석사학위논문

SHA-256 해시함수에 대한 양자 회로 depth 최적 구현

2024년

한 성 대 학 교 대 학 원

I T 융 합 공 학 과

I T 융 합 공 학 전 공

임 세 진

석 사 학 위 논 문
지도교수 서화정

SHA-256 해시함수에 대한 양자 회로 depth 최적 구현

Depth Optimization of Quantum Circuits for SHA-256 Hash Function

2023년 12월 일

한 성 대 학 교 대 학 원

I T 융 합 공 학 과

I T 융 합 공 학 전 공

임 세 진

석 사 학 위 논 문
지도교수 서화정

SHA-256 해시함수에 대한 양자 회로 depth 최적 구현

Depth Optimization of Quantum Circuits
for SHA-256 Hash Function

위 논문을 공학 석사학위 논문으로 제출함

2023년 12월 일

한 성 대 학 교 대 학 원

I T 융 합 공 학 과

I T 융 합 공 학 전 공

임 세 진

임세진의 공학 석사학위 논문을 인준함

2023년 12월 일

심사위원장 박 명 서 (인)

심 사 위 원 서 화 정 (인)

심 사 위 원 이 웅 희 (인)

국 문 초 록

SHA-256 해시함수에 대한
양자 회로 depth 최적 구현

한 성 대 학 교 대 학 원
I T 융 합 공 학 과
I T 융 합 공 학 전 공
임 세 진

SHA-256 해시함수는 데이터 무결성, 인증, 디지털 서명 등 다양한 분야에서 사용되고 있다. SHA-256은 128-bit의 보안강도를 가진다고 평가되지만, 양자컴퓨터가 빠르게 발전함에 따라 안전성을 위협받고 있다. 양자컴퓨터에서 동작하는 전수조사 가속화 알고리즘인 Grover 알고리즘을 사용하면 Brute-force 공격의 시간 복잡도가 $O(N)$에서 $O(\sqrt{N})$으로 줄어들기 때문이다. Grover 알고리즘을 SHA-256에 적용하면 2^{128}번의 입력 시도가 아닌 2^{64}번의 입력 시도만으로 Brute-force 공격을 수행할 수 있게 되어, 보안강도가 128-bit에서 절반인 64-bit로 감소하게 된다. 이러한 보안 위협에 대비하기 위해 현재 사용되는 암호에 대한 양자 공격 자원을 추정하여 양자컴퓨터 상에서의 보안강도를 추정하는 연구가 활발히 수행되고 있다. 특정 암호에 대해 Grover 알고리즘이 적용된 공격 회로를 동작시킬 수 있는 고성능의 양자컴퓨터가 개발되는 시점이 해당 암호의 보안강도가 절반이 되는 시점이라고 볼 수 있다. Grover 알고리즘을 적용하기 위해서는 공격 대상 알고리즘을 양자

회로로 구현해야하며, 회로를 최적 구현하여 사용되는 양자 자원을 최소화할수록 해킹 비용도 감소하게 된다. NIST는 AES와 SHA2, SHA3에 대한 양자 공격 회로에 필요한 양자 비용을 기준으로, 특정 암호의 양자 공격 회로 비용과 비교하여 해당 암호가 양자컴퓨터에서 갖는 보안강도를 추정하고 있다. NIST는 양자 공격의 복잡도를 양자 회로의 크기(사용된 모든 양자 게이트 수 × 전체 회로 depth)로 측정한다. 따라서 SHA-256의 공격 회로에 대해 효율적이고 정확한 자원을 추정하는 것은 암호의 안전성 확인을 위한 매우 중요한 작업임을 알 수 있다. 또한, 양자 공격의 복잡도 감소를 위해서는 전체 회로 깊이(depth)를 최적화해야함을 알 수 있다. 또한 Toffoli 게이트 분해 시 사용되는 T 게이트는 높은 구현 비용을 차지하기 때문에 T 게이트와 관련된 자원인 T 게이트 수와 T-depth를 최적화하는 것도 매우 중요하다. Toffoli-depth의 최적화는 T-depth의 최적화로 이어지며, 최적화된 Toffoli 게이트 분해 기법을 사용할수록 T 게이트 관련 자원을 줄일 수 있다. 따라서 본 논문에서는 양자컴퓨터상에서 보안강도 기준이 되는 알고리즘 중 하나인 SHA-256에 대해 T-depth를 포함하여 최적화된 depth를 가지는 양자 회로를 제안하고, 이를 기반으로 Grover 공격 비용을 추정한다.

【주요어】 양자컴퓨터, Grover 알고리즘, 양자 회로 구현, depth 최적화, T-depth 최적화, SHA-256 해시함수, 역상 공격, Carry-save 덧셈기

목 차

표 목 차

그 림 목 차

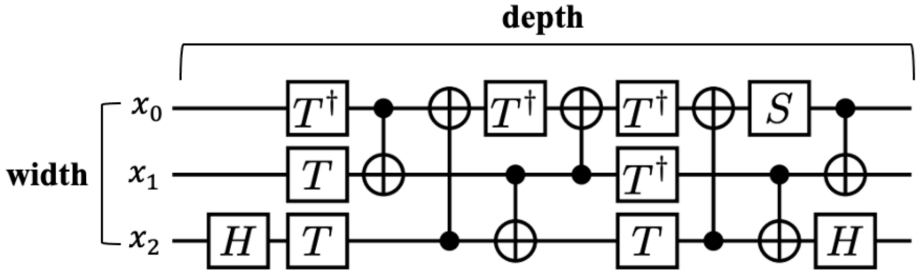
알 고 리 즘 목 차

수 식 목 차

Ⅰ. 서 론

1.1 양자컴퓨터

양자컴퓨터는 양자 역학의 특성을 활용하여 기존의 슈퍼컴퓨터가 할 수 없었던 복잡한 연산을 다항 시간 안에 수행할 수 있는 차세대 컴퓨터이다. 기존 고전컴퓨터의 기본 연산 단위인 비트와 같이 큐비트(qubit)를 기본 연산 단위로 사용하여 빠른 연산이 가능하다. 큐비트는 양자 현상인 얽힘(entanglement)과 중첩(superposition) 성질을 가진다. 큐비트들이 서로 얽혀 있는 경우, 어떤 큐비트의 상태가 변화하면 다른 큐비트의 상태에도 영향을 줄 수 있는데 이 성질을 얽힘이라고 한다. 중첩은 0과 1의 상태를 동시에 확률적으로 가질 수 있는 상태를 말한다. 고전컴퓨터는 n개의 비트로 n개의 값을 나타낼 수 있지만, 양자컴퓨터는 중첩 성질에 의해 n개의 큐비트로 2^n개의 값을 동시에 나타낼 수 있다. 따라서 모든 경우의 수에 대해 순차적으로 연산하는 것이 아니라 동시에 연산할 수 있어 고전컴퓨터에 비해 연산 속도가 획기적으로 빠르다.



[그림 1-1] 양자 회로의 width와 depth

양자컴퓨터에서는 큐비트의 상태를 변경하고 제어하기 위해 고전컴퓨터의

논리게이트와 유사한 양자 게이트를 사용한다. 이러한 양자 게이트를 사용하여 설계된 양자 회로는 [그림 1-1]과 같이 너비(width)와 깊이(depth)를 갖게 된다. 여기에서 width는 물리적인 큐비트 수를 나타내고, depth는 입력에서 출력까지의 경로에 대한 최대 길이, 즉 연속적인 양자 게이트 연산의 수를 의미하며 이는 곧 물리적 연산 시간을 의미한다.

양자컴퓨터를 활용하면 인공지능과 빅데이터 처리 및 시뮬레이션과 같은 최첨단 분야에서 높은 연산 부하로 인해 발생했던 난제를 해결할 수 있기 때문에, 구글, IBM과 같은 세계적인 기업을 선두로 하여 활발히 개발되고 있다. 2019년 10월 구글은 오류율을 대폭 낮춘 54 큐비트 양자컴퓨터인 시커모어(Sycamore)를 공개하였으며, 같은 시기에 국제학술지 Nature에 고전컴퓨터 상에서의 난제인 난수 증명 문제를 3분 20초 만에 해결하여 양자 우월성을 달성했음을 발표하였다. 이는 양자컴퓨터가 현존하는 슈퍼컴퓨터의 성능을 능가할 수 있음을 세계 최초로 증명한 결과였다. 구글의 양자컴퓨터 실용성 검증은 양자컴퓨팅의 가능성에 회의적이었던 연구자들에게 미래의 양자컴퓨터 시대를 준비하는 연구에 집중할 수 있는 중요한 기반을 제공한 것으로 해석될 수 있다. 경쟁기업인 IBM은 지난 2022년 11월 세계 최초로 433 큐비트 양자컴퓨터인 오스프리(Osprey)를 발표했다. IBM은 새로운 시대를 열겠다는 포부를 가지고 양자컴퓨터 개발 로드맵에 따라 매년 큐비트 목표를 달성한 양자컴퓨터를 발표하고 있으며, 오는 2025년까지 4158 큐비트를 달성하는 것을 목표로 하고 있다.

대부분의 양자컴퓨터 개발 연구는 양자컴퓨터에서 운용할 수 있는 큐비트 수의 증가에 초점을 맞추고 있지만, 양자컴퓨터의 성능은 단순히 큐비트의 개수만으로 평가되어서는 안 된다. 2022년 Nature physics의 논문에서 IBM 양자컴퓨터에 대해 width와 depth에 따른 양자 회로 동작 벤치마킹을 수행한 결과, width와 depth가 커질수록 회로의 오류율이 증가함을 보였다. 이를 통해 안정적으로 양자 회로를 동작시키기 위해서는 큐비트 수뿐만 아니라 양자 회로의 depth도 고려해야 함을 알 수 있다. depth도 고려하여 지난 2022년 IBM은 27 큐비트 Falcon 계열 양자컴퓨터(Prague)에서 9 큐비트와 회로 depth가 9인 양자 회로를 성공적으로 실행하였다. 안정성 확보를 위해 100

큐비트 이상의 양자컴퓨터로 실행할 수 있는 회로의 depth를 늘려가는 것을 목표로 하고 있다.

한편, 암호학 측면에서 양자컴퓨터는 큰 위협이 되는데, 양자 알고리즘에 의해 현재 사용하는 암호의 보안성이 낮아지기 때문이다. 대표적인 양자 알고리즘에는 Shor 알고리즘과 Grover 알고리즘이 있다. 이 알고리즘을 동작시킬 수 있는 고성능의 양자컴퓨터가 개발되는 시점에 공개키 암호의 경우, RSA와 ECC(Elliptic Curve Cryptography)의 보안성이 무너지게 되며, 대칭키 암호의 경우, 고전컴퓨터에서 보장되던 보안강도가 절반으로 감소하게 된다.

1994년, Peter Shor는 양자 알고리즘을 통해 소인수 분해와 이산 로그 문제를 다항 시간 안에 해결할 수 있음을 증명하였다. 현재 널리 사용되는 공개키 암호 시스템인 RSA와 ECC는 각각 소인수 분해와 이산 로그 문제의 복잡성에 기반을 두고 있다. 따라서 고성능 양자컴퓨터가 등장하면 Shor 알고리즘에 의해 해당 암호가 기반하고 있는 난제를 풀 수 있게 되어 더 이상 사용이 불가능하다. 따라서 NIST는 이러한 암호 시스템의 붕괴 위협에 대비하여 양자컴퓨터가 등장하더라도 안전한 암호인 양자내성암호에 대한 공모전을 개최하여 새로운 공개키 암호 표준화를 진행하였다. 현재 공개키/KEM(Key Encapsulation Mechanism)의 경우 격자 기반 암호인 CRYSTAL-Kyber가 표준화되었으며, 전자서명에는 격자 기반의 CRYSTAL-Dilithium, FALCON과 해시 기반의 SPHINCS+가 표준화된 상태이다. NIST는 공개키/KEM 분야에 격자 기반 문제가 아닌 다른 수학적 문제에 기반 한 암호 알고리즘을 표준화하고자 추가로 Round 4를 진행하고 있다.

1996년, Lov Grover는 N개의 데이터에서 특정 데이터를 찾아내는 양자 알고리즘을 제안하였다. Grover 알고리즘은 Shor 알고리즘 수준의 기하급수적인 속도 향상을 보여주진 않지만, 제곱근 속도 향상을 보여준다. 고전컴퓨터에서 Brute-force 공격을 수행하면 2^n번의 시도가 필요하지만, Grover 알고리즘을 활용하면 $2^{n/2}$번만으로도 공격에 성공할 수 있다. 따라서 n-bit 보안강도의 대칭키 암호를 $n/2$-bit 보안강도로 감소시킬 수 있다. 해시함수의 경우도 마찬가지로 보안강도를 절반으로 감소시킨다. 대칭키 암호의 경우 키 길이를, 해시함수의 경우 해시 출력 길이를 2배 증가시킴으로써 기존의 보안강

도를 유지할 수 있지만, NIST 공모전의 양자내성암호 알고리즘의 내부에서도 대칭키 암호 요소들이 활용되므로, 이러한 보안 위협은 양자내성암호의 보안성을 우회하는 요인이 될 수 있다. 이에 AES 암호를 포함한 다양한 대칭키 암호 및 해시함수를 대상으로 Grover 알고리즘을 적용한 양자 암호 분석 연구들이 수행되고 있다. 양자 공격 회로에 대한 정확한 자원을 추정하는 것은 양자컴퓨터 상에서 암호가 가지는 안전성을 평가하기 위한 매우 중요한 작업이다.

1.2 양자 회로 depth 최적화의 필요성

NIST는 [표 1-1]과 같이 AES와 SHA2, SHA3에 대한 양자 공격 회로에 필요한 양자 비용을 기준으로, 특정 암호의 양자 공격 회로 비용과 비교하여 해당 암호가 양자컴퓨터에서 가지는 보안강도를 추정하고 있다[1]. NIST는 양자 공격의 복잡도를 양자 회로의 크기(사용된 모든 양자 게이트 수 × 전체 회로 depth)로 측정한다. 보안강도 1, 3, 5에 해당하는 AES에 대한 양자 공격 복잡도의 경우, 현재 2016년 Grassl의 연구 결과에서 더욱 최적화된 결과인 2020년도 Jaques의 AES에 대한 양자 공격 회로 연구를 인용하고 있다. 보안강도 1에 대한 공격 비용은 Grover 알고리즘을 사용하는 키 복구 회로의 총 게이트 수 × 전체 회로 depth인 2^{157}이며, 보안강도 3과 5는 각각 2^{221}과 2^{285}가 된다.

이에 반해 SHA2와 SHA3를 대상으로 하는 보안강도 2, 4의 양자 공격 비용은 아직까지 채택되지 않았다. 따라서 본 논문에서는 SHA-256을 대상으로 하여 최적화된 양자 회로 제안하고, Grover 알고리즘을 적용하여 공격 회로에 대해 정확한 자원을 추정함으로써 보안강도 2에 해당하는 양자 공격 복잡도를 제시하고자 한다. 또한 전체 회로 깊이(depth)를 줄임으로써 양자 공격의 복잡도를 낮출 수 있으므로, SHA-256 양자 회로의 depth를 줄이는 것

1) https://csrc.nist.rip/csrc/media/Projects/pqc-dig-sig/documents/call-for-proposals-dig-sig-sept-2022.pdf

에 초점을 맞춰 최적화하였다.

[표 1-1] NIST 양자 후 보안 기준

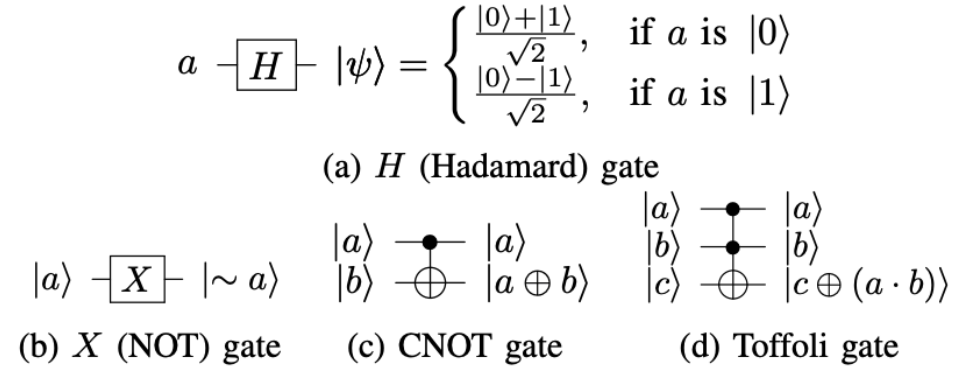
보안강도	요구 사항
Level 1	AES-128 이상의 양자 해킹을 위한 공격 복잡도
	2^{157}/MAXDEPTH 양자 게이트 또는 2^{143} 고전 게이트
Level 2	SHA256/SHA3-256 이상의 양자 해킹을 위한 공격 복잡도
	2^{146} 고전 게이트 (SHA3-256)
Level 3	AES-192 이상의 양자 해킹을 위한 공격 복잡도
	2^{221}/MAXDEPTH 양자 게이트 또는 2^{207} 고전 게이트
Level 4	SHA384/SHA3-384 이상의 양자 해킹을 위한 공격 복잡도
	2^{210} 고전 게이트 (SHA3-384)
Level 5	AES-256 이상의 양자 해킹을 위한 공격 복잡도
	2^{285}/MAXDEPTH 양자 게이트 또는 2^{272} 고전 게이트

II. 관련 연구

2.1 양자 회로 구현

2.1.1 양자 게이트

양자컴퓨터는 큐비트와 양자 게이트로 구성된 양자 회로를 통해 계산을 수행한다. 고전컴퓨터에서 NOT, XOR, AND, OR과 같은 논리 게이트를 통해 비트 값을 변경하듯이, 양자컴퓨터에서도 양자 게이트를 통해 큐비트의 상태를 변경하게 된다. 양자 회로 구현 시, 주로 사용되는 양자 게이트는 아래 [그림 2-1]과 같다. Hadamard 게이트라고도 불리는 H 게이트는 큐비트의 상태를 |0⟩과 |1⟩의 중첩 상태로 만들어주며, X 게이트, CNOT 게이트, Toffoli 게이트는 각각 고전컴퓨터의 NOT, XOR, AND 연산을 대체할 수 있다. X 게이트는 큐비트는 상태를 반전시키며, CNOT 게이트는 두 개의 큐비트를 대상으로 하는데, 제어 큐비트인 |a⟩가 1일 때만 대상 큐비트인 |b⟩의 상태가 반전된다. Toffoli 게이트는 두 개의 제어 큐비트가 모두 1일 때만 대상 큐비트의 값을 반전시킨다.

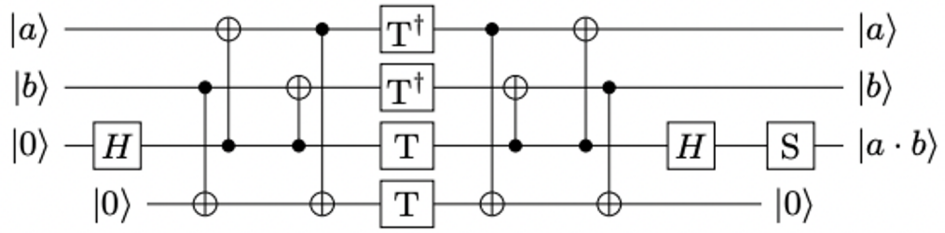


[그림 2-1] 주요 양자 게이트

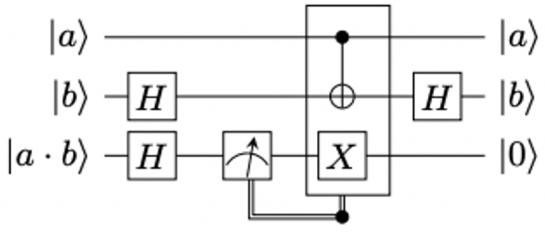
2.1.2 Toffoli 게이트 분해

Toffoli 게이트는 3개의 큐비트를 조작하는 복잡한 연산이므로 구현 비용이 높다. 따라서 Toffoli 게이트를 T 게이트가 포함된 더 간단한 게이트들의 조합으로 분해함으로써 구현 비용을 줄일 수 있으며, 다양한 분해 기법들이 연구되고 있다. 이때 T 게이트는 상대적으로 높은 구현 비용을 가지기 때문에 T 게이트 수와 T-depth를 최소화하는 것이 중요하다.

본 논문에서는 Toffoli 게이트 분해 기법으로 Jaques[2]가 가장 최근에 개선한 AND 게이트를 사용하였다. Jaques의 AND 게이트를 적용한 AES 양자 회로의 경우 1.2절에서 살펴본 것과 같이, NIST의 AES의 양자 공격 비용으로 채택되어 양자 후 보안강도 표준으로 사용되고 있다. 본 논문에서 사용한 AND 게이트는 채택 이후 Jaques가 depth를 최적화하여 개선한 회로이다. 대상 큐비트가 |0⟩인 것만 제외하면 Toffoli 게이트와 동일하게 동작하며, 구체적인 회로는 [그림 2-2]와 같다.



(a) AND gate



(b) AND† gate

[그림 2-2] AND 게이트와 AND† 게이트

2) Implementing Grover oracles for quantum key search on AES and LowMC., https://eprint.iacr.org/2019/1146.pdf

대부분의 분해 기법은 7개의 T 게이트가 사용되며, ancilla 큐비트가 0개, 1개, 4개일 때 각각 3, 2, 1의 T-depth를 가진다. 그에 반해 AND 게이트는 T 게이트를 4개로 줄였으며, 1개의 ancilla 큐비트를 사용하면서 T-depth 1을 가진다. AND† 게이트는 이 전에 AND 게이트를 적용한 연산을 해제할 때 AND 게이트를 reverse하지 않고도 같은 결과를 낼 수 있는 게이트이다. Measure 기반의 AND† 게이트를 사용하여 연산 해제를 할 경우, T 게이트를 사용하지 않으면서도 매우 작은 depth를 갖게 된다. 이는 Toffoli 게이트 쌍으로 이루어진 연산의 경우, T 게이트 비용을 절반으로 줄일 수 있다는 것을 의미한다. 따라서 AND 게이트는 T 게이트 관련 비용을 줄이는 기법 중 가장 강력한 기법이라고 할 수 있다.

2.1.3 양자 덧셈기

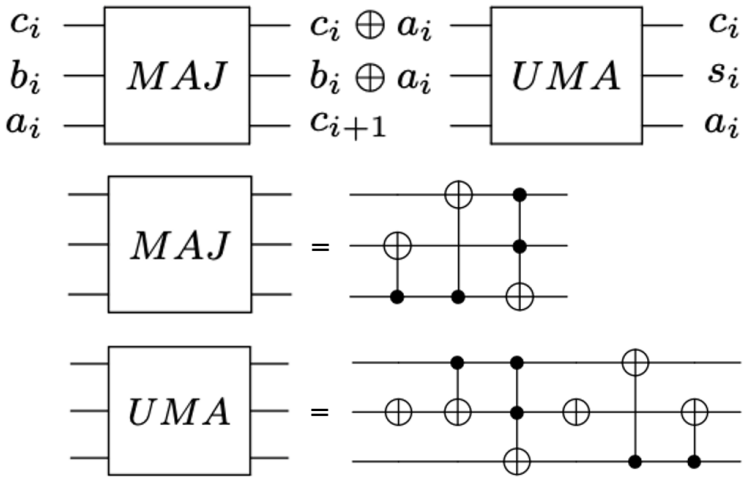
양자 회로를 통해 알고리즘을 구현하기 위해서는 덧셈과 같은 기본 연산도 양자 회로로 구현되어야 한다. 일반적인 n-bit 양자 덧셈기는 최상위 비트의 캐리도 고려하여 $(n+1)$-bit의 결과를 반환한다. 하지만, SHA-256과 같은 암호 알고리즘에서 사용되는 덧셈은 대부분 최상위 비트의 캐리를 고려하지 않는 mod 2^n상에서의 덧셈기 회로를 사용하며 SHA-256의 경우 mod 2^{32}상에서 덧셈을 수행한다. 따라서 알고리즘에 대한 수정이 필요하며, 이에 따라 사용되는 양자 자원의 수도 달라진다. SHA-256의 주 연산은 덧셈 연산이며, 가장 많은 자원을 차지하는 Critical Path도 덧셈 연산이다. 따라서 최적화된 덧셈기를 최대한 병렬로 사용하는 것이 SHA-256 최적화 구현에 굉장히 중요한 요소이다. 2.1.3절에서는 SHA-256 연산에서 가장 효율적인 덧셈기를 찾기 위해 주요 양자 덧셈기의 알고리즘을 mod 2^n 덧셈을 수행하도록 일부 수정하여 살펴보고, 성능을 비교하였다.

2.1.3.1 이항 양자 덧셈기

피연산자가 2개인 이항 양자 덧셈기는 poly-depth를 가지는 Ripple-carry 덧셈기와 log-depth를 가지는 Carry-lookahead 덧셈기로 나눌 수 있다. Carry-lookahead 덧셈기는 캐리 연산을 병렬로 수행함으로써 log-depth를 갖게

된다. 또한 덧셈 결과를 저장하는 위치에 따라 피연산자 중 하나에 저장하는 in-place 구현과 피연산자를 제외한 결과용 큐비트에 저장하는 out-of-place 구현이 있다. 대표적으로 많이 사용되는 이항 양자 덧셈기에는 Cuccaro 덧셈기(CDKM 덧셈기), Takahashi 덧셈기, Draper 덧셈기가 있다.

1) Cuccaro 덧셈기는 0으로 초기화된 하나의 보조 큐비트 X를 초기 캐리 비트(c_0)로 사용하여 덧셈을 수행하며, 두 입력인 a, b 중 b에 덧셈 결과를 저장하는 in-place 형태의 Ripple-carry 덧셈기이다. [그림 2-3]은 Cuccaro 덧셈기를 구성하는 MAJ 게이트와 UMA 게이트 회로를 나타낸다. UMA 게이트는 MAJ 게이트에서 수행한 연산을 해제하여 초기값으로 되돌리고 덧셈 결과를 b에 저장해 준다. 이 과정에서 보조 큐비트도 초기값으로 되돌아가기 때문에 이후 연산에서 재사용할 수 있다.



[그림 2-3] MAJ와 UMA 게이트 회로

[알고리즘 2-1]은 Cuccaro 덧셈기 논문에서 제안한 덧셈기 회로의 알고리즘을 mod 2^n으로 수정한 결과이며, 같은 라인에 속한 연산은 병렬로 수행된다. 최상위 캐리 비트를 고려하지 않으므로 보조 큐비트 z가 사용되지 않으며, b_{n-1}이 최상위 비트가 된다. 따라서 수정된 부분은 다음과 같다. 먼저 n-bit 덧셈 대신 $(n-1)$-bit 덧셈을 기존 덧셈기 알고리즘을 사용하여 구현하

고, z에 수행되는 연산을 b_{n-1}로 변경한다. 마지막으로 $B_{n-1} \oplus = A_{n-1}$ 연산을 추가하면 mod 2^n 덧셈에 대한 알고리즘이 완성된다. 이때, $B_{n-1} \oplus = A_{n-1}$ 연산을 7, 8번 라인에 추가하는 경우 B_{n-1}에 접근하는 연산이 겹치게 되므로 병렬 연산이 수행되지 않아 전체 깊이가 늘어나게 된다. 따라서 16번 라인에 추가하여 병렬로 수행되도록 구현하였다.

Algorithm 1 : Cuccaro adder (Addition mod 2^n)

Input: $A_i = a_i,\ B_i = b_i,\ X = 0$

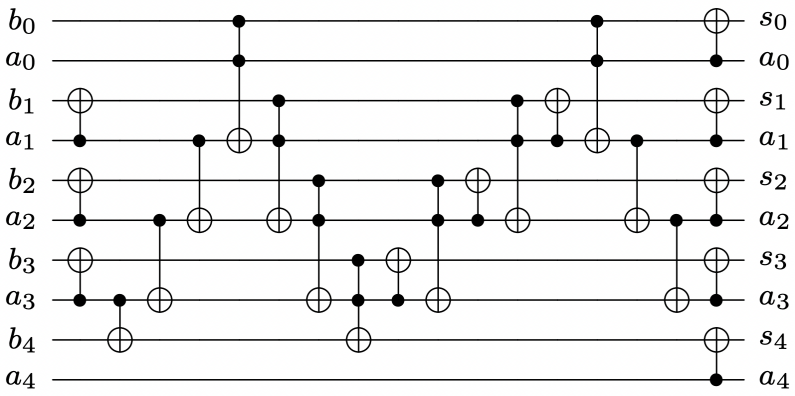
Output: $A_i = a_i,\ B_i = s_i,\ X = 0$

1: for $i = 1$ to $n-2$: $B_i \oplus = A_i$
2: $X \oplus = A_1$
3: $X \oplus = A_0 B_0$; $A_1 \oplus = A_2$
4: $A_1 \oplus = X B_1$; $A_2 \oplus = A_3$
5: for $i = 2$ to $n-3$:
6: $A_i \oplus = A_{i-1} B_i$; $A_{i+1} \oplus = A_{i+2}$
7: $A_{n-3} \oplus = A_{n-4} B_{n-3}$; $B_{n-1} \oplus = A_{n-2}$
8: $B_{n-1} \oplus = A_{n-3} B_{n-2}$; for $i = 1$ to $n-3$: $NOT(B_i)$
9: $B_1 \oplus = X$; for $i = 2$ to $n-2$: $B_i \oplus = A_{i-1}$
10: $A_{n-3} \oplus = A_{n-4} B_{n-3}$
11: for $i = n-4$ down to 2:
12: $A_i \oplus = A_{i-1} B_i$; $A_{i+1} \oplus = A_{i+2}$; $NOT(B_{i+1})$
13: $A_1 \oplus = X B_1$; $A_2 \oplus = A_3$; $NOT(B_2)$
14: $X \oplus = A_0 B_0$; $A_1 \oplus = A_2$; $NOT(B_1)$
15: $X \oplus = A_1$
16: for $i = 0$ to $n-1$: $B_i \oplus = A_i$

[알고리즘 2-1] 수정된 Cuccaro 덧셈기 알고리즘 ($n \geq 4$)

2) Takahashi 덧셈기는 Cuccaro 덧셈기의 MAJ 게이트를 두 개의 CNOT 게이트로 구성된 부분과 Toffoli 게이트 부분으로 나누어 적용함으로써 보조 큐비트 없이 덧셈을 수행하도록 개선하였다. [그림 2-4]는 $n = 5$일 때 mod 2^5 상에서 덧셈을 수행하도록 수정한 Takahashi 덧셈기의 회로이다. Cuccaro 덧셈

기에서 알고리즘을 수정한 것과 동일한 방식으로 수정하였다.



[그림 2-4] mod 2^5에 대해 수정된 Takahashi 덧셈기 회로

3) Draper 덧셈기는 대표적인 Carry-lookahead 덧셈기이며, 논문에서 in-place와 out-of-place 회로를 모두 제시한다. 캐리 연산을 병렬로 수행하기 위해 보조 큐비트가 사용되는데, in-place의 경우 $-2+2n-w(n-1)-\lfloor \log(n-1) \rfloor$ 개가 사용되며 덧셈 연산 이후 모두 재사용이 가능하다. out-of-place는 in-place보다 1개의 보조 큐비트를 더 사용되는데, 이 중 n개는 덧셈 결과를 저장하기 위한 큐비트이므로 이를 제외한 나머지 $-1+n-w(n-1)-\lfloor \log(n-1) \rfloor$ 개만 재사용이 가능하다. 이때, 보조 큐비트 개수는 mod 2^n 덧셈을 기준으로 하며, $w(n)$은 [수식 2-1]과 같다.

캐리를 미리 계산하는 작업은 하위 비트의 캐리 값에 따른 캐리 발생 여부를 판단하는 P (Propogate) 연산과, 하위 비트의 캐리 값에 관계없이 반드시 캐리가 발생하는지 여부를 판단하는 G (Generate) 연산을 통해 수행된다. Draper 덧셈기에서는 P, G, C, P^{-1} 라운드를 통해 P, G 연산을 수행하여 캐리를 병렬로 계산할 수 있도록 알고리즘을 구성하였다.

$$w(n) = n - \sum_{k=1}^{\infty} \left\lfloor \frac{n}{2^k} \right\rfloor$$

[수식 2-1] $w(n)$

Algorithm 2 : mod 2^n Draper adder (out-of-place)

Input: $A_i = a_i,\ B_i = b_i,\ Z_i = 0,\ ancilla_k = 0$

Output: $A_i = a_i,\ B_i = b_i,\ Z_i = s_i,\ ancilla_k = 0$

1: for $i = 0$ to $n-2$: $Z_{i+1} \oplus = A_i B_i$

2: for $i = 1$ to $n-2$: $B_i \oplus = A_i$

3: $P,\ G,\ C,\ P^{-1}$-rounds $\langle\ n \rightarrow (n-1)\ \rangle$

4: for $i = 0$ to $n-2$: $Z_i \oplus = B_i$

5: $Z_0 \oplus = A_0$; for $i = 1$ to $n-2$: $B_i \oplus = A_i$

6: $Z_{n-1} \oplus = A_{n-1}$; $Z_{n-1} \oplus = B_{n-1}$

[알고리즘 2-2] 수정된 out-of-place Draper 덧셈기 알고리즘

Algorithm 3 : mod 2^n Draper adder (in-place)

Input: $A_i = a_i,\ B_i = b_i,\ ancilla1_{i-1} = 0,\ ancilla2_k = 0$

Output: $A_i = a_i,\ B_i = s_i,\ ancilla1_{i-1} = 0,\ ancilla2_k = 0$

1: for $i = 0$ to $n-2$: $ancilla1_i \oplus = A_i B_i$

2: for $i = 0$ to $n-1$: $B_i \oplus = A_i$

3: $P,\ G,\ C,\ P^{-1}$-rounds $\langle\ n \rightarrow (n-1)\ \rangle$

4: for $i = 1$ to $n-1$: $B_i \oplus = ancilla1_{i-1}$

5: for $i = 0$ to $n-2$: $NOT(B_i)$

6: for $i = 1$ to $n-2$: $B_i \oplus = A_i$

7: line 3 reverse

8: for $i = 1$ to $n-2$: $B_i \oplus = A_i$

9: for $i = 0$ to $n-2$: $ancilla1_i \oplus = A_i B_i$

10: for $i = 0$ to $n-2$: $NOT(B_i)$

[알고리즘 2-3] 수정된 in-place Draper 덧셈기 알고리즘

[알고리즘 2-2]는 mod 2^n 덧셈을 수행하도록 out-of-place 알고리즘을 수정한 결과이다. n 대신 $n-1$을 대입하여 $(n-1)$-bit 덧셈까지는 기존 알고리즘과 동일하게 진행되며, 최상위 비트 간의 합이 저장되는 보조 큐비트 Z_{n-1}에 A_{n-1}과 B_{n-1}를 각각 제어 큐비트로 하여 CNOT 게이트를 적용한다. [알고리즘 2-3]은 in-place 알고리즘을 수정한 결과로, 1번 라인 for문의 반복횟수를

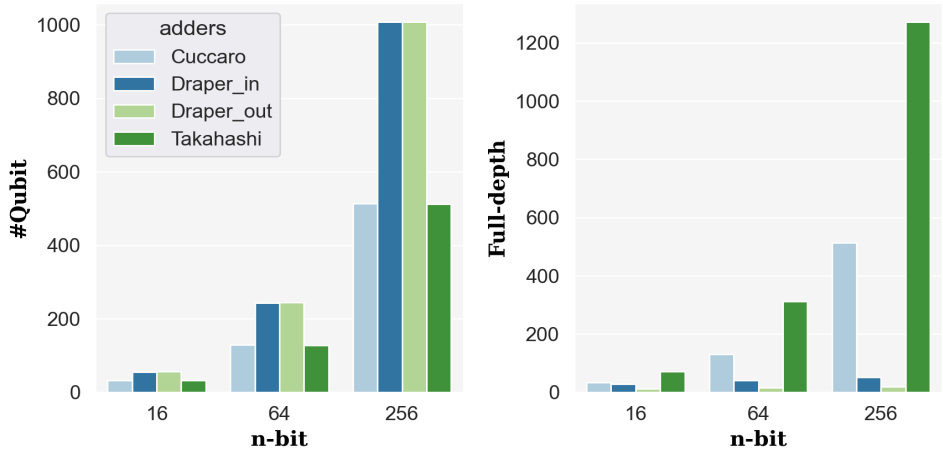
1만큼 줄여 $n-2$로 변경하였으며, 3, 7번 라인의 캐리 병렬 연산 단계를 $n-1$에 대해 수행하도록 변경한 것 외에는 기존 알고리즘과 동일하다.

Adder		#Qubit	#Toffoli	Toffoli-depth	Full-depth
Cuccaro		$2n+1$	$2n-3$	$2n-3$	$2n+2$
Takahashi		$2n$	$2n-3$	$2n-3$	$5n-8$
Draper	in	$-2+4n$ $-w(n-1)$ $-\lfloor \log(n-1) \rfloor$	$-12+10n$ $-6w(n-1)$ $-6\lfloor \log(n-1) \rfloor$	7 $+2\lfloor \log(n-1) \rfloor$ $+2\left\lfloor \log\frac{n-1}{3} \right\rfloor$	14 $+2\lfloor \log(n-1) \rfloor$ $+2\left\lfloor \log\frac{n-1}{3} \right\rfloor$
	out	$-1+4n$ $-w(n-1)$ $-\lfloor \log(n-1) \rfloor$	$-6+5n$ $-3w(n-1)$ $-3\lfloor \log(n-1) \rfloor$	4 $+\lfloor \log(n-1) \rfloor$ $+\left\lfloor \log\frac{n-1}{3} \right\rfloor$	7 $+\lfloor \log(n-1) \rfloor$ $+\left\lfloor \log\frac{n-1}{3} \right\rfloor$

[표 2-1] mod 2^n상의 양자 덧셈기에 대한 자원 비교 ($n \geq 5$)

[표 2-1]은 3가지 양자 덧셈기를 통한 n-bit 덧셈에 사용되는 큐비트 수, CNOT 게이트 수, Toffoli 게이트 수, Toffoli-depth, Full-depth를 정리한 것으로, 정확한 수치를 확인할 수 있도록 수식으로 나타내었다.

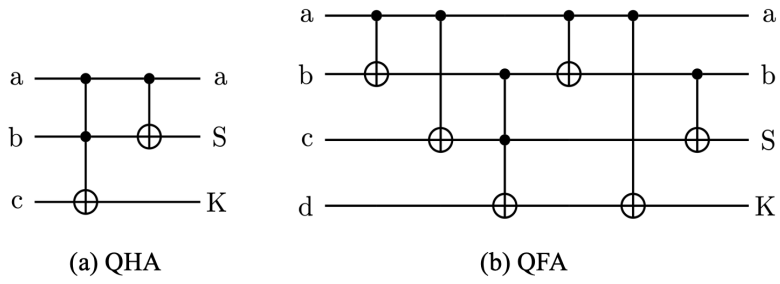
다음으로 [그림 2-5]는 Trade-off 관계인 큐비트 수와 회로 depth 측면에서 한눈에 이점을 비교할 수 있도록 그래프로 나타낸 것이다. n이 16, 64, 256으로 커짐에 따라 뚜렷한 차이를 보이는데, Ripple-carry 덧셈기인 Cuccaro와 Takahashi 덧셈기의 경우 큐비트를 적게 사용하는 반면, 높은 회로 깊이를 보인다. 반대로 Carry-lookahead 덧셈기인 Draper 덧셈기는 캐리 병렬 연산을 위해 큐비트 수가 비교적 많이 사용되지만, 병렬 연산으로 인해 낮은 회로 깊이를 가지는 것을 알 수 있다. 또한 Takahashi 덧셈기는 보조 큐비트를 사용하지 않는 대신 Cuccaro 덧셈기보다 약 2.5배 높은 전체 깊이를 가지는 것을 알 수 있다. 결론적으로 양자 회로 최적화 구현 시에 큐비트 수를 최소화하는 경우에는 Ripple-carry 덧셈기를, 회로 depth를 최소화하는 경우에는 Carry-lookahead 덧셈기를 사용하는 것이 성능 개선에 유리하다는 것을 알 수 있다. 본 논문은 depth 최적화를 위해 Draper 덧셈기를 채택하였다.



[그림 2-5] 큐비트 수와 회로 depth 측면에서의 덧셈기 자원 비교

2.1.3.2 Carry-save 양자 덧셈기

피연산자가 3개 이상인 경우, 병렬성을 이용하여 계산 성능을 크게 향상시킬 수 있다. 본 논문에서 SHA-256 구현에 사용한 Carry-save 양자 덧셈기는 Kim[3]의 덧셈기이다. 해당 논문에서는 Wallce와 Dadda 트리 기반의 Carry-save 양자 덧셈기를 제안하였다. 두 구조 모두 QHA(Quantum Half Adder)와 QFA(Quantum Full Adder)를 이용하여 피연산자의 bit 단위씩 병렬로 덧셈을 수행한다. QHA와 QFA의 회로는 [그림 2-6]과 같다.

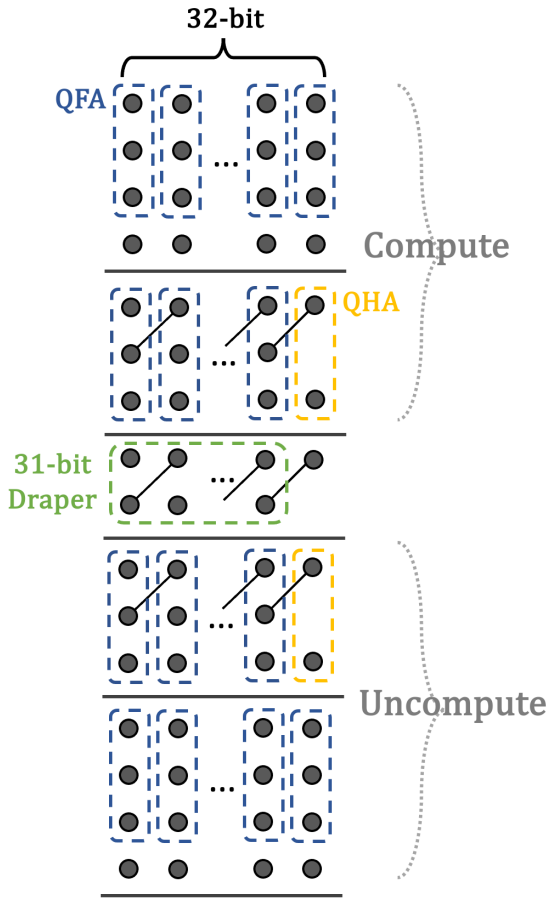


[그림 2-6] Carry-save 덧셈기에서 사용되는 QHA와 QFA

3) Tree-based Quantum Carry Save Adder.

QHA는 두 피연산자 큐비트와 캐리 값을 저장하기 위한 큐비트를 입력으로 받아 두 번째 큐비트에 합을 저장하고 세 번째 큐비트에 캐리 값을 저장한다. QFA는 QHA에서 피연산자가 하나 늘어난 것이며, (b) 회로의 경우 Toffoli 게이트를 2개에서 1개로 줄여, 하나의 Toffoli 게이트와 Toffoli-depth 1만으로 3개의 피연산자의 캐리 값을 연산할 수 있도록 최적화된 것이다.

본 논문에서는 Toffoli-depth가 더 작은 Wallace 트리 기반의 Carry-save 양자 덧셈기를 채택하여 mod 2^{32}상에서의 덧셈기로 수정하였다. 수정된 Wallce 트리 기반의 Carry-save 양자 덧셈기의 동작 방식은 [그림 2-7]과 같다. 32-bit 숫자 4개를 더하는 예제로, 먼저 QFA를 통해 bit당 피연산자를 3개씩 묶어 덧셈을 수행한다. 이 작업을 3개씩 묶음이 나올 수 있는 만큼 반복해서 수행하면 최하위 비트에 2개의 피연산자만 남게 된다. 이때 QHA를 통해 두 피연산자에 대한 덧셈을 수행한다. 최하위 비트 단에서 함께 QHA로 묶이는 경우를 제외하고, 나머지 비트에서 2개의 피연산자만 남게 되면 QHA로 묶는 과정을 중단한다. [그림 2-7]의 경우 최종적으로 남은 31-bit에 대해 Draper 덧셈을 수행하게 된다. QHA로 연산된 결과인 최하위 1-bit는 31-bit 덧셈 결과에 붙여준다. 이때 그림에서 선을 통해 상위 비트에 추가되는 피연산자는 하위 비트에서 발생한 캐리 비트를 의미한다. 정리하면, n개의 피연산자에 대해 QFA와 QHA를 적용하는 과정을 반복하여 2개 이하의 피연산자만 남긴 후에 2개의 피연산자에 대해 이항 덧셈기를 적용해주는 것이다. 본 구현에서는 2.1.3.1절의 Draper 덧셈기를 사용하였다. 이항 덧셈 이후에는 앞에서 수행했던 QFA와 QHA 연산을 해제하여 피연산자와 캐리 값을 초기 값으로 되돌린다. 이때 캐리 비트는 초기 값인 0으로 되돌아가기 때문에 다음 라운드에서 재사용 가능하다. 또한 가로 일직선을 기준으로 한번에 병렬로 수행되며, 한번 병렬로 수행될 때 Toffoli-depth가 1이 된다. 따라서 이항 덧셈 연산을 제외한 [그림 2-7]의 Toffoli-depth는 4가 된다. 4개의 피연산자의 합을 구하기 위해 32-bit 덧셈을 3번 수행하는 경우보다 Carry-save 덧셈기를 1번 사용하는 것이 32-bit 덧셈 1번보다 크고 2번보다는 작은 Toffoli-depth를 가지는 것을 알 수 있다.



[그림 2-7] Wallce 트리 기반의 mod 2^{32} Carry-save 덧셈기 동작 방식

2.2 Grover 알고리즘을 통한 해시함수 역상 공격

Grover 알고리즘은 N개의 데이터 집합에서 특정 데이터를 검색할 때 사용되는 양자 알고리즘이다. 고전컴퓨터에서는 전수 조사로 특정 데이터를 찾기 위해 $O(N)$의 시간 복잡도를 갖지만, Grover 알고리즘을 적용하면 $O(\sqrt{N})$의 시간 복잡도만으로 특정 데이터를 찾아낼 수 있다. 즉, 탐색 횟수를 2^n에서 $2^{n/2}$으로 줄일 수 있는 것이다. Grover 알고리즘을 해시함수에 적용하는 연구는 주로 역상 공격(Preimage)에 대해 수행되므로 본 논문에서도 마찬가

지로 역상 공격에 대한 Grover 비용을 추정하였다. 역상 공격은 해시 값이 주어졌을 때의 원본 메시지를 찾아내는 공격이다. 이를 위해서는 원본 메시지 길이를 제한해야하는데, 본 논문에서는 128-bit로 제한하였다.

Grover 알고리즘을 통해 해시함수의 역상 공격을 수행하는 과정은 크게 입력 설정, Oracle 함수, Diffusion 연산자로 구성된다. 각 단계는 [수식 2-2], [수식 2-3], [수식 2-4]와 같이 나타낼 수 있다. 먼저 Hadamard 게이트를 사용하여 중첩 상태의 n 큐비트 입력 메시지를 준비한다. 중첩 상태의 입력 메시지는 모든 경우의 메시지 값을 확률로 갖게 된다.

$$|\psi\rangle H^{\otimes n}|0\rangle^{\otimes n}=\left(\frac{|0\rangle \ +|1\rangle}{\sqrt{2}}\right)^{\otimes n}=\frac{1}{2^{n/2}}\sum_{x=0}^{2^n-1}|x\rangle$$

[수식 2-2] Grover 알고리즘 첫 번째 단계 - 입력 설정

$$f(x)=\begin{cases}1 \text{ if } Hash(m)=h\\0 \text{ if } Hash(m)\neq h\end{cases}$$

[수식 2-3] Grover 알고리즘 두 번째 단계 - Oracle 함수

다음으로 [수식 2-3]에 해당하는 Oracle 함수 $f(x)$에서 중첩 상태의 메시지로 구현된 대상 해시 함수 양자 회로를 통해 중첩 상태의 해시 결과 값을 생성한다. 원하는 해시 결과 값과 일치하는 경우, 즉 $f(x)=1$인 경우에는 이후 해당 메시지 큐비트 값의 부호를 반전시키는 방식으로 Oracle 함수는 정답 메시지를 반환한다. 일치하지 않는 경우에는 0을 반환한다.

마지막으로 Diffusion 연산자를 수행한다. Diffusion 연산자는 Oracle 함수에서 반전된 정답의 관측 확률을 증폭시킨다. Grover 알고리즘은 Oracle 함수와 Diffusion 연산자를 반복 수행한다. 충분한 반복 수행을 통해 정답을 관측할 확률은 점점 증가하게 되고 정답이 아닌 데이터를 관측할 확률은 감소하게 된다.

$$U_f(|\psi\rangle|-\rangle)=\frac{1}{2^{n/2}}\sum_{x=0}^{2^n-1}(-1)^{f(x)}|x\rangle|-\rangle$$

[수식 2-4] Grover 알고리즘 세 번째 단계 - Diffusion 연산자

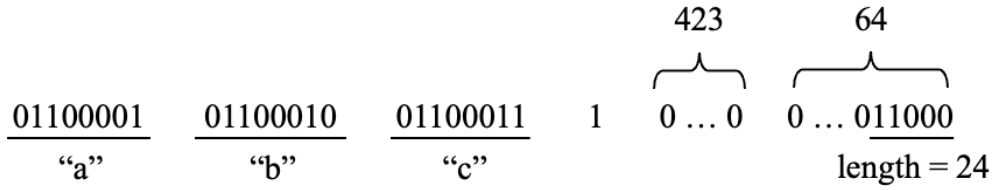
2.3 SHA-256 해시함수 알고리즘

SHA-2(Secure Hash Algorithm 2)는 2002년에 미국 국립표준기술연구소(NIST, National Institute of Standards and Technology)에서 표준화한 해시함수의 집합이다. 고정된 해시 값을 생성하며, 메시지의 일부만 변경해도 해시 값이 크게 달라진다. 해시 값의 길이에 따라 SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224, SHA-512/256으로 나뉘며, 일부 상수와 라운드 수를 제외하고는 구조적으로 동일하다. [표 2-2]는 SHA-2 해시함수에 대한 상세 정보를 나타낸다. 메시지의 크기는 원본 메시지의 길이를 나타내며, 블록의 크기는 해시 값을 생성 시 블록 단위로 해시 연산이 수행되는데, 이때 단위가 되는 길이를 가리킨다. 마지막으로 메시지 다이제스트의 크기는 입력 메시지의 길이에 관계없이 해시함수의 결과로 반환되는 고정된 해시 값의 길이를 나타낸다. 본 논문에서는 6가지의 SHA-2 해시함수 중 SHA-256 알고리즘에 초점을 맞추어 설명하고자 한다.

[표 2-2] SHA-2 상세 정보

Hash Function	Message Size (bit)	Block Size (bit)	Message Digest Size (bit)
SHA-224	$< 2^{64}$	512	224
SHA-256	$< 2^{64}$	512	256
SHA-384	$< 2^{128}$	1024	384
SHA-512	$< 2^{128}$	1024	512
SHA-512/224	$< 2^{128}$	1024	224
SHA-512/256	$< 2^{128}$	1024	256

SHA-256의 동작 방식은 크게 전처리 단계와 해시 연산 단계로 구성되는데, 전처리 단계에서는 메시지를 512-bit의 배수가 되도록 패딩 및 파싱을 수행하고, 해시 값이 생성되는 해시 연산 단계는 전처리 단계를 통해 생성된 메시지의 블록 수에 따라 전체 알고리즘을 반복적으로 수행한다. 두 단계를 상세히 살펴보자.

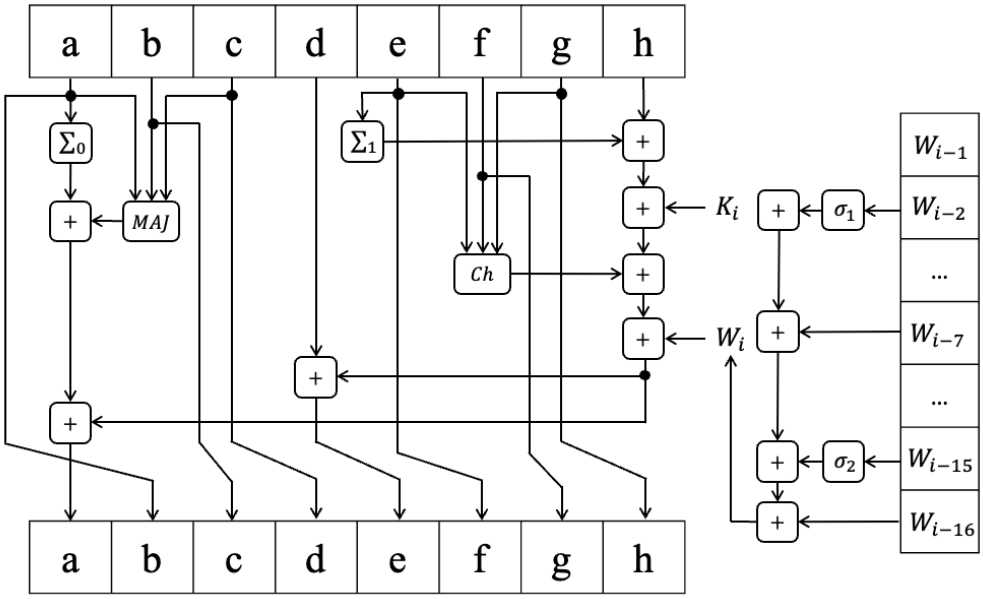


[그림 2-8] SHA-256 메시지 패딩 방식

먼저 전처리 단계의 패딩은 [그림 2-8]과 같이 수행된다. 전체 512-bit의 블록 길이에서 입력된 메시지의 길이를 나타내기 위한 맨 뒤 64-bit를 제외하고 나머지 448-bit에 입력된 메시지와 메시지 바로 뒤에 1-bit의 값인 1을 삽입하여 메시지의 끝을 나타내고 나머지는 0으로 채워준다. “abc”가 입력된 메시지라고 할 때, 각 단어를 1-byte(8-bit)로 변환하여 24-bit로 나타낸 후, 메시지 바로 뒤에 1을 삽입한다. 그 후 448-bit 중 앞의 25-bit를 제외한 423-bit를 0으로 채운 뒤 입력 메시지의 길이인 24를 이진수로 변환한 값이 나머지 64-bit를 차지하게 된다.

파싱은 앞에서 생성한 512-bit의 패딩된 메시지를 32-bit씩 16개로 나누는 과정이다. 해시 연산에 사용되는 W 배열의 0부터 15까지의 요소가 전처리 단계에서 파싱할 때 생성된다. 본 논문의 양자 구현에서는 기존 연구들과 마찬가지로 메시지 블록 수를 1로 가정한다. 해시 연산 단계에서 해시 값을 업데이트 할 때 값을 복사하는 연산자가 필요하지만, 이는 양자 환경에서의 복제 불가능 정리에 위배되기 때문이다. 따라서 메시지 블록 수를 1로 가정함으로써 해시 값 업데이트 과정을 생략한다. 이로써 SHA-256 양자 회로가 처리할 수 있는 최대 입력 메시지 길이는 최소 패딩 비트인 65-bit를 제외한

447-bit이지만, 본 논문에서는 입력 메시지의 길이를 128-bit로 제한하여 연산 최적화를 수행하였다.



[그림 2-9] SHA-256의 라운드 함수 및 메시지 확장 함수

다음으로 해시 연산 단계에선 전처리 단계에서 생성된 메시지 블록 수에 따라 라운드 함수 알고리즘을 블록 당 64번씩 반복해서 실행한다. [그림 2-9]는 왼쪽부터 라운드 함수 알고리즘과 메시지 확장 함수를 나타낸다. 메시지 확장 함수는 [수식 2-5]와 같이 전처리 단계에서 생성한 16개의 32-bit 메시지로 이루어진 W 배열을 통해 추가로 48개의 W 배열을 생성한다. [그림 2-9]에서 i의 범위는 $16 \le i \le 63$이며, W 배열을 생성하는 과정에서 논리 함수 σ_0, σ_1이 사용된다. SHA-256 알고리즘에서 사용되는 모든 논리 함수 연산은 [표 2-3]에 정리하였다.

$$W_i = M_i \quad (0 \le i \le 15)$$
$$= \sigma_1(W_{i-2}) + W_{i-7} + \sigma_0(W_{i-15}) + W_{i-16} \quad (16 \le i \le 63)$$

[수식 2-5] 메시지 확장 함수

[표 2-3] SHA-256 논리 함수 연산

$\sigma_0(a)$	$ROTR^7(a) \oplus ROTR^{18}(a) \oplus SHR^3(a)$
$\sigma_1(a)$	$ROTR^{17}(a) \oplus ROTR^{19}(a) \oplus SHR^{10}(a)$
$\Sigma_0(a)$	$ROTR^2(a) \oplus ROTR^{13}(a) \oplus ROTR^{22}(a)$
$\Sigma_1(a)$	$ROTR^6(a) \oplus ROTR^{11}(a) \oplus ROTR^{25}(a)$
$Maj(a,b,c)$	$(a \wedge b) \oplus (a \wedge c) \oplus (b \wedge c)$
$Ch(a,b,c)$	$(a \wedge b) \oplus (\neg a \wedge c)$

라운드 함수는 상수 값으로 정해진 초기 해시 값 $H^{(0)}$가 32-bit로 이루어진 8개의 변수 a, b, c, d, e, f, g, h에 할당되어 매 라운드마다 [수식 2-6]과 같이 연산 및 업데이트 된다.

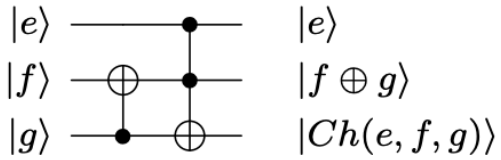
$$
\begin{aligned}
&T_1 = h + \Sigma_1(e) + Ch(e,f,g) + K_i + W_i,\\
&T_2 = \Sigma_0(a) + Maj(a,b,c),\\
&a = T_1 + T_2,\ b = a,\ c = b,\ d = c,\\
&e = d + T_1,\ f = e,\ g = f,\ h = g
\end{aligned}
$$

[수식 2-6] 라운드 함수

여기에서 K_i는 SHA-256에서 사용되는 상수 값으로, 32-bit씩 64개로 구성된다. 라운드 함수 내에서는 논리함수 $Maj, Ch, \Sigma_0, \Sigma_1$이 사용되며, 해당 연산도 마찬가지로 [표 2-3]에 정리하였다. 64번의 라운드 함수를 마치면, 초기 해시 값 $H^{(0)}$과 a, b, c, d, e, f, g, h를 더해 최종 해시 값 $H^{(1)} = H_0^{(1)} \| \cdots \| H_7^{(1)}$이 구해진다. 메시지 블록 수가 1 이상인 경우에는 해시 연산 단계를 통해 생성된 최종 해시 값을 누적해서 더해주면 된다. 본 논문에서 제안하는 SHA-256에 대한 양자 구현에서는 맨 마지막에 초기 해시 값과 더하는 연산을 생략한다. Grover 알고리즘에 사용되는 양자 회로의 크기를 최소화 할수록 공격에 유리한데, 마지막 상수 덧셈 연산을 생략한 해시 값을 입력 메시지에 대한 정답 데이터로 지정함으로써 양자 회로 비용을 줄일 수 있기 때문이다.

2.4 SHA-256 양자 회로 최적화 구현에 대한 이전 연구

본 논문에서 비교 대상으로 삼은 연구는 Lee[4]가 2023년에 발표한 SHA-256 양자 회로의 T-depth 최적화 논문이다. 저자는 T 게이트와 $T^{\dagger}$ 게이트가 양자컴퓨팅 계산 복잡도 및 성능에 큰 영향을 주기 때문에 T 게이트 수와 T-depth를 최소화하는 것에 초점을 맞추었다. 이를 위해 Toffoli 게이트 사이의 서브 회로를 통해 T-depth를 줄이는 새로운 기술을 제안하였다. 덧셈 연산을 제외하고 Toffoli 게이트가 사용되는 연산에는 *Maj* 함수와 *Ch* 함수가 있는데, 저자는 *Ch* 함수 연산을 [그림 2-10]처럼 최적화하였다. 또한 덧셈이 핵심 연산인 SHA-256을 최적화하기 위해 이전 연구에서 7 또는 10으로 구성되었던 덧셈기-depth를 3으로 줄였다. 이를 바탕으로 SHA-256에 대한 양자 회로를 4가지 버전으로 최적화하여 각 버전마다 효율적인 덧셈기를 사용하여 구현하였는데, 본 논문에서는 비교 대상인 depth를 가장 최적화한 버전을 중심으로 설명한다.

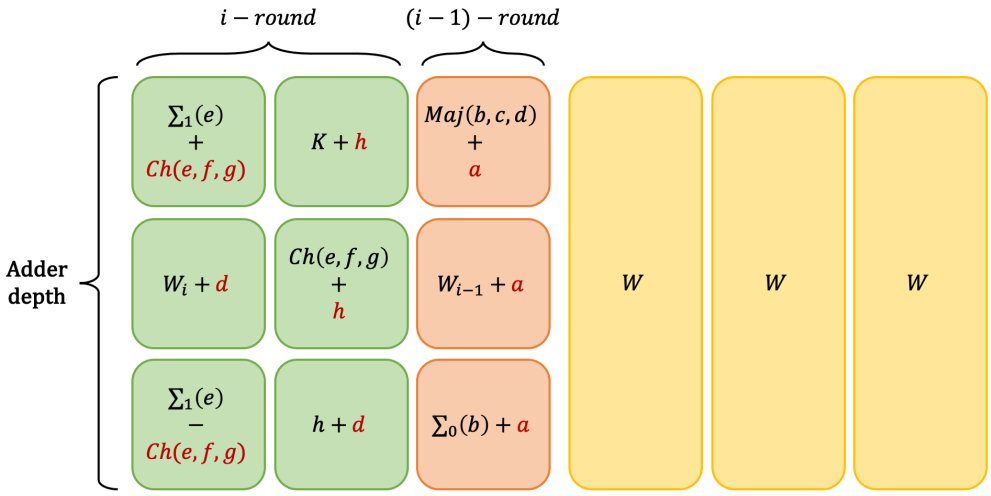


[그림 2-10] 이전 연구에서 제안한 최적화된 *Ch* 함수의 양자 회로

[그림 2-11]은 해당 논문에서 덧셈기-depth를 3으로 최적화한 방식을 나타낸다. 덧셈기는 Draper의 in-place 구현을 사용하였으므로 덧셈 결과가 저장되는 b에 해당하는 피연산자를 빨간색으로 표시하였다. 따라서 in-place 양자 구현에서 T_1이 d와 더해진 결과는 T_1에 해당하는 요소를 d와 각각 더하여 d에 저장한 뒤, 다음 라운드에서 e로 전달되고, T_1과 T_2가 더해지는 결과는 T_1과 T_2에 해당하는 요소들을 h와 각각 더한 결과를 h에 저장하여 다음

4) T-depth reduction method for efficient SHA-256 quantum circuit construction.

라운드에서 a로 전달하게 된다. 저자는 덧셈기-depth를 줄이기 위해 전체 라운드 수를 64 라운드에서 65 라운드로 늘렸다. 맨 처음 라운드에서는 초록색에 해당하는 부분만 연산한다. 이 연산은 T_1에 해당하는 요소들을 d와 h에 나누어 더하는 과정이다. 그 다음 라운드부터는 초록색과 주황색 부분을 동시에 수행하게 되는데, 주황색 부분은 이전 라운드에서 완성하지 못한 T_2에 해당하는 요소들을 이전 라운드에서 h에 해당하는 현재 라운드의 a에 더하는 과정이다. 이러한 덧셈 구조는 큐비트 값이 달라지지 않는 선에서 치밀하게 짜여진 구조이기 때문에 덧셈 순서를 임의로 바꿀 수 없다. 나머지 노란색 부분은 메시지 확장 함수에 해당하는 연산으로, 저자는 Draper 덧셈기-depth 3의 Toffoli-depth에 근접하는 TK 덧셈기를 사용하여 4 라운드부터 53라운드에 걸쳐 W_{16}부터 W_{63}까지 구하였다. 해당 구현에 대한 양자 자원 비교는 4절에서 비교한다.



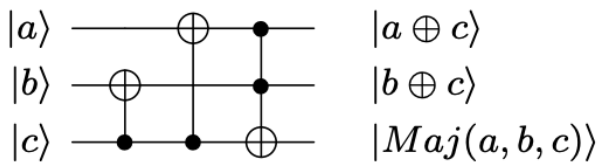
[그림 2-11] Lee의 덧셈기-depth 3 최적화 방식

III. SHA-256 양자 회로의 depth 최적화 구현 기법

본 절에서는 SHA-256 양자 회로를 Toffoli-depth와 전체 회로 depth에 대해 최적화 구현한 기법에 대해 설명하고자 한다. SHA-256을 구성하는 연산들을 중심으로 차례대로 살펴본다. 마지막으로 Toffoli 게이트가 사용된 연산에 AND 게이트를 적용하여 T-depth를 최적화한 방식에 대해서도 알아본다. 양자 구현에는 ProjectQ 프레임워크를 사용하였다.

3.1 주요 논리 연산 최적화

SHA-256의 주요 논리 연산에는 σ_0, σ_1, Σ_0, Σ_1, Maj, Ch가 있다. σ_0, σ_1, Σ_0, Σ_1의 경우, 이전 연구에서는 PLU 분해를 통해 보조 큐비트를 사용하지 않고 구현한 대신 큰 depth를 가지게 된다. 따라서 본 논문에서는 해당 연산이 적용되는 output용 보조 큐비트 32개를 할당하여 depth를 3으로 줄였다. *Maj*와 *Ch*는 Toffoli 게이트가 사용되는 연산이다. *Ch* 함수 연산에는 2.4절의 이전 연구에서 제안한 최적화된 회로를 사용하였으며, *Maj* 함수 연산도 마찬가지로 이전 연구에서 사용한 [그림 3-1]의 회로를 사용하였다. 두 연산 모두 추가 큐비트 없이 1개의 Toffoli 게이트를 사용하며, Toffoli-depth 1을 가진다.



[그림 3-1] 이전 연구에서 사용한 *Maj* 함수의 양자 회로

3.2 라운드 함수 최적화

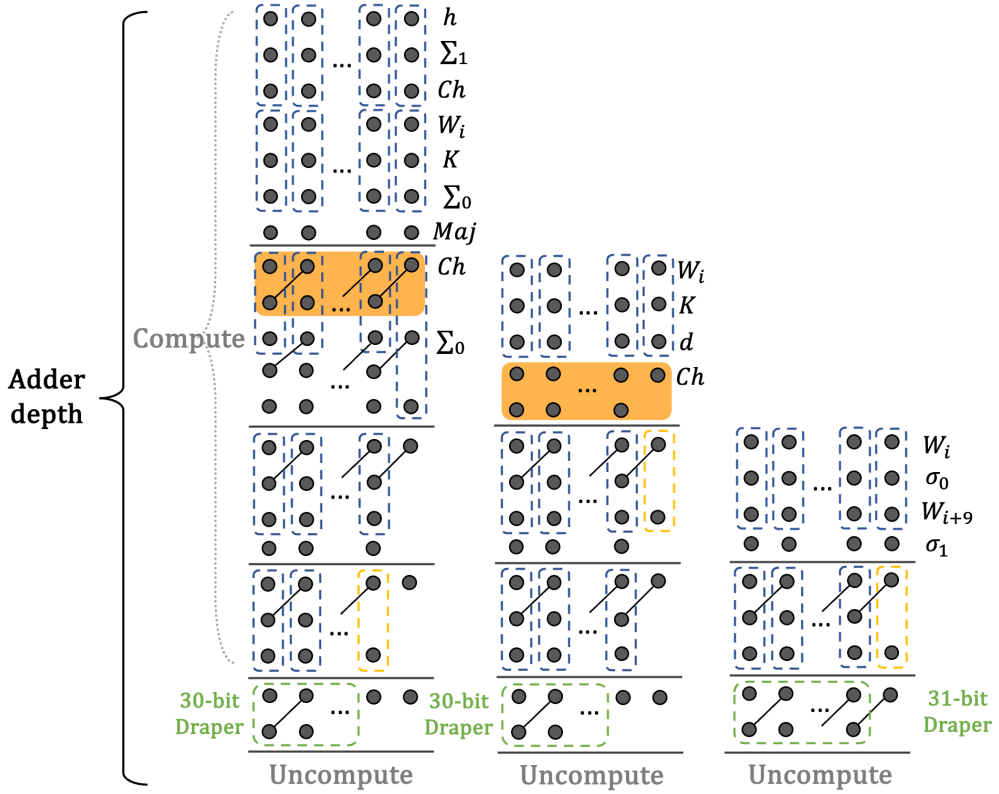
SHA-256에서 사용되는 연산 중 Maj와 Ch를 제외하면 덧셈기에서만 Toffoli 게이트가 사용된다. 3.1절에서 살펴보았듯이 두 연산은 Toffoli-depth 1로 구현되었으므로, SHA-256 양자 회로의 Toffoli-depth를 줄이려면 덧셈 연산을 최적화하거나 효율적인 덧셈기를 사용해야한다. 매 라운드마다 덧셈을 통해 업데이트 되어야하는 내부 변수는 h와 d이며, 양자 구현을 위해 h와 d에 더해져야하는 연산을 정리하면 [수식 3-1]과 같다. Lee의 구현에서는 in-place 덧셈기를 사용하였기 때문에 덧셈기-depth가 최소한 3이 유지되어야한다. 하지만 본 논문에서는 두 내부 변수에 대한 다중 피연산자 덧셈을 2.1.3.2절에서 살펴본 Carry-save 양자 덧셈기의 out-of-place 구현을 사용하여 덧셈기-depth 1로 구현하였다. 이항 덧셈기를 여러 번 사용하는 방식보다 Kim의 Carry-save 양자 덧셈기를 사용하면 Toffoli-depth를 크게 줄이면서 덧셈기-depth 1만으로도 SHA-256의 라운드 함수와 메시지 확장 함수에서 사용되는 덧셈 연산을 모두 수행할 수 있다.

$$h = \Sigma_0(a) + Maj(a,b,c) + \Sigma_1(e) + Ch(e,f,g) + h + K_i + W_i,$$
$$d = d + \Sigma_1(e) + Ch(e,f,g) + h + K_i + W_i,$$

[수식 3-1] 양자 구현을 위한 라운드 함수의 덧셈 연산 정리

[그림 3-2]를 보면, 가장 왼쪽이 h를, 가운데가 d를 업데이트하는 덧셈 연산이다. 두 번의 다중 피연산자 덧셈을 2개의 Carry-save 덧셈기를 사용하여 병렬로 연산하였는데, 이때 피연산자가 겹치는 경우를 고려하지 않고 구현하면 Toffoli-depth 값이 증가하게 된다. 본 구현에서는 Carry-save 덧셈기의 2가지 특징을 고려하여 Toffoli-depth가 최적화하면서 덧셈을 병렬로 수행할 수 있도록 구현하였는데, 그 특징은 다음과 같다. QFA에 입력된 3개의 큐비트 중 마지막 큐비트에 합이 저장되어 나머지 2개의 큐비트는 값이 유지된다는 것과 한번에 3개 묶음씩 덧셈을 수행하는 Carry-save 덧셈기 구조를 활용하여 같은 피연산자에 동시에 접근하지 않도록 분리하였다. 이를 W_i와

K에 대해 적용하여 h와 d 연산에서 겹치지 않고 병렬로 사용할 수 있도록 구현하였다. 또한 d가, h와 겹치는 피연산자가 3개 이상이므로, 반복해서 동일한 연산을 수행하지 않도록, h에서 먼저 연산한 결과를 연산에서 사용한 이후, d 연산에 가져와서 사용하는 방식으로 구현하였으며, [그림 3-2]의 주황색 부분과 같다. $h + \Sigma_1(e) + Ch(e,f,g)$ 연산의 합이 Ch에 저장되고 Carry 값이 보조 큐비트에 저장된 후, 두 번째 병렬 QFA 연산에서 사용된 이후에 d에서 사용된다. h 연산을 기준으로, 세 번째 병렬 QFA 연산에서 사용된다. QFA와 QHA로 묶음 연산을 수행한 후에는 30-bit의 Draper 덧셈을 병렬로 수행하게 된다. Draper 덧셈으로 업데이트된 h와 d 값을 구했으면 앞서 수행했던 묶음 연산을 해제함으로써 피연산자들을 초기 값으로 되돌리고, 캐리 큐비트도 재사용 가능한 상태로 만들어준다.



[그림 3-2] 덧셈기-depth 1로 최적화된 Carry-save 덧셈기 구조

본 논문에서는 입력 메시지의 길이를 128-bit로 제한하였는데, 이를 통해 W 배열의 일부 값이 고정된다. 입력 메시지에 해당하는 $W_0 \sim W_3$을 제외한 나머지 12개의 값은 고정되어있다. 또한 W_4와 W_{15}를 제외한 10개의 W의 값은 0이 된다. 따라서 64 라운드 중 10 라운드에서 h와 d를 업데이트하는 덧셈 연산 수행 시, W를 제외하고 덧셈을 수행할 수 있다. 이를 통해 [그림 3-2]의 덧셈기의 묶음 연산 및 연산 해제 부분에서 Toffoli-depth를 1씩 줄임으로써 매 라운드마다 2씩 총 20만큼 줄일 수 있다.

3.3 메시지 확장 함수 최적화

메시지 확장 함수도 라운드 함수에서 수행되는 덧셈 연산과 함께 병렬로 수행된다. [그림 3-2]에서 가장 오른쪽에 해당하는 덧셈 연산이 메시지 확장 함수 연산인데, 3.2절에서 피연산자끼리 겹치지 않도록 순서를 조절했던 방식과 동일하게 라운드 함수의 h와 d 연산에서 W_i를 사용한 이후에 메시지 확장 함수에서 사용하게 된다. 이전 연구에서는 피연산자가 겹치기 때문에 4 라운드부터 메시지 확장 함수 연산을 수행했지만, 본 구현에서는 해당 라운드에서 사용되는 W 배열이여도 병렬로 사용할 수 있으며, out-of-place 구현을 사용하기 때문에 1 라운드부터 메시지 확장 함수 연산을 수행하고, 중간에 몇 라운드를 제외하고 58라운드까지 메시지 확장 함수 연산을 수행한다. 또한 라운드 함수에서 h와 d 연산 결과 저장을 위한 32-bit 큐비트 배열 중 38개의 큐비트 배열 일부를 결과 저장에 사용되기 전까지 확장된 W 배열을 저장하고 추가적인 연산을 수행하는 용도로 사용하였다. 33 라운드부터 일부 라운드를 제외하고 58 라운드까지 이러한 추가 큐비트를 활용하여 Carry-save 덧셈기 1개를 추가로 사용하였다. 즉, 해당 라운드 구간에서는 4개의 Carry-save 덧셈기가 Toffoli-depth를 그대로 유지하면서 병렬로 수행된다. 추가된 Carry-save 덧셈기는 빌린 큐비트에 저장된 일부 W 배열 값을 연산 해제하기 위해 사용된다. 즉, 뺄셈을 수행하여 큐비트를 초기 값으로 되돌려 준다. 초기 값으로 돌아간 큐비트는 본래의 목적이었던 h와 d 연산 결과 저

장에 사용된다. 이러한 과정을 통해서 총 896개의 큐비트를 절약하였다.

입력 메시지의 길이를 128-bit로 제한함으로써 고정된 12개의 W 배열도 초기 라운드에서 사용된 이후 다시 재사용하여 확장된 W 배열을 저장하는데 사용하였으며, 384개의 큐비트를 절약할 수 있었다. 0으로 고정된 10개의 W 값은 메시지 확장 함수에서 덧셈을 수행할 때 전부 고려하여 0인 경우 덧셈에서 제외하였다. 6 라운드의 경우 메시지 확장 함수의 피연산자 중 W_{14}와 W_6 , W_5가 모두 0이어서 덧셈이 아예 수행되지 않는다. 물론 라운드 함수에서 수행되는 덧셈이 Toffoli-depth가 가장 크기 때문에 전체 Toffoli-depth를 감소시키진 않지만, 불필요한 연산을 제거함으로써 Toffoli 게이트를 비롯한 양자 게이트 수를 줄이고자 하였다.

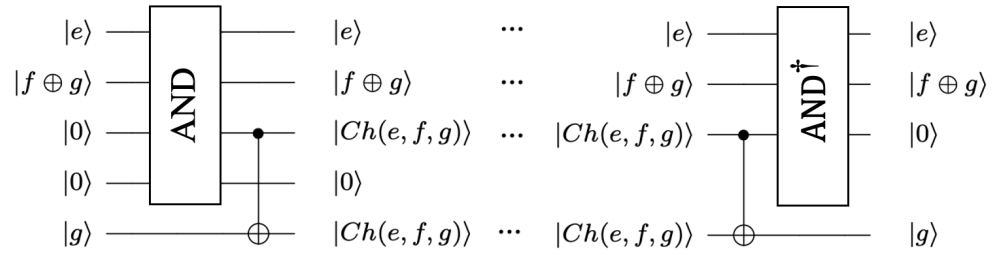
3.4 AND 게이트 적용을 통한 최적화

Toffoli-depth의 최적화는 T-depth의 최적화로 이어지며, 최적화된 Toffoli 게이트 분해 기법을 사용할수록 T 게이트 관련 자원을 줄일 수 있다. 앞 절에서는 여러 최적화 구현 기법을 적용하여 Toffoli-depth를 최적화한 방식에 대해 설명하였다. 본 절에는 SHA-256의 회로에서 Toffoli 게이트를 사용하는 *Maj*와 *Ch* 및 Carry-save 덧셈기 회로에 2.1.2절에서 살펴본 AND 게이트를 적용하여 T-depth를 최적화한 방식에 대해 설명한다. AND† 게이트의 특징으로 인해 T 게이트의 수의 최적화도 함께 수행된다.

AND 게이트를 적용할 때 주의할 점은 대상 큐비트의 상태가 $|0\rangle$이어야 한다는 점이다. *Maj*와 *Ch* 함수를 제외하면 대상 큐비트가 $|0\rangle$이 아닌 임의의 상태인 경우는 없기 때문에 해당 함수들만 [그림 3-3]과 같은 방식으로 AND 게이트를 적용하였다. Draper 덧셈기의 일부 알고리즘도 주의할 부분이 있는데, 이 부분은 본 절의 끝에서 설명한다.

*Maj*와 *Ch* 함수는 매 라운드의 시작에 연산되고, 라운드 끝에서 연산 해제된다. 따라서 라운드마다 사용되지 않는 보조 큐비트를 32-bit씩 할당하여 AND 게이트의 대상 큐비트로 사용하고, AND† 게이트를 통해 다시 $|0\rangle$으

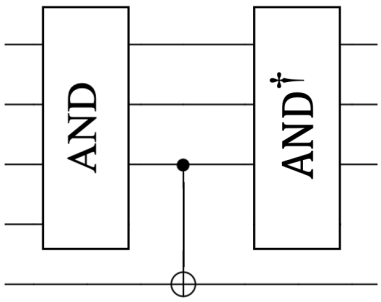
로 되돌아간다. 또한 AND 게이트에서는 추가로 1개의 보조 큐비트가 필요한데, AND 게이트 연산을 수행하더라도 초기 값이 그대로 유지되기 때문에 해당 라운드 초기에 사용되지 않는 큐비트를 빌린 뒤 반환하여 라운드 후기에 사용되도록 구현하였다. 한 라운드 당 *Maj*와 *Ch* 함수를 연산하고 해제하는데 사용되는 Toffoli-depth는 2였지만 AND 게이트를 적용함으로써 T-depth는 1로 줄어든다.



[그림 3-3] *Maj*, *Ch* 함수에서 AND 게이트를 적용한 방식

QFA와 QHA의 경우 AND 게이트 적용에 필요한 보조 큐비트는 Draper 덧셈기에 할당된 보조 큐비트를 사용하였다. QFA와 QHA는 하나의 Carry-save 덧셈기 회로에서 연산과 연산 해제가 수행되므로 T-depth는 Toffoli-depth의 절반이 된다.

마지막으로 Draper 덧셈기에 AND 게이트를 적용하는 방법에 대해 알아본다. 본 논문에서 사용되는 out-of-place 형태의 Draper 덧셈기는 P, P^{-1} 라운드 쌍을 제외하면 연산과 연산 해제를 수행하는 Toffoli 게이트 쌍이 존재하지 않으며, 맨 처음에 결과 큐비트에 적용되는 Toffoli 게이트 이후에 G, C 라운드에서 결과 큐비트에 적용되는 Toffoli 게이트의 경우, 대상 큐비트의 상태가 |0⟩이 아니다. 따라서 G, C 라운드에서는 [그림 3-4]와 같이 하나의 Toffoli 게이트를 AND 게이트, AND† 게이트 쌍과 하나의 CNOT 게이트로 분해하여 적용하였다. 하나의 Toffoli 게이트 분해에 AND 게이트 쌍을 적용하더라도 AND 게이트 쌍의 회로 Depth는 12로, 일반적인 Toffoli 분해 회로의 Depth와 비슷한 수치이다. 또한 T 게이트도 AND 게이트에서만 사용되기 때문에 비용이 크게 증가하지 않는다.



[그림 3-4] AND 게이트 쌍을 통한 Toffoli 게이트 분해

대상 큐비트 대신 사용되는 보조 큐비트는 G 라운드에서 동시에 수행되는 Toffoli 게이트 수인 $n-1-w(n-1)$만큼 할당하여 재사용하였다. 메시지 확장 함수에서 뺄셈을 수행하는 경우에는 Draper 덧셈기를 reverse하도록 구현하였다. 이때 Draper 덧셈기 구현에서 맨 처음 적용된 AND 게이트를 연산 해제하기 위해 reverse 구현의 경우, 맨 마지막에 수행되는 Toffoli 게이트에 $\text{AND}^{\dagger}$ 게이트를 적용하였다.

IV. 성능 평가

본 절에서는 SHA-256 양자 회로 구현에 사용된 양자 자원을 이전 연구와 비교하고 제안하는 회로에 대한 Grover 공격 비용을 추정한다. 정확한 자원 측정을 위해 ProjectQ와 IBM Qiskit, Google Cirq를 통해 자원 비용에 대한 교차 검증을 수행하였다.

4.1 SHA-256 양자 자원 비교

먼저, 3절에서 최적화한 주요 연산을 중심으로 이전 연구 결과와 비교를 수행하고, SHA-256 전체 회로에 요구되는 양자 자원에 대해 알아보고자 한다.

[표 4-1] Σ_0, Σ_1, σ_0, σ_1 논리 연산에 대한 자원 비교

		#Ancilla	#CNOT	Depth
Σ_0	Lee	0	166	55
	Ours	32	96	3
Σ_1	Lee	0	166	44
	Ours	32	96	3
σ_0	Lee	0	193	50
	Ours	32	93	3
σ_1	Lee	0	142	40
	Ours	32	86	3

[표 4-1]은 Σ_0, Σ_1, σ_0, σ_1 논리 연산 구현에 사용된 양자 자원을 나타낸다. 논리 연산 결과를 저장하는 보조 큐비트를 할당하여 연산을 수행하였기

때문에 32개의 큐비트가 사용되었지만 해당 큐비트는 라운드마다 재사용이 가능하며, 모든 구현에서 depth를 크게 줄였다. Lee의 이전 연구에서는 PLU 분해 기법을 사용하여 구현하였으므로 보조 큐비트가 사용되지 않은 반면, 높은 CNOT 게이트 수와 Depth를 가진다.

다음으로 [표 4-2]는 Toffoli 게이트가 사용된 연산에 대한 비교이다. *Maj*와 *Ch* 함수 연산은 동일한 회로가 사용되었고 Toffoli 게이트 분해 방식만 다르게 적용하였는데, 2개의 보조 큐비트를 통해 T-depth를 4에서 1로 줄였다. 해당 결과는 각 함수 연산과 역연산을 포함한 결과이다. Draper 덧셈기의 경우, Lee는 in-place 구현을 사용하였고 본 논문에서는 out-of-place 구현을 사용하였다. 사용되는 보조 큐비트 중 32개는 덧셈 결과 저장용 큐비트이고, 나머지 79개는 재사용이 가능하다. Carry-save 덧셈기(CSA)의 경우 라운드 함수에서 병렬로 사용되는 7, 5, 4개의 피연산자에 대한 버전으로 나누어 자원을 비교하였다. 피연산자에 따라 Toffoli-depth가 다르지만 3개의 덧셈기가 동시에 수행되므로, 매 라운드에서 사용되는 Carry-save 덧셈기의 Toffoli-depth는 3개 중 가장 큰 19가 된다. 보조 큐비트는 결과용으로 할당된 32 큐비트를 제외하고 다음 라운드에서 재사용할 수 있다.

[표 4-2] Toffoli 게이트가 사용된 연산에 대한 자원 비교

		#Ancilla	T-depth	Toffoli-depth
Maj, *Ch* ($Maj^{\dagger}$, $Ch^{\dagger}$)	Lee	0	4	2
	Ours	2	1	2
Draper	Lee (in-place)	53	24	22
	Ours (out-of)	111	9	11
CSA	7-version	260	13	19
	5-version	199	12	17
	4-version	172	11	15

[표 4-3]은 SHA-256 양자 회로에 대한 자원을 정리하여 나타낸 결과이

다. Lee는 SHA-256 양자 회로에 사용된 자원에 대해서 큐비트 수와 T-depth, Toffoli-depth만 논문에서 언급하였기 때문에 현 시점에서 비교할 수 있는 지표로 두 구현 결과에 대해 비교를 수행하고자 하였다. 또한 본 논문에서 최종적으로 제안하는 구현 결과는 Grover 공격을 위해 입력 메시지 비트 값을 128-bit로 제한하였다. 하지만 Lee의 회로는 입력 메시지 bit가 최대 447-bit까지 지원되기 때문에 공정한 비교를 위해서 447과 128-bit에 대한 결과를 함께 비교하였다. Lee가 최적화를 위해 설정한 덧셈기-depth 3의 구조에서는 입력 메시지 bit가 변경되어도 큐비트 수와 T-depth, Toffoli-depth 측면에서는 동일한 값을 갖게 된다. 반면 본 논문의 구현은 덧셈기-depth 1만에 모든 덧셈을 수행하기 때문에 입력 메시지 bit에 따라 일부 라운드에서 달라지는 피연산자의 개수가 T-depth와 Toffoli-depth의 감소에 영향을 주게 된다. 따라서 3가지 자원 지표만으로 성능을 비교하는 경우에는 입력 메시지 bit를 무시하여도 무방하다. 큐비트와 Toffoli-depth 간의 비교 지표로는 $TD^2 \times M$이 있는데, 이는 Toffoli-depth를 제곱한 값과 큐비트 값을 곱한 결과를 비교하는 것이다. 128-bit 기준으로 본 논문의 결과가 46% 개선하였고, 447-bit 기준으로 40% 개선되었다.

[표 4-3] SHA-256 양자 회로에 대한 자원 비교

Source	Message (bit)	#Qubit (M)	T-depth	Toffoli-depth (TD)	$TD^2 \times M$
Lee	447	962	4,936	4,418	18,777,012,488
Ours	447	6,135	896	1,344	11,081,871,360
	128	5,751	886	1,324	10,081,364,976

[표 4-4]는 본 논문에서 제안하는 SHA-256 양자 회로에 대한 전체 양자 자원을 나타내며, 전체 Depth는 Toffoli 게이트를 AND 게이트를 분해한 회로에 대하여 측정한 결과이다.

[표 4-4] 제안하는 양자 회로에 대한 전체 양자 자원

Source	#Qubit	#Clifford	#T	T-depth	#Toffoli	Toffoli depth	Depth
Ours (128-bit)	5751	951,228	167,120	886	69,182	1,324	9461

4.2 Grover 공격 비용

제안하는 SHA-256 양자 회로를 기반으로 SHA-256에 대한 Grover 역상 공격 비용을 추정한 결과는 [표 4-5]에 정리하였다. Grover 공격 비용을 추정하는 방식은 다음과 같다. Grover 공격은 Oracle 함수와 Diffusion 연산자를 반복해서 수행되지만, Diffusion 연산자의 비용이 상대적으로 매우 작아 주로 Oracle 함수의 반복 비용이 공격 비용으로 간주된다. Oracle 함수 내부에서는 SHA-256 회로가 2번 연속해서 동작한다. 따라서 앞의 [표 4-4]에서 큐비트 수를 제외한 메트릭에 $\times 2$한 것이 한번의 Oracle 함수에 대한 비용이며, Oracle은 약 2^{64}번 반복된다. 이에 따라 Grover 해킹의 최종 비용은 [표 4-5]와 같으며, NIST는 Grover 공격 비용을 사용된 모든 양자 게이트 수 $\times$ 전체 회로 depth인 양자 회로의 크기로 측정하는데, 동일한 방식으로 추정하면 비용은 1.5224×2^{162}이 된다. 보안강도 1에 해당하는 2^{157}와 보안강도 3에 해당하는 2^{221} 사이의 값을 가지므로 적합한 비용으로 구현되었음을 알 수 있다.

[표 4-5] Grover 공격 비용

Source	Total gates	Total depth	Cost
Ours	1.6783×2^{84}	1.8142×2^{77}	1.5224×2^{162}

V. 결 론

다가오는 양자컴퓨터 시대를 대비하기 위해 기존에 사용되는 암호의 양자 후 보안강도를 확인하는 작업은 매우 중요하다. SHA-256에 대한 양자 공격 비용은 NIST에서 양자 후 보안강도 2에 대한 기준으로 제시되고 있지만, 아직까지 구체적인 양자 공격 비용으로 인용되는 연구 결과가 존재하지 않는다. 따라서 본 논문에서는 최적화된 depth를 가지는 SHA-256 양자 회로를 구현하여 보안강도 2에 대한 양자 공격 비용의 기준을 제시하고자 하였다.

구현 기법으로 out-of-place 형태의 Carry-save 덧셈기를 사용하여 덧셈기-depth를 1로 최적화하였으며, 라운드 후반부에서 사용되는 큐비트를 중간에 빌려 연산에 사용한 뒤 다시 반환함으로써 약 900개의 큐비트를 절약하였다. 또한 사용되는 연산에 특성에 맞게 AND 게이트 적용하고, 연산 대상 및 순서를 조정함으로써 최적화된 Toffoli-depth와 T-depth, T 게이트를 비롯한 양자 게이트, 큐비트 수를 가지도록 하였다. 결과적으로 이전 연구 대비 큐비트와 Toffoli-depth 측면에서 46%의 성능 개선을 보였다.

또한 구현한 SHA-256 회로를 기반으로 Grover 공격 비용을 계산하였으며, NIST의 Grover 공격 비용 추정 방식으로 계산한 결과 보안강도 2에 적합하게 구현되었음을 확인하였다. 본 연구는 양자컴퓨터 상에서 기존 암호의 보안강도를 확인하기 위한 기준 중 하나를 제시한 연구로써 그 의의가 있다.

참 고 문 헌

1. 국외문헌

Proctor, T., Rudinger, K., Young, K., Nielsen, E., & Blume-Kohout, R. (2022). Measuring the capabilities of quantum computers. Nature Physics, 18(1), 75-79.

Arute, F., Arya, K., Babbush, R., Bacon, D., Bardin, J. C., Barends, R., ... & Martinis, J. M. (2019). Quantum supremacy using a programmable superconducting processor. Nature, 574(7779), 505-510.

Shor, P.W. (1994) Algorithms for quantum computation: Discrete logarithms and factoring. In Proceedings of the 35th IEEE Annual Symposium on Foundations of Computer Science, 124-134.

Grover, L.K. (1996) A fast quantum mechanical algorithm for database search. In Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing, Philadelphia, 212-219.

Grassl, M., Langenberg, B., Roetteler, M., Steinwandt, R. (2016) Applying Grover's algorithm to AES: Quantum resource estimates. In Post-Quantum Cryptography; Springer: Berlin/Heidelberg, 29-43.

Jaques, S., Naehrig, M., Roetteler, M., & Virdia, F. (2020). Implementing Grover oracles for quantum key search on AES and LowMC. In Advances in Cryptology-EUROCRYPT 2020: 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10-14, 2020, Proceedings, Part II 30 (pp. 280-310). Springer International Publishing.

Selinger, P. (2013). Quantum circuits of T-depth one. Physical Review A,

87(4), 042302.

Wang, S., Baksi, A., & Chattopadhyay, A. (2023). A Higher Radix Architecture for Quantum Carry-lookahead Adder. arXiv preprint arXiv:2304.02921.

Cuccaro, S.A., Draper, T.G., Kutin, S.A., Moulton, D.P. (2004) A new quantum ripple-carry addition circuit. arXiv 2004, arXiv:quant-ph/0410184.

Takahashi, Y., Tani, S., & Kunihiro, N. (2009). Quantum addition circuits and unbounded fan-out. arXiv preprint arXiv:0910.2530.

Draper, T.G., Kutin, S.A., Rains, E.M., Svore, K.M. (2004) A logarithmic-depth quantum carry-lookahead adder. arXiv 2004, arXiv:quant-ph/0406142.

Gidney, C. (2018). Halving the cost of quantum addition. Quantum, 2, 74.

Gossett, P. (1998). Quantum carry-save arithmetic. arXiv preprint quant-ph/9808061.

Jang, K., Baksi, A., Kim, H., Song, G., Seo, H., & Chattopadhyay, A. (2022). Quantum analysis of aes. Cryptology ePrint Archive.

Oh, Y., Jang, K., Yang, Y., & Seo, H. (2023). Optimized Quantum Implementation of SEED. Cryptology ePrint Archive.

Gallagher, P., & Director, A. (1995). Secure hash standard (shs). FIPS PUB, 180, 183.

Amy, M., Di Matteo, O., Gheorghiu, V., Mosca, M., Parent, A., & Schanck, J. (2016, August). Estimating the cost of generic quantum pre-image attacks on SHA-2 and SHA-3. In International Conference on Selected Areas in Cryptography (pp. 317-337). Cham: Springer International Publishing.

Kim, P., Han, D., & Jeong, K. C. (2018). Time-space complexity of quantum search algorithms in symmetric cryptanalysis: applying

to AES and SHA-2. *Quantum Information Processing*, *17*, 1-39.

Lee, J., Lee, S., Lee, Y. S., & Choi, D. (2023). T-depth reduction method for efficient SHA-256 quantum circuit construction. IET Information Security, 17(1), 46-65.

Oonishi, K., Tanaka, T., Uno, S., Satoh, T., Van Meter, R., & Kunihiro, N. (2021). Efficient construction of a control modular adder on a carry-lookahead adder using relative-phase Toffoli gates. *IEEE Transactions on Quantum Engineering*, *3*, 1-18.

Khalus, V. I. (2019). T-count Optimization of Quantum Carry Look-Ahead Adder.

Kim, H., Lim, S., Wang, S., Jang, K., Baksi, A., Chattopadhyay, A., & Seo, H. (2024). Tree-based Quantum Carry Save Adder. Cryptology ePrint Archive.

ABSTRACT

Depth Optimization of Quantum Circuits for SHA-256 Hash Function

Lim, Se-Jin

Major in IT Convergence Engineering

Dept. of IT Convergence Engineering

The Graduate School

Hansung University

The SHA-256 hash function is used in various fields such as data integrity, authentication, and digital signatures. SHA-256 is considered to have a security strength of 128-bit, but its safety is threatened by the rapid advancement of quantum computers. This is because the time complexity of brute-force attacks reduces from $O(N)$ to $O(\sqrt{N})$ when using Grover's algorithm, an exhaustive search acceleration algorithm that operates on quantum computers. By applying Grover's algorithm to SHA-256, brute-force attacks can be performed with just 2^{64} attempts instead of 2^{128}, thus reducing the security strength from 128-bit to 64-bit. To counter these security threats, research is actively being conducted to estimate the quantum attack resources for current cryptography and to assess the security strength on quantum computers. The point at which a high-performance quantum computer capable of

operating an attack circuit applying Grover's algorithm to a specific cipher is developed can be seen as the time when the security strength of that cipher is halved. To apply Grover's algorithm, the target algorithm must be implemented in a quantum circuit, and the more optimized the circuit implementation is, the lower the hacking cost will be due to the minimized quantum resources used. NIST estimates the security strength of ciphers on quantum computers by comparing the quantum attack circuit cost of a specific cipher with the quantum cost required for quantum attack circuits of AES, SHA2, and SHA3. NIST measures the complexity of quantum attacks by the size of the quantum circuit (the total number of used quantum gates × Full circuit depth). Therefore, efficiently and accurately estimating the resources for the SHA-256 attack circuit is a very important task for verifying the safety of the cipher. Additionally, to reduce the complexity of quantum attacks, the overall circuit depth must be optimized. Also, the T gates used in the decomposition of Toffoli gates have a high implementation cost, so optimizing resources related to T gates, such as the number of T gates and T-depth, is very important. The optimization of Toffoli-depth leads to T-depth optimization, and using an optimized Toffoli gate decomposition technique can reduce T gate-related resources. Thus, this paper proposes a quantum circuit with optimized depth, including T-depth, for SHA-256, one of the standard algorithms in quantum computing, and estimates the Grover attack cost based on this.

【Keywords】Quantum Computer, Grover Algorithm, Implementation of Quantum Circuit, Depth Optimization, T-depth Optimization, SHA-256 Hash Function, Preimage Attack, Carry-save Adder