



저작자표시-비영리-변경금지 2.0 대한민국

이용자는 아래의 조건을 따르는 경우에 한하여 자유롭게

- 이 저작물을 복제, 배포, 전송, 전시, 공연 및 방송할 수 있습니다.

다음과 같은 조건을 따라야 합니다:



저작자표시. 귀하는 원저작자를 표시하여야 합니다.



비영리. 귀하는 이 저작물을 영리 목적으로 이용할 수 없습니다.



변경금지. 귀하는 이 저작물을 개작, 변형 또는 가공할 수 없습니다.

- 귀하는, 이 저작물의 재이용이나 배포의 경우, 이 저작물에 적용된 이용허락조건을 명확하게 나타내어야 합니다.
- 저작권자로부터 별도의 허가를 받으면 이러한 조건들은 적용되지 않습니다.

저작권법에 따른 이용자의 권리는 위의 내용에 의하여 영향을 받지 않습니다.

이것은 [이용허락규약\(Legal Code\)](#)을 이해하기 쉽게 요약한 것입니다.

[Disclaimer](#)

석사학위논문

8-bit AVR 마이크로 컨트롤러에서의
양자내성암호 Crystals-Kyber 최적화
구현 연구

Efficient Implementation of
Post-Quantum Cryptography Crystals-Kyber on 8-bit
AVR microcontroller

국민대학교 일반대학원
금융정보보안학과 정보융합보안전공

김 영 범

2022

8-bit AVR 마이크로 컨트롤러에서의
양자내성암호 Crystals-Kyber 최적화
구현 연구

Efficient Implementation of
Post-Quantum Cryptography Crystals-Kyber on 8-bit
AVR microcontroller

지도교수 서 석 충

이 논문을 석사학위 청구논문으로 제출함

2022년 12월

국민대학교 일반대학원

금융정보보안학과 정보융합보안전공

김 영 범

2022



김영범의
석사학위 청구논문을 인준함

2022년 12월

심사위원장 유일선 ①

심사위원 김종성 ①

심사위원 서석중 ①

국민대학교 일반대학원

Contents

List of Figure	iv
List of Algorithm	v
List of Table	vi
Abstract (Korean)	vii
제 1 장. Introduction	1
제 1 절. Context	1
제 2 절. Motivation	2
제 3 절. Contribution	4
제 4 절. Outline	4
제 2 장. Preliminaries	5
제 1 절. Post-Quantum Cryptography	5
1.1. Learning with Errors	5
1.2. Ring-LWE	6
1.3. Module-LWE	7
제 2 절. Sampler for Cryptography	8
2.1. Rejection Sampler	11
2.2. CDT Sampler	12
2.3. Binomial Sampler	14
2.4. Comparison of Sampler	15
제 3 절. Number Theoretic Transform	17

제 3 장. Crystals-Kyber	21
제 1 절. Changelog of Kyber during NIST PQC Competition	21
1.1. Changelog	21
제 2 절. Specification for Crystals-Kyber	23
2.1. Notation and Definition	23
2.2. Parameter	24
2.3. Sampling of Crystals-Kyber	25
2.4. Encoding and Decoding	27
2.5. Kyber.CPAPKE	29
2.6. Kyber.CCAKEM	34
2.7. Analysis source code	36
제 4 장. Kyber on 8-bit AVR	40
제 1 절. 8-bit AVR ISA	40
제 2 절. NTT implementation on 8-bit AVR microcontroller	42
2.1. Barrett Reduction	43
2.2. Montgomery Reduction	46
2.3. Butterflies	48
제 3 절. Stack-friendly Kyber Implementation	50
3.1. Streaming coefficient strategy	50
제 4 절. Speed-friendly Kyber Implementation	51
4.1. Asymmetric Multiplication	51

4.2. Better Accumulation	51
4.3. Pre-hashed public key	52
4.4. keccak implementation with XKCP library	52
제 5 장. Evaluation	53
제 1 절. Setup 8-bit AVR board	53
제 2 절. Result of Implementation	54
제 6 장. Conclusion	56
Abstract (English)	57
Reference	58



List of Figure

[그림 1] Crystlas-Kyber 소스코드 파일 계층도	39
[그림 2] 8-bit AVR 환경에서의 16-bit 기반의 곱셈구현 코드	42
[그림 3] 8-bit AVR 환경에서의 Barrett Reduction 구현 코드	43
[그림 4] 8-bit AVR 환경에서의 Montgomery Reduction 구현 코드	46
[그림 5] NTT 구현을 위한 AVR 환경에서의 레지스터 스케줄링 ...	49
[그림 6] 8-bit AVR 환경에서 CT 기반 NTT 변환 코드	49
[그림 7] Streaming public matrix A 의 구현 방안	50



List of Algorithm

[Algorithm 1]	CDT Sampler 알고리즘	12
[Algorithm 2]	CT 기반의 NTT 구현 기법	19
[Algorithm 3]	GS 기반의 NTT 구현 기법	20
[Algorithm 4]	Crystals-Kyber의 중심이항분포 알고리즘	26
[Algorithm 5]	Crystals-Kyber의 균등분포 알고리즘	28
[Algorithm 6]	Crystals-Kyber의 CPAPKE.KeyGen 알고리즘	29
[Algorithm 7]	Crystals-Kyber의 CPAPKE.Enc 알고리즘	30
[Algorithm 8]	Crystals-Kyber의 CPAPKE.Dec 알고리즘	34
[Algorithm 9]	Crystals-Kyber의 CCAKEM.KeyGen 알고리즘	35
[Algorithm 10]	Crystals-Kyber의 CCAKEM.Enc 알고리즘	35
[Algorithm 11]	Crystals-Kyber의 CCAKEM.Dec 알고리즘	36
[Algorithm 12]	Signed Barrett Reduction	43
[Algorithm 13]	Signed Montgomery Reduction	46



List of Table

[표 1] Crystlas-Kyber의 파라미터	24
[표 2] PQclean Crystlas-Kyber 프로젝트 구성	37
[표 3] PQclean Crystlas-Kyber ref 소스파일 구성	38
[표 4] 8-bit AVR 환경에서의 주요 명령어	41
[표 5] Crystals-Kyber 보안레벨별 매개변수 크기	53
[표 6] 8-bit AVR 환경에서 NTT/FFT 구현결과 비교	54
[표 7] 최종 결과물에 적용한 방법론 정리	55
[표 8] 8-bit AVR 환경에서 Crystals-Kyber의 구현 결과 비교	55



국문 요약

8-bit AVR 마이크로 컨트롤러에서의 양자내성암호 Crystals-Kyber 최적화 구현 연구

김영범

국민대학교 금융정보보안학과

양자내성암호는 타원곡선기반 암호 스킴과 비교하여 일반적으로 낮은 성능과 높은 메모리를 요구한다. 따라서, 최적화 구현연구는 PQM4를 시작으로 32-bit 및 64-bit 기반의 ARM 환경에서 활발하게 이루어졌다. 8-bit AVR 환경은 임베디드 기기들 중 성능과 메모리가 가장 제한된 장치로 아직 표준화 대상 양자내성암호의 구현연구가 진행되지 않았다. 향후 모든 기기에서의 양자보안을 달성하기 위해서는 AVR 환경에서도 양자내성암호의 탑재가 필요하다. 따라서, 본 논문에서는 8-bit AVR 환경에서 표준화 대상 양자내성암호 중 공개키 암호 및 키 설정 알고리즘인 Crystals-Kyber에 대한 최초의 최적화 구현 결과를 제시하고 표준화 대상 양자내성암호의 성능평가에 기여한다.

- 주제어(키워드, 핵심어)

양자내성암호, Crystals-Kyber, 8-bit AVR, 구현 최적화 연구

제 1 장 Introduction

제 1 절 Context

IT 산업에서 정보보안은 가장 작은 단위인 개인정보보호에서부터 크게는 국가의 안전까지 컴퓨터 과학에서 사회의 안전과 질서를 유지하고 있다. 그중 암호학은 정보보안의 기반이 되는 학문으로 기밀성 (Confidentiality), 무결성 (Integrity), 가용성(Availability)의 보안 목표를 유지하기 위해 대칭 암호화 기법과 비대칭 암호화 기법을 사용한다. 대칭 암호화 기법은 가장 기본적인 암호화 방법으로 Alice와 Bob이 통신할 때 서로가 나누어 가진 키를 통해 데이터를 대칭 암호화를 통해 서로에게 전달한다. 1,900년대 이후 통신 가용 범위가 증가함에 따라, 통신 자간 키를 설정하는 방법에 관한 연구들이 진행되었다. 비대칭 암호화 기법은 수학적 난제를 통해 Alice와 Bob이 서로 다른 키를 사용하여 데이터를 암호화하는 방식으로 일반적으로 대칭 암호화보다 낮은 성능을 보이기 때문에 주로 대칭 암호화에 사용할 키를 생성하는 목적으로 주로 사용된다.

비대칭 암호화 시스템은 수학적 난제인 이산 로그 문제 (Discrete Logarithm Problem, DLP)와 소인수 분해 (Prime Factorization, PF)를 기반으로 이루어져 있다.

그러나, 구글과 IBM이 주도하여 양자 컴퓨터가 활발하게 연구됨에 따라, 양자 컴퓨팅 환경은 현재 사용하고 있는 대칭 및 비대칭 암호화 시스템에 대한 보안성을 위협하고 있다. 양자컴퓨팅 환경에서는 1994년 Shor가 제안한 소인수 분해 알고리즘과^[1], 1996년에 Grover가 제안한 탐색 알고리즘을 적용할 수 있기

때문이다[2]. 이러한 양자 컴퓨팅의 부상은 기존 컴퓨터와 양자컴퓨터 모두의 공격에 대응할 수 있는 체계로 마이그레이션해야 함을 시사한다. 해당 체계에서 사용하는 암호화를 양자내성암호 (Post-Quantum Cryptography, PQC)라 하며 최근 20년 동안 암호학자들의 관심을 끌고 있다.

이에 따라, 미국 국립표준 기술 연구소 (National Institute for Standards and Technology, NIST)는 2016년에 PQC에 대한 공모전을 개최하였다. NIST의 목적은 기존의 DLP 및 PF 기반의 암호화 체계가 아닌 새로운 수학적 난제 기반의 스킴의 생성이다. 따라서, 기존의 Advanced Encryption Standard (AES), Secure Hash Algorithm 3 (SHA-3) 공모전과 달리 단일 우승자를 가리지 않고 여러 대책을 고려한다고 발표하였다. NIST PQC 공모전은 2가지 분야 Public Key Encryption/Key Encapsulation Mechanism (PKE/KEM)과 Digital Signature Algorithm (DSA)로 진행되며 최근 2022년 7월에 표준화 대상 PQC-PKE/KEM 1종 (Crystals-Kyber), PQC-DSA 3종 (Crystals-Dilithium, Falcon, SPHINCS+)이 선정되었다. 표준화 대상 알고리즘 중 4종 중 3종은 격자기반 알고리즘이고, 1종은 해시함수 기반의 알고리즘이다.

제 2 절 Motivation

NIST PQC 공모전이 2016년도에 개최된 이후 제안된 알고리즘들에 대한 다양한 최적화 연구가 진행되었고 공모전의 성능평가에 기여했다. NIST의 성능평가 기준의 첫 번째는 Software (SW)에서의 속도 및 메모리 사용량 및 Hardware (HW)에서의 면적 및 전력 사용량이다. NIST가 지정한 SW 성능평가 기기는 ARM-Cortex-M4, CPU, AVX2이고, 하드웨어의 경우는 Artix-7이

다. 두 번째 평가사항은 부채널 공격에 대한 대응책이다. NIST는 모든 제출자에게 상수시간 구현을 요구하였으며 제출된 알고리즘에 대해 부채널 분석 관점의 평가를 진행하였다. 마지막으로 PQC의 적용 가능성이다. 다양한 상용 프로토콜 (TLS/SSL, DNSSEC, IPSEC 등)에 대한 마이그레이션의 용이성의 여부는 향후 암호화 체계에 있어 중요한 이슈이기 때문이다.

NIST PQC 공모전 개최 초기에는 주로 PQC의 속도에 대한 관심이 높았으나, 다양한 최적화 연구를 통해 타원곡선 기반의 기존 암호화 체계에 비해 더 좋은 성능을 가지게 되는 알고리즘들이 생겨남에 따라, 현재는 PQC의 메모리 사용량에 관한 최적화 연구들이 주로 진행되고 있다[3]. 특히, 코어의 성능과 메모리가 제한된 환경인 임베디드 환경에서의 구현 방법론에 대한 최적화 연구가 활발하게 진행되었다[4]. 최적화 연구의 대상 플랫폼은 주로 NIST의 성능평가 대상 기기인 ARM-Cortex M4이다. ARM-Cortex M4 환경에서 PQC 공모전에 제출된 알고리즘들을 벤치 마킹한 연구[5][6]가 존재하며 해당 라이브러리에 최신 연구들이 계속해서 업데이트 되는 중이다.

향후 모든 응용체계에서 PQC의 적용은 필수적이기 때문에, 모든 기기에서의 마이그레이션은 반드시 고려해야 한다. 그러나, 가장 성능과 메모리가 제한된 코어인 8-bit AVR 환경에서의 PQC 구현연구는 아직 전무하다. 8-bit AVR 환경은 현재 Wireless Sensor Network (WSN) 환경에서 전 세계적으로 가장 많이 사용되는 임베디드 장치로 현재 다양한 산업화 분야에서 사용 중이다[7]. 따라서, 본 논문에서는 표준화 대상 PQC-KEM 알고리즘 Crystals-Kyber를 최적화하려고 한다.

제 3 절 Contribution

본 논문의 기여는 다음과 같이 요약할 수 있다. 첫째, 8-bit AVR 환경에서의 최초의 표준화 대상 양자내성암호의 구현을 달성하고 Crystlas-Kyber의 AVR 환경에 대한 성능평가에 기여한다. 둘째, ARM Cortex-M/A 시리즈에서 제안되었던 모든 Crystlas-Kyber의 최적화 방안을 AVR 환경에 도입하여 적용 가능성을 증명한다. 셋째 AVR 환경에서 Crystlas-Kyber의 2가지 버전을 제안함으로써, 다양한 절충안을 고려하였다. 넷째, 8-bit AVR 환경에서 NTT 구현 방법론을 제안하고 AVR 환경에서 가장 빠른 NTT 구현을 달성한다. 마지막으로 본 논문의 제안 기법은 Constant-time 구현을 달성한다. 본 연구의 구현물은 오픈 소스 형태로 제공된다¹⁾.



제 4 절 Outline

본 논문의 구성은 다음과 같다. 2장에서 PQC의 개요와 PQC에서의 샘플링 기법, NTT의 정의와 구현기법에 관해 설명하고, 3장에서 본 논문의 대상 알고리즘인 Crystals-Kyber를 분석한다. 4장에서는 8-bit AVR 환경에서 Crystals-Kyber의 NTT 구현기법, 메모리 최적화 구현기법, 속도 최적화 구현기법에 대해 제안한다. 5장에서는 제안 최적화 기법들의 성능을 평가하고 마지막으로 6장에서 본 논문을 끝맺는다.

1) <https://github.com/Youngbeom94/Crystals-Kyber-on-AVR>

제 2 장 Preliminaries

제 1 절 Post-Quantum Cryptography

1.1 Learning with Errors

Learning With Errors (LWE) 난제는 2009년 Regev에 의해 제안되었으며 NP-Hard 난제인 Shortest Vector Problem (SVP)으로의 reduction에 근거한 보안성을 가진다. 즉, 상위 수준에서 격자기반 암호화에 속한다[8]. LWE 문제는 비밀(에러, 노이즈) 벡터가 추가된 Z_q 의 선형 시스템을 해결하는 것으로 볼 수 있다. Z_q 에 에러가 없는 경우 일반적으로 가우시안 소거법을 이용하여 쉽게 해결할 수 있지만, 에러가 있는 경우는 해결할 수 있다. LWE는 다음과 같이 정의될 수 있다.

■ **Definition of LWE** : 벡터 $\mathbf{a}$ 와 비밀 벡터 $\mathbf{s}$ 가 Z_q^n 에서 균등하게 무작위로 선택되었다고 가정하고, 에러 e 가 Z_q 에서 표준편차가 σ 인 가우스 분포에서 샘플링되었다고 가정한다.

■ **Search LWE** : $(\mathbf{a}, \langle \mathbf{a}, \mathbf{s} \rangle + e) \in Z_q^n \times Z_q$ 형태의 m 개의 샘플이 존재할 때 비밀벡터 $\mathbf{s}$ 를 복구한다.

■ **Decisional LWE** : $(\mathbf{a}, b) \in Z_q^n \times Z_q$ 형태의 m 개의 샘플이 주어지면 해당 샘플들이 $(\mathbf{a}, \langle \mathbf{a}, \mathbf{s} \rangle + e) \in Z_q^n \times Z_q$ 인지 또는 $Z_q^n \times Z_q$ 에서 균등하게 무작위

로 샘플되었는지 구별한다.

1.2 Ring-LWE

Ring-LWE 문제는 LWE 문제와 같지만 각 벡터를 Z_q^n 이 아닌 n 미만의 차수를 가지는 다항식 환 R_q 에서 샘플링한다[9]. Ring-LWE의 장점으로서는 특정 파라미터의 선택에 의해 Fast Fourier Transform (FFT) 변환을 적용할 수 있다[10]. Ring-LWE는 하나의 벡터 $\mathbf{a}_1 = (x_1, x_2, \dots, x_n) \in Z_q^n$ 만 샘플링하여 LWE 시스템에 일부 구조를 제공할 수 있다[8]. 나머지 $n-1$ 개의 벡터는 $\mathbf{a}_i = (x_i, \dots, x_n, -x_1, \dots, -x_{i-1})$ 규칙에 따라 얻을 수 있으므로 $\mathbf{a}_1$ 을 통해 Ring-LWE의 전체구조를 나타낼 수 있다. 따라서, 이 표현은 환 R_q 로 변환될 수 있다. Ring-LWE은 다음과 같이 정의될 수 있다.

■ **Definition of Ring-LWE** : 벡터 $\mathbf{a} \in R_q$ 와 비밀 벡터 $\mathbf{s} \in R_q$ 의 원소들이 Z_q 에서 균등하게 무작위로 선택되었다고 가정하고, 에러 e 가 R_q 에서 표준편차가 σ 인 가우스 분포에서 샘플링되었다고 가정한다.

■ **Search Ring-LWE** : $(\mathbf{a}, \mathbf{a} \cdot \mathbf{s} + e) \in R_q \times R_q$ 형태의 m 개의 샘플이 존재할 때 비밀벡터 $\mathbf{s}$ 를 복구한다.

■ **Decisional Ring-LWE** : $(\mathbf{a}, b) \in R_q \times R_q$ 형태의 m 개의 샘플이 주어지면 해당 샘플들이 $(\mathbf{a}, \mathbf{a} \cdot \mathbf{s} + e) \in R_q \times R_q$ 인지 또는 $R_q \times R_q$ 에서 균등하게 무작위로 샘플되었는지 구별한다.

1.3 Module-LWE

Module-LWE 문제는 Ring-LWE 문제와 유사하지만 각 벡터를 다항식 환 R_q 에서 하나를 샘플링하는 것이 아닌 R_q^n 에서 n 개의 다항식을 샘플링한다[11]. 해당 벡터를 Module (모듈) 이라고 하며 n 은 모듈의 rank이다. $n = 1$ 인 경우는 Ring-LWE와 같다. Module-LWE의 보안강도는 모듈 격자 문제 해결의 보안강도와 비교할 수 있다. Module-LWE은 다음과 같이 정의될 수 있다.

■ **Definition of Module-LWE** : 벡터 $a \in R_q^n$ 와 비밀 벡터 $s \in R_q^n$ 의 원소들이 R_q 에서 균등하게 무작위로 선택되었다고 가정하고 (R_q 의 원소들은 Z_q 에서 균등하게 무작위로 선택되었다고 가정), 여러 e 가 R_q 에서 표준편차가 σ 인 가우스 분포에서 샘플링되었다고 가정한다.

■ **Search Module-LWE** : $(a, a \cdot s + e) \in R_q^n \times R_q$ 형태의 m 개의 샘플이 존재할 때 비밀벡터 s 를 복구한다.

■ **Decisional Module-LWE** : $(a, b) \in R_q^n \times R_q$ 형태의 m 개의 샘플이 주어지면 해당 샘플들이 $(a, a \cdot s + e) \in R_q^n \times R_q$ 인지 또는 $R_q^n \times R_q$ 에서 균등하게 무작위로 샘플되었는지 구별한다.

제 2 절 Sampler for Cryptography

격자 기반 암호화에서 각 격자의 구성요소와 환원된 문제인 LWE에서의 공개 행렬과 에러는 범위를 갖는 난수가 선택된다. 일반적으로 공개 행렬의 범위는 스킴의 모듈러스를 따르며, 에러의 범위는 매우 작은 정수의 값을 가진다. LWE의 보안성은 에러의 전파범위와의 분포가 가우스 분포 (정규분포)를 따른다는 것에 기인한다. NIST PQC 공모전에서 제안된 LWE는 대부분 다항식 환에서 정의되므로, 에러는 이산(Discrete) 가우스 분포를 따르도록 샘플링된다. 또한 공개행렬의 원소들의 분포도 범위 내에서 가우스 분포를 따라야 하므로 샘플링 방법을 이용하여 생성한다.

완벽한 이산/연속 가우스 분포는 구현상 정규화된 구조와 속도 최적화를 위한 테이블 생성 등으로 인해 부채널 분석(타이밍 공격)에 매우 취약하다. 따라서, 공모전에 제출된 알고리즘들은 추출된 값이 이산 가우스 분포를 따르는 특정 샘플러를 사용하여 구현된다. 일반적으로 샘플러의 성능 및 분포를 결정하기 위한 변수는 다음 세 가지의 튜플에 의해 결정된다[12].

■ σ (표준편차) : 평균에서의 데이터 분산을 조정하는 변수로, 예를 들어 $\mathbf{GD}(0, \sigma)$, with $\sigma = 2.6$ 의 경우 평균이 0인 표본들에 대해 가우시안 샘플링을 한다면 약 65%의 확률로 $(-\sigma, \sigma)$ 안에 있는 수가 선택된다는 것을 의미하며, 가우스 분포에서 표준편차별 확률변수는 고정값으로 정해져 있다.

■ λ (정밀도 매개변수) : LWE에서 완벽한 이산 가우스 샘플링을 통해 e 를 샘플하는 것은 불가능하므로, 이산 가우스 분포를 따르는 특정한 이산 샘플러

(Rejection Sampling, CDT Sampling, Bernoulli Sampler, etc)를 사용하여 값을 추출하는데 이때 발생하는 통계적 차이를 제어하기 위해 사용하는 매개변수이다.

■ τ (Tail-Cut): 테일 컷 매개변수는 선택한 총 분포에서 무시하고 싶은 분포를 나타낸다. 일반적으로 주어진 λ 에 대해 정의역을 제한하는 변수로 사용된다.

이러한 세가지 튜플 (σ, λ, τ) 의 각 매개변수는 샘플러의 보안과 효율성을 결정한다. 예를 들어, 표준 편차 σ 가 작을수록 미리 계산된 테이블을 저장하는데 필요한 메모리 공간이 줄어든다. LWE의 암호화/복호화 방식의 경우 $\sigma = 3.33$, 전자서명 샘플링에는 $\sigma = 215$ 을 사용하는 것이 제안되었으며, 추후 다양한 연구들에서 표준편차의 자릿수를 줄이는 방안이 제안되었고 최종적으로 샘플러에 사용되는 테이블의 크기가 개선되고 있다.

샘플러의 속도와 메모리 사용량은 λ 에 따라 달라진다. 일반적으로, λ 가 높을수록 보안성이 높아지고 더 큰 샘플 크기를 갖지만, 낮은 성능을 갖는다. 가우스 분포의 양 끝(꼬리)의 확률은 무시할 수 있는 확률로 생각하여 샘플한다. 즉, Tail-cut 매개변수(τ)를 분포의 양 끝을 무시하려는 분포의 값으로 표현한다면, 난수 e 는 $|e| \in \{0, \infty\}$ 대신에 $|e| \in \{0, \sigma\tau\}$ 에서 샘플된다. 샘플러가 사용하는 분포와 이산 가우스 분포의 통계적 거리는 일반적으로 λ 에 대해 $2^{-\lambda}$ 보다 작게 설정돼야 한다.

이산 가우스 분포를 샘플링하는 것은 높은 정밀도, 많은 임의 비트 및 거대한 테이블의 사용으로 인해 격자 기반 암호 스킴에서 가장 시간과 메모리를 많이 사용하는 부분 중 하나이다. 즉, 이산 가우스 샘플러(근사값)의 구현과 이론적인

(완벽한) 이산 가우스 분포 사이의 무시할 수 있는 통계적 거리의 계산은 정밀한 부동 소수점 산술(지수 함수 계산)을 요구하거나, 큰 메모리 공간 (미리 계산된 값을 저장)을 요구한다. 통계적 거리를 $2^{-\lambda}$ 미만으로 유지하려면 기본 플랫폼에서 기본적으로 표준 배정밀도보다 큰 부동 소수점 정밀도가 필수적이다. 따라서, 특정 환경에서는 완벽한 이산 가우스 샘플러를 구현하는 것은 불가능하다.

일반적으로, 이산 가우스 샘플러를 구현하고 직접 샘플링하는 것은 비효율적이기 때문에, 이를 해결하기 위해 λ 비트 균일(uniform) 난수(random) 정수(integer)를 이산 샘플러가 이산 가우스 분포를 따르도록 근사화하는 데 사용한다. 현재 λ 의 보안 레벨을 가지는 거의 완벽한 근사 이산 가우스 샘플러 알고리즘이 존재하지 않기 때문에 $\lambda/2$ 을 선택하여 근사 이산 가우스 샘플러를 구현한다. $\lambda/2$ 를 선택하면, 샘플러에서 비트의 절반을 잘라 더 작고 더 빠른 샘플러를 만들 수 있다. 정밀도 매개변수의 감소 (λ 에서 $\lambda/2$ 로)를 이용하면 Tail-Cut 매개변수(τ)도 $\tau = \sqrt{\lambda \times 2 \cdot \ln(2)}$ 로 변경될 수 있다.

2.1 Rejection Sampler

Rejection Sampling (이하, 리젝션 샘플링)은 샘플링하고 싶은 분포(타겟 분포) X 의 확률밀도함수가 $f(x)$ 일 때, 확률밀도함수가 $g(x)$ 인 다른 분포(제안 분포) Y 에서 샘플 x 를 추출하고, 이를 $\frac{f(x)}{Mg(x)}$ 의 확률을 통해 f 의 샘플로 인정하는 방식이다 ($M \geq 1, \text{with } \forall x, f(x) \leq Mg(x)$). 먼저 제안 분포 $g(x)$ 와 제안 분포가 $f(x)$ 를 덮을 수 있는 적당한 1이상의 값 M 을 선택한다. 그 후, $y \leq Mg(x)$ 을 만족하는 (x, y) 쌍을 임의로 선택한다. 일반적으로, 제안 분포 안에서 임의의 값을 선택하며 x 는 X 의 범위 내에 존재해야 한다. 그 뒤에, x 에 대해 $f(x)$ 를 계산하고 만약 $y \leq f(x)$ 이면 x 를 타겟 분포 X 의 확률변수로 선택하고, 아닌 경우 기각한다. 일반적으로 M 의 선택에 따라, 기각율을 낮출 수 있다.

LWE 환경에서 x 는 $|x| \in \{0, \sigma\tau\}$ 에서 선택된다. x 는 modulus q 이하 혹은 여러 e 의 범위를 가져야 하므로 특정 범위의 타겟 분포가 생성된다. 따라서, $\sigma\tau$ 범위를 갖는 x 를 선택하고 추출된 x 가 이산 가우스 분포를 따라야 하므로 σ 와 τ 은 가우스 분포의 특성에 맞게 유도된다. 즉, (x, y) 에서 $y \in [0, 1]$ 일 때, 이산 가우스 분포를 따르는 x 를 추출할 수 있다. 이때, x 는 이론적으로, $1 - e^{(-x^2/2\sigma^2)}$ 에 비례하는 확률로 리젝된다. 즉, 일반적으로 한 번의 accept을 위해 평균적으로 $2\tau/\sqrt{2\pi}$ 번의 reject이 발생한다. 따라서, 일반적으로 리젝션 샘플링은 다른 샘플러에 비해 상대적으로 느린 샘플링 방법이다.

2.2 CDT Sampler

Constant-time cumulative Distribution Table (CDT)는 누적 분포함수 (Cumulative density function, CDF)의 역함수를 통해 역함수의 값을 테이블로 만들어 샘플링하는 방법이다. CDT는 부동 소수점(Floating-Point) 연산을 사용하지 않고, 테이블 참조를 이용하기 때문에 일반적으로 리저션 샘플러보다 높은 성능을 보여준다. 누적 분포에서는 모든 숫자가 1보다 작으므로 분수의 이진 전개를 사용하면 충분하고, 이산 분포를 샘플링하는 경우 특정 값이 가지는 범위를 선택하여 테이블을 생성하기 때문에 연속분포보다 훨씬 효율성이 좋다. 하기 [Algorithm 1]은 CDT 샘플러와 이진탐색을 이용한 $D_{\mathbb{Z},\sigma}$ 에서의 샘플링 의사 코드로, 실제 동작 과정은 다음과 같다.

Algorithm 1 Sampling $D_{\mathbb{Z},\sigma}$ using the Cumulative Distribution Table (CDT) and binary searches

Require: Three Integers min , cur and jmp
Discrete Gaussian CDT Samples such that, $N = \tau \times \sigma$, $0 = S[0] < S[1] < \dots < S[N-3] = 1$
Sample a bit b uniformly in $\{0, 1\}$
Sample r uniformly in $\{0, \dots, (2^\lambda - 1)\}$
Ensure: $min \leftarrow 0$; $cur \leftarrow (N/2)$; $jmp \leftarrow cur$;
while ($jmp > 0$) **do**
 $cur \leftarrow min + jmp$;
 if ($r \geq S[cur]$) **then**
 $min \leftarrow cur$;
 $jmp \leftarrow jmp >> 1$;
return $x = (-1)^b min$

[Algorithm 1] CDT Sampler 알고리즘^[12]

- 1 : 초기값은 $\min = 0, \text{cur} = N/2, \text{jmp} = N/2$ 이며, jmp 는 이진탐색을 할 때 구간을 정해주는 변수로 0이 될 때까지 while 문을 진행한다. 이때, cur 은 $S[x]$ 의 값을 확인할 중간점이고 $\min$ 은 시작점을 의미한다.
- 2 : 반복문 안에서 cur 은 시작점 $\min$ 에서 중간값인 jmp 를 더한값으로 바뀌고 if문을 통해 r 값과 $S[\text{cur}]$ 값을 비교한다. 이후, r 값이 더 크다면 시작점 $\min$ 을 중간 값이었던 cur 로 바꾸고 jmp 를 2로 나눠주며 그렇지 않다면 $\min$ 의 값은 바꾸지 않고 jmp 만 2로 나눈다.
- 3 : 마지막으로, jmp 가 0이 되면 그 때의 $\min$ 값이 $S[\min] \leq r < S[\min + 1]$ 을 만족하기 때문에 샘플값으로 선택한다. 최종적으로 $(-1)^b \min$ 을 $D_{Z,\sigma}$ 의 샘플링 값으로 선택한다.

표준 CDT에는 최소 $\sigma\lambda\tau$ 비트 크기의 테이블이 필요하다. 예를 들어 BLISS-IV($\sigma = 13.4, \tau = 215, \lambda = 128$) 및 BLISS-I($\sigma = 13.4, \tau = 19.54, \lambda = 128$)는 최소 630-kbit 및 370-kbit 크기의 사전계산 테이블이 필요하다. 따라서, 리소스가 제한된 임베디드 장치에 CDT 샘플러는 그리 실용적이지 않다. 최근 테이블 크기를 줄이는 연구가 제안되기는 하였지만, 테이블을 사용하는 CDT 샘플러에 대한 부채널 분석(타이밍 공격)이 최근에 연구되어 실제 NIST 표준화 대상 PQC 알고리즘들은 사용하지 않고 있다.

2.3 Binomial Sampler

중심 이항 분포 (Centered Binomial distribution, CBD)는 $\exp()$ 를 계산하지 않고 사전 테이블을 생성할 필요가 없는 반올림된 가우스 샘플러(ξ_σ)의 근사치이다. $\sigma = \sqrt{8}$ 을 ξ_σ 의 표준편차라 정의하면 이항 분포는 $k = 2\sigma^2$ 으로 매개변수화시킬 수 있다. 샘플링 분포를 ψ_k 로 선택하면 $\sigma = \sqrt{8}$ 을 사용하는 반올림된 가우스 샘플러(ξ_σ)와의 통계적 거리는 무시할 수 있을 정도로 작다. 정수 $k \geq 0$ 에 대한 중심 이항 분포(ψ_k)는 $\{0, 1\}$ 에서 균일하게 $2 \cdot k$ 의 랜덤한 값 $(a_1, \dots, a_k, b_1, \dots, b_k)$ 을 추출하여 출력을 $\sum_{i=1}^k (a_i, b_i)$ 로 내보낸다. k 는 2^σ 로 선택되므로, 표준 편차가 큰 전자서명에 ψ_k 를 사용하는 것은 효율적이지 않다. NIST PQC 표준화 대상 알고리즘으로는 PKE/KEM 알고리즘인 Crystals-Kyber, SABER가 CBD를 사용중이다. 그러나, 전자서명인 Crystals-Dilithium에서는 보안 때문에 리젝션 샘플링을 사용한다.

2.4 Comparison of Sampler

본 절에서 분석한 샘플링 기법에 대한 성능, 메모리 요구량, 엔트로피, 특징 등을 정리하면 다음과 같다.

1. Rejection Sampler

- Speed : 느림
- FP $\exp()$: 10
- Table Size : 0
- Table Lookup : 0
- Entropy : $45 + 10\log_2\sigma$
- 특징 : 성능과 메모리가 제한된 환경에 알맞음

2. CDT Sampler

- Speed : 빠름
- FP $\exp()$: 0
- Table Size : $\sigma\tau\lambda$
- Table Lookup : $\log_2(\tau\sigma)$
- Entropy : $2.1 + \log_2\sigma$
- 특징 : 전자서명 스킴에서 구현하기 쉬움

3. Binomial Sampler

- Speed : 빠름
- FP $\exp()$: 0
- Table Size : 0



■ Table Lookup : 0

■ Entropy : $4\sigma^2$

■ 특징 : 전자서명 스킴에 알맞지 않음

리젝션 샘플링은 속도는 상대적으로 느리지만, 테이블을 사용하지 않는다는 점과 엔트로피가 높다는 점 그리고, 임베디드 장치에서도 사용될 수 있다는 장점 때문에 Crystals-Kyber 및 Crystals-Dilithium의 공개행렬 생성에 사용되고 있다. SABER의 경우 modulus $q = 2^{13}$ 이기 때문에 리젝션 샘플링을 사용하지 않고 스트림된 SHAKE의 출력값을 13-bit씩 조개어 사용한다. CDT, 샘플링 기법은 빠르지만, 테이블 크기 및 상대적으로 적은 엔트로피 (이항 분포에 비해), 그리고 부채널 분석(타이밍 공격) 때문에 NIST PQC 최종 후보 알고리즘들은 사용하지 않고 있다. 따라서, Crystals-Kyber와 SABER는 비밀 벡터 와 에러를 샘플링 할 때는 중심 이항 분포를 사용하고 있으며, Crystals-Dilithium의 경우 에러를 샘플링 할 때 큰 σ 을 사용해야하기 때문에 중심 이항 분포를 사용하지 않고 리젝션 샘플링을 사용한다.

제 3 절 Number Theoretic Transform

LWE 기반 암호체계는 보안강도 및 구현 효율성을 위해 원분 다항식 $f(x)$ 기반의 다항식 환 $R_q = \mathbb{Z}_q[x]/\langle f(x) \rangle$ 을 사용한다. NTRU와 같은 SVP/CVP (Shortest/Closest Vector Problem) 기반의 스킴들은 주로 $x^n + x^{n-1} + \dots + 1$ 을 원분 다항식으로 사용하고, LWE 기반의 스킴들은 주로 $x^n + 1$ 을 원분 다항식으로 사용한다. 모듈러스의 용이성을 위해 원분 다항식은 주로 최고차항의 계수가 1인 monic 구조를 사용한다. 다항식 환 상의 원소는 모두 $f(x)$ 미만의 차수를 갖는 다항식이므로 다항식 곱셈 시 q 와 $f(x)$ 을 통해 감산을 수행해야 한다. 일반적으로 $n-1$ 차 두 개의 다항식 $a(x)$ 와 $b(x)$ 를 곱하는 방법은 School Book, Karatsuba, Toom-Cook, Number-Theoretic Transform (NTT)이 존재한다. Crystals-Kyber는 NTT 기반 다항식 곱셈을 사용하므로, 본 절에서는 NTT 기반 구현기법에 대해 분석한다.

1차 직선을 정의하기 위해 2개의 점이 필요한 것과 마찬가지로 라그랑주 보간 공식에 의해 $n-1$ 차 다항식은 n 개의 점을 통해 유일하게 결정할 수 있다. 두 개의 $n-1$ 차 다항식 $a(x)$, $b(x)$ 를 서로 곱하는 경우 $2n-2$ 차의 다항식을 가지므로 $2n-1$ 개의 계수를 평가한다면 $c(x) = a(x) \times b(x)$ 를 유일하게 결정할 수 있다. $2n-1$ 개의 계수를 평가하기 위해 $2n-1$ 을 위수로 가지는 cyclic group의 generator (primitive roots of unity)를 다항식 계수의 범위 안에서 찾을 수 있다면, $2n-1$ 개의 계수를 평가할 수 있다. 평가된 계수들을 곱하고 보간하면 $c(x)$ 의 계수를 복원할 수 있다. 상기 방법은 NTT 기반 곱셈의 핵심 아이디어이며

평가 과정을 NTT 변환, 보간 과정을 InvNTT 변환이라고 한다.

다항식 환 R_q 상의 $a(x), b(x)$ 간의 곱셈 시 결과인 $c(x)$ 도 $n-1$ 차 다항식이 되어야 하는데 만약, 원분 다항식이 $f(x) = x^n + 1$ 꼴인 경우 Z_q 상에서 primitive n -th roots of unity를 찾을 수 있다면, n 개의 계수만 평가할 수 있다는 장점이 있다. $f(x) = x^n + 1$ 을 사용하는 R_q 상의 NTT 구현 기법을 Cyclic NTT라고 한다. R_q 상의 다항식 $a \in R_q$, $a = \sum_{i=0}^{n-1} a_i x^i$ 에 대해 Z_q 의 primitive n -th roots of unity를 w_n 라 하면 Cyclic NTT변환은 다음과 같이 정의할 수 있다.

$$\blacksquare \hat{a} = \text{NTT}(a) = \sum_{i=0}^{n-1} \hat{a}_i x^i \text{ with } \hat{a}_i = \sum_{j=0}^{n-1} a_j \cdot w_n^{ij}.$$

Cyclic InvNTT 변환은 다음과 같이 정의할 수 있다.

$$\blacksquare a = \text{InvNTT}(\hat{a}) = \sum_{i=0}^{n-1} a_i x^i \text{ with } a_i = n^{-1} \sum_{j=0}^{n-1} \hat{a}_j \cdot w_n^{-ij}.$$

그러나, 원분 다항식 $f(x) = x^n + 1$ 을 사용하는 체계에 대한 공격이 가능할 수 있다는 관점이 존재하여 표준화 대상 LWE 기반 양자내성암호 알고리즘들은 모두 $f(x) = x^n - 1$ 의 원분 다항식을 사용하고 있다. $f(x) = x^n - 1$ 을 원분 다항식으로 사용하는 경우 Cyclic NTT와 같이 평가값의 개수는 동일하지만 다항식 특성상 primitive $2n$ -th roots of unity인 w_{2n} ($w_{2n}^2 = w_n$)을 찾아야 한다는 단점이 있다. 상기 원분 다항식을 사용하는 NTT를 Negacyclic NTT라고 한다. Cyclic NTT에서 Negacyclic NTT로의 변환은 다항식 a 의 모든 계수 a_i 에 w_{2n}^i 를 곱한 다음에 Cyclic NTT 변환을 수행하면 된다. Negacyclic NTT변환은 다음과 같이 정의할 수 있다.

$$\blacksquare \hat{a} = \text{NTT}(a) = \sum_{i=0}^{n-1} \hat{a}_i x^i \text{ with } \hat{a}_i = \sum_{j=0}^{n-1} \hat{a}_j \cdot w_{2n}^j \cdot w_n^{ij}.$$

Negacyclic InvNTT 변환은 다음과 같이 정의할 수 있다.

$$\blacksquare a = \text{InvNTT}(\hat{a}) = \sum_{i=0}^{n-1} a_i x^i \text{ with } a_i = n^{-1} \cdot w_{2n}^{-i} \sum_{j=0}^{n-1} \hat{a}_j \cdot w_n^{-ij}.$$

정리하면, R_q 상의 다항식 a, b 에 대해 $c = a \times b$ 를 구하기 위해서는 각 다항식

에 대한 NTT 변환인 $\hat{a} = \text{NTT}(a)$, $\hat{b} = \text{NTT}(b)$ 를 연산한 후

$c = \text{InvNTT}(\hat{a} \cdot \hat{b})$ 를 계산하면 된다.

Algorithm 2 CT FFT implementing forward NTT

Input: Polynomial $a \in \mathbb{Z}_q/(x^{2^k} \pm 1)$; time domain, normal order
Input: Root of unity: ω_{2^k} for $\mathbb{Z}_q/(x^{2^k} - 1)$, $\omega_{2^{k+1}}$ for $\mathbb{Z}_q/(x^{2^k} + 1)$
Output: $\hat{a}$; NTT domain, bit-reversed order

```

1: for  $l = k - 1; l \geq 0; l \leftarrow l - 1$  do
2:   for  $i = 0; i < 2^{k-1-l}; i \leftarrow i + 1$  do
3:      $\psi \leftarrow \begin{cases} \omega_{2^k}^{\text{brv}_{k-1}(i)} & \text{for } \mathbb{Z}_q/(x^{2^k} - 1) \\ \omega_{2^{k+1}}^{\text{brv}_k(2^{k-1-l}+i)} & \text{for } \mathbb{Z}_q/(x^{2^k} + 1) \end{cases}$ 
4:     for  $j = i \cdot 2^{l+1}; j < i \cdot 2^{l+1} + 2^l; j \leftarrow j + 1$  do
5:        $t_0 \leftarrow a_j$ 
6:        $t_1 \leftarrow \psi \cdot a_{j+2^l}$ 
7:        $a_j \leftarrow t_0 + t_1$ 
8:        $a_{j+2^l} \leftarrow t_0 - t_1$ 
9:     end for
10:   end for
11: end for
```

[Algorithm 2] CT 기반의 NTT 구현 기법^[13]

NTT 변환시 값을 평가하는 과정에서 구현 효율성을 위해 중국인의 나머지 정리와 Fast Fourier Transform (FFT)에서 사용하는 Cooley-Tukey (CT)와 Gentleman-Sande (GS) 알고리즘을 사용하여 2-Radix ($f(x) = x^{2^k} \pm 1$)의 NTT 변환을 수행할 수 있다. CT와 GS 모두 반복문 또는 재귀함수를 이용하여 구현할 수 있으나 일반적으로 구현 효율성을 위해 반복문을 통해 구현한다.

[Algorithm 2]과 [Algorithm 3]은 CT와 GS를 이용한 NTT/InvNTT 변환의 반복문 기반 구현기법이다[13].

Algorithm 3 GS FFT implementing NTT^{-1}

Input: $\hat{a}$; NTT domain, bit-reversed order
Input: Root of unity: ω_{2^k} for $\mathbb{Z}_q/(x^{2^k} - 1)$, $\omega_{2^{k+1}}$ for $\mathbb{Z}_q/(x^{2^k} + 1)$
Output: Polynomial $a \in \mathbb{Z}_q/(x^{2^k} \pm 1)$; time domain, normal order

```

1: for  $l = 0; l < k; l \leftarrow l + 1$  do
2:   for  $i = 0; i < 2^{k-1-l}; i \leftarrow i + 1$  do
3:      $\psi \leftarrow \begin{cases} \omega_{2^k}^{-\text{brv}_{k-1}(i)} & \text{for } \mathbb{Z}_q/(x^{2^k} - 1) \\ \omega_{2^k}^{-\text{brv}_k(2^{k-1-l}+i)} & \text{for } \mathbb{Z}_q/(x^{2^k} + 1) \end{cases}$ 
4:     for  $j = i \cdot 2^{l+1}; j < i \cdot 2^{l+1} + 2^l; j \leftarrow j + 1$  do
5:        $t_0 \leftarrow a_j + a_{j+2^l}$ 
6:        $t_1 \leftarrow a_j - a_{j+2^l}$ 
7:        $a_j \leftarrow t_0$ 
8:        $a_{j+2^l} \leftarrow \psi \cdot t_1$ 
9:     end for
10:   end for
11: end for
12: for  $i = 0; i < 2^k; i \leftarrow i + 1$  do
13:    $a_i \leftarrow n^{-1} a_i$ 
14: end for

```

▷ Cancel out factor of 2 for each butterfly

[Algorithm 3] GS 기반의 InvNTT 구현 기법^[13]

제 3 장 Crystals-Kyber

제 1 절 Changelog of Crystals-kyber during NIST PQC Competition

1.1 Changelog

Crystals-Kyber^[14]은 Cryptographic Suite for Algebraic Lattices을 기반으로 설계된 PQC-PKE/KEM 알고리즘이다. Module-LWE를 기반으로 R_q 상의 다항식을 원소로 가지는 공개행렬 A 와 비밀 벡터 s , 에러 e 를 통해 알고리즘을 구성한다. 공개행렬 A 는 정방행렬로 크기는 보안수준 k 에 따라 결정된다. Crystals-Kyber는 Kyber.CPAPKE를 통해 Chosen-Plaintext Attack 보안 암호화 체계를 달성하고, Fujisaki-Okamoto의 변환의 변형을 사용하여 Kyber.CCAKEM을 구성하여 CCA 보안을 구축한다^[14]. Module-LWE 구조를 사용하기 때문에 공개 행렬 A 및 비밀 벡터와 오류 s, e 샘플링에 사용되는 난수 생성 및 해시 함수를 제외하고 $A \cdot s + e$ 구조에서의 다항식 곱셈 및 덧셈이다.

NIST PQC 공모전의 Round 2가 진행됨에 따라, Crystals-Kyber의 보안증명이 공개 키가 압축되지 않은 경우에만 유효하다는 의견이 제시됨에 따라 새로운 버전의 Crystals-Kyber가 Round 2에서 제안되었다. 공개키 압축을 하지 않고 기존의 공개 키의 크기를 유지하기 위하여 Crystals-Kyber의 소수 계수 q 를

기존의 7681에서 3329로 변경되어 $R_{3329} = \mathbb{Z}_{3329}[x]/(X^{256}+1)$ 로 업데이트 되었다. 또한, 공개키를 압축하지 않으므로 공개키가 NTT 도메인에서 전송될 수 있으며 이는 암호화 과정 및 KEM의 재 암호화 과정에서 NTT 작업이 절약된다. q 값이 작아짐에 따라, Module-LWE의 동일한 보안 강도를 제공하기 위해 더 작은 값의 e 가 필요하게 되었고, 이에 따라 η 는 모든 보안 강도에서 2로 고정되었다.

Round 2 에서 R_{3329} 를 선택함에 따라 $n = 256$ 에 대한 primitive $2n$ -th roots of unity가 없어 Crystals-Kyber은 온전한 Negacyclic NTT 변환을 사용할 수 없게 되었다. Crystals-Kyber의 개발팀은 기존의 Negacyclic NTT를 그대로 사용하지만, 분할되는 레이어의 수를 줄이는 선택을 하였다.

Crystals-Kyber에서 다항식 $a \in R_q$ 에 대한 NTT 변환은 다음과 같이 다시 정의되었다.

$$NTT(a) = \hat{a} = (\hat{a}_0 + \hat{a}_1 X + \hat{a}_2 X + \dots + \hat{a}_{255} X),$$

where,

$$\hat{a}_{2i} = \sum_{j=0}^{127} a_{2j} \zeta^{(2br_7(i) + 1)j}$$

$$\hat{a}_{2i+1} = \sum_{j=0}^{127} a_{2j+1} \zeta^{(2br_7(i) + 1)j}$$

primitive n -th roots of unity $\zeta = 17$ 이고, $br_7(x)$ 는 x 의 bit-reverse 표현이다. NTT 변환 후의 계수는 1차 다항식이므로 계수별 연산은 $X^2 - \zeta^{2br_7(i) + 1}$ 에서의 1차 다항식에 대한 모듈 다항식 곱셈으로 재정의 된다.

제 2 절 Specification for Crystals-Kyber

2.1 Notation and Definition

본 항목에서는 Crystals-Kyber 명세에서 사용되는 표기법과 그 의미를 설명한다.

■ B, B^k, B^* : $\{0, \dots, 255\}$ 범위의 바이트 값, 길이가 k 인 바이트 배열, 가변 바이트 배열

■ β, β_i : 비트 값, 비트 배열에서 i 번째 비트

■ BytesToBits() : l 바이트 배열을 입력으로 받아서 $8l$ 비트 배열을 반환하는 함수

■ $R_q = \mathbb{Z}_q[X]/(X^n + 1)$: Kyber가 기반하고 있는 n 차의 다항식 링 (계수는 소수 q 상의 값, $n = 2^{n'-1}$, $q = 3329$, $n' = 9$)

■ $\mathbf{v}, \mathbf{V}, \mathbf{v}^T, \mathbf{V}^T$: 다항식 벡터 (기본적으로 열벡터), 다항식 행렬, 전치 벡터, 전치 행렬

■ $\mathbf{v}[i], \mathbf{V}[i][j]$: 벡터에서 i 번째 다항식, i 행- j 열에 위치한 다항식

■ $r' = r \bmod^{\pm} \alpha$: 짝수인 경우 (또는, 홀수인 경우)

$-\frac{\alpha}{2} < r' \leq \frac{\alpha}{2}$ (또는 $-\frac{\alpha-1}{2} \leq r' \leq \frac{\alpha-1}{2}$) 상에 존재하는 유일한 r' 값

($r' = r \bmod \alpha$)

■ $r' = r \bmod^+ \alpha$: $0 \leq r' < \alpha$ 상에 존재하는 r' 값 ($r' = r \bmod \alpha$)

■ $\lceil x \rceil$: x 값에 대한 라운딩 (x 에 가장 가까운 정수)

- $s \leftarrow S$: S 로부터 랜덤 유니폼하게 선택됨 (S 가 확률분포일 경우, s 는 S 분포로부터 선택됨을 의미함)
- $\text{PRF} : B^{32} \times B \rightarrow B^*$: 수도랜덤함수 (Pseudo Random Function)
- $\text{XOF} : B^* \times B \times B \rightarrow B^*$: 확장가능한 출력함수(extendable output function)
- $H : B^* \rightarrow B^{32}$, $G : B^* \rightarrow B^{32} \times B^{32}$: 2종 해시함수
- $\text{Compress}_q(x, d)$: $x \in \mathbb{Z}_q$ 값을 입력으로 받아 $\{0, \dots, 2^d - 1\}$ 상의 하나의 값을 출력하는 함수 ($q < \lceil \log_2(q) \rceil$. 복호화 기능에 크게 영향을 미치지 않음을 보장하면서, 암호문의 하위 비트를 버림으로써 전송 크기를 줄이기 위한 기능)
- $x' = \text{Decompress}_q(X, d)$: $X = \text{Compress}_q(x, d)$ 로서, x 에 근접한 x' 값을 반환하는 함수
- $\text{NTT}, \text{NTT}^{-1} : R_q$ 상에서의 다항식에 대해 NTT 변환을 수행, 역 NTT 변환을 수행
- $\text{NTT}^{-1}(\text{NTT}(f) \circ \text{NTT}(g))$: f, g 에 대해 NTT 변환을 수행한 후, point-wise 곱셈을 수행. 다시 이 결과에 대해 역 NTT 변환을 수행
- $\hat{f}, \hat{\mathbf{v}}, \hat{\mathbf{V}}$: NTT 변환이 적용된 다항식, 다항식 벡터, 다항식 행렬

2.2 Parameter

[표 1] Crystals-Kyber의 파라미터[14]

파라미터	n	k	q	η_1	η_2	(d_u, d_v)	δ
512	256	2	3329	3	2	(10, 4)	2^{-139}
768	256	3	3329	2	2	(10, 4)	2^{-164}
1024	256	4	3329	2	2	(11, 5)	2^{-174}

Crystals-kyber는 보안 수준에 따라 총 3개의 파라미터 셋을 제공하며 [표 1]와 같다^[15]. 즉, NIST가 정의한 보안수준 1, 3, 5에 대해 kyber512, kyber768, kyber1024가 대응된다. n 은 다항식의 차수를 의미하고, k 는 공개정방행렬에서의 차수를 의미한다. q 는 다항식 계수가 정의되는 소수를 의미하고, η_1, η_2 각각은 kyber 암호화 시에 오류값(잡음값)을 랜덤하게 생성할 때 사용되는 값의 범위이다. (d_u, d_v) 는 Compress와 Decompress에서 사용되는 값이다. δ 는 파라미터 셋에 대한 복호화 실패확률이다.

2.3 Sampling of Crystals-Kyber

■ 이진분포 샘플링 (Centered Binomial Distribution)

Crystals-Kyber에서의 잡음원은 중심이항분포 $B_\eta(\eta=2,3)$ 로부터 샘플된다. 즉, B_η 은 다음과 같이 정의된다. 즉, 다항식의 각 계수가 B_η 로부터 샘플됨을 의미한다.

$$B_\eta = \sum_{i=1}^{\eta} (a_i - b_i), \text{ Sample } (a_1, \dots, a_\eta, b_1, \dots, b_\eta) \leftarrow \{0,1\}^{2\eta}$$

Algorithm 4 $\text{CBD}_\eta: \mathcal{B}^{64\eta} \rightarrow R_q$

Input: Byte array $B = (b_0, b_1, \dots, b_{64\eta-1}) \in \mathcal{B}^{64\eta}$

Output: Polynomial $f \in R_q$

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

for i from 0 to 255 **do**

$a := \sum_{j=0}^{\eta-1} \beta_{2i\eta+j}$

$b := \sum_{j=0}^{\eta-1} \beta_{2i\eta+\eta+j}$

$f_i := a - b$

end for

return $f_0 + f_1X + f_2X^2 + \dots + f_{255}X^{255}$

[Algorithm 4] Crystals-Kyber의 중심이항분포 알고리즘[14]

[Algorithm 4]은 256차 다항식을 B_η 으로 샘플링하는 알고리즘이며 256차 다

항식의 각 계수를 η 값 범위의 중심이항분포의 값으로 샘플링한다. 입력값 $B^{64\eta}$

에 대해 BytesToBits()를 적용하여 비트열로 변환한 후에, B_η 의 정의에 따라 i 번

째 계수 f_i 값을 샘플링한다. 예를 들어, $\eta=2$ 인 경우, $(\beta_0, \dots, \beta_{1023})$ 에 대해

$f_i = a - b$, $a = \sum_{j=0}^1 \beta_{4i+j}$, $b = \sum_{j=0}^1 \beta_{4i+2+j}$ 를 계산한다. B_η 의 원래 정의와는 달리 실제

로는 $(\beta_0, \dots, \beta_{1023})$ 에 a 와 b 가 2비트씩 번갈아 가면서 저장되어있다. B_η 으로부터

샘플되었기 때문에 다항식 계수의 최소값과 최대값은 $\eta=2$ (또는, $\eta=3$)인 경우

각각 -3과 3된다 (또는, -7와 7가 된다). 키생성과정과 암호화 과정에서 생성되

는 노이즈 또는 에러 다항식이 바로 CBD_η 로 샘플링된다.

■ 균등분포 샘플링 (Uniform Distribution)

Crystals-Kyber에서 노이즈, 에러를 제외한 R_q 의 다항식은 균등분포를 이용

하여 샘플링된다. 즉, 결정론적인 난수생성방법을 이용하여 랜덤한 바이트 스트

링을 생성하며 몇 개의 바이트를 조합하여 q 범위의 값으로 매핑한다.

[Algorithm 5]은 입력된 가변 길이의 바이트 스트링에 저장된 랜덤값을 이용하여 NTT 도메인으로 표현된 R_q 상의 다항식 계수를 반환한다. 즉, B^* 은 8비트 랜덤값이 연결된 데이터로 볼 수 있으며, 이 데이터를 적절히 분할하여 q 에 속하는 256개의 계수를 생성하는 것이다. 이를 수행하기 위해 바이트 스트링에서 3바이트 씩 로드되어 12비트값 d_1, d_2 값을 추출한다. Crystals-Kyber에서 사용하는 q 값은 3329 ($q < 2^{12}$)이기 때문에 d_1, d_2 값이 3329보다 작은 경우에 대해서만 계수로 취하고 그렇지 않으면 버린다. 이처럼 값의 범위에 만족하는 균등 분포의 값만을 취하는 것을 리젝션 샘플링이라고 한다. 실제 구현상에서는 몇 개의 바이트가 기각 샘플링을 통과할 수 있을지 결정할 수 없으므로 랜덤 씨드 값으로부터 가변 길이의 난수열을 출력하는 XOF를 이용하여 특정 길이만큼의 랜덤 바이트 스트링을 생성한 후 기각 샘플링을 적용하는 단계를 반복한다. 즉, 256개의 계수가 충족되지 않으면 추가적으로 XOF를 수행한 후 기각 샘플링을 적용한다.

2.4 Encoding and Decoding

■ Encoding (인코딩)

인코딩은 메모리 상에서 표현되었던 다항식을 전송을 위해 바이트 배열로 직렬화 (serialization)하는 과정이다. 즉, 메모리 상에서 다항식을 구성하는 하나의 계수는 12비트를 차지한다 (하지만 실제로 16비트 자료형에 저장된다). 즉, 12비트는

2개의 바이트에 걸친 데이터이기 때문에 인코딩 과정을 거치지 않고 전송할 경우 바이트를 전송해야 한다. 하지만, 2개의 12비트 계수는 3바이트에 저장될 수 있기 때문에, 인코딩 과정을 거쳐 바이트만으로 전송할 수 있다.

Algorithm 5 Parse: $\mathcal{B}^* \rightarrow R_q^n$

Input: Byte stream $B = b_0, b_1, b_2 \dots \in \mathcal{B}^*$

Output: NTT-representation $\hat{a} \in R_q$ of $a \in R_q$

```

 $i := 0$ 
 $j := 0$ 
while  $j < n$  do
     $d_1 := b_i + 256 \cdot (b_{i+1} \bmod^{+} 16)$ 
     $d_2 := \lfloor b_{i+1}/16 \rfloor + 16 \cdot b_{i+2}$ 
    if  $d_1 < q$  then
         $\hat{a}_j := d_1$ 
         $j := j + 1$ 
    end if
    if  $d_2 < q$  and  $j < n$  then
         $\hat{a}_j := d_2$ 
         $j := j + 1$ 
    end if
     $i := i + 3$ 
end while
return  $\hat{a}_0 + \hat{a}_1 X + \dots + \hat{a}_{n-1} X^{n-1}$ 

```



[Algorithm 5] Crystals-Kyber의 균등분포 알고리즘[14]

■ Decoding (디코딩)

디코딩 과정은 바이트 배열에 저장되어 전송된 다항식 계수들을 연산을 위해 12비트씩 파싱하여 메모리에 위치하도록 하는 것이다. 즉, 바이트 배열을 비트 열로 변환한 후 12비트 단위씩 파싱하여 16비트 자료형에 저장하는 것이며, 이를 256번 반복하면 하나의 다항식을 구성할 수 있다.

2.5 Kyber.CPAPKE

■ CPAPKE.Key Generation

[Algorithm 3]은 Crystals-Kyber의 CPA 안전성을 만족하는 PKE 스킴의 키 생성 과정이다. 본 과정은 CCA 안전성을 만족하는 KEM의 키 생성과정에 서브 함수로서 사용된다. 알고리즘의 출력값은 pk, sk 로서 각각 공개키와 개인키를 의미한다.

Algorithm 6 KYBER.CPAPKE.KeyGen(): key generation

Output: Secret key $sk \in \mathcal{B}^{12 \cdot k \cdot n/8}$

Output: Public key $pk \in \mathcal{B}^{12 \cdot k \cdot n/8+32}$

```

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

```

▷ Generate matrix $\hat{\mathbf{A}} \in R_q^{k \times k}$ in NTT domain

▷ Sample $\mathbf{s} \in R_q^k$ from B_{η_1}

▷ Sample $\mathbf{e} \in R_q^k$ from B_{η_1}

▷ $pk := \mathbf{A}\mathbf{s} + \mathbf{e}$

▷ $sk := \mathbf{s}$

[Algorithm 6] Crystals-Kyber의 CPAPKE.KeyGen 알고리즘[14]

과정	Algorithm 6 설명
1	- 32바이트 랜덤 데이터 생성
2	- 해시함수 G 를 이용하여 2개의 랜덤 씨드값 (ρ, σ) 생성
3	- 과정 9-12, 과정 13-16의 PRF에 입력되는 카운터 값 초기화
4-8	- 공개 행렬 생성 $\hat{\mathbf{A}} \in R_q^{k \times k}$ ($\hat{\mathbf{A}}$ 행렬의 다항식은 모두 NTT 도메인으로 표현됨) - 과정 2에서 생성된 씨드값 ρ 과 루프 카운터 i, j 를 입력으로 XOF 함수를 호출하면서 생성되는 랜덤 바이트 값을 Parse 함수를 이용하여 q 상에서 값으로 변환 (리저렉션 샘플링 기반)
9-12	- 비밀키 역할을 하는 벡터 $\mathbf{s} \in R_q^k$ 생성 - 랜덤 씨드값 σ 와 카운터 값 N 을 입력으로 하여 PRF 함수를 구동하여 그 결과값을 CBD_{η_1} 로 샘플링하여 벡터 생성
13-16	- 과정 9-12와 동일한 과정을 거쳐서 에리 벡터 생성
17-18	- $\mathbf{s}$ 와 $\mathbf{e}$ 벡터에 포함된 각 k 개의 다항식을 NTT 도메인으로 변환
19	- NTT 도메인으로 표현된 행렬과 벡터간의 곱셈 수행 후 에리 벡터 덧셈 수행 (NTT 도메인으로 표현되었기 때문에 다항식 곱셈은 point-wise 형태로 수행됨) - 최종 결과 $\hat{\mathbf{t}}$ 는 공개키로서 k 개의 다항식으로 구성된 벡터임
20-21	- 공개키 pk 는 $\hat{\mathbf{t}}$ 와 씨드값 ρ 를 연결하여 인코딩하여 바이트 배열로 저장 - 개인키 sk 는 $\hat{\mathbf{s}}$ 값을 인코딩하여 바이트 배열로 저장
return	- (pk, sk) 반환

■ CPAPKE.Encryption

[Algorithm 7]은 KYBER의 CPA 안전성을 만족하는 PKE 스킴의 암호화 과정이다. 본 과정은 CCA 안전성을 만족하는 KEM의 암호화 과정의 서브함수로 사용된다. 입력값은 공개키 pk , 메시지 m , 그리고 랜덤 씨드값 r 이 되며, 출력값은 암호문 $c = (c_1 || c_2)$ 을 출력한다.

Algorithm 7 KYBER.CPAPKE.Enc(pk, m, r): encryption

Input: Public key $pk \in \mathcal{B}^{12 \cdot k \cdot n/8 + 32}$

Input: Message $m \in \mathcal{B}^{32}$

Input: Random coins $r \in \mathcal{B}^{32}$

Output: Ciphertext $c \in \mathcal{B}^{d_u \cdot k \cdot n/8 + d_v \cdot n/8}$

```

1:  $N := 0$ 
2:  $\hat{\mathbf{t}} := \text{Decode}_{12}(pk)$ 
3:  $\rho := pk + 12 \cdot k \cdot n/8$ 
4: for  $i$  from 0 to  $k - 1$  do                                ▷ Generate matrix  $\hat{\mathbf{A}} \in R_q^{k \times k}$  in NTT domain
5:   for  $j$  from 0 to  $k - 1$  do
6:      $\hat{\mathbf{A}}^T[i][j] := \text{Parse}(\text{XOF}(\rho, i, j))$ 
7:   end for
8: end for
9: for  $i$  from 0 to  $k - 1$  do                                ▷ Sample  $\mathbf{r} \in R_q^k$  from  $B_{\eta_1}$ 
10:   $\mathbf{r}[i] := \text{CBD}_{\eta_1}(\text{PRF}(r, N))$ 
11:   $N := N + 1$ 
12: end for
13: for  $i$  from 0 to  $k - 1$  do                                ▷ Sample  $\mathbf{e}_1 \in R_q^k$  from  $B_{\eta_2}$ 
14:   $\mathbf{e}_1[i] := \text{CBD}_{\eta_2}(\text{PRF}(r, N))$ 
15:   $N := N + 1$ 
16: end for
17:  $e_2 := \text{CBD}_{\eta_2}(\text{PRF}(r, N))$                                 ▷ Sample  $e_2 \in R_q$  from  $B_{\eta_2}$ 
18:  $\hat{\mathbf{r}} := \text{NTT}(\mathbf{r})$ 
19:  $\mathbf{u} := \text{NTT}^{-1}(\hat{\mathbf{A}}^T \circ \hat{\mathbf{r}}) + \mathbf{e}_1$                                 ▷  $\mathbf{u} := \mathbf{A}^T \mathbf{r} + \mathbf{e}_1$ 
20:  $v := \text{NTT}^{-1}(\hat{\mathbf{t}}^T \circ \hat{\mathbf{r}}) + e_2 + \text{Decompress}_q(\text{Decode}_1(m), 1)$     ▷  $v := \mathbf{t}^T \mathbf{r} + e_2 + \text{Decompress}_q(m, 1)$ 
21:  $c_1 := \text{Encode}_{d_u}(\text{Compress}_q(\mathbf{u}, d_u))$ 
22:  $c_2 := \text{Encode}_{d_v}(\text{Compress}_q(v, d_v))$ 
23: return  $c = (c_1 \| c_2)$                                 ▷  $c := (\text{Compress}_q(\mathbf{u}, d_u), \text{Compress}_q(v, d_v))$ 

```

[Algorithm 7] Crystlas-Kyber의 CPAPKE.Enc 알고리즘^[14]



단계	Algorithm 4 설명
1	- PRF에서 사용되는 카운터 값 N 초기화
2	- pk 로부터 디코딩 과정을 거쳐 $\hat{\mathbf{t}}$ 벡터를 추출
3	- pk 에 포함된 씨드값 ρ 추출 ($\hat{\mathbf{A}}$ 생성 시에 사용됨)
4-8	- 공개 행렬 생성 $\hat{\mathbf{A}}^T$ ($\hat{\mathbf{A}}$ 행렬의 전치) 생성 - 씨드값 ρ 과 루프 카운터 i, j 를 입력으로 XOF 함수를 호출하면서 생성되는 랜덤 바이트 값을 Parse 함수를 이용하여 q 상에서의 값으로 변환 (리젝션 샘플링 기반)
9-12	- 암호화 과정에서 랜덤성을 부여하는 벡터 $\mathbf{r} \in R_q^k$ 생성 - 랜덤 씨드값 r 과 카운터 값 N 을 입력으로 하여 PRF 함수를 구동하여 그 결과값을 CBD_{η_1} 로 샘플링하여 벡터 생성
13-16	- 에러 벡터 $\mathbf{e}_1 \in R_q^k$ 생성 - 랜덤 씨드값 r 과 카운터 값 N 을 입력으로 하여 PRF 함수를 구동하여 그 결과값을 CBD_{η_2} 로 샘플링하여 벡터 생성
17	- 랜덤 씨드값 r 과 카운터 값 N 을 입력으로 하여 PRF 함수를 구동하여 그 결과값을 CBD_{η_2} 로 샘플링하여 다항식 $\mathbf{e}_2 \in R_q$ 생성
18	- $\mathbf{r}$ 에 대해 NTT 변환을 적용하여 $\hat{\mathbf{r}}$ 계산
19	- $\hat{\mathbf{A}}^T \circ \hat{\mathbf{r}}$: 행렬과 벡터 곱 연산 수행 (다항식 곱셈은 point-wise 형태로 수행됨) - $NTT^{-1}(\hat{\mathbf{A}}^T \circ \hat{\mathbf{r}})$: 역 NTT 변환 수행 ($\mathbf{A}^T \mathbf{r}$) - $NTT^{-1}(\hat{\mathbf{A}}^T \circ \hat{\mathbf{r}}) + \mathbf{e}_1$: ($\mathbf{A}^T \circ \mathbf{r}$) + $\mathbf{e}_1$ 을 계산하여 $\mathbf{u}$ 벡터에 저장
20	- $\hat{\mathbf{t}}^T \circ \hat{\mathbf{r}}$: 벡터와 벡터 곱 연산 수행 (다항식 곱셈은 point-wise 형태로 수행됨) - $NTT^{-1}(\hat{\mathbf{t}}^T \circ \hat{\mathbf{r}})$: 역 NTT 변환 수행 ($\mathbf{t}^T \mathbf{r}$) - $Decode_1(m)$: 메시지 $m \in B^{32}$ 은 256비트이므로 1비트씩 256개의 데이터로 분할됨 - $Decompress_q(Decode_1(m), 1)$: 비트에 대해서 0은 0으로 매핑하고, 1은 $\lceil q/2 \rceil$ 로 매핑 - $NTT^{-1}(\hat{\mathbf{t}}^T \circ \hat{\mathbf{r}}) + Decompress_q(Decode_1(m), 1) + \mathbf{e}_2$: 계산을 완료하여 v 로 저장
21	- $Compress_q(\mathbf{u}, d_u)$: 과정 19에서 생성된 $\mathbf{u}$ 의 계수를 $\{0, \dots, 2^{d_u} - 1\}$ 범위로 매핑 - $Encode_{d_u}(Compress_q(\mathbf{u}, d_u))$: d_u 비트 범위의 각 계수를 바이트 배열로 저장하여 암호문 $\mathbf{c}_1$ 벡터 생성

22	- $\text{Compress}_q(v, d_v)$: 과정 20에서 생성된 v의 계수를 $\{0, \dots, 2^{d_u} - 1\}$ 범위로 매핑 - $\text{Encode}_{d_v}(\text{Compress}_q(v, d_v))$: d_v비트 범위의 각 계수를 바이트 배열로 저장하여 암호문 c_2 다항식 생성
return	- $c = (\mathbf{c}_1, c_2)$ 반환

[Algorithm 7]을 통하여 최종 생성된 $\mathbf{c}_1$ 과 c_2 는 실제 과정 19, 20을 통해 계산된 값에서 압축된 데이터이다. 즉, 전송효율성을 고려하여 암호문은 복호화 실패율을 크게 떨어뜨리지 않는 선에서 압축되어 저장되는 것이다. 예를 들어, $\mathbf{u}$ 벡터에 포함된 다항식들의 각 계수는 q 상의 원소이지만 압축 과정을 통해 각 계수는 d_u 비트로 표현된다 (q 는 12비트, d_u 는 10비트 또는 11비트). 또한, v 다항식의 계수 역시 압축 과정을 통해 각 계수는 d_v 비트로 표현된다. 따라서, 암호문의 총 크기는 d_u 비트 계수를 n 개 가진 다항식을 k 개 가진 $\mathbf{c}_1$ 벡터의 바이트 크기인 $d_u \cdot k \cdot n/8$ 과 d_v 비트 계수를 n 개 가진 다항식 c_2 의 바이트 크기 $d_v \cdot n/8$ 를 합한 것이 된다. 과정 20에서 메시지 m 을 구성하는 각 비트를 구분한 후, 비트의 값이 1인 경우는 $\lceil q/2 \rceil$ 로 매핑 한다.

■ CPAPKE.Decryption

Algorithm 8 KYBER.CPAPKE.Dec(sk, c): decryption

Input: Secret key $sk \in \mathcal{B}^{12 \cdot k \cdot n/8}$

Input: Ciphertext $c \in \mathcal{B}^{d_u \cdot k \cdot n/8 + d_v \cdot n/8}$

Output: Message $m \in \mathcal{B}^{32}$

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

[Algorithm 8] Crystals-Kyber의 CPAPKE.Dec 알고리즘[14]

단계	Algorithm 8 설명
1	- $\text{Decode}_{d_u}(c, d_u)$: 전송된 암호문 c에서 $\mathbf{c}_1$를 디코딩 - $\text{Decompress}_q(\mathbf{c}_1, d_u)$: $\mathbf{c}_1$에 대해서 압축 해제 적용
2	- $\text{Decode}_{d_v}(c + d_u \cdot k \cdot n/8, d_v)$: 전송된 암호문 c에서 c_2를 디코딩 - $\text{Decompress}_q(c_2, d_v)$: c_2에 대해서 압축 해제 적용
3	- 비밀키 sk 디코딩
4	- $a \leftarrow \text{NTT}^{-1}(\hat{\mathbf{s}}^T \circ \text{NTT}(\mathbf{u}))$: 개인키 $\hat{\mathbf{s}}^T$와 $\mathbf{u}$를 NTT 도메인상에서 곱한 후, 역 NTT를 적용하여 원본 도메인으로 복원 - $t \leftarrow v - a$: v에서 a를 뺀 - $\text{Encode}_1(\text{Compress}_q(t, 1))$: t의 각 계수를 1비트로 압축하여 인코딩하여 반환
return	- 원본 메시지 m 반환

2.6 Kyber.CCAKEM

KYBER KEM은 slightly tweaked Fujisaki-Okamoto transform 방법을 이용하여 IND-CCA2에 안전하도록 설계된 것으로 내부적으로 KYBER PKE 방법을 서브함수로서 사용한다.

■ CCAKEM.Key Generation

Algorithm 9 KYBER.CCAKEM.KeyGen()

Output: Public key $pk \in \mathcal{B}^{12 \cdot k \cdot n/8 + 32}$

Output: Secret key $sk \in \mathcal{B}^{24 \cdot k \cdot n/8 + 96}$

- 1: $z \leftarrow \mathcal{B}^{32}$
- 2: $(pk, sk') := \text{KYBER.CPAPKE.KeyGen}()$
- 3: $sk := (sk' || pk || H(pk) || z)$
- 4: **return** (pk, sk)

[Algorithm 9] Crystlas-Kyber의 CCAKEM.KeyGen 알고리즘[14]

단계	Algorithm 9 설명
1	- 랜덤한 B^{32} 바이트 스트링 z 생성
2	- KYBER.CPAPKE.KeyGen() 함수를 이용하여 pk, sk' 생성
3	- 과정 2에서 생성된 sk' 에 $pk, H(pk), z$ 를 연결하여 sk 생성
return	- pk, sk 반환

■ CCAKEM.Enc

Algorithm 10 KYBER.CCAKEM.Enc(pk)

Input: Public key $pk \in \mathcal{B}^{12 \cdot k \cdot n/8 + 32}$

Output: Ciphertext $c \in \mathcal{B}^{d_u \cdot k \cdot n/8 + d_v \cdot n/8}$

Output: Shared key $K \in \mathcal{B}^*$

- 1: $m \leftarrow \mathcal{B}^{32}$
- 2: $m \leftarrow H(m)$
- 3: $(\tilde{K}, r) := G(m || H(pk))$
- 4: $c := \text{KYBER.CPAPKE.Enc}(pk, m, r)$
- 5: $K := \text{KDF}(\tilde{K} || H(c))$
- 6: **return** (c, K)



▷ Do not send output of system RNG

[Algorithm 10] Crystlas-Kyber의 CCAKEM.Enc 알고리즘[14]

단계	Algorithm 10 설명
1-3	- sk 를 분할하여 pk, h, z 값을 추출
4	- 개인키와 암호문을 KYBER.CPAPKE.Dec() 함수에 전달하여 메시지 m' 복원
5	- 복원된 메시지와 h 값을 연결하여 해시값을 계산하여 $\tilde{K}'$ 과 r' 생성
6	- pk, m', r' 을 입력으로 하여 KYBER.CPAPKE.Enc() 함수를 호출하여 c' 생성
7-11	- c 와 c' 이 일치할 경우 의미있는 공유키 생성, 그렇지 않을 경우 랜덤 z 값과 $H(c)$ 값을 연결하여 KDF 함수를 구동하여 반환된 의미없는 공유키 생성
return	- 공유키 반환

■ CCAKEM.Dec

Algorithm 11 KYBER.CCAKEM.Dec(c, sk)

Input: Ciphertext $c \in \mathcal{B}^{d_u \cdot k \cdot n/8 + d_v \cdot n/8}$

Input: Secret key $sk \in \mathcal{B}^{24 \cdot k \cdot n/8 + 96}$

Output: Shared key $K \in \mathcal{B}^*$

```

1:  $pk := sk + 12 \cdot k \cdot n/8$ 
2:  $h := sk + 24 \cdot k \cdot n/8 + 32 \in \mathcal{B}^{32}$ 
3:  $z := sk + 24 \cdot k \cdot n/8 + 64$ 
4:  $m' := \text{KYBER.CPAPKE.Dec}(s, (u, v))$ 
5:  $(\bar{K}', r') := G(m' || h)$ 
6:  $c' := \text{KYBER.CPAPKE.Enc}(pk, m', r')$ 
7: if  $c = c'$  then
8:   return  $K := \text{KDF}(\bar{K}' || H(c))$ 
9: else
10:  return  $K := \text{KDF}(z || H(c))$ 
11: end if
12: return  $K$ 

```

[Algorithm 11] Crystals-Kyber의 CCAKEM.Dec 알고리즘[14]

단계	Algorithm 11 설명
1-3	- sk 를 분할하여 pk, h, z 값을 추출
4	- 개인키와 암호문을 KYBER.CPAPKE.Dec() 함수에 전달하여 메시지 m' 복원
5	- 복원된 메시지와 h 값을 연결하여 해시값을 계산하여 $\bar{K}'$ 과 r' 생성
6	- pk, m', r' 을 입력으로 하여 KYBER.CPAPKE.Enc() 함수를 호출하여 c' 생성
7-11	- c 와 c' 이 일치할 경우 의미있는 공유키 생성, 그렇지 않을 경우 랜덤 z 값과 $H(c)$ 값을 연결하여 KDF 함수를 구동하여 반환된 의미없는 공유키 생성
return	- 공유키 반환

2.7 Analysis source code

■ PQClean Crystals-Kyber 프로젝트 구성

PQClean 라이브러리 코드는 레퍼런스 코드를 비롯하여, 최신 최적화 방안들이 적용되어있는 코드이다. 본 논문에서는 PQClean 라이브러리의 Crystals-Kyber의 코드를 분석하도록 한다. PQClean에 포함되어있는 Crystals-Kyber 프로젝트는 [표 2]와 같은 구성을 가진다. 즉, 기본 CPU 환경에서의 프로젝트, CPU환경에서 AVX2를 이용한 프로젝트, ARMv8환경에서의

프로젝트로 구성된다. 또한, 추가적으로 랜덤 데이터 생성 시 AES가 사용되는 경우 90s 버전으로 구분된다.

[표 2] PQclean Crystals-Kyber 프로젝트 구성

환경	연관 프로젝트	비고
CPU	pqclean-kyber512-ref	랜덤 데이터 생성 시, SHA3 이용
	pqclean-kyber768-ref	
	pqclean-kyber1024-ref	
	pqclean-kyber512-90s-ref	랜덤 데이터 생성 시, AES 이용
	pqclean-kyber768-90s-ref	
	pqclean-kyber1024-90s-ref	
AVX2	pqclean-kyber512-avx2	AVX2 벡터 명령어를 이용하여 연산 고속화 적용
	pqclean-kyber768-avx2	
	pqclean-kyber1024-avx2	
	pqclean-kyber512-90s-avx2	AVX2 벡터 명령어를 이용하여 연산 고속화 적용 및 랜덤 데이터 생성 시, AES 이용
	pqclean-kyber768-90s-avx2	
	pqclean-kyber1024-90s-avx2	
64-bit ARMv8	pqclean-kyber512-aarch64	64-bit ARMv8 환경에서 NEON 명령어를 이용하여 연산 고속화 적용
	pqclean-kyber768-aarch64	
	pqclean-kyber1024-aarch64	

■ PQClean Crystals-Kyber 프로젝트 소스파일 구성

Crystals-Kyber의 각 프로젝트는 ref 프로젝트를 기본으로 각 환경(CPU, AVX2, aarch64)에서의 최적화를 위한 추가 소스파일로 구성된다. 따라서, 가장 기본이 되는 ref 프로젝트를 구성하는 소스코드 파일과 각 소스코드 파일의 목적을 정리한다. [표 5]는 pqclean-kyber768-ref 프로젝트를 구성하는 소스코드 목록과 각 소스코드 파일의 목적을 설명한다. 또한, Crystals-Kyber 연산에 대한 계층과 연관된 소스코드 파일은 [그림 1]와 같다. 즉, 사용자 레벨에서는 kem.h와 kem.c에서 정의하고 있는 키쌍생성, 암호화, 복호화 함수를 사용하며, 이 함수의 정의를 위하여 그림의 아래쪽에 정의되어있는 기능과 연관된 소스코드 파일을 이용한다.

[표 3] PQCclean Crystals-Kyber ref 소스파일 구성

파일명	목적
api.h	- Crystals-KYBER가 제공하는 상위 수준 키쌍생성, 암호화, 복호화 함수 원형 정의
cbd.h	- 랜덤 바이트 배열로부터 CBD(Centered Binomial Distribution)에 의거하여 다항식으로 변환하는 함수 원형 선언과 정의
cbd.c	
indcpa.h	- IND_CPA 안전성을 제공하는 PKE 키쌍생성, 암호화, 복호화 함수 원형 선언과 정의
indcpa.c	
kem.h	- IND_CCA2를 제공하는 KEM 키쌍생성, 암호화, 복호화 함수 원형 선언과 정의
kem.c	- 내부적으로 indcpa.h/indcpa.c에서 제공하는 키쌍생성, 암호화, 복호화를 이용하여 KEM 함수 구성
ntt.h	- 효율적인 다항식 곱셈 연산을 수행하기 위한 NTT (Number-Theoretic Transform) 기반 다항식 곱셈 함수 원형 선언과 정의
ntt.c	
params.h	- KYBER의 보안 레벨에 따른 파라미터 상수값 정의
poly.h	- KYBER가 기반하고 있는 $R_q = \mathbb{Z}_q[X]/(X^n + 1)$ 상에서의 다항식 연산 함수 원형 선언과 정의
poly.c	- 다항식 덧셈/뺄셈/곱셈/감산 연산, CBD 기반의 잡음원 다항식 생성, 다항식 압축/압축해제, 메시지와 다항식간의 변환
polyvec.h	- KYBER의 Module 연산을 위한 다항식 벡터 연산 함수 원형 선언과 정의
polyvec.c	- 다항식 벡터 덧셈/감산/곱셈 연산, 다항식 벡터 압축/압축해제, 다항식 벡터와 바이트 배열간의 변환
reduce.h	- NTT 기반 다항식 곱셈과 다항식 감산에서 사용되는 감산 알고리즘 (Montgomery, Barret)에 대한 함수 원형 선언과 정의
reduce.c	
symmetric.h	- 90s의 경우 AES기반 PRF, AES기반 XOF 함수 원형 선언 - 90s가 아닐 경우, SHAKE 기반의 PRF, SHAKE 기반 XOF 함수 원형 선언
symmetric-shake.c	- SHAKE128 absorb와 SHAKE256 prf 함수 원형 선언과 정의 - 내부적으로 common 폴더에 정의되어있는 sha3 함수 호출
verify.h	- constant-time으로 동작하는 배열비교, 배열복사 함수 원형 선언과 정의
verify.c	

연산 레벨	연관 파일
IND CCA 기능 (키쌍생성, 암호화, 복호화)	kem.h, kem.c
IND CPA 기능 (키쌍생성, 암호화, 복호화, 행렬생성)	indcpa.h, indcpa.c
벡터 연산 (변환, 사칙연산)	polyvec.h, polyvec.c
다항식 연산 (생성, 샘플링, 변환, 사칙연산)	- poly.h, poly.c - ntt.h, ntt.c - reduce.h, reduce.c
랜덤 데이터 생성 (SHAKE기반, AES 기반)	symmetric.h, symmetric-shake.c

[그림 1] Crystlas-Kyber 소스코드 파일 계층도



제 4 장 Kyber on 8-bit AVR

제 1 절 8-bit AVR ISA

8비트 AVR 마이크로컨트롤러는 메모리와 I/P를 마이크로프로세서에 추가하여 단일 칩 회로로 만든 임베디드 장치이며, 현재 WSN 환경에서 전 세계적으로 가장 많이 사용되고 있는 장치이다. 8비트 AVR 마이크로컨트롤러의 명령은 연산 코드와 피연산자로 구성된다. 8비트 AVR-MCU는 플래시 메모리, SRAM 및 Harvard 아키텍처의 EEPROM으로 구성된다.

■ General-Purpose Register of 8-bit AVR

AVR-MCU는 32개의 8비트 범용 레지스터를 가지고 있으며 기본 연산 및 비트 연산과 같은 다양한 용도로 사용된다. 특히, R26-R31 레지스터는 결합되어 3개의 16비트 형식으로 사용할 수 있다. 이러한 레지스터(X, Y 및 Z)는 데이터 메모리에 대한 16비트 주소를 간접적으로 지정하기 위한 포인터로 사용된다. R2-R17 및 R28, R29 레지스터는 Callee Saved Registers이며, R1은 항상 종료 시 제로화를 해줘야 한다.

■ Instruction of 8-bit AVR

[표 4] 8-bit AVR 환경에서의 주요 명령어

Asm	Operands	Description	Operation	cc
MOV	Rd, Rr	레지스터의 복사	$Rd \leftarrow Rr$	1
MOVW	Rd, Rr	2개의 연결한 레지스터 복사	$Rd+1:Rd \leftarrow Rr+1:Rr$	1
LDI	Rd, K	상수 K 즉시 로드	$Rd \leftarrow K$	1
LD	Rd, X(Y,Z)	간접 참조 로드	$Rd \leftarrow X(Y, Z)$	2
LDD	Rd, Y(Z)+q	offset 기반의 간접 참조 로드	$Rd \leftarrow Y(Z) + q$	2
ST	X(Y,Z), Rd	간접 참조 스토어	$X(Y, Z) \leftarrow Rd$	2
STD	Y(Z)+q, Rd	offset 기반의 간접 참조 스토어	$Y(Z) + q \leftarrow Rd$	2
PUSH	Rr	스택에 레지스터를 푸쉬	$Stack \leftarrow Rr$	2
POP	Rd	스택에서 레지스터로 팝	$Rd \leftarrow Stack$	2
ADD	Rd, Rr	2개의 레지스터의 값을 더함	$Rd \leftarrow Rd + Rr$	1
ADC	Rd, Rr	2개의 레지스터의 값과 캐리를 더함	$Rd \leftarrow Rd + Rr + C$	1
SUB	Rd, Rr	2개의 레지스터의 값을 뺌	$Rd \leftarrow Rd - Rr$	1
SBC	Rd, Rr	2개의 레지스터의 값과 캐리를 뺌	$Rd \leftarrow Rd - Rr - C$	1
ASR	Rd	1-bit 산술 시프트	$Rd(n) \leftarrow Rd(n+1)$	1
MUL	Rd, Rr	부호 없는 곱셈	$R1:R0 \leftarrow Rd \times Rr$	2
MULS	Rd, Rr	부호 있는 곱셈	$R1:R0 \leftarrow Rd \times Rr$	2
MULSU	Rd, Rr	다른 부호 간의 곱셈	$R1:R0 \leftarrow Rd \times Rr$	2
INC	Rd	레지스터 값 1 증가	$Rd \leftarrow Rd + 1$	1
DEC	Rd	레지스터 값 1 감소	$Rd \leftarrow Rd - 1$	1
RJMP	k	상대적 점프	$PC \leftarrow PC+k+1$	2

8-bit AVR은 2단계 파이프라인을 가지고 있기는 하나, 명령어의 처리는 단일로 이루어지기 때문에, ARMv8 환경과 같이 병목현상을 해제하여 성능을 가속하는 최적화 방안은 적용하기 힘들다. 명령어의 수행이 단일로 이루어지므로 clock cycles는 고정된다. 메모리 접근과 곱셈 명령어는 기본적으로 2 clock cycles을 요구하며 그 외의 대부분의 명령어는 1 clock cycles을 요구한다. 상기 표는 본 논문에서 사용하는 명령어 집합에 대한 설명이다.

제 2 절 NTT implementation on 8-bit AVR microcontroller

Crystals-Kyber의 소수 계수는 3329이고, 레퍼런스 구현에서는 다항식의 계수를 16-bit 자료형으로 표현한다. 8-bit AVR 환경에서는 8-bit 단위의 곱셈만 지원되므로, 16-bit간의 곱셈은 새로 정의해야 한다. 하기 [그림 2]은 부호 있는 자료형(Signed)과 부호 없는 자료형(Unsigned) 기반의 16-bit 간의 곱셈에 대한 구현 코드이다. AVR 환경에서는 8-bit 단위의 곱셈 가속기만 지원하므로, 16-bit 단위의 곱셈을 위해 본 연구에서는 직접 구현하여 사용한다.

```

/*****
* macro : mc_mulsu16x16_32
* description : Signed with Unsigned multiply of two 16bits numbers with 32bits result
* usage: int32_t c3|c2|c1|c0 =(int16_t a1|a0) * (int16_t b1|b0)
* register usage : 11 registers
*****/
.macro mc_mulsu16x16_32 a0, a1, b0, b1, c0,
    \c1, \c2, \c3, carry, fixed_r0, fixed_r1
1 : mulsu \a1, \b1
2 : movw \c2, \fixed_r0
3 : mul \a0, \b0
4 : movw \c0, \fixed_r0
5 : mulsu \a1, \b0
6 : sbc \c3, \carry
7 : add \c1, \fixed_r0
8 : adc \c2, \fixed_r1
9 : adc \c3, \carry
10 : mul \b1, \a0
11 : sbc \c3, \carry
12 : add \c1, \fixed_r0
13 : adc \c2, \fixed_r1
14 : adc \c3, \carry
.endm

```

[그림 2] 8-bit AVR 환경에서의 16-bit 기반의 곱셈구현 코드

2.1 Barrett Reduction

Algorithm 13 Signed Barrett Reduction

Input : q with $0 < q < \frac{\beta}{2}$, $2 \nmid q$ and a with $-\frac{\beta}{2} \leq a < \frac{\beta}{2}$
Output: r with $r = a \pmod{q}$, $0 \leq r \leq q$

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

[Algorithm 12] Signed Barrett Reduction^[18]

```

/*****
* macro : mc_barrett_reduce
* description : Barrett reduction; given a 16-bit integer a, computes
*               centered representative congruent to a mod q in {-(q-1)/2,...,(q-1)/2}
* usage: int16_t res1|res0 = barrett_reduce (int16_t a1|a0)
* register usage : 13 registers
* note : Registers which hold value of a0, a1, barret0, barret1,
*        \tmp2, and \tmp3 must be in R16 or higher.
*****/

.macro mc_barrett_reduce a0, a1, tmp0, tmp1, tmp2, tmp3, carry, barret0, barret1,
        \Q0, Q1, fixed_r0, fixed_r1
1 : mc_mulsu16x16_32 \a0, \a1, \barret0, \barret1, \tmp1, \tmp2, \tmp3,
        \carry, \fixed_r0, \fixed_r1
2 : inc \carry
3 : inc \carry
4 : add \tmp3, \carry
5 : clr \carry
6 : asr \tmp3
7 : muls \tmp3, \barret0
8 : movw \tmp0, \fixed_r0
9 : muls \tmp3, \barret1
10 : add \tmp1, \fixed_r0
11 : sub \a0, \tmp0
12 : sbc \a1, \tmp1
.endm

```

[그림 3] 8-bit AVR 환경에서의 Barrett Reduction 구현 코드

현재 표준화 대상 양자내성암호는 주로 다항식 환 R_q 을 기반으로 사용한다

($R_q = \mathbb{Z}_q[x]/(X^n+1)$, $f(x) = x^n + 1$ or $x^n + x^{n-1} + \dots + 1$). 표준화 대상 격자기반 양자

내성암호의 자료형은 Falcon을 제외하고 일반적으로 16-bit와 32-bit를 사용한

다. Crystals-Kyber 경우 홀수 q 와 $f(x) = x^n + 1$ 을 사용한다. q 가 2의 거듭제곱

풀린 경우(ex : SABER) 쉬프트 연산으로 간단히 구현될 수 있으나, q 가 홀수일 경우 C언어에서 % 연산이 들어가야 하므로 연산량이 증가하기 때문에 해당 연산량을 줄이는 것을 목표로 삼아야 한다.

Kyber의 구현 Round 2부터 부호있는 (Signed) 자료형을 사용한다. 부호 없는 자료형(Unsigned)을 사용하면, 내부 연산 시 Unsigned의 값을 뺄 때 결과가 음수가 될 수 있으므로, q 에 적절한 배수를 추가하여 더해줘야 한다. 부호 있는 자료형을 사용하면 해당 연산을 생략할 수 있다. 부호 없는 자료형에서 계수 a 는 0과 $q-1$ 사이로 감산이되며 $\text{mod } q$ 로 표현할 수 있다. 해당 표현을 Canonical Unsigned representative of a 라고 한다. 부호있는 표현에서는 a 를 $(-\frac{q-1}{2}, +\frac{q-1}{2})$ 로 감산하여 범위를 맞추며 $\text{mod } q$ 로 표현한다. 해당 표현을 Canonical Signed representative of a 라 한다.

a 의 절대값이 작은 경우 $a > \frac{q}{2}$ ($a < -\frac{q}{2}$)의 가장 쉬운 감산은 q 를 빼(더하)는 것이다. $|a| < \frac{3q}{2}$ 인 경우 Canonical Signed representative로 쉽게 계산이 가능하나, $|a| > \frac{3q}{2}$ 인 경우는 q 를 빼(더하)는 과정을 여러번 수행해야한다. 따라서, Crystals-Kyber에서는 Barrett(바렛) Reduction을 사용한다.

바렛 감산은 $a \text{ mod } q = a - q \left\lceil \frac{a}{q} \right\rceil$ 에서 감산 시 $\left\lceil \frac{a}{q} \right\rceil$ 을 매번 계산하는 것이 비효율적이므로 $\left\lceil \frac{a}{q} \right\rceil$ 의 근사값을 이용하여 $a \text{ mod } q$ 에 가장 가까운 값을 계산한다.

$\left\lceil \frac{a}{q} \right\rceil$ 값을 $\left\lceil a \left[\frac{R}{q} \right] / R \right\rceil$ 로 근사하므로써 $a \text{ mod } q = a - q \left\lceil \frac{a}{q} \right\rceil \cong a - q \left\lceil a \left[\frac{R}{q} \right] / R \right\rceil$,

where $R = 2^k > q$ 을 만족한다.

$\left\lfloor \frac{R}{q} \right\rfloor$ 값은 사전 계산이 가능하고 $\left\lfloor a \left\lfloor \frac{R}{q} \right\rfloor / R \right\rfloor$ 은 비트 연산으로 계산할 수 있다. 일반

적으로 감산 결과는 $[-\frac{3}{2}q, +\frac{3}{2}q]$ 이내로 보장되나 구체적인 매개변수에 의해 경계를

바꿀 수 있다^[13]. [Algorithm 12]는 부호 있는 바렛감산을 보여주고, [그림 3]은 AVR

환경에서 구현을 보여준다. 속도 및 감산 범위 때문에 일반적으로 덧/뺄셈에서는 바렛

감산, 곱셈에서는 몽고메리 감산을 사용한다.



2.2 Montgomery Reduction

Algorithm 12 Signed Montgomery reduction

Input: q modulus, $R = 2^n > q$

Input: $q^{-1} \bmod^{\pm} R$

Input: $a \in \mathbb{Z}, a < qR$

Output: $t \equiv aR^{-1} \pmod{q}, |t| \leq q$

1: $t \leftarrow a(-q^{-1}) \bmod^{\pm} R$

2: $t \leftarrow (tq)/R$

3: $t \leftarrow \lfloor a/R \rfloor - t$

4: **return** t

[Algorithm 13] Signed Montgomery Reduction^[16]

```

/*****
* macro : mc_montgomery_reduce
* description : Montgomery reduction; given a 32-bit integer a,
*               computes 16-bit integer congruent to a * R^-1 mod q, where R=2^16
* usage: int16_t res1|res0 = motgomery (int32_t a3|a2|a1|a0)
* register usage : 17 registers
* note : Registers which hold value of res0, res1, Q_asm0, and Q_asm1 must be in R16 or higher.
*****/

.macro mc_montgomery_reduce a0, a1, a2, a3, res0, res1, t_Qinv0, t_Qinv1, t_Qinv2,
\t_Qinv3, carry, Qinv_asm0, Qinv_asm1, Q_asm0, Q_asm1, fixed_r0, fixed_r1
1 : mul    \Qinv_asm0, \a0
2 : movw   \res0, \fixed_r0
3 : mul    \a1, \Qinv_asm0
4 : add    \res1, \fixed_r0
5 : mul    \res0, \Qinv_asm1
6 : add    \res1, \fixed_r0

7 : mc_mulsu16x16_32 \res0, \res1, \Q_asm0, \Q_asm1, \t_Qinv0, \t_Qinv1, \t_Qinv2,
\t_Qinv3, \carry, \fixed_r0, \fixed_r1

8 : sub    \a0, \t_Qinv0
9 : sbc    \a1, \t_Qinv1
10 : sbc   \a2, \t_Qinv2
11 : sbc   \a3, \t_Qinv3
12 : movw  \res0, \a2
.endm

```

[그림 4] 8-bit AVR 환경에서의 Montgomery Reduction 구현 코드

바렛감산과 유사하게 몽고메리감산의 기본 개념은 $\bmod q$ 감산에 필요한 나눗셈을

비교적 간단한 연산인 2^k 의 나눗셈으로 바꾸는 것이다.

실제 구현에서 2^k 의 나눗셈은 간단히 쉬프트 연산으로 대체할 수 있다. 즉, $a \bmod^{\pm} q = a - q \cdot [a/q]$ 에서 $[a/q]$ 을 2의 거듭제곱 기반의 나눗셈으로 변환한다. 만약 a 가 $R = 2^k > q$ 의 배수라면 $a/R (a \cdot R^{-1})$ 은 간단한 쉬프트 연산으로 구현할 수 있다. 몽고메리 감산의 과정은 일반적으로 a 의 범위를 탐색 후 $a - t \cdot q (t \in \mathbb{Z})$ 가 R 로 나눌 수 있도록 t 를 선택한다. 이를 정리하면 다음과 같이 표현 가능하다.

$$\begin{aligned} t &= a \cdot q^{-1} \bmod R \\ a - t \cdot q &\rightarrow a - (a \cdot q^{-1}) \cdot q \bmod R = 0 \\ a - t \cdot q / R &\equiv a R^{-1} \pmod{q} \end{aligned}$$

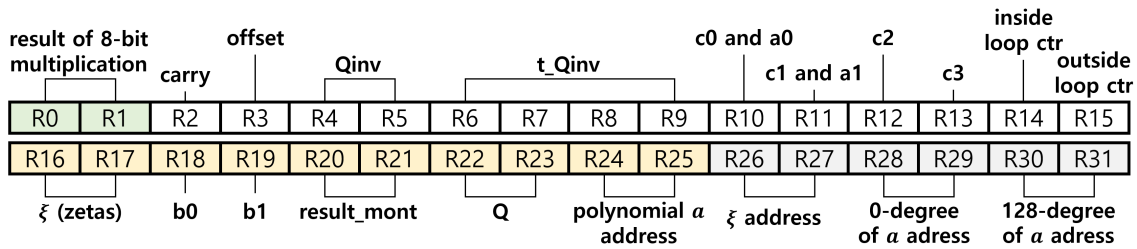
일반적인 몽고메리 감산을 이용한 곱셈은 곱셈의 피연산자 a, b 에 대해 a, b 가 몽고메리 도메인(aR, bR)에 있도록 변환하나, 실제로 NTT 구현은 3에 몽고메리 인자를 포함하여 layer의 연산 중에 몽고메리 인자의 중복이 발생하지 않도록 구성된다.

[Algorithm 13]는 [16]에서 제안된 부호있는 표현에 대한 몽고메리 감산 기법이고, [그림 4]은 몽고메리 인자를 2^{16} 으로 선택한 8-bit AVR 환경에서의 Kyber의 몽고메리 감산 구현이다. 부호있는 표현을 사용하면, 감산과정의 전체를 계산하는 것보다 상위 절반만 계산할 수 있다는 장점이 있다.

2.3 Butterflies

32-bit 기반 아키텍처와 달리, 8-bit AVR 환경은 범용 레지스터의 크기가 8-bit 이므로 바렛 감산에 필요한 상수와 Kyber의 α 및 β 이 2개의 레지스터에 저장되어야 한다 (barret0, barret1, Q0, Q1, Qin0, Qin1). 각 상수는 구현 효율성을 위하여 레지스터에 항상 유지되어야 하는 점을 고려하여 [그림 5]과 같이 레지스터 스케줄링을 하고, [그림 6]과 같이 구현하였다.

CT와 GS 알고리즘은 일반적으로 함수 호출 비용 및 매개변수 관리를 위해 반복문 기반으로 구현한다. 32-bit 기반 프로세서에서의 구현은 NTT의 layer를 merging하트로서 다항식 계수의 메모리 접근 비용을 감소하는 방향으로 지금까지 진행되었다. 그러나, AVR 환경의 경우 NTT 구현에 필요한 16-bit 단위의 필수 상수 및 매개변수(중간 값)들을 저장하기 때문에, 레지스터에 4개 이상의 계수를 유지하기 어렵다. 계수들을 유지하는 대신, 상수들을 로드하는 경우 동일한 메모리 접근 비용이 발생할 뿐만 아니라, 다항식 주소를 저장하는 레지스터에 대한 추가 연산 비용이 발생하여 AVR 환경에서는 1개의 butterfly 단위로 계산하는 것이 효율적이다.



[그림 5] NTT 구현을 위한 AVR 환경에서의 레지스터 스케줄링

```

/*****
* macro : mc_butterfly
* description : Cooley-Tukey butterfly
* usage: (int16_t a0|a1), (int16_t b0|b1) = (int16_t a0|a1 + (b0|b1 * zetas0|zetas1)),
*      (int16_t a0|a1 - (b0|b1 * zetas0|zetas1))
* register usage : 23 registers
*****/
.macro mc_butterfly a0,a1,b0,b1,zetas0,zetas1,c0,c1,c2,c3,carry,fixed_r0,fixed_r1
    res_mont0,res_mont1,tmp0,tmp2,tmp3,Qinv0,Qinv1,Q0,Q1

1 : ldd  \b0, Z+0
2 : ldd  \b1, Z+1

3 : mc_muls16x16_32    \b0,\b1,\zetas0,\zetas1,\c0,\c1,\c2,\c3,\fixed_r0,\fixed_r1
4 : mc_Montgomery_reduce \c0,\c1,\c2,\c3,\res_mont0,\res_mont1,\tmp0,\tmp1,\tmp2,
    \tmp3,\carry,\Qinv0,\Qinv1,\Q0,\Q1,\fixed_r0,\fixed_r1

5 : ldd  \a0, Y+0
6 : ldd  \a1, Y+1
7 : movw \b0, \a0
8 : add  \a0, \res_mont0
9 : adc  \a1, \res_mont1
10 : sub \b0, \res_mont0
11 : sbc \b1, \res_mont1
12 : st  Y+, \a0
13 : st  Y+, \a1
14 : st  Z+, \b0
15 : st  Z+, \b1
.endm

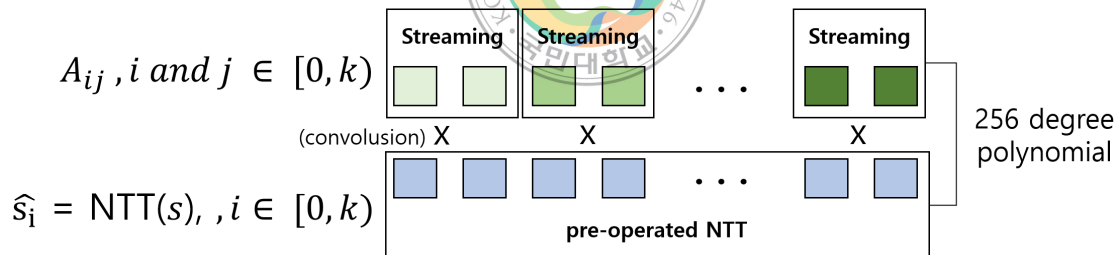
```

[그림 6] 8-bit AVR 환경에서 CT기반 NTT 변환 코드

제 3 절 Stack-friendly Kyber Implementation

3.1 Streaming coefficient strategy

선행 연구로 PQM4에서의 Crystals-Kyber의 구현은 계수 단위로 공개행렬 A 와 에러 e 를 생성하여 $As+e$ 를 계산하였다^[5]. 본 논문에서도 해당 방안과 동일하게 코드를 작성하는 것을 목표로 삼았다. 비밀 벡터 s 의 경우 NTT 변환을 수행해야하기 때문에 스트리밍이 불가능 하다. 하기 [그림 7]와 같이 A 계수를 스트리밍 기반으로 샘플링 할 때 Crystals-Kyber는 InComplete NTT 기반이므로 2개의 계수가 생성되는 시점을 확인하기 위한 분기문이 추가된다. 공개 행렬 A 는 공개정보이므로 constant-time 구현이 아니더라도 보안에 문제되지 않는다.



[그림 7] Streaming public matrix A 의 구현 방안

제 4 절 Speed-friendly Kyber Implementation

4.1 Asymmetric Multiplication

핵심연산인 $A \cdot s + e$ 을 계산할 때 Crystals-Kyber의 비밀벡터 s 의 비밀벡터는 고정되고 반복되어 사용된다. Kyber는 Incomplete NTT를 사용하므로 NTT 도메인 위의 A, s 의 곱셈은 2개 계수 끼리의 school-book 기반의 곱셈을 수행한다. Crystals-Kyber에서 다항식 곱셈을 위해 Incomplete NTT 변환을 수행하면 링

의 분할은 $X^{256} + 1 = \prod_{i=0}^{127} (X^2 - \zeta^{2i+1})$ 과 같다. 처음에 $A_{0j} \cdot s_i$ ($i, j \in [0, k)$) 계산 시

s_i 의 원소와 ζ^{2i+1} 의 곱셈에 대한 몽고메리 감산의 결과를 저장한다면 다른 행에 대한 곱셈 시 일부 계산을 생략할 수 있다[47]. 단, $k \times 16 \times 128/8$ bytes의 추가 스택을 사용해야하는 단점이 있다.

예를 들어 $X^2 - \zeta^{2i+1}$ 상에서의 다항식 곱셈 시 $X^2 = \zeta^{2i+1}$ 이고, $a_0 + a_1x$ 와 $b_0 + b_1x$ 의 곱셈의 결과는 $(a_1b_1 + a_0b_0\zeta^{2i+1}) + (a_1b_0 + a_0b_1)x$ 이다. 행렬곱셈에서 비밀 벡터는 반복적으로 사용되므로 비밀 벡터에 대응되는 $b_0\zeta^{2i+1}$ 을 저장하면 각 계수별 몽고메리 곱셈을 생략할 수 있다.

4.2 Better Accumulation

$A \cdot s + e$ 을 계산 결과 또한 R_{3329} 의 원소여야하기 때문에 $A \cdot s$ 연산 시 각 다항식의 계수 곱셈은 레퍼런스 구현의 경우 몽고메리 감산을 통해 16-bit로 감산 및 누적 후 마지막에 바렛 감산을 수행한다. $As + e$ 에서 누적 매개변수를 32-bit 다

항식을 사용하고 마지막 다항식 곱셈에서만 몽고메리 감산을 수행하면 각 계수별 곱셈시 몽고메리 감산을 사용하지 않아 성능을 가속화 할 수 있다. Kyber의 q 는 12-bit이기 때문에 곱셈 시 최대 24-bit이고 k 번 누적시 최대 29-bit 이므로 32-bit 자료형에 대해 오버플로가 발생하지 않는다[18]. 단 누적 다항식을 사용하기 위해서는 $32 \times 256/8$ bytes의 추가 스택을 사용해야한다. Asymmetric multiplication과 달리 보안레벨과 상관없이 32-bit 자료형의 크기를 갖는 다항식을 저장한다.

4.3 Pre-hashed public key

마이크로컨트롤러에서는 특수한 경우를 제외하고, 일반적으로 세션마다 키를 생성하는 경우는 드물다. 또한, KEM의 목적이 세션에 사용할 키를 생성하는 것이므로, KEM의 공개 및 개인키의 수명은 세션 키의 수명보다 길다. 따라서, KEM의 암호화 과정에 존재하는 공개키를 해시하는 연산(sha3_256)에 대한 사전 계산이 가능하다. 테이블의 크기는 32 bytes 이며, 키를 재생성하지 않는 이상 상수값이므로 Flash memory에도 저장가능하다.

4.4 keccak implementation with XKCP library

Kyber의 프로파일링 결과 공개행렬 생성 및 KEM 구조에서 keccak 알고리즘이 높은 빈도로 호출되므로, 8-bit AVR 환경에서 assembly로 최적화된 keccak 구현을 사용하는 것이 바람직하다. 본 논문에서는 범용성을 위해 keccak 개발팀이 제공하는 XKCP의 keccak AVR 구현물을 사용한다.

제 5 절 Evaluation

5.1 Setup 8-bit AVR board

Crystals-Kyber의 공개/개인 키 및 암호문의 크기는 보안레벨에 따라 다르므로 이를 고려하여 AVR 보드를 선택하는 것이 필요하다. [표 5]은 Crystals-Kyber의 보안레벨별 공개키, 개인키, 암호문의 크기를 보여준다.

[표 5] Crystals-Kyber 보안레벨별 매개변수 크기

보안레벨/파라미터	pk(B)	sk(B)	ct(B)
kyber512	800	1632	768
kyber768	1184	2400	1088
kyber1024	1568	3168	1568

본 논문에서는 모든 보안 레벨에서의 성능 측정을 수행하기 위해 16,384 bytes 의 SRAM 크기를 갖는 ATmega1284p를 선택하고, Microchip Studio를 통해 시뮬레이션을 수행한다. gcc 옵션은 O3 및 Os를 사용하였으며 비교 구현물 PQM4의 구현은 ARM에 대응되는 assembly 코드들을 레퍼런스 코드로 치환하였다. PQM4 코드와 레퍼런스 코드 또한 O3로 컴파일하였다. 운영체제의 도움을 받아야하는 엔트로피는 추출되었다고 가정한다.

5.2 Result of Implementation

■ NTT Implementation

[표 6] 8-bit AVR 환경에서 NTT/FFT 구현 결과 비교
(K = 1,000 clock cycle)

NTT/FFT software	clock cycles	
	128-bit	256-bit
This work	67K	67K
	119K	119K
Refernce ^[19]	126K	126K
	365K	365K
Boorghany et al. ^[20]	1,216K	-
Boorghany et al. ^[21]	754K	2,207K
Pöppelemann et al. ^[22]	334K	855K
Liu et al. ^[23]	193K	441K
Liu et al. ^[24]	194K	516K
Seo et al. ^[25]	158K	403K
Seo et al. ^[26]	144K	344K

[표 6]는 8-bit AVR 환경에서 NTT 구현 결과를 보여준다. 본 연구의 구현은 레퍼런스 구현에 비해 89%(NTT), 206%(InvNTT)의 성능향상을 달성하였다. 기존의 AVR 환경에서 Ring-LWE의 연구들과 비교했을 때, Module-LWE 구조 기반의 Crystals-Kyber는 보안 레벨에 따라 동일한 클럭주기를 보여주고 타 구현물과 비교했을 때 높은 성능을 보여준다.

■ NTT Implementation

본 논문에서 제공하는 구현은 **stack** 버전과 **speed** 버전 두가지로 나뉜다. 각 버전에 적용된 최적화 방안은 [표 7]와 같다.

[표 7] 최종 결과물에 적용한 방법론 정리

software	methodology
Speed	♦ Streaming A and e ♦ Asymmetric multiplication ♦ Better accumulation ♦ Pre-hashed public key ♦ Using XKCP library ♦ Signed barrett/montgomery reduction ♦ hand-written assembly NTT/InvNTT
Stack	♦ Streaming A and e ♦ Signed barrett/montgomery reduction ♦ Using XKCP library ♦ hand-written assembly NTT/InvNTT

[표 8] 8-bit AVR 환경에서 Crystals-Kyber의 구현 결과 비교
(cc : clock cycles, K : 1,000 cc)

software	variant	avr-gcc	Kyber-512		Kyber-768		Kyber-1024	
			cc	stack[B]	cc	stack[B]	cc	stack[B]
This work	speed	O3	K 5,775K	3,756	9,259K	4,796	14,448K	4,372
			E 7,000K	4,890	10,483K	5,940	15,920K	6,946
			D 7,567K	4,866	11,277K	5,914	16,982K	6,930
	stack	Os	K 6,059K	2,220	10,100K	2,736	16,081K	3,256
			E 8,354K	2,308	12,911K	2,808	19,684K	3,328
			D 8,117K	2,324	12,550K	2,824	19,233K	3,352
PQM4 ^[5]	stack	O3	K 9,541K	2,220	15,554K	2,736	24,433K	3,256
			E 12,879K	2,308	19,500K	2,808	29,347K	3,328
			D 12,484K	2,324	18,796K	2,824	28,377K	3,352
Reference ^[19]	-	O3	K 9,588K	6,020	15,581K	10,052	12,500K	15,180
			E 12,890K	8,668	19,530K	13,212	18,849K	18,844
			D 12,500K	9,444	29,423K	14,308	28,417K	20,420

[표 8]은 본 논문에서 제안하는 Crystals-Kyber의 전체 스킴의 구현 결과를 보여준다. `stack` 버전의 경우 레퍼런스 구현에 비해 Kyber512의 경우 KeyGen/Encapsulation/Decapsulation 과정에서 57/54/52 %의 성능향상을 달성했고, Kyber768의 경우 54/51/49%의 성능향상을 달성했다. Kyber1024에서는 54/50/48%의 성능향상을 달성했다.

`speed` 버전의 경우 레퍼런스 구현에 비해 Kyber512의 경우 KeyGen/Encapsulation/Decapsulation 과정에서 66/68/69%의 성능향상을 달성했고, Kyber768의 경우 84/86/84%의 성능향상을 달성했다. Kyber1024에서는 65/67/67%의 성능향상을 달성했다.

제 6 절 Conclusion



본 논문에서는 표준화 대상 양자내성암호 중 PKE/KEM 알고리즘인 Crystals-Kyber에 대한 8-bit AVR 환경에서의 최적화 방법론과 첫 번째 구현을 제안한다. 기존에 임베디드 환경에서 제안된 속도 및 메모리 사용량 관점의 모든 구현기법을 적용하였다. 또한, 8-bit AVR 환경에서의 NTT 구현 방법론을 제안하여 Crystals-Kyber의 성능을 가속화 하였다. 제안한 구현기법을 통해 NTT 및 InvNTT에서 레퍼런스 구현에 비해 89%, 206%의 성능향상을 달성하였으며, Crystals-Kyber의 전체 스킴에 대해 레퍼런스 구현보다 최대 86%의 성능향상을 달성하였다.

Abstract

Efficient Implementation of Post-Quantum Cryptography Crystals-Kyber on 8-bit AVR microcontroller

Department of Financial Information Security, Kookmin University

YoungBeom Kim

Post-Quantum Cryptography generally requires lower performance and higher memory compared to elliptic curve-based cryptographic schemes. Therefore, optimization research has been actively conducted in a 32-bit and 64-bit ARM-based devices, starting with PQM4. The 8-bit AVR environment has the most constrained performance and memory among embedded devices, and research on the implementation of PQC has not been conducted yet. In order to achieve quantum security in all devices in the future, it is necessary to install PQC even in the 8-bit AVR environment. Therefore, in this paper, we present the first optimization implementation of Crystals-Kyber and contribute to the performance evaluation of NIST PQC project.

- Keywords

Post-Quantum Cryptography, Crystals-Kyber, 8-bit AVR, Efficient Implementation

Reference

- [1] Shor, P. W. (1994, November). Algorithms for quantum computation: discrete logarithms and factoring. In Proceedings 35th annual symposium on foundations of computer science (pp. 124–134). Ieee.
- [2] Grover, L. K. (1996, July). A fast quantum mechanical algorithm for database search. In Proceedings of the twenty-eighth annual ACM symposium on Theory of computing (pp. 212–219).
- [3] Bos, J. W., Renes, J., & Sprenkels, A. (2022). Dilithium for memory constrained devices. In International Conference on Cryptology in Africa (pp. 217–235). Springer, Cham.
- [4] Chung, C. M. M., Hwang, V., Kannwischer, M. J., Seiler, G., Shih, C. J., & Yang, B. Y. (2021). NTT multiplication for NTT-unfriendly rings: New speed records for Saber and NTRU on Cortex-M4 and AVX2. IACR Transactions on Cryptographic Hardware and Embedded Systems, 159–188.
- [5] Kannwischer, M. J., Rijneveld, J., Schwabe, P., & Stoffelen, K. (2019). pqm4: Testing and Benchmarking NIST PQC on ARM Cortex-M4.
- [6] Greconici, D. O., Kannwischer, M. J., & Sprenkels, D. (2021). Compact dilithium implementations on Cortex-M3 and Cortex-M4. IACR Transactions on Cryptographic Hardware and Embedded Systems, 1–24.
- [7] Kim, Y., & Seo, S. C. (2021). Efficient Implementation of AES and CTR_DRBG on 8-bit AVR-based Sensor Nodes. IEEE Access, 9, 30496–30510.
- [8] Regev, Oded. "On lattices, learning with errors, random linear codes, and cryptography." Journal of the ACM (JACM) 56.6 (2009): 1–40.

- [9] Lyubashevsky, V., Peikert, C., & Regev, O. (2010, May). On ideal lattices and learning with errors over rings. In Annual international conference on the theory and applications of cryptographic techniques (pp. 1-23). Springer, Berlin, Heidelberg.
- [10] Greconici, D. (2020). Kyber on RISC-V (Doctoral dissertation, Master's Thesis).
- [11] Langlois, Adeline, and Damien Stehlé. "Worst-case to average-case reductions for module lattices." *Designs, Codes and Cryptography* 75.3 (2015): 565-599.
- [12] Howe, J., Khalid, A., Rafferty, C., Regazzoni, F., & O'Neill, M. (2016). On practical discrete Gaussian samplers for lattice-based cryptography. *IEEE Transactions on Computers*, 67(3), 322-334.
- [13] Kannwischer, M. J. (2022). Polynomial Multiplication for Post-Quantum Cryptography (Doctoral dissertation, [S1]:[Sn]).
- [14] Fujisaki, Eiichiro, and Tatsuaki Okamoto. "Secure integration of asymmetric and symmetric encryption schemes." Annual international cryptology conference. Springer, Berlin, Heidelberg, 1999.
- [15] Bos, J., Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schanck, J. M., ... & Stehlé, D. (2018, April). CRYSTALS-Kyber: a CCA-secure module-lattice-based KEM. In 2018 IEEE European Symposium on Security and Privacy (EuroS&P) (pp. 353-367). IEEE.
- [16] Seiler, Gregor. "Faster AVX2 optimized NTT multiplication for Ring-LWE lattice cryptography." *Cryptology ePrint Archive* (2018).
- [17] Becker, H., Hwang, V., Kannwischer, M. J., Yang, B. Y., & Yang, S. Y.

(2021). Neon NTT: faster dilithium, kyber, and saber on cortex-a72 and apple M1. Cryptology ePrint Archive.

[18] Abdulrahman, A., Hwang, V., Kannwischer, M. J., & Sprenkels, D. (2022). Faster Kyber and Dilithium on the Cortex-M4. Cryptology ePrint Archive.

[19] NIST Post-Quantum Cryptography Project Selected Algorithms 2022, Crystals-Kyber, 2022-11-02, <https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022>.

[20] Boorghany, A.; Jalili, R. Implementation and comparison of lattice-based identification protocols on smart cards and microcontrollers. Iacr Cryptol. Eprint Arch. 2014, 2014, 78.

[21] Boorghany, A.; Sarmadi, S.B.; Jalili, R. On constrained implementation of lattice-based cryptographic primitives and schemes on smart cards. Acm Trans. Embed. Comput. Syst. (TECS) 2015, 14, 1 - 25.

[22] Pöppelmann, T.; Oder, T.; Güneysu, T. High-performance ideal lattice-based cryptography on 8-bit ATxmega microcontrollers. In Proceedings of the International Conference on Cryptology and Information Security in Latin America, Guadalajara, Mexico, 23 - 26 August 2015; pp. 346 - 365.

[23] Liu, Z.; Seo, H.; Roy, S.S.; Großschädl, J.; Kim, H.; Verbauwhede, I. Efficient Ring-LWE encryption on 8-bit AVR processors. In Proceedings of the International Workshop on Cryptographic Hardware and Embedded Systems, Saint Malo, France, 13 - 16 September 2015; pp. 663 - 682.

[24] Liu, Z.; Pöppelmann, T.; Oder, T.; Seo, H.; Roy, S.S.; Güneysu, T.; Großschädl, J.; Kim, H.; Verbauwhede, I. High-performance ideal lattice-based cryptography on 8-bit AVR microcontrollers. Acm Trans. Embed. Comput. Syst. (TECS) 2017, 16, 1 - 24.

[25] Seo, H.; Liu, Z.; Park, T.; Kwon, H.; Lee, S.; Kim, H. Secure number theoretic transform and speed record for Ring-LWE encryption on embedded processors. In Proceedings of the International Conference on Information Security and Cryptology, Seoul, Korea, 29 November - 1 December 2017; pp. 175 - 188.

[26] Seo, Hwajeong, et al. "Fast number theoretic transform for ring-lwe on 8-bit avr embedded processor." Sensors 20.7 (2020): 2039.

