에선 Encode를 Decode의 역으로 정의 시킨다.

알고리즘 15. Decode$_l$: $B^{32l} \rightarrow R_q$

입력 : 바이트 배열 $B \in B^{32l}$

출력 : 다항식 $f \in \mathbb{R}_q$

1: $(\beta_0, \ldots, \beta_{512\eta-1}) :=$BytesToBits(B)

2: **for** i from 0 to 255 **do**

3: $f_i := \sum_{j=0}^{l-1} \beta_{il+j} 2^j$

4: **end for**

5: **return** $f_0 + f_1X + f_2X^2 + \cdots + f_{255}X^{255}$

III. 제안하는 알고리즘

A. Rainbow의 효율적 $\mathbb{F}_{16}$ 상의 곱셈연산 설계

본 장에서는 양자 내성 암호 알고리즘 PQC 중 Finalist 알고리즘까지 진출하였고 그 중 유일한 다변수이차방정식 기반의 전자서명 알고리즘인 Rainbow의 효율적인 유한체 연산 방법을 제안한다.

Rainbow는 기저체로 $\mathbb{F}_{16}$ 을 사용하는데 본 장에서는 Rainbow의 $\mathbb{F}_{16}$ 상에서의 곱셈연산 연상을 직접 구현함으로써 Riera가 제시한 $\mathbb{F}_{16}$ 에서 테이블 룩업을 통해 곱의 역원을 구하는 방법보다 성능을 높인 2021년 Chou의 제안보다 높은 성능을 구현하는 연산방법을 제시한다. 특히 시간차에 기반한 부채널 분석에 내재적 저항성을 갖도록 중간값에 상관없이 항상 일정한 시간을 갖는 조건 분기가 없는 연산 방법이 필요하다. $\mathbb{F}_{16}$ 연산을 가속하는 것으로 전체 서명시간을 단축시킬 수 있다 [61][62].

$\mathbb{F}_{16}$ 연산은 4비트 크기의 원소를 다루는 연산으로 32비트의 MCU환경에서는 한 개의 변수에 4비트 원소를 8개를 배열하는 것이 가능하다. 실제 곱셈연산에서는 각 비트마다 비트 단위의 연산이 가능하므로 32개의 원소를 가지고 한 변수에 대입하고 각 비트에 대하여 MCU가 제공하는 비트 단위의 AND/XOR 연산을 이용하면 32개의 연산을 병렬로 동시에 처리할 수가 있다.

본 장에서 제안하는 방법 역시 같은 방법에 의해 성능을 높이는 것이 가능하며, 제안하는 방법에 사용되는 AND/XOR 연산을 비트슬라이스로 구현할 경우 병렬화를 통한 가속 효과를 마찬가지로 얻을 수 있다.

2개의 $\mathbb{F}_{16}$ 요소가 1비트에 맞듯이 8개의 $\mathbb{F}_{16}$ 요소를 하나의 32비트 레지스터에 맞출 수 있는데, 위에서 언급한 것 과 같이 필드 요소를 총 32개를 갖는 4개의 개별 레지스터로 비트슬라이스 할 때 일정한 시간에 실행되는 훨씬 빠른 병렬 $\mathbb{F}_{16}$ 곱셈연산을 달성할 수 있다.

빠른 비트슬라이스 곱셈연산을 사용하기 위해선 $\mathbb{F}_{16}$ 요소의 압축된 니블 표현을 Bitslicing 표현으로 바꾸는 작업이 필요한데, 간단하게 접근하자면 각 필드 요소를

개별적으로 로드하고 원하는 비트를 마스킹한 후 입력과 동일한 순서로 해당 레지스터에 압축하는 것이나 32개의 요소를 한 번에 4개의 레지스터에 로드하고 그림 4와 같이 인터리브 방식으로 요소를 재구성하는 것이 훨씬 효율적이다. 아래 그림 4에서의 각 행은 8개의 필드 요소를 포함하는 레지스터를 표현한다.

색상이 밝은 회색은 LSB이고 진한 회색이 MSB인 필드 요소내의 비트를 나타내는데, 아래의 의사코드에 표시된 대로 28개의 사이클로 구성된 효율적 구현이 가능하다. 밑의 알고리즘 16은 28개의 사이클로 효율적으로 표현한 일반표현에서 비트슬라이스로, 비트슬라이스에서 일반표현으로의 변환 과정을 나타낸 의사코드이다.

$\mathbb{F}_{16}$ 의 덧셈 연산은 Bitwise-XOR이므로 비트슬라이스 표현에서 동일하게 동작한다.

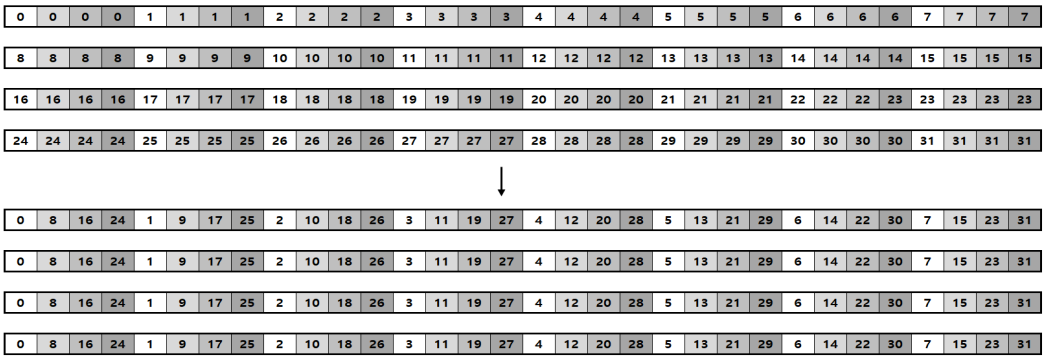

그림 4. 인터리브 방식으로 재구성한 8개의 필드 요소를 포함한 레지스터 구성

알고리즘 16. $\mathbb{F}_{16}$ 의 일반표현에서 비트슬라이스 혹은 비트슬라이스에서 일반표현으로의 변환

입력 : 32bits $\mathbb{F}_{16}$ elements in a_0, a_1, a_2, a_3

출력 : Bitsliced $\mathbb{F}_{16}$ elements in b_0(LSB), b_1, b_2, b_3 (MSB), t(temp)

1. and b_0, a_0, #0x11111111
2. and t, a_1, #0x11111111
3. orr b_0, b_0, t, lsl#1
4. and t, a_2, #0x11111111
15. and b_2, a_2, #0x44444444
16. and t, a_0, #0x44444444
17. orr b_2, b_2, t, lsr#2
18. and t, a_1, #0x44444444

5. orr b_0, b_0, t, lsl#2	19. orr b_2, b_2, t, lsr#1
6. and t, a_3, #0x11111111	20. and t, a_3, #0x44444444
7. orr b_0, b_0, t, lsl#3	21. orr b_2, b_2, t, lsr#1
8. and b_1, a_1, #0x22222222	22. and b_2, a_2, #0x88888888
9. and t, a_0, #0x22222222	23. and t, a_0, #0x88888888
10. orr b_1, b_1, t, lsr#1	24. orr b_3, b_3, t, lsr#3
11. and t, a_2, #0x22222222	25. and t, a_1, #0x88888888
12. orr b_1, b_1, t, lsr#1	26. orr b_3, b_3, t, lsr#2
13. and t, a_3, #0x22222222	27. and t, a_2, #0x88888888
14. orr b_1, b_1, t, lsr#2	28. orr b_3, b_3, t, lsr#1

Rainbow에서는 AES의 S-box에서 널리 이용되고 있던 방식인 Tower Field의 형태로 유한체 연산을 수행하는데 이때 $\mathbb{F}_{16}$ 은 부분체 $\mathbb{F}_4$ 를 사용해서 다음 식 (3.1)과 같이 나타낼 수 있다.

$$\mathbb{F}_{16} := \mathbb{F}_4[y]/(y^2+y+\beta) \tag{3.1}$$

여기서 다시 $\mathbb{F}_4$ 는 하위체 $\mathbb{F}_2$ 를 사용해서 다음 식 (3.2)와 같이 나타낼 수 있다.

$$\mathbb{F}_4 := \mathbb{F}_2[x]/(x^2+x+\beta) \tag{3.2}$$

$\mathbb{F}_4$ 상의 두 원소를 $a=a_1x+a_0$ 와 $b=b_1x+b_0$ 라 하자. 그러면 두 원소의 곱은 다음 식 (3.3), (3.4)와 같이 나타낼 수 있다.

$$c=c_1x+c_0=ab=(a_1x+a_0)(b_1x+b_0) \tag{3.3}$$

$$ab=(a_1b_0+a_0b_1+a_1b_1)x+(a_0b_0+a_1b_1) \tag{3.4}$$

$\mathbb{F}_{16}$ 상의 두 원소의 곱 ef 는 두 개의 $\mathbb{F}_4$ 상의 원소의 곱셈연산을 이용해서 구

할 수 있다. 여기서 $e = e_1 y + e_0,\ f = f_1 y + f_0,\ e_i, f_i \in F_4$ 이다. 그러면 $g_i \in F_4$ 에 대해 $ef = g_1 y + g_0$로 나타낼 수 있고 $c_i \in \mathbb{F}_2$ 에 대해 식 (3.5)

$$g_0 = c_1 x + c_0,\ g_1 = c_3 x + c_2 \tag{3.5}$$

로 나타낼 수 있다. 이때 $a_i, b_i \in \mathbb{F}_2$ 에 대해 식 (3.6)과 (3.7)

$$e_1 = a_3 x + a_2,\ e_0 = a_1 x + a_0 \tag{3.6}$$

$$f_1 = b_3 x + b_2,\ f_0 = b_1 x + b_0 \tag{3.7}$$

라 하면, $\mathbb{F}_{16}$ 상에서의 두 원소 e 와 f 의 곱은 다음과 같은 관계가 성립한다.

$$c_0 = a_0 b_0 + a_1 b_1 + a_3 b_2 + a_2 b_3 + a_3 b_3 \tag{3.8}$$

$$c_1 = a_1 b_0 + a_0 b_1 + a_1 b_1 + a_2 b_2 + a_3 b_2 + a_2 b_3 \tag{3.9}$$

$$c_2 = a_2 b_0 + a_3 b_1 + a_0 b_2 + a_2 b_2 + a_1 b_2 + a_3 b_3 \tag{3.10}$$

$$c_3 = a_3 b_0 + a_2 b_1 + a_3 b_1 + a_1 b_2 + a_3 b_2 + a_0 b_3 + a_1 b_3 + a_2 b_3 + a_3 b_3 \tag{3.11}$$

이것을 다시 반복되는 연산에 따라 분류하면 다음 식 (3.12)와 같이 변경할 수 있다.

$$\begin{aligned} t_0 &= a_3 (b_2 + b_3) \\ t_1 &= a_1 b_1 + a_2 b_3 \\ t_2 &= a_3 b_1 + a_1 b_2 \\ c_0 &= a_0 b_0 + t_1 + t_0 \\ c_1 &= a_1 b_0 + a_0 b_1 + t_1 + (a_2 + a_3) b_2 \\ c_2 &= a_2 b_0 + a_0 b_2 + t_2 + a_2 b_2 + a_3 b_3 \\ c_3 &= a_3 b_0 + a_2 b_1 + (a_0 + a_1 + a_3) b_3 + t_0 \end{aligned} \tag{3.12}$$

[62]에서 Chou 등이 제안한 방법에서 필요한 연산의 개수는 16개의 AND 연산과 22개의 XOR 연산이었으나, 본 논문에서는 위의 식과 같이 연산 순서를 재조정함으로써 16개의 AND 연산을 유지하면서 XOR 연산을 19개로 감소시키는 것이 가능하였다.

곱셈연산의 역원은 $ab=c$에서 주어진 a 에 대해 $c=(0,0,0,1)$ 이 되는 b 를 찾는 문제로 정의할 수 있다. 곱셈연산의 관계에서 위의 관계는 다시 아래의 식 (3.13)과 같은 행렬로 표현할 수 있다.

$$\begin{bmatrix} a_0 & a_1 & a_3 & a_2+a_3 \\ a_1 & a_0+a_1 & a_2+a_3 & a_2 \\ a_2 & a_3 & a_0+a_2 & a_1+a_3 \\ a_3 & a_2+a_3 & a_1+a_3 & a_0+a_1+a_2+a_3 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{3.13}$$

즉, 좌측 행렬을 A라 하면 $Ab=c$ 에서 $b=A^{-1}c$ 를 연산하는 문제로 바꿀 수 있다. 이때 4x4 행렬 $A=[a_{i,j}]$ (여기서 $0 \le i,j \le 3$)는 $\mathbb{F}_2$ 상에서의 Gauss 소거법을 통해 구현할 수 있으며, $\mathbb{F}_2$ 상의 연산을 통해 구현하는 것이 가능하다.

B. NTT에서의 효율적 연산을 통한 최적화

본 장에서는 여러 다항식 환의 원소 간에 곱셈연산이 필요한 다양한 분야에 사용되고, 특히 양자내성암호에서 격자 기반의 알고리즘에서 효율적인 연산을 위해 필수적으로 사용되며 현재 NIST에서 표준으로 선발된 알고리즘 및 진행 중인 Round의 다수의 알고리즘이 사용하는 NTT (Number Theoretic Transfrom)에서 Montgomery 곱셈연산을 줄여 최적화하는 방법에 대해서 제시한다.

1. Radix 2의 Lazy 연산을 이용한 최적화

Montgomery 곱셈연산에서 사용하는 R 값을 선택하며 $R>q^2$과 같은 조건을 만족시키는 값 R 을 선택할 경우, 첫 곱셈연산에서는 Montgomery 곱셈연산을 하지 않더라도, 다음 단계의 두 번째 곱셈연산에서 Montgomery 곱셈연산을 실시함에 따라

그 결과로 한 번에 modulo q가 된 값을 획득 할 수 있게 된다[63][64].

이것을 일반적으로 적용시켜 $R > q^t$ 의 조건으로 확장 할 경우 총 t 번의 곱셈연산을 할 때, $t-1$ 번의 곱셈연산에 대해서는 Montgomery 곱셈연산을 수행하지 않고, 마지막 한 번의 곱셈연산에만 Montgomery 곱셈연산을 사용하는 방식으로 확장시킬 수 있다. 다항식의 NTT를 계산할 때, 가장 기본적으로 많이 사용하는 구조인 Radix 2 를 방식으로 구현하는 것이다. 예를 들어, $\mathbb{Z}_q[x]/f(x)$ 에서 $f(x)$ 가 8차인 다항식이라고 할 때, 이 다항식 환에 속하는 원소 $g(x)$를 식 (3.14)와 같이,

$$g(x) = g_0 + g_1x + g_2x^2 + g_3x^3 + g_4x^4 + g_5x^5 + g_6x^6 + g_7x^7 \tag{3.14}$$

있다고 하자. 위의 원소의 다항식을 Radix 2 인 NTT 를 계산을 할 때, 아래의 식 (3.15)로 표현할 수 있는데,

$$g(x) = ((g_0 + g_4x^4) + x^2(g_2 + g_6x^4)) + x((g_1 + g_5x^4) + x^3(g_3 + g_7x^4) \tag{3.15}$$

이와 같이 전개, Radix 2 구조처럼 정리할 수 있는데 단계별로 정리하자면,

1단계 : 아래의 식 (3.16)과 같이 정리하여 계산

$$A_0 = (g_0 + g_4x^4),\ A_1 = (g_1 + g_5x^4),\ A_2 = (g_2 + g_6x^4),\ A_3 = (g_3 + g_7x^4) \tag{3.16}$$

2단계 : 아래의 식 (3.17)과 같이 정리하여 계산

$$B_0 = (A_0 + A_2x^2),\ B_1 = (A_1 + A_3x^2) \tag{3.17}$$

3단계 : 아래의 식 (3.18)과 같이 정리하여 계산

$$C_0 = (B_0 + B_1x) \tag{3.18}$$

이러한 방식으로 Radix 2 구조와 같이 정리할 수 있다.

$g(x)$의 NTT를 계산한다는 것은, $g(\alpha^{2i+1})$, $0 \le i \le 7$, 의 값을 구한다는 것이며, $\alpha \in \mathbb{Z}_q$ 이고, $\alpha^8 = 1$, $\alpha^4 = -1$ 의 조건을 만족시킨다. 그러므로 x, x^2, x^4 이 될 수 있는 값들은 다음의 표 6과 같이 나타낼 수 있다. x 를 곱한다는 것은 $\alpha, \alpha^3, \alpha^5, \alpha^7$ 의 네 가지 값들을 곱한다는 것을 의미하므로 네 번의 곱셈연산을 해야 한다는 것을 의미한다. x^2 을 곱한다는 것은 α^2, α^6 의 두 가지 값들을 곱한다는 것을 의미하므로 두 번의 곱셈연산을 수행해야 한다는 것을 의미한다. x^4 을 곱한다는 것은 α^4 의 값을 곱한다는 것을 의미하므로 즉, 한 번의 곱셈연산을 해야 한다는 것을 의미한다.

표 6. x, x^2, x^4가 될 수 있는 값들의 목록

	a	a^3	a^5	a^7	$-a$	$-a^3$	$-a^5$	$-a^7$
x^4	a^4	$-a^4$	a^4	$-a^4$	a^4	$-a^4$	a^4	$-a^4$
x^2	a^2	a^6	$-a^2$	$-a^6$	a^2	a^6	$-a^2$	$-a^6$
x	a	a^3	a^5	a^7	$-a$	$-a^3$	$-a^5$	$-a^7$

A_0 를 구하는 과정에서 발생하는 곱셈연산인 $g_4 x^4$ 는 NTT의 마지막 단계까지 더 이상 곱셈연산이 없으므로 Montgomery 곱셈연산으로 계산을 해줘야 NTT연산의 출력 값 modulo q 를 얻을 수 있다. 하지만, A_2 을 구하는 과정에서 발생하는 $g_6 x^4$ 의 곱셈연산에서는 Montgomery 곱셈연산을 하지 않더라도 다음 단계에서 x^2을 곱하는 과정에서 Montgomery 곱셈연산을 해주면 modulo q 가 된 값을 얻을 수 있다. 마찬가지로, A_1 을 구하는 과정에서 발생하는 $g_5 x^4$ 의 곱셈연산에서는 Montgomery 곱셈연산을 하지 않더라도 다음 단계에서 x 을 곱하는 과정에서 Montgomery 곱셈연산을 해주면 modulo q 가 된 값을 얻을 수 있다.

그리고 B_1 을 구하는 과정에서 발생하는 $A_3 x^2$ 의 곱셈연산에서는 Montgomery 곱셈연산을 하지 않더라도 다음 단계에서 x 를 곱하는 과정에서 Montgomery 곱셈연산을 해주면 modulo q 가 된 값을 얻을 수 있는데, 아래의 그림 5는 이러한 과정을 표현한다.

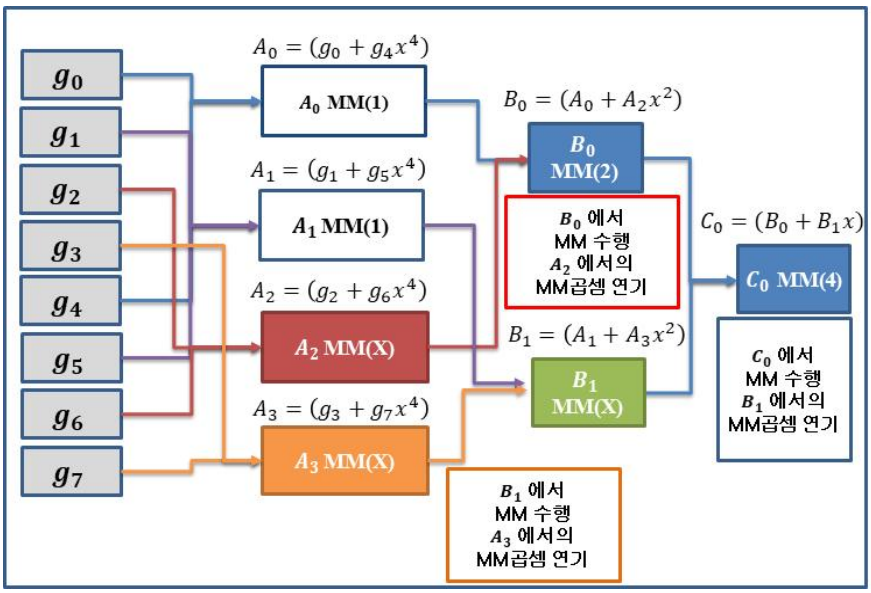

그림 5. Radix-2 Lazy 연산을 통한 최적화 방안

여기서 x^2 을 곱한다는 것은 두 번의 곱셈연산을 내포하고 있다. 앞의 내용을 정리하자면, 전체 12번의 곱셈연산 중 4번의 계산할 때, 곱셈연산의 결과에 대해 더 이상의 곱셈연산이 필요하지 않은 경우에 대해서는 Montgomery 곱셈연산을 실시하고, 마지막 곱셈연산에서부터 그 전의 곱셈연산에 대해서는 Montgomery 곱셈연산을 실시하지 않으며, 그 전의 곱셈연산에 대해서는 Montgomery 곱셈연산을 실시하는 방식으로 번갈아가며 Montgomery 곱셈연산을 적용하여 Montgomery 곱셈연산의 횟수를 줄일 수 있다.

이 방식을 일반화하면 $f(x)$ 의 차수가 2^k 인 경우 Radix 2 로 구현한 NTT의 경우 $k \cdot 2^{k-1}$ 번의 Montgomery 곱셈연산을 하게 되는데, 이 중에 $(k-1) \cdot 2^{k-2}$ 번의 곱셈연산에 대해서는 Montgomery 곱셈연산 대신 일반 곱셈연산으로 대체할 수 있다. 총 k개의 단계가 필요한데, 각 단계 마다 2^{k-1} 번의 Montgomery 곱셈연산을 실시하고 있기 때문에 총 $k \cdot 2^{k-1}$ 번의 곱셈연산이 필요했는데, 앞에서 제시한 방법에 따르면 마지막 단계를 제외한 $k-1$ 개의 단계에서 기존 곱셈연산의 절반만 Montgomery 곱셈연산을 시행해도 되기 때문에 각 단계마다 2^{k-2} 개의 곱셈연산은 일반 곱셈연산으로 실시할 수 있다. 그리하여 총 $(k-1) \cdot 2^{k-2}$ 개의 곱셈연산은 일

반 곱셈연산으로 실시할 수 있게 된다.

Radix 2로 구현된 NTT에서 $f(x)$의 차수가 2^k 인 경우, 1 단계부터 k 단계까지 각 단계별로 곱셈연산을 진행하여 최종 결과를 구하게 된다. 이때, 다항식 환의 원소 $g(x)$는 Radix 2 인 NTT를 구현하기 위해 다음 식 (3.19)과 같이 나타낼 수 있다.

$$g(x)=\sum_{i_k=0}^{1} x^{i_k}\Big(\sum_{i_{k-1}=0}^{1} x^{2i_{k-1}}\Big(\sum_{i_{k-2}=0}^{1} x^{2^2 i_{k-2}}\Big(\ldots\Big(\sum_{i_1=0}^{1} x^{2^{k-1}i_1} g_{2^{k-1}i_1+\ldots+2^2 i_{k-2}+2i_{k-1}+i_k}\Big)\Big)\Big)\Big) \qquad (3.19)$$

가장 마지막 k 단계의 곱셈연산은 기존 방식대로 모두 Montgomery 곱셈연산으로 실시하여 계산 결과 modulo q 가 된 값을 얻을 수 있도록 한다. $k-1$ 단계에서는 총 2^{k-1} 번의 곱셈연산이 필요한데, 이 곱셈연산의 결과가 다음 k 단계로 넘어 갔을 때, 추가적인 곱셈연산을 해야 하는 경우가 있고, 더 이상 추가적인 곱셈연산을 하지 않아도 되는 경우가 있다. 2^{k-1} 번의 곱셈연산의 결과 중에 2^{k-2} 번의 곱셈연산 결과에 대해서는 다음 단계로 넘어 갔을 때, 추가적인 곱셈연산이 필요하다.

그러므로 이런 2^{k-2} 번의 곱셈연산에 대해서는 Montgomery 곱셈연산을 실시하지 않더라도 k 단계에서 Montgomery 곱셈연산을 실시하여 modulo q 연산의 결과를 얻을 수 있기 때문에, 일반 곱셈연산을 적용시킬 수 있다. 그리고 나머지 절반인 2^{k-2} 번의 곱셈연산에 대해서만 Montgomery 곱셈연산을 실시하면 된다. $k-2$ 단계에서도 2^{k-1} 번의 곱셈연산이 필요한데, 이 중 절반인 2^{k-2} 번의 곱셈연산에 대해서는 Montgomery 곱셈연산을 이 단계에서 실시하지 않더라도 $k-1$ 단계에서 Montgomery 곱셈연산을 실시하거나, $k-1$ 단계에서 곱셈연산을 실시하지 않는 경우 이는 k 단계에서 Montgomery 곱셈연산을 실시하게 되기 때문에 최종적으로 modulo q 가 된 값을 얻는데 문제가 없다.

일반적으로 l 단계에서 실시하는 2^{k-1} 번의 곱셈연산 중 2^{k-2} 번의 곱셈연산만 Montgomery 곱셈연산을 실시하고, 나머지 곱셈연산은 일반 곱셈연산으로 실시하더라도 l 단계에서 나온 결과가 다음 단계들로 넘어가면서 추가적으로 곱셈연산을 하여야 할 필요가 있고, 그 때, Montgomery 곱셈연산을 하도록 설계를 할 수 있기 때문에 각 단계 별로 지금 하고 있는 곱셈연산의 절반에 대해서는 일반 곱셈연산을 함으로서 NTT의 수행 속도를 높일 수 있다.

2. 2. Radix 2 & Radix 4의 혼합사용을 이용한 최적화

다항식의 NTT는 Radix number 에 따라 다양한 방식으로 계산될 수 있다. 흔히 사용하는 Radix 2 방식은 각 단계 마다 이전 단계에서 계산된 값들을 두 개씩 모아서 계산을 하여 다음 단계로 보내게 된다. 일반적으로 Radix n 방식은 이전 단계에서 계산된 값들을 n 개씩 모아서 계산을 하고 다음 단계로 보내게 된다.

예를 들어, $\mathbb{Z}_q[x]/f(x)$ 에서 $f(x)$ 가 8차인 다항식이라고 할 때, 이 다항식 환에 속하는 원소를 식 (3.20)과 같이,

$$g(x) = g_0 + g_1x + g_2x^2 + g_3x^3 + g_4x^4 + g_5x^5 + g_6x^6 + g_7x^7 \tag{3.20}$$

이 있다고 하자. 위의 다항식을 Radix 2로 NTT를 구할 경우 이전에 나온 식(3.21)과 같이 정리되는데,

$$g(x) = ((g_0 + g_4x^4) + x^2(g_2 + g_6x^4)) + x((g_1 + g_5x^4) + x^3(g_3 + g_7x^4) \tag{3.21}$$

이와 같이 전개를 하고, 단계별로 정리 할 수 있다.

1단계 : Radix 2의 계산인 아래의 식 (3.22)을 계산

$$A_0 = (g_0 + g_4x^4),\ A_1 = (g_1 + g_5x^4),\ A_2 = (g_2 + g_6x^4),\ A_3 = (g_3 + g_7x^4) \tag{3.22}$$

2단계 : Radix 2의 계산인 아래의 식 (3.23)을 계산

$$B_0 = (A_0 + A_2x^2),\ B_1 = (A_1 + A_3x^2) \tag{3.23}$$

3단계 : Radix 2의 계산인 아래의 식 (3.24)를 계산

$$C_0 = (B_0 + B_1x) \tag{3.24}$$

이 다항식을 Radix 2 와 Radix 4를 섞어서 NTT를 구할 경우, 첫 번째 단계에서 Radix 2 방법으로 아래의 식 (3.25)와 같이 계산한다.

$$A_0 = (g_0 + g_4 x^4),\ A_1 = (g_1 + g_5 x^4),\ A_2 = (g_2 + g_6 x^4),\ A_3 = (g_3 + g_7 x^4) \quad (3.25)$$

두 번째 단계에서 Radix 4 방법으로 식 (3.26)과 같이,

$$B_0 = (A_0 + A_1 x + A_2 x^2 + A_3 x^3) \quad (3.26)$$

구하는 방식으로 총 두 단계에 NTT를 구하게 된다.

아래 그림 6의 Proposed NTT와 같이 일반적으로 주어진 다항식에 대해서 NTT를 구할 때 m 개의 단계가 필요한 경우에 있어, 단계 1 부터 단계 m 까지 계산을 할 때, 단계 m 에서는 Montgomery 곱셈연산을 하고, 단계 $m-1$ 에서는 Montgomery 곱셈연산을 하지 않고, 단계 $m-2$에서는 Montgomery 곱셈연산을 하는 방식으로 각 단계 마다 번갈아 가며 Montgomery 곱셈연산을 실시한다. 이 방법은 Radix 2로만 구성된 NTT에서는 효과가 없고, Radix 2 초과의 Radix 로만 구성된 NTT나 Radix 2 와 다른 Radix가 섞여 있는 방식으로 NTT를 구할 때, 기존의 방식에 비해 Montgomery 곱셈연산의 횟수를 줄이는 효과가 있다.

위에서 사용한 $g(x)$ 의 경우 Radix 2 + Radix 4로 구현하였을 때, Radix 4의 계산에서는 Montgomery 곱셈연산을 실시하고, Radix 2 계산에서는 Montgomery 곱셈연산을 실시하지 않는 방법을 사용할 수 있다.

이런 방식을 적용시키기 위해서는 Montgomery 곱셈연산에서 사용하는 R 의 값을 선택함에 있어, $R > q^2$ 이라는 조건을 추가적으로 고려해야한다. 이런 조건의 R 을 선택할 경우, 한 번의 곱셈연산에서는 Montgomery 곱셈연산을 하지 않더라도, 두 번째 곱셈연산에서 Montgomery 곱셈연산을 실시함에 따라 그 결과가 한 번에 modulo q 가 된 값을 얻을 수 있게 된다. 이것을 일반적으로 $R > q^t$ 의 조건으로 확장시키면 총 t 번의 곱셈연산을 할 때, $t-1$ 번의 곱셈연산에 대해서는 Montgomery 곱셈연산을 하지 않고, 마지막 한 번의 곱셈연산에만 Montgomery 곱셈연산을 적용하는 방식으로 확장시킬 수 있다.

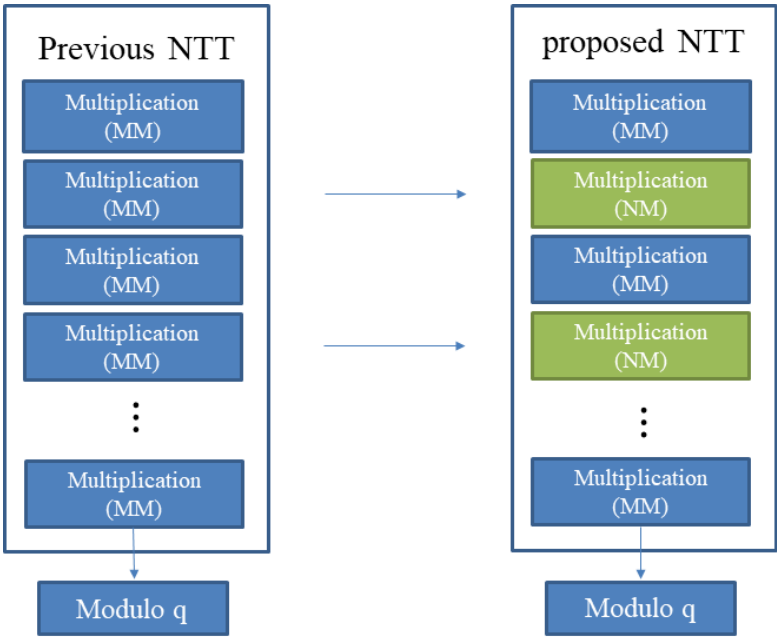

그림 6. Radix n 혼합사용 Lazy 연산 최적화 방안

NTT 단계 1에서 단계 m 까지의 계산 후 최종 결과는 modulo q 연산이 된 결과가 나와야 한다. 그러므로 마지막 단계 m 에서는 Montgomery 곱셈연산을 하여야 한다. 하지만, 그 전 $m-1$ 단계에서는 Montgomery 곱셈연산을 하지 않고, 일반 곱셈연산을 한 후, 그 결과를 다음 단계로 넘기더라도, 다음 단계에서 Montgomery 곱셈연산을 실시하므로 한번에 modulo q 가 된 값을 얻을 수 있다.

정리하자면 NTT의 계산 시 최종적인 결과만 modulo q 연산이 된 것이 필요하고 중간 과정에 나오는 값들은 그렇지 않아도 된다는 점을 고안하여 NTT시 사용되는 Montgomery 곱셈연산의 횟수를 줄여 NTT의 속도를 높일 수 있다. 또한 각 단계마다 Radix 2와 Radix 4를 번갈아가며 사용함에 따라 Radix 2에서 Montgomery 곱셈연산이 되지 않아 modulo q 연산이 되어 있지 않는 값들에 대해 Radix 4 계산 시에 Montgomery 곱셈연산을 실시함으로써 modulo q 연산이 한 번에 되도록 할 수 있다.

IV. 시뮬레이션 결과

A. Rainbow의 효율적 $\mathbb{F}_{16}$ 상의 곱셈연산

앞의 연구를 통해 $\mathbb{F}_{16}$ 상에서의 유한체 연산을 기존 대비 cycle 수 92.1%로 감소시킬 수 있었다. 표 7은 이러한 결과를 나타낸다.

표 7. Chou[62]과 본 논문의 필요 연산과정의 비교

Algorithm	AND	XOR	Ratio
Chou[62]	16	22	100%
Proposed	16	19	92.1%

또한, 곱셈연산은 논리 곱셈연산 AND과 같고 +는 논리합 XOR과 같으며, 두 개의 연산은 모두 carry가 없는 상태로 처리가 가능하다. 따라서 32비트 변수 R_3, R_2, R_1, R_0 에 32개의 4비트 변수를 대입하여 bitslicing을 통한 병렬 연산이 가능하다.

예를 들어 32비트 변수 $R_i = (r_{i,31}, r_{i,30}, r_{i,29}, \ldots, r_{i,1}, r_{i,0})$ 라 가정하면 $\mathbb{F}_4$ 상의 원소 $e_j = (e_{3,j}, e_{2,j}, e_{1,j}, e_{0,j})$ 는 $e_{i,j} = r_{i,j}$ 에 대응되도록 하면, $e_i = (r_{3,i}, r_{2,i}, r_{1,i}, r_{0,i})$ 이다. Chou는 McEliece 암호를 위해 제안된 방법을 응용하여 Interleaving 방식으로 네 개의 변수를 사용해서 비트 단위 데이터 저장 방법을 제안하였고, 이는 32개의 기본 연산이 아닌 28개의 연산으로 구현할 수 있음을 보였다. 본 논문에서 제안하는 방법도 같은 최적화가 그대로 적용될 수 있다[65].

본 절에선 제안한 방법을 실제로 Raspberry PI 3B+ 상에서 구현하여 성능을 평가하였다. 이때 구현에 사용된 알고리즘은 Rainbow 알고리즘 Level I Classic 버전과 Level I Compressed Cyclic 버전이다. 언급한 두 가지 방식 모두 본 논문에서 제안한 $\mathbb{F}_{16}$ 상에서의 연산을 핵심 연산으로 이용한다.

표 8에서 나타난 것처럼 Rainbow 알고리즘은 Level I에 해당하는 알고리즘의 경우 키 생성, 서명, 및 검증을 Classic의 경우 80ms대에서 실행 할 수 있었고 Compressed를 적용한 경우 verification의 속도는 15ms로 비교적 경량 환경에서 매우 효율적 연산이 가능함을 확인할 수 있다.

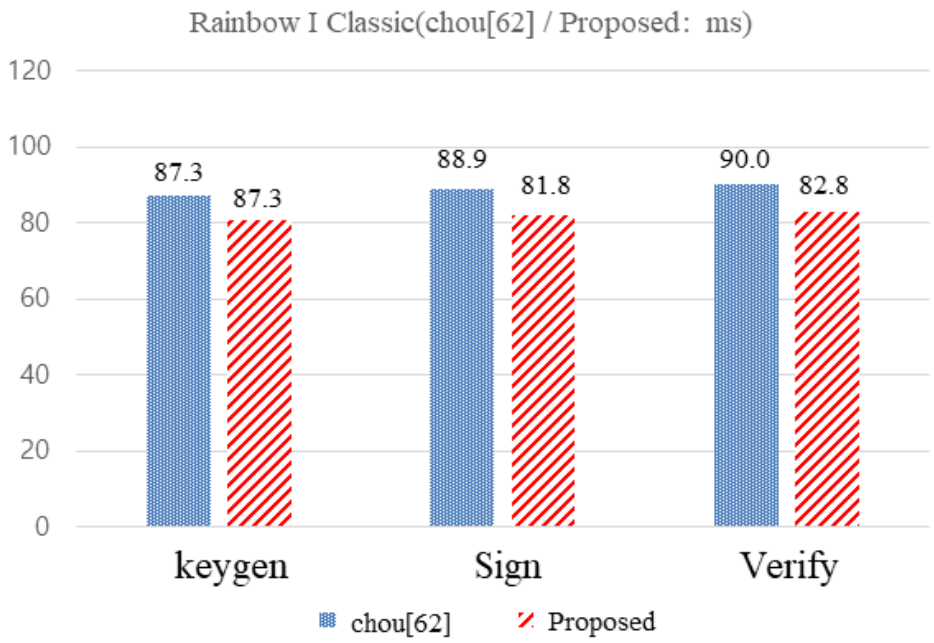

그림 7. Rainbow I Classic의 제안된 연산이 적용되었을 때 실행 속도

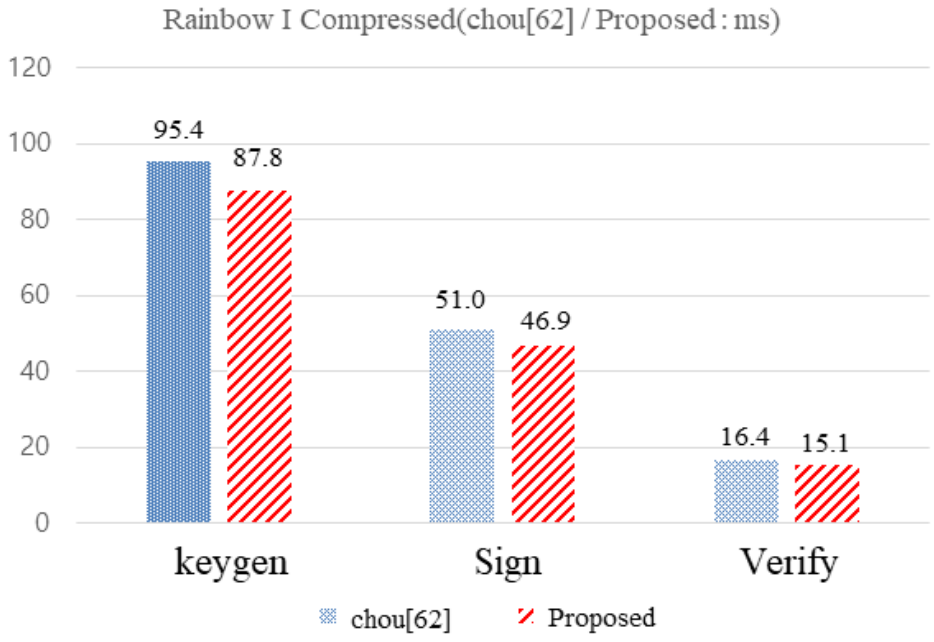

그림 8. Rainbow I Compressed의 제안된 연산이 적용되었을 때 실행 속도

표 8. 라즈베리파이 3 B+ 모델에서 제안이 적용된 Rainbow 성능평가

Algorithm`		Key Gen	Sign	Verify
I Classic	Chou[62]	87.3ms	88.9ms	90.0ms
	Proposed	80.8ms	81.8ms	82.8ms
I Compressed	Chou[62]	95.4ms	51.0ms	16.4ms
	Proposed	87.8ms	46.9ms	15.1ms

B. NTT에서의 효율적 연산을 통한 최적화

첫 번째로 Radix 2의 Lazy 연산을 이용한 최적화 방안을 제시하였다. 제안을 바탕으로 예를 들어 $\mathbb{Z}_q[x]/f(x)$ 에서 $f(x)$ 가 16 차인 다항식이라고 할 때, 이 다항식 환에 속하는 원소 $g(x)$ 가 있다고 하자. 이 다항식을 Radix 2인 NTT를 구할 경우, 아래의 식 (3.27)로 정리할 수 있는데,

$$g(x) = \sum_{i_4=0}^{1} x^{i_4}\left(\sum_{i_3=0}^{1} x^{2i_3}\left(\sum_{i_2=0}^{1} x^{2^2 i_2}\left(\sum_{i_1=0}^{1} x^{2^3 i_1} g_{2^3 i_1 + 2^2 i_2 + 2 i_3 + i_4}\right)\right)\right) \tag{3.27}$$

각 계수에 대해서 각 단계 별로 곱셈연산이 필요한 경우를 다음과 같이 표 9로 나타낼 수 있다.

표 9. 계수에 대한 단계별 곱셈연산

	i_1	i_2	i_3	i_4
g_0	0	0	0	0
g_8	1	0	0	0
g_4	0	1	0	0
g_{12}	1	1	0	0
g_2	0	0	1	0
g_{10}	1	0	1	0
g_6	0	1	1	0

g_{14}	1	**1**	1	0
g_1	0	0	0	1
g_9	**1**	0	0	1
g_5	0	**1**	0	1
g_{13}	1	**1**	0	1
g_3	0	0	**1**	1
g_{11}	1	0	**1**	1
g_7	0	1	**1**	1
g_{15}	**1**	1	**1**	1

주어진 다항식은 4 단계를 거쳐서 NTT를 계산할 수 있는데, 위의 표에서 1로 표시된 부분은 각 단계에서 곱셈연산이 수행되는 위치라고 생각하면 된다.

i_4 열에 표시된 것이 4 단계에 실행되는 곱셈연산을 의미하는데 9번의 1 이라고 적힌 부분은 모두 Montgomery 곱셈연산이 수행되어야 한다. i_3 열에 표시되어 있는 3단계의 곱셈연산 중 강조 표시된 부분은 4 단계에서 Montgomery 곱셈연산을 실시할 것이기 때문에 3 단계에서는 일반 곱셈연산으로 구현 되어도 되는 부분이다.

i_2 열에 표시되어 있는 1 중에서 강조 표시된 부분은 3 단계 또는 4 단계에서 Montgomery 곱셈연산을 실시할 것이기 때문에 2 단계에서는 일반 곱셈연산으로 구현 되어도 되는 부분이다.

i_1 열에 표시되어 있는 1 중에서 강조 표시된 부분은 2 단계 또는 3 단계 또는 4 단계에서 Montgomery 곱셈연산을 실시할 것이기 때문에 1 단계에서는 일반 곱셈연산으로 구현 되어도 되는 부분이다.

이러한 방식으로 제일 마지막 단계를 기준으로 이전 단계로 내려오면서 번갈아 가면서 Montgomery 곱셈연산을 실시할 수 있게 NTT를 구현할 수 있고, 이런 방식으로 구현 시에 k 가 커질수록 아래의 그림 9과 같이 전체 곱셈연산 중 50% 에 가까운 곱셈연산에 대해서 Montgomery 곱셈연산 대신 일반 곱셈연산으로 계산할 수 있는 장점이 있다.

두 번째로 Radix 2 & Radix 4의 혼합사용을 이용한 최적화 방안 제안하였다. 제안을 바탕으로 또 다른 혼합 사용의 예를 들자면 $\mathbb{Z}_q[x]/f(x)$ 에서 $f(x)$가 16차인

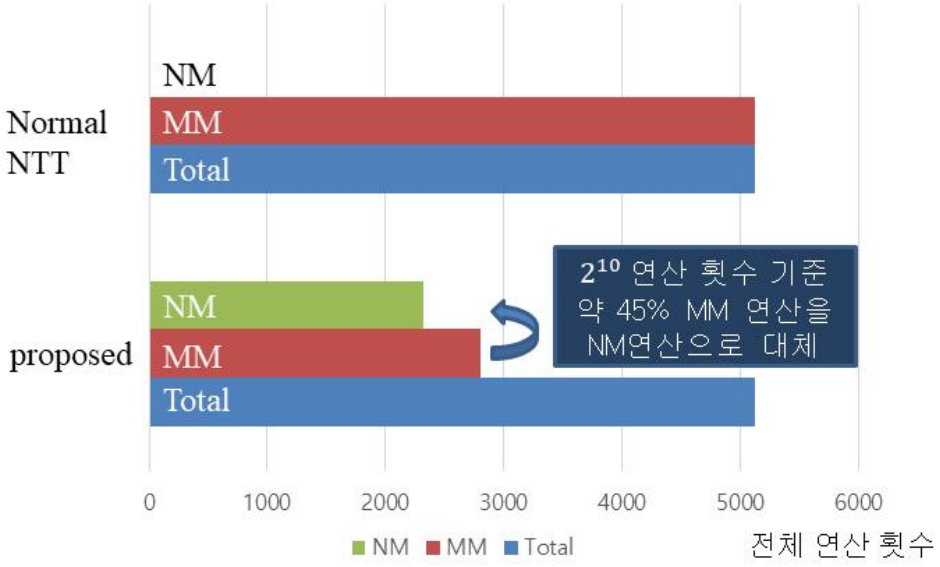

그림 9. 연산횟수 2^{10}을 기준으로 Radix-2 Lazy 연산 방식을 적용한 곱셈연산 결과

다항식이라고 할 때, 이 다항식 환에 속하는 원소 $g(x)$ 는 Radix 2 + Radix 4 + Radix 2의 방식으로 다음의 식 (3.28)와 같이 NTT를 구할 수 있다.

$$g(x) = ((g_0 + g_8 x^8) + x(g_1 + g_9 x^8) + x^2(g_2 + g_{10} x^8) + x^3(g_3 + g_{11} x^8)) \quad (3.28)$$

$$+ x^4((g_4 + g_{12} x^8) + x(g_5 + g_{13} x^8) + x^2(g_6 + g_{14} x^8) + x^3(g_7 + g_{15} x^8))$$

그림 9. 연산횟수 2^{10}을 기준으로 Radix-2 Lazy 연산 방식을 적용한 곱셈연산 결과

1단계 : Radix 2의 계산인 아래의 식 (3.29)를 계산

$$A_0 = (g_0 + g_8 x^8),\ A_1 = (g_1 + g_9 x^8),\ A_2 = (g_2 + g_{10} x^8)$$

$$A_4 = (g_4 + g_{12}x^8),\ A_5 = (g_5 + g_{13}x^8) \qquad (3.29)$$
$$A_6 = (g_6 + g_{14}x^8),\ A_7 = (g_7 + g_{15}x^8)$$

2단계 : Radix 4의 계산인 아래의 식 (3.30)을 계산

$$B_0 = A_0 + A_1x + A_2x^2 + A_3x^3 \qquad (3.30)$$
$$B_1 = A_4 + A_5x + A_6x^2 + A_7x^3$$

3단계 : Radix 2의 계산인 아래의 식 (3.31)을 계산

$$C_0 = B_0 + B_1x^4 \qquad (3.31)$$

이때, 단계 3 에서는 Montgomery 곱셈연산을 사용해야 한다. 그리고 단계 2 에서 Montgomery 곱셈연산을 하고, 단계 1에서 Montgomery 곱셈연산을 하지 않는 방법을 적용할 수 있다. 또는, 단계 2에서 Montgomery 곱셈연산을 하지 않고, 단계 1에서 Montgomery 곱셈연산을 적용할 수도 있다.

동일한 다항식에 대해서 Radix 4 + Radix 4 의 방식으로 NTT를 구현하면 다음과 같은 식 3.32로 나타낼 수 있다.

$$g(x) = (g_0 + g_4x^4 + g_8x^8 + g_{12}x^{12}) + (g_1 + g_5x^4 + g_9x^8 + g_{13}x^{12})x \qquad (3.32)$$
$$+ (g_2 + g_6x^4 + g_{10}x^8 + g_{14}x^{12})x^2 + (g_3 + g_7x^4 + g_{11}x^8 + g_{15}x^{12})x^3$$

1단계 : Radix 4의 계산인 아래의 식 (3.33)을 계산

$$A_0 = (g_0 + g_4x^4 + g_8x^8 + g_{12}x^{12})$$
$$A_1 = (g_1 + g_5x^4 + g_9x^8 + g_{13}x^{12}) \qquad (3.33)$$
$$A_2 = (g_2 + g_6x^4 + g_{10}x^8 + g_{14}x^{12})$$
$$A_3 = (g_3 + g_7x^4 + g_{11}x^8 + g_{15}x^{12})$$

2단계 : Radix 4의 계산인 아래의 식 (3.34)를 계산

$$B_0 = A_0 + A_1 x + A_2 x^2 + A_3 x^3 \tag{3.34}$$

이 경우 단계 2 에서 Montgomery 곱셈연산을 실시하고, 단계 1 에서 일반 곱셈연산을 실시하는 방법으로 NTT에서 사용되는 Montgomery 곱셈연산의 횟수를 줄일 수 있다. 예를 들어, $\mathbb{Z}_q[x]/f(x)$에서 $f(x)$가 1024차인 다항식이라고 할 때, 이 다항식 환에 속하는 원소 $g(x)$는 Radix 2 + Radix 4 + Radix 2 + Radix 4 + Radix 2 + Radix 4 + Radix 2의 방식으로 다음과 같이 NTT를 구할 수 있다.

이 경우 단계 NM(일반 곱셈연산)+MM(Montgomery 곱셈연산)+NM+MM+NM+MM +MM으로 섞어가면서 Montgomery 곱셈연산을 하거나 MM+NM+MM+NM+MM+N M+MM'의 방식으로 섞어서 Montgomery 곱셈연산을 할 수 있다. 또는 Radix 4 + Radix 4 + Radix 4 + Radix 4 + Radix 4 의 방식으로 NTT를 계산하고 M+NM+M+NM+M 방식으로 섞어서 Montgomery 곱셈연산을 할 수도 있다.

V. 결론

오늘날 양자컴퓨터 기술의 급속한 발전으로 인해 양자내성암호 알고리즘의 중요성이 더 커지고 있다. 미국 NIST에서도 차세대 표준 양자내성암호를 선정하기 위한 절차를 지난 5년간 진행하고 있으며, 미래 보안 표준을 위한 양자내성암호 알고리즘에 관한 여러 연구가 활발하게 진행되고 있다. 특히 McNie, pqsigRM, 그리고 Lizard와 같은 국내 연구팀에서 자체 개발한 알고리즘이 NIST PQC 표준화 1라운드에서 평가받기도 하였다. NIST에서 주관하는 양자내성암호 표준화 외에도 중국, 일본, 유럽 등에서도 자체적인 표준화 과정을 진행 중이며, 국내에서도 국가 양자내성암호 공모전을 통해서, 양자내성암호 내재화를 위한 노력을 진행하고 있다. 특히 Layered ROLLO-I, Peregrine, pqsigRM 등 총 16개의 알고리즘이 양자내성암호 공모전에 제안되어 평가중이다.

이에 따라 양자내성암호 알고리즘의 연산 효율성을 향상하기 위한 최적화 작업 및 안전성에 대한 다양한 분석이 진행 중이며, 특히 하드웨어 동작 중에 발생하는 부채널공격을 통한 구현상의 취약점을 분석하고 대응 방법을 마련하는 연구도 다양하게 이루어지고 있다.

본 논문에서는 NIST PQC에 제안되어 3라운드까지 진출했던, 다변수이차방정식 기반의 Rainbow 알고리즘의 효율적 구현을 위한 새로운 방법을 제시한다. 효율적인 유한체 연산 구현을 통해 Chou 등이 제안하였던 Rainbow의 효율적 최적화 연구에서 사용한 22개의 XOR 연산보다 더 적은 19개의 XOR 연산을 사용하여 더 효율적인 곱셈연산이 가능하도록 Cycle 수를 줄여 최적화 시켰으며 유한체 상에서의 역원 연산도 테이블 검색이 아닌 4x4 행렬 역원을 통해 계산할 수 있도록 하였다. $\mathbb{F}_2$ 상에서 연산이 이루어지기 때문에 Gauss 소거법을 통해 고속 구현이 가능한 방법이다. 현재 실제 성능평가는 RaspberryPI 3B+ 환경에서 구현하여 연산 시간을 측정하는 방법으로 이루어졌으며 향후엔 최근 IoT 연산을 위한 경량 환경으로 활용되는 Cortex M4상에서의 실제 구현을 통해 Chou 등이 제안한 방법과 같은 환경에서 직접 비교 연구를 수행할 예정이다.

두 번째로 NIST PQC 표준에 제안된 많은 알고리즘에서 공통으로 광범위하게 사용하는 NTT 알고리즘 및 곱셈 과정에서 필연적으로 사용하는 Modulo reduction 방

법에 lazy 연산을 적용하여 필요한 Montgomery 연산의 수를 50% 정도로 줄일 수 있는 두 가지 방법을 제안한다. 제안하는 방법을 통해 Montgomery reduction의 수를 줄임으로써 일반 곱셈만으로 연산을 수행하는 단계와 reduction을 수행하는 단계로 나누어 구현할 수 있으며, 이를 통해 연산 시간을 단축할 수 있었다. 차후 더욱 효율적 NTT 연산을 위해 여러 연산에 관한 연구를 수행할 예정이다.

참 고 문 헌

[1] F. Arute, K. Arya, R. Babbush et al., “Quantum supremacy using a programmable superconducting processor,” *Nature*, vol. 574, pp. 505-510, Oct. 2019.

[2] R. P. Feynman, “Simulating Physics with Computers,” *International Journal of Theoretical Physics*, vol. 21, pp. 467-488, Jun. 1982.

[3] P. W. Shor, “Polynomial-time algorithms for prime factorization anddiscrete logarithms on a quantum computer,” *SIAM journal on Computing*, vol. 26, Oct. 1997, pp. 1484-1509..

[4] G. Alagic, et al., “Status report on the Third round of the NIST postquantum cryptography standardization process,” US Department of Commerce, NIST, 2022.

[5] 김소선, “효율적인 유한체 역원 알고리즘에 연구,” 고려대학교 대학원 석사학위 논문, 서울, 대학민국, 2005년.

[6] J. Ding, M. Chen, M. Kannwischer, J. Patarin, A. Petzoldt, D. Schmidt, and B. Yang. “Rainbow. Submission to the NIST Post-Quantum Cryptography Standardization Project,” US Department of Commerce, NIST, 2020.

[7] A. Ferozpuri and K. Gaj, "High-speed FPGA Implementation of the NIST Round 1 Rainbow Signature Scheme," in *International Conference on ReConFigurable Computing and FPGAs (ReConFig)*, 2018, pp. 1-8.

[8] D. Soni and R. Karri, “Efficient hardware implementation of pqc primitives and pqc algorithms using high-level synthesis,” in *IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 2021, pp. 296-301.

[9] W. Beullens, “Improved Cryptoanaylsis of UOV and Rainbow,” in *Annual International Conference on the Theory and Applications of Cryptographic Techniques EUROCRYPT 2021*, 2021, pp. 348-373.

[10] N. Shuhei, I. Yasuhiko, W. Yacheng, J. Ding and T. Tsuyoshi, “New complexity estimation on the Rainbow-Band-Separation attack,” in *Theoretical Computer Science*, vol. 896, Jul. 2021, pp. 1-18.

[11] R. Döring and M. Geitz, “Post-Quantum Cryptography in Use: Empirical Analysis

of the TLS Handshake Performance," in *NOMS 2022-2022 IEEE/IFIP Network Operations and Management Symposium*, 2022, pp. 1-5.

[12] 김광식, 김영식, "NIST PQC Rainbow의 효율적 유한체 연산 구현," 한국 정보보호학회, 정보보호학회 논문지, vol. 31, no. 3, pp. 527-532, 2021년 6월.

[13] P. Kurariya, A. Bhargava, S. Sailada, N. Subramanian, J. Bodhankar and A. Kumar, "Experimentation on Usage of PQC Algorithms for eSign," in *2022 IEEE International Conference on Public Key Infrastructure and its Applications (PKIA)*, 2022, pp. 1-6.

[14] R. Kuang, M. Perepechaenko, R. Toth and M. Barbeau, "Benchmark Performance of a New Quantum-Safe Multivariate Polynomial Digital Signature Algorithm," in *2022 IEEE International Conference on Quantum Computing and Engineering (QCE)*, 2022, pp. 454-464.

[15] A. Petzoldt, "Efficient Key Generation for Rainbow," in *International Conference on Post-Quantum Cryptography PQCrypto 2020*, 2020, pp. 92-107.

[16] R. Kuang, M. Perepechaenko and M. Barbeau, "A new quantum-safe multivariate polynomial public key digital signature algorithm," *Nature Scientific Reports*, vol. 12, no. 1, pp. 1-21, Aug. 2022.

[17] Z. Peng and S. Tang, "Circulant Rainbow: A New Rainbow Variant With Shorter Private Key and Faster Signature Generation," IEEE Access, 2017, vol. 5, pp. 11877-11886.

[18] M. Raavi, P. Chandramouli, S. Wuthier, X. Zhou and S. Y. Chang, "Performance Characterization of Post-Quantum Digital Certificates," in *2021 International Conference on Computer Communications and Networks (ICCCN)*, 2021, pp. 1-9.

[19] D. Micciancio and O. Regev, "Lattice-based Cryptography," Post-quantum cryptography Springer, 2009, pp. 147-191.

[20] M. Ajtai, "Generating hard instances of lattice problem," in *Proc. The 28th Annual ACM Symposium on Theory of Computing*, 1996, pp. 99-108.

[21] 박애선, 원유승, 한동국, "Ring-LWE 기반 공개키 암호시스템의 선택 암호문 단순전력분석 공격 대응법," 한국정보보호학회, 정보보호학회 논문지, vol. 27, no. 5, pp. 1001-1011, 2017년 10월.

[22] D. P. Chi, J. W. Choi, J. S. Kim and T. W. Kim, “Lattice Based Cryptography for Beginners,” Dec. 2015, [online].

[23] J. Zhang, J. Huang, Z. Liu and S. S. Roy, “Time-Memory Trade-Offs for Saber+ on Memory-Constrained RISC-V Platform,” *IEEE Transactions on Computers*, vol. 71, no. 11, pp. 2996-3007, Nov. 2022.

[24] J. Quisquater and D. Samyde, “Electromagnetic Analysis (EMA): Measures and Countermeasures for Smart Cards,” in *International Conference on Research in Smart Cards*, 2001, pp. 200-210.

[25] O. Regev, “New lattice-based cryptographic constructions,” *journal of ACM*, vol. 51, no. 6, pp. 296-301, Nov. 2004.

[26] A. Li et al., “A Flexible Instruction-based Post-quantum Cryptographic Processor with Modulus Reconfigurable Arithmetic Unit for Module LWR&E,” in *2022 IEEE Asian Solid-State Circuits Conference (A-SSCC)*, 2022, pp. 1-3.

[27] 박애선, “곱 연산 기반 Post-Quantum 암호의 부채널 분석 및 대응 기법에 관한 연구,” 국민대학교 대학원 박사학위 논문, 서울, 대한민국, 2019년.

[28] V. Lyubashevsky, C. Peikert, and O.Regev, “On Ideal Lattices and Learning with Errors over Rings,” in *Annual International Conference on the Theory and Applications of Cryptographic Techniques EUROCRYPT 2010*, 2010, pp. 1-23.

[29] M Rosca, D Stehlé and A Wallet, “On the ring-LWE and polynomial-LWE problems,” in *Annual International Conference on the Theory and Applications of Cryptographic Techniques EUROCRYPT 2018*, 2018, pp. 146-173.

[30] C. Saliba, L. Luzzi and C. Ling, “A reconciliation approach to key generation based on Module-LWE,” in *2021 IEEE International Symposium on Information Theory (ISIT)*, 2021, pp. 1636-1641.

[31] R. Román, R. Arjona, P. López-González and I. Baturone, “A Quantum-Resistant Face Template Protection Scheme using Kyber and Saber Public Key Encryption Algorithms,” in *2022 International Conference of the Biometrics Special Interest Group (BIOSIG)*, 2022, pp. 1-5.

[32] C. P. Mejía and J. V. Medina, “High-Throughput Ring-LWE Crypto-processors,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 8,

pp. 2332-2345, Aug. 2017.

[33] M. Bisheh-Niasar, R. Azarderakhsh and M. Mozaffari-Kermani, “High-Speed NTT-based Polynomial Multiplication Accelerator for Post-Quantum Cryptography,” in *2021 IEEE 28th Symposium on Computer Arithmetic (ARITH)*, 2021, pp. 94-101.

[34] N. Zhang, B. Yang, C. Chen, S. Yin, S. Wei and L. Liu, “Highly Efficient Architecture of NewHope-NIST on FPGA using Low-Complexity NTT/INTT,” *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2020, no. 2, pp. 49-72, Mar. 2020.

[35] 신예린, “격자 기반 양자 내성 암호를 위한 고효율, 저 면적 부분 병렬 NTT 프로세서 구조,” 충남대학교 대학원 석사학위 논문, 대전, 대한민국, 2022년

[36] 김성재, “양자내성암호를 위한 고성능 Crystal-Kyber 암호 아키텍처,” 인하대학교 대학원 석사학위논문, 인천, 대한민국, 2022년.

[37] C. M. Chung, V. Hwang, M. J. Kannwischer, G. Seiler, C. J. Shih and B. Y. Yang, “NTT Multiplication for NTT-unfriendly Rings,” *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2021, no. 2, pp. 159-188, Feb. 2021.

[38] 김현준, “효율적인 모듈러 곱셈연산 알고리즘에 대한 연구,” 고려대학교 정보보호대학원 석사학위 논문, 서울, 대한민국, 2006년.

[39] J. W. Bos, P. L. Montgomery, D. Shumow and G. M. Zaverucha, “Montgomery Multiplication Using Vector Instructions,” in *International Conference on Selected Areas in Cryptography*, 2014, pp. 471-489.

[40] M. Mukhopadhyay and P. Sarkar, “Combining Montgomery Multiplication with Tag Tracing for the Pollard Rho Algorithm in Prime Order Fields,” in *international Conference on Security, Privacy, and Applied Cryptography Engineering*, 2022, pp. 138-146.

[41] B. N. Vangapandu and A. Chalil, “FPGA Implementation of High-Performance Montgomery Modular Multiplication with Adaptive Hold Logic,” in *2022 6th International Conference on Computing Methodologies and Communication (ICCMC)*, 2022, pp. 506-511.

[42] D. Greconici, “KYBER on RISC-V,” Master’s Thesis Computing Science in Cyber

Security, Rabound University, Gelderland, Netherlands, 2020.

[43] E. Alkim, et al., "Cortex-M4 optimizations for {R, M} LWE schemes," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2020, no. 3, pp. 336-357, jun. 2020.

[44] R. Avan, et al., "Crystals-Kyber: algorithm specification and supporting documentation (version 3.0)," Submission to the NIST Post Quantum Cryptography Standardization Project, NIST, 2020.

[45] T. Hasija, K. R. Ramkumar, A. Kaur, S. Mittal and B. Singh, "A Survey on NIST Selected Third Round Candidates for Post Quantum Cryptography," in *2022 7th International Conference on Communication and Electronics Systems (ICCES)*, 2022, pp. 737-743.

[46] O. Regev. "On lattices, learning with errors, random linear codes, and cryptography," *Journal of ACM*, vol. 56, no. 6, pp. 84–93, Sep. 2009.

[47] E. Fujisaki and T. Okamoto. "Secure integration of asymmetric and symmetric encryption schemes," *Journal of Crypto*, vol. 26, no. 1, pp. 80-101, Dec. 2013.

[48] J. Hoffstein, J. Pipher, and J. H. Silverman. "NTRU: a ring-based public key cryptosystem," in *Algorithmic number theory*, 1998, pp. 267-288.

[49] M. Alekhnovich, "More on average case vs approximation complexity," in *Proc. 44th Annual IEEE Symposium on Foundations of Computer Science*, 2003, pp. 298-307.

[50] V. Lyubashevsky, A. Palacio, and G. Segev, "Public-key cryptographic primitives probably assecure as subset sum," in *Theory of Cryptography Conference*, 2010, pp. 382-400.

[51] B. Applebaum, D. Cash, C. Peikert, and A. Sahai, "Fast cryptographic primitives and circular secure encryption based on hard learning problems," in *Annual International Cryptology Conference CRYPTO 2009*, 2009, pp. 595-618.

[52] E. Alkim, L. Ducas, T. Pöppelmann, and P. Schwabe, "Post-quantum key exchange a new hope," in *USENIX Security Symposium*, 2016, pp. 327–343.

[53] A. Banerjee, C. Peikert, and A. Rosen, "Pseudorandom functions and lattices," in *Advances in Cryptology EUROCRYPT 2012*, 2012, pp. 719–737.

[54] T. C. Peikert, “Public-key cryptosystems from the worst-case shortest vector problem: extended abstract,” in *Proc. ACM symposium on Theory of computing*, 2009, pp. 333−342.

[55] T. Pöppelmann and T. Güneysu. “Towards practical lattice-based public-key encryption on reconfigurable hardware,” in *International Conference on Selected Areas in Cryptography*, 2013, pp. 68−85.

[56] R. C. Policarpo, A. S. Nery and R. D. O. Albuquerque, “Quantum-resistant Cryptography in FPGA,” in *2022 Workshop on Communication Networks and Power Systems (WCNPS)*, 2022, pp. 1-5.

[57] A. F. Dursun, K. Seyhan and S. Akleylek, “Mobil Cihazların Kuantum Sonrası Güvenliği İçin Uyarlanmış KEM Uygulamaları: Adapted KEM Applications for Post-Quantum Security of Mobile Devices,” in *2022 15th International Conference on Information Security and Cryptography (ISCTURKEY)*, 2022, pp. 31-37.

[58] E. Dubrova, K. Ngo and J. Gartner, “Breaking a Fifth-Order Masked Implementation of CRYSTALS-Kyber by Copy-Paste,” unpublished.

[59] R. Kuang, M. Perepechaenko and M. Barbeau, “A new post-quantum multivariate polynomial public key encapsulation algorithm,” *Quantum Inf Process*, vol. 21, no. 360, pp. 1-25, Oct. 2022.

[60] B. A. Dang and K. Krishnamoorthy, “Confidence intervals, prediction intervals and tolerance intervals for negative binomial distributions,” *Statistical Papers*, vol. 63, no. 6, pp. 795-820, Jun. 2022.

[61] J. M. Riera, “Performance Analysis of Rainbow on ARM Cortex-M4,” Bachelor’s Thesis, Technische Universität München, München, Germany, 2019.

[62] T. Chou, M. Kannwischer, and B. Yang, “Rainbow on Cortex-M4,” *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2021, no. 4, pp. 650-675, Aug. 2021.

[63] S. Li and Z. Gu, “Lazy Reduction and Multi-Precision Division Based on Modular Reductions,” in *2018 IEEE Asia Pacific Conference on Circuits and Systems (APCCAS)*, 2018, pp. 407-410.

[64] M. Scott, “A note on the implementation of the Number Theoretic Transform,” in

IMA International Conference on Cryptography and Coding, 2017, pp. 247-258.

[65] T. Chou. "Mcbits revisited," in *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2017, pp. 213-231.