Quantum Implementation of SHA-1

Seyoung Yoon[0009−0008−3254−9256], Gyeongju Song[0000−0002−4337−1843], Kyungbae Jang[0000−0001−5963−7127], Sangmin Cha[0009−0003−3614−6247], and Hwajeong Seo[0000−0003−0069−9061]

Hansung University, South Korea
{sebbang99,thdrudwn98,starj1023,xhxhdwkdpsyentific,hwajeong84}@gmail.com

Abstract. 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 Computer · Grover's Algorithm · Quantum Collision Search · SHA-1 · Post-Quantum Security.

1 Introduction

Advances in quantum computing technology have revealed that the security of conventional symmetric-key cryptography and public-key cryptography algorithms is seriously threatened by quantum algorithms. Quantum algorithms are a major source of these threats. In particular, Shor's algorithm can compromise widely used public-key cryptosystems such as RSA and ECC by solving fundamental mathematical problems in polynomial time. Similarly, Grover's algorithm is known to be able to break symmetric-key cryptography by accelerating brute-force search operations by a square root. This ability stems from the unique properties of quantum computers. Quantum computers use qubits instead of classical bits (0 and 1). Leveraging the principles of superposition and entanglement, qubits can simultaneously represent states $|0\rangle$ and $|1\rangle$. This fundamental property allows quantum computers to perform parallel computations quickly. Therefore, in preparation for the era of quantum computing, active research is underway to attack existing cryptographic systems using quantum computers and develop new quantum-resistant algorithms. Attacks on classical cryptosystems, such as symmetric key cryptography and hash functions, can be performed

using Grover's algorithm, which is known to break n-bit hash functions with a complexity of $O(2^{n/2})$. To execute such an attack, the target algorithm must first be implemented as a quantum circuit composed of elementary quantum gates. The performance of a quantum circuit is primarily evaluated by the total circuit depth, which represents the number of qubits used and the number of sequential steps required for execution. When implementing a quantum circuit for the well-known hash function SHA-1, we focused on minimizing the circuit depth and took care not to waste qubits, contributing to the field of quantum circuit implementation.

This paper is organized as follows. Section 2 provides background information for this research. We first explain the Grover algorithm. Grover's algorithm is a quantum algorithm that can find a target solution with a high probability using approximately $\sqrt{2^n}$ queries in an n-qubit search space. Next, we describe the three main steps of the algorithm: state initialization, oracle, and diffusion operator. Finally, we describe the architecture and operation of the original SHA-1 algorithm. Section 3 presents our main contribution to a complete quantum circuit implementation of SHA-1, demonstrating how to reduce the circuit depth while using fewer qubits. Section 4 presents the results of a performance evaluation of two quantum adders that were at the core of the optimization process: a depth-optimized Draper adder and a qubit-optimized TTK adder. Finally, Section 5 concludes the paper by summarizing the research results and suggesting future directions.

1.1 Contribution

Our contributions are summarized as follows:

Complete Quantum Implementation of the SHA-1 Algorithm. We present a full quantum circuit implementation of the SHA-1 hash function, designed to serve as a verifiable oracle for quantum cryptanalysis. This includes message padding and scheduling (W_t expansion), the 80-round compression function, and final hash computation. A key aspect of our design is the reversible logic construction of the non-linear round function (F_t), which changes across rounds, as well as the reuse of qubits for the message scheduling (W_t expansion).

Resource Optimization with Minimal Circuit Depth. Our circuit achieves a low quantum circuit depth while using a minimal number of qubits. The depth of a quantum circuit is the time it takes to execute all the gates in the designed quantum circuit. Typically, minimizing the depth of a quantum circuit wastes significant qubit resources. Therefore, we addressed this issue by using the following methods: in-place computation to preserve ancilla qubits, gate parallelization based on data dependencies, and a quantum adder optimized for circuit depth. Consequently, this study demonstrates a more realistic quantum threat to SHA-1 and sets a new standard for future research in quantum cryptography.

2 Background

2.1 Grover's Algorithm

For an n-qubit search space, Grover's search algorithm can find a target solution with high probability using approximately $\sqrt{2^n}$ queries. The algorithm consists of three main steps: *state initialization*, *oracle*, and *diffusion operator*.

1. *State initialization*: Apply Hadamard gates to each of the n input qubits to prepare a uniform superposition state $|\psi\rangle$, where all 2^n basis states are equally probable:

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

2. *Oracle*: The oracle encodes the function to be searched (such as a hash function) as a quantum circuit. It evaluates the function on all inputs in superposition and flips the sign of the amplitude corresponding to a solution state (i.e., a pre-image x such that $\text{Hash}(x) = \text{target}$):

$$f(x) = \begin{cases} 1, & \text{if Hash}(x) = \text{target output} \\ 0, & \text{otherwise} \end{cases} \tag{2}$$

$$U_f(|\psi\rangle|-\rangle) = \frac{1}{2^{n/2}} \sum_{x=0}^{2^n-1} (-1)^{f(x)}|x\rangle|-\rangle \tag{3}$$

   The comparison between the computed and target hash outputs is typically omitted from quantum resource estimates [1,2,3,4].
3. *Diffusion operator*: This operator amplifies the amplitude of the marked solution states identified by the oracle, thereby increasing the probability of measuring the correct result. Due to its relatively low complexity compared to the oracle, the diffusion operator is often neglected in resource estimations for Grover's algorithm [1,2,3,4].

2.2 SHA-1

The Secure Hash Algorithm 1, commonly known as SHA-1, is a cryptographic hash function designed by the United States National Security Agency (NSA) as part of the U.S. Government's Capstone project. The original specification was first published by the National Institute of Standards and Technology (NIST) in 1993 as a Federal Information Processing Standard (FIPS) PUB 180 [5]. This initial version is now referred to as SHA-0. Shortly after its publication, the NSA withdrew this version to address a flaw that could reduce its cryptographic security. In 1995, a revised version was released as FIPS PUB 180-1, which became widely known as SHA-1 [6]. The primary purpose for its development was to be incorporated into the Digital Signature Standard (DSS) [7], making

it a crucial component for ensuring data integrity and authenticity in various government and commercial applications. For years, SHA-1 was a foundational element in numerous security protocols, including TLS, SSL, PGP, SSH, and IPsec.

The SHA-1 algorithm works by taking a variable-length input message and producing a fixed-size 160-bit hash value, a message digest. This output is typically expressed as a 40-digit hexadecimal number. The computation begins with a preprocessing step that pads the input message so that its overall length is a multiple of 512 bits. The padded message is then processed sequentially in 512-bit chunks. The core of the algorithm is a compression function that utilizes the internal state of five 32-bit words initialized to a specific constant value. For each 512-bit chunk, the algorithm performs 80 rounds of computation. A key feature of this process is the use of a varying non-linear round function, F_t, whose logical operation changes throughout the rounds, along with round-specific constants and bitwise rotations. These operations iteratively update the internal state. After all message chunks have been processed, the final 160-bit hash value is produced by concatenating the five final internal state words. The complete specification of this process is detailed in the most recent standard, FIPS 180-4 [8]. The compression function of the SHA-1 algorithm, which forms the core of its iterative operation, is illustrated in Figure 1.

Despite its widespread adoption, the security of SHA-1 began to face scrutiny in the early 2000s. In 2005, a research team led by Xiaoyun Wang presented a theoretical attack that could find collisions in SHA-1 with a complexity significantly lower than the brute-force benchmark of 2^{80} operations [9]. This discovery marked a turning point, after which SHA-1 was no longer considered secure against well-funded adversaries. Consequently, in 2011, NIST recommended against using SHA-1 for new digital signature generation and only use it when permitted by NIST protocol-specific guidelines [10]. Furthermore, NIST announced plans to deprecate SHA-1 for cryptographic applications by December 31, 2030. The need to transition to more secure alternatives, such as hash functions from the SHA-2 or SHA-3 family, has become even more pressing, especially for applications that require strong collision resistance.

The theoretical threat to SHA-1 became concrete and widely known in February 2017, when researchers at CWI Amsterdam and Google announced the first public collision attack, SHAttered ([11]). The researchers successfully generated two different PDF files that generated exactly the same SHA-1 hash, achieving a computational speed approximately 100,000 times faster than a brute-force attack. Subsequent research revealed vulnerabilities in SHA-1. In 2020, it was confirmed that chosen-prefix collisions compromise the signature schemes and handshake security of secure channel protocols (TLS, SSH), making chosen-prefix attacks against SHA-1 practical and enabling targeted attacks by sophisticated attackers. [12]. Although SHA-1 is currently considered cryptographically weak for mainstream use cases, particularly digital signatures, it has been widely used for some time in Keyed Hash Message Authentication Codes (HMACs), as speci-

fied in FIPS 198-1 [13]. Although the revised version, SP 800-224 [14], deprecated SHA-1 for HMAC.

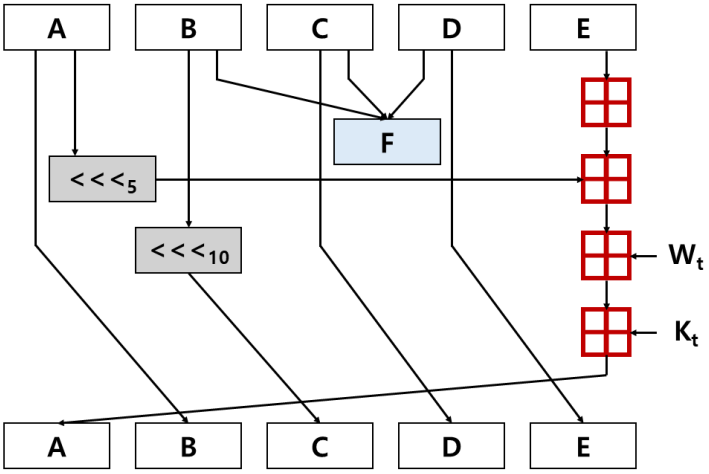

Fig. 1: An illustration of the compression function for a single round of the SHA-1 algorithm.

3 Quantum Circuit of SHA-1

This section describes a depth-optimized quantum circuit for the SHA-1 component. Since our primary goal is to minimize the depth of the quantum circuit, we employ a depth-optimized adder for our implementation. However, the number of qubits is just as important as the circuit depth. Therefore, we implement a SHA-1 quantum circuit that reduces the depth while reducing the number of qubits.

3.1 Message Padding and Round Word Computation

Classical Preprocessing The SHA-1 algorithm commences with a critical message preprocessing stage designed to transform a variable-length input into a standardized format. This process ensures the message length is congruent to 448 modulo 512 by padding it. The outcome of this stage is a message formatted into one or more sequential 512-bit blocks, which serve as the input for the subsequent computational steps.

Algorithm 1: Quantum SHA-1 Internal Functions

procedure FuncF1(*result, x, y, z, temp*)
for $i \leftarrow 0$ to 31 **do**
    CNOT | ($y[i]$, $temp[i]$)
    CNOT | ($z[i]$, $temp[i]$)
    Toffoli | ($x[i]$, $temp[i]$, $result[i]$)
    CNOT | ($y[i]$, $temp[i]$)
    CNOT | ($z[i]$, $temp[i]$)
    CNOT | ($z[i]$, $result[i]$)
end for
end procedure

procedure FuncF2(*result, x, y, z*)
for $i \leftarrow 0$ to 31 **do**
    CNOT | ($x[i]$, $result[i]$)
    CNOT | ($y[i]$, $result[i]$)
    CNOT | ($z[i]$, $result[i]$)
end for
end procedure

procedure FuncF3(*result, x, y, z, temp*)
for $i \leftarrow 0$ to 31 **do**
    CNOT | ($y[i]$, $temp[i]$)
    CNOT | ($z[i]$, $temp[i]$)
    Toffoli | ($x[i]$, $temp[i]$, $result[i]$)
    CNOT | ($y[i]$, $temp[i]$)
    CNOT | ($z[i]$, $temp[i]$)
    Toffoli | ($y[i]$, $z[i]$, $result[i]$)
    CNOT | ($temp[i]$, $result[i]$)
    Toffoli | ($y[i]$, $z[i]$, $temp[i]$)
end for
end procedure

Algorithm 2: Quantum SHA-1 Compression Function

procedure SHA1($(H_0, \ldots, H_4)$, M_{chunk})

▷ 1. Initialization
$a, b, c, d, e \leftarrow$ AllocateQubit(32) ▷ Working variables
$T, f, K_t \leftarrow$ AllocateQubit(32) ▷ Temporary registers
$(a, b, c, d, e) \leftarrow (H_0, H_1, H_2, H_3, H_4)$
$W[0..15] \leftarrow$ LoadClassicalChunk(M_{chunk})

▷ 2. Main loop: 80 rounds
for $t = 0 \rightarrow 79$ **do** ▷ *Round Logic*
  if $0 \leq t \leq 19$ **then**
    $f \leftarrow$ FuncF1(b, c, d) ▷ Choose function
    $K_t \leftarrow$ ClassicToQuantum(`0x5A827999`)
  else if $20 \leq t \leq 39$ **then**
    $f \leftarrow$ FuncF2(b, c, d) ▷ Parity function
    $K_t \leftarrow$ ClassicToQuantum(`0x6ED9EBA1`)
  else if $40 \leq t \leq 59$ **then**
    $f \leftarrow$ FuncF3(b, c, d) ▷ Majority function
    $K_t \leftarrow$ ClassicToQuantum(`0x8F1BBCDC`)
  else
    $f \leftarrow$ FuncF2(b, c, d)
    $K_t \leftarrow$ ClassicToQuantum(`0xCA62C1D6`)
  end if

  ▷ Compute round word W_t in-place for $t \geq 16$
  if $t \geq 16$ **then**
    $W[t \pmod{16}] \leftarrow \text{ROTL}(W[(t-16) \pmod{16}] \oplus W[(t-14) \pmod{16}] \oplus W[(t-8) \pmod{16}] \oplus W[(t-3) \pmod{16}], 1)$
  end if

  ▷ Compute temporary value T using sequential quantum additions
  $T \leftarrow \text{ROTL}(a, 5)$
  $T \leftarrow$ QuantumAdder$(T, f, carry)$
  $T \leftarrow$ QuantumAdder$(T, e, carry)$
  $T \leftarrow$ QuantumAdder$(T, K_t, carry)$
  $T \leftarrow$ QuantumAdder$(T, W[t \pmod{16}], carry)$

  ▷ Update state variables (quantum register swaps)
  $e \leftarrow d$
  $d \leftarrow c$
  $c \leftarrow \text{ROTL}(b, 30)$
  $b \leftarrow a$
  $a \leftarrow T$
  ▷ Uncompute this round's temporary values (f, K_t) to free qubits
end for

▷ 3. Compute final hash value
$(H'_0, \ldots, H'_4) \leftarrow (H_0 + a, H_1 + b, H_2 + c, H_3 + d, H_4 + e)$

return $(H'_0, \ldots, H'_4)$
end procedure

Algorithm 3: Quantum Operators

```
procedure ClassicToQuantum(op1, op2)
    for i ← 0 to 31 do
        b ← (op2 ≫ i) ∧ 1
        if b = 1 then
            X | op1[i]
        end if
    end for
end procedure

procedure LeftCircularShift(op1, amount)
    n ← 32
    visited ← [False] × n
    cycles ← []
    for i ← 0 to n − 1 do
        if not visited[i] then
            cycle ← []
            j ← i
            while not visited[j] do
                append j to cycle
                visited[j] ← True
                j ← (j − amount) mod n
            end while
            if length(cycle) > 1 then
                append cycle to cycles
            end if
        end if
    end for
    for all cycle ∈ cycles do
        for i ← 0 to length(cycle) - 2 do
            Swap(op1[cycle[i]], op1[cycle[i + 1]])
        end for
        Swap(op1[cycle[end]], op1[cycle[0]])
    end for
end procedure
```

Dynamic Qubit Allocation and Qubit Reuse After classical preprocessing, each 512-bit message block is loaded into the quantum circuit. This is achieved by allocating a 512-qubit register, organized into 16 blocks of 32 qubits, to store the initial words W_0 through W_{15}. The subsequent 64 words (W_{16} through W_{79}), required for the 80 compression rounds of the algorithm, are fully computed within the quantum circuit. A simple implementation would allocate new qubits for these additional words, resulting in significant resource overhead. To avoid this, this implementation utilizes a qubit reuse scheme. While the primary goal of this work is to optimize circuit depth, we introduce a crucial auxiliary optimization for qubit efficiency. The same initial 512-qubit register (organized into 16 blocks) is reused to sequentially compute and store the expanded words. This

approach allows for a full 80-word message schedule expansion without allocating any additional qubits beyond the initial 512, significantly reducing overall resource requirements. A hint at qubit reuse lies in the SHA-1 message expansion algorithm (4). We implemented it to update the word corresponding to W_{i-16} (for example, store the value of W_{16} in W_0, and store the value of W_{32} in W_{16}, i.e., W_0) by taking advantage of the regularity of the index value corresponding to each word.

$$W_i = (W_{i-3} \oplus W_{i-8} \oplus W_{i-14} \oplus W_{i-16}) \lll 1 \tag{4}$$

A key advantage of this architecture is its resource efficiency for inputs larger than a single block. Since the compression function processes the message iteratively one 512-bit block at a time, the same quantum circuit and its 512-qubit register are reused for each block. Consequently, the total qubit count remains constant regardless of the message size. It is important to note that while the qubit requirement does not grow, the total circuit depth increases linearly with the number of message blocks, as the compression function is executed for each one.

3.2 Rounds (0-79)

SHA-1 Round Structure The core of the SHA-1 algorithm is its compression function, which iteratively processes the 80-word message schedule (W_t) through 80 rounds of computation. This process updates five 32-bit working variables, denoted A, B, C, D, and E, which collectively form the internal state of the hash. The 80 rounds are partitioned into four distinct stages of 20 rounds each. Each stage is characterized by two components that vary depending on the current round t: a non-linear logical function, F_t, and a unique additive round constant, K_t. The function F_t operates on three of the working variables, namely B, C, and D, to produce a 32-bit output based on a logical operation specific to that stage. The exact logical definition of $F_t(B, C, D)$ changes for each of the four stages, drawing from a set of three distinct underlying boolean functions. Concurrently, a unique constant K_t is assigned to each 20-round stage, beginning with $K_t =$ `0x5A827999` for the first stage and concluding with $K_t =$ `0xCA62C1D6` for the final stage. These two components, the output of the function F_t and the constant K_t, along with the message word W_t for the current round, provide the necessary parameters for the main state update operations that follow. The translation of these classical, non-linear logical operations into the quantum domain represents a critical aspect of our work. Thanks to these designs, we present resource-efficient quantum circuits for each of the distinct internal functions that constitute F_t. The detailed construction of these circuits from fundamental quantum gates is provided in Algorithm 1.

State Update within a Round Once the round-specific function F_t and constant K_t are determined, the core state update takes place. First, a temporary 32-bit word, T, is computed by summing five values: the working variable a after

a circular left shift of 5 bits ($a \lll 5$), the output of the logical function F_t, the working variable e, the round constant K_t, and the current message schedule word W_t. All additions are performed modulo 2^{32}. Following this calculation, the working variables are updated in a sequential cascade. The variable e takes the value of d, d takes the value of c, and c takes the value of b after it has been circularly shifted left by 30 bits ($b \lll 30$). Subsequently, b takes the value of the original a. Finally, the variable a is updated with the value of the newly computed temporary word, T. This entire sequence of operations ensures that the internal state is thoroughly mixed in each round, which is fundamental to the algorithm's security properties. The quantum implementation of this state update, with a focus on resource optimization, is presented in the following section.

Resource-Efficient Quantum Circuit Implementation We implement the aforementioned state update process as a quantum circuit, presented in Algorithm 2. As detailed in the algorithm, our implementation prioritizes qubit efficiency through two primary resource management strategies. Firstly, we employ the compute-uncompute paradigm for the temporary qubits that store the output of the non-linear function F_t (denoted as the 'result' variable). Instead of allocating new qubits for this calculation in every round, we compute the output of the function, use it, and then immediately uncompute the result. This procedure returns the ancillary qubits to their initial $|0\rangle$ state, allowing them to be cleanly reused in the subsequent round. Secondly, this optimization approach is extended to the management of the round constants, K_t. Rather than dedicating a permanent register for each of the four distinct constant values, we dynamically load the appropriate classical value into a single 32-qubit register. Once the constant is no longer needed for the round's computation, we uncompute this register to reset it. This just-in-time allocation strategy is crucial, as it prevents the redundant allocation of qubits and contributes significantly to minimizing the overall qubit footprint.

The two functions required to successfully build a complete quantum circuit are presented in the following algorithm 3. The first function assigns classical bit strings to qubits. It is essential for initializing the quantum registers with appropriate values, using the input string and constants such as K_t. If the input value (classical bit) is 1, the X gate is used to flip the qubit value from 0 to 1. The second important subroutine is the quantum left rotation function, which implements the circular left shift ($\lll$) operation. The quantum analog of this function is implemented as an efficient network of SWAP gates that permutate the qubit states within the register. The definition of these basic functions also significantly impacts the performance of the main circuit. Therefore, it is important to keep the implementation simple so as not to waste circuit depth and the number of qubits.

Gate-level Visualization of the Internal Functions Algorithm 1 provides the procedural description for the quantum implementation of the three distinct

internal functions (F_t) used in SHA-1. To complement this, Figure 2, 3, 4 offers a detailed visualization of the gate-level circuits corresponding to each of these procedures. The diagrams directly map each line of the pseudocode into its physical quantum gate operation, illustrating the flow of quantum information and the resource utilization for each function.

Figure 2 shows the quantum circuit of `FuncF1`. First, function F1 is an internal function of SHA-1, which means (5). As can be seen in the pseudocode, this circuit first computes the logical XOR of the inputs $y[i]$ and $z[i]$ using CNOT gates and stores the result in the auxiliary qubit $temp[i]$. Next, it performs the Toffoli (CCNOT) operation using $x[i]$ and $temp[i]$ as control qubits and stores the result in $result[i]$. Subsequent gates reset the auxiliary qubits to their initial states, demonstrating the compute-uncompute paradigm. Finally, the value of the result qubit $result[i]$ is modified using $z[i]$. In contrast, the `FuncF2` circuit shown in Figure 3 is much more efficient because it does not require auxiliary qubits. Function F2 is an internal function of SHA-1, which means (6), and this algorithm implements a three-input XOR logic by applying a sequence of three CNOT gates. Each input qubit ($x[i]$, $y[i]$, $z[i]$) acts as a control for the target qubit ($result[i]$).

Finally, Figure 4 shows the implementation of `FuncF3`, the most complex of the three. Function F3 stands for (7). Similar to `FuncF1`, this circuit also requires auxiliary *temp* qubits and uses a compute-uncompute paradigm. The simple explanation for F3 is as follows. First, y and z are XORed, and then the result is ANDed with x. This is equivalent to using the first CNOT gate twice, followed by a Toffoli gate. This stores the value $(x \wedge y) \oplus (x \wedge z)$ in *result*. After performing an operation to empty *temp*, the result of the AND operation of y and z is stored in *temp* using a Toffoli gate. Finally, *result* is modified using the CNOT gate and the value stored in *temp*. After storing the final result, we use Toffoli to reset *temp* to 0. These three schematics provide a complete and transparent blueprint of the core computational units of the SHA-1 implementation, clearly showing the complexity and resource requirements of each unit.

$$(x \wedge y) \vee (\bar{y} \wedge z) \tag{5}$$

$$x \oplus y \oplus z \tag{6}$$

$$(x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z) \tag{7}$$

3.3 Final Hash Value Computation

The final stage of the SHA-1 algorithm commences after the 80 rounds of the main loop have been executed for each 512-bit message chunk. This concluding step is responsible for updating the internal state and, ultimately, producing the final message digest. The five working variables (a, b, c, d, e) that result from the main loop's computation for a given chunk are used to update the five 32-bit intermediate hash values, denoted as h_0 through h_4. Specifically, the new h_0 is

the sum of the old h_0 and a, the new h_1 is the sum of the old h_1 and b, and so on for all five variables. These newly computed hash values then serve as the initial hash values for the processing of the next message chunk. It ensures that the entire message content sequentially influences the final output. Once the very last message chunk has been processed, the five final 32-bit hash values $(h_0, h_1, h_2, h_3, h_4)$ are concatenated in order to form the classical 160-bit message digest. In the context of quantum circuit implementation, this final 160-bit value would be encoded in the quantum state of the 160 qubits comprising five hash registers. To extract this result in the quantum domain, a standard computational measurement is applied to each of these 160 qubits. This measurement compresses the quantum state into a classical 160-bit string, representing the final output of SHA-1, typically expressed as a 40-digit hexadecimal number in big-endian format.

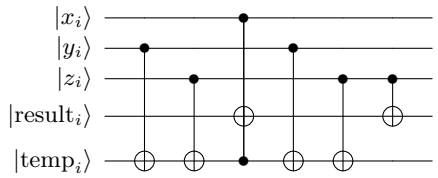

Fig. 2: The quantum circuit diagram for FuncF1.

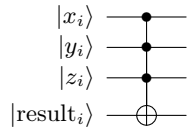

Fig. 3: The quantum circuit diagram for FuncF2.

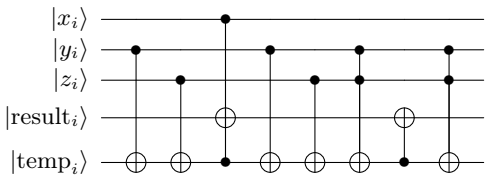

Fig. 4: The quantum circuit diagram for FuncF3.

4 Benchmark

4.1 SHA-1 Quantum Circuit With Draper Adder

The Draper adder, known as the Quantum Carry-Lookahead Adder (QCLA), is a widely recognized quantum arithmetic circuit notable for its logarithmic depth performance [15]. Drawing inspiration from classical carry-lookahead techniques, this adder contrasts sharply with earlier linear-depth ripple-carry adders. As detailed by Draper et al., the adder can sum two n-bit numbers in $O(\log n)$ depth using $O(n)$ ancillary qubits. This logarithmic depth provides a significant reduction in the required runtime for quantum algorithms that rely heavily on arithmetic, such as Shor's algorithm. For the implementation of our SHA-1 quantum circuit, we prioritized minimizing the overall circuit depth, as depth is a critical bottleneck on near-term quantum devices. Utilizing the Draper adder model allowed us to construct a circuit with a total depth of 9026, requiring 985 qubits for the complete implementation.

4.2 SHA-1 Quantum Circuit With TTK Adder

An alternative approach to quantum addition is presented by Takahashi, Tani, and Kunihiro (TTK), with a primary focus on minimizing the number of required qubits [16]. The foundational TTK adder design is remarkable for its ability to sum two n-bit numbers with no ancillary qubits, a significant achievement in qubit economy. However, this qubit efficiency comes at the cost of increased circuit depth, with the base model exhibiting a linear depth of $O(n)$. While the authors propose methods to reduce the depth, these often involve trade-offs or the use of specialized gates. To evaluate this trade-off in our context, we implemented the SHA-1 circuit using the TTK adder model. This configuration reduced the total qubit requirement to 929, but at the expense of a substantial increase in circuit depth, which measured 43232.

4.3 Comparative Resource Analysis

In selecting the optimal adder for our SHA-1 quantum circuit, we conducted a detailed comparison of the resource requirements for implementations based on the Draper and TTK models. A high-level summary of the qubit and depth trade-off is presented in Table 1. The implementation using the Draper adder resulted in a configuration of 985 qubits and a circuit depth of 9026, whereas the TTK-based implementation required 929 qubits but had a depth of 43232.

Although the TTK adder offered a savings of 56 qubits, this came with a nearly five-fold increase in total circuit depth. To understand the underlying reasons for this trade-off, we performed a detailed gate-level resource estimation, summarized in Table 1.

Gate-Level Analysis Table 1 reveals the distribution of fundamental quantum gates for each implementation. While the qubit count is lower for the TTK model, it necessitates a significantly larger number of multi-qubit gates, particularly Toffoli gates. This proliferation of complex gates is the primary contributor to its substantial circuit depth.

Fault-Tolerant Resource Metrics For a more holistic assessment of computational cost, especially in the context of fault-tolerant quantum computing, it is useful to consider metrics that capture both space (qubits) and time (depth) resources. One such metric is the Depth-Width (DW) Cost, which represents the overall space-time volume of the computation. As shown in Table 2, the Draper-based implementation demonstrates superior efficiency. Its DW-Cost of 8.89×10^6 is more than four times lower than the 4.02×10^7 cost of the TTK model. This significant difference highlights the practical advantage of the Draper adder's architecture.

In the era of Noisy Intermediate-Scale Quantum (NISQ) computers and beyond, circuit depth is often a more critical limiting factor than the absolute qubit count. The substantial performance gain and lower overall computational cost, as evidenced by the DW-Cost, offered by the much shallower Draper adder were determined to far outweigh its modest additional qubit cost. Consequently, we selected the Draper adder as the foundational arithmetic component for our final, optimized SHA-1 circuit.

Table 1: Detailed Gate-Level Resource Estimation for the two SHA-1 circuit implementations. All gate counts are obtained using the ProjectQ [17] resource counter. (* **Input size $\leq$ 512 bits**)

Metric	**Draper Adder**	**TTK Adder**
Qubits	985	929
Toffoli (CCX) Gates	90,940	24,315
CNOT (CX) Gates	67,879	78,279
X Gates	20,339	189
Circuit Depth	**9,026**	**43,232**

Table 2: Fault-Tolerant Resource Estimation and Costs. The DW-Cost is calculated as (Qubits $\times$ Circuit Depth). (* **Input size $\leq$ 512 bits**)

Metric	**Draper Adder**	**TTK Adder**
Full Depth	9,026	43,232
DW-Cost	8.89×10^6	4.02×10^7

5 Conclusion

Quantum computing advancements pose a significant threat to the security of ubiquitous cryptographic algorithms, including classical hash functions such as SHA-1. Although the theoretical susceptibility of SHA-1 to quantum attacks like Grover's algorithm is well-established, a precise assessment of the danger requires a detailed understanding of the practical resources involved. Our study aimed to fill this knowledge gap by creating a comprehensive and resource-optimized quantum circuit for SHA-1. We placed a special emphasis on reducing the circuit's depth, as this is a primary performance limitation for current and near-future quantum hardware.

This paper presents a depth-optimized quantum circuit design for SHA-1, including gate-level implementations of all core components. Key optimizations, such as just-in-time qubit allocation and extensive compute-uncompute recycling, significantly reduce qubit overhead. We also compare two quantum adders, finding that although the TTK adder saves some qubits, the Draper adder's nearly five-fold depth reduction makes it the clear choice for minimizing decoherence on Noisy Intermediate-Scale Quantum (NISQ) hardware.

First, it provides a concrete blueprint and accurate resource estimates for the quantum implementation of SHA-1, offering a solid baseline for quantum pre-image attacks using Grover's algorithm. Second, our modular design and optimization strategies—especially dynamic resource management and systematic compute-uncompute—offer a practical framework for implementing other complex cryptographic algorithms on quantum computers. The presented algorithms and circuit diagrams form a transparent foundation for future quantum cryptanalysis research.

Our subsequent research will investigate better depth-qubit tradeoffs for SHA-1 quantum circuits and accurate resource estimation for Grover's algorithm. We will also focus on quantum circuit implementations for other cryptographic algorithms. We are confident that the circuit implementation methodology proposed in this study can be applied to quantum circuit optimization for various cryptographic functions, including other hash functions, which will advance quantum threat research for modern cryptography.

6 Acknowledgment

This work was supported by the Institute of Information & Communications Technology Planning & Evaluation(IITP) grant funded by the Korea government(MIST) (No. RS-2019-II190033, Study on Quantum Security Evaluation of Cryptography based on Computational Quantum Complexity, 75%) and this work was supported by Institute for Information & communications Technology Promotion(IITP) grant funded by the Korea government(MSIT) (No.2018-0-00264, Research on Blockchain Security Technology for IoT Services, 25%).

References

1. M. Grassl, B. Langenberg, M. Roetteler, and R. Steinwandt, "Applying Grover's algorithm to AES: Quantum resource estimates," in *Post-Quantum Cryptography* (T. Takagi, ed.), (Cham), pp. 29–43, Springer International Publishing, 2016. 3
2. S. Jaques, M. Naehrig, M. Roetteler, and F. Virdia, "Implementing Grover Oracles for quantum key search on AES and LowMC," in *EUROCRYPT 2020, Zagreb, Croatia, May 10-14, 2020, Proceedings, Part II* (A. Canteaut and Y. Ishai, eds.), vol. 12106 of *Lecture Notes in Computer Science*, pp. 280–310, Springer, 2020. 3
3. K. Jang, A. Baksi, H. Kim, G. Song, H. Seo, and A. Chattopadhyay, "Quantum analysis of AES." Cryptology ePrint Archive, Paper 2022/683, 2022. https://eprint.iacr.org/2022/683. 3
4. Q. Liu, B. Preneel, Z. Zhao, and M. Wang, "Improved quantum circuits for AES: Reducing the depth and the number of qubits." Cryptology ePrint Archive, Paper 2023/1417, 2023. https://eprint.iacr.org/2023/1417. 3
5. National Institute of Standards and Technology, "Secure Hash Standard," Tech. Rep. FIPS PUB 180, Fips pub, May 1993. 3
6. National Institute of Standards and Technology, "Secure Hash Standard," Tech. Rep. FIPS PUB 180-1, Fips pub, April 1995. 3
7. National Institute of Standards and Technology, "Digital Signature Standard (DSS)," Tech. Rep. FIPS PUB 186-5, Fips pub, February 2023. 3
8. National Institute of Standards and Technology, "Secure Hash Standard (SHS)," Tech. Rep. FIPS PUB 180-4, Fips pub, August 2015. 4
9. X. Wang, Y. L. Yin, and H. Yu, "Finding collisions in the full sha-1," *Advances in Cryptology – CRYPTO 2005: 25th Annual International Cryptology Conference, Santa Barbara, California, USA, August 14-18, 2005, Proceedings*, vol. Proceedings 25, pp. 17–36, 2005. 4
10. Barker, Elaine, and A. Roginsky, "Transitioning the Use of Cryptographic Algorithms and Key Lengths," Tech. Rep. SP 800-131A Revision 2, National Institute of Standards and Technology, 2019. 4
11. M. Stevens, E. Bursztein, P. Karpman, A. Albertini, and Y. Markov, "The first collision for full sha-1," *Advances in Cryptology – CRYPTO 2017*, pp. 570–596, 2017. 4
12. G. Leurent and T. Peyrin, "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. 271–288, USENIX Association, 2020. 4
13. Turner, James M., "The Keyed-Hash Message Authentication Code (HMAC)," Tech. Rep. FIPS 198-1, Federal Information Processing Standards Publication, July 2008. 5
14. Sönmez Turan, Meltem, and Luís TAN Brandão, "Keyed-Hash Message Authentication Code (HMAC): Specification of HMAC and Recommendations for Message Authentication," Tech. Rep. NIST SP 800-224(Draft), National Institute of Standards and Technology, June 2024. 5
15. T. G. Draper, S. A. Kutin, E. M. Rains, and K. M. Svore, "A logarithmic-depth quantum carry-lookahead adder," *arXiv preprint quant-ph/0406142*, 2004. 13
16. Y. Takahashi, S. Tani, and N. Kunihiro, "Quantum addition circuits and unbounded fan-out," *arXiv preprint arXiv:0910.2530*, 2009. 13
17. D. S. Steiger, T. Häner, and M. Troyer, "ProjectQ: An open source software framework for quantum computing," *Quantum*, vol. 2, p. 49, 2018. 14