정 보 보 호 학 회 지
제34권 제2호, 2024. 4

저사양 장치에서의 격자 기반 양자내성암호 구현

신 동 현*, 김 영 범*, 서 석 충**

요 약

양자내성암호는 기존 공개키 암호 스킴과 비교하여 일반적으로 많은 연산량과 높은 메모리 사용량을 요구한다. 양자내성암호 마이그레이션을 위한 최적화 연구들이 활발하게 이루어지고 있다. PQM4 프로젝트를 시작으로 하여, ARM 환경에서 다양한 연구 결과들이 존재하지만, 저사양 장치에서의 연구 결과는 미진하다. 따라서, 본 논문에서는 현재까지 격자 기반 암호의 최적화 연구를 리소스 자원이 제한된 저사양 장치 관점으로 살펴본다.

Ⅰ. 서 론

현재의 공개키 시스템은 이산 대수 난제와 소인수 분해 난제를 기반으로 설계되었다. 1994년 Shor 알고리즘이 발표되고, 시간이 지남에 따라 지속적으로 양자 컴퓨터가 발전하면서 Shor 알고리즘의 실현 가능성이 높아지고 있다. 이에 따라 NIST(National Institute of Standards and Technology)는 양자 컴퓨팅 환경에서 안전한 양자내성암호 공모전을 개최하였다.

NIST 양자내성암호 공모전은 2016년을 시작으로 Crystals-Kyber, Crystals-Dilithium, Falcon, SPHINCS+ 총 4개의 알고리즘이 표준화 대상으로 선정되었다. 표준화 대상 알고리즘 4개 중 3개의 알고리즘이 격자 기반 암호이기 때문에 다양한 난제를 위해 추가적인 공모를 진행 중에 있다. 국내에서도 양자 보안 달성을 위해 KPQC 공모전을 개최하였다. 현재 Round 2를 진행 중이며, 총 8개의 알고리즘이 선정되었다.

양자 보안을 달성하기 위해 알고리즘 선정뿐만 아니라, 양자내성암호를 기존 공개키 시스템에 마이그레이션하기 위한 노력도 활발히 이어지고 있다. 양자내성암호는 기존 공개키 암호에 비해 많은 연산량과 높은 메모리 사용량을 필요로 하기 때문에 실제 응용 프로토콜(TLS/SSL 등)에서 사용하기 위해 최적화에 관한 연구도 활발하게 이루어지고 있다.

대부분의 스킴들이 타원곡선 기반의 알고리즘과 비교하여 낮은 성능과 높은 메모리를 요구한다. 따라서, 양자내성암호에 대한 최적화 연구는 주로 성능과 메모리가 제한된 임베디드 환경에서 이루어졌다. 현재 임베디드 기기 중 가장 대중적으로 사용되는 장비는 ARM 기반의 장치이다. 따라서, PQM4 프로젝트[1]를 시작으로 대부분의 연구는 주로 Cortex-M과 Cortex-A 시리즈에서 진행되었다.

AVR이나 MSP430과 같은 저사양 장치들도 저전력, 저가의 특징을 지니고 있어, IoT 환경에서 많이 사용되고 있지만, 최적화 연구는 미진하다. 향후 모든 기기에서 양자 보안을 달성하기 위해선 저사양 장치에서도 최적화 연구가 고려되어야 하며, 저전력 장치는 다른 환경에 비해 리소스 제약이 크기 때문에 최적화 연구는 필요하다.

본 논문에서는 현재까지 알려진 격자 기반 암호의 최적화 연구를 조사하고, 저사양 장치의 관점으로 소개한다.

Ⅱ. 배 경

2.1. Crystals-Kyber

Kyber는 Module-LWE 기반 양자내성암호 알고리

이 성과는 2024년도 정부(과학기술정보통신부)의 재원으로 정보통신기획평가원의 지원을 받아 수행된 연구임(No. 2022-00207416, 안전한 차세대 IoT 통신 환경 구축을 위한 양자내성암호 최적화 및 보안 프로토콜 적용 연구)
* 국민대학교 금융정보보안학과 (대학원생, dkldkl13@kookmin.ac.kr, darania@kookmin.ac.kr)
** 국민대학교 금융정보보안학과, 교신저자 (부교수, scseo@kookmin.ac.kr)

즘이다. IND-CPA보안을 만족하는 PKE(Public KeyEncryption) 알고리즘이 존재하며, Fujisaki Okamoto 변환을 통해 IND-CCA2 보안을 만족하는 KEM을 구성한다[2].

다항식 환 $\mathcal{R}_q = \mathbb{Z}_q[X]/(X^{256}+1)$에서 연산이 진행되며, $q = 3329$이므로, 주로 16-bit 자료형을 사용한다. Kyber의 공개키/개인키 쌍은 Number Theoretic Transfrom (NTT) 도메인 값이므로 NTT 운영은 필수적이며, 주 연산 부하 지점으로 알려져 있다. 현재 NIST에서 ML-KEM 라는 이름으로 FIPS 203 문서를 공개하였다[3].

2.2. Crystals-Dilithium

Dilithium은 Module-LWE 기반 전자서명 알고리즘이다[4]. Dilithium은 크게 키 생성 과정, 서명 생성 과정, 서명 검증 과정으로 이루어진다. 다항식 환 $\mathcal{R}_q = \mathbb{Z}_q[X]/(X^{256}+1)$에서 연산이 진행되며, $q = 8380417$이므로, 주로 32-bit 자료형을 사용한다. Kyber와 달리 주요 요소들이 정수도메인 값으로 표현되기 때문에 NTT가 필수 운영은 아니지만, 효율성을 위해 NTT 사용을 권고하고 있다. Kyber와 마찬가지로 NTT가 주요 연산 부하 지점이며, 현재 NIST에서 ML-DSA 라는 이름으로 FIPS 204 문서를 공개하였다[5].

2.3. Number Theoretic Transfrom

NTT는 다항식 곱셈 알고리즘으로 $O(nlogn)$ 복잡도를 갖는다. 두 다항식 $f, g \in \mathcal{R}_q$를 예시로, $\hat{f} = \mathrm{NTT}(f)$, $\hat{g} = \mathrm{NTT}(g)$와 같이 두 다항식을 각각 여러 개의 다항식으로 분할한다. 이후 두 다항식은 점별 곱셈을 수행하며 NTT 역변환을 통해 계산되며, 전체 곱셈 과정은 아래와 같다.

$$\mathrm{NTT}^{-1}(\mathrm{NTT}(f) \circ \mathrm{NTT}(g))$$

일반적으로 NTT/NTT^{-1}은 Cooley-Tukey[6] (CT)/Gantleman-Sande[7] (GS) butterfly를 사용하여 변환 및 역 변환을 수행한다.

2.4. 저사양 장치 (AVR/MSP430)

AVR/MSP430과 같은 저사양 장치들은 저렴하고 저전력에서 동작하기 때문에 다양한 IoT 산업에서 활용되고 있지만, 사용 가능한 리소스가 작다는 제한 사항이 있다. 주요 제한 사항은 RAM 용량, ROM/Flash memory, 동작 주파수이다.

AVR은 Atmel사에서 제조하는 저사양 MCU이며 8-bit 아키텍처이다. 범용 레지스터는 8-bit 크기를 가지며, R0~R31로 총 32개를 갖는다. ATtiny85처럼 극저사양 제품의 경우, RAM 512B, Flash memory 8KB이며, 10MHz 이하로 동작한다. ATmega2560 같은 고사양 제품의 경우, RAM 8KB, Flash memory 256KB이며 16MHz로 동작한다.

MSP430은 Texas Instruments사에서 제조하는 저사양 MCU이며, 16-bit 아키텍처이다. 범용 레지스터 크기는 16-bit이며, R4~R15 총 12개를 갖는다. 평균적으로 RAM은 16~128KB, ROM은 128KB~1MB이며, 1~25MHz로 동작한다.

Ⅲ. 격자 기반 암호 최적화 방안 소개

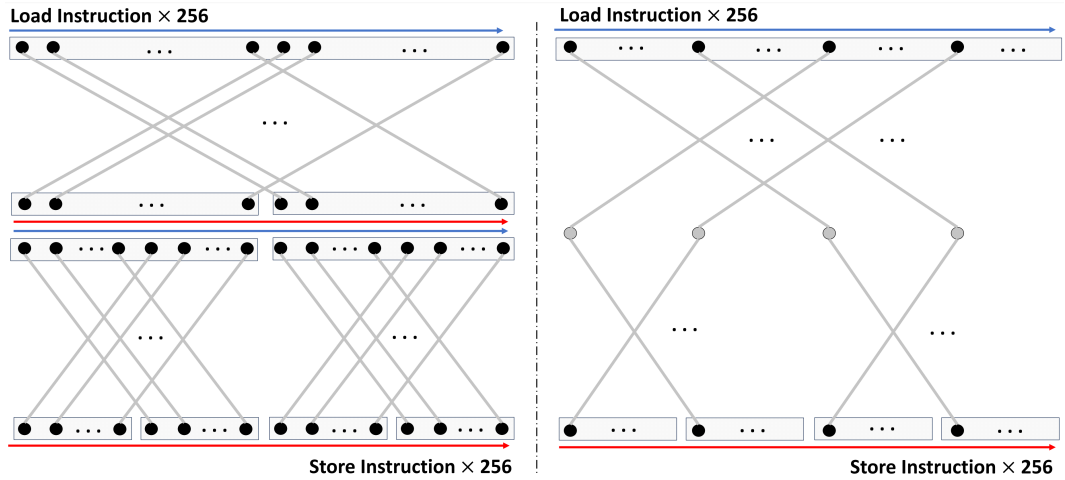
본 장에서는 격자 기반 암호에 대한 최적화 방안들을 소개한다. 3.1절에서는 속도 최적화 관점에서 격자 기반 암호의 주요 연산 부하 지점인 다항식 곱셈과 Hashing에 관련된 기법들을 소개한다. 3.2절에서는 메모리 사용량 관점에서의 최적화 방안들을 소개한다.

3.1. 속도 최적화 구현 방안

3.1.1. NTT 레이어 병합 구현

격자 기반 암호는 주로 다항식 곱셈에서 많은 연산 부하가 발생하며, NTT 알고리즘을 주로 사용한다. 일반적으로 NTT는 여러 개의 레이어로 구성된다. 고급 언어 레벨에서 NTT를 구현하는 방식은 단일 레이어 단위로 계산한다. 즉, [그림 1]의 왼쪽과 같이 2개의 계수를 메모리에서 load 한 뒤, butterfly를 수행하고 store한다. 이렇게 모든 계수에 대해 128번의 butterfly를 수행하면 한 개의 레이어가 종료되며, 다음 레이어를 계산한다.

위와 같은 방식은 각 레이어마다 모든 계수들에



〔그림 1〕(좌) 고급언어 레벨에서의 NTT 구현 (우) 어셈블리언어 레벨에서 NTT 두 개 레이어 병합 구현

대한 load와 store가 발생한다. load와 store를 줄이기 위한 기법으로 어셈블리 레벨에서 레이어 병합 구현이 잘 알려져 있다. 서로 연관 있는 계수들에 대해 여러 개의 레이어를 한번에 계산하여 계수에 대한 메모리 접근 횟수를 최소화하는 기법이다. n개의 레이어를 병합하기 위해서는 n개의 레이어에서 서로 연관있는 2^n개의 계수들을 레지스터에 저장한 뒤 n개의 레이어를 모두 계산하고 store 한다. [그림 1]의 오른쪽은 두 개 레이어를 병합하여 구현하는 방안의 예시이다.

Donghyun, Shin, et al.[8]는 MSP430 환경에서 Kyber에 대해 레지스터 크기와 개수를 고려하여, 6-4, 3-2, 1-0 레이어의 병합 구현을 제안하였으며, NTT/NTT^{-1} 각각 113%/131% 성능 향상을 달성하였다. 또한 Dilithium에 대해 7-6, 5-4, 3-2, 1-0 레이어의 병합 구현을 제안하였으며 NTT/NTT^{-1} 각각 77%/84% 성능 향상을 달성하였다[9].

3.1.2. Small NTT and FNT

Dilithium의 서명 생성 과정에서 c와 $\boldsymbol{s_1}, \boldsymbol{s_2}$를 생성한다. 이때 다항식 c의 원소들은 τ개의 ± 1과 256-τ개의 0으로 구성된다. 또한 $\boldsymbol{s_1}, \boldsymbol{s_2}$를 구성하는 원소들은 [-$\eta$,$\eta$] 범위를 갖기 때문에 $c\boldsymbol{s_1}$과 $c\boldsymbol{s_2}$의 최대 범위는 $2\tau\eta$이다. Dilithium은 $R_{8380417}$에서 계산되지만, $c\boldsymbol{s_1}$과 $c\boldsymbol{s_2}$를 계산할 때는 최대 범위보다 큰 $q'(q' > 2\tau\eta)$을 선택하여 $R_{q'}$에서 계산해도 무방하다.

$F_n = 2^{2^n}+1$꼴인 페르마 소수를 q'으로 선택하여 NTT를 계산하는 방안을 Fermat number transform(FNT)라고 부르며, Abdulrahman, Amin, et al.[10]은 Dilithium2, Dilithium5의 경우 $q' = F_3 = 257$, Dilithium3의 경우 $q' = 769$로 사용하여 구현하는 방안을 제안하였다. q'을 F_n으로 선택하면 초기 레이어에 대한 twiddle factor가 2의 지수 승 형태로 계산되기 때문에, butterfly 연산 시 발생하는 곱셈을 비트 시프트 연산으로 대체할 수 있다. 추가로, Kyber와 Dilithium을 같이 사용하는 어플리케이션의 경우, q'을 Kyber와 동일한 3329로 선택하여 코드를 공유하는 방안도 제안하였다.

AVR/MSP430과 같이 저사양 환경에서는 곱셈 크기가 커질수록 부하가 크게 상승한다. 예를 들어, AVR은 8x8 곱셈인 **mul** 명령어만 지원한다. 따라서 16x16 곱셈 또는 32x32 곱셈을 구현하기 위해서는 **mul**, **add**, **mov** 명령어들을 조합하여 계산해야 하기 때문에 FNT를 사용하는 것을 고려해야 한다.

3.1.3. Hashing

Kyber의 성능에 기여하는 또 다른 중요 요소는 SHA-3의 코어 함수인 Keccak이다[2]. Cortex-M4에서의 Kyber/Dilithium 벤치마킹 결과, SHA-3와 SHAKE가 가장 많은 비중을 차지하였으며 약 60~70% 정도로 나타났다[1]. 장치마다 명령어 집합

이나 내부 로직이 다르기 때문에 결과의 차이가 있을 수 있다.

Kyber/Dilithium에서의 SHA-3와 SHAKE는 공통적으로 Keccak-f1600을 코어 함수로 사용하기 때문에 Keccak-f1600에 대한 최적화 연구가 이어지고 있다. Keccak-f1600의 연산은 선형적이기 때문에, 임베디드 장치에서는 주로 메모리 접근을 감소시키는 연구가 이루어졌다. 섬세한 레지스터 관리를 통해 Keccak의 각 process를 병합하는 것으로 메모리 접근 부하를 감소시켰으며, AVR/MSP430 환경에서 각각 약 26%/15% 성능 향상을 달성하였다[11,12].

3.1.4. Constant Reduction

격자 기반 암호의 주 연산은 다항식 환에서 계산하므로 연산 도중 감산이 필요하다. 격자 기반 암호에서는 shift 연산을 활용하여 상수 시간 내에 효율적으로 감산하는 기법들을 사용한다. mod q 계산 시에는 Barrett reduction을 주로 사용하며[13], 곱셈 후 감산에서는 Montgomery reducton을 주로 사용한다[14].

CPU, Cortex-M4와 같은 아키텍처들은 베럴시프터가 존재하기 때문에 다중 shift 연산을 하나의 명령어로 처리한다. 또한 Becker, Hanno, et al.[15]은 Armv8 환경에서 Neon 명령어를 사용하여, 다중 계수에 대한 병렬 감산 방안도 제안하였다. 하지만 AVR/MSP430과 같은 저사양 장치는 베럴시프터가 존재하지 않을 뿐만 아니라, 여러 개의 레지스터에 피연산자를 나누어 저장하기 때문에 shift 명령어를 여러 번 사용하여 구현한다.

Montgomery reduction과 Barrett reduction을 저사양 장치인 AVR/MSP430 환경에서 구현하는 경우, 레지스터 크기가 각각 8-bit, 16-bit이므로 8의 배수, 16의 배수에 해당하는 shift 연산을 생략하여 효율적인 구현이 가능하다. 예를 들어, PQCLEAN 프로젝트[16]의 Kyber 구현물의 Barrett reduction은 32-bit 값에 대해 26번의 right shift 연산을 수행하고, 다음 연산에는 하위 16-bit만을 사용한다. 이를 MSP430 환경에서 구현하는 경우에는 32-bit 값이 두 개의 레지스터에 저장되어 있기 때문에, 상위 16-bit가 저장되어 있는 레지스터만 10번의 right shift 연산을 수행하는 것으로 동일한 계산이 가능하다. 따라서 저사양 장치에서는 레지스터 크기에 따라 shift 명령어를 생략하는 것을 고려할 수 있다.

3.1.5. Asymmetric multiplication and Better accumulation

핵심연산인 $\boldsymbol{A}\cdot\boldsymbol{s}+\boldsymbol{e}$을 계산할 때 Kyber의 비밀 벡터 $\boldsymbol{s}$는 고정되고 반복되어 사용된다. Kyber는 불완전한 NTT를 사용하므로, NTT 변환 이후 점별 곱셈은 1차 다항식끼리의 곱셈을 수행한다. Kyber에서 불완전한 NTT를 수행하면 링의 분할은 $X^{256}+1=\prod_{i=0}^{127}(X^2-\zeta^{2i+1})$과 같다. $\boldsymbol{A}$의 첫 번째 행에 대한 $A_{0j}\cdot s_i\ (i,j\in[0,k))$계산 시 s_i의 원소와 ζ^{2i+1}의 곱셈 후 몽고메리 감산을 수행한 결과를 저장한다면, 다른 행에 대한 곱셈 시 ζ^{2i+1}에 대한 곱셈 수행을 생략할 수 있다. 단, $k\times(16/8)\times128$ bytes를 갖는 추가 스택을 사용해야 하는 단점이 있다.

또한, 연산 중 $\boldsymbol{A}\cdot\boldsymbol{s}+\boldsymbol{e}$의 결과는 $\mathcal{R}_{3329}$원소여야 하므로, 각 다항식의 계수 곱셈은 레퍼런스 구현에서는 Montgomery reduction을 통해 16-bit로 감산 및 누적한다, 최종적으로 연산이 종료된 후, 모든 계수에 대해 Barrett reduction을 수행한다. 이 때 32-bit 다항식을 사용하고, Montgomery reduction을 생략한 채 누적한 뒤 최종 다항식 곱셈에서 Montgomery reduction을 수행하여, 각 계수별 곱셈 시 Montgomery reduction을 사용하지 않아 성능을 더욱 가속화할 수 있다. Kyber의 경우 q가 12-bit이므로 누적 시에는 오버플로우가 발생하지 않는다. 단, $(32/8)\times256$ bytes의 추가 스택을 사용해야 한다. 김영범[17] 등은 해당 기법들을 AVR 환경에 적용하여, 최대 86% 성능 향상을 달성하였다.

3.2. 메모리 최적화 구현 방안

3.2.1. 행렬 곱셈 스트리밍 구현

Kyber/Dilithium에서 가장 많은 스택을 사용하는 요소는 공개행렬 $\boldsymbol{A}$, 비밀 벡터 $\boldsymbol{s}$와 에러 벡터 $\boldsymbol{e}$이다. $\boldsymbol{A}$, $\boldsymbol{s}$, $\boldsymbol{e}$는 Shake와 sampling을 통해 생성하며, 주 연산은 $\hat{\mathbf{t}} = \hat{\boldsymbol{A}}\cdot\hat{\boldsymbol{s}} + \hat{\boldsymbol{e}}$이다. Reference 구현의 경우, [Algorithm 1]과 같이 $\boldsymbol{A}$, $\boldsymbol{s}$, $\boldsymbol{e}$를 모두 생성한 뒤 계산한다. 해당 구현은 $\boldsymbol{A}$, $\boldsymbol{s}$, $\boldsymbol{e}$의 크기에 대한 메모리를 모두 사용해야 하기 때문에 메모리 공간의 제약

〔표 1〕 Kyber/Dilithium의 주요 컴포넌트 스택 사용량

알고리즘	$\boldsymbol{A}$ (Bytes/크기)	$\boldsymbol{s}$ (Bytes/크기)	$\boldsymbol{e}$ (Bytes/크기)
Kyber512	2,048/ 2×2	1,024/ 2×1	1,024/ 2×1
Kyber768	4,608/ 3×3	1,536/ 3×1	1,536/ 3×1
Kyber1024	8,192/ 4×4	2,048/ 4×1	2,048/ 4×1
Dilithium2	16,384/ 4×4	4,096 4×1	4,096/ 4×1
Dilithium3	30,720/ 6×5	5,120/ 5×1	5,120/ 5×1
Dilithium5	57,344/ 8×7	7,168/ 7×1	7,168/ 7×1

이 크지 않은 환경에서 적합하다. [표 1]은 Kyber/Dilithium 알고리즘에서 $\boldsymbol{A}$, $\boldsymbol{s}$, $\boldsymbol{e}$ 각각의 스택 사용량을 나타낸다.

김영범 등[17]은 스택 사용량을 줄이기 위한 방법으로 스트리밍 기법을 소개하였다. 스트리밍 기법은 행렬 또는 벡터를 전부 메모리에 할당하지 않고, 다항식 단위로 생성 및 계산을 반복하는 기법이다. [Algorithm 2]는 $\boldsymbol{A}$, $\boldsymbol{s}$, $\boldsymbol{e}$ 모두 다항식 단위의 스트리밍 기법을 적용한 수도 코드이다.

Algorithm 1 행렬×벡터 곱셈 수도 코드

Output : $\hat{\mathbf{t}} = \hat{\boldsymbol{A}} \cdot \hat{\boldsymbol{s}} + \hat{\boldsymbol{e}}$

```
for i = 0 ... n-1 do
    for j = 0 ... m-1 do
        Â[i][j] ← sampling
    end for
end for
for i = 0 ... m-1 do
    s[i] ← sampling
end for
for i = 0 ... m-1 do
    e[i] ← sampling
end for
ŝ = NTT(s)
ê = NTT(e)
t̂ = Â · ŝ + ê
```

Algorithm 2 다항식 단위의 스트리밍 기법을 적용한 행렬×벡터 곱셈 수도 코드

Output : $\hat{\mathbf{t}} = \hat{\boldsymbol{A}} \cdot \hat{\boldsymbol{s}} + \hat{\boldsymbol{e}}$

```
for i = 0 ... n-1 do
    for j = 0 ... m-1 do
        t̂′ ← sampling (t̂′ = Â[i][j])
        t″ ← sampling (t″ = s[j]
        t̂″ = NTT(t″)
        t̂[i] += t̂′ · t̂″
    end for
end for
for i = 0 ... m-1 do
    t′ ← sampling (t′ = e[i])
    t̂′ = NTT(t′)
    t̂[i] += t̂′
end for
```

공개행렬 $\boldsymbol{A}$의 경우, $n \times m$ 크기를 가지며, 비밀 벡터 $\boldsymbol{s}$와 에러 벡터 $\boldsymbol{e}$는 $m \times 1$ 크기를 갖는다. 다항식 단위의 스트리밍 기법을 적용하면, 공개 행렬 $\boldsymbol{A}$를 위한 스택 사용량은 $1/nm$만큼 감소하며, 비밀 벡터 $\boldsymbol{s}$는 $\boldsymbol{A}$의 모든 행과 곱해져야 하기 때문에, $\boldsymbol{A}$와 $\boldsymbol{e}$만을 스트리밍 방식으로 구현하는 것이 일반적이다. 스택 사용량을 극한으로 줄이기 위해서 $\boldsymbol{s}$도 스트리밍 방식을 선택할 수 있지만, 생성 횟수가 많아짐에 따라 연산 부하가 증가한다는 점을 고려해야 한다.

3.2.2. Flash Memory 활용

NTT를 계산하는 과정에서 Ring 분할 및 합체를 위해 고정 상수 값인 twiddle factor가 필요하다. 대부분의 구현들은 twiddle factor를 사전 계산 후 스택에 저장하여 사용한다. 스택 공간이 여유롭지 않은 저사양 장치에서는 부담이 될 수 밖에 없다. 따라 서 twiddle factor와 같은 고정 상수 값에 대한 데이터는 비교적 공간이 여유로운 flash memory/ROM을 활용하는 방안도 고려할 수 있다.

Ⅳ. 결 론

본 논문에서는 저사양 장치에서의 격자 기반 양자내성암호 구현과 관련된 내용들을 살펴보았다. 저사

양 장치는 저전력, 저가 등 이점이 있지만 다른 장치들에 비해 활용 가능한 리소스에 제약이 있다는 단점이 있다.

ARM, GPU, AVX와 같은 장치에서는 연구가 활발히 진행되고 있는 반면, 저전력 장치에서의 연구 결과는 부진하다. 향후 양자내성암호 마이그레이션을 저사양 장치에서도 원활하게 진행하기 위해서는 최적화 연구가 활발히 이루어져야 할 것으로 보인다.

참 고 문 헌

[1] Kannwischer, M. J., Rijneveld, J., Schwabe, P., & Stoffelen, K. (2019). pqm4: Testing and Benchmarking NIST PQC on ARM Cortex-M4.

[2] AVANZI, Roberto, et al. CRYSTALS-Kyber algorithm specifications and supporting documentation. NIST PQC Round, 2019, 2.4: 1-43.

[3] https://csrc.nist.gov/pubs/fips/203/ipd, (2024. 03. 17.)

[4] Ducas, Léo, et al. "Crystals-dilithium: A lattice-based digital signature scheme." IACR Transactions on Cryptographic Hardware and Embedded Systems (2018): 238-268.

[5] https://csrc.nist.gov/pubs/fips/204/ipd, (2024. 03. 17.)

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

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

[8] DongHyun, Shin, Young Beom, Kim, and Seog Chung Seo. "Optimized Kyber implementation on 16-bit MSP430 Microcontroller." World conference on Information Security Applications (WISA), 2023.

[9] 신동현, 김영범, 서석충, “16-bit MSP430 환경에서 하드웨어 곱셈기를 활용한 Dilithium 고속화 연구”, 2023 한국정보보호학회, 동계학술대회.

[10] Abdulrahman, Amin, et al. "Faster kyber and dilithium on the cortex-m4." International Conference on Applied Cryptography and Network Security. Cham: Springer International Publishing, 2022.

[11] Kim, Young Beom, Taek-Young Youn, and Seog Chung Seo. "Chaining optimization methodology: a new sha-3 implementation on low-end microcontrollers." Sustainability 13.8 (2021): 4324.

[12] 신동현, 김영범, 서석충, “16-bit MSP430 환경에서 SHA-3 최적화 구현”, 2023 한국정보보호학회, 충청지부 학술논문발표대회.

[13] Barrett, Paul. "Implementing the Rivest Shamir and Adleman public key encryption algorithm on a standard digital signal processor." Conference on the Theory and Application of Cryptographic Techniques. Berlin, Heidelberg: Springer Berlin Heidelberg, 1986.

[14] Montgomery, Peter L. "Modular multiplication without trial division." Mathematics of computation 44.170 (1985): 519-521.

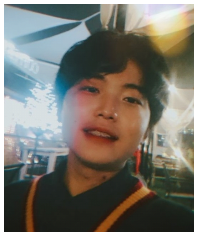
[15] Becker, Hanno, et al. "Neon ntt: Faster dilithium, kyber, and saber on cortex-a72 and apple m1." Cryptology ePrint Archive (2021).

[16] Kannwischer, Matthias J., et al. "Improving software quality in cryptography standardization projects." 2022 IEEE European Symposium on Security and Privacy Work shops (EuroS&PW). IEEE, 2022.

[17] 김영범, 서석충, “8-bit AVR 환경에서 표준화 대상 양자내성암호 Crystals-Kyber 최적화 구현 연구”, 2022 한국정보보호학회, 동계학술대회.

〈저 자 소 개〉

신 동 현 (DongHyun Shin)
학생회원
2023년 8월 : 국민대학교 정보보안 암호수학과 졸업
2023년 8월~현재 : 국민대학교 금융정보보안학과 석사과정
<관심분야> 암호최적화, 양자내성암호



김 영 범 (YoungBeom Kim)
학생회원
2021년 2월 : 국민대학교 정보보안 암호수학과 졸업
2023년 2월 : 국민대학교 금융정보보안학과 졸업
2023년 3월~현재 : 국민대학교 금융정보보안학과 박사과정
<관심분야> 암호최적화, 양자내성암호

서 석 충 (Seog Chung Seo)
정회원
2011년 8월 : 고려대학교 정보보호대학원 박사
2013년 11월 : 삼성전자 종합기술원 전문연구원
2014년 4월 : 삼성전자 DMC 연구소 책임 연구원
2019년 2월 : 국가보안기술연구소 선임연구원
2019년 3월~현재 : 국민대학교 금융정보보안학과 부교수
<관심분야> 암호최적화, 공개키 암호, 양자내성암호, 암호모듈검증, 네트워크보안