석사학위논문

LSH 해시함수에 대한 양자회로 구현 및 Grover 알고리즘 적용

2023년

한 성 대 학 교 대 학 원

I T 융 합 공 학 과

I T 융 합 공 학 전 공

송 경 주

석 사 학 위 논 문
지도교수 서화정

LSH 해시함수에 대한 양자회로 구현 및 Grover 알고리즘 적용

Implementation of quantum circuit and application of Grover algorithm for LSH hash function

2022년 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

송 경 주

석 사 학 위 논 문
지도교수 서화정

LSH 해시함수에 대한 양자회로 구현 및 Grover 알고리즘 적용

Implementation of quantum circuit and application of Grover algorithm for LSH hash function

위 논문을 공학 석사학위 논문으로 제출함

2022년 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

송 경 주

송경주의 공학 석사학위 논문을 인준함

2022년 12월 일

심사위원장 최 원 석 (인)

심 사 위 원 구 동 영 (인)

심 사 위 원 서 화 정 (인)

국 문 초 록

LSH 해시함수에 대한 양자회로 구현 및 Grover 알고리즘 적용

한 성 대 학 교 대 학 원
I T 융 합 공 학 과
I T 융 합 공 학 전 공
송 경 주

양자컴퓨터의 Grover 알고리즘은 해시함수에 대해 pre-image attack을 가속화 시켜 n-bit 보안 레벨의 해시함수를 $n/2$-bit 보안 레벨로 감소시킨다. Grover 알고리즘을 사용한 quantum pre-image attack을 수행하기 위해서는 공격 대상 해시함수가 Grover 내부 Oracle에 양자회로로 구현되어야 한다. 이러한 연구 동기로 본 논문에서는 한국 표준 해시함수인 LSH에 대한 최초의 양자회로를 제안한다. LSH 양자회로를 양자 자원의 효율적인 측면에 따라 Sequential 구조의 LSH 양자회로와 Parallel 구조의 LSH 양자회로로 나누어 제시하고 두 양자회로에 대한 양자자원 추정 결과를 통해 Grover 공격 자원을 계산하고 평가한다. Grover 공격에 필요한 양자 게이트를 하위 수준의 T+Clifford 게이트로 분해하고, 양자자원 Cost를 계산하여 두 양자회로의 양자게이트 및 양자회로 Depth에 대한 종합적인 자원 trade-off를 분석한다. 양자 자원 분석 결과 Sequential LSH 양자회로가 큐비트 수 측면에서 효율적으로 구현되었으며 Parallel LSH 양자회로가 Depth 측면에서 효율적으로 구

현되었음을 확인한다.

【주요어】 Grover 알고리즘, LSH 해시함수, 양자 pre-image 공격, 양자 회로, 양자 컴퓨터

목 차

표 목 차

그 림 목 차

수 식 목 차

알 고 리 즘 목 차

1. 서론

양자 컴퓨터는 중첩 성질의 큐비트를 사용하여 특정 문제에 대해 클래식 컴퓨터보다 훨씬 빠르게 문제를 해결할 수 있다고 알려져 있다. 양자 컴퓨터에서 동작하는 대표적인 양자 알고리즘으로는 Shor 알고리즘과 Grover 알고리즘이 있다. Shor 알고리즘은 대표적인 공개키 암호 방식인 Riverst-Shamir-Adleman(RSA) 및 Elliptic Curve Cryptography(ECC)에서 NP-hard problem로 여겨지는 인수분해 및 이산로그 문제를 다항시간 내에 해결하여 공개키 암호에 대해 취약성을 제공한다. Grover 알고리즘은 Advanced Encryption Standard(AES), Data Encryption Standard(DES) 등의 n-bit 대칭키 알고리즘에 대해 brute-force attack query를 2^n에서 $2^{n/2}$ 로 줄이며 Secure Hash Algorithm(SHA), LSH 등 해시함수에 대해 pre-image attack을 가속화 한다. 대규모 양자컴퓨터에 대응하기 위해 National Institute of Standards and Technology (NIST) 에서는 양자 후 시대에 안전한 Post-Quantum Cryptography(PQC) 표준화 대회를 2022년 4 Round 까지 진행하였다. [표 1-1]과 [표 1-2]는 각각 3 Round에서 발표된 PQC Finalists와 Alternates를 보여준다.

[표 1-1] Post-Quantum Cryptography(PQC) 3 Round Finalist

국가	알고리즘	기반문제	기능
북미	NTRU	격자	PKE/KEM
	Rainbow	다변수	전자서명
유럽	CRYSTALS-KYBER	격자	PKE/KEM
	SABER		
	Classic McEliece	코드	
	CRYSTALS-DILITHIUM	격자	전자서명
	FALCON		

[표 1-2] Post-Quantum Cryptography(PQC) 3 Round Alternates

국가	알고리즘	기반문제	기능
북미	FrodoKEM	격자	PKE/KEM
	NTRU-Prime		
	SIKE	아이소제니	
유럽	HQC	코드	
	BIKE		
	GeMMS	다변수	전자서명
	SPHINCS+	해시	
	Picnic	영지식 증명	

NIST에서는 2022년 7월 1개의 공개키 암호 알고리즘(PKE/KEM)과 3개의 전자서명 알고리즘을 최종 PQC로 선발하였으며 [표 1-3]과 같다.

[표 1-3] Post-Quantum Cryptography(PQC) 4 Round 최종 암호군

PKE/KEM	전자서명
CRYSTALS-KYBER	CRYSTALS-DILITHIUM
	FALCON
	SPHINCS+

양자 알고리즘을 동작하기 위해서는 공격 대상 암호가 가역 회로로 구현되어야 하며 대규모 양자컴퓨터의 발전으로 양자 컴퓨터의 가용 양자 자원이 암호 공격에 필요한 자원에 도달했을 때, 해당 암호가 깨질 것으로 예상한다. 이러한 연구 동기로 대칭암호와 해시함수를 가역 양자회로로 구현하여 Grover 알고리즘 공격에 필요한 가용자원을 추정하는 연구들이 활발하게 진행되고 있다. 현재의 소규모 양자 컴퓨터는 사용 가능한 큐비트 수와 오류 등의 성능 한계로 인해 실제 암호 공격 동작이 어렵다. 양자회로는 Width(너

비)와 Depth(깊이)로 나타낼 수 있는데, 너비는 물리적인 큐비트 수를 의미하며 Depth는 입력에서 출력까지의 경로에 대한 최대 길이를 의미한다. 현재 세계적인 양자 컴퓨터 개발 기업들은 물리적인 큐비트 수를 달성하기 위한 개발을 활발히 진행 중이다. 양자 회로의 Depth는 물리적인 연산 시간을 나타내며 양자 하드웨어에서 실행되는 양자 연산에 필요한 시간(i.e. 시간 복잡도)으로 이어진다. 또한, Depth가 클수록 양자 컴퓨터의 오류가 커지므로 물리적인 큐비트 수를 달성하더라도 큰 Depth의 양자회로는 오류로 인해 동작이 어렵다. 이러한 연구동기로 앞선 많은 연구들은 두 가지 방식으로 나누어 최적화 양자회로 구현을 진행하였으며 양자회로 구현에서 큐비트 수와 오류(i.e. Depth)를 줄이기 위한 최적화 방식을 적용한 양자회로를 제안하였다. 결과적으로 양자자원 최적화는 물리적인 양자 자원의 최적화뿐만 아니라 양자회로의 Depth를 최적화 하는 방식 모두 중요하다.

본 논문에서는 LSH 해시함수에 대해 큐비트, 양자 자원, Depth를 줄이는 방식으로 양자회로를 구현하였다. LSH 양자회로 구현은 순차적 구조의 Sequential LSH 양자회로와 병렬 구조의 Parallel LSH 양자회로 두 가지로 나누어 진행하였다. 순차적 구조의 Sequential LSH 양자회로는 큐비트 수를 줄이는 것에 가장 최적화 된 방식을 채택하여 설계되었으며 병렬 구조의 Parallel LSH 양자회로는 Depth를 줄이는 것에 가장 최적화 된 방식을 채택하여 설계하였다. 두 가지 방식의 양자회로에 적용된 최적화 방식은 제안기법에서 자세히 설명한다. 본 논문에서는 두 가지 방식에 대한 양자회로를 제시하며 LSH 해시함수 Grover 공격에 필요한 양자자원을 추정한다. 순차적 구조의 Sequential LSH 양자회로는 더 적은 양자 자원을 사용하지만 매우 큰 Depth로 구현되었으며 병렬 구조의 Parallel LSH 양자회로는 순차적 구조의 양자회로와 비교했을 때 효율적인 양자 자원 trade-off를 통해 훨씬 작은 Depth로 구현 되었다. 효율적인 양자 자원 trade-off 결과는 평가에서 자세히 설명한다.

1.1 Contribution

이것은 최초의 LSH 해시함수 양자회로 구현이며 LSH 해시함수에 대한 두 가지 방식의 최적화 양자회로를 제시한다. 첫 번째로 제시하는 Sequential 구조의 LSH 양자회로는 큐비트 수에 최적화 되도록 설계했으며 두 번째로 제시하는 Parallel 구조의 LSH 양자회로는 양자회로 Depth에 최적화 되도록 설계하였다. 마지막으로 두 가지 양자회로를 사용하여 LSH 해시함수에 대한 Grover 알고리즘 공격 비용을 추정하고 양자자원 trade-off를 평가한다.

따라서 본 논문의 Contribution을 요약하면 다음과 같다:

1. 큐비트 수에 최적화 된 Sequential 구조의 LSH 양자회로 구현 제시
2. 양자회로 Depth에 최적화 된 Parallel 구조의 LSH 양자회로 구현 제시
3. LSH 해시함수에 대한 Grover 알고리즘 공격 비용 추정
4. Sequential 구조와 Parallel 구조의 양자자원 trade-off 결과 비교

2. 관련연구

2.1 양자 컴퓨터

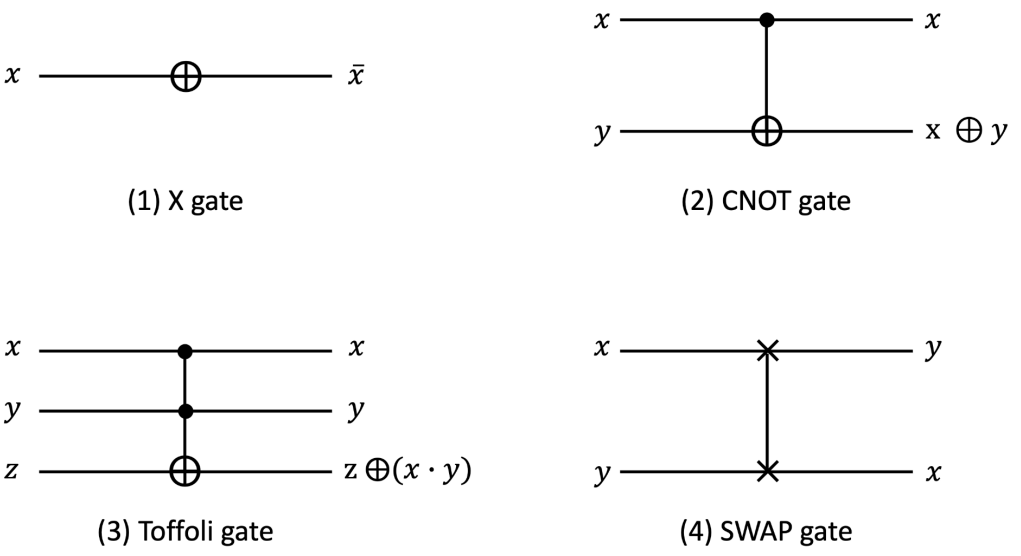
양자컴퓨터는 큐비트의 중첩 및 얽힘의 양자역학적인 상태를 활용하여 계산을 수행하는 컴퓨터이다. 큐비트의 중첩 성질로 인해 n개의 큐비트로 2^n개의 경우를 표현하고 2^n개의 모든 경우에 대한 연산을 한 번에 수행할 수 있다. 큐비트는 Hadamard 게이트를 통해 중첩상태를 가질 수 있다. 중첩 상태의 큐비트는 다음과 같이 표현된다.

$$|\psi\rangle = \alpha|1\rangle + \beta|0\rangle$$

중첩 상태 큐비트에 대한 모든 측정은 항상 두 개의 고유 값 중 하나를 생성하며 어느 것 인지는 알 수 없다. α 및 β 는 확률의 진폭을 나타내며 값이 1인 결과 $|1\rangle$의 측정 확률은 α^2 이며 값이 0인 결과 $|0\rangle$의 측정 확률은 β^2이다. 해당 상태를 정규화 하면 다음 방정식이 보장된다.

$$|\alpha|^2 + |\beta|^2 = 1$$

양자 컴퓨터는 디지털 논리회로에서 사용하는 디지털 논리 게이트와 유사한 양자 게이트를 사용하여 큐비트의 상태를 제어하며 연산을 진행한다. 이러한 큐비트와 양자 게이트로 구성된 회로를 양자 회로(quantum circuit)라고 한다. 양자회로에 입력되는 큐비트는 크게 제어(control) 큐비트와 타겟(target) 큐비트로 나눠진다. 제어 큐비트는 연산에 영향을 주는 큐비트이며 값이 바뀌지 않는다. 타겟 큐비트는 연상 대상이 되는 큐비트로서 결과 값이 저장된다. 양자 게이트는 측정을 제외한 모든 연산에 대해 가역적인 특징이 있어 역연산이 가능하다. [그림 2-1]은 대표적인 양자 게이트인 X 게이트, CNOT 게이트, Toffoli 게이트, Swap 게이트를 보여준다.



[그림 2-1] 양자 게이트 (1) X 게이트. (2) CNOT 게이트, (3) Toffoli 게이트, (4) SWAP 게이트

(1) X 게이트 : X 게이트는 디지털 논리회로의 NOT 게이트와 동일한 연산을 수행하는 양자게이트이다. 단일 입력 큐비트로 동작하며 입력된 큐비트의 상태가 반전된다. 본 논문의 양자회로 알고리즘에서 X(x) 로 작성된다.

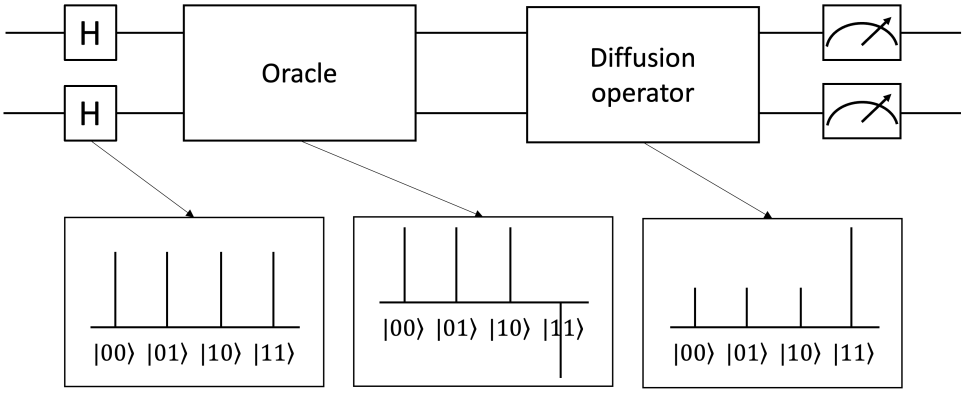
(2) CNOT 게이트 : CNOT 게이트는 두 개의 입력 큐비트로 동작하는 양자게이트이다. 두 개의 입력 중 한 개는 제어 큐비트가 되며 나머지 한 개는 타겟 큐비트가 된다. 제어 큐비트의 상태는 입력된 상태로 유지되며 타겟 큐비트는 입력 큐비트가 1일 때 반전된다. 본 논문의 양자회로 알고리즘에서 y←CNOT(x, y) 으로 작성된다.

(3) Toffoli 게이트 : Toffoli 게이트는 세 개의 입력 큐비트로 동작하는 양자게이트이다. 세 개의 입력 중 두 개는 제어 큐비트가 되며 나머지 한 개는 타겟 큐비트가 된다. 두 개의 제어 큐비트의 상태는 입력된 상태로 유지되며 타겟 큐비트는 두 개의 입력 큐비트가 모두 1일 때 반전된다. 본 논문의 양자회로 알고리즘에서 z←Toffoli(x, y, z) 로 작성된다.

(4) SWAP 게이트 : SWAP 게이트는 두 개의 입력 큐비트로 동작하는 양자 게이트이다. 입력된 두 개의 큐비트는 상태를 유지하며 서로 물리적인 위치만 변경된다. 본 논문의 양자회로 알고리즘에서 SWAP(x, y)으로 작성된다.

2.2 Grover 알고리즘

Grover 알고리즘은 1996년 Lov Grover가 제안한 양자 알고리즘이다. 블랙박스 함수에 대한 특정 출력을 생성하는 입력을 높은 확률로 찾기 위해 N개의 입력 도메인을 탐색한다. 일반 컴퓨터에서 정렬되지 않은 N개의 데이터를 탐색하기 위해서는 N번의 검색이 필요하다. 하지만 Grover 알고리즘을 사용하면 N개의 데이터를 탐색하기 위해 $\sqrt{N}$번의 검색만으로 알맞은 입력을 높은 확률로 찾아낸다. 즉, 기존 컴퓨터에서 $O(N)$의 계산 복잡도를 가진 탐색 연산을 $O(\sqrt{N})$ 계산 복잡도로 감소시킨다. 이러한 Grover 탐색 알고리즘의 성능이 알려지면서 블록 암호에 대한 brute-force attack과 해시함수에 대한 pre-image attack에 사용하는 방법들이 연구되었다. Grover 알고리즘은 brute-force attack 및 pre-image attack 에 필요한 시간을 단축시켜 대칭 키 암호와 해시함수의 보안을 위협한다.



[그림 2-2] Grover 알고리즘 (answer x = 11)

Grover 알고리즘은 내부에서 Oracle과 Diffusion operator의 반복으로 동작하며 [그림 2-2]는 answer x = 11을 찾는 전체적인 Grover 알고리즘

동작 과정을 보여준다. 먼저, 입력된 큐비트에 모두 Hadamard 게이트를 적용하여 큐비트를 중첩상태로 만든다. Oracle 함수 f(x)는 입력된 데이터가 정답일 때 1을 반환하고 정답을 나타내는 큐비트 상태 |11⟩의 위상을 반전시킨다. Diffusion operator는 Oracle에서 반환된 정답 큐비트의 진폭을 증폭시켜 관측 확률을 높인다. Oracle과 Diffusion operator의 반복을 통해 정답 큐비트의 관측 확률이 임계점을 초과하며 임계 값을 초과했을 때 관측된 큐비트 값이 높은 확률로 정답이 된다.

2.2.1 Grover 알고리즘을 사용한 블록암호 brute-force attack

Grover 알고리즘을 사용한 블록암호 brute-force attack은 블록암호의 평문-암호문 쌍을 알고 있을 때 수행할 수 있는 Known-Plaintext Attack (KPA)이다. 블록암호에서 사용된 n-bit 키는 brute-force attack의 대상이다. Grover 알고리즘을 동작하기 위해서는 대상 암호에 대한 양자회로가 필요하며 양자 회로는 Oracle 내부에 위치한다. Oracle 내부에 위치한 암호화 양자회로는 알려진 평문과 n-bit 키를 큐비트 입력으로 하여 암호화를 진행한다. 이때, 평문은 이미 알고 있으므로 X 게이트를 사용하여 값을 설정한다. n-bit 키는 Grover 알고리즘에서 찾아야 하는 정답이므로 Hadamard 게이트를 사용하여 중첩 상태로 설정한다. Oracle은 암호화에 중첩 상태의 키를 사용하므로 단일 쿼리로 모든 키에 대한 암호문을 가질 수 있다. 생성된 모든 암호문에서 알려진 암호문과 동일한 암호문을 생성하는 키 상태를 찾기 위해 Oracle 끝에 알려진 암호문을 설정한다. Oracle에서는 정답 키의 부호가 반전되며 반환된 정답 키의 관측 확률을 Diffusion operator에서 증폭 시킨다. 올바른 키는 해당 과정을 $\left\lfloor \frac{\pi}{4}\sqrt{2^n} \right\rfloor$ 번 반복하면 찾을 수 있다고 알려져 있다.

2.2.2 Grover 알고리즘을 사용한 해시함수 pre-image attack

Grover 알고리즘을 사용한 해시함수 pre-image attack은 블록 암호의 경우와 유사하다. Grover 알고리즘을 동작하기 위해서는 대상 해시함수에 대

한 양자 회로가 필요하며 양자 회로는 Oracle 내부에 위치한다. Oracle 내부에 위치한 암호화 양자회로는 입력된 메시지로 해시를 진행한다. Oracle에서는 중첩 상태의 메시지를 사용하므로 단일 쿼리로 모든 평문에 대한 해시 값을 가질 수 있다. 정답 메시지 상태를 찾기 위해 Oracle 끝에 알려진 해시 값을 설정한다. Oracle에서는 정답 메시지의 부호가 반전되며 반환된 정답 메시지의 관측 확률을 Diffusion operator에서 증폭 시킨다. 올바른 메시지는 해당 과정을 $\left\lfloor \frac{\pi}{4}\sqrt{2^n} \right\rfloor$ 번 반복하면 찾을 수 있다고 알려져 있다.

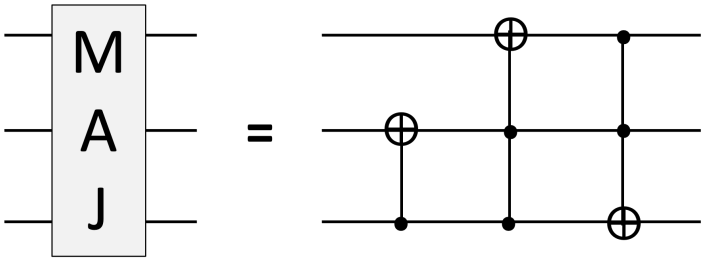
2.3 Quantum addition

양자회로에서 사칙연산을 수행하기 위해서는 연산이 양자회로로 구현되어야 한다. 양자회로에 대한 덧셈 구현은 다양하게 연구되고 있으며 사용 양자 게이트, 큐비트 수, Depth 등에 차이를 가진다. 그 중 본 논문의 Sequential LSH 양자회로에서 사용한 Simple ripple-carry 양자 덧셈기와 Parallel LSH 양자회로에서 사용한 Ripple-carry 양자 덧셈기는 다음과 같다.

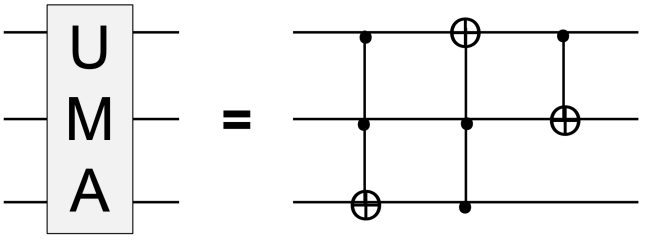
2.3.1 Simple ripple-carry 양자 덧셈기

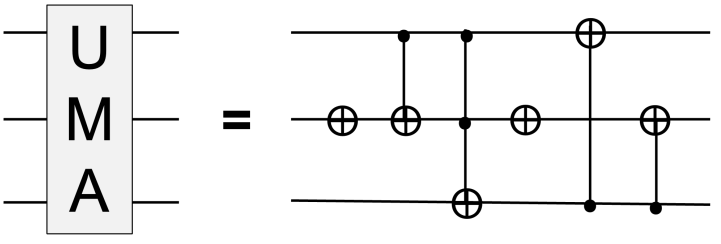
Simple ripple-carry 양자 덧셈기는 덧셈 대상이 되는 두 n-bit 입력 $a = a_0 \cdots a_n$, $b = b_0 \cdots b_n$ 중 한 곳에 덧셈 결과가 저장된다. 해당 덧셈 양자회로에서는 이전 비트 덧셈에서 발생하는 carry 큐비트를 저장하기 위한 1 큐비트를 추가로 사용한다. 따라서 양자회로의 입력으로는 두 n-bit 덧셈 대상이 되는 $2n$ 큐비트와 carry 값을 임시로 저장하기 위한 1 큐비트가 사용된다. 덧셈에서 1-carry 큐비트 c는 사용 후 0으로 리셋 되기 때문에 다음 덧셈 연산에서 재사용 할 수 있다. Simple ripple-carry 덧셈기는 총 (6n-2)의 양자회로 Depth를 가진다. 덧셈을 마친 후 큐비트의 상태는 ADD(a, b, c) = (a, $a+b$, c) 과 같다. Simple ripple-carry 양자회로는 [그림 2-3]의 MAJ 게이트와 [그림 2-4]의 UMA 게이트로 구성된다. MAJ 게이트는 1개의 CNOT 게이트, 2개의 Toffoli 게이트로 구성되어 있다. UMA 게이트는 두 가지 구

현 방식으로 나누어진다. [그림 2-4]에서 위의 UMA 게이트는 2개의 CNOT 게이트와 1개의 Toffoli 게이트로 구현되었으며 아래의 UMA 게이트는 1개의 Toffoli 게이트, 3개의 CNOT 게이트, 2개의 X 게이트로 구성되어 있다. $n=6$에 대한 simple ripple-carry 덧셈기의 전체 동작은 [그림 2-5]와 같다. 입력 $a=a_0\cdots a_5$, $b=b_0\cdots b_5$ 은 MAJ 게이트와 UMA 게이트만으로 덧셈이 진행되며 덧셈 후 $a=a_0\cdots a_5$ 는 값을 유지하고 $b=b_0\cdots b_5$ 는 덧셈 결과를 저장한다. 사용된 carry 큐비트 c는 0으로 리셋 되어 다음 덧셈에서 재사용 할 수 있다.

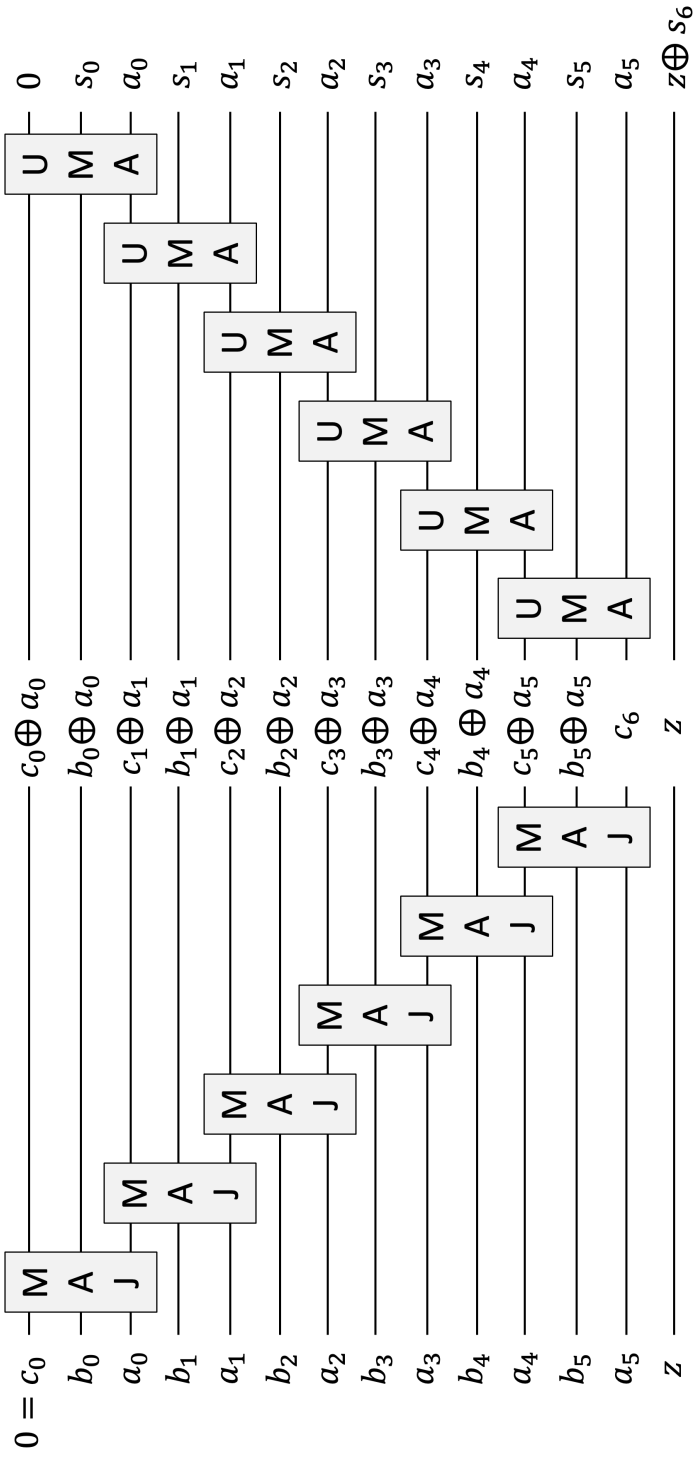


[그림 2-3] Simple ripple-carry MAJ 게이트





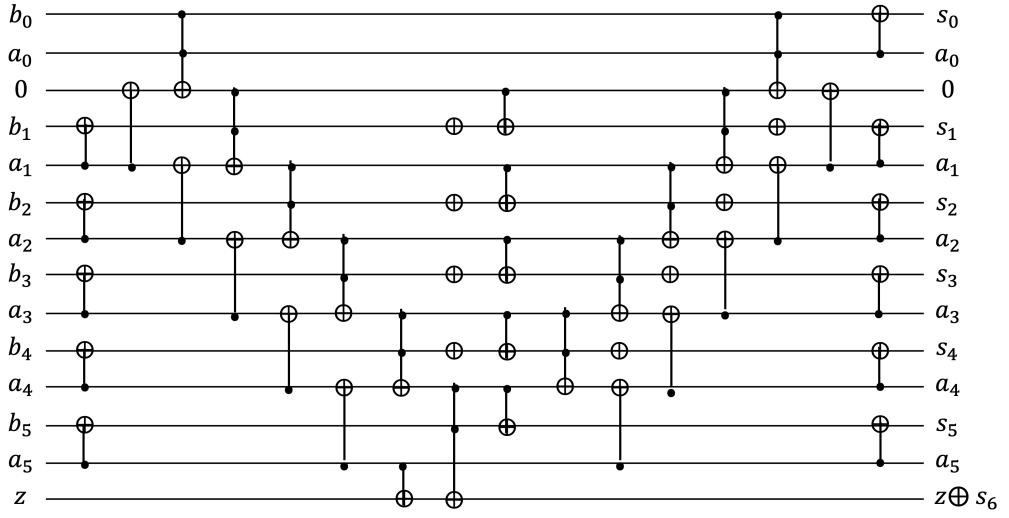
[그림 2-4] Simple ripple-carry UMA 게이트



[그림 2-5] Simple ripple-carry 양자 덧셈기 ($n=6$)

2.3.2 Ripple-carry 양자 덧셈기

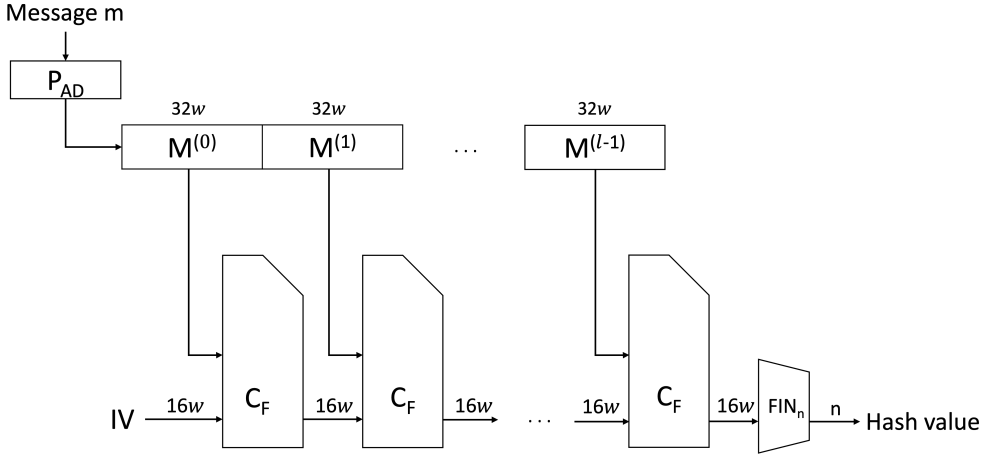
Ripple-carry 양자 덧셈기는 덧셈 대상이 되는 두 n-bit 입력 $a = a_0 \cdots a_n$, $b = b_0 \cdots b_n$ 중 한 곳에 덧셈 결과가 저장된다. 해당 덧셈 양자회로에서는 이전 비트 덧셈에서 발생하는 carry 큐비트를 저장하기 위한 1 큐비트를 추가로 사용한다. 따라서 양자회로의 입력으로는 두 n-bit 덧셈 대상이 되는 $2n$ 큐비트와 carry 값을 임시로 저장하기 위한 1 큐비트가 사용된다. 덧셈에서 1 carry 큐비트 c는 사용 후 0으로 리셋 되기 때문에 다음 덧셈 연산에서 재사용 할 수 있다. Ripple-carry 덧셈기는 총 (2n+3)의 양자회로 Depth를 가진다. 덧셈을 마친 후 큐비트의 상태는 ADD(a, b, c) = (a, $a+b$, c) 과 같다. $n = 6$에 대한 Ripple-carry 덧셈기의 전체 동작은 [그림 2-6]과 같다. 덧셈 후 $a = a_0 \cdots a_5$는 값을 유지하고 $b = b_0 \cdots b_5$는 덧셈 결과를 저장한다. 사용된 carry 큐비트 c는 0으로 리셋 되어 다음 덧셈에서 재사용 할 수 있다.



[그림 2-6] Ripple-carry 양자 덧셈기 ($n = 6$)

2.4 LSH 해시함수

LSH는 Korean Cryptographic Module Verification Program (KCMVP)에서 인증 받은 한국 국가 표준 해시함수이다. LSH는 워드(w=32, 64) 단위로 동작하며 n비트의 해시함수를 출력한다. 이에 대해 LSH를 LSH-$8w$-n (family : LSH-256-224, LSH-256-256, LSH-512-224, LSH-512-256, LSH-512-384, LSH-512-512) 로 정의한다. LSH 해시함수는 [그림 2-7] 과 같이 크게 initialization, compression, final 단계로 진행된다.



[그림 2-7] LSH 해시함수 전체 동작

Initialization 단계에서는 입력된 메시지를 w의 배수가 되도록 패딩 시킨 후 w크기의 메시지 블록들로 나눈다. 패딩은 입력된 메시지 m의 맨 뒤에 “1”을 덧붙인 뒤 길이가 $32wt$ ($t = \lceil (|m|+1)/32w \rceil$) 비트가 되도록 “0”이 채워진다. 패딩 된 메시지 $m_p = m_0 \| m_1 \| \cdots \| m_{32wt-1}$는 $4wt$ 바이트의 배열 $m_a = (m[0], \cdots, m[4wt-1])$로 표현될 수 있으며 $4wt$ 바이트 배열 m_a 은 [수식 2-1]을 통해 $32t$ 워드 배열의 $M = (M[0], \cdots, M[32t-1])$로 변환된다.

$$M[s] \leftarrow m[ws/8+(w/8-1)] \| \cdots \| m[ws/8+1] \| m[ws/8], \ (0 \le s \le 32t-1)$$

[수식 2-1] $32t$ 워드 배열 메시지 변환

[수식 2-1]로 변환된 워드 배열은 [수식 2-2]를 통해 t개의 메시지 블록 $M^{(0)}, M^{(1)}, \cdots, M^{(t-1)}$으로 나눈다.

$$M^{(i)} \leftarrow (M[32i], M[32i+1], \ \cdots, M[32i+31]), \ (0 \le i \le t-1)$$

[수식 2-2] 메시지 블록 분해

Chaining Variable (CV)는 Initialization Vector (IV)을 통해 초기화 되며 LSH-256-224, LSH-512-224 에서 정의된 IV는 각각 [표 2-1], [표 2-2]와 같다. IV의 데이터 포맷은 16진수이며 LSH-256에서는 512bit, LSH-512에서는 1024bit의 IV를 CV 초기화에 사용한다.

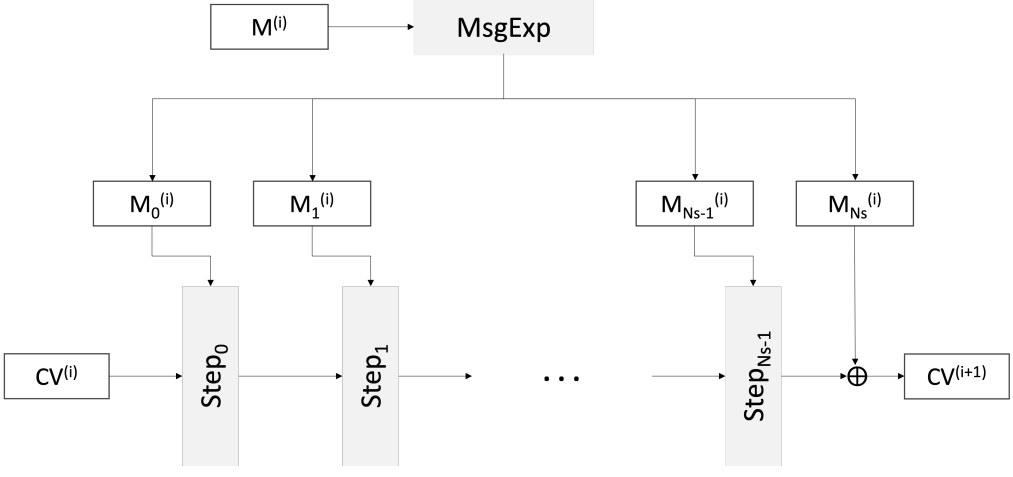
[표 2-1] LSH-256-224 에 대한 initialization vector (IV)

IV[0]	IV[1]	IV[2]	IV[3]
068608D3	62D8F7A7	D76652AB	4C600A43
IV[4]	**IV[5]**	**IV[6]**	**IV[7]**
BDC40AA8	1ECA0B68	DA1A89BE	3147D354
IV[8]	**IV[9]**	**IV[10]**	**IV[11]**
707EB4F9	F65B3862	6B0B2ABE	56B8EC0A
IV[12]	**IV[13]**	**IV[14]**	**IV[15]**
CF237286	EE0D1727	33636595	8BB8D05F

[표 2-2] LSH-512-224 에 대한 initialization vector (IV)

IV[0]	IV[1]	IV[2]	IV[3]
0C401E9FE8813A55	4A5F446268FD3D35	FF13E452334F612A	F8227661037E354A
IV[4]	**IV[5]**	**IV[6]**	**IV[7]**
A5F223723C9CA29D	95D965A11AED3979	01E23835B9AB02CC	52D49CBAD5B30616
IV[8]	**IV[9]**	**IV[10]**	**IV[11]**
9E5C2027773F4ED3	66A5C8801925B701	22BBC85B4C6779D9	C13171A42C559C23
IV[12]	**IV[13]**	**IV[14]**	**IV[15]**
31E2B67D25BE3813	D522C4DEED8E4D83	A79F5509B43FBAFE	E00D2CD88B4B6C6A

CV는 Compression function (CF) 내부에서 확장된 메시지 블록들을 사용하여 업데이트가 진행된다. 초기화 단계에서 생성된 메시지 블록들은 CF $W^{16} \times W^{32} \rightarrow W^{16}$ 의 입력으로 사용된다. CF 함수 동작은 다음 네 단계로 진행된다. (1) 메시지 확장($MsgExp$) : $W^{32} \rightarrow W^{16(N_s+1)}$, (2) 메시지 덧셈($MsgAdd$): $W^{16} \times W^{16} \rightarrow W^{16}$, 메시지 섞음($Mix$) : $W^{32} \rightarrow W^{16}$, (4) Word permutation($WordPerm$) : $W^{16} \rightarrow W^{16}$. [그림 2-8]은 CF 의 전체적인 동작을 보여준다.



[그림 2-8] LSH 해시함수의 Compression Function (CF)

Compression function CF 에 입력된 메시지 블록 $M^{(i)}$은 메시지 확장 함수 $MsgExp$ 에서 N_s+1개의 16 word(ω) 배열 $M_j^{(i)}\,(0 \le j \le N_s)$ 로 확장된다 (if $\omega=32$ then $N_s=26$, if $\omega=64$ then $N_s=28$). $CV^{(i)}$로 초기화된 $T=T[0],\cdots T[15]$ 는 $Step$ 함수에서 메시지 블록을 통해 업데이트 된다. CF의 내부함수인 단계 함수 $Step: W^{16}\times W^{16}\rightarrow W^{16}$ 은 다음과 같은 순서로 동작한다:

$$Step_j : WordPerm \circ Mix \circ MsgAdd \quad (0 \le j \le N_s-1)$$

메시지 확장 함수 $MsgExp$은 N_s+1개의 w 배열 $M_j^{(i)}$을 생성하며 생성된 메시지는 $Step$ 함수의 입력으로 사용된다. 메시지 생성 방식은 [수식 2-3]과 같다. [수식 2-3]에서 $\tau(l)$는 [표 2-3]에 나타난 Z_{16} 상에서의 치환을 의미한다.

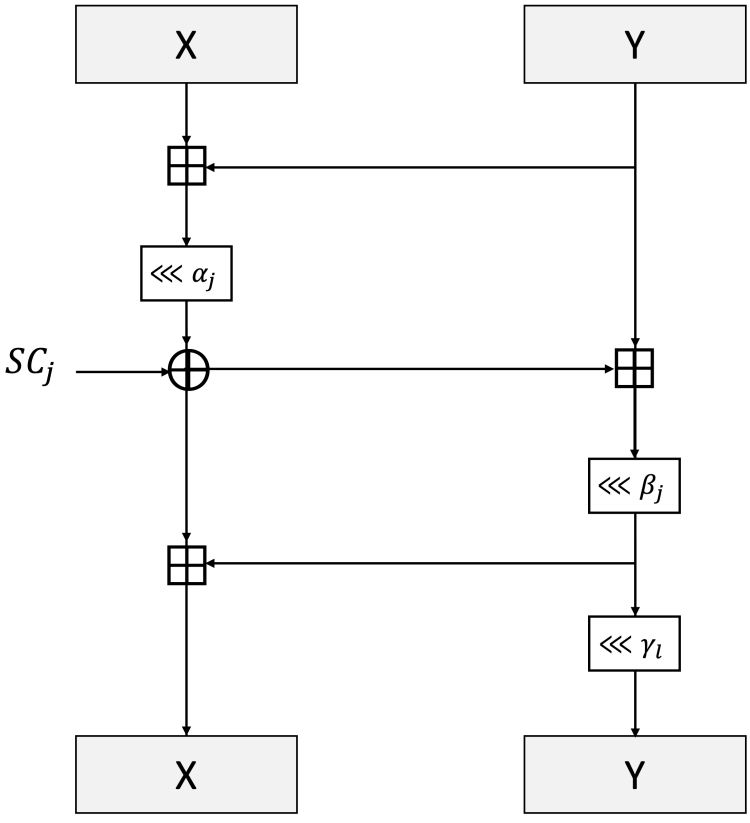
$$M_0^{(i)} \leftarrow M^{(i)}[0], M^{(i)}[1], \cdots, M^{(i)}[15]$$
$$M_1^{(i)} \leftarrow M^{(i)}[16], M^{(i)}[17], \cdots, M^{(i)}[31]$$
$$M_j^{(i)}[l] \leftarrow ADD(M_{j-1}^{(i)}, M_{j-2}^{(i)}[\tau(l)] \quad (0 \le l \le 15)$$

[수식 2-3] 메시지 확장 함수 $MsgExp$

[표 2-3] $MsgExp$ 함수 Z_{16} 상에서의 치환 $\tau(l)$

l	0	1	2	3	4	5	6	7
$\tau(l)$	3	2	0	1	7	4	5	6
l	8	9	10	11	12	13	14	15
$\tau(l)$	11	10	8	9	15	12	13	14

$MsgAdd$ 함수는 같은 위치의 두 16ω 배열 X, Y 를 XOR 연산한다 : $MsgAdd(X,Y) := (X[0]\oplus Y[0], \cdots, X[15]\oplus Y[15])$. Mix 함수는 한 개의 $T[l], T[l+8]\ (0 \le l \le 7)$ 쌍을 사용하여 T를 업데이트 한다. Mix 함수의 동작은 [그림 2-9] 및 [수식 2-4]와 같다. Mix 함수에서 사용되는 bit rotation 크기는 [표 2-4]와 같으며 bit rotation의 크기는 w와 $Step$ 함수 라운드의 even/odd에 따라 결정된다.



[그림 2-9] LSH 해시함수의 Mix 함수

$$X \leftarrow ADD(X, Y); \quad X \leftarrow X^{\lll \alpha_j}; \quad X \leftarrow X \oplus SC_j[l];$$
$$Y \leftarrow ADD(X, Y); \quad Y \leftarrow Y^{\lll \beta_j}; \quad X \leftarrow ADD(X, Y); \; Y \leftarrow Y^{\lll \gamma_l}$$

[수식 2-4] Mix 함수

[표 2-4] Mix 함수의 Bit rotation 크기

w	j	α_j	β_j	γ_0	γ_1	γ_2	γ_3	γ_4	γ_5	γ_6	γ_7
32	Even	29	1	0	8	16	24	24	16	8	0
	Odd	5	17								
64	Even	23	59	0	16	32	48	8	24	40	56
	Odd	7	3								

$WordPerm$ 함수는 [표 2-5]에 작성된 Z_{16} 상에서의 치환을 동작한다.

$$WordPerm(X) := (X[\sigma(0)], \cdots, X[\sigma(15)])$$

[표 2-5] $WordPerm$ 함수 Z_{16} 상에서의 치환 $\sigma(l)$

l	0	1	2	3	4	5	6	7
$\sigma(l)$	6	4	5	7	12	15	14	13
l	8	9	10	11	12	13	14	15
$\sigma(l)$	2	0	1	3	8	11	10	9

Final 함수 $FIN_n : W_{16} \rightarrow 0,1_n$은 CF을 통해 업데이트 된 최종 $CV^{(i)} = CV^{(t)}[0], \cdots, CV^{(t)}[15]$ 을 사용하여 해시 값을 출력한다. Final 함수 FIN_n은 [수식 2-5]를 따라 $8w$ 배열 $H = H[0], \cdots, H[7]$과 w-byte 배열 $h_b = h_b[0], \cdots, h_b[w-1]$을 통해 최종 해시 값 h를 생성한다.

$$H[i] = CV^{(t)}[i] \oplus CV^{(t)}[i+8] \ (0 \le i \le 7)$$
$$h_b[s] = H[(8s/w)^{\ggg (8s \bmod w)}][7:0] \ (0 \le s \le w-1)$$
$$h = (h_b[0] \| \cdots \| h_b[w-1])_{[0:n-1]}$$

[수식 2-5] Final 함수 FIN_n

3. 제안기법

본 논문에서는 LSH 해시함수의 모든 암호군 (LSH-256-224/256, LSH-512-224/256/384/512)에 대해 두 가지 방식의 양자회로를 제안한다. 우리는 먼저 내부의 모든 함수 동작에서 순차적으로 계산을 수행하는 Sequential 구조의 LSH 양자회로를 제안하고 두 번째로 일부 함수에서 병렬로 계산을 수행하는 Parallel 구조의 LSH 양자회로를 제안한다. 병렬 구조의 Parallel LSH 양자회로는 순차적 구조의 Sequential LSH 양자회로에 대해 양자 자원 trade-off를 통해 양자 회로의 총 Depth를 줄이기 위해 고안하였다. 우리는 순차적 구조의 LSH 동작에서 각 메시지 블록들이 독립적으로 연산 가능한 부분을 찾아 병렬 구조로 설계하였다. LSH 내부의 메시지 확장 $MsgExp$ 함수와 섞음 Mix 함수는 메시지 블록 단위로 독립적으로 연산되기 때문에 서로의 결과에 영향을 주지 않는 특징이 있다. 따라서 이 두 함수에 대해 병렬 양자회로 구조를 설계한다. 두 양자회로는 양자자원(e.g. 큐비트 수, 양자 게이트, Depth)을 줄이는 최적화 방식을 사용하여 설계되었으며 최초의 LSH 양자회로 구현이다. 제안하는 두 방식에 대한 양자회로를 활용하여 양자자원 추정 결과를 제시하고 Grover 알고리즘에 필요한 양자 자원을 계산한다. 마지막으로 제안하는 두 방식의 양자자원 trade-off 결과를 비교한다.

3.1 LSH 양자회로

순차적 구조의 Sequential LSH 양자회로에서 LSH-256-n은 $\omega = 32$ 단위로 해시함수를 진행하며 패딩 된 1024 bit 메시지 M을 저장하기 위한 1024개의 큐비트를 사용하고 LSH 해시함수 진행에서 라운드 함수의 입력이 되는 연결 변수 CV를 저장하기 위한 512개의 큐비트, 덧셈을 수행하기 위한 carry 큐비트 1개를 사용한다. LSH-512-n은 $\omega = 64$ 단위로 해시함수를 진행하며 패딩 된 2048 bit 메시지 M을 저장하기 위한 2048개의 큐비트를 사용하고 LSH 해시함수 진행에서 라운드 함수의 입력이 되는 연결 변수 CV를

저장하기 위한 큐비트 1024개, 덧셈을 수행하기 위한 carry 큐비트 1개를 사용한다.

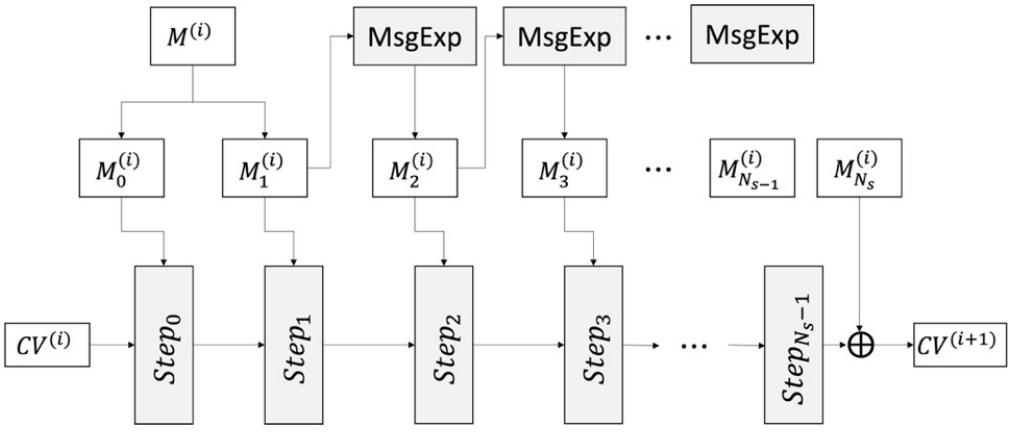
병렬 구조의 Parallel 양자회로에서 LSH-256-m은 $\omega = 32$ 단위로 해시 함수를 진행하며 패딩 된 1024 bit 메시지 M을 저장하기 위한 1024개의 큐비트를 사용하고 LSH 해시함수 진행에서 라운드 함수의 입력이 되는 연결 변수 CV를 저장하기 위한 큐비트 512개, 병렬 덧셈을 수행하기 위한 carry 큐비트 16개를 사용한다. LSH-512-m은 $\omega = 64$ 단위로 해시함수를 진행하며 패딩 된 2048 bit 메시지 M을 저장하기 위한 2048개의 큐비트를 사용하고 LSH 해시함수 진행에서 라운드 함수의 입력이 되는 연결 변수 CV를 저장하기 위한 1024개의 큐비트, 덧셈을 수행하기 위한 carry 큐비트 16개를 사용한다. LSH-256-n과 LSH-512-m 은 ω 단위, 단계 상수, 해시 값 출력에서 차이점을 가지며 전체적인 해시 과정은 동일하다. 순차적 구조의 Sequential LSH 양자회로와 병렬구조의 Parallel LSH 양자회로는 메시지 확장 함수 $MsgExp$와 Mix 함수에서 구조적인 차이점을 보이며 그 이외의 구조는 동일하다. 본 논문에서는 LSH 해시함수의 내부 동작 Initialization, Compression 순서로 양자회로를 설명한다.

3.1.1 Initialization 양자회로

양자회로의 Initialization 단계에서는 패딩 된 메시지를 저장하기 위한 큐비트와 연결변수 CV를 저장할 큐비트를 할당한다. 연결 변수 CV는 초기화 벡터 IV를 통해 초기화 되는데, IV는 이미 알고 있는 상수이므로 따로 큐비트를 할당하지 않는다. IV을 이진수로 나타내고, 인덱스가 1인 위치와 동일한 위치의 CV에 X 게이트를 수행한다. 해당 함수에서의 순차적 구조와 병렬 구조 양자회로 동작은 동일하다.

3.1.2 Compression 양자회로

Compression function CF 은 Initialization에서 패딩 된 메시지를 사용하여 초기화 된 연결변수 CV을 업데이트 한다. CF은 크게 메시지 확장 함수 $MsgExp$와 단계 함수 $Step$로 구성된다. 메시지 확장 함수 $MsgExp$에서는 기존 메시지 블록을 수식에 따라 여러개의 메시지 블록으로 확장시킨다. 단계 함수 $Step$은 $Step_j : WordPerm \circ Mix \circ MsgAdd \ (0 \le j \le N_s - 1)$ 로 진행된다. $MsgExp$에서는 Initialization 단계에서 패딩 된 메시지 블록을 $N_s + 1$ 개의 메시지 블록으로 확장 시킨다. 기존 LSH 알고리즘은 $MsgExp$에서 $N_s + 1$개의 워드 배열 $M_j^{(i)} (0 \le j \le N_s)$을 생성한 후 $Step$을 진행한다. 이러한 방식을 양자회로에 적용하면 $N_s + 1$개의 메시지를 저장하기 위한 큐비트를 할당해야 하므로 (LSH-256 : 13,824개, LSH-512 : 29,696개) 양자 자원 측면에서 비효율적이다. 따라서 큐비트 할당을 줄이기 위해 양자회로에서는 $MsgExp$ 함수와 $Step$ 함수를 반복적으로 동작하는 on-the-fly 방식을 사용하여 메시지 큐비트를 재활용하는 방식을 채택하였다. 따라서 기존 [그림 2-8]의 CF 의 전체적인 동작은 양자회로에서 그림 [그림 3-1]로 동작한다. 이러한 방식으로 이전 $Step$ 함수에서 사용한 워드 배열 메시지 큐비트를 다음 $MsgExp$ 함수에서 재사용 할 수 있다.



[그림 3-1] Compression function 양자회로 동작과정

MsgExp 에서는 [수식 2-3] 의 메시지 확장 식 $M_j^{(i)}[l] \leftarrow ADD(M_{j-1}^{(i)}, M_{j-2}^{(i)}[\tau(l)]), (0 \le l \le 15)$을 통해 j번째 워드 배열 M_j이 j-2 번째 워드 배열 M_{j-2}과 j-1 번째 워드 배열 M_{j-1}의 덧셈으로 생성된다. j번째 워드 배열 M_j을 *Step* 함수에서 사용 할 때, j-2번째 워드 배열 M_{j-2}은 이미 사용이 끝났으므로 값을 유지할 필요가 없으며 j-1번째 워드 배열은 M_{j+1}배열 생성에 필요하므로 값을 유지해야 한다. 즉, 두 번째 전의 워드 배열은 값을 유지할 필요가 없으므로 다음 메시지 생성에 재사용하는 방법을 사용할 수 있다. 따라서 해당 방식으로 두 개의 워드 배열 M_{k-2}, M_{k-1}을 저장할 큐비트만 할당하고 다음 워드 M_k 배열 생성에 M_{k-2}의 큐비트를 재사용한다. 결과적으로 이러한 최적화 방식으로 LSH-256에서는 전체 27개의 워드 메시지 블록을 저장하기 위해 사용하는 13,824개의 큐비트 대신 두 개의 워드 메시지 블록을 저장하기 위한 1,024개의 큐비트를 사용하여 총 12,800개의 큐비트 사용을 줄인다. LSH-512에서는 전체 29개의 워드 메시지를 저장하기 위해 사용하는 29,696개의 큐비트 대신 두 개의 워드 메시지 블록을 저장하기 위한 2,048개의 큐비트를 사용하여 총 27,648개의 큐비트 사용을 줄인다. 메시지 확장은 [수식 2-3]으로 진행되는데, j번째 워드 배열 $M_j^{(i)}$은 (j-2)번째 워드 배열 $M_{j-2}^{(i)}$과 (j-1)번째 워드 배열 $M_{j-1}^{(i)}$의 덧셈으로 생성된다. Compression function *CF*에 대한 양자회로 설명은 LSH-256 기준으로 진행한다.

순차적 구조의 Sequential LSH 양자회로에 대한 전체적인 Compression function *CF* 알고리즘은 [Algorithm 3-1]과 같다. Line 1의 *MsgExp permutation*은 [표 2-3]에 나타난 Z_{16} 상에서의 치환이다. Line 3의 #R은 순차적 구조 양자회로에서 사용된 Simple ripple-carry 덧셈을 나타낸다. 순차적 구조 양자회로의 Simple ripple-carry 덧셈기는 이전 워드 배열 쌍의 덧셈에서 사용한 1 carry 큐비트를 다음 워드 배열 쌍 덧셈에서 재사용하는 방식을 사용하기 때문에 16개의 워드 배열 쌍이 순차적으로 연산된다. LSH-256에서는 512 bit의 두 워드 배열 $M_{j-2}^{(i)}$, $M_{j-1}^{(i)}$에 대해 $\omega = 32$ 단위로 16번 덧셈을 차례로 수행하여 $M_j^{(i)}$을 생성하며 LSH-512에서는 1024 bit의

두 워드 배열 $M_{j-2}^{(i)}$, $M_{j-1}^{(i)}$에 대해 $\omega = 64$ 단위로 16번의 덧셈을 차례로 수행하여 $M_j^{(i)}$을 생성한다. 이러한 방식은 carry 큐비트를 모든 덧셈에서 차례로 재사용하므로 하나의 큐비트로 양자회로 전체의 덧셈을 진행할 수 있다. 메시지 확장에서 덧셈은 Depth가 $6n-2$ 인 simple ripple-carry 덧셈기가 16번 순차적으로 진행되므로 $(6n-2)\times 16$의 Depth를 가진다. [그림 3-2]는 순차적 구조의 메시지 확장 덧셈 방식을 보여준다.

[Algorithm 3-1] Sequential LSH-256의 Compression function CF 양자회로

Input : M_k and M_{k-1} pair, T_k, ancilla c $(1 \le k \le 16)$

```
//Message expansion function MsgExp round r ≥ 2
1: MsgExp permutation(M_{r%2})
2: for k=0 to 16 do
3:     M_{k-1} ← #R(M_k, M_{k-1}, c)
4: end for

//Step function Step
5: for k=0 to 16 do
6:     T_k ← CNOT(M_k, T_k)
7: end for
8: for k=0 to 8 do
9:     Mix(T_k, T_{k+7})
10: end for
11: WordPerm(T)
```

[Algorithm 3-1] Line 9의 Mix 함수는 [그림 2-9]의 연산을 수행하며 양자회로 동작은 [Algorithm 3-2]와 같다. 내부에서 사용하는 덧셈기는 $MsgExp$ 함수와 동일한 Simple ripple-carry 덧셈기를 사용한다. Simple ripple-carry 덧셈기는 이전 임시 변수 T 쌍의 덧셈에서 사용한 1 carry 큐비트를 다음 임시 변수 쌍 덧셈에서 재사용하는 방식을 사용하기 때문에 8개의 임시 변수 쌍이 순차적으로 연산된다. Line 11의 $WordPerm$ 함수는 [표 2-5]에 작성된 Z_{16} 상에서의 치환을 동작한다. 치환은 SWAP 게이트로 큐비트 간의 물리적인 위치만 바꾸므로 별로의 양자비용을 사용하지 않는다.

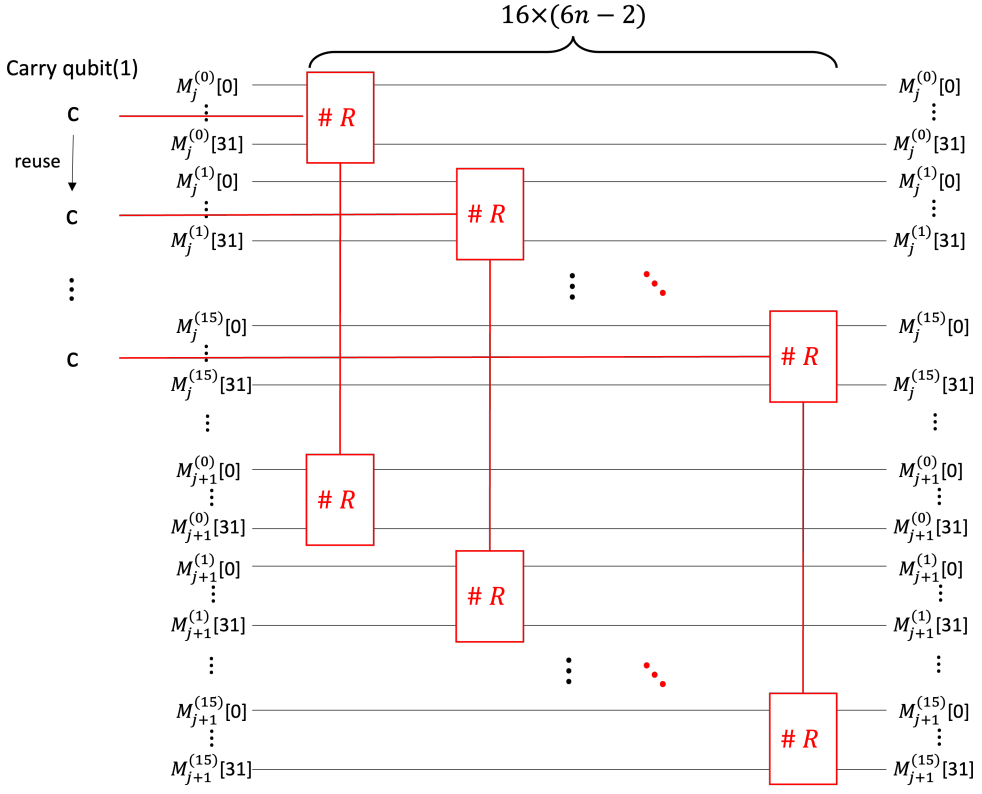
[Algorithm 3-2] Sequential LSH-256의 Mix 함수 양자회로

Input : $T[i], T[i+8]\ (0 \le i \le 7)$

1: $T[i+8] \leftarrow \#\mathrm{R}(T[i],\ T[i+8])$
2: a_rotation($T[i]$)

3: **Applying X gate to** $T[i]$ **according to** $SC[i]$

4: b_rotation($T[i+8]$)
5: $T[i] \leftarrow \#\mathrm{R}(T[i+8],\ T[i])$
6: c_rotation($T[i+8]$)



[그림 3-2] Sequential 구조의 LSH-256 메시지 확장 덧셈 구조

병렬 구조의 Parallel LSH 양자회로에 대한 전체적인 Compression function CF 알고리즘은 [Algorithm 3-3]과 같다. line 1의 $MsgExp\ permutation$은 [표 2-3]에 나타난 Z_{16} 상에서의 치환이다. Line 3의 #Parallel 은 병렬구조 양자회로에서 사용된 ripple-carry 덧셈을 나타낸다. 병렬 구조 양자회로는 simple-ripple-carry 덧셈기를 사용한 순차적 구조 양자회로와 다르게 ripple-carry 덧셈기를 사용하여 설계되었다. 순차적 구조의 Sequential 양자회로 알고리즘인 [Algorithm 3-1]과의 차이점은 16개의 carry 큐비트를 사용하여 16개의 워드 배열 쌍 덧셈이 병렬로 진행된다. ripple-carry 덧셈기는 simple-ripple-carry 덧셈기보다 더 작은 Depth를 가지기 때문에 Depth 최적화를 위한 병렬구조에 더욱 적합하다. 모든 워드 배열 쌍은 독립적인 carry 큐비트를 사용하도록 설계하여 (총 16개 큐비트 사용) 16개의 워드 배열 쌍은 독립적으로 병렬 연산되도록 하였다. LSH-256에서는 512 bit의 두 워드 배열 $M_{j-2}^{(i)}$, $M_{j-1}^{(i)}$에 대해 $\omega=32$ 단위로 16개 덧셈을 병렬로 수행하여 $M_j^{(i)}$을 생성하며 LSH-512에서는 1024 bit의 두 워드 배열 $M_{j-2}^{(i)}$, $M_{j-1}^{(i)}$에 대해 $\omega=64$ 단위로 16개의 덧셈을 병렬로 수행하여 $M_j^{(i)}$을 생성한다. 이러한 방식은 carry 큐비트를 모든 덧셈에서 독립적으로 사용하므로 워드 쌍 만큼의 큐비트로 덧셈을 병렬로 진행할 수 있다. 메시지 확장에서 덧셈은 Depth가 $2n+3$ 인 ripple-carry 덧셈기 16개가 병렬로 진행되므로 $(2n+3)\times 1$의 Depth를 가진다. [그림 3-3]은 병렬 구조 메시지 확장 덧셈 방식을 보여준다. 그림에 나타난 병렬 구조의 덧셈기 #Parallel의 내부 동작은 [Algorithm 3-5] 와 같다.

[Algorithm 3-3] Parallel LSH-256의 Compression function CF 양자회로

Input : M_k and M_{k-1} pair, T_k, ancilla c_k $(1 \le k \le 16)$

```
//Message expansion function MsgExp round r ≥ 2
1: MsgExp permutation(M_{r%2})
2: for k=0 to 16 do
3:     M_{k-1} ← #Parallel(M_k, M_{k-1}, c_k)
4: end for

//Step function Step
5: for k=0 to 16 do
6:     T_k ← CNOT(M_k, T_k)
7: end for
8: for k=0 to 8 do
9:     Mix(T_k, T_{k+7})
10: end for
11: WordPerm(T)
```

[Algorithm 3-3] Line 9의 Mix 함수는 [그림 2-9]의 연산을 수행하며 양자회로 동작은 [Algorithm 3-4]와 같다. 내부에서 사용하는 덧셈기는 $MsgExp$ 함수와 동일한 ripple-carry 덧셈기(#Parallel)를 사용한다. 덧셈에 모든 임시변수 T 쌍이 독립적인 carry 큐비트를 사용하도록 설계하여 (총 8개 큐비트 사용) 8개의 워드 임시 변수 T 쌍은 독립적으로 병렬 연산된다. Line 11의 $WordPerm$ 함수는 [표 2-5]에 작성된 Z_{16} 상에서의 치환을 동작한다. 치환은 SWAP 게이트로 큐비트 간의 물리적인 위치만 바꾸므로 별로의 양자비용을 사용하지 않는다.

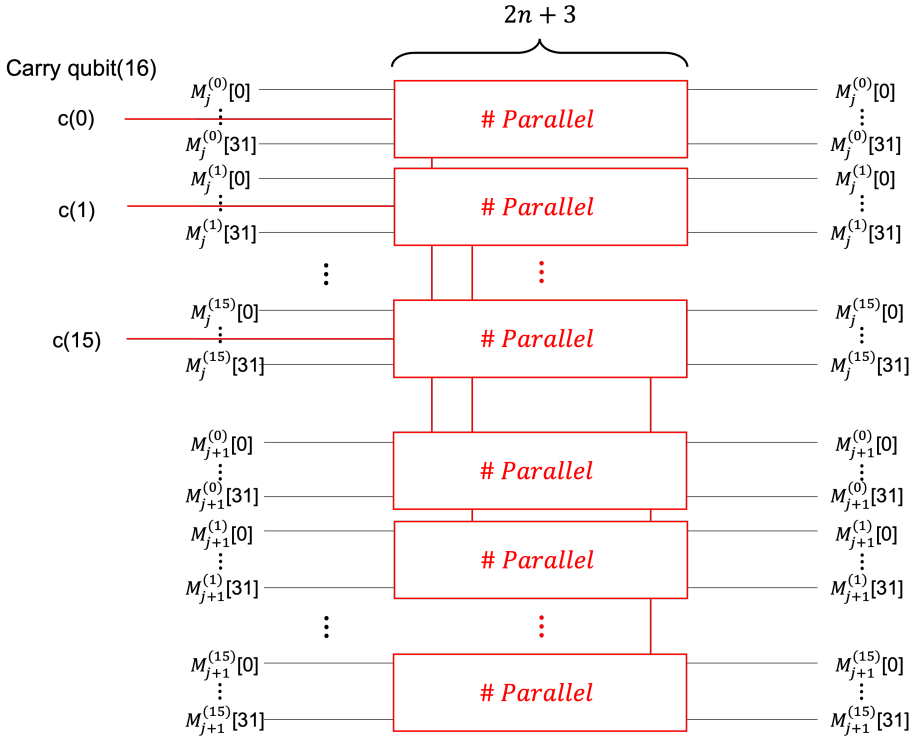
[Algorithm 3-4] Parallel LSH-256 의 Mix 함수 양자회로

Input : $T[i], T[i+8]\ (0 \le i \le 7)$

1: $T[i+8] \leftarrow$ #Parallel$(T[i],\ T[i+8])$
2: a_rotation$(T[i])$

3: **Applying X gate to** $T[i]$ **according to** $SC[i]$

4: b_rotation$(T[i+8])$
5: $T[i] \leftarrow$ #Parallel$(T[i+8],\ T[i])$
6: c_rotation$(T[i+8])$



[그림 3-3] Parallel 구조의 LSH-256 메시지 확장 덧셈 구조

[Algorithm 3-5] Parallel 구조의 LSH-256 덧셈 양자회로 #Parallel

Input : M_k and M_{k-1} pair, ancilla c_k $(1 \le k \le 16)$

1: **for** i=0 to 29 **do**
2: $M_{k-1}[i+1] \leftarrow \mathrm{CNOT}(M_k[i+1],\ M_{k-1}[i+1])$
3: **end for**

4: $c_k \leftarrow \mathrm{CNOT}(M_k[1],\ c_k)$
5: $c_k \leftarrow \mathrm{Toffoli}(M_k[0],\ M_{k-1}[0],\ c_k)$
6: $M_k[1] \leftarrow \mathrm{CNOT}(M_k[2],\ M_k[1])$
7: $M_k[1] \leftarrow \mathrm{Toffoli}(c_k,\ M_{k-1}[1],\ M_k[1])$
8: $M_k[2] \leftarrow \mathrm{CNOT}(M_k[3],\ M_k[2])$

9: **for** i=0 to 26 **do**
10: $M_k[i+2] \leftarrow \mathrm{Toffoli}(M_k[i+1],\ M_{k-1}[i+2],\ M_k[i+2])$
11: $M_k[i+3] \leftarrow \mathrm{CNOT}(M_k[i+4],\ M_k[i+3])$
12: **end for**

13: $M_k[29] \leftarrow \mathrm{Toffoli}(M_k[28],\ M_{k-1}[29],\ M_k[29])$
14: $M_{k-1}[31] \leftarrow \mathrm{CNOT}(M_k[30],\ M_{k-1}[31])$
15: $M_{k-1}[31] \leftarrow \mathrm{CNOT}(M_k[31],\ M_{k-1}[31])$
16: $M_{k-1}[31] \leftarrow \mathrm{Toffoli}(M_k[29],\ M_{k-1}[30],\ M_{k-1}[31])$

17: **for** i=0 to 28 **do**
18: $\mathrm{X}(M_{k-1}[i+1])$
19: **end for**

20: $M_{k-1}[1] \leftarrow \mathrm{CNOT}(c_k,\ M_{k-1}[1])$

21: **for** i=0 to 28 **do**
22: $M_{k-1}[i+2] \leftarrow \mathrm{CNOT}(M_k[i+1],\ M_{k-1}[i+2])$
23: **end for**

24: $M_k[29] \leftarrow \mathrm{Toffoli}(M_k[28],\ M_{k-1}[29],\ M_k[29])$

25: **for** i=0 to 26 **do**
26: $M_k[28-i] \leftarrow \mathrm{Toffoli}(M_k[27-i],\ M_{k-1}[28-i],\ M_k[28-i])$
27: $M_k[29-i] \leftarrow \mathrm{CNOT}(M_k[30-i],\ M_k[29-i])$

28: X($M_{k-1}[29-i]$)
29: **end for**

30: $M_k[1]$←Toffoli(ck, $M_{k-1}[1]$, $M_k[1]$)
31: $M_k[2]$←CNOT($M_k[3]$, $M_k[2]$)
32: X($M_{k-1}[2]$)
33: ck ←Toffoli($M_k[0]$, $M_{k-1}[0]$, c_k)
34: $M_k[1]$←CNOT($M_k[2]$, $M_k[1]$)
35: X($M_{k-1}[1]$)
36: ck ←CNOT($M_k[1]$, c_k)

37: **for** i=0 **to** 30 **do**
38: $M_{k-1}[i]$←CNOT($M_k[i]$, $M_{k-1}[i]$)
39: **end for**

4. 평가

제안된 LSH 양자 회로는 IBM에서 제공하는 양자 시뮬레이터인 ProjectQ를 사용하여 양자회로를 설계하고 자원을 추정하였다. ProjectQ의 양자 컴파일러는 양자 회로의 큐비트 수, Toffoli 게이트, CNOT 게이트, X 게이트, 양자회로 Depth를 측정한다. 양자 회로 최적화의 중요한 요소는 최소한의 양자 자원과 Depth로 동작하도록 설계하는 것이다. 따라서 해당 논문에서는 최적화 방식을 사용하여 최소한의 양자 자원 및 Depth를 사용할 수 있는 두 가지 LSH 양자회로를 제안하였다.

본 논문에서는 물리적인 양자 자원을 최적화 한 방식인 순차적 구조의 Sequential LSH 양자회로를 제안하였으며 순차적 구조에서 효율적인 양자자원 trade-off를 통해 Depth를 크게 줄인 병렬 구조의 Parallel LSH 양자회로를 제안하였다. 순차적 구조의 양자회로는 사용 큐비트를 줄이기 위해 simple-ripple carry 덧셈기를 사용하였으며 병렬 구조의 양자회로는 Depth를 줄이기 위해 ripple carry 덧셈기 사용하였다. ripple-carry 덧셈기는 simple-ripple-carry 덧셈기 보다 더 많은 큐비트를 사용하지만 더 작은 Depth를 가진다. 병렬 구조 양자회로는 ripple-carry 덧셈기를 이용하여 병렬 덧셈 구조로 구현되었으며 LSH의 *MsgExp* 함수와 *Mix* 함수가 모든 워드배열 메시지에 대해 병렬로 동작한다. 결과적으로 병렬 구조의 양자회로는 순차적 구조의 양자회로와 비교했을 때 양자 자원 측면에서 효율적인 trade-off로 Depth를 크게 줄였다. 순차적 구조의 덧셈 방식은 큐비트 수를 줄이기 위해 이전 워드배열 메시지에서 사용된 ancilla 큐비트를 재사용하는 방식으로 동작하기 때문에 모든 메시지 블록들이 순차적으로 연산된다. 하지만 병렬 구조가 가능한 *MsgExp* 함수와 *Mix* 함수에서 이러한 방식은 매우 비효율적인 양자회로 Depth 결과를 가져온다.

평가에서는 제안한 두 방식의 양자회로에 대한 양자자원을 추정하고 효율적인 양자자원 trade-off 결과를 비교한다. LSH 양자회로 구현에 대한 최적화 결과를 설명하기 위해서는 이전의 LSH 양자회로와 비교해야 하지만 이

것이 최초의 LSH 양자회로 구현이므로 이전의 LSH 양자회로와의 비교가 어렵다. 따라서 제안하는 두 양자회로 간의 비교를 진행한다. [표 4-1]은 순차적 구조의 Sequential LSH 양자회로에 대한 양자자원 추정 결과를 보여주며 [표 4-2]는 병렬 구조의 Parallel LSH 양자회로에 대한 양자자원 추정 결과를 보여준다. 순차적 구조의 LSH 양자회로는 큐비트 수 및 양자 게이트 최적화에 초점을 두고 설계했으며 병렬 구조의 LSH 양자회로는 Depth 및 양자 게이트 최적화에 초점을 두고 설계하였다. [표 4-1]의 순차 양자회로는 LSH-256에서 1,537개의 큐비트를 사용하고 약 210K Depth를 가지며 LSH-512에서 3,073개의 큐비트를 사용하고 약 421K Depth를 가진다. [표 4-2]의 병렬 양자회로는 LSH-256에서 1,552개의 큐비트를 사용하고 약 6.8K Depth를 가지며 LSH-512에서 3,088개의 큐비트를 사용하고 약 14.5K Depth를 가진다.

두 양자회로에 대한 큐비트 수, Depth를 비교했을 때 Sequential 구조에 비해 Parallel 구조가 15 큐비트가 증가하였지만 LSH-256에서 Depth가 약 96.73% 감소하였으며 LSH-512에서 Depth가 약 96.56% 로 크게 감소하였다. 두 양자회로에 대한 양자 게이트를 비교하면 Sequential 구조 양자회로가 Parallel 구조 보다 더 적은 X 게이트와 CNOT 게이트를 사용하지만 더 많은 Toffoli 게이트를 사용한다. 그러나 Toffoli 게이트는 X 게이트 및 CNOT 게이트보다 더 비싼 리소스이며 Clifford+T 게이트로 분해했을 때 T 게이트를 사용하는 양자 자원이다. 이러한 양자 자원의 trade-off 결과 회로 Depth가 크게 줄어들었다. 결과적으로 본 논문에서는 적절한 양자 자원 trade-off를 통해 효율적인 병렬구조의 LSH 양자회로를 설계할 수 있다고 주장한다. 양자 컴퓨터는 미래 기술인만큼 변화가 뚜렷하고 불확실한 연구 분야이므로 두 가지 방향으로의 최적화 모두 연구되어야 한다.

[표 4-1] Sequential LSH 양자회로 양자자원 추정 결과

Algorithm	Quantum resources				
	Qubit	X gate	CNOT gate	Toffoli gate	Depth
LSH-256-224	1,537	1,536	145,152	63,448	210,051
LSH-256-256	1,537	3,492	145,152	63,448	210,049
LSH-512-224	3,073	7,663	312,832	139,104	421,851
LSH-512-256	3,073	7,696	312,832	139,104	421,851
LSH-512-384	3,073	7,668	312,832	139,104	421,850
LSH-512-512	3,073	7,680	312,832	139,104	421,852

[표 4-2] Parallel LSH 양자회로 양자자원 추정 결과

Algorithm	Quantum resources				
	Qubit	X gate	CNOT gate	Toffoli gate	Depth
LSH-256-224	1,552	59,392	170,752	62,464	6,879
LSH-256-256	1,552	59,392	170,752	62,464	6,879
LSH-512-224	3,088	134,688	375,760	138,000	14,517
LSH-512-256	3,088	134,688	375,760	138,000	14,517
LSH-512-384	3,088	134,688	375,760	138,000	14,517
LSH-512-512	3,088	134,688	375,760	138,000	14,517

[표 4-3]과 [표 4-4]는 제안하는 양자회로를 사용하여 Grover 알고리즘 공격에 필요한 양자 자원 추정 결과 보여준다. 양자회로는 Grover 알고리즘은 내부에서 Oracle과 Diffusion operator를 반복으로 동작하며 해시 양자회로는 Oracle에 위치한다. Diffusion operator에서 사용하는 양자자원은 고정된 자원이므로 Oracle에 사용되는 양자자원을 추정하여 Grover 알고리즘 공격에 필요한 양자자원을 계산한다. Oracle 내부에는 암호화와 복호화가 진행되므로 [표 4-1]과 [표 4-2]에서 추정된 암호화 양자자원의 두 배를 사용한다. 또한, Grover 알고리즘은 $\left\lfloor \frac{\pi}{4}\sqrt{2^n} \right\rfloor$ 번 반복하므로 Sequential LSH 양자회로의 Grover 알고리즘 공격비용 [표 4-3]은 $2\times[\text{표 } 4-1]\times \left\lfloor \frac{\pi}{4}\sqrt{2^n} \right\rfloor$ 로 계산되며 Parallel LSH 양자회로의 Grover 알고리즘 공격비용 [표 4-4]는 $2\times[\text{표 } 4-2]\times \left\lfloor \frac{\pi}{4}\sqrt{2^n} \right\rfloor$ 로 계산된다.

[표 4-3] Sequential LSH 양자회로에 대한 Grover 공격 비용

Algorithm	Quantum resources				
	Qubit	Toffoli gates	CNOT gates	X gates	Depth
LSH-256-224	1,537	1.94×2^{128}	1.74×2^{129}	1.18×2^{123}	1.26×2^{130}
LSH-256-256	1,537	1.94×2^{144}	1.74×2^{145}	1.34×2^{140}	1.26×2^{146}
LSH-512-224	3,073	1.06×2^{130}	1.87×2^{130}	1.47×2^{125}	1.26×2^{131}
LSH-512-256	3,073	1.06×2^{146}	1.87×2^{146}	1.47×2^{141}	1.26×2^{147}
LSH-512-384	3,073	1.06×2^{210}	1.87×2^{210}	1.47×2^{205}	1.26×2^{211}
LSH-512-512	3,073	1.06×2^{274}	1.87×2^{274}	1.47×2^{269}	1.26×2^{275}

[표 4-4] Parallel LSH 양자회로에 대한 Grover 공격 비용

Algorithm	Quantum resources				
	Qubit	Toffoli gates	CNOT gates	X gates	Depth
LSH-256-224	1,552	1.91×2^{128}	1.3×2^{130}	1.81×2^{128}	1.68×2^{125}
LSH-256-256	1,552	1.91×2^{144}	1.3×2^{146}	1.81×2^{144}	1.68×2^{141}
LSH-512-224	3,088	1.05×2^{130}	1.43×2^{131}	1.02×2^{130}	1.77×2^{126}
LSH-512-256	3,088	1.05×2^{146}	1.43×2^{147}	1.02×2^{146}	1.77×2^{142}
LSH-512-384	3,088	1.05×2^{210}	1.43×2^{211}	1.02×2^{210}	1.77×2^{206}
LSH-512-512	3,088	1.05×2^{274}	1.43×2^{275}	1.02×2^{274}	1.77×2^{270}

양자 게이트와 양자회로 Depth의 종합적인 양자자원 trade-off 결과를 비교하기 위해 Grover 공격에 필요한 기본 양자 게이트(e.g X 게이트, CNOT 게이트, Toffoli 게이트)를 하위 레벨인 T+Clifford 게이트로 분해하여 양자자원 Cost를 계산하였다. [표 4-5]은 Sequential LSH 양자회로의 Grover 공격 자원인 [표 4-3]을 분해 한 결과이며 [표 4-6]은 Parallel LSH 양자회로의 Gover 공격 자원인 [표 4-4]를 분해 한 결과이다. 각 표의 Cost는 분해된 T+Clifford 게이트의 총 합인 Total 게이트와 총 양자회로 Depth를 곱한 결과를 나타낸다. [표 4-5]의 Sequential LSH 양자회로의 Cost와 [표 4-6]의 Parallel LSH 양자회로의 Cost를 비교하면 Sequential LSH 양자회로가 Parallel LSH 양자회로 보다 약 8배 높은 Cost를 가진다.

두 양자회로에 대한 모든 양자자원 추정 결과를 종합하면 Sequential LSH 양자회로가 큐비트 측면에서 최적화 되었으며 Parallel LSH 양자회로가 종합적인 양자 자원(양자 게이트 + 양자회로 Depth) 측면에서 최적화 되었다는 것을 확인 할 수 있다.

[표 4-5] Sequential LSH 양자회로 양자자원 비용 계산

Algorithm	Quantum resources				
	T	Clifford	Total gates	Total Depth	Cost
LSH-256-224	1.33×2^{131}	1.96×2^{131}	1.65×2^{132}	1.26×2^{130}	1.64×2^{263}
LSH-256-256	1.33×2^{147}	1.97×2^{147}	1.65×2^{148}	1.26×2^{146}	1.64×2^{295}
LSH-512-224	1.46×2^{132}	1.07×2^{133}	1.8×2^{133}	1.26×2^{131}	1.13×2^{265}
LSH-512-256	1.46×2^{148}	1.07×2^{149}	1.8×2^{149}	1.26×2^{147}	1.13×2^{297}
LSH-512-384	1.46×2^{212}	1.07×2^{213}	1.8×2^{213}	1.26×2^{211}	1.13×2^{425}
LSH-512-512	1.46×2^{276}	1.07×2^{277}	1.8×2^{277}	1.26×2^{275}	1.13×2^{553}

[표 4-6] Parallel LSH 양자회로 양자자원 비용 계산

Algorithm	Quantum resources				
	T	Clifford	Total gates	Total Depth	Cost
LSH-256-224	1.67×2^{131}	1.06×2^{134}	1.27×2^{134}	1.68×2^{125}	1.07×2^{260}
LSH-256-256	1.67×2^{147}	1.06×2^{150}	1.27×2^{150}	1.68×2^{141}	1.07×2^{292}
LSH-512-224	1.84×2^{132}	1.41×2^{133}	1.17×2^{134}	1.77×2^{126}	1.04×2^{261}
LSH-512-256	1.84×2^{148}	1.41×2^{149}	1.17×2^{150}	1.77×2^{142}	1.04×2^{293}
LSH-512-384	1.84×2^{212}	1.41×2^{213}	1.17×2^{214}	1.77×2^{206}	1.04×2^{421}
LSH-512-512	1.84×2^{276}	1.41×2^{277}	1.17×2^{278}	1.77×2^{270}	1.04×2^{549}

5. 결론

본 논문에서는 LSH에 대한 최초의 양자회로를 제안하였다. LSH 양자회로 구현에서 양자 자원인 큐비트 수, 양자 게이트, 회로 Depth의 최적화에 초점을 맞추었으며 관점에 따른 두 가지 방식의 양자회로를 제안하였다. 큐비트 수 측면에 가장 최적화 된 Sequential LSH 양자회로와 Depth 측면에서 가장 최적화 된 Parallel LSH 양자회로를 제시하고 사용된 양자 자원을 추정하였다. 추정된 양자 자원을 통해 Grover 알고리즘에 필요한 양자 자원을 계산하고 양자 게이트와 양자회로 Depth의 종합적인 양자자원 trade-off 결과를 비교하기 위해 기본 양자게이트(e.g. X 게이트, CNOT 게이트, Toffoli 게이트)를 더 하위 레벨인 T+Clifford 게이트로 분해하여 Cost를 계산하였다. Cost 계산 결과 Sequential LSH 양자회로가 Parallel LSH 양자회로 보다 약 8배 높은 Cost를 가졌다. 따라서 Sequential LSH 양자회로가 큐비트 측면에서 최적화 되었으며 Parallel LSH 양자회로가 양자 게이트, 양자 회로 Depth 측면에서 더 효율적인 구현으로 볼 수 있다. 양자 컴퓨터는 미래 기술 인 만큼 변화가 뚜렷한 분야이므로 추후 양자컴퓨터의 동작 방향이 불분명하다. 따라서 양자 자원 및 양자회로 Depth를 효율적으로 구현하는 두 가지 방향의 연구 모두 중요하다. 해당 논문에서 제안한 LSH 양자회로는 향후 대규모 양자컴퓨터가 개발되면 LSH 해시함수에 대한 post-quantum 강도를 평가에 사용될 수 있을 것이라 예상한다.

참 고 문 헌

1. 국외문헌

Shor, P. W. (1999). Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. SIAM review, 41(2), 303-332.

Grover, L. K. (1996, July). A fast quantum mechanical algorithm for database search. In Proceedings of the twenty-eighth annual ACM symposium on Theory of computing (pp. 212-219).

Grassl, M., Langenberg, B., Roetteler, M., & Steinwandt, R. (2016, February). Applying Grover's algorithm to AES: quantum resource estimates. In Post-Quantum Cryptography (pp. 29-43). Springer, Cham.

Jaques, S., Naehrig, M., Roetteler, M., & Virdia, F. (2020, May). Implementing Grover oracles for quantum key search on AES and LowMC. In Annual International Conference on the Theory and Applications of Cryptographic Techniques (pp. 280-310). Springer, Cham.

Anand, R., Maitra, A., & Mukhopadhyay, S. (2020). Grover on SIMON., Quantum Information Processing, 19(9), 1-17.

Jang, K., Choi, S., Kwon, H., & Seo, H. (2020). Grover on SPECK: Quantum resource estimates. Cryptology ePrint Archive.

Jang, K., Kim, H., Eum, S., & Seo, H. (2020). Grover on GIFT. Cryptology ePrint Archive.

Schlieper, L. (2020). In-place implementation of Quantum-Gimli. arXiv preprint arXiv:2007.06319.

Jang, K., Choi, S., Kwon, H., Kim, H., Park, J., & Seo, H. (2020). Grover on Korean block ciphers. Applied Sciences, 10(18), 6407.

Jang, K., Song, G., Kim, H., Kwon, H., Kim, H., & Seo, H. (2021). Efficient implementation of present and gift on quantum computers. Applied Sciences, 11(11), 4776.

Song, G., Jang, K., Kim, H., Lee, W. K., & Seo, H. (2021). Grover on Caesar and Vigenere ciphers. Cryptology ePrint Archive.

Jang, K., Song, G., Kwon, H., Uhm, S., Kim, H., Lee, W. K., & Seo, H. (2021). Grover on PIPO. Electronics, 10(10), 1194.

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

Baksi, A., Jang, K., Song, G., Seo, H., & Xiang, Z. (2021). Quantum implementation and resource estimates for rectangle and knot. Quantum Information Processing, 20(12), 1-24.

Song, G., Jang, K., Kim, H., Eum, S., Sim, M., Kim, H., ... & Seo, H. (2022). SPEEDY Quantum Circuit for Grover's Algorithm. Applied Sciences, 12(14), 6870.

Huang, Z., & Sun, S. (2022). Synthesizing Quantum Circuits of AES with Lower T-depth and Less Qubits. Cryptology ePrint Archive.

Amy, M., Matteo, O. D., 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). Springer, Cham.

Debnath, S., Linke, N. M., Figgatt, C., Landsman, K. A., Wright, K., & Monroe, C. (2016). Demonstration of a small programmable quantum computer with atomic qubits. Nature, 536(7614), 63-66.

Kielpinski, D., Monroe, C., & Wineland, D. J. (2002). Architecture for a large-scale ion-trap quantum computer. Nature, 417(6890), 709-711.

Farhi, E., Goldstone, J., & Gutmann, S. (2014). A quantum approximate optimization algorithm. arXiv preprint arXiv:1411.4028.

Farhi, E., Goldstone, J., Gutmann, S., & Neven, H. (2017). Quantum algorithms for fixed qubit architectures. arXiv preprint arXiv:1703.06199.

Barends, R., Kelly, J., Megrant, A., Veitia, A., Sank, D., Jeffrey, E., ... & Martinis, J. M. (2014). Superconducting quantum circuits at the surface code threshold for fault tolerance. Nature, 508(7497), 500-503.

Kim, D. C., Hong, D., Lee, J. K., Kim, W. H., & Kwon, D. (2014, December). LSH: A new fast secure hash function family. In International Conference on Information Security and Cryptology (pp. 286-313). Springer, Cham.

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

ABSTRACT

Implementation of quantum circuit and application of Grover algorithm for LSH hash function

Song, Gyeong-Ju

Major in IT Convergence Engineering

Dept. of IT Convergence Engineering

The Graduate School

Hansung University

Grover algorithm in the quantum computer accelerates the pre-image attack on the hash function, thus reducing the hash function of the n-bit security level to the $\sqrt{n}$-bit security level. In order to perform a quantum pre-image attack using the Grover algorithm, the hash function to be attacked must be implemented as a quantum circuit in Grover's internal Oracle. With this research motivation, this paper proposes the first quantum circuit for LSH, a Korean standard hash function. LSH quantum circuits are divided into sequential structured LSH quantum circuit and parallel structured LSH quantum circuit according to the efficiency of quantum resources. Also, the Grover attack resource is calculated and evaluated through the quantum resource estimation results for the two quantum circuits. The quantum gate required for the Grover attack is decomposed into the low-level

T+Clifford gate, and the quantum resource cost is calculated to analyze the comprehensive resource trade-off for quantum gate and the quantum circuit Depth. As a result of quantum resource analysis, it is confirmed that the sequential LSH quantum circuit is efficiently implemented in terms of the number of qubits, and the parallel LSH quantum circuit is efficiently implemented in terms of depth.

【Key words】 Grover algorithm, LSH hash function, Quantum pre-image attack, Quantum circuit, Quantum computer