Journal of The Korea Institute of Information Security & Cryptology ISSN 1598-3986(Print)
VOL.34, NO.5, Oct. 2024 ISSN 2288-2715(Online)
https://doi.org/10.13089/JKIISC.2024.34.5.841

AVX2 환경에서 HQC의 GF(2)[x] 곱셈 최적화*

장 지 훈,[1†] 이 명 훈,[1] 김 수 리,[2‡] 서 석 충,[3] 홍 석 희[4]
[1,4]고려대학교 (대학원생, 교수), [2]성신여자대학교 (교수), [3]국민대학교 (교수)

Optimized Implementation of GF(2)[x] Multiplication for HQC on AVX2*

Jihoon Jang,[1†] Myeonghoon Lee,[1] Suhri Kim,[2‡] Seogchung Seo,[3] Seokhie Hong[4]
[1,4]Korea University (Graduate student, Professor),
[2]Sungshin Women's University (Professor), [3]Kookmin University (Professor)

요 약

본 논문에서는 AVX2 환경에서 HQC 내 GF(2)[x] 곱셈 연산의 최적화 방안을 제안한다. HQC는 NIST PQC 공모전 round 4의 후보 알고리즘 중 하나로, 이진 코드 기반의 키 교환 알고리즘이다. GF(2)[x] 곱셈 연산은 HQC의 내부 연산 중 시간복잡도가 높은 연산 중 하나로, AVX2 환경에서 전체 클럭 사이클의 약 30%를 차지한다. GF(2)[x] 곱셈의 최적화를 위해 카라추바, 툼-쿡 알고리즘을 사용하였다. 두 알고리즘 모두 분할 정복 방식으로, 그 내부에서 더 작은 차수의 GF(2)[x] 곱셈 연산이 필요하다. 본 논문에서는 가장 효율적인 카라추바, 툼-쿡 알고리즘의 조합을 찾아 HQC 다항식 곱셈을 최적화하는 방법을 제안하고, 기존 구현물과의 성능을 비교한다. 비교 결과 hqc-128, -192, -256의 GF(2)[x] 곱셈에서 기존 대비 4.5%, 2.5%, 30.3%의 성능 향상을 보인다. 이를 키 생성, 캡슐화, 디캡슐화에 적용하였을 때, 기존 대비 hqc-128에서 각각 2.2%, 2.4%, 2.3%, hqc-192에서 각각 1.6%, 4.2%, 2.6%, hqc-256에서 각각 13.3%, 14.7%, 13.3%의 성능 향상을 보인다.

ABSTRACT

This paper proposes an optimization method for the GF(2)[x] multiplication operation in HQC on AVX2. HQC is a candidate in NIST PQC standardization round 4 and is a binary code-based key exchange algorithm. The multiplication operation is one of the most time-complex operations in HQC, accounting for about 30% of the total clock cycles in the AVX2 environment. For the optimization, we used Karatsuba and Toom-Cook algorithms. Both algorithms are based on divide-and-conquer methods, which require multiplications of smaller order within them. We propose a method to optimize polynomial multiplication in HQC by finding the most efficient combination of Karatsuba and Toom-Cook algorithms, and compare the performance of the proposed method based on the implementation submitted to the PQC standardization. The results of the comparison demonstrate a performance improvement of 4.5%, 2.5%, and 30.3% over the GF(2)[x] multiplications of original hqc-128, -192, and -256. When applied to key generation, encapsulation, and decapsulation, the performance improvement over the original HQC is 2.2%, 2.4%, and 2.3% for hqc-128, 1.6%, 4.2%, and 2.6% for hqc-192, and 13.3%, 14.7%, and 13.3% for hqc-256, respectively.

Keywords: PQC, Finite field multiplication, AVX2, Karatsuba, Toom-Cook

Received(07. 10. 2024), Accepted(08. 05. 2024)
* 이 성과는 정부(과학기술정보통신부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임(No.2022R1F1A1063611)

† 주저자, jhw1031506@korea.ac.kr
‡ 교신저자, suhrikim@sungshin.ac.kr(Corresponding author)

I. 서 론

NIST PQC(Post Quantum Cryptography, 양자 내성 암호) 공모전 round 4에 후보로 오른 HQC[1]는 이진 코드를 기반으로 하는 코드 기반 키 교환 알고리즘으로, 이진 형태의 변수들을 $GF(2)[x]/(x^n-1)$ 상의 다항식으로 취급하여 연산이 이루어진다. 그중 곱셈 연산은 키 생성, 캡슐화, 디캡슐화 과정에 모두 포함되어 있으며, 시간복잡도 기준으로 큰 비중을 차지한다.

$GF(2)[x]$ 위의 곱셈을 포함한 다항식의 곱셈은 다른 연산에 비해 계산복잡도가 높아 전체 알고리즘 과정에서 상당한 비중을 차지한다. 이에 따라 효율적인 다항식 곱셈이 가능한 NTT(number theoretic transform), 카라추바[2], 툼-쿡[3][4], Additive-FFT[5] 등의 여러 알고리즘이 존재한다. 그중 Dilithium[6], Kyber[7] 등에 사용되는 NTT 곱셈 알고리즘은 카라추바, 툼-쿡 알고리즘 등의 다른 알고리즘들보다 효율적인 것으로 알려져 있다. 하지만 이는 $GF(2)[x]$에 적용할 수 없어, $GF(2)[x]$ 상에서의 다항식 곱셈은 카라추바, 툼-쿡 알고리즘 혹은 Additive-FFT 알고리즘을 이용하여 연산을 진행한다. Additive-FFT 알고리즘의 계산복잡도는 $O(n\log n)$으로, 각각 $O(n^{\log_k k(k+1)/2})$, $O(n^{\log_k(2k-1)})$의 복잡도를 가지는 카라추바, 툼-쿡 알고리즘에 비해 작은 복잡도를 가진다. 그러나 `PCLMULQDQ` 명령어를 통해 63차 이하의 $GF(2)[x]$ 곱셈을 효율적으로 연산할 수 있는 AVX2(advanced vector extension 2) 환경에서는 해당 연산의 비중이 높은 카라추바, 툼-쿡 알고리즘이 Additive-FFT 알고리즘보다 좋은 성능을 보인다. 이에 HQC, BIKE 등 $GF(2)[x]$ 곱셈이 필요한 암호의 AVX2 구현물에서는 대부분 카라추바, 툼-쿡 알고리즘을 사용한다. 두 알고리즘은 모두 분할정복 방식으로 다항식을 나누어 가며 곱셈의 수를 줄이고, 상대적으로 계산복잡도가 낮은 덧셈의 수를 늘려 전체적인 계산복잡도를 줄임으로써 효율적으로 다항식 곱을 계산하는 알고리즘이다. 본 논문에서는 AVX2 환경에서 덧셈 및 곱셈의 수를 기준으로 가장 효율적인 카라추바, 툼-쿡 알고리즘의 조합을 찾아 HQC 다항식 곱셈을 최적화하는 방법을 제안한다. 그리고 HQC 상의 $GF(2)[x]$에 대해, 제안한 방법의 구현물과 기존 구현물의 성능을 비교한다.

본 논문의 구성은 다음과 같다. 2장에서는 본 논문의 이해를 돕기 위한 배경지식을 기술한다. 3장에서는 255차 이하 곱셈에 대한 구현, 그리고 카라추바, 툼-쿡 알고리즘의 구현을 설명하고, 이를 이용한 최적화 방안을 제안한다. 4장에서는 본 논문의 최적화 구현 결과를 기존 HQC 구현물과 비교하고, 5장의 결론으로 논문을 마친다.

II. 배경 지식

2.1 $GF(2)[x]$ 다항식 곱셈

$A(x), B(x) \in GF(2)[x]$에 대해 $A(x)=\sum_{i=0}^{n} a_i x^i$, $B(x)=\sum_{i=0}^{m} b_i x^i$ $(n \geq m)$이라 할 때, 덧셈과 곱셈은 아래와 같이 정의된다.

$$\text{Addition} \quad : A(x)+B(x)=\sum_{i=0}^{n}(a_i+b_i)x^i$$

$$\text{Multiplication} : A(x)\times B(x)=\sum_{i=0}^{n}\sum_{j=0}^{m} a_i b_j x^{i+j}$$

$(b_i = 0 \text{ for } m < i \leq n)$.

이때 계수 간의 연산은 $GF(2)$ 상의 연산이다. $GF(2)$ 상의 덧셈은 XOR 연산과 동일하고, 곱셈은 AND 연산과 동일하다. 이러한 $GF(2)[x]$ 상에서의

Algorithm :
Schoolbook multiplication on $GF(2)[x]$

Input : $A(x)=\sum_{i=0}^{n} a_i x^i$, $B(x)=\sum_{i=0}^{n} b_i x^i$
Output : $C(x)=A(x)\times B(x)$

```
1  for i=0 to n+m do
2      c_i ← 0
3  end for
4  for i=0 to n do
5      for j=0 to m do
6          c_{i+j} ← c_{i+j} + a_i × b_j
7      end for
8  end for
9  return C(x) = Σ_{i=0}^{n+m} c_i x^i
```

Fig. 1. Schoolbook multiplication on $GF(2)[x]$

곱셈 연산은 받아올림(carry)이 없어 carry-less 곱셈이라고도 불리며, Fig. 1과 같이 단순한 형태의 schoolbook 방식으로 나타낼 수 있다.

2.2 카라추바 알고리즘

m-카라추바 알고리즘은 $mk-1$차 이하 다항식을 m개의 $k-1$차 이하 다항식으로 나누어서 곱셈하는 다항식 곱셈 알고리즘이다. 본 논문에서는 2-, 3-, 5-카라추바 알고리즘을 사용한다.

2.2.1 2-카라추바 알고리즘

2-카라추바 알고리즘은 $2k-1$차 이하 다항식을 2개의 $k-1$차 이하 다항식으로 나누어 다항식 곱을 계산하는 알고리즘이다. $2k-1$차 이하 다항식 $A(x)$와 $B(x)$에 대해 아래와 같이 다항식을 나누고, 연산을 진행하여 $C(x)=A(x)\times B(x)$를 구한다.

$$\begin{aligned}A(x)&=a_{2k-1}x^{2k-1}+\cdots+a_1x+a_0\\&=A_1X+A_0\\B(x)&=b_{2k-1}x^{2k-1}+\cdots+b_1x+b_0\\&=B_1X+B_0\end{aligned}$$

X는 x^k이고, A_0, A_1, B_0, B_1는 아래와 같은 $k-1$차 이하 다항식이다.

$$\begin{aligned}A_0(x)&=a_{k-1}x^{k-1}+\cdots+a_1x+a_0\\A_1(x)&=a_{2k-1}x^{k-1}+\cdots+a_{k+1}x+a_k\\B_0(x)&=b_{k-1}x^{k-1}+\cdots+b_1x+b_0\\B_1(x)&=b_{2k-1}x^{k-1}+\cdots+b_{k+1}x+b_k\end{aligned}$$

이때 $C(x)=A(x)\times B(x)$은 아래와 같이 나타낼 수 있다.

$$\begin{aligned}C(x)&=(A_1X+A_0)(B_1X+B_0)\\&=A_1B_1X^2+(A_1B_0+A_0B_1)X+A_0B_0\\&=A_1B_1X^2\\&\quad+\{(A_1+A_0)(B_1+B_0)-A_1B_1-A_0B_0\}X\\&\quad+A_0B_0\end{aligned}$$

위 식을 통해 계산할 경우, $(A_1+A_0)(B_1+B_0)$, A_0B_0, A_1B_1 총 3번의 $k-1$차 이하 다항식 곱셈을 통해 $2k-1$차 이하 다항식 곱셈을 수행할 수 있다. 4번의 $k-1$차 이하 곱셈이 필요한 schoolbook 방식에 비해 효율적인 연산이 가능하다.

2.2.2 3-카라추바 알고리즘

3-카라추바 알고리즘은 $3k-1$차 이하 다항식을 3개의 $k-1$차 이하 다항식으로 나누어 다항식 곱셈을 계산하는 다항식 곱셈 알고리즘이다. $3k-1$차 이하 다항식 $A(x)$와 $B(x)$에 대해 $C(x)=A(x)\times B(x)$ 를 구하는 다항식 곱을 계산한다고 가정하자. 2-카라추바 알고리즘과 유사한 방법으로 $A(x)$와 $B(x)$를 3개의 $k-1$차 이하 다항식으로 나누어 $X=x^k$로 나타내고 $C(x)$를 계산한다.

$$\begin{aligned}A(x)&=a_{3k-1}x^{3k-1}+\cdots+a_1x+a_0\\&=A_2X^2+A_1X+A_0\\B(x)&=b_{3k-1}x^{3k-1}+\cdots+b_1x+b_0\\&=B_2X^2+B_1X+B_0\\C(x)&=(A_2X^2+A_1X+A_0)(B_2X^2+B_1X+B_0)\\&=A_2B_2X^4\\&\quad+\{(A_1+A_2)(B_1+B_2)-A_1B_1-A_2B_2\}X^3\\&\quad+\{(A_2+A_0)(B_2+B_0)\\&\qquad-A_2B_2-A_0B_0+A_1B_1\}X^2\\&\quad+\{(A_0+A_1)(B_0+B_1)-A_0B_0-A_1B_1\}X\\&\quad+A_0B_0\end{aligned}$$

총 6번의 $k-1$차 이하 다항식 곱셈을 통해 $3k-1$차 이하 다항식 곱셈을 수행할 수 있다. 9번의 $k-1$차 이하 곱셈이 필요한 schoolbook 방식에 비해 효율적인 연산이 가능하다.

2.2.3 5-카라추바 알고리즘

5-카라추바 알고리즘은 다항식을 5개로 나누어 다항식 곱셈을 계산하는 알고리즘이다. $5k-1$차 이하 다항식 $A(x)$와 $B(x)$에 대해 $C(x)=A(x)\times B(x)$ 를 구하는 다항식 곱을 계산한다고 가정하자. 2-, 3-카라추바 알고리즘과 유사한 방법으로 $A(x)$와 $B(x)$를 5개의 $k-1$차 이하 다항식으로 나누고 $C(x)$를 계산한다. 5-카라추바 알고리즘을 사용하면 15번의 $k-1$차 이하 다항식

곱셈을 통해 다항식 곱셈을 수행할 수 있다. 25개의 곱셈이 필요한 schoolbook 방식보다 10회 적다.

2.3 툼-쿡 알고리즘

툼-쿡 m-way 알고리즘은 $mk-1$차 이하 다항식을 m개의 $k-1$차 이하 다항식으로 나누고, 나눈 각 다항식을 계수로 하는 $m-1$차 이하 다항식에 대해 분할, 평가, 보간의 과정을 통해 다항식 곱셈을 계산하는 알고리즘이다.

분할: 분할 과정은 $mk-1$차 이하 다항식 $A(x)$, $B(x)$을 아래 식과 같이 m개의 $k-1$차 이하 다항식으로 나누어 $X=x^k$를 기저로 하는 $m-1$차 이하 다항식으로 변환하는 과정이다. 이후 과정에서 $A(X)$, $B(X)$는 이 과정에서 얻어진 $X=x^k$를 기저로 하는 $m-1$차 이하의 다항식이다.

$$A(x)=\sum_{i=0}^{mk-1} a_i x^i=\sum_{i=0}^{m-1} A_i(x)X^i$$
$$B(x)=\sum_{i=0}^{mk-1} b_i x^i=\sum_{i=0}^{m-1} B_i(x)X^i$$

평가: 평가 과정은 분할 과정에서 얻은 두 $(m-1)$차 다항식 $A(X)$와 $B(X)$에 $(2m-1)$개의 평가점을 대입하여, 이들끼리의 곱을 얻는 과정이다. 즉, 아래 식과 같이 평가점 $p_1,\ldots,p_{2m-1}$에 대해 $C(p_1),\ldots,C(p_{2m-1})$를 얻는 과정이다.

$$\begin{aligned} C(p_1)&=A(p_1)\times B(p_1)\\ &\vdots\\ C(p_{2m-1})&=A(p_{2m-1})\times B(p_{2m-1}) \end{aligned}$$

보간: 보간 과정은 평가 과정으로부터 얻은 $(2m-1)$개의 결과로부터 $C(x)=A(x)\times B(x)$의 계수를 계산하는 과정이다. 이 과정은 $(2m-1)$원 연립 일차 방정식과 동일하다. 이후 얻은 $(2m-1)$개의 $C(x)$의 계수를 차수에 맞게 결합하여 $A(x)$와 $B(x)$의 곱셈 결과를 구한다.

툼-쿡 m-way 알고리즘은 $(2m-1)$개의 평가점이 필요하나 계수가 0과 1로 이루어진 GF(2)[x] 상에서는 평가점을 잡기 어려워, 본 논문에서는 5개의 평가점이 필요한 툼-쿡 3-way만을 다룬다.

2.3.1 GF(2)[x]에서의 툼-쿡 3-way 알고리즘

툼-쿡 3-way 알고리즘은 다항식을 3개의 작은 다항식으로 나누고, 나눈 각 다항식을 계수로 하는 이차다항식에 대해 곱셈을 계산하는 방식의 알고리즘이다. $3k-1$차 이하 다항식 $A(x)$와 $B(x)$를 3-카라추바 알고리즘에서와 같이 이차다항식 $A(X)$, $B(X)$으로 나타내고, 두 다항식의 곱 $C(X)=A(X)\times B(X)$는 아래와 같이 사차다항식으로 나타낸다.

$$\begin{aligned} A(X)&=A_2X^2+A_1X+A_0,\\ B(X)&=B_2X^2+B_1X+B_0,\\ C(X)&=C_4X^4+C_3X^3+C_2X^2+C_1X+C_0. \end{aligned}$$

이후 $C(x)$의 5개 계수(C_4, C_3, C_2, C_1, C_0)를 구하기 위해 5개의 평가점을 대입하여 5원 연립 일차 방정식을 푼다. 우리는 GF(2)[x]에서 툼-쿡 알고리즘을 적용시키기 위해 [9]에서의 방법을 사용한다. 평가점은 $0,1,W,W+1,\infty$이다. W는 1 워드만큼의 left shift를 의미하는 상수로, GF(2)[x]에서 5개의 평가점을 설정하기 위해 정의된 상수이다.

2.4 HQC

HQC는 NIST PQC round 4의 후보로 오른 이진 코드 기반의 키 교환 알고리즘이다. HQC의 특징으로는 Classic McEliece, BIKE 등의 다른 코드 기반 암호와는 달리 인코딩 및 디코딩 과정에서 비밀 정보가 필요하지 않아 누구나 인코딩 및 디코딩을 할 수 있다. 그 대신 디코딩 과정에서 수정할 수 있는 오류 크기의 범위가 존재하는데, 이를 이용하여 암호화 및 복호화가 진행된다. 메시지를 인코딩한 후 공개키를 이용하여 오류를 주입함으로써 암호화가 진행되고, 복호화 과정에서 비밀키를 이용하여 오류의 크기를 줄인 후, 디코딩을 통해 오류를 수정함으로써 메시지를 복원한다. 자세한 알고리즘은 [1]을 참조한다. 오류를 설정하는 과정과 오류의 크기를 줄이는 과정에서 이진 형태의 값을 GF(2)[x]/(x^n-1)의 원소에 대응시켜 연산이 진행된다. 덧셈 연산은 차수에 영향을 주지 않기 때문에 GF(2)[x] 상에서의 덧셈 연산과 동일하며, 곱셈 연산은 GF(2)[x] 상에서의 n차 다항식 곱셈과 x^n-1에 대한 감산으로 이루어진다.

〔Table 1〕은 AVX2 환경에서 양자 128-bit 보안 강도를 가지는 hqc-128의 키 생성, 캡슐화, 디캡슐화 과정 중 GF(2)[x] 곱셈이 차지하는 클럭 사이클의 비율을 보여준다. HQC의 최근 제출물 중 AVX2 환경 구현물(2024/02/23, Optimized implementation)을 기준으로 측정하였으며, 컴파일러로는 GNU GCC version 11.3.0을 사용하였고, 최적화 수준은 O3로 컴파일하였다. 타겟 컴퓨터의 프로세서는 13th Gen Intel(R) Core(TM) i7-13700K (3.40 GHz)이며 OS는 Ubuntu 22.04.2 LTS 이다. 단위는 클럭 사이클이다.

〔Table 1〕에서 볼 수 있듯, GF(2)[x] 곱셈은 HQC의 전체 과정에서 큰 비중을 차지한다. hqc-128의 경우 17668차 GF(2)[x] 곱셈이 이루어지며, hqc-192, -256의 경우 각각 35850차, 57636차 GF(2)[x] 곱셈이 이루어진다.

HQC의 최근 제출물의 GF(2)[x] 곱셈은 〔8〕에서 제안한 방법의 곱셈 알고리즘이 적용되어 있다. hqc-128, -192의 GF(2)[x] 곱셈에는 가장 먼저 툼-쿡 3-way 알고리즘을 사용하고, 그 내부에서 필요한 곱셈에 대해 모두 3-카라추바를 사용한 후, 3-카라추바 내부의 곱셈에 대해서는 2-카라추바를 계속해서 재귀적으로 사용하는 방법이 적용되어 있다. hqc-256의 GF(2)[x] 곱셈에는 hqc-128, -192의 곱셈과 유사하지만 3-카라추바 대신 5-카라추바를 적용한다는 차이점을 가진다.

Table 1. Proportion of the GF(2)[x] multiplication in the recently submitted AVX2 implementation of hqc-128 (Optimized Implementation)

	entire process	GF(2)[x] multiplication	proportions
keygen	46,772	15,134	32.4%
encaps	109,914	31,343	28.5%
decaps	206,529	64,050	31.0%

2.5 AVX2 명령어 집합

AVX2 명령어 집합은 Intel의 x86 명령어 집합의 확장으로, 256-bit SIMD(Single Instruction Multiple Data) 명령어 집합이다. AVX2는 128-bit의 xmm 레지스터와 256-bit의 ymm 레지스터를 사용한다. 각각 16개로, xmm0 - xmm15와 ymm0 - ymm15가 존재한다. xmm와 ymm 레지스터를 사용하면 128-bit 혹은 256-bit 단위로 SIMD 연산을 할 수 있다.

AVX2 명령어 집합을 이용하면 GF(2)[x] 상에서의 곱셈 및 덧셈을 효과적으로 연산할 수 있다. 먼저 AVX2 환경에서 사용할 수 있는 `PCLMULQDQ` 명령어의 인트린직 함수

```
__m128i _mm_clmulepi64_si128 (__m128i b, __m128i c, const int Imm8)
```

를 이용하면 효율적인 GF(2)[x] 곱셈 연산이 가능하다. 이 함수는 127차 이하 GF(2)[x] 다항식 $b(x)=b_0(x)+b_1(x)x^{64}$와 $c(x)=c_0(x)+c_1(x)x^{64}$에 대해, Imm8의 값에 따라 아래와 같이 63차 이하 다항식의 곱셈 결과 128-bit 값을 반환하는 함수이다.

$$returns\ b_0(x)\times c_0(x)\ \text{ if Imm8}=0\text{x00}$$
$$returns\ b_1(x)\times c_0(x)\ \text{ if Imm8}=0\text{x01}$$
$$returns\ b_0(x)\times c_1(x)\ \text{ if Imm8}=0\text{x10}$$
$$returns\ b_1(x)\times c_1(x)\ \text{ if Imm8}=0\text{x11}$$

또한 AVX2 명령어 집합을 이용하면 GF(2)[x] 상에서의 덧셈(XOR)을 효율적으로 연산할 수 있다. 인트린직 함수

```
__m256i _mm256_xor_si256 (__m256i a, __m256i b)
```

는 256-bit a,b에 대해 a와 b의 bitwise-XOR 결과값 256-bit를 반환하는 함수로, 이는 255차 이하 GF(2)[x] 다항식의 덧셈과 동일한 연산이다.

III. HQC GF(2)[x] 곱셈의 최적화 구현

본 장에서는 HQC 내 GF(2)[x] 곱셈의 최적화 구현을 제시한다. 먼저 255차 다항식 단위 곱셈의 구현 방법과 카라추바, 툼-쿡 알고리즘의 여러 구현 방법을 소개하고, 각각에 대한 곱셈 연산 및 덧셈(XOR) 연산의 개수를 분석한다. 또한 연산 수를 기준으로 곱셈 알고리즘의 최적 조합을 탐색하는 방법을 제시하고, 제시한 방법으로 탐색한 최적 조합을 제시한다.

3.1 단위 곱셈

카라추바, 툼-쿡 알고리즘 모두 다항식을 나누어 계산하는 분할 정복 방식으로 이루어지는 다항식 곱셈 알고리즘이다. 이에 따라 최종적으로 효율적으로 곱할 수 있을 만큼 충분히 낮은 차수의 다항식 곱셈이 이루어진다.

AVX2 환경에서는 2.5절의 명령어들에 의해 256-bit XOR 연산 및 64-bit GF(2)[x] 다항식 곱셈 연산이 가능하다. 이에 따라 GF(2)[x] 다항식은 256-bit 자료형 배열로 저장하여 다룬다. 예를 들어 17668차 다항식을 사용하는 hqc-128에 대해서는 길이가 $\lceil 17669/256 \rceil = 70$인 256-bit 자료형 배열에 저장된다. 따라서 카라추바, 툼-쿡 알고리즘을 사용하는 경우 차수가 255차 이하가 될 때까지 나누고, 255차 이하 다항식에 대해 AVX2 명령어를 이용하여 곱셈을 진행하는 것이 효율적이다.

본 논문에서는 이러한 255차 이하 다항식 곱셈을 '단위 곱셈'이라 부른다. 또한, 255차 이하 다항식 덧셈(XOR)을 '단위 덧셈'이라 부른다. 추가로 단위 곱셈을 1차원 곱셈으로, $(256n-1)$차 GF(2)[x] 다항식 곱셈을 n차원 곱셈으로 정의하고, $(256n-1)$차 GF(2)[x] 다항식을 n차원 다항식으로 정의한다.

단위 곱셈의 구현물로 [8]의 구현물을 사용한다. 두 개의 255차 이하 다항식을 입력받아 두 GF(2)[x] 다항식 곱셈 결과인 510차 이하 다항식을 반환하는 함수로, 2-카라추바 알고리즘을 통해 진행된다. 255차 다항식을 네 개의 63차 다항식으로 나누고, `_mm_clmulepi64_si128` 를 이용하여 63차 다항식의 곱셈을 계산한다. 이후 2-카라추바 알고리즘을 두 번 중첩하여 255차 다항식 곱셈을 계산한다.

3.2 카라추바 알고리즘의 구현 및 분석

3.2.1 2-카라추바 알고리즘

Fig. 2는 $2k$차원 곱셈에 대한 2-카라추바 알고리즘이다. Fig. 2의 line 1, 2, 3에서 3개의 다항식 곱셈이 이루어진다. 이때 A_0, A_1와 B_0, B_1 모두 k차원 다항식이므로 세 가지 곱셈 모두 k차원의 곱셈이다.

다항식 A, B의 차원이 홀수 $2k+1$인 경우, A_1과 B_1은 k차원, A_0와 B_0는 $(k+1)$차원의 다항식이고, 내부에서 필요한 곱셈은 k차원 하나와 $(k+1)$차원 두 개이다.

Algorithm : 2-Karatsuba algorithm

Input : $A(X)=A_1X+A_0$, $B(X)=B_1X+B_0$

Output : $C(X)=A(X)\times B(X)$

1 $C_0 \leftarrow A_0 \times B_0$

2 $C_2 \leftarrow A_1 \times B_1$

3 $C_1 \leftarrow (A_0+A_1)\times(B_0+B_1)$

4 **return** $C(X)=C_2X^2+(C_0+C_1+C_2)X+C_0$

Fig. 2. 2-Karatsuba algorithm

2-카라추바 알고리즘에서 곱셈의 차원에 따른 필요 내부 곱셈의 수와 단위 덧셈 수를 표로 나타내면 [Table 2]와 같다.

Table 2. Required internal multiplications and base additions in the 2-Karatsuba algorithm

dimension	# of internal multiplications		# of base additions
	k -dimension	$(k+1)$ -dimension	
$2k$	3	0	$7k$
$2k+1$	1	2	$7k+3$

3.2.2 3-카라추바 알고리즘

Fig. 3은 $3k$차원 곱셈에 대한 3-카라추바 알고리즘이다. Fig. 3의 line 1, 2, 3, 4, 5, 6에서 6개의 다항식 곱셈을 필요로 한다. 이때 A_0, A_1, A_2

Algorithm : 3-Karatsuba algorithm

Input : $A(X)=A_2X^2+A_1X+A_0$,
$B(X)=B_2X^2+B_1X+B_0$

Output : $C(X)=A(X)\times B(X)$

1 $R_0 \leftarrow A_0 \times B_0$, $R_1 \leftarrow A_1 \times B_1$,

2 $R_2 \leftarrow A_2 \times B_2$, $R_3 \leftarrow (A_0+A_1)\times(B_0+B_1)$

5 $R_4 \leftarrow (A_0+A_2)\times(B_0+B_2)$

6 $R_5 \leftarrow (A_1+A_2)\times(B_1+B_2)$

7 $C_5 \leftarrow R_0$, $C_1 \leftarrow R_3+R_0+R_1$

9 $C_2 \leftarrow R_4+R_0+R_1+R_2$

10 $C_3 \leftarrow R_5+R_1+R_2$, $C_4 \leftarrow R_2$

12 return $C(X)=C_2X^2+(C_0+C_1+C_2)X+C_0$

Fig. 3. 3-Karatsuba algorithm

와 B_0, B_1, B_2 모두 k차원 다항식이므로 모두 k차원 곱셈이다.

그러나 다항식 A, B의 차원이 $3k+1$ 형태인 경우, A_2와 B_2는 $(k+1)$차원, A_0, A_1와 B_0, B_1는 k차원의 다항식으로 설정한다. 이 경우 k차원 세 개와 $(k+1)$차원 세 개의 내부 곱셈이 필요하다.

또한 다항식 A, B의 차원이 $3k+2$ 형태인 경우, A_2와 B_2는 k차원, A_0, A_1와 B_0, B_1는 $(k+1)$차원의 다항식으로 설정한다. 이 경우 내부에서 필요한 곱셈은 $(k+1)$차원 다섯 개와 k차원 한 개이다.

3-카라추바 알고리즘에서 곱셈의 차원에 따른 필요 내부 곱셈의 수와 단위 덧셈 수를 표로 나타내면 〔Table 3〕과 같다.

Table 3. Required internal multiplications and base additions in the 3-Karatsuba algorithm

dimen -sion	# of internal multiplications		# of base additions
	k -dimension	$(k+1)$ -dimension	
$3k$	6	0	$20k$
$3k+1$	3	3	$20k+8$
$3k+2$	1	5	$20k+14$

3.2.3 5-카라추바 알고리즘

Fig. 4는 $5k$ 차원 곱셈에 대한 5-카라추바 알고리즘이다. Fig. 4의 line 1, 2, 3, 4, 5, 6에서 총 15개의 k차원 다항식 곱셈을 필요로 한다. 5-카라추바 알고리즘의 계산복잡도는 $O(n^{\log_5 15})$로 다른 알고리즘에 비해 크다. 따라서 본 논문에서는 곱셈의 차원이 5의 배수인 경우에만 5-카라추바 알고리즘을 고려하였다. $5k$차원 곱셈에 대해 5-카라추바 알고리즘을 적용하면, 15개의 k차원 곱셈과 $80k$개의 단위 덧셈이 필요하다.

3.3 툼-쿡 알고리즘의 구현 및 분석

Fig. 5는 〔9〕에서 제시한 GF(2)$[x]$에서의 툼-쿡 3-way 계산 방식이다. 곱셈은 5번 진행되고, 각각은 아래의 곱셈이 계산되는 과정이다.

$$C_1 \leftarrow C_2 \times C_5 = (A_0+A_1+A_2)\times(B_0+B_1+B_2) \quad (1)$$

Algorithm :
5-Karatsuba algorithm

Input : $A(X)=A_4X^4+A_3X^3+A_2X^2+A_1X+A_0$,
$B(X)=B_4X^4+B_3X^3+B_2X^2+B_1X+B_0$
Output : $C(X)=A(X)\times B(X)$

1 $R_0 \leftarrow A_0 \times B_0$, $R_1 \leftarrow A_1 \times B_1$, $R_2 \leftarrow A_2 \times B_2$
2 $R_3 \leftarrow A_3 \times B_3$, $R_4 \leftarrow A_4 \times B_4$
3 $R_5 \leftarrow (A_0+A_1)\times(B_0+B_1)$
4 $R_6 \leftarrow (A_0+A_2)\times(B_0+B_2)$
5 $R_7 \leftarrow (A_0+A_3)\times(B_0+B_3)$
6 $R_8 \leftarrow (A_0+A_4)\times(B_0+B_4)$
7 $R_9 \leftarrow (A_1+A_2)\times(B_1+B_2)$
8 $R_{10} \leftarrow (A_1+A_3)\times(B_1+B_3)$
9 $R_{11} \leftarrow (A_1+A_4)\times(B_1+B_4)$
10 $R_{12} \leftarrow (A_2+A_3)\times(B_2+B_3)$
11 $R_{13} \leftarrow (A_2+A_4)\times(B_2+B_4)$
12 $R_{14} \leftarrow (A_3+A_4)\times(B_3+B_4)$
13 $C_0 \leftarrow R_0$, $C_1 \leftarrow R_5+R_0+R_1$
14 $C_2 \leftarrow R_1+R_6+R_0+R_2$
15 $C_3 \leftarrow R_7+R_0+R_3+R_9+R_1+R_2$
16 $C_4 \leftarrow R_2+R_8+R_0+R_4+R_{10}+R_1+R_3$
17 $C_5 \leftarrow R_{11}+R_1+R_4+R_{12}+R_2+R_3$
18 $C_6 \leftarrow R_3+R_{13}+R_2+R_4$
19 $C_7 \leftarrow R_{14}+R_3+R_4$, $C_8 \leftarrow R_4$
22 **return** $C(X)=C_8X^8+C_7X^7+C_6X^6+C_5X^5+C_4X^4+C_3X^3+C_2X^2+C_1X+C_0$

Fig. 4. 5-Karatsuba algorithm

Algorithm :
Toom-Cook 3-way algorithm on GF(2)$[x]$

Input : $A(X)=A_2X^2+A_1X+A_0$,
$B(X)=B_2X^2+B_1X+B_0$ in GF(2)$[x]$
Output : $C(X)=A(X)\times B(X)$

1 Let $W=x^w$
2 **Evaluation:**
3 $C_0 \leftarrow A_1W+A_2W^2$, $C_4 \leftarrow B_1W+B_2W^2$
4 $C_5 \leftarrow A_0+A_1+A_2$, $C_2 \leftarrow B_0+B_1+B_2$
5 $C_1 \leftarrow C_2 \times C_5$, $C_5 \leftarrow C_5+C_0$, $C_2 \leftarrow C_2+C_4$
6 $C_0 \leftarrow C_0+A_0$, $C_4 \leftarrow C_4+B_0$, $C_3 \leftarrow C_2 \times C_5$
7 $C_2 \leftarrow C_0 \times C_4$, $C_0 \leftarrow A_0 \times B_0$, $C_4 \leftarrow A_2 \times B_2$
8 **Interpolation:**
9 $C_3 \leftarrow C_3+C_2$, $C_2 \leftarrow C_2+C_0$, $C_2 \leftarrow C_2/W+C_3$
10 $C_2 \leftarrow (C_2+(1+W^3)C_4)/(1+W)$, $C_1 \leftarrow C_1+C_0$,
11 $C_3 \leftarrow C_3+C_1$, $C_3 \leftarrow C_3/(W^2+W)$,
11 $C_1 \leftarrow C_1+C_2+C_4$, $C_2 \leftarrow C_2+C_3$
12 **return** $C(X)=C_4X^4+C_3X^3+C_2X^2+C_1X+C_0$

Fig. 5. Toom-Cook 3-way algorithm on GF(2)$[x]$

$$C_3 \leftarrow C_2 \times C_5 = (A_0 + A_1(W+1) + A_2(W+1)^2) \times (B_0 + B_1(W+1) + B_2(W+1)^2) \quad (2)$$

$$C_2 \leftarrow C_0 \times C_4 = (A_0 + A_1 W + A_2 W^2) \times (B_0 + B_1 W + B_2 W^2) \quad (3)$$

$$C_0 \leftarrow A_0 \times B_0 \quad (4)$$

$$C_4 \leftarrow A_2 \times B_2 \quad (5)$$

식 (1)의 차원은 A_0, A_1, A_2의 차원의 최댓값, 식 (4), (5)의 차원은 각각 A_0, A_2의 차원과 같다. 그리고 식 (2), (3)의 차원은 A_2W^2, A_1W, A_0의 차원의 최댓값으로 정해진다. 따라서 $W=x^w$의 크기를 어떻게 정의하는지에 따라 필요한 다항식 곱셈의 크기가 달라진다.

본 논문에서는 다항식들을 모두 AVX2 환경의 256-bit 크기 자료형 배열로 다루므로, $W=x^{256}$인 툼-쿡 알고리즘을 고려한다. $A(X)$와 $B(X)$의 차원이 $3k$, $3k+1$, $3k+2$일 때 알고리즘 내에서 필요한 곱셈인 식 (1), (2), (3), (4), (5)의 차원은 각각 〔Table 4〕와 같이 결정된다. 그리고 필요한 내부 곱셈의 수를 정리하여 단위 덧셈 수와 함께 나타내면 〔Table 5〕와 같다.

Table 4. The dimensions of the eq.(1) - (5) (Toom-Cook 3-way)

dimen-sion	eq.(1)	eq.(2)	eq.(3)	eq.(4)	eq.(5)
$3k$	k	$k+2$	$k+2$	k	k
$3k+1$	$k+1$	$k+2$	$k+2$	$k+1$	$k-1$
$3k+2$	$k+1$	$k+2$	$k+2$	$k+1$	k

Table 5. Required internal multiplications and base additions in Toom-Cook 3-way

dimen-sion	# of internal multiplications				# of base additions
	$k-1$	k	$k+1$	$k+2$	eq.(5)
$3k$	0	3	0	2	$38k+18$
$3k+1$	1	0	2	2	$38k+22$
$3k+2$	0	1	2	2	$38k+32$

3.4 HQC GF(2)[x] 곱셈의 최적화

3.2, 3.3 절에서 소개한 카라추바 및 툼-쿡 알고리즘 구현물의 곱셈과 덧셈 연산 수를 모두 정리하면 〔Table 6〕과 같다.

본 절에서는 〔Table 6〕을 기준으로 최적의 조합을 찾아, HQC 내 GF(2)[x] 곱셈의 최적화 방안을 제시한다.

GF(2)[x] 곱셈은 단위 곱셈과 단위 덧셈으로 이루어진 연산이다. 이때 256-bit XOR 연산인 단위 덧셈과 256-bit carry-less 곱셈인 단위 곱셈에 드는 클럭 사이클이 항상 일정하다고 가정하면, 단위 곱셈과 단위 덧셈의 개수가 GF(2)[x] 곱셈의 성능을 결정한다고 할 수 있다. 따라서 본 논문에서는 〔Table 6〕을 기준으로 가중치 w, 곱셈의 수 N_{mul}와 단위 덧셈의 수 N_{add}에 대해, $N_{mul} \times w + N_{add}$을 최소화하는 카라추바, 툼-쿡 알고리즘의 조합을 찾아 GF(2)[x] 곱셈을 최적화한다. 그러나 load, store 등의 명령어 또한 실행되기 때문에, 가중치 w를 단순히 단위 곱셈과 XOR의 클럭 사이클 비율로 설정하는 것은 적절하지 않다. 또한 AVX2 명령어 집합을 사용하는 환경의 경우 캐시 메모리의 존재로 인해 가중치 w는 더욱 확정하기 어렵다. 따라서 본 절에서는 가중치 w를 확정하지 않고, $1 \le w \le 10000$ 에 대해 각 가중치에 대한 최적의 조합을 찾는다. 그리고 찾은 조합에 대한 구현물의 성능을 측정하여 최적의 조합을 결정함으로써 최적화를 진행한다.

Table 6. Required internal multiplications and base additions (2-, 3-, 5-Karatsuba and Toom-Cook 3-way)

Alg.	dimen-sion	# of internal multiplications				# of base additions
		$k-1$	k	$k+1$	$k+2$	
2-Karat	$2k$	0	3	0	0	$7k$
	$2k+1$	0	1	2	0	$7k+3$
3-Karat	$3k$	0	6	0	0	$20k$
	$3k+1$	0	3	3	0	$20k+8$
	$3k+2$	0	1	5	0	$20k+14$
5-Karat	$5k$	0	15	0	0	$80k$
TC3	$3k$	0	3	0	2	$38k+18$
	$3k+1$	1	0	2	2	$38k+22$
	$3k+2$	0	1	2	2	$38k+32$

3.4.1 hqc-128 내 $GF(2)[x]$ 곱셈 최적화

hqc-128의 경우 17668차 다항식 곱셈이 이루어지며, 이는 256-bit 단위로 $\lceil 17669/256 \rceil = 70$ 차원 곱셈이다. 70차원 곱셈에 대해 $N_{mul} \times w + N_{add}$을 최소화하는 카라추바 및 툼-쿡 알고리즘의 조합을 가중치 w의 범위에 따라 구하고, 이를 트리로 나타내면 다음과 같다.

각 트리는 각 노드의 값은 곱셈의 차원이고, 하위 노드는 상위 노드 차원의 곱셈 알고리즘에 필요한 내부 곱셈의 차원을 의미한다. 이때 사용하는 곱셈 알고리즘의 종류는 간선 위에 표기한다. 같은 차원의 다항식 곱셈은 같은 자식노드를 가지기 때문에, 중복된 경우 하나의 노드에만 자식노드를 표기하고 다른 중복된 노드는 표기를 생략한다. 1차원 곱셈은 단위 곱셈을 의미한다.

3.4.1.1 case 1: $0 \le w \le 14$ 인 경우

$0 \le w \le 14$인 경우 $N_{mul} \times w + N_{add}$을 최소화하는 70차원 곱셈에 대한 트리는 Fig. 6과 같이 나타난다. 이 경우 849개의 단위 곱셈과 5803개의 단위 덧셈이 진행된다.

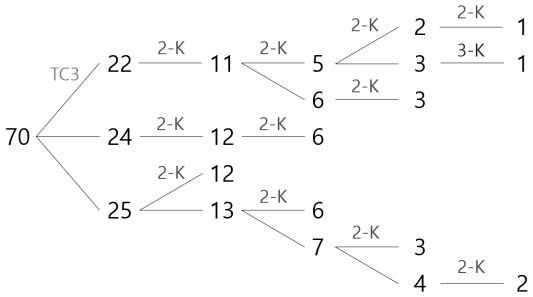

Fig. 6. Tree for 70-dimensional polynomial multiplication with $0 \le w \le 14$ (hqc-128)

3.4.1.2 case 2: $15 \le w$ 인 경우

$15 \le w$인 경우 $N_{mul} \times w + N_{add}$을 최소화하는 70차원 곱셈에 대한 트리는 Fig. 7과 같이 나타난다. 이 경우 840개의 단위 곱셈과 5929개의 단위 덧셈이 진행된다.

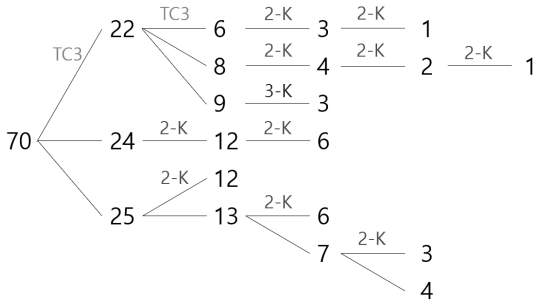

Fig. 7. Tree for 70-dimensional polynomial multiplication with $15 \le w$ (hqc-128)

3.4.2 hqc-192 내 $GF(2)[x]$ 곱셈 최적화

hqc-192의 경우 35850차 다항식 곱셈이 이루어지며, 이는 256-bit 단위로 $\lceil 35851/256 \rceil = 141$ 차원 곱셈이다. 141차원 곱셈에 대해 $N_{mul} \times w + N_{add}$을 최소화하는 카라추바 및 툼-쿡 알고리즘의 조합을 가중치 w의 범위에 따라 구하고, 이를 트리로 나타내면 다음과 같다.

3.4.2.1 case 1: $0 \le w \le 2$ 인 경우

$0 \le w \le 2$인 경우 $N_{mul} \times w + N_{add}$을 최소화하는 141차원 곱셈에 대한 트리는 Fig. 8과 같이 나타난다. 이 경우 2517개의 단위 곱셈과 17212개의 단위 덧셈이 진행된다.

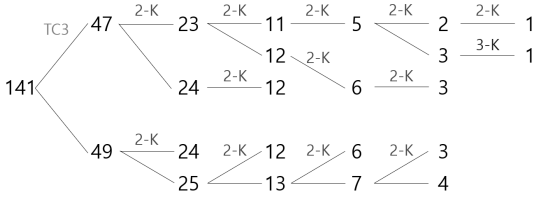

Fig. 8. Tree for 141-dimensional polynomial multiplication with $0 \le w \le 2$ (hqc-192)

3.4.2.2 case 2: $3 \le w$ 인 경우

$3 \le w$인 경우 $N_{mul} \times w + N_{add}$을 최소화하는 141차원 곱셈에 대한 트리는 Fig. 9와 같이 나타난다. 이 경우 2298개의 단위 곱셈과 17692개의 단위 덧셈이 진행된다.

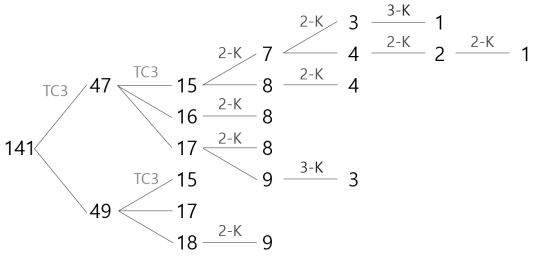

Fig. 9. Tree for 141-dimensional polynomial multiplication with $3 \leq w$ (hqc-192)

3.4.3 hqc-256 내 $GF(2)[x]$ 곱셈 최적화

hqc-256의 경우 57636차 다항식 곱셈이 이루어지며, 이는 256-bit 단위로 $\lceil 57637/256 \rceil = 226$ 차원 곱셈이다. 226차원 곱셈에 대해 $N_{mul} \times w + N_{add}$을 최소화하는 카라추바 및 툼-쿡 알고리즘의 조합을 가중치 w의 범위에 따라 구하고, 이를 트리로 나타내면 다음과 같다.

3.4.3.1 case 1: $0 \leq w \leq 14$ 인 경우

$0 \leq w \leq 14$인 경우 $N_{mul} \times w + N_{add}$을 최소화하는 226차원 곱셈에 대한 트리는 Fig. 10과 같이 나타난다. 이 경우 4890개의 단위 곱셈과 36231개의 단위 덧셈이 진행된다.

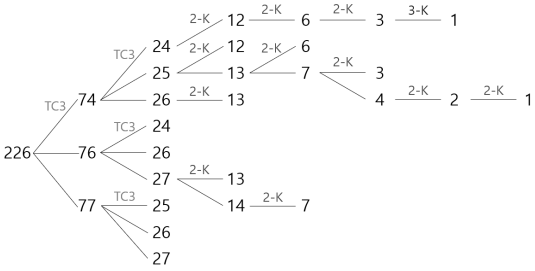

Fig. 10. Tree for 226-dimensional polynomial multiplication with $0 \leq w \leq 14$ (hqc-256)

3.4.3.2 case 2: $15 \leq w$ 인 경우

$15 \leq w$인 경우 $N_{mul} \times w + N_{add}$을 최소화하는 226차원 곱셈에 대한 트리는 Fig. 11과 같이 나타난다. 이 경우 4800개의 단위 곱셈과 37531개의 단위 덧셈이 진행된다.

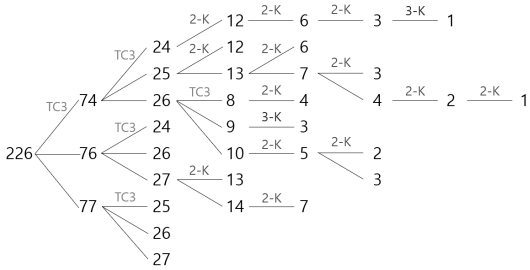

Fig. 11. Tree for 226-dimensional polynomial multiplication with $15 \leq w$ (hqc-256)

IV. 최적화 결과

본 장에서는 HQC의 $GF(2)[x]$ 곱셈 최적화 구현 결과를 제시한다. 3.4절에서 제시한 최적화 트리의 구현 결과, 그리고 NIST PQC 공모전 라운드 4에 제출된 HQC의 AVX2 환경 구현물(2024/02/23, Optimized implementation)의 $GF(2)[x]$ 곱셈을 비교한다. 측정에 사용한 컴파일러는 GNU GCC version 11.3.0이며, 최적화 수준은 O3로 컴파일하였다. 타겟 컴퓨터의 프로세서는 13th Gen Intel(R) Core(TM) i7-13700K (3.40 GHz)이며 OS는 Ubuntu 22.04.2 LTS 이다.

4.1 hqc-128의 그림의 $GF(2)[x]$ 곱셈

hqc-128의 다항식 곱셈에 대해 Fig. 6, Fig. 7에서 제시한 최적화 트리를 기반으로 구현한 결과는 〔Table 7〕과 같다. case 1은 $0 \leq w \leq 14$, case 2는 $15 \leq w$인 경우의 최적화 결과이다. 〔Table 7〕의 previous work는 HQC의 최근 제출물 중 AVX2 환경 구현물인 Optimized implementation을 뜻하며, keygen, encaps, decaps는 제안한 $GF(2)[x]$ 곱셈을 hqc-128의 키 생성, 캡슐화, 디캡슐화 과정에 적용한 결과를 뜻한다. 단위는 클럭 사이클이다.

Table 7. Implementation results of hqc-128

	previous work〔1〕	This work	
		case 1	case 2
GF(2)[x] mul	15,134	**14,486**	15,896
keygen	46,772	**45,787**	47,128
encaps	109,914	**107,360**	108,975
decaps	206,529	**201,911**	205,861

〔Table 7〕의 결과를 보면, $0 \le w \le 14$에 해당하는 case 1의 곱셈이 가장 좋은 결과를 보인다는 것을 확인할 수 있다. 또한 case 1의 곱셈은 기존 hqc-128 구현물의 곱셈에 비해 4.5%의 성능 향상이 있고, 이를 hqc-192에 적용하면 키 생성, 캡슐화, 디캡슐화에서 각각 2.2%, 2.4%, 2.3%의 향상을 보인다.

4.2 hqc-192의 그림의 $GF(2)[x]$ 곱셈

hqc-192의 다항식 곱셈에 대해 Fig. 8, Fig. 9에서 제시한 최적화 트리를 기반으로 구현한 결과는 〔Table 8〕과 같다. case 1은 $0 \le w \le 2$, case 2는 $3 \le w$인 경우의 최적화 결과이다. 〔Table 8〕의 previous work는 HQC의 최근 제출물 중 AVX2 환경 구현물인 Optimized implementation을 뜻하며, keygen, encaps, decaps는 제안한 $GF(2)[x]$ 곱셈을 hqc-192의 키 생성, 캡슐화, 디캡슐화 과정에 적용한 결과를 뜻한다. 단위는 클럭 사이클이다.

〔Table 8〕을 통해 $3 \le w$에 해당하는 case 2의 곱셈이 가장 좋은 결과를 보인다는 것을 확인할 수 있다. 또한 case 2의 곱셈은 기존 hqc-192 구현물의 곱셈에 비해 2.5%의 성능 향상이 있고, 이를 hqc-192에 적용하면 키 생성, 캡슐화, 디캡슐화에서 각각 1.6%, 4.2%, 2.6%의 향상을 보인다.

Table 8. Implementation results of hqc-192

	previous work〔1〕	This work	
		case 1	case 2
$GF(2)[x]$ mul	45,641	46,848	**44,538**
keygen	111,874	112,923	**110,080**
encaps	255,247	255,232	**245,054**
decaps	436,914	434,595	**425,889**

4.3 hqc-256의 그림의 $GF(2)[x]$ 곱셈

hqc-256의 다항식 곱셈에 대해 Fig. 10, Fig. 11에서 제시한 최적화 트리를 기반으로 구현한 결과는 〔Table 9〕와 같다. case 1은 $0 \le w \le 14$, case 2는 $15 \le w$인 경우의 최적화 결과이다. 〔Table 9〕의 previous work는 HQC의 최근 제출물 중 AVX2 환경 구현물인 Optimized implementation을 뜻하며, keygen, encaps, decaps는 제안한 $GF(2)[x]$ 곱셈을 hqc-256의 키 생성, 캡슐화, 디캡슐화 과정에 적용한 결과를 뜻한다. 단위는 클럭 사이클이다.

Table 9. Implementation results of hqc-256

	previous work〔1〕	This work	
		case 1	case 2
$GF(2)[x]$ mul	112,984	**86,707**	92,798
keygen	228,694	**201,798**	206,661
encaps	502,587	**438,344**	452,714
decaps	875,627	**772,856**	798,075

〔Table 9〕를 보면, $0 \le w \le 14$에 해당하는 case 1의 곱셈이 가장 좋은 결과를 보인다는 것을 확인할 수 있다. 이 경우 기존 hqc-256 구현물의 곱셈보다 30.3%의 성능 향상이 있고, case 1의 곱셈을 hqc-256에 적용하면 키 생성, 캡슐화, 디캡슐화에서 각각 13.3%, 14.7%, 13.3%의 향상을 보인다.

V. 결 론

본 논문에서는 카라추바, 툼-쿡 알고리즘의 필요 덧셈 및 곱셈의 수를 분석하고, 이를 통해 AVX2 환경에서 가장 효율적인 조합을 찾아 HQC의 $GF(2)[x]$ 곱셈을 최적화하는 방법을 제안하였다. 256-bit 단위 곱셈과 단위 덧셈에 대하여, 각각의 개수를 기준으로, 복잡도가 더 큰 단위 곱셈의 개수에 가중치 w를 두어서 그 합이 최소가 되는 조합을 탐색하였다. 가중치 w의 범위에 따라 최적의 조합이 다르기 때문에, 기존 HQC의 AVX2 환경 구현물(2024/02/23, Optimized implementation)의 곱셈과 함께 각각의 조합을 모두 비교하였다. 그 결과, hqc-128, -192, -256에서 이루어지는 17668, 35850, 57636차 $GF(2)[x]$ 곱셈의 경우 기존 HQC 구현물의 $GF(2)[x]$ 곱셈 대비 4.5%, 2.5%, 30.3%의 성능 향상을 보였고, 이를 각각의 키 생성, 캡슐화, 디캡슐화에 적용하였을 때, 기존에 비해 hqc-128에서는 각각 2.2%, 2.4%, 2.3%, hqc-192에서는 각각 1.6%, 4.2%, 2.6%, hqc-256에서는 각각 13.3%, 14.7%, 13.3%의 성능 향상을 보였다. 본 논문의 방식은 임의 차수의 다항식 곱셈에도 사용될 수 있어, $GF(2)[x]$ 곱셈이 이루어지는 BIKE 등의 다른 암호 알고리즘에 적용할

시 최적화가 가능할 것으로 보인다.

References

[1] Melchor, Carlos Aguilar, et al, "Hamming quasi-cyclic (HQC)," NIST PQC submissions, Round 4, Feb. 2024.

[2] A. Karatsuba, "Multiplication of multidigit numbers on automata," *Soviet physics doklady*, vol. 7, pp. 595-596, Sep. 1963.

[3] A. L. Toom, "The complexity of a scheme of functional elements realizing the multiplication of integers," *Soviet Mathematics Doklady*, vol. 4, no. 3, pp. 714-716, May. 1963.

[4] S. A. Cook and S. O. Aanderaa, "On the minimum computation time of functions," *Transactions of the American Mathematical Society*, vol. 142, pp. 291-314, Aug. 1969.

[5] Gao, Shuhong, Mateer, Todd. "Additive fast Fourier transforms over finite fields," *IEEE Transactions on Information Theory*, vol. 56, no. 12, pp. 6265-6272, Dec. 2010.

[6] R. Avanzi, et al, "Crystals-kyber algorithm specifications and supporting documentation," NIST PQC submissions, Round 3, Feb. 2021.

[7] L. Ducas, et al, "Crystals-Dilithium algorithm specifications and supporting documentation," NIST PQC submissions, Round 3, Oct. 2020.

[8] J. M. Robert, P. Véron, "Faster multiplication over F2[x] using avx512 instruction set and vpclmulqdq instruction," *Journal of Cryptographic Engineering*, vol. 13, no. 1, pp. 37-55, Apr. 2023.

[9] Richard P. Brent, Pierrick Gaudry, Emmanuel Thomé, Paul Zimmermann. "Faster Multiplication in GF(2)[x]." Proceedings of the *ANTS-VIII 2008*, pp.153-166, May. 2008.

〈저자소개〉

장 지 훈 (Jihoon Jang) 학생회원
2023년 2월: 고려대학교 수학과 졸업
2023년 3월~현재: 고려대학교 정보보호대학원 석사과정
〈관심분야〉 후양자암호

이 명 훈 (Myeonghoon Lee) 학생회원
2020년 2월: 고려대학교 수학과 졸업
2020년 3월~현재: 고려대학교 정보보호대학원 석박사 통합과정
〈관심분야〉 후양자암호

김 수 리 (Suhri Kim) 정회원
2014년 2월: 고려대학교 수학과 졸업
2016년 8월: 고려대학교 정보보호대학원 공학석사
2020년 2월: 고려대학교 정보보호대학원 공학박사
2020년 3월~2021년 2월: 고려대학교 정보보호대학원 연구교수
2020년 7월~2021년 2월: K.U. Leuven ESAT/SCD-COSIC 박사후연구원
2021년 3월~현재: 성신여자대학교 수리통계데이터사이언스학부 조교수
〈관심분야〉 공개키 암호시스템, 후양자암호

서 석 충 (Seogchung Seo) 정회원
2011년 8월: 고려대학교 정보보호대학원 박사
2013년 11월: 삼성전자 종합기술원 전문연구원
2014년 4월: 삼성전자 DMC 연구소 책임연구원
2019년 2월: 국가보안기술연구소 선임연구원
2019년 3월~현재: 국민대학교 금융정보보안학과 부교수
〈관심분야〉 암호최적화, 공개키 암호, 암호모듈검증, 네트워크보안

홍 석 희 (Seokhie Hong) 종신회원
2001년: 고려대학교 수학과 박사
1999년 8월~2004년 2월: ㈜시큐리티 테크놀리지 선임연구원
2003년 3월~2004년 2월: 고려대학교 정보보호기술연구센터 선임연구원
2004년 4월~2005년 2월: K.U. Leuven ESAT/SCD-COSIC 박사후연구원
2005년 3월~2013년 8월: 고려대학교 정보보호대학원 부교수
2013년 9월~현재: 고려대학교 정보보호대학원 정교수
〈관심분야〉 대칭키 및 공개키 암호 알고리즘, 부채널 공격 및 대응기법, 디지털포렌식