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

권혁동[1], 송경주[1], 심민주[1], 이민우[2], 서화정[3]
[1]한성대학교 정보컴퓨터공학과 박사과정
[2]한성대학교 융합보안학과 석사과정
[3]한성대학교 융합보안학과 부교수
korlethean@gmail.com, thdrudwn98@gmail.com, minjoos9797@gmail.com,
minunejip@gmail.com, hwajeong84@gmail.com

High-speed Implementation of KpqC candidate algorithm SMAUG on 64-bit ARMv8 processor

Hyeok-Dong Kwon[1], Gyeong-Ju Song[1], Min-Joo Sim[1], Min-Woo Lee[2],
Hwa-Jeong Seo[3]
[1]Dept. of Information Computer Engineering, Hansung University
[2]Dept. of Convergence Security, Hansung University
[3]Dept. of Convergence Security, Hansung University

요 약

SMAUG는 2023년 한국형 양자내성암호 표준화 공모전인 KpqC의 공개키 부문의 1차 후보로 당선된 양자내성암호 알고리즘이다. SMAUG는 MLWE와 MLWR을 사용한 격자 기반 알고리즘으로, 비슷한 문제를 사용하는 CRYSTALS-Kyber에 비해 키 크기가 작다는 장점이 존재한다. 본 논문에서는 SMAUG를 ARMv8 프로세서 상에서 구현하였다. 곱셈 연산의 가장 최하위 모듈을 병렬 구현하여 연산 속도를 빠르게 하는데 집중하였다. 구현 결과 곱셈 알고리즘은 최대 24.62배, 암호 연산에 적용할 경우 최대 3.51배 성능 향상이 있었다.

1. 서론

미국 국립표준기술연구소(NIST, National Institute of Standards and Technology)에서는 양자내성암호 표준화를 위해 공모전을 개최하였다. 22년에는 표준 알고리즘을 선정하였고, 추가 표준을 선정하기 위해 Round 4에 들어갔다[1]. 이러한 흐름에 발맞추어 국내에서도 국내 표준 양자내성암호 선정을 위한 KpqC(Korea Post-Quantum Cryptography) 공모전을 개최하였다[2]. 22년 12월, 해당 공모전에는 공개키 알고리즘 7종과 전자서명 알고리즘 9종이 Round 1을 통과하였다.

본 논문에서는 KpqC Round 1 후보 알고리즘 중 하나인 SMAUG를 64-bit ARMv8 프로세서 상에서 고속 구현을 제안한다. 제안하는 기법은 Vector register를 사용한 병렬 곱셈 방법을 적용하여 속도를 개선하였다. 본 구성은 다음과 같다. 2장에서 양자내성암호 SMAUG에 관해서 확인한다. 3장에서 제안하는 기법을 소개한다. 4장에서 성능 평가를 진행하고, 5장에서 결론을 맺는다.

2. 양자내성암호 SMAUG

양자내성암호 SMAUG는 공개키 알고리즘으로 Module-LWE(Learning With Errors)와 Module-LWR(Learning with Rounding)을 사용한 격자기반암호에 속한다[3]. SMAUG는 양자내성암호 알고리즘으로 제안된 Lizard[4] 및 Ring-Lizard[5]의 구조를 따르기 위해 M-LWE와 M-LWR을 사용했으며 대신 암호 생성 과정에서 해시함수 사용은 제외했다. SMAUG의 모든 moduli는 2의 배수가 되도록 설정되었는데, 이는 modular reduction을 시프트 연산으로만 진행할 수 있게 하였고, 샘플링을 쉽게 하기 위함이다. 대신 NTT(Number Theoretic Transform)를 사용하지 못하기 때문에 곱셈 성능이 떨어질 수 있으나, 사용된 modulus가 작기 때문에 NTT를 사용한 곱셈 성능과 큰 차이가 나지 않는다. 마지막으로 M-LWE와 M-LWR을 사용하였기 때문에 에러 값이 연산에 누적된다. 따라서 복호화 결과 값은 원본 메시지와는 다를 수 있다. 하지만 SMUAG는 오류가 발생할 확률을 최소한으로 만들기 위해 매개변수를 조정하였으며, 복호화가 실패했을 경우에는 의사난수 값을 반환하여 Fujisaki-Okamoto의 변종 알고리즘을 통해 QROM(Quantum Random Oracle Model)의 보안을 보장할 수 있게 하였다[6]. 표 1에서는 SMAUG에서 사용하는 매개변수를 확인할 수 있다.

<Table 1> Parameters of SMAUG.

Scheme	n	k	q	p	t	h_s	h_r	σ	sk(byte)	pk(byte)	ct(byte)
SMAUG-128	256	2	1,024	256	2	140	132	1.0625	174	672	768
SMAUG-192	256	3	1,024	256	2	150	147	1.0625	185	992	1,024
SMAUG-256	256	5	1,024	256	2	145	140	1.0625	182	1632	1,536

SMAUG는 XOF(eXtendable Output Function)로 SHAKE-128을 사용하고 KDF(Key Derivation Function)로 SHAKE-256을 사용한다. 해시함수는 부분적으로 사용되었는데, SHA3-256과 SHA3-512를 사용하였다.

3. 제안 기법

제안하는 기법은 SMAUG의 곱셈기 중에 최하위 모듈을 병렬적으로 구현하였다. 구현에는 Vector instruction과 Vector register 및 약간의 General instruction을 사용하였으며, 구현에 사용한 명령어는 표 2에서 확인할 수 있다.

<Table 2> List of instructions used in implementation.

Instruction	Description
LSL	Logical shift left.
ADD	Addition.
LD1	Load multiple single-element structures.
ST1	Store multiple single-element structures.
SUB	Subtraction.
RET	Return from subroutine.

SMAUG의 곱셈 연산은 Add, Sub, Reduce 세 단계로 구성되어 있으며, 각각 특정 차원에 대한 덧셈, 뺄셈 그리고 리덕션을 진행한다. 제안 기법은 해당 연산을 병렬화하여 빠르게 연산할 수 있도록 하였다. SMAUG의 곱셈은 16-bit 단위로 연산하며 ARMv8의 Vector register는 128-bit까지 저장할 수 있다. 따라서 한번에 8개의 연산 결과 값을 생성할 수 있다. 이를 LWE, LWR dimension 전체를 순회할 수 있도록 반복해주면 연산이 완성된다. SMAUG의 모든 dimension N은 256으로 동일하기 때문에 모든 곱셈기의 성능은 동일하게 도출된다. 또한 Add와 Sub는 연산자만 다르고 형태가 동일하기 때문에 연산에 소요되는 시간이 거의 같다. Reduce의 경우에는 연산 중간 값을 누적시키는 구간이 존재하기 때문에 다소 더 많은 시간이 소요된다. 표 3은 구현에 사용한 소스코드의 일부이다.

<Table 3> Source code used for implementation.

Add(Sub)	Reduce

LD1.8h {v0, v1, v2, v3}, [x0]
LD1.8h {v4, v5, v6, v7}, [x1], #64
ADD.8h v0, v0, v4
ADD.8h v1, v1, v5
ADD.8h v2, v2, v6
ADD.8h v3, v3, v7
ST1.8h {v0, v1, v2, v3}, [x0], #64
	LD1.8h {v0, v1, v2, v3}, [x0]
	LD1.8h {v4, v5, v6, v7}, [x1]
	ADD x1, x1, #512
	LD1.8h {v8, v9, v10, v11}, [x1]
	SUB x1, x1, #448
	SUB.8h v4, v4, v8
	SUB.8h v5, v5, v9
	SUB.8h v6, v6, v10
	SUB.8h v7, v7, v11
	ADD.8h v0, v0, v4
	ADD.8h v1, v1, v5
	ADD.8h v2, v2, v6
	ADD.8h v3, v3, v7
	ST1.8h {v0, v1, v2, v3}, [x0], #64

4. 성능 평가

제안하는 기법은 Xcode 14.3 Framework를 사용하여 구현하였으며, 레퍼런스 소스코드는 SMAUG[3]에서 배포 중인 코드를 사용하였다. 다만, 해당 소스코드는 OpenSSL 의존성이 있기 때문에 의존성을 제거한 스탠드 얼론 형식으로 변형하였다. 대상 프로세서는 Apple M1 Pro 프로세서로 최대 3.2MHz 클록 사이클로 동작한다. 컴파일 옵션은 -O1(fast)을 적용하였다. 곱셈 알고리즘 성능의 비교를 위해서 각 알고리즘을 1,000,000회 반복한 시간의 평균값을 사용하였다. 비교 결과는 표 4와 같다.

알고리즘의 특성상 Add와 Sub의 성능 차이는 거의 없으며, 제안하는 기법이 약 8.9배 빠르다. Reduce의 경우에는 값을 누적시키는 단계가 추가로 존재하기 때문에 24.62배로 더 많은 성능 차이가 발생한다.

<Table 4> Performance evaluation result of multiplier.

Algorithm	Reference		**This work**		**Diff.**
	ms	clock count	**ms**	**clock count**	**x**
Add	0.089	284.8	**0.01**	**32**	**8.9**
Sub	0.089	284.8	**0.01**	**32**	**8.9**
Reduce	0.32	1,024	**0.013**	**41.6**	**24.62**

추가로 제안하는 곱셈 알고리즘을 SMAUG 알고리즘에 적용하여 성능평가를 진행하였다. 성능 측정에는 곱셈 알고리즘과 동일한 환경이며, 반복 횟수는 10,000회를 사용한다. 결과는 표 5에서 확인할 수 있다. 전체적으로 제안하는 기법이 레퍼런스 구현물보다 더 좋은 성능을 지니며, 최대 3.51배 차이가 발생한다. 키 생성 알고리즘의 경우 다른 알고리즘에 비해 성능 격차가 더 적게 발생하는 경향을 가진다. 이는 키 생성 단계에서는 곱셈을 더 적게 사용하기 때문이다.

<Table 5> Performance evaluation result of SMAUG algorithm. (K: Keygen, E: Encapsulation, D: Decapsulation.)

Scheme		Reference		**This work**		**Diff.**
		ms	clock count	**ms**	**clock count**	**x**
128	K	47.30	151,360	**24.39**	**78,048**	**1.93**
	E	53.72	171,904	**21.28**	**68,096**	**2.52**
	D	61.52	196,864	**17.55**	**56,160**	**3.51**
192	K	68.38	218,816	**32.04**	**102,528**	**2.13**
	E	77.71	248,672	**29.41**	**94,112**	**2.64**
	D	86.80	277,760	**35.53**	**113,696**	**2.44**
256	K	111.33	356,256	**51.72**	**165,504**	**2.15**
	E	119.75	383,200	**47.86**	**153,152**	**2.50**
	D	129.20	413,440	**44.44**	**142,208**	**2.90**

5. 결론

본 논문에서는 KpqC Round 1 후보 알고리즘 중 하나인 SMAUG를 64-bit ARMv8 프로세서 상에서 고속 최적 구현을 진행하였다. 제안하는 기법은 곱셈 알고리즘의 최하위 모듈을 병렬 연산으로 변경하였다. 제안 기법은 레퍼런스 구현물보다 곱셈 알고리즘에서 최대 24.62배, 암호 알고리즘에서 최대 3.51배 더 좋은 성능을 보여주었다.

이후 본 알고리즘을 더 개선할 수 있도록 상위 모듈에 대한 최적 구현을 진행하여 추가적인 성능 개선을 이끌어 내도록 한다.

6. Acknowledgements

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, 25%) and this work was partly supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government(MSIT) (No.2021-0-00540, Development of Fast Design and Implementation of Cryptographic Algorithms based on GPU/ASIC, 25%) 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] R. Bavdekar, E. J. Chopde, A. Agrawal, A. Bhatia, and K. Tiwari, “Post Quantum Cryptography: A Review of Techniques, Challenges and Standardizations,” *International Conference on Information Networking (ICOIN)*, Bangkok, Thailand, 2023, pp. 146-151.

Roger S. Pressman "Software Engineering A Practltiners' Approach" 3rd Ed. McGraw Hill

[2] D. E. Yoo, J. S. Kim, and K. M. Kim, “국내·외 양자내성암호 전환 정책 및 상용화 동향,” Vol. 33, No. 1, pp. 59-63, 2023.

[3] J. H. Cheon, H. M. Choe, D. Y. Hong, J. D. Hong, H. E. Seong, J. B. Shin, and M. J. Yi, “SMAUG: the Key Exchange Algorithm based on Module-LWE and Module-LWR,” *KpqC Competition Round 1*, 2022.

[4] J. H. Cheon, D. Kim, J. Lee, and Y. Song, “Lizard: Cut off the tail! A practical post-quantum public-key encryption from LWE and LWR,” *In Security and Cryptography for Networks: 11th International Conference (SCN)*, Amalfi, Italy, 2018, pp. 160-177.

[5] J. Lee, D. Kim, H. Lee, and J. H. Cheon, “RLizard: Post-quantum key encapsulation mechanism for IoT devices,” *IEEE Access*, Vol. 7, pp. 2080-2091, 2018.

[6] D. Hofheinz, K. Hövelmanns, and E. Kiltz, “ A modular analysis of the Fujisaki-Okamoto transformation,” *In Theory of Cryptography: 15th International Conference (TCC)*, Baltimore, USA, 2017, pp. 341-371.