석사학위논문

해시함수 SHA-1에 대한 양자 회로 구현

2026년

한 성 대 학 교 대 학 원

융 합 보 안 학 과

융 합 보 안 전 공

윤 세 영

석사학위논문
지도교수 서화정

해시함수 SHA-1에 대한 양자 회로 구현

Quantum Implementation of SHA-1

2025년 12월 일

한 성 대 학 교 대 학 원

융 합 보 안 학 과

융 합 보 안 전 공

윤 세 영

석사학위논문
지도교수 서화정

해시함수 SHA-1에 대한 양자 회로 구현

Quantum Implementation of SHA-1

위 논문을 공학 석사학위 논문으로 제출함

2025년 12월 일

한 성 대 학 교 대 학 원

융 합 보 안 학 과

융 합 보 안 전 공

윤 세 영

윤세영의 공학 석사학위 논문을 인준함

2025년 12월 일

심사위원장 박명서 (인)

심 사 위 원 서화정 (인)

심 사 위 원 석병진 (인)

국 문 초 록

해시함수 SHA-1에 대한 양자 회로 구현

한 성 대 학 교 대 학 원
융 합 보 안 학 과
융 합 보 안 전 공
윤 세 영

양자 컴퓨팅 기술이 급속히 발전함에 따라 기존 대칭키 및 공개키 암호 시스템의 안정성은 점점 더 큰 위협에 직면하고 있으며, 이러한 위협의 핵심에는 양자 컴퓨터에서 실행되는 양자 알고리즘이 있다. 양자 알고리즘의 성능은 일반적으로 사용되는 큐비트 수(qubit, width)와 회로의 깊이(depth)라는 두 가지 지표에 의해 판단할 수 있다. 본 논문에서는 양자 컴퓨팅 환경에서 효율적으로 동작하는 SHA-1의 완전한 양자 회로를 구현하고, 큐비트와 depth 자원을 동시에 줄이는 것을 목표로 양자 회로를 균형 있게 최적화하였다. 그 결과 제안한 SHA-1 양자 회로는 985개의 큐비트를 사용하며, 측정된 depth는 9,026이다. 또한 이 회로를 바탕으로 Grover 알고리즘에 따른 공격 비용을 추정함으로써, 양자 컴퓨팅 환경에서 SHA-1에 대한 실제적인 공격 가능성을 평가하고 향후 양자 암호 위협 분석 연구에 활용할 수 있는 기초 자료를 제공한다.

【주요어】 양자 컴퓨팅, 양자 회로 구현, 그루버 알고리즘, SHA-1

목 차

표 목 차

그 림 목 차

알 고 리 즘 목 차

수 식 목 차

제 1 장 서 론

제 1 절 연구 배경

양자 컴퓨터의 개발과 양자 컴퓨팅 기술의 빠른 발전으로 인해 기존 대칭키 암호와 공개키 암호 알고리즘의 안전성이 위협받고 있다. 이러한 위협의 핵심에는 양자 컴퓨터에서 실행되는 각종 양자 알고리즘이 존재한다. 대표적인 양자 알고리즘의 예로는 Shor 알고리즘과 Grover 알고리즘을 들 수 있다. Shor 알고리즘은 정수의 소인수분해와 이산대수 문제를 다항 시간 안에 해결할 수 있는 알고리즘으로, 이들 수학적 문제의 계산 난이도에 보안을 의존하는 RSA, ECC와 같이 널리 사용되는 공개키 암호 시스템을 무력화할 수 있다고 알려져 있다. 한편 Grover 알고리즘은 무차별 대입(brute-force) 탐색 연산을 제곱근 수준으로 가속하여 대칭키 암호를 깨뜨릴 수 있는 것으로 알려져 있다.

이러한 능력은 양자 컴퓨터가 가진 특성으로부터 비롯된다. 양자 컴퓨터는 0과 1의 값을 갖는 고전 비트(bit) 대신 큐비트(qubit)를 사용한다. 큐비트는 중첩(superposition)과 얽힘(entanglement)이라는 양자역학적 원리를 활용하여 상태 $|0\rangle$과 $|1\rangle$을 동시에 중첩된 형태로 표현할 수 있다. 즉, 큐비트 n개로 2^n개의 상태를 동시에 표현하고 처리할 수 있으며 이러한 병렬성 덕분에 양자 컴퓨터는 고전 컴퓨터보다 훨씬 빠른 속도로 연산을 수행할 수 있다. 이에 따라 양자 컴퓨팅 시대에 대비하기 위해, 양자 컴퓨터를 이용해 기존 암호 시스템을 공격하는 방법을 분석하는 연구와 양자 내성 암호(Post-Quantum Cryptography)를 개발하려는 연구 등이 활발하게 진행되고 있다.

대칭키 암호와 해시 함수와 같은 고전 암호 시스템에 대한 공격은 Grover 알고리즘을 기반으로 수행될 수 있다. 정렬되지 않은 n비트 키 공간에서 무차별 대입과 같은 고전적인 탐색 알고리즘을 사용할 경우 평균적으로 $O(2^n)$의 시간 복잡도가 필요하지만, Grover 알고리즘을 사용할 경우 동일

한 크기의 키 공간을 약 $O(\sqrt{2^n})$ 시간 복잡도만으로 탐색할 수 있는 것으로 알려져 있다. 따라서 해시 함수에 대해서도 preimage attack을 수행할 수 있으며 충분히 큰 규모의 양자 컴퓨터가 실현될 경우 기존 해시 함수의 보안성을 심각하게 약화시킬 수 있다. 이러한 공격을 실제로 수행하려면 먼저 목표가 되는 알고리즘을 양자 게이트로 구성된 양자 회로 형태로 구현해야 한다. 이때 양자 회로의 성능은 일반적으로 사용된 큐비트 수(qubit, width)와 실행에 필요한 순차적 연산 단계 수를 나타내는 전체 회로 깊이(depth)에 의해 평가된다. 널리 알려진 해시 함수 SHA-1에 대한 양자 회로를 구현하는 과정에서 본 논문은 큐비트 사용량을 줄이는 동시에 회로의 깊이 또한 감소시키는, 두 자원 지표 간의 균형 잡힌 최적화를 달성하는 것을 목표로 한다.

본 논문의 구성은 다음과 같다. 먼저 제 2장에서는 본 연구를 이해하기 위한 배경 지식을 다룬다. 양자 알고리즘에서 사용되는 기본 양자 게이트와 양자 회로 구현할 때 성능 평가 지표로 활용되는 자원에 대해 설명한 뒤 Grover 알고리즘의 개념을 소개한다. Grover 알고리즘은 n큐비트 탐색 공간에서 약 $\sqrt{2^n}$번의 쿼리만으로도 높은 확률로 목표 해를 찾을 수 있는 양자 알고리즘이며 상태 초기화(state initialization), 오라클(oracle), 확산 연산자(diffusion operator)라는 Grover 알고리즘의 세 가지 주요 단계로 구성되어 있다. 이어서 기존 SHA-1 알고리즘의 전체 구조와 동작 방식을 상세히 기술하고, 다른 암호 알고리즘에 대한 기존의 양자 회로 구현 연구를 검토한다. 제 3장에서는 SHA-1의 완전한 양자 회로 구현이라는 본 논문 핵심 기여를 제시한다. 메시지 패딩, 라운드 워드 확장, 비선형 함수 및 압축 함수 구조 등 SHA-1의 각 구성 요소를 양자 회로로 설계하는 방법을 설명하고 회로의 깊이를 줄이면서도 더 적은 큐비트를 사용하기 위한 구체적인 최적화 기법을 제안한다. 제 4장에서는 회로 깊이 최적화 과정의 핵심이 되었던 두 가지 양자 덧셈기, 즉 회로의 깊이를 최소화하는 Draper 덧셈기와 큐비트 사용량을 줄이는 데 초점을 맞춘 TTK 덧셈기를 기반으로 구현한 SHA-1 양자 회로에 대해 큐비트 수와 회로의 깊이를 정량적으로 분석한다. 마지막으로는 Grover 알고리즘에 따른 공격 비용을

계산하여 양자 컴퓨팅 환경에서의 SHA-1 보안 수준을 논의한다. 제 5장에서는 전체 연구 결과를 요약하고 향후 연구 방향을 제안하며 논문을 마무리한다.

제 2 절 연구 기여

본 논문의 기여는 다음과 같이 요약할 수 있다.

1) SHA-1에 대한 완전한 양자 회로 구현

본 논문에서는 양자 컴퓨팅 환경에서 암호 해독을 위해 Grover 알고리즘의 오라클로 활용될 수 있도록 설계한 SHA-1 해시 함수의 완전한 양자 회로를 제안한다. 제안한 회로는 메시지 패딩과 스케줄링(W_i 확장), 80라운드에 걸친 압축 함수, 그리고 최종 해시 값 계산까지 SHA-1의 전 과정을 모두 포함한다.

2) 자원의 균형적 최적화

본 논문에서 제안하는 SHA-1 양자 회로는 큐비트 수와 회로 깊이 사이의 trade-off를 고려하여 두 자원 지표를 균형 있게 최적화한 구현이다. 양자 회로의 성능은 일반적으로 사용된 큐비트 수와 게이트가 순차적으로 배치된 회로의 깊이에 의해 결정된다. 이러한 자원의 규모는 오류가 쉽게 누적되고 이런 오류에 의해 영향을 받는 양자 컴퓨터의 특성상 양자 회로가 실제로 실행 가능한지를 평가하고, 더 나아가 기존 암호 알고리즘의 보안성이 양자 공격에 의해 언제 붕괴될지를 가늠하는 기준이 된다. 단순히 기존 암호 알고리즘과 매핑할 경우 양자 회로의 깊이를 줄이기 위해 많은 보조 레지스터를 사용하게 되어 큐비트 사용량이 크게 증가하게 된다. 반대로 큐비트를 절약하기 위해 게이트 수를 늘려 깊이가 선형적으로 증가하는 비현실적인 회로가 될 수 있는 문제가 발생한다.

따라서 이러한 한계를 극복하기 위해 다음과 같은 구체적인 최적화 전략을 도입하였다. 첫째, 보조 큐비트(ancilla qubit, 보조 레지스터)를 불필요하

게 늘리지 않도록 in-place 연산과 함께 compute-uncompute 함수를 적극적으로 활용했다. 중간 결과를 동일한 큐비트 내에서 갱신하고 사용이 끝난 보조 큐비트는 다시 |0> 상태로 복원하여 재사용한다. 둘째, 메시지 스케줄링과 내부 상태 업데이트에서 데이터 흐름을 분석하여 동시에 수행 가능한 게이트를 최대한 병렬로 배치함으로써 회로 깊이를 줄였다. 셋째, 산술 연산이 집중되는 부분에 대해서는 회로의 깊이를 크게 줄일 수 있는 Draper 덧셈기를 채택하였다.

제 2 장 배 경

제 1 절 양자 컴퓨팅

본 절에서는 양자 컴퓨팅 환경에서 양자 회로를 설계하고 분석하기 위해 필요한 기본 개념을 정리한다. 먼저 양자 컴퓨터와 양자 컴퓨팅 플랫폼의 발전 동향을 간단히 살펴보며 실제 사용 가능한 큐비트 수의 증가와 양자 오류 정정이 가능한 양자 프로세서의 등장 등으로 양자 기술이 빠르게 고도화되고 있음을 확인하고, 이에 맞추어 실질적으로 활용 가능한 양자 회로 구현의 필요성을 강조한다. 이후 SHA-1 양자 회로의 설계와 자원 분석에 앞서 기본 논리 연산과는 다른 양자 게이트의 동작 원리를 설명한다. 마지막으로 양자 회로에서 사용되는 핵심 자원 지표인 큐비트와 회로 깊이의 개념 및 이들이 양자 컴퓨팅 환경에서 갖는 의미를 살펴본다.

1) 양자 컴퓨터 및 양자 컴퓨팅 플랫폼 발전 동향

양자 상태는 외부 환경과의 아주 작은 상호작용에도 쉽게 붕괴되는 특성을 가진다. 온도 변화나 기계적 진동, 전자기적 잡음 등 여러 요인에 매우 민감하다. 따라서 양자 정보를 일정 시간 이상 유지하기 위해서는 극저온 환경과 특수 장비, 정밀한 제어 회로로 이루어진 안정된 실험 환경 등이 필요하다. 그러나 큐비트 수가 늘어나면 각 큐비트와 게이트 연산에서 발생하는 작은 오류가 점차 누적된다. 그 결과 전체 계산 결과가 왜곡될 가능성이 커진다. 대규모 양자 컴퓨터를 실현하려면 큐비트 수를 늘리는 것뿐 아니라 오류율을 낮추고 양자 오류 정정 기술을 고도화하는 일이 함께 요구된다. 한편 양자 컴퓨터에 대한 학문적 연구와 산업적 관심이 빠르게 증가하면서 실제 양자 프로세서를 원격으로 사용해 보거나 고전 컴퓨터에서 양자 회로를 실험해 볼 수 있는 여러 양자 컴퓨팅 플랫폼과 시뮬레이터가 등장하고 있다.

양자 컴퓨터 개발을 이끄는 주요 기업들의 프로세서 로드맵은 [표 2-1]에

요약되어 있다. Google의 경우 Bristlecone 72큐비트를 시작점으로 삼아 Sycamore를 거쳐 2024년 Willow 105큐비트까지 하드웨어 규모와 성능을 지속적으로 향상시키고 있다. 특히 최근에는 오랜 기간 난제로 여겨졌던 오류 감소 문제를 상당 부분 완화하여 실제 양자 알고리즘을 비교적 안정적인 환경에서 실행할 수 있을 정도의 신뢰도를 확보한 것으로 평가된다. IBM은 사전에 공개한 로드맵에 따라 큐비트 수와 게이트 품질을 단계적으로 확장해 왔으며 현재는 1,121큐비트급 Condor 이후 세대의 QPU까지 계획하고 있어 대형 양자 프로세서 경쟁에서 주도적인 행보를 보이고 있다. 이온 트랩 기반 양자 컴퓨터를 개발하는 IonQ는 Harmony, Aria, Forte와 같은 프로세서를 통해 상대적으로 적은 수이지만 높은 품질의 큐비트를 제공하고 이를 상용 클라우드 서비스와 연계하는 데 강점을 두고 있다. Microsoft는 일반적인 큐비트와는 다르게 Majorana 입자를 기반으로 하는 토폴로지 큐비트 기반의 Majorana 1을 중심으로 장기적인 오류 정정 가능 구조를 모색하고 있으며 Rigetti는 Agave, Aspen, Ankaa 시리즈와 같이 초전도 큐비트 기반 QPU를 연속적으로 선보이며 독자적인 하드웨어 생태계를 구축하고 있다. 이처럼 현 단계의 양자 컴퓨터는 여전히 오류율과 동작 시간 측면에서 한계가 존재하지만, [표 2-1]에서 확인할 수 있듯이 이용 가능한 큐비트 수는 꾸준히 증가하는 추세이며, 각 기업은 오류 정정이 가능한 대규모 양자 컴퓨터 실현을 목표로 경쟁적으로 기술 개발을 진행하고 있다.

기업명	발전 현황		
	출시년도	QPU	큐비트 수
Google	2018	Bristlecone	72
	2019	Sycamore	54
	2024	Willow	105
IBM	2021	Hummingbird	65
	2022	Eagle	127
	2023	Osprey	433
	2025	Condor	1,121
	2025	Loon	-
IonQ	2018	Harmony	9
	2022	Aria	25
	2022	Forte	36
	2025	Forte Enterprise	36
	예정	Tempo	-
Microsoft	2025	Majorana 1	8 (topological)
Rigetti Computing	2017	Agave	8
	2017	Acorn	19
	2020	Aspen-8	32
	2022	Aspen-M	80
	2023, 2024	Ankaa	84

[표 2-1] 주요 기업의 양자 컴퓨터 개발 현황

양자 컴퓨팅 플랫폼	지원 큐비트 수	비고
Google Cirq	약 20큐비트	기본 Python 시뮬레이터
IBM Qiskit	최대 156큐비트	무료 요금제 시뮬레이터
Intel Quantum SDK	최대 32큐비트	단일 노드
Microsoft Azure Quantum	최대 100큐비트	PASQAL 하드웨어 백엔드
ProjectQ	수만 개 수준	고전 컴퓨터 환경

[표 2-2] 주요 양자 컴퓨팅 플랫폼 특징

주요 양자 컴퓨팅 소프트웨어 플랫폼 및 시뮬레이터의 특징은 [표 2-2]에 정리되어 있다. Google Cirq는 약 20큐비트 수준의 비교적 작은 회로를 손쉽게 설계하고 시뮬레이션할 수 있도록 지원한다. IBM Qiskit은 무료 요금제 기준 최대 156큐비트까지 시뮬레이터를 제공하며 실제 IBM QPU에 대한 클라우드 접속 기능도 함께 제공한다. Intel Quantum SDK는 단일 노드 환경에서 최대 32큐비트까지 시뮬레이션을 지원한다. Microsoft Azure Quantum은 여러 업체의 백엔드를 통합하여 최대 100큐비트 규모의 회로를 실행할 수 있는 시뮬레이션 및 하드웨어 접근 환경을 제공한다. ProjectQ는 고전 컴퓨터에서 대규모 양자 회로를 실행할 수 있는 프레임워크로, 'ClassicalSimulator'를 사용하면 수만 개 수준의 큐비트를 포함하는 회로도 이론적으로 시뮬레이션할 수 있다는 점이 특징이다. 본 논문에서 구현한 SHA-1 양자 회로는 이 가운데 ProjectQ를 사용하여 설계하고 시뮬레이션하였다. 특히 ProjectQ의 'Resource counter' 기능을 이용하여 회로에 사용된 큐비트 수와 게이트 수, 회로 깊이와 같은 양자 자원을 정량적으로 측정하였다. 따라서 본 논문에서 제안하는 양자 회로 구조가 실제 양자 컴퓨팅 환경에서 어느 정도 자원을 요

구하는지 보다 구체적으로 평가할 수 있었다.

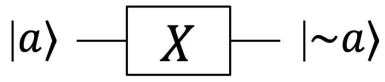
2) 양자 게이트

양자 회로에서는 고전 컴퓨터에서 사용하는 AND, OR, NOT과 같은 논리 연산자 대신 양자 게이트를 사용한다. 양자 게이트는 큐비트의 상태를 바꾸는 연산이다. 한 번 적용한 뒤에 다시 원래 상태로 되돌리는 연산을 정의할 수 있다는 점이 특징이다. 각 게이트는 회로도에서 선 위에 간단한 기호로 표시하며 수식이나 표에서는 입력 상태와 출력 상태의 대응 관계로 설명할 수 있다. 아래에서 다루는 네 가지 게이트(X, CNOT, Toffoli, SWAP)는 이후 장에서 SHA-1 양자 회로를 구성할 때 반복해서 사용되는 기본 구성 요소이다.

X 게이트는 하나의 큐비트에 작용하는 기본 단일 큐비트 게이트이다. 고전 논리에서의 NOT 연산과 비슷하게 동작한다. 입력이 |0>이면 |1>로, 입력이 |1>이면 |0>으로 바꾼다. [표 2-3]의 진리표에서는 입력과 출력이 서로 뒤집히는 것을 확인할 수 있고, [그림 2-1]과 같이 회로도에서는 선 위에 X 기호 하나로 표현할 수 있다. 중첩 상태인 $a|0> + b|1>$에 X 게이트를 적용하면 계수는 그대로 두고 |0>과 |1>의 위치만 맞바뀐다.

Input	Output
|0>	|1>
|1>	|0>

[표 2-3] X 게이트 진리표

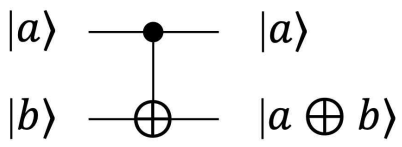


[그림 2-1] X 게이트 회로도

CNOT, Toffoli, SWAP은 두 개 이상의 큐비트에 작용하는 다중 큐비트 게이트이다. CNOT 게이트는 하나의 제어 큐비트와 하나의 타깃 큐비트에 작용한다. 제어가 0이면 아무 변화가 없고 제어가 1일 때만 타깃 큐비트를 X 게이트로 뒤집는다. 고전 논리에서의 XOR 연산과 비슷하게 동작한다. [표 2-4]의 CNOT 게이트 진리표에서는 a가 제어 큐비트이고, b가 타깃 큐비트이다. 회로도에서는 [그림 2-2]와 같이 제어에 ●, 타깃에 ⊕를 사용해 두 선을 연결한다.

Input		Output	
a	b	a'	b'
|0>	|0>	|0>	|0>
|0>	|1>	|0>	|1>
|1>	|0>	|1>	|1>
|1>	|1>	|1>	|0>

[표 2-4] CNOT 게이트 진리표

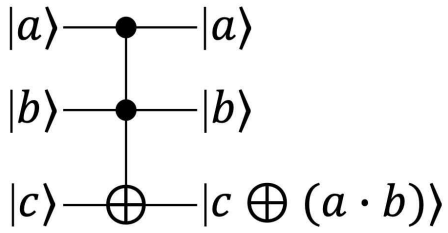


[그림 2-2] CNOT 게이트 회로도

Toffoli 게이트는 CNOT을 확장한 형태로 제어 큐비트가 두 개이고 타깃이 하나이다. 두 제어가 모두 1일 때만 타깃을 뒤집는다. 고전 계산에서는 이 게이트 하나로 AND와 NOT을 구성할 수 있어 보편적인 가역 게이트로 알려져 있다. [표 2-5]에서는 a와 b가 제어 큐비트이며, c가 타깃 큐비트이다. Toffoli 게이트에 대한 회로도는 [그림 2-3]과 같다.

Input			Output		
a	*b*	*c*	*a*’	*b*’	*c*’
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	0	1	0
0	1	1	0	1	1
1	0	0	1	0	0
1	0	1	1	0	1
1	1	0	1	1	1
1	1	1	1	1	0

[표 2-5] Toffoli 게이트 진리표



[그림 2-3] Toffoli 게이트 회로도

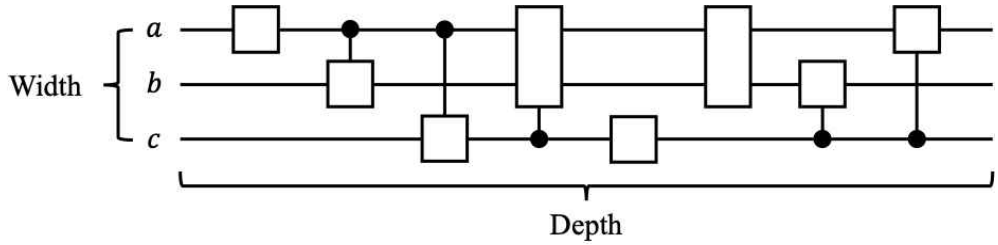
SWAP 게이트는 두 큐비트의 상태를 서로 교환하는 게이트이다. 예를 들어 *a*큐비트에 대한 값은 |0>이고 *b*큐비트에 대한 값이 |1>이라면 SWAP 게이트를 사용했을 때 *a*는 |1>, *b*는 |0> 값을 갖게 된다.

3) 큐비트와 회로 깊이

양자 컴퓨팅 상에서 동작하는 양자 회로는 두 가지 자원 지표로 요약할

수 있다. 하나는 회로에 포함된 큐비트의 개수를 나타내는 너비 width이고, 다른 하나는 게이트가 순차적으로 이어지는 단계의 길이를 나타내는 깊이 depth이다. 양자 회로에서 width와 depth의 의미는 [그림 2-4]와 같다. width는 주어진 회로를 실행하기 위해 동시에 준비해야 하는 논리 큐비트의 수를 의미한다. 알고리즘의 입력과 출력을 담당하는 데이터 큐비트뿐 아니라 중간 계산에 사용되는 보조 큐비트도 모두 포함한다. width가 커질수록 더 많은 물리 큐비트와 제어 장비가 필요하다. 양자 오류 정정을 고려하면 하나의 논리 큐비트를 유지하기 위해 다수의 물리 큐비트가 요구되므로 width는 실제 하드웨어에서 구현 가능성을 가늠할 수 있는 기본 지표가 된다. depth는 게이트가 얼만큼 직렬로 배치되어 있는지를 나타낸다. depth가 클수록 전체 게이트 수가 많다는 것이고 그만큼 실행 시간이 증가한다. 양자 컴퓨터는 시간이 길어질수록 잡음과 외부요인에 인한 오류가 누적되기 쉬우므로 실제 장치에서는 실행 가능한 최대 깊이를 고려한다.

NIST의 경우 [표 2-6]과 같이 보안 강도의 요구사항을 논의할 때 양자 공격의 복잡도에서 허용 가능한 최대 회로 깊이(MAXDEPTH)를 중요한 기준으로 다룬다. 일반적으로 width를 늘리면 일부 연산을 병렬로 처리해 depth를 줄일 수 있고, 반대로 큐비트를 아껴서 width를 줄이면 같은 연산을 더 많은 단계로 나누어 수행해야 하므로 depth가 늘어난다. 따라서 양자 회로를 설계할 때에는 사용 가능한 하드웨어와 오류율을 고려하여 width와 depth 사이의 균형을 맞추는 것이 중요하다. 본 논문에서도 SHA-1 양자 회로의 성능을 평가할 때 이 두 지표를 중심으로 자원 사용량을 분석한다.



[그림 2-4] 양자 회로에서 Width와 Depth

보안 강도	요구사항
Level 1	AES-128 이상의 키 검색 공격 수준
	2^{170}/MAXDEPTH 양자 게이트 또는 2^{143} 고전 게이트
Level 2	SHA256/SHA3-256 이상의 충돌 공격 수준
	2^{146} 고전 게이트
Level 3	AES-192 이상의 키 검색 공격 수준
	2^{233}/MAXDEPTH 양자 게이트 또는 2^{207} 고전 게이트
Level 4	SHA384/SHA3-384 이상의 충돌 공격 수준
	2^{210} 고전 게이트
Level 5	AES-256 이상의 키 검색 공격 수준
	2^{298}/MAXDEPTH 양자 게이트 또는 2^{272} 고전 게이트

[표 2-6] NIST PQC 보안강도

제 2 절 Grover's Algorithm

Grover 알고리즘은 정렬이되지 않은 데이터의 집합에서 특정 값을 찾는 양자 검색 알고리즘이다. Lov Grover가 1996년에 제안했으며 고전적인 무차별 대입 탐색보다 훨씬 적은 연산으로 동일한 작업을 수행할 수 있다. n큐비트 탐색 공간에서 고전 알고리즘은 평균적으로 약 2^n번의 시도가 필요하지만, Grover 알고리즘은 $2^{n/2}$회 정도의 반복으로 목표 값을 찾아 낼 수

있다. 따라서 데이터 규모가 클수록 연산 효율성이 향상된다는 장점이 있다. Grover 알고리즘은 상태 초기화(state initialization), 오라클(oracle), 확산 연산자(diffusion operator)라는 세 단계로 이루어진다.

첫째, 상태 초기화 단계이다. n개의 입력 큐비트에 Hadamard 게이트를 각각 적용하여 모든 기본 상태가 동일한 확률로 포함된 균일 중첩 상태를 만든다. [수식 2-1]에서 $|\psi\rangle$는 2^n개의 모든 상태를 똑같은 비율로 포함하는 시작 상태를 나타낸다.

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

[수식 2-1] 상태 초기화

둘째는 오라클 단계이다. 오라클은 찾고자 하는 함수를 양자 회로로 구현한 부분을 말한다. 예를 들어 해시 함수에 대한 preimage 탐색을 생각하면 오라클은 입력 x에 대해 다음과 같은 함수를 계산할 수 있다.

$$f(x) = 1,\ \text{if}\ \ Hash(x) = target\ \ output$$
$$f(x) = 0,\ \ otherwise$$

[수식 2-2] 오라클 함수 f(x)의 정의

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

[수식 2-3] 확산 연산자

양자 회로에서는 이 함수를 U_f라는 연산으로 나타내며, 중첩 상태 $|\psi\rangle$와 보조 상태 $|-\rangle$에 작용했을 때 [수식 2-2], [수식 2-3]과 같이 f(x)=1인 해 상태의 부호만 뒤집힌다. 실제 자원 추정에서는 해시 출력과 목표 값의 비교 회로는 전체 비용에 비해 작다고 보고 종종 생략한다.

마지막 단계는 확산 연산자 단계이다. 확산 연산자는 오라클에서 표시된 해

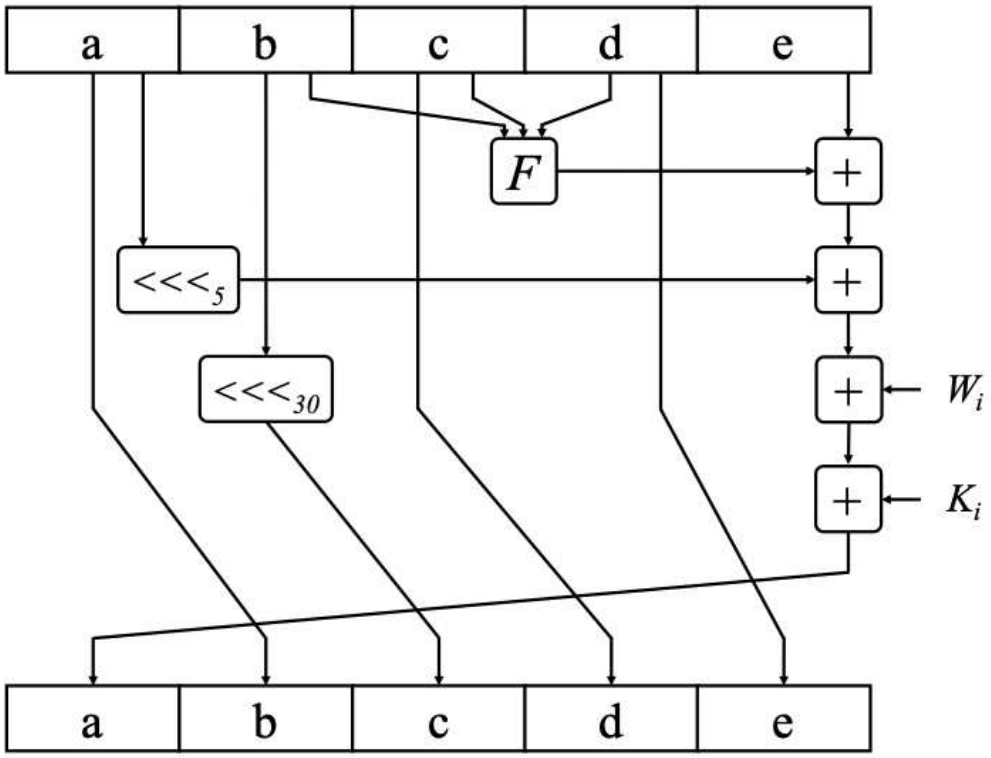
상태의 진폭을 키워 주는 역할을 한다. 평균 진폭을 기준으로 모든 상태를 한 번 뒤집는 연산이라고 볼 수 있으며, 이 과정을 반복하면 해에 해당하는 상태의 진폭은 점점 커지고 나머지 상태의 진폭은 점점 작아진다. 그 결과 마지막에 측정을 하면 정답 상태가 나올 가능성이 크게 증가한다. 확산 연산자는 회로 구조가 단순하고 깊이도 얕은 편이기 때문에, Grover 알고리즘의 전체 자원을 계산할 때는 보통 오라클 쪽 비용이 훨씬 더 큰 것으로 본다.

제 3 절 SHA-1 해시함수

SHA-1(Secure Hash Algorithm-1) 알고리즘은 미국 국가안보국(NSA)이 설계한 암호학적 해시 함수이다. 최초 규격은 1993년 미국 국립표준기술연구소(NIST)가 FIPS PUB-180 문서를 통해 공개했으며 이 초기 버전은 현재 SHA-0이라고 부른다. 공개 이후 NSA는 암호 강도를 떨어뜨릴 수 있는 설계상의 결함을 보완하기 위해 초기 버전을 곧바로 철회하였다. 이후 수정 작업을 거쳐 1995년에 개정판이 FIPS PUB 180-1이라는 이름으로 다시 발표되었고 이것이 일반적으로 알려진 SHA-1이다. SHA-1의 주요 목적은 디지털 서명 표준(Digital Signature Standard, DSS)에 포함되어 전자 서명에 사용되는 것이었으며, 그 결과 다양한 정부 및 상용 응용 분야에서 데이터 무결성과 인증을 보장하는 핵심 구성 요소로 자리 잡았다. 수년 동안 SHA-1은 TLS와 SSL, PGP, SSH, IPsec 등 수많은 보안 프로토콜에서 기본 해시 함수로 널리 사용되었다.

SHA-1 알고리즘은 가변 길이의 입력 메시지를 받아 길이가 고정된 160비트 메시지 다이제스트를 출력한다. 이 값은 일반적으로 40자리 16진수 형태로 표현된다. 처리 과정은 전처리 단계에서 시작되며 입력 메시지에 패딩을 추가하여 전체 길이가 512비트의 배수가 되도록 맞춘다. 패딩이 완료된 메시지는 512비트씩 나누어 블록 단위로 순차적으로 처리된다. 이때 각 512비트 블록은 다시 32비트 워드 16개로 분할되고 [수식 2-4]의 메시지 스케줄링 과정을 통해 80개의 워드로 확장된다. 이렇게 확장된 80개의 워드는 이후 80라운드에 걸친 압축 함수 연산에서 순서대로 사용된다. 알고리즘의 핵심은 32

비트 워드 a, b, c, d, e 다섯 개로 구성된 내부 상태를 중심으로 동작하는 압축 함수이다. 이 다섯 개의 워드는 사전에 정의된 상수 값으로 초기화되어 있다. 라운드가 진행됨에 따라 사용되는 비선형 라운드 함수 F의 논리 연산 형태가 구간별로 다르고 라운드마다 서로 다른 상수 값과 비트 단위 순환 회전 연산이 함께 적용된다. 이러한 연산이 반복되면서 내부 상태가 단계적으로 갱신된다. 모든 메시지 블록에 대한 처리가 끝난 뒤에는 최종 내부 상태를 이루는 다섯 개의 32비트 워드를 순서대로 이어 붙여 최종 160비트 해시 값을 생성한다. 이와 같은 전체 동작 절차는 최신 표준인 FIPS 180-4에 상세히 정의되어 있으며, SHA-1 알고리즘의 반복 구조에서 중심적인 역할을 하는 압축 함수는 [그림 2-5]에 도식화되어 있다.



[그림 2-5] SHA-1의 라운드 함수

$$W_i = W_{i-3} \oplus W_{i-8} \oplus W_{i-14} \oplus W_{i-16} \; ROTL \; 1$$

[수식 2-4] SHA-1의 메시지 스케줄링 과정

SHA-1은 오랫동안 광범위하게 사용되어 왔지만 2000년대 초반부터 그 안전성에 대한 본격적인 의문이 제기되기 시작했다. 2005년 Xiaoyun Wang이 이끄는 연구팀은 SHA-1에서 충돌을 찾아내는 데 필요한 연산 복잡도가 무차별 대입 공격의 기준인 약 2^{80}번의 연산보다 훨씬 낮은 수준이라는 이론적 공격 방법을 제시하였다. 이 결과는 SHA-1의 보안성을 평가하는 데 중요한 전환점이 되었으며 그 이후 SHA-1은 충분한 계산 자원을 갖춘 공격자에 대해서는 더 이상 안전한 해시 함수로 보기 어렵다는 인식이 확산되었다. 이러한 흐름 속에서 2011년 NIST는 새로운 디지털 서명을 생성할 때 SHA-1 사용을 지양하고 NIST가 규정한 프로토콜별 지침에서 허용하는 경우에만 제한적으로 사용할 것을 권고하였다. 나아가 NIST는 2030년 12월 31일까지 암호학적 용도에서 SHA-1을 공식적으로 폐지할 계획을 발표하였다. 특히 강한 충돌 저항성이 요구되는 응용 분야에서는 SHA-1을 계속 사용하는 대신 SHA-2나 SHA-3 계열 해시 함수와 같이 더 높은 보안성을 제공하는 대체 알고리즘으로의 전환이 시급한 과제로 부상하였다.

SHA-1에 대한 이러한 이론적 위협은 2017년 2월 CWI Amsterdam과 Google 연구진이 최초의 공개 충돌 공격인 SHAttered를 발표하면서 현실의 문제로 구체화되었다. 연구진은 서로 다른 두 개의 PDF 파일에 대해 완전히 동일한 SHA-1 해시 값을 만들어 내는 데 성공했으며 무차별 대입 공격보다 약 10만 배 빠른 방식으로 달성된 것으로 보고되었다. 이 결과는 이론적인 가능성을 넘어 실제 환경에서도 SHA-1이 충분히 깨질 수 있는 수준의 취약성을 갖고 있음을 명확히 보여주었다. 이후 진행된 연구들은 SHA-1의 약점을 더욱 분명하게 드러냈다. 2020년에는 chosen prefix collision 공격을 통해 TLS와 SSH와 같은 보안 채널 프로토콜의 전자서명 구조와 핸드셰이크 과정이 실제로 위협받을 수 있다는 사실이 확인되었다. 이를 통해 고도화된 공격자가 SHA-1을 악용해 특정 대상을 정밀하게 노리는 공격을 수행하는 것이 이론이 아닌 현실적인 위험이라는 점이 입증되었다.

현재 SHA-1은 특히 전자서명과 같이 높은 수준의 보안이 요구되는 일반적인 사용 환경에서는 암호학적으로 더 이상 안전하지 않은 알고리즘으로 평가된다. 그럼에도 한동안은 FIPS 198-1에서 정의한 대로 Keyed-Hash Message Authentication Code(HMAC) 용도로 널리 사용되어 왔다. 이렇게 실제 서비스와 프로토콜의 핵심 구성 요소로 오랫동안 활용되었다는 점에서 여전히 중요한 암호학적 의미를 가진다. SHA-1은 양자 공격 관점에서도 유용한 연구 대상이다. SHA-1을 양자 회로로 구현한다면 양자 공격을 시도했을 때와 고전 공격을 시도했을 때 필요한 자원에 대한 차이를 정량적으로 비교할 수 있는 지표를 생성할 수 있다. 또한, SHA-1 양자 공격에 대한 자원 추정은 향후 양자 내성 암호 설계와 양자 위협 평가 연구에 의미 있는 기준점을 제공한다.

제 4 절 기존 양자 회로 구현 연구

본 절에서는 양자 회로로 구현된 키 유도 함수(Key Derivation Function, KDF) Scrypt와 Argon2에 대한 연구를 살펴본다.

Scrypt는 대규모 메모리 접근을 이용해 공격자의 병렬 연산 효율을 떨어뜨리도록 설계되었다. Song et al.의 Scrypt 양자 회로 구현 연구에서는 SMix와 PBKDF2의 결합으로 복잡도가 높은 Scrypt를 최적화하였으며, 특히 보조 큐비트의 재활용과 일부 연산의 병렬화를 통해 회로 자원을 효율적으로 구성하였다. 구체적으로 SMix 단계에서 역연산을 활용해 중간 저장용 큐비트를 최소화하고 SHA-256의 라운드별 보조 큐비트를 복원하여 재사용함으로써 8,128개의 큐비트를 절감하였다. 그 결과 18,140개의 큐비트만으로 구현된 Scrypt의 양자 회로는 Grover 알고리즘 기반 공격에 필요한 자원 요구량을 유의미하게 낮출 수 있음을 입증하였다.

이와 더불어 Song et al.의 Argon2에 대한 양자 회로 구현 연구에서는 내부 Blake2b 압축 함수를 위한 양자 덧셈기를 도입하고, 사용 목적에 따라 큐비트 수 최소화와 회로 깊이 최적화라는 두 가지 방향의 설계 전략을 제시하였다. 연구 결과 큐비트 최적화 방식은 1,089개의 큐비트로 G

함수를 구현하였으며 깊이 최적화 방식은 13,830개의 큐비트를 사용하여 전체 회로 깊이를 69%까지 단축하는 등 자원 간의 trade off를 확인하였다.

[표 2-7]은 앞서 설명한 Scrypt와 Argon2에 대한 양자 자원 추정치를 정리하여 나타낸 표이다. 이러한 최적화 설계 기법은 향후 해당 알고리즘들의 양자 저항성을 정량적으로 분석하고 평가할 수 있는 중요한 기반이 된다.

	Scrypt	Argon2	Argon2
Qubit	18,140	13,830 (Depth Opt, ripple)	1,090 (Qubit Opt, ripple)
Full Depth	173,408	1.01×2^{21} (Depth Opt, ripple)	1.65×2^{22} (Qubit Opt, ripple)

[표 2-7] Scrypt, Argon2 양자 회로에 대한 양자 자원 추정

제 3 장 SHA-1 양자 회로 구현

본 장에서는 SHA-1에 대해 큐비트 사용량과 회로의 깊이를 균형 있게 최적화한 양자 회로 구현을 단계별로 제안한다.

제 1 절 메시지 패딩 및 라운드 워드 계산

1) 고전적 전처리

SHA-1 알고리즘은 먼저 가변 길이의 입력 메시지를 표준화된 형식으로 변환하는 메시지 전처리 단계에서 시작된다. 이 단계에서 패딩을 수행하여 메시지 길이가 512로 나누었을 때 나머지가 448이 되도록 조정한다. 전처리가 끝나면 메시지는 이후 연산에 사용되는 하나 이상의 연속된 512비트 블록으로 구성된 형태가 된다.

2) 동적 큐비트 할당 및 큐비트 재사용

고전적 전처리 이후에는 각 512비트 메시지 블록이 양자 회로에 로드된다. 이를 위해 512개의 큐비트로 이루어진 레지스터를 할당하고 이를 32큐비트씩 16개의 블록으로 나누어 초기 워드 W_0부터 W_{15}를 저장한다. 그다음 80번의 압축 라운드에 필요한 나머지 64개의 워드(W_{16}부터 W_{79}까지)는 모두 양자 회로 안에서 계산된다. 가장 단순하게 구현한 방식은 이 추가 워드들을 위해 새로운 큐비트를 계속 할당하는 것이다. 그러나 이런 방식은 총 2048큐비트(32큐비트씩 64워드)를 더 쓰게 되어 매우 큰 자원 오버헤드를 초래한다. 따라서 본 구현에서는 큐비트 재사용(qubit reuse) 방식을 사용한다. 회로의 깊이를 줄여서 최적화하는 것도 중요하지만, 큐비트 사용 효율을 높여 자원을 최소한으로 하는 것 또한 중요하기 때문이다.

처음 할당한 512큐비트 레지스터(W_0부터 W_{15}까지, 32큐비트씩 16워드로 구성)를 그대로 재사용하여, 확장된 워드들을 순차적으로 계산하고 저장한다. 이 방법을 통해 초기 512큐비트 이외에 추가 큐비트를 전혀 할당하지 않고도 전체 80개의 메시지 워드를 모두 확장할 수 있으며, 그 결과 전체 자원 요구량을 크게 줄일 수 있었다. 큐비트 재사용의 실마리는 SHA-1 메시지 확장 알고리즘에 내재된 인덱스의 규칙성에 있다. 우리는 각 워드의 인덱스가 갖는 이러한 규칙을 활용하여, 이전에 사용되었던 워드(인덱스가 [i-16]인 워드)를 새로 계산된 값으로 갱신하는 방식으로 구현하였다. [수식 3-1]과 같이 W_{16}의 값을 W_0에 덮어쓰고, W_{32}의 값을 W_{16}에 저장하는 식으로 워드를 순환시키며 재사용하도록 설계하였다.

$$W_{16} = W_{16-3} \oplus W_{16-8} \oplus W_{16-14} \oplus W_{16-16}$$

$$\sim$$

$$W_{32} = W_{32-3} \oplus W_{32-8} \oplus W_{32-14} \oplus W_{32-16}$$

[수식 3-1] 메시지 스케줄링 과정 규칙

제 2 절 Rounds (0-79)

1) SHA-1 Round Structure

SHA-1 알고리즘의 핵심은 압축 함수이다. 이 압축 함수는 80개의 워드로 이루어진 메시지 스케줄 W_t를 80라운드에 걸쳐 반복적으로 처리한다. 이 과정에서 a, b, c, d, e로 표시되는 다섯 개의 32비트 작업 변수가 계속 갱신되며 이들이 합쳐져 해시의 내부 상태를 이룬다. 80개 라운드는 20라운드씩 네 개 구간으로 나뉜다. 각 구간은 현재 라운드 t에 따라 달라지는 두 가지 요소로 특징지어진다. 하나는 비선형 논리 함수 F이고, 다른 하나는 고유한 덧셈용 라운드 상수 K이다. 함수 F는 작업 변수 b,

c, d 세 개를 입력으로 받아 그 구간에 정해진 논리 연산에 따라 32비트 출력을 만든다. 함수 F의 정확한 논리식은 범위에 따라 다르다. 또한 각 20라운드 구간에는 서로 다른 상수 K가 하나씩 할당된다. 첫 번째 구간은 $K = 0x5A827999$로 시작하고, 마지막 구간은 $K = 0xCA62C1D6$ 값을 사용한다. 이처럼 함수 F의 출력과 상수 K 그리고 현재 라운드의 메시지 워드 W_t가 결합되어 이후에 수행되는 주요 상태 갱신 연산에 필요한 모든 입력 값이 된다. 이와 같은 고전적인 비선형 논리 연산을 양자 회로로 옮겨 구현하는 것이 본 연구에서 매우 중요한 부분이다. 이러한 설계를 바탕으로 본 구현에서는 F를 구성하는 각 내부 함수에 대해 자원 효율적인 양자 회로를 제안한다. 기본 양자 게이트만을 사용해 이러한 회로를 어떻게 구성하는지는 [알고리즘 3-1]에 자세히 나타나 있다. 각 구간의 구별을 위해 함수 F의 이름을 임의로 F_1, F_2, F_3 로 나타냈다.

Algorithm 3-1 : Quantum SHA-1 Internal F Functions

```
1: procedure F1(result, x, y, z, temp)
2: for i ← 0 to 31 do
3: CNOT | (y[i], temp[i])
4: CNOT | (z[i], temp[i])
5: Toffoli | (x[i], temp[i], result[i])
6: CNOT | (y[i], temp[i])
7: CNOT | (z[i], temp[i])
8: CNOT | (z[i], result[i])
9: end for
10: end procedure

11: procedure F2(result, x, y, z )
12: for i ← 0 to 31 do
13: CNOT | (x[i], result[i])
14: CNOT | (y[i], result[i])
15: CNOT | (z[i], result[i])
16: end for
17: end procedure

18: procedure F3(result, x, y, z, temp)
19: for i ← 0 to 31 do
20: CNOT | (y[i], temp[i])
21: CNOT | (z[i], temp[i])
22: Toffoli | (x[i], temp[i], result[i])
23: CNOT | (y[i], temp[i])
24: CNOT | (z[i], temp[i])
25: Toffoli | (y[i], z[i], result[i])
26: CNOT | (temp[i], result[i])
27: Toffoli | (y[i], z[i], temp[i])
28: end for
29: end procedure
```

[알고리즘 3-1] SHA-1 F 함수

2) 라운드 내 상태 업데이트

라운드별 함수 F와 상수 K가 결정되면, 핵심 상태 업데이트 과정이 수행된다. 먼저, 32비트 임시 워드 T를 계산하는데 이는 다섯 개의 값을 더해 얻는다. 구체적으로 5비트 왼쪽 순환 이동된 작업 변수 a(≪5), 논리 함수 F의 출력값, 작업 변수 e, 라운드 상수 K, 그리고 현재 메시지 스케줄 워드 W_t를 모두 더한 값이다. 이때 모든 덧셈 연산은 2^{32}를 모듈러 연산으로 수행된다. 이 계산이 끝난 뒤에는 작업 변수들이 순차적으로 갱신된다. 변수 e는 d의 값을, d는 c의 값을, c는 30비트 왼쪽 순환 이동된 b(≪30)의 값을 각각 갖게 된다. 이어서 b는 기존의 a 값을 갖게 되고 마지막으로 a는 새로 계산된 임시 워드 T의 값으로 갱신된다. 이 일련의 연산 과정을 통해 각 라운드마다 내부 상태가 충분히 뒤섞이게 되며, 이는 알고리즘의 보안 특성을 보장하는 데 핵심적인 역할을 한다. 이러한 상태 업데이트를 양자 회로 상에서 자원 효율적으로 구현하는 방법은 다음 절에서 자세히 설명한다.

3) 자원 효율적인 양자 회로 구현

앞서 설명한 상태 갱신 과정을 [알고리즘 3-2]와 같이 양자 회로 형태로 구현한다. 이 구현의 목표는 두 가지 큐비트 최적화 전략을 통해 사용되는 큐비트 수를 최대한 줄이는 것이다. 첫째, 비선형 함수 F의 출력(변수 result)을 저장하는 임시 큐비트에 대해 compute-uncompute 방식을 사용한다. 매 라운드마다 새로운 큐비트를 할당하는 대신 함수 값을 한 번 계산해 사용한 뒤에 곧바로 역연산을 수행해 결과를 지운다. 이렇게 하면 보조 큐비트가 다시 |0> 상태로 되돌아가 다음 라운드에서 깨끗한 상태로 재사용될 수 있다. 둘째, 이러한 최적화 아이디어를 라운드 상수 K의 관리에도 적용한다. 서로 다른 네 개의 상수를 위해 각각 레지스터를 두지 않고 하나의 32큐비트 레지스터에 해당 라운드에 필요한 상수 값을 동적으로 적재한다. 라운드의 계산이 끝나면 이 레지스터에도 역연산을 적용

하여 초기 상태로 되돌린다. 이렇게 즉시 할당하는 방식을 사용할 경우 불필요한 큐비트 중복 할당을 막아 주고 전체 큐비트 사용량을 줄이는 데 큰 역할을 한다.

완전한 양자 회로를 구성하기 위해 필요한 두 가지 보조 함수는 [알고리즘 3-3]에 제시되어 있다. 첫 번째 함수는 고전 비트열을 각 큐비트에 할당하는 역할을 한다. 입력 문자열과 K와 같은 상수 값을 이용해 양자 레지스터를 올바른 초기 상태로 설정하는 데 필수적인 기능이다. 이때 입력 비트 값이 1이면 X 게이트를 적용하여 큐비트 값을 |0>에서 |1>로 뒤집는다. 두 번째로 중요한 서브루틴은 양자 좌측 회전 함수로, 순환 좌측 시프트 연산을 구현한다. 이 연산의 양자 버전은 레지스터 안에서 큐비트들의 위치를 서로 교환하는 SWAP 게이트들을 효율적으로 연결한 회로로 구성된다. 이러한 기본 함수들 또한 주 회로의 성능에도 직접적인 영향을 미치므로 회로 깊이와 큐비트 수를 낭비하지 않도록 구현을 가능한 한 단순하게 유지하는 것이 중요하다.

Algorithm 3-2 : Quantum SHA-1 Compression Function

1: **procedure** SHA1(($H_0 \cdots, H_4$), M_{chunk})

2: ▷ 1. Initialization
3: a, b, c, d, e ← AllocateQubit(32) ▷ Working variables
4: T, f, K ← AllocateQubit(32) ▷ Temporary registers
5: (a, b, c, d, e) ← (H_0, H_1, H_2, H_3, H_4)
6: W[0..15] ← LoadClassicalChunk(M_{chunk})

7: ▷ 2. Main loop: 80 rounds
8: **for** t = 0 → 79 **do** ▷ Round Logic
9: **if** $0 \le t \le 19$ **then**
10: f ← F_1(b, c, d) ▷ Choose function

```
11:         K ← ClassicToQuantum(0x5A827999)
12:     else if 20 ≤ t ≤ 39 then
13:         f ← F2(b, c, d)                    ▷ Parity function
14:         K ← ClassicToQuantum(0x6ED9EBA1)
15:     else if 40 ≤ t ≤ 59 then
16:         f ← F3(b, c, d)                    ▷ Majority function
17:         K ← ClassicToQuantum(0x8F1BBCDC)
18:     else
19:         f ← F2(b, c, d)
20:         K ← ClassicToQuantum(0xCA62C1D6)
21:     end if
                ▷ Compute round word Wt in-place for t ≥ 16
22:     if t ≥ 16 then
23:       W[t(mod 16)] ← ROTL( W[(t−16) (mod
16)]⊕ W[(t−14) (mod 16)]⊕ W[(t−8) (mod 16)]⊕ W[(t−3)
(mod 16)], 1)
24:     end if
  ▷ Compute temporary value T using sequential quantum additions
25:     T ← ROTL(a, 5)
26:     T ← QuantumAdder( T, f, carry)
27:     T ← QuantumAdder( T, e, carry)
28:     T ← QuantumAdder( T, K, carry)
29:     T ← QuantumAdder( T, W[t (mod 16)], carry)
30:           ▷ Update state variables (quantum register swaps)

31:     e ← d
32:     d ← c
33:     c ← ROTL(b, 30)
34:     b ← a
35:     a ← T
  ▷ Uncompute this round's temporary values (f, Kt) to free qubits
36:   end for

37:   ▷ 3. Compute final hash value
38: (H0',..., H4') ← (H0 + a, H1 + b, H2 + c, H3 + d, H4 + e)
39: return (H0',..., H4')
40: end procedure
```

[알고리즘 3-2] SHA-1 압축 함수

Algorithm 3-3 : Quantum Operators

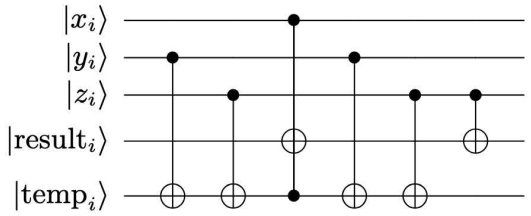
```
1: procedure ClassicToQuantum(const, q)
2:     for i ← 0 to length(q)−1 do
3:         if (const & 1 == 1) then
4:           X | q[i]
5:         end if
6:         const = const >> 1
7:     end for
8: end procedure

9: procedure LeftCircularShift(op1, amount)
10:     n ← 32
11:     visited ← [False] × n
12:     cycles ← [ ]
13:     for i ← 0 to n−1 do
14:         if not visited[i] then
15:           cycle ← [ ]
16:           j ← i
17:           while not visited[j] do
18:             append j to cycle
19:             visited[j] ← True
20:             j ← (j−amount) mod n
21:           end while
22:           if length(cycle) > 1 then
23:              append cycle to cycles
24:           end if
25:         end if
26:     end for
27:     for all cycle ∈ cycles do
28:         for i ← 0 to length(cycle) − 2 do
29:            SWAP(op1[cycle[i]], op1[cycle[i + 1]])
30:         end for
31:         SWAP(op1[cycle[end]], op1[cycle[0]])
32:     end for
33: end procedure
```

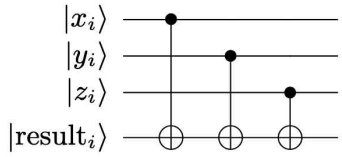
[알고리즘 3-3] SHA-1 양자 연산 보조 함수

4) 내부 함수의 게이트 수준 시각화

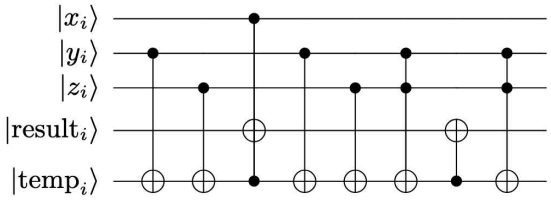
[알고리즘 3-1]은 SHA-1에서 사용되는 세 가지 내부 함수 F의 양자 구현 절차를 단계별로 설명한다. 이를 보완하기 위해 [그림 3-1], [그림 3-2], [그림 3-3]에서는 각 절차에 대응하는 게이트 수준 양자 회로를 자세히 나타낸다. 이 회로도들은 의사코드의 각 줄을 실제 양자 게이트 연산에 직접 대응시키며 이를 통해 양자 정보가 어떻게 흐르는지와 각 함수가 어느 정도의 자원을 사용하는지를 한눈에 볼 수 있다.



[그림 3-1] 함수 F_1에 대한 양자 회로도



[그림 3-2] 함수 F_2에 대한 양자 회로도



[그림 3-3] 함수 F_3에 대한 양자 회로도

[그림 3-1]은 F_1의 양자 회로를 나타낸다. 의사코드에서 볼 수 있듯이 먼저 입력 y[i]와 z[i]의 논리 XOR를 CNOT 게이트로 계산하고 그 결과를 보조 큐비트 temp[i]에 저장한다. 다음으로 x[i]와 temp[i]를 제어 큐비트로 사용하는 Toffoli 게이트를 적용하여 그 결과를 result[i]에 저장한다. 그 이후 이어지는 게이트들은 compute-uncompute에 따라 보조 큐비트들을 다시 초기 상태로 되돌리는 역할을 한다. 마지막 단계에서는 result[i]에 z[i]를 이용해 추가 연산을 가해 최종 결과를 완성한다.

이에 반해 [그림 3-2]에 제시된 F_2 회로는 보조 큐비트를 필요로 하지 않기 때문에 훨씬 더 효율적이다. 이 함수는 세 개의 CNOT 게이트를 순차적으로 적용함으로써 세 입력(x[i], y[i], z[i])의 XOR를 구현한다. 각 입력 큐비트(x[i], y[i], z[i])는 타깃 큐비트 result[i]에 작용하는 제어 큐비트로 사용된다.

마지막으로 [그림 3-3]은 세 함수 중 가장 복잡한 F_3의 구현을 보여준다. F_3은 F_1과 마찬가지로 보조 큐비트 temp를 필요로 하며 compute-uncompute을 사용한다. 이를 간단히 설명하면 다음과 같다. 먼저 y와 z를 XOR한 뒤, 그 결과를 x와 AND한다. 이는 첫 번째 CNOT 게이트를 두 번 사용하고 그 뒤에 Toffoli 게이트를 적용하는 것과 동치이며, 그 결과 $(x \wedge y) \oplus (x \wedge z)$ 값이 result에 저장된다. 이어서 temp를 비우기 위한 연산을 수행한 뒤, y와 z의 AND 결과를 Toffoli 게이트를 사용해 temp에 저장한다. 마지막으로, temp에 저장된 값을 CNOT 게이트를 통해 result에 반영한다. 최종 결과가 result에 저장된 후에는 Toffoli 게이트를 다시 사용하여 temp를 0으로 초기화한다. 이 세 도식은 SHA-1 구현의 핵심 연산 블록에 대한 완전하고 투명한 청사진을 제공하며, 각 연산 단위가 가지는 복잡도와 자원 요구량을 명확하게 보여준다.

제 3 절 최종 해시 값 계산

SHA-1 알고리즘의 마지막 단계는 각 512비트 메시지 블록에 대해 80라운드의 메인 루프가 모두 수행된 이후에 시작된다. 이 단계는 내부 상태를 갱신하고 궁극적으로 최종 메시지 다이제스트를 생성하는 역할을 한다. 메인 루프의 계산 결과로 얻어진 다섯 개의 작업 변수(a, b, c, d, e)는 다섯 개의 32비트 중간 해시 값(H_0, H_1, H_2, H_3, H_4)을 갱신하는 데 사용된다. 구체적으로, 새로운 H_0는 이전 H_0와 a를 더한 값이고, 새로운 H_1는 이전 H_1와 b를 더한 값이며 나머지도 마찬가지 방식으로 다섯 변수 각각을 더해 갱신한다. 이렇게 새로 계산된 해시 값들은 다음 메시지 청크를 처리할 때 초기 해시 값으로 사용된다. 이를 통해 전체 메시지 내용이 순차적으로 최종 출력에 반영되도록 한다. 마지막 메시지 청크까지 모두 처리되고 나면, 최종적으로 얻어진 다섯 개의 32비트 해시 값 H_0, H_1, H_2, H_3, H_4를 순서대로 이어 붙여 하나의 160비트 메시지 다이제스트를 구성한다. 양자 회로 구현 관점에서 보면, 이 최종 160비트 값은 다섯 개의 해시 레지스터를 이루는 160개의 큐비트에 양자 상태로 인코딩된다. 양자 영역에서 이 결과를 추출하기 위해서는 이 160개 큐비트 각각에 대해 표준 계산 기저에서의 측정을 수행한다. 측정이 끝나면 양자 상태는 하나의 고전적인 160비트 비트열로 붕괴하며, 이것이 SHA-1의 최종 출력이 된다. 이 값은 일반적으로 빅엔디언(big-endian) 형식의 40자리 16진수 수로 표현된다.

제 4 장 성능 평가

제 1 절 Draper

Draper 덧셈기(Quantum Carry Lookahead Adder, QCLA)는 로그 스케일의 회로 깊이 성능으로 잘 알려진 대표적인 양자 산술 회로이다. 이 덧셈기는 고전적인 캐리 룩어헤드(carry lookahead) 기법에서 영감을 얻었으며, 선형 깊이를 갖는 기존의 리플 캐리(ripple carry) 덧셈기와 뚜렷한 대조를 이룬다. Draper의 연구에 따르면 이 덧셈기는 $O(n)$개의 보조 큐비트를 사용하여 두 개의 n비트 수를 $O(\log n)$의 깊이로 더할 수 있다. 이러한 로그 깊이 특성은 Shor 알고리즘처럼 산술 연산에 크게 의존하는 양자 알고리즘의 실행 시간을 크게 줄여 준다. 본 연구에서 SHA-1 양자 회로를 구현할 때 전체 회로 깊이를 최소화하는 것을 최우선 과제로 삼았으며 회로 깊이는 근시일 양자 장치에서 핵심적인 병목 요소이다. Draper 가산기 모델을 채택함으로써 총 회로 깊이 9026, 전체 구현에 985개의 큐비트를 사용하는 회로를 설계할 수 있었다.

제 2 절 TTK

양자 덧셈에 대한 또 다른 접근법은 Takahashi, Tani, Kunihiro(TTK)가 제안한 방식으로, 필요한 큐비트 수를 최소화하는 데 초점을 맞추고 있다. TTK 덧셈기의 기본 설계는 두 개의 n비트 값을 덧셈하면서도 보조 큐비트를 전혀 사용하지 않는다는 점에서 주목할 만하며 이는 큐비트 사용 측면에서 매우 큰 장점이다. 그러나 이러한 큐비트 효율성은 회로 깊이 증가라는 대가를 수반하며, 기본 TTK 가산기는 깊이가 $O(n)$인 선형 회로 구

조를 가진다. 구현된 양자 덧셈기들 중 회로 깊이를 줄이기 위해 제안된 기법의 덧셈기들은 대개 추가적인 큐비트 사용으로 trade off를 요구하거나 특수한 게이트의 사용을 전제로 한다. 이러한 trade off가 본 논문에서 구현한 SHA-1 양자 회로에서 어떤 의미를 가지는지 평가하기 위해 SHA-1 양자 회로를 TTK 덧셈기 모델을 사용해 구현해 보았다. 그 결과 전체 큐비트 요구량은 929개로 줄어드는 효과를 얻었으나, 회로 깊이는 43,232까지 크게 증가하는 단점이 발생하였다.

제 3 절 자원 분석

1) 덧셈기 비교 분석

SHA-1 양자 회로에 사용할 가산기를 선택하기 위해 Draper 덧셈기와 TTK 덧셈기를 기반으로 한 구현의 자원 요구량을 비교하였다. 큐비트 수와 회로 깊이 간의 trade off는 [표 4-1]에 정리되어 있다. Draper 덧셈기를 사용한 구현에서는 985개의 큐비트와 9,026의 회로 깊이를 갖는 구성이 도출된 반면, TTK 덧셈기를 사용한 구현에서는 929개의 큐비트를 사용하지만 회로 깊이는 43,232에 이르렀다. TTK 덧셈기는 56개의 큐비트를 절약해 주지만 그 대가로 전체 회로 깊이가 거의 5배 가까이 증가하는 문제가 있었다. 이 trade off가 발생하는 근본적인 이유를 파악하기 위해 우리는 게이트 수준에서 자원 소모를 상세히 추정하였다.

	Draper 덧셈기	TTK 덧셈기
Qubit	985	929
Toffoli 게이트	90,940	24,315
CNOT 게이트	67,879	78,279
X 게이트	20,339	189
Depth	9,026	43,232

[표 4-1] SHA-1 회로 구현에 대한 게이트 수준 자원 추정
ProjectQ의 리소스 카운터 사용 (* 입력 크기 ≤ 512비트)

2) Fault Tolerant 관점 자원 지표

Fault Tolerant 양자컴퓨팅 환경에서 계산 비용을 보다 총체적으로 평가하기 위해서는 큐비트 수와 회로의 깊이를 동시에 반영하는 지표를 사용하는 것이 유용하다. 그중 하나가 Depth-Width (DW) 비용으로 계산에 필요한 전체 시공간 자원을 나타낸다. [표 4-2]에서 볼 수 있듯이 Draper 덧셈기를 기반으로 한 구현이 더 우수한 효율을 보인다. Draper 기반 회로의 DW 비용은 8.89×10^6으로, TTK의 4.02×10^7에 비해 4배 이상 낮다. 이러한 차이는 Draper 가산기 아키텍처가 갖는 실질적인 이점을 잘 보여준다. NISQ(Noisy Intermediate Scale Quantum) 컴퓨터 및 그 이후의 시대에는 전체 큐비트 수 자체보다는 회로 깊이가 더 중요한 제한 요소가 되는 경우가 많다. Draper 덧셈기는 TTK 덧셈기에 비해 회로가 훨씬 얕기 때문에 DW 비용 기준으로 보았을 때 전반적인 계산 비용이 크게 줄어드는 동시에 성능 이득도 상당하다. 이러한 이점은 소폭의 추가 큐비트 사용량을 충분히 상쇄한다고 판단되었고 이에 따라 최종 최적화된 SHA-1 회로의 기본 산술 구성 요소로 Draper 덧셈기를 채택하였다.

	Draper 덧셈기	TTK 덧셈기
Full Depth	9,026	43,232
DW−Cost	8.89×10^6	4.02×10^7

[표 4−2] Fault Tolerant 환경에서의 자원 추정

DW−Cost = (큐비트 수 × 회로 깊이) (* 입력 크기 ≤ 512비트)

3) Grover 공격 비용 추정

SHA−1에 대한 양자 preimage 공격에서는 앞에서 설계한 SHA−1 양자 회로를 Grover 알고리즘의 오라클로 사용한다. 입력 공간의 크기를 n비트라고 하면, Grover 알고리즘은 대략 $(\pi/4)\sqrt{2^n}$회의 오라클 호출로 목표 해를 찾을 수 있다. 본 논문에서는 SHA−1의 입력 메시지 길이가 512비트 이내인 경우를 가정하였으므로 필요한 반복 횟수는 약 $(\pi/4)\sqrt{2^{256}}$회 수준이 된다. 한 번의 Grover 반복에서 요구되는 게이트 수는 Draper 덧셈기를 사용한 SHA−1 회로의 자원 추정값 [표 4−1]에 따르며 오라클에 비해 비용이 상대적으로 작은 확산 연산자의 자원 소모는 무시하였다. 이 반복 횟수를 [표 4−1]의 단일 오라클 비용에 곱하여 계산한 전체 자원 소모는 [표 4−3]에 정리하였다. 입력 길이가 최대 512비트인 메시지에 대해 Grover 알고리즘을 적용할 경우 SHA−1 preimage 공격에는 Toffoli 게이트가 약 1.08×2^{273}, CNOT 게이트가 약 1.62×2^{272}, X 게이트가 약 1.94×2^{270} 필요하다. 회로 깊이는 약 1.73×2^{269}로 추정된다. 이 값들은 Grover 공격 시 소모되는 비용으로 예상해볼 수 있다.

Toffoli	CNOT	X	Depth
1.08×2^{273}	1.62×2^{272}	1.94×2^{270}	1.73×2^{269}

[표 4-3] Grover 공격 비용 추정

제 5 장 결론

양자컴퓨팅 기술의 발전은 SHA-1과 같은 고전적 해시 함수를 포함한 여러 암호 알고리즘의 보안성에 큰 위협이 되고 있다. SHA-1이 Grover 알고리즘과 같은 양자 공격에 이론적으로 취약하다는 사실은 잘 알려져 있지만, 실제로 어느 정도 위험한지 판단하려면 이를 수행하는 데 필요한 구체적인 자원량을 자세히 분석할 필요가 있다. 따라서 SHA-1에 대해 포괄적이면서도 자원 사용을 줄인 양자 회로를 설계함으로써 이러한 공백을 메우는 것을 목표로 하였다. 특히 현재와 근미래의 양자 하드웨어에서 가장 큰 제약 요인인 회로 깊이를 낮추고 큐비트 사용량을 줄이는 데 초점을 맞추었다.

본 논문에서는 SHA-1의 모든 핵심 구성 요소에 대해 게이트 수준 구현을 포함하는 회로 깊이 및 큐비트 최적화 양자 회로 설계를 제시한다. 동적으로 큐비트를 할당하고 compute-uncompute를 적극적으로 사용하여 큐비트를 재활용하는 것과 같은 최적화 기법을 통해 불필요한 큐비트 낭비를 크게 줄였다. 또한 두 가지 양자 덧셈기 Draper 덧셈기와 TTK 덧셈기를 비교한 결과 TTK 덧셈기는 일부 큐비트를 절약하지만, Draper 덧셈기는 큐비트를 조금 더 쓰는 대신 회로 깊이를 거의 다섯 배 정도 줄여 주기 때문에 NISQ 장치에서 노이즈에 의한 양자 정보 손실을 줄이기 위한 실제 구현 측면에서는 더 적합한 선택임을 확인하였다.

SHA-1에 대한 알고리즘 분석 및 완전한 양자 회로 구현은 향후 양자 암호해독 연구를 위한 투명하고 재현 가능한 기반이 될 것이다. 향후 연구에서는 SHA-1 양자 회로에 대해 더 최적화된 회로 깊이 및 큐비트 간 절충점을 찾고, Grover 알고리즘을 적용했을 때 필요한 자원을 보다 정밀하게 추정할 예정이다. 아울러 다른 암호 알고리즘에 대한 양자 회로 구현으로 대상을 확대할 계획이다. 본 연구에서 제안한 회로 구현 방법은 다른 해시 함수를 포함

한 다양한 암호 함수의 양자 회로 최적화에도 적용 가능하며, 현대 암호가 직면한 양자 위협을 분석하고 이해하는 데 의미 있는 기여를 할 것으로 기대한다.

참 고 문 헌

1. 국내문헌

송경주, 윤세영, & 서화정. (2025). 양자컴퓨터 발전동향.『정보보호학회지』, 35(3), 47-54.

김현지, 김덕영, 윤세영, & 서화정. (2024). 양자컴퓨터 플랫폼 동향.『Review of KIISC』, 21-27.

윤세영, 송경주, & 서화정. (2025). 양자컴퓨팅에 강인한 암호화 알고리즘의 양자회로 구현 분석.『Journal of the Korea Institute of Information & Communication Engineering』, 29(6).

2. 국외문헌

Yoon, S., Song, G., Jang, K., Cha, S., & Seo, H. (2025). Quantum Implementation of SHA-1. Cryptology ePrint Archive.

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

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). Cham: Springer International Publishing.

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

Liu, Q., Preneel, B., Zhao, Z., & Wang, M. (2023, December). Improved quantum circuits for AES: Reducing the depth and the number of qubits. In International conference on the theory and application of cryptology and information security (pp. 67-98). Singapore: Springer Nature Singapore.

Standard, S. H. (1993). FIPS Pub 180. National Institute of Standards and Technology.

Standard, S. H. (1995). FIPS Pub 180-1. National Institute of Standards and Technology, 17(180), 15.

Standard, S. H. (2015). FIPS Pub 180-4. National Institute of Standards and Technology.

Gallagher, P. (2013). Digital signature standard (DSS). Federal Information Processing Standards Publications, volume FIPS, 186, 18.

Wang, X., Yin, Y. L., & Yu, H. (2005, August). Finding collisions in the full SHA-1. In Annual international cryptology conference (pp. 17-36). Berlin, Heidelberg: Springer Berlin Heidelberg.

Barker, E., & Roginsky, A. (2018). Transitioning the use of cryptographic algorithms and key lengths (No. NIST Special Publication (SP) 800-131A Rev. 2 (Draft)). National Institute of Standards and Technology.

Stevens, M., Bursztein, E., Karpman, P., Albertini, A., & Markov, Y. (2017, July). The first collision for full SHA-1. In Annual international cryptology conference (pp. 570-596). Cham: Springer International Publishing.

Leurent, G., & Peyrin, T. (2020). {SHA-1} is a shambles: First {Chosen-Prefix} collision on {SHA-1} and application to the {PGP} web of trust. In 29th USENIX Security Symposium (USENIX Security 20) (pp. 1839-1856).

Turner, J. M. (2008). The keyed-hash message authentication code

(hmac). Federal Information Processing Standards Publication, 198(1), 1-13.

Sönmez Turan, M., & Brandão, L. T. (2024). Keyed-Hash Message Authentication Code (HMAC): Specification of HMAC and Recommendations for Message Authentication (No. NIST Special Publication (SP) 800-224 (Draft)). National Institute of Standards and Technology.

Song, G., & Seo, H. (2024). Grover on scrypt. Electronics, 13(16), 3167.

Song, G., Eum, S., Kwon, H., Sim, M., Lee, M., & Seo, H. (2023). Optimized Quantum Circuit for Quantum Security Strength Analysis of Argon2. Electronics, 12(21), 4485.

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

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

Steiger, D. S., Häner, T., & Troyer, M. (2018). ProjectQ: an open source software framework for quantum computing. Quantum, 2, 49.

ABSTRACT

Quantum Implementation of SHA-1

Yoon, Se-Young

Major in Convergence Security

Dept. of Convergence Security

The Graduate School

Hansung University

As quantum computing technology rapidly advances, threats to existing symmetric-key and public-key cryptosystems are becoming increasingly real. In this study, we implement a SHA-1 quantum circuit that operates efficiently in a quantum computing environment. We optimize the quantum circuit, focusing on minimizing total circuit depth, a key performance indicator of quantum algorithms. The SHA-1 quantum circuit implementation used 985 qubits, resulting in a measured circuit depth of 9,026. Furthermore, by integrating this optimized circuit with the Grover algorithm, we establish the foundation for an efficient quantum attack on the SHA-1 algorithm. This research is significant not only because it presents a resource-efficient SHA-1 quantum implementation but also because it enables accelerated attacks in a quantum computing environment.

【Keywords】 Quantum Computing, Quantum Circuit Implementation, Grover's Algorithm, SHA-1