64-bit ARMv8 프로세서 상에서의 KpqC 후보 알고리즘 GCKSign의 고속 구현

심민주*, 엄시우*, 권혁동*, 송경주* 이민우*, 서화정*†

*한성대학교 (대학원생)

*†한성대학교 (교수)

High-Speed Implementation of KpqC candidate algorithm GCKSign on 64-bit ARMv8 Processor

Min-Joo Sim*, Si-Woo Eum*, Hyeok-Dong Kwon*, Gyeong-Ju Song*, Min-Woo Lee*, Hwa-Jeong Seo*†

*Hansung University(Graduate student)

*†Hansung University(Professor)

요 약

GCKSign은 국내에서 진행 중인 양자내성암호 표준화 공모전 Round 1에 선정된 격자 기반 전자서명 알고리즘이다. GCKSign은 NIST PQC 표준으로 선정된 Falcon과 CRYSTALS-Dilithum보다 간단하고 효율적이다. 본 논문에서는 ARMv8 프로세서 상에서 GCKSign을 고속 구현하였다. 곱셈 연산의 일부 모듈과 AES 암호 알고리즘을 ARM 명령어로 구현하여 연산 속도를 빠르게 하는 것에 중점을 두었다. GCKSign 암호 알고리즘에 구현 결과를 적용하였을 때, 키 생성 알고리즘의 경우, 1.28배, 서명 생성 알고리즘의 경우, 2.70배 성능 향상을 확인하였다.

I. 서론

NIST(National Institute of Standards and Technology)에서는 양자컴퓨터가 빠르게 발전함에 따라 이를 대비하기 위해 양자내성암호 표준화 공모전(PQC)이 진행되고 있다[1]. 국내에서도 독자적인 양자내성암호 알고리즘을 개발하기 위해 KpqC 공모전이 2021년 말부터 진행 중이다[2]. NIST PQC 공모전 표준화 및 후보군 알고리즘에 대한 최적화 연구가 활발히 진행되고 있다[3]. 이처럼 KpqC 공모전 후보 알고리즘에 대한 최적화 연구도 활발히 진행되어야 한다.

따라서, 본 논문에서는 KpqC 공모전 Round1 후보 알고리즘 중 하나인 GCKSign을 64-bit ARMv8 프로세서 상에서의 고속 구현을 제안한다. 제안하는 기법은 GCKSign에서 사용되는 몽고메리 연산과 NTT 곱셈 내부에서 사용되는 모듈을 고속 구현하였다.

II. 관련 연구

2.1 GCKSign

GCKSign은 격자 기반 전자서명 알고리즘으로, Discrete Gaussian Distribution에 대한 샘플링이 필요하지 않다[4].

Scheme	GCKSign II	GCKSign III	GCKSign V
n	256	256	512
q	$\approx 2^{54}$	$\approx 2^{60}$	$\approx 2^{44}$
m	4	4	3
h	39	45	44
challenge space	192	212	256
B	2^{14} - 1	2^{14} + 2^{9}	2^{15} - 1
η	1	1	1
L_s	18	18	19

[표 1] Parameters of GCKSign

따라서, NIST PQC 표준으로 선정된 Falcon과

CRYSTALS-Dilithum보다 간단하고 효율적이다. GCKSign은 Target-Modified One-wayness(TMO) 문제를 개량하여 사용하였다[5]. 그리고 서명과 키 크기를 줄이기 위해 오류 함수 분석과 Central Limit Theorem(CLT)를 사용하였다. 이를 통해 GCKSign은 서명 체계가 단순해져 빠르고 효율적인 구현이 가능하고 키 관리도 편리하게 설계되었다. [표 1]은 GCKSign에서 사용하는 매개변수를 나타낸 것이다.

2.2 64-bit ARMv8 Processor

ARM(Advanced RISC Machine)은 ISA(Instruction Set Architecture) 고성능 임베디드 프로세서이다[6]. ARMv8-A는 31개의 64-bit general 레지스터(x0~x30)과 128-bit vector 레지스터(v0~v31)를 지원한다. [표 2]는 본 논문에서 사용된 ARM 명령어를 요약한 것이다.

ASM	Description	Operation
MOV	Move	Xd←Xn
LSL	Logical Shift Left (immediate)	Xd←Xn<<#shift
ASR	Arithmetic shift right (immediate)	Xd←Xn>>#shift
ORR	Bitwise OR (shifted register)	Xd←Xn|(Xm <<#amount or Xm >>#amount)
AND	Bitwise AND (shifted register)	Xd←Xn&(Xm <<#amount or Xm >>#amount)
ADD	Add	Xd←Xn+Xm
SUB	Subtract	Xd←Xn-Xm
RET	Return from subroutine	Return

[표 2] Summary of instruction set used in implementation for ARMv8 architecture.

III. 제안 기법

제안하는 기법은 GCKSign의 몽고메리 연산과 NTT 곱셈의 하위 모듈을 구현하였다. 그리고 ARM에서 제공하는 AES 암호화 가속 명령어를 사용하여 고속 구현하였다.

소스코드는 GCKSign[4]에서 배포한 코드를 사용하였다. 하지만, ARMv8 상에서의 GCKSign을 구현하기 위해 해당 코드의 OpenSSL 의존성을 제거하였다. 의존성을 제거하기 위해 PQClean에서 제공하는 AES와 SHA2으로 대체하였다[7].

3.1 몽고메리 연산 구현

ARMv8의 general 레지스터의 크기는 64비트이지만, 하나의 레지스터에 MOV 명령어를 사용하여 레지스터에 값을 지정할 수 있는 범위는 고정적이다.

```
.macro mk_Q
    mov x3, #0x3FFFFFFFFF
    mov x2, #0xD601
    orr x3, x2, x3, lsl #16
.endm

.macro caddp
    and x12, x3, x0, asr #63
    add x0, x12, x0
.endm

.macro csubp
    mk_Q
    sub x0, x0, x3
    caddp
.endm
```

[표 3] Source code used for montgomery operation implementation.

모듈러 연산을 위한 Q값을 하나의 레지스터에 위치하게 하는 작업이 필요하다. 따라서, [표 3]의 .macro mk_Q와 같이 MOV 명령어를 사용하여 하나의 레지스터에 0x3FFFFFFFFF을 위치하게 한다. 그리고 또 다른 레지스터에는 나머지 값인 0xD601을 위치하게 한다. 그 후, 배럴 쉬프트 명령어를 사용하여 0x3FFFFFFFFF을 16비트만큼 왼쪽 로테이션 연산을 한 후, 0xD601 값을 가진 레지스터와 OR 연산을 수행한다. 이렇게 Q값을 위치하게 하여 몽고메리 연산을 위한 모듈들을 ADD, SUB 등을 사용하여 효율적으로 구현하였다.

3.2 NTT 곱셈의 하위 모듈

```
.macro mk_Q_Inv
    mov x3, #0x4460
    lsl x3, x3, #48
    mov x4, #0x1c31
    orr x3, x3, x4, lsl #32
    mov x4, #0x6ee4
    orr x3, x3, x4, lsl #16
    mov x4, #0x2a01
    orr x4, x3, x4
.endm
.macro mont_ivt
    mul x10, x5, x4
    mul x11, x10, x3

    sub x11, x5, x11
    asr x11, x11, #63
    asr x11, x11, #1

    caddp
    sub x11, x11, x3

    and x12, x3, x11, asr #63
    add x0, x12, x11
.endm
```

[표 4] Source code used for NTT multiplication implementation.

NTT 곱셈의 하위 모듈에는 몽고메리 곱셈 연산이 포함된다. [표 4]는 NTT 곱셈의 하위 모듈인 몽고메리 곱셈을 구현한 것의 일부이다.

[표 3]의 .macro mk_Q와 유사한 방법으로 inverse를 위한 Q값은 .macro MK_Q_Inv와 같이 MOV와 ORR 명령어를 사용하여 구현하였다. 몽고메리 곱셈을 위한 연산이 수행되는 ,macro mont_ivt은 128비트 곱셈 연산을 MUL 명령어로 수행한다. 그리고 SUB, ASR, AND, ADD 명령어를 활용하여 모듈러 곱셈 연산을 위한 고속 구현을 [표 4]와 같이 수행하였다.

3.3 AES 암호화

AES-NI는 Intel에서 제안한 x86 명령어 집합의 확장으로, AES의 암호화와 복호화의 성능을 향상시키기 위해 제공되는 명령어이다.

GCKSign 내부에서는 키 생성, 서명 생성, 서명 검증 알고리즘의 모든 단계에서 AES 암호화를 사용하고 있다. 따라서, ARMv8에서 지원하는 AES 암호화 명령어인 [표 5]를 활용하여 AES 암호화 고속 구현을 하여 AES 암호화의 전체적인 연산을 고속화 하였다.

ASM	Description
AESD	AES single round decryption
AESE	AES single round encryption
AESIMC	AES inverse mix columns
AESMC	AES mix columns

[표 5] AES encryption instructions supported by ARMv8.

IV. 성능 평가

GCKSign -II		Keygen	Sign	Verify
Ref	ms	436	1,695	391
	cc	139,520	542,400	125,120
This work	**ms**	**434**	**678**	**399**
	cc	**138,880**	**216,960**	**127,680**
This work(A)	**ms**	**340**	**630**	**398**
	cc	**108,800**	**201,600**	**127,360**

[표 6] Performance evaluation result of GCKSign-II algorithm; Notation(A) indicates with optimization technique using AES-NI.

성능 측정은 ARMv8 아키텍처를 사용하는 Apple M1칩이 탑재된 2020년형 MacBook Pro 13에서 성능 측정을 하였다. xcode상에서 구현을 진행하였으며, 최적화 옵션은 -O2를 사용한다. 성능 측정은 GCKSign-II의 PARAM_1 NTT_S 구현에 대하여 10,000번 반복 동작한 시간(ms)과 Clock cycle(cc)을 측정하였다. 성능 측정 결과는 [표 6]과 같다.

몽고메리 연산과 NTT 곱셈 연산의 일부 모듈 고속 구현 결과(This work)와 레퍼런스 코드를 비교한 결과는 기존 대비 서명 생성 알고리즘의 경우, 2.54배 성능 향상을 확인하였다.

몽고메리 연산과 NTT 곱셈 연산의 일부 모듈 고속 구현 그리고 AES 고속 구현한 결과 (This work(A))를 레퍼런스와 코드와 비교한 결과는 키 생성 알고리즘의 경우, 1.28배, 서명 생성 알고리즘의 경우, 2.70배 성능 향상을 확인하였다.

AES 고속 구현이 적용되기 전과 적용 후를 비교하였을 때, 모두 3가지 알고리즘 모두 성능 향상을 보였다. 키 생성 알고리즘의 경우, 1.28배, 서명 생성 알고리즘의 경우, 1.08배, 서명 검정 알고리즘의 경우, 1.00배 성능 향상을 확인하였다.

키 생성 알고리즘과 검증 알고리즘의 성능이 서명 생성 알고리즘 대비 내부에서 사용되는 곱셈 연산이 적기 때문에 성능이 미미한 것으로 판단된다.

V. 결론

본 논문에서는 KpqC Round1에 진출한 격자 기반 전자서명 알고리즘 GCKSign을 ARMv8 프로세서 상에서 고속 구현하였다. GCKSign의 몽고메리 연산과 NTT 곱셈 내부 모듈에 대해 고속 구현을 하였으며, ARM에서 지원하는 AES-NI를 활용하여 AES 암호화를 고속 구현하였다. 결과적으로, 곱셈 연산의 일부 모듈 고속 구현과 AES 암호화 알고리즘 구현 결과, 키 생성 알고리즘에서는 1.28배, 서명 생성 알고리즘에서는 2.70배 성능 향상을 확인하였다. 향후 연구로, ARMv8 상에서 GCKSign의 모든 파라미터에 대한 NTT 곱셈 최적 구현 연구를 제안한다.

VI. Acknowledgement

This work was partly supported by Institute for Information & communications Technology Promotion(IITP) grant funded by the Korea government(MSIT) (No.2018-0-00264, Research on Blockchain Security Technology for IoT Services, 50%) and this work was partly supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government(MSIT) (No.2022-0-00627, Development of Lightweight BIoT technology for Highly Constrained Devices, 50%).

[참고문헌]

[1] NIST Post-Quantum Cryptography, [Online] https://csrc.nist.gov/projects/post-quantum-cryptography.

[2] 양자내성암호연구단, [Online] https://kpqc.or.kr/

[3] Abdulrahman, Amin, et al. "Faster kyber and dilithium on the cortex-m4." Applied Cryptography and Network Security: 20th International Conference, ACNS 2022, Rome, Italy, June 20–23, 2022, Proceedings. Cham: Springer International Publishing, 2022.

[4] J.Woo, K.Lee, and J.H.Park, “GCKSign:Simple and Efficient Signatures fromGeneralized Compact Knapsacks,”Cryptology eP rint archive, 2022.

[5] V. Lyubashevsky, “Fiat-Shamir with aborts: Applications to lattice and factoringbased signatures,” In *Advances in Cryptology-ASIACRYPT 2009: 15th International Conference on the Theory and Application of Cryptology and Information Security*, pp. 598-616, Dec. 2009.

[6] Armv8-a instruction set architecture. https://developer.arm.com/documentation/den0024/a/An-Introduction-to-the-ARMv8-Instruction-Sets. Accessed: 2022-07-29.

[7] PQClean project. Available online: https://github.com/PQClean/PQClean. Accessed: 2022-07-29.