

비트 연산과 다항식 연산 최적화를 통한 ARM Cortex-M4 상에서의 HQC 가속에 대한 연구

김도진, 김동현, 이수중, 전재호, 김영식
대구경북과학기술원 (DGIST)

{ehwls5111,dhkim200426,dltnewhd7426, dgwogh, ysk}@dgist.ac.kr

Research of accelerating HQC on ARM Cortex-M4 by optimizing polynomial multiplication and bitwise operation

Do Jin Kim, Dong Hyun Kim, Soo Jong Lee, Jae Ho Jeon, Young-Sik Kim
Daegu Gyeongbuk Institute of Science&Technology (DGIST)

요 약

본 논문은 포스트 양자 암호(PQC) 알고리즘 중 하나인 HQC(Hamming Quasi-Cyclic)를 저전력/저사양 환경에서 효율적으로 실행하기 위한 최적화 기법을 제안한다. HQC-128 의 병목이 되는 연산을 분석하였고 전체 연산의 대부분을 차지하는 vect_mul 함수와 trailing_zero_bits_count(), gf_carryless_mul() 함수를 최적화하였다. 이를 개선하기 위해 희소 벡터 특성을 활용한 Shift-and-Add 방식, de Bruijn sequence 기반의 빠른 Count Trailing Zeros 계산, 그리고 4 비트 Look-Up Table 기반의 GF(2) 필드 곱셈 최적화를 적용하였다. 제안한 최적화 기법을 통해 HQC-128 의 Keygen, Encapsulation, Decapsulation 연산 시간을 기존 Clean 구현 대비 각각 34 배, 24 배, 24 배 단축하였다.

I. 서 론

기존의 공개키 암호 체계는 양자 컴퓨터의 발전으로 인해 무력화될 위기에 처해있다.[1] 이러한 양자컴퓨터를 사용한 공격에 내성을 가지는 PQC (Post-Quantum Cryptography)가 차세대 암호 체계로 주목받고 있다. NIST는 2016년 부터 PQC 표준을 위한 절차를 시작하였다. 2022년 8월, Dilithium, KYBER, SPHINCS 3 가지 알고리즘이 표준으로 지정되었으며 2025년 3월, HQC(Hamming Quasi-Cyclic) 알고리즘이 새로운 표준 알고리즘으로 선정되었다. HQC의 경우 Error Correction Code 기반의 PQC 알고리즘으로 기존의 ML-KEM 알고리즘들과는 다른 방식을 사용하기 때문에, 앞서 지정된 표준으로 알고리즘이 무력화될 경우 백업 알고리즘으로써 사용될 수 있다. [2] 양자 컴퓨터에 내성을 가지기 위해 PQC는 일반적으로 더 많은 연산을 필요로 한다. 이러한 PQC의 특성과 함께 HQC는 비교적 최근에 표준으로 채택되었기 때문에 IoT 기기 와 센서, 스마트 카드와 같은 저사양/저전력 디바이스에서 HQC 알고리즘을 빠르게 구동하는 연구가 많지 않다.[3] 상호 연결되어 네트워크를 구성하는 IoT 디바이스들의 특성상, 하나의 디바이스에서 양자컴퓨터의 공격에 취약점이 발견된다면, 전체 기기 네트워크의 안전성 또한 침해하게 될 것이다[4].

따라서 본 연구에서는 자원이 제한적인 디바이스에서 HQC를 최적화하는 것을 목표로 한다. 여러 저전력/저사양 프로세서 중에서 HQC를 구동하는 것이 가능 하면서 효율적으로 작동시키는 것에는 무리가 있고, 범용적으로 사용되어 다양한 분야에서 활용할 수 있는 ARM Cortex-M4를 선택하여 해당 환경에서 최적화를 진행하였다.

II. 본론

최적화를 위해 Cortex-M4 프로세서를 장착한 STM32F407G 보드에서 HQC를 구동하며 각 함수별로 소모하는 시간을 프로파일링 하였다. Cortex-M4 환경에 맞게 Reference HQC 코드를 가공한 PQClean의 코드를 사용하였으며 그 결과 HQC에서 PQCLEAN_HQC_128_CLEAN_vect_mul() 함수가 전체 연산의 96%~98%를 차지하는 것을 확인하였다. [5]

추가적으로 Reference 코드를 Ubuntu 환경에서 Vtune 분석을 통해 trailing_zero_bits_count(), gf_carryless_mul() 함수가 많은 시간을 소모하고 있는 사실을 확인했다. 각 연산의 특성을 분석하여 기존 구현의 비효율성을 개선하기 위해 세 가지 최적화 방안을 설계 및 적용하였다.

Cortex - M4 (STM32F407G)		Function	Cpu cycle	Cycle %
Key_gen	331883196	random_fixed_weight	4489190.3	1.35
		gf_mul	0.0	0.00
		vect_mul	325920971.3	98.20
		vect_set_random	1460625.0	0.44
		vect_add	12409.0	0.00
enc	663926069.0	random_fixed_weight	7735495.8	1.17
		gf_mul	2753850.5	0.41
		vect_mul	651821935.0	98.18
		vect_set_random	1578523.3	0.24
		vect_add	36264.5	0.01
dec	1015395072	random_fixed_weight	12243995.0	1.21
		gf_mul	23786825.8	2.34
		vect_mul	977737716.8	96.29
		vect_set_random	1578517.3	0.16
		vect_add	48017.3	0.00

표 1 HQC-128 프로파일링 결과

1) PQCLEAN_HQC128_CLEAN_vect_mul() 함수의 GF(2) 벡터 곱셈을 최적화하였다. HQC 알고리즘에서 사용되는 17669-bit 벡터가 실제로는 대부분의 원소가 0인 Sparse Vector라는 점에 착안하였다. 기존의 Karatsuba 곱셈 방식은 이러한 희소성을 활용하지 못하는 한계가 있었다. 이를 개선하기 위해, 벡터 전체를 저장하는 대신 1의 위치 값(index)만을 저장하는 데이터 구조를 도입하였다. 곱셈 연산 시, 이 인덱스 벡터를 기반으로 희소 벡터의 Weight만큼 반복을 수행하도록 알고리즘을 변경하였다. 각 반복에서는 해당 인덱스 값만큼 Dense Vector를 Shift하고, 그 결과를 이전 연산 결과와 누적 XOR하는 Shift-and-Add 방식을 통해 최종 곱셈 결과를 도출한다.

Algorithm 1 Shift-and-Add Multiplication

```

1: Input: Multiplicand  $M$ , Positions of ones  $Q$ , Hamming weight  $n$ 
2: Output: Product  $A$ 
3:  $A \leftarrow 0$ 
4: for  $i \leftarrow 0$  to  $n - 1$  do
5:    $A \leftarrow A \oplus \text{RightShift}(M, Q[i])$ 
6: end for
7: return  $A$ 

```

2) trailing_zero_bits_count 함수의 성능을 개선하였다. 해당 함수는 입력된 16 비트 값에서 후행 0 비트의 개수(Count Trailing Zeros, CTZ)를 계산한다. 기존 구현은 결과와 무관하게 14 회의 비트 시프트 및 AND 연산을 고정적으로 수행하여 비효율적이었다. 본 연구에서는 이를 개선하기 위해 de Bruijn sequence를 활용하였다. (a & -a) 비트 연산을 통해 입력 a의 최하위 비트(Least Significant Bit, LSB)를 분리하고, 이를 사전 정의된 de Bruijn sequence 상수(0x077CB531)와 곱한다. 이 곱셈 결과는 후행 0의 개수에 비례하는 시프트 연산과 동일한 효과를 내며, 상위 5 비트는 후행 0의 개수에 따라 고유한 값을 갖게 된다. 이 5 비트 값을 인덱스로 사용하여 사전 계산된 Look-Up Table (LUT)에 접근함으로써, 단 한 번의 메모리 참조로 CTZ 값을 즉시 획득할 수 있다.

Algorithm 2 CTZ using De Bruijn Sequence

```

1: Input: 16-bit integer  $a$ , precomputed table  $table[32]$ 
2: Output: Number of trailing zero bits of  $a$ 
3: if  $a = 0$  then
4:   return 16
5: else
6:    $isolated\_bit \leftarrow a \& (-a)$ 
7:    $index \leftarrow (isolated\_bit \times 0x077CB531) \gg 27$ 
8:   return  $table[index]$ 
9: end if

```

3) Reed-Solomon 부호화에 사용되는 gf_carryless_mul() 함수의 GF() 필드 곱셈을 최적화하였다. 기존의 word-by-word 곱셈 방식은 8 비트 단위의 연산에 있어 비효율적이다. 본 연구에서는 8 비트 곱셈의 제한된 입력 범위에 주목하여 4 비트 단위의 Look-Up Table (LUT)을 활용하는 방식을 제안한다. 16×16 크기의 테이블에 4 비트 피연산자의 모든 곱셈 결과를 사전 저장한다. 8 비트 입력값 두 개를 각각 상위 4 비트와 하위 4 비트로 분할한 후, 총 네 번의 테이블 조회와 시프트 및 XOR 연산을 통해 16 비트 중간 결과를 합성한다. 이 방식은 반복적인 산술 연산을 빠른 메모리 접근으로 대체함으로써, 최종 modular reduction이 적용되기 전의 순수 다항식 곱셈, 즉 carry-less multi-plication 단계의 성능을 극대화한다.

Algorithm 3 Karatsuba-like Multiplication

```

1: Input: Operands  $a, b$ , precomputed multiplication table  $mul\_table$ 
2: Output: Product  $c$ 
3:  $c \leftarrow 0$ 
4: Split  $a$  into  $(a_{low}, a_{high})$ 
5: Split  $b$  into  $(b_{low}, b_{high})$ 
6:  $z_0 \leftarrow mul\_table[a_{low}][b_{low}]$ 
7:  $z_2 \leftarrow mul\_table[a_{high}][b_{high}]$ 
8:  $mid_1 \leftarrow mul\_table[a_{high}][b_{low}]$ 
9:  $mid_2 \leftarrow mul\_table[a_{low}][b_{high}]$ 
10:  $middle \leftarrow mid_1 \oplus mid_2$ 
11:  $c_0 \leftarrow z_0 \oplus (middle \ll 4)$ 
12:  $c_1 \leftarrow z_2 \oplus (middle \gg 4)$ 
13: return  $c$ 

```

최적화한 HQC-128의 성능을 Cortex-M4를 탑재한 STM32F4Discovery 보드 환경에서 측정하였다. 성능 비교는 기존 PQClean 구현, 선행 연구, 본 연구의 최적화 구현, 또다른 표준 알고리즘인 Kyber를 대상으로 진행했다.

Algorithm	Keygen (cycle)	Encapsulation (cycle)	Decapsulation (cycle)
HQC-128 (PQClean) [5]	49,469,943	99,681,960	150,204,454
HQC-128 [6]	1,837,507	4,878,515	7,502,580
HQC-128 Ours	1,449,093	4,078,067	6,090,755
Kyber-768 pqm4 [7]	1,244,545	1,267,141	1,352,252

표 2 성능 측정 결과 비교
(cortex M4 168MHz, 1000 회 측정 후, 최대 사이클 수집)

III. 구현 결과 및 결론

결과적으로 새로운 최적 구현 방식이 Clean 구현 대비 Keygen은 약 34 배, Encapsulation과 Decapsulation은 약 24 배 성능이 향상되었다. 또한 단일 함수 최적화를 한 기존 연구와 달리, 추가적인 함수 최적화를 통해 성능을 더 개선하였다. 최적화된 구현에서 Encapsulation은 약 4M 사이클, Decapsulation은 6M 사이클로 저전력 임베디드 시스템에서도 실용적으로 사용할 수 있는 수준에 도달했다.

가장 대표적인 NIST 표준 PQC인 Kyber-768과 비교했을 때 추가로 선정된 HQC는 구조적 특성상 여전히 성능 차이가 있지만, Kyber와 마찬가지로 실용적 수준에서 사용가능한 수준을 확인하였다. 향후 HQC의 경량화 및 고속화를 위한 추가 연구가 필요하며, 특히 ARM Cortex-M 계열 프로세서의 SIMD 명령어를 활용한 벡터 연산 병렬화를 통해 성능을 더욱 향상시킬 수 있을 것으로 기대된다.

ACKNOWLEDGMENT

본 논문은 DGIST UGRP 프로그램의 일환으로 수행되었음.

참고 문헌

- [1] Shor, Peter W. "Algorithms for quantum computation: discrete logarithms and factoring." In Proceedings 35th annual symposium on foundations of computer science, pp. 124-134, 1994.
- [2] Boutin, C. "NIST Selects HQC for Fifth Algorithm for Post-Quantum Encryption," NIST News, Mar. 2025. (<https://www.nist.gov/news-events/news/2025/03/nist-selects-hqc-fifth-algorithm-post-quantum-encryption>)
- [3] Asif, Rameez. 2021. "Post-Quantum Cryptosystems for Internet-of-Things: A Survey on Lattice-Based Algorithms" IoT 2, no. 1: 71-91.
- [4] Jae-Dong, Kim. "A Comprehensive Analysis of Routing Vulnerabilities and Defense Strategies in IoT Networks." arXiv preprint arXiv:2410.13214 (2024).
- [5] Kannwischer, Matthias J., Peter Schwabe, Douglas Stebila, and Thom Wiggers. "Improving software quality in cryptography standardization projects." In 2022 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW), pp. 19-30. IEEE, 2022.
- [6] Aissaoui, Ridwane, Jean-Christophe Deneuville, Christophe Guerber, and Alain Pirovano. "A performant quantum-resistant KEM for constrained hardware: optimized HQC." In 21st International Conference on Security and Cryptography, pp. 668-673. SCITEPRESS-Science and Technology Publications, 2024.
- [7] Matthias J. Kannwischer, Richard Petri, Joost Rijneveld, Peter Schwabe, and Ko Stoffelen. "PQM4: Post-quantum crypto library for the ARM Cortex-M4."