

# Grover on SM3

Gyeongju Song<sup>1</sup>, Kyungbae Jang<sup>1</sup>, Hyunji Kim<sup>1</sup>,  
Wai-Kong Lee<sup>2</sup>, Zhi Hu<sup>3</sup>, and Hwajeong Seo<sup>1</sup>[0000-0003-0069-9061]

<sup>1</sup>IT Department, Hansung University, Seoul (02876), South Korea,  
{thdrudwn98, starj1023, khj1594012, hwajeong84}@gmail.com

<sup>2</sup>Department of Computer Engineering,  
Gachon University, Seongnam, Incheon (13120), Korea,  
waikonglee@gachon.ac.kr <sup>3</sup>Central South University, China,  
huzhi\_math@csu.edu.cn

**Abstract.** Grover’s search algorithm accelerates the key search on the symmetric key cipher and the pre-image attack on the hash function. To perform Grover’s search algorithm, the target algorithm should be implemented in a quantum circuit. For this reason, we propose an optimal SM3 hash function (Chinese standard) in a quantum circuit. We focused on minimizing the use of qubits and reducing the use of quantum gates. To do this, the on-the-fly approach is utilized for message expansion and compression functions. In particular, the previous value is restored and used without allocating new qubits in the permutation operation. Finally, we estimate the quantum resources required for the quantum pre-image attack based on the proposed SM3 hash function implementation in the quantum circuit.

**Keywords:** Quantum Computer · Grover Algorithm · SM3 Hash Function.

## 1 Introduction

Quantum computers can solve specific problems in quantum algorithms much faster than classical computers. Two representative quantum algorithms that work on quantum computers are Shor’s algorithm [1] and Grover’s algorithm [2]. Shor’s algorithm leads to vulnerability of Rivest–Shamir–Adleman (RSA) and elliptic curve cryptography (ECC), the most commonly used public key cryptography approaches. Integer factorization and discrete logarithm problems used in RSA and ECC are hard problems in classical computers. However, quantum computers using Shor’s algorithm solve these hard problems within a polynomial time. To prevent this kind of attack, the National Institute of Standards and Technology (NIST) is working on standardizing post-quantum cryptography. In the standardization process, various post-quantum algorithms have been submitted. Grover’s algorithm accelerates finding specific data in databases (i.e. brute force attacks). If  $O(n)$  queries were required in a brute force attack, this number can be reduced to  $O(\sqrt{n})$  queries by using Grover’s algorithm. In cryptography, Grover’s algorithm lowers the  $n$ -bit security-level symmetric key cipher and hash function to  $\frac{n}{2}$ -bit (i.e. half) for key search and pre-image attack.

In recent years, there has been active research on how to optimize and implement symmetric key ciphers [3–13] and hash functions [14] as quantum circuits to minimize the quantum resources required for Grover’s algorithm. In [15], quantum cryptanalysis benchmarking was performed by comparing resources required to attack public key cryptography, symmetric key cryptography, and hash function.

In quantum circuit optimization, it is important to reduce the number of qubits and quantum gates. The most important factor is to reduce the number of required qubits. As the number of qubits increases, quantum computers become more difficult to operate in a practical manner. International companies, such as IBM, Google, and Honeywell, are in the process of increasing the number of qubits for high computing quantum computers.

In this work, we focused on minimizing the number of qubits required to implement the SM3 hash function in a quantum circuit, while at the same time reducing the complexity of quantum gates. The existing message expansion function was divided into the first extension and second extension. The compression function was divided into the first compression and second compression, and then mixed and used. Through this method, the total number of qubits used was reduced by reusing the qubits used in the message. In the permutation operation, the value was returned through the CNOT-gate repetition rather than using a qubit to store the original value. Thus, we achieved an optimal quantum circuit of the SM3 hash function. In this paper, we used 2,176 qubits for storing the extended message ( $W_j$  ( $j = 0, 1, \dots, 67$ )), 32 qubits for the  $T$  constant to be used for the update, and 256 qubits for the register update and output of the final hash value. We also used 32 qubits for permutation operations, 1 qubit for ripple-carry addition, and 224 qubits for AND and OR operations.

## 1.1 Contribution

- **First implementation of the SM3 hash function in a quantum circuit** To the best of our knowledge, this is the first implementation of the SM3 hash function in a quantum circuit. We obtained the optimal quantum circuit by minimizing the use of qubits and reducing the quantum gate complexity.
- **Efficient design of SM3 operations in a quantum circuit** We reduced the use of qubits by dividing the expansion function and compression function of the original SM3 hash function and mixing them. Permutation operations were also performed with minimum qubits.
- **Quantum resource estimation of Grover’s search algorithm for the SM3 hash function** We evaluate quantum resources for the quantum pre-image attack on the SM3 hash function. A quantum programming tool, namely, IBM ProjectQ [16], was used to evaluate the proposed quantum implementation of SM3 hash function.

## 2 Related Work

### 2.1 SM3 Hash Function

The hash function completely changes the output value with only small changes in the input value, thus ensuring the integrity by detecting errors in a message. The hash function efficiently generates the hashed value, allowing it to be digitally signed and verified, and to generate and verify messages. The SM3 hash function is operated in units of 32 words, and it finally outputs a hash value of 256 bits. After increasing the message length using padding, the message expansion calculation is performed by Equation 1 to expand the message to  $W_0, W_1, \dots, W_{67}, W'_0, \dots, W'_{63}$ .

$$\begin{aligned} W_j &\leftarrow P_j(W_{j-16} \oplus W_{j-9} \oplus (W_{j-3} \lll 15)) \oplus (W_{j-13} \lll 7) \oplus W_{j-6} \\ W'_j &= W_j \oplus W_{j+4}, \quad (16 \leq j \leq 67) \end{aligned} \quad (1)$$

The message expansion function expands the message block  $B^{(i)}$  to 132 words ( $W_0, W_1, \dots, W_{67}, W'_0, \dots, W'_{63}$ ). First, the existing message block  $B^{(i)}$  is divided into 16 words  $W_0, W_1, \dots, W_{15}$  and expanded to  $W_{16}, \dots, W_{67}$  using this. The expanded message makes  $W_0, W_1, \dots, W_{67}, W'_0, W'_1, \dots, W'_{63}$  through the Equation 1. The extended 132-word message is updated to registers  $A$  to  $H$  through the compression function. Registers  $A$  to  $H$  are 32 bits each, and initial values are stored. The final hash value is generated by performing the XOR operation to the updated register value with the previous register value through the compression function.

---

**Algorithm 1** Compression function of the SM3 hash function.

---

**Input:**  $W_0, W_1, \dots, W_{67}, W'_1, \dots, W'_{63}$

**Output:** 32-qubits-register  $A, B, C, D, E, F, G, H$  after the message compression.

```

1: for  $j = 0$  to 63 do
2:    $SS1 \leftarrow ((A \lll 12) + E + (T_j \lll (j \bmod 32)) \lll 7)$ 
3:    $SS2 \leftarrow SS1 \oplus (A \lll 12)$ 
4:    $TT1 \leftarrow FF_j(A, B, C) + D + SS2 + W'_j$ 
5:    $TT2 \leftarrow GG_j(E, F, G) + H + SS1 + w_j$ 
6:    $D \leftarrow C$ 
7:    $C \leftarrow B \lll 9$ 
8:    $B \leftarrow A$ 
9:    $A \leftarrow TT1$ 
10:   $H \leftarrow G$ 
11:   $G \leftarrow F \lll 19$ 
12:   $F \leftarrow E$ 
13:   $E \leftarrow P_0(TT2)$ 
14: end for
15:  $V(i+1) \leftarrow ABCDEFGH \oplus V(i)$ 
16: return  $A, B, C, D, E, F, G, H$ 

```

---

The compression function proceeds to  $V^{(i+1)} = CF(V^{(i)}, B^{(i)})$ ,  $i = 0, \dots, n-1$  with the message of 132 words expanded in the message expansion function and previous 256-bit values as parameters.  $SS1$ ,  $SS2$ ,  $TT1$ , and  $TT2$  are intermediate variables of 32 bits, and  $T$  in the  $SS1$  update process contains the initial value of 32 bits. The  $FF$  and  $GG$  functions are Boolean functions that perform XOR, AND, and OR operations of parameters and output a value of 32 bits.  $FF$  and  $GG$  are used to update  $TT1$  and  $TT2$ . Equation 2 is the calculation of  $FF$  and  $GG$  functions.

$$\begin{aligned}
 FF_j(X, Y, Z) &= X \oplus Y \oplus Z, \quad 0 \leq j \leq 15 \\
 FF_j(X, Y, Z) &= (X \wedge Y) \vee (X \wedge Y) \vee (Y \wedge Z), \quad 16 \leq j \leq 63 \\
 GG_j(X, Y, Z) &= X \oplus Y \oplus Z, \quad 0 \leq j \leq 15 \\
 GG_j(X, Y, Z) &= (X \wedge Y) \vee (\neg X \wedge Z), \quad 16 \leq j \leq 63
 \end{aligned} \tag{2}$$

In the compression function, the last register value is stored by updating the register 64 times, and the final hash value of 256-bits is generated through an XOR operation with the register before the update.

## 2.2 Quantum Computing

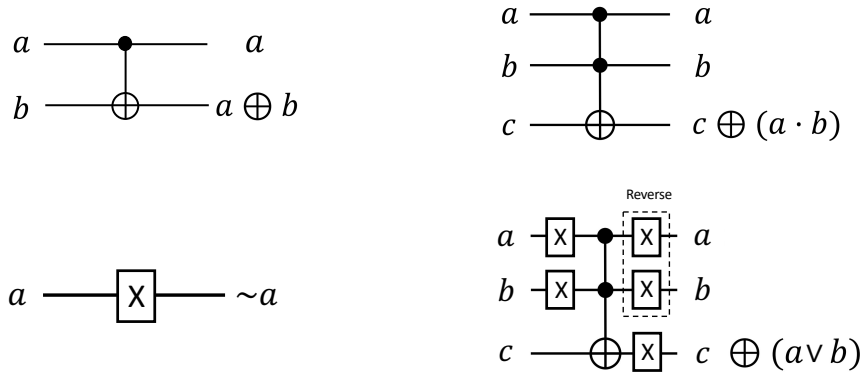


Fig. 1: CNOT gate, Toffoli gate, X gate, and OR gate in quantum gates.

Quantum computers utilize quantum mechanics phenomena, such as superposition and entanglement. A classical computer has bits, while a quantum computer has qubits that can superpose 0 and 1. In other words, a qubit has both values in the probability of being 0 and 1, and it is determined when it is measured.

As shown in Figure 1, quantum circuits also have quantum logic gates, such as digital logic gates in digital circuits. The quantum gate can control the state of the qubit. The X gate is a quantum logic gate that corresponds to the NOT gate of a digital logic gate. The probability that the qubit state becomes 0 is

changed to the probability that it is determined as 1. The CNOT gate represents an entangled state in which one qubit affects another qubit. It performs a NOT gate operation for the second qubit when the first qubit is 1. If the first qubit is 1, the NOT gate is applied to the second qubit. Otherwise, the second qubit is the output as it is. With the Toffoli gate, the states of two qubits affect the state of one qubit. If the first two qubits among the three qubits are 1, a NOT operation is performed for the third qubit. Otherwise, the value of the third qubit is not changed.

In addition, there is a Hadamard gate, which puts qubits in a superposition state, and a SWAP gate, which changes the state of two qubits. The Toffoli gate is an expensive gate. X gate and CNOT gate are relatively inexpensive in comparison with the Toffoli gate. Because quantum computers with a large number of qubits have not been developed, quantum circuits must be designed with consideration of resources, such as qubits and quantum gates.

There are platforms for quantum computing, such as IBM's ProjectQ, Qiskit, or Microsoft's Q#. These platforms provide quantum gates, a variety of libraries, and simulators. Through the Qiskit platform, it is possible to use real quantum processors in the cloud platform. Thus quantum computing technologies are actively being developed, including various quantum computing platforms and quantum languages.

### 2.3 Grover Search Algorithm

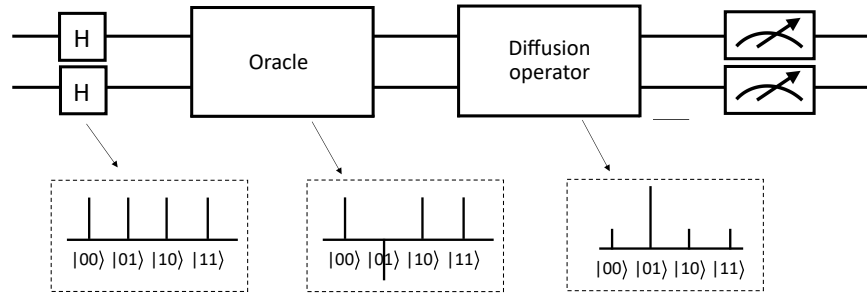


Fig. 2: Grover search algorithm (answer  $x = 01$ ).

Grover's search algorithm [2] is a quantum algorithm that searches a space with  $n$  elements to find the input data that generates the output of a particular function. On a classic computer,  $n$  searches are required to search an unsorted database. Because Grover's search algorithm can find the answer by searching for the  $\sqrt{n}$ , the time complexity is reduced from  $O(n)$  to  $O(\sqrt{n})$ . In other words, Grover's algorithm threatens the symmetric key cryptography because it shortens the time required for brute force attacks.

Grover's search algorithm consists of oracle and diffusion operators, and its steps are carried out as follows. First, Hadamard gates are applied to qubits. The oracle function  $f(x)$  returns 1 when  $x$  is the answer, and it inverts the phase of qubits representing the answer. Then, the diffusion operator amplifies the amplitude of inverted qubits through the oracle, increasing the probability of becoming the answer. Through repetition of the oracle and diffusion process, the probability of the answer is over the threshold. Finally, the value of  $x$  that exceeds the threshold becomes the answer. The overall structure of Grover's search algorithm when the answer  $x = 01$  is shown in Figure 2.

### 3 Proposed Method

#### 3.1 SM3 Hash Function on Quantum Circuit

In the SM3 hash function designed in quantum circuits, we estimate the resource for applying Grover's algorithm based on a message padded with 512 bits. We propose a method of recycling message qubits by mixing the padded message with the message expansion function and the compression function. Two word messages ( $W_j, W'_j$  ( $j = 0, 1, \dots, 63$ )) are included to update the register once with the compression function. First, we propose the method to update  $W'_j$  ( $j = 0, 1, \dots, 63$ ) to the  $W_j$  ( $j = 0, 1, \dots, 63$ ) message and save qubits through recycling. Second, we present how to update and use existing assigned qubits instead of additional qubits for intermediate variables ( $SS1, SS2, TT1$ ), and  $TT2$  used by the existing SM3 hash function. Because a qubit cannot be reset, its own ongoing permutation operations require a qubit assignment for each register update. To prevent this, we save qubits by not allocating qubits and replacing the CNOT gates with repetitive tasks. Figure 3 shows the progress of the proposed system. Our SM3 quantum circuit proceeds in the order of first message expansion, first compression, second message expansion, second compression.

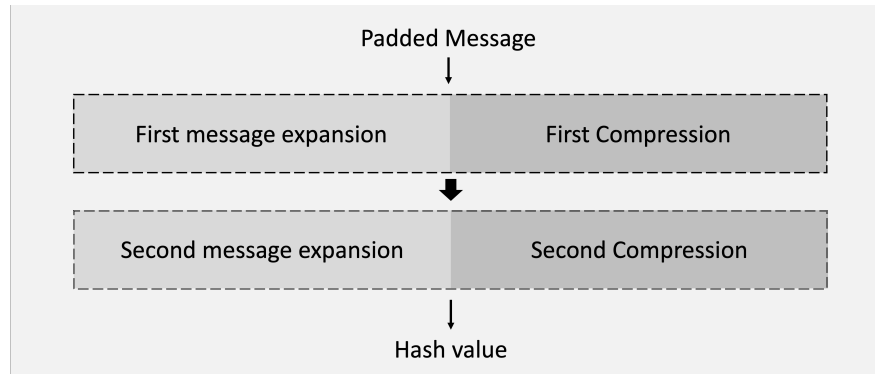


Fig. 3: System configuration for the proposed method.

### 3.2 Message Expansion

The original SM3 hash function outputs a hash function by expanding a message and then updating a register through a compression function. However, applying these methods to quantum circuits is inefficient, because 4,224 qubits are required only for message expansion. To solve this problem, we store the padded 512-bit message  $B$  in  $W_0, W_1, \dots, W_{15}$  and update  $W_{16}, W_{17}, \dots, W_{67}$  using permutation operations and CNOT gates. Updated values ( $W_0, W_{17}, \dots, W_{67}$ ) are used for the first compression function and then recycled to update  $W'_0, W'_1, \dots, W'_{63}$  in the second compression function without allocating additional qubits. Therefore, the message expansion function and the compression function are divided into the first message expansion function and the second message expansion function, the first compression function and the second compression function, which are used in combination.

---

**Algorithm 2** First message expansion quantum circuit algorithm.

---

**Input:**  $W_0, W_1, \dots, W_{15}$ .

**Output:**  $W_{16}, W_{17}, \dots, W_{67}$ .

```

1: Update:
2:   for  $i = 0$  to 31 do
3:      $W_{j-16}[i] \leftarrow \text{CNOT}(W_{j-9}[i], W_{j-16}[i]), j = 16, \dots, 67$ 
4:      $W_{j-16}[i] \leftarrow \text{CNOT}(W_{j-3}[(i+15)\%32], W_{j-16}[i]), j = 16, \dots, 67$ 
5:   end for
6:    $\text{Permutation}_{P1}(W_{j-16})$ 
7: for  $i = 0$  to 31 do
8:    $W_j[i] \leftarrow \text{CNOT}(W_{j-16}[i], W_j[i]), j = 16, \dots, 67$ 
9:    $W_j[i] \leftarrow \text{CNOT}(W_{j-13}[(i+15)\%32], W_j[i]), j = 16, \dots, 67$ 
10:   $W_j[i] \leftarrow \text{CNOT}(W_{j-6}[(i+15)\%32], W_j[i]), j = 16, \dots, 67$ 
11: end for
12: Update(reverse)
13: return  $W_{16}, W_{17}, \dots, W_{67}$ 

```

---

Algorithm 2 is the first message expansion quantum circuit, which updates  $W_{16}, W_{17}, \dots, W_{67}$ . In the first message expansion algorithm,  $W_j$  ( $16 \leq j \leq 67$ ) is generated using  $W_{(j-16)}, W_{(j-9)}, W_{(j-3)}, W_{(j-13)}$  and  $W_{(j-6)}$  ( $16 \leq j \leq 67$ ). Since qubits cannot perform simple allocation operations, CNOT gate operation values in lines 3 and 4 are stored in  $W_{(j-16)}$ . Because  $W_j$  is generated and the previous message value should not be changed, the update result value is stored in  $W_j$ , and the value of  $W_{(j-16)}$ , which is changed during the update process, is reversed and returned. The  $\text{Permutation}_{P1}$  function in line 6 performs Equation 4. Line 12 reverses lines 2 through 10.

In the second expansion function, the CNOT gate operation is performed on the message ( $W_0, \dots, W_{67}$ ) used in the first compression function, and a new message ( $W'_0, \dots, W'_{63}$ ) is output. In this way, the qubit is reused. The message

---

**Algorithm 3** Second message expansion quantum circuit algorithm.

---

**Input:**  $W_k, W_{k+4}, k = 0, \dots, 63$ .

**Output:**  $W'_t, t = 0, \dots, 63$ .

```

1: for  $i = 0$  to 31 do
2:    $W'_j[i] \leftarrow \text{CNOT}(W_j[i], W_{j+4}[i]), j = 0, \dots, 63$ 
3: end for
4: return  $W'_t, t = 0, \dots, 63$ 

```

---

$(W'_0, \dots, W'_{63})$  generated by the second expansion function is used by the second compression function.

### 3.3 Message Compression

The compression function uses an extended message to update the register. Both  $W_0, \dots, W_{63}$  and  $W'_0, \dots, W'_{63}$  are required to use the compression function. After using  $W_0, \dots, W_{63}$ , we reuse it as  $W'_0, \dots, W'_{63}$  to reduce the use of qubits. The first expansion function is executed, and the obtained value  $(W_0, \dots, W_{63})$  is to run the first compression function. Then, the second expansion function is to generate the value  $(W'_0, \dots, W'_{63})$  and performs the second compression function. Algorithms 4 and 5 are the first compression function and the second compression function, respectively.

---

**Algorithm 4** First compression quantum circuit algorithm.

---

**Input:** 32-qubits-register  $A, B, C, D, E, F, G, H, W_0, \dots, W_{63}$ .

**Output:** 32-qubits-register  $A, B, C, D, E, F, G, H$  after the first compression.

```

1: Update:
2:  $T_j \leftarrow (T_j \lll j \bmod 32) \lll 7, j = 0, \dots, 63$ 
3:  $value0 \leftarrow GG$ 
4:  $value1 \leftarrow FF$ 
5:  $E \leftarrow SS1$ 
6:  $A \leftarrow SS2$ 
7:  $H \leftarrow TT2$ 
8: return  $A, B, C, D, E, F, G, H$ 

```

---

$$\begin{aligned}
SS1 &= ((A \lll 12) + E + T_j) \lll 7, \quad j = 0, \dots, 63 \\
SS2 &= E \oplus (A \lll 12) \\
TT1 &= FF_j + D + A + W'_j \\
TT2 &= GG_j + H + SS1 + W_j
\end{aligned} \tag{3}$$

The first compression function given in Algorithm 4 calculates the constants required for the register update. Using qubits as intermediate variables in quantum circuits consumes a lot of resources. Therefore, the calculation is performed



---

**Algorithm 5** Second compression quantum circuit algorithm.

---

<b>Input:</b> 32-qubits-register	$A, B, C, D,$	
	$E, F, G, H, W'_0, \dots, W'_{63}.$	5: $B \leftarrow B \lll 9$
<b>Output:</b> 32-qubits-register	$A, B, C, D,$	6: $F \leftarrow F \lll 19$
	$E, F, G, H$ after the second compression.	7: $Swap(A, H)$
1: $D \leftarrow TT1$		8: $Swap(B, H)$
		9: $Swap(C, H)$
2: <b>Update of first compression (reverse)</b>		10: $Swap(D, H)$
		11: $Swap(E, H)$
3: $H \leftarrow Permutation_{p0}$		12: $Swap(F, H)$
		13: $Swap(G, H)$
4: $Swap(D, H)$		14: <b>return</b> $A, B, C, D, E, F, G, H$

---

in the register where the final value will be stored. In the first compression function, qubits of each 32-bit intermediate constant ( $SS1, SS2, TT1$ , and  $TT2$ ) are stored. Constants ( $SS1, SS2, TT1$ , and  $TT2$ ) are calculated by equation 3. In the first compression function, Boolean functions ( $GG$  and  $FF$ ) are used to calculate the value.  $GG$  and  $FF$  are calculated as 2, and the final result is stored in the variables ( $value0, value1$ ) and used for calculating  $TT1$  and  $TT2$ . The value of existing register  $E$  is not used after  $GG$  function and  $SS1$  update. Therefore, the value of  $SS1$  is calculated in the register  $E$ . Because the value of the existing register  $A$  is not used after the  $FF$  function, it is stored and used in the  $SS2$  value register  $A$ .  $TT2$  is updated with the extended message ( $W_0, \dots, W_{63}$ ) and the  $SS1$  value stored in the register  $E$ . At this time, the value of register  $H$  is not used after  $TT2$  operation. The  $TT2$  value is stored in register  $H$ . As a result, the value of  $TT2$  after the first compression function is stored in the register  $H$ . Since the extended message ( $W_0, \dots, W_{63}$ ) in the first message expansion function is not use after being used for the  $TT2$  update in the first compression function. Thus, the first compression function is finished, and the message ( $W'_0, \dots, W'_{63}$ ) is updated to the message ( $W_0, \dots, W_{63}$ ) through the second expansion function based on Algorithm 3. Finally, we use the updated message ( $W'_0, \dots, W'_{63}$ ) to proceed with the second compression function(Algorithm 5).

$TT1$  updates with the extended message ( $W'_0, \dots, W'_{63}$  and  $SS2$ ) stored in register  $A$ . We use the updated message ( $W'_0, \dots, W'_{63}$ ) to proceed with the second compression function.  $TT1$  updates with the extended message ( $W'_0, \dots, W'_{63}$ ) and  $SS2$  stored in register  $A$ . At this time, the value of register  $D$  is not used after the  $TT1$  operation. The  $TT1$  value is stored in register  $D$ . To update the register, the original  $A$  and  $E$  register values are required. Therefore, lines 2 to 6 of the first compression function are reversed. Then, register  $H$  is computed with the  $permutation_{P1}$  operation and all registers are updated through a swap

operation. A swap operation only changes the bit position. For this reason, there are no additional resources.

### 3.4 Hash Value

After the first expansion function is used, the first compression function, second expansion function, and second compression function are repeated 64 times in order. By completing the iteration, the updated registers ( $A, B, C, D, E, F, G$ , and  $H$ ) are XOR with the previous registers ( $A, B, C, D, E, F, G$ , and  $H$ ).

### 3.5 Permutation

In the SM3 hash function, there are two permutation functions ( $P_0$  and  $P_1$ ). Equation 4 is the expression of  $P_0$  and  $P_1$ .

$$\begin{aligned} P_0(X) &= X \oplus (X \lll 9) \oplus (X \lll 17) \\ P_1(X) &= X \oplus (X \lll 15) \oplus (X \lll 23) \end{aligned} \quad (4)$$

The  $P_0$  and  $P_1$  permutation operations shift themselves and use the CNOT gate. If the operation value is saved, it is difficult to find the original qubit value, which causes problems in subsequent operations. In normal cases, original values of qubits should be stored and used qubits are used. However in the  $P_1$  operation, a qubit is to store the value before the operation is allocated. Then, it can be used again in the next operation through the reverse operation. Therefore, a 32-bit storage qubit is allocated and used. In the  $P_0$  operation, the stored qubit cannot be reused by the reverse operation. There is a problem that 32 qubits must be allocated every time, and the compression function update should be repeated. To solve this problem, if the same bit is counted twice as the CNOT gate, the counting is canceled. As a result, in  $P_0$ , the permutation operation is performed through the repeated use of the CNOT gate without allocation of a qubit.  $P_0$  and  $P_1$  are used in the compression function. Algorithm 6 represents a part of this operation, and Table 1 presents the state changes as the operation progresses.

When  $A = a_{31}, \dots, a_0$  is given, and  $a_0$  is the most significant bit. The CNOT gate is executed in the order of  $a_{31}, \dots, a_{17}$ . It is difficult to find the original  $a_{31}$  required in the calculation of  $a_{16}$ . Therefore, the operation to find the existing value is performed by repeatedly using the CNOT gate. Algorithm 6 computes  $a_{16}$  as part of  $P_0$ . At this time, the CNOT gate is repeatedly used to use the original  $a_{31}$ , and the state change for each use is shown in Table 1. Because the XOR operation values of  $a_{16}$ ,  $a_7$  and  $a_{31}$  should be stored in  $a_{16}$ , they are calculated in order. Because the calculation is performed from  $a_{31}$ , the values of  $a_{16}$  and  $a_7$  are preserved. In line 1, the XOR values of  $a_{16}$  and  $a_7$  are stored in  $a_{16}$ . In line 2, the value of  $a_{31}$  is executed with the XOR operation. At this time, the XOR values of  $a_{31}$ ,  $a_{22}$ , and  $a_{14}$  are stored in  $a_{31}$ . Because  $a_{22}$  and  $a_{14}$  are unnecessary values, we use the CNOT gate once more to cancel them. In line 4, the CNOT gate is used to neutralize the  $a_{14}$  value. In line 3, the CNOT gate is

---

**Algorithm 6** Part of the  $P_0$  calculation.

---

**Input:**  $a_{16}$ .  
**Output:**  $a_{16} \leftarrow a_{16} \oplus a_7 \oplus a_{31}$ .  
1:  $a_{16} \leftarrow \text{CNOT}(a_7, a_{16})$   
2:  $a_{16} \leftarrow \text{CNOT}(a_{31}, a_{16})$   
3:  $a_{16} \leftarrow \text{CNOT}(a_{22}, a_{16})$   
4:  $a_{16} \leftarrow \text{CNOT}(a_{14}, a_{16})$   
5:  $a_{16} \leftarrow \text{CNOT}(a_{13}, a_{16})$   
6:  $a_{16} \leftarrow \text{CNOT}(a_5, a_{16})$   
7: **return**  $a_{16} \leftarrow a_{16} \oplus a_7 \oplus a_{31}$

---

Table 1: Changes of states during Algorithm 6.

Line	Qubit	State
1	$a_{16}$	$a_{16} \oplus a_7$
2	$a_{16}$	$a_{16} \oplus a_7 \oplus a_{31} \oplus a_{22} \oplus a_{14}$
3	$a_{16}$	$a_{16} \oplus a_7 \oplus a_{31} \oplus a_{14} \oplus a_{13} \oplus a_5$
4	$a_{16}$	$a_{16} \oplus a_7 \oplus a_{31} \oplus a_{13} \oplus a_5$
5	$a_{16}$	$a_{16} \oplus a_7 \oplus a_{31} \oplus a_5$
6	$a_{16}$	$a_{16} \oplus a_7 \oplus a_{31}$

used to neutralize the  $a_{22}$  value. Because the XOR operation values of  $a_{22}$ ,  $a_{13}$ , and  $a_5$  are stored in  $a_{22}$ , only the  $a_{22}$  value is obtained by performing  $a_{13}$ ,  $a_5$ , and the CNOT gate in lines 5 and 6.

## 4 Evaluation

The proposed SM3 quantum circuit implementation was evaluated by using quantum emulator, namely, IBM ProjectQ. Among various compilers provided by IBM, the ProjectQ, quantum compiler can estimate the resources of implemented quantum circuits. It measures the number of Toffoli gates, CNOT gates, X gates, and qubits used in a quantum circuit.

We focused on optimizing the quantum gates and qubits for the implementation of the SM3 quantum circuit. One of important elements of a quantum circuit is making it work with minimal resources. Currently, the number of qubits available in quantum computer technology is limited, and it is efficient to reduce the quantum gate cost. Therefore, it can be used as an index to confirm the efficiency of the quantum circuit by comparing the quantum circuit resources of the SM3 quantum circuit proposed in this paper with other hash functions. First, in the proposed SM3 quantum circuits, the number of qubits to be used for message storage was reduced by mixing the expansion function and the compression function. By dividing the expansion function and the compression function into two, the message qubits used in the first compression function can be reused in the second compression function. Second, in the permutation operation, we found the original value with the CNOT gate without allocating a bit to store the original value. In this way, we reduced the number of qubits. Finally, Finally, it is implemented using minimal quantum gates. Based on this optimal

Table 2: Quantum resources required for SHA2 and SHA3 quantum circuits and the proposed SM3 quantum circuit.

Algorithm	Qubits	Toffoli gates	CNOT gates	X gates	Depth
SHA2-256 [14]	2,402	57,184	534,272	–	528,768
SHA3-256 [14]	3,200	84,480	33,269,760	85	10,128
Proposed SM3	2,721	43,328	134,144	2,638	128,129

quantum circuit, we can minimize the quantum resources required for Grover’s search algorithm for the SM3 hash function.

As far as we know, there have been no previous studies that implemented SM3 as a quantum circuit. Therefore, it is difficult to compare the implementation of our SM3 quantum circuit with other SM3 quantum circuits. As an alternative, we compare the quantum resources of SHA2-256 and SHA3-256 with SM3. Table 2 shows the amount of quantum resources used in the proposed SM3 quantum circuit and SHA[14]. SHA is a standard hash function announced by the National Institute of Standards and Technology (NIST) in the US. SHA2-256 and SM3 are calculated through a 512-bit message block and output a 256-bit hash value. SHA3-256 is a sponge structure, it outputs a hash value of 256 bits at all input lengths. When the available resources of a quantum computer reach the resources required for a hash function attack, it can be seen as the time when the security of the hash function can be broken. Therefore, the quantum resources of the optimized quantum circuit are used as an indicator to confirm the safety in the quantum computer. When SM3 was compared with SHA, it used more qubits than SHA2, but much fewer qubits than SHA3. In terms of quantum gates, much fewer quantum gates than SHA2 and SHA3 were used. In SHA2, X gate was not used, but more Toffoli and CNOT gates were used than in SM3. In quantum computers, Toffoli and CNOT gates are more expensive resources than X gates, so it was considered that more quantum gate resources were used.

## 5 Conclusion

In this paper, we implemented and optimized the SM3 hash function as a quantum circuit and estimated the required quantum resources. The quantum resources required for a quantum pre-image attack using Grover’s search algorithm are determined according to the quantum circuit of the target hash function. Utilizing the proposed SM3 quantum circuits, Grover’s search algorithm can be efficiently applied, and its performance was assessed by comparing it in terms of quantum resources with approaches proposed in other research. It is expected that the proposed implementation of the SM3 hash function in quantum circuits can be effectively applied to Grover’s search algorithm.

## 6 Acknowledgment

This work was partly supported by Institute for Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government(MSIT) (<Q|Crypton>, No.2019-0-00033, Study on Quantum Security Evaluation of Cryptography based on Computational Quantum Complexity, 40%) and this work was partly 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, 40%). Zhi Hu was partially supported by the National Natural Science Foundation of China (61972420, 61602526) and the Natural Science Foundation of Hunan Province (2020JJ3050, 2019JJ50827).

## References

1. P. W. Shor, “Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer,” *SIAM J. Comput.*, vol. 26, p. 1484–1509, Oct. 1997.
2. L. K. Grover, “A fast quantum mechanical algorithm for database search,” in *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pp. 212–219, 1996.
3. M. Grassl, B. Langenberg, M. Roetteler, and R. Steinwandt, “Applying Grover’s algorithm to AES: quantum resource estimates,” in *Post-Quantum Cryptography*, pp. 29–43, Springer, 2016.
4. B. Langenberg, H. Pham, and R. Steinwandt, “Reducing the cost of implementing AES as a quantum circuit,” tech. rep., Cryptology ePrint Archive, Report 2019/854, 2019.
5. S. Jaques, M. Naehrig, M. Roetteler, and F. Virdia, “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, 2020.
6. R. Anand, A. Maitra, and S. Mukhopadhyay, “Grover on SIMON,” *Quantum Information Processing*, vol. 19, no. 9, pp. 1–17, 2020.
7. K. Jang, S. Choi, H. Kwon, and H. Seo, “Grover on SPECK: Quantum resource estimates.” Cryptology ePrint Archive, Report 2020/640, 2020. <https://eprint.iacr.org/2020/640>.
8. K. Jang, H. Kim, S. Eum, and H. Seo, “Grover on GIFT.” Cryptology ePrint Archive, Report 2020/1405, 2020. <https://eprint.iacr.org/2020/1405>.
9. L. Schlieper, “In-place implementation of quantum-Gimli,” *arXiv preprint arXiv:2007.06319*, 2020.
10. K. Jang, S. Choi, H. Kwon, H. Kim, J. Park, and H. Seo, “Grover on Korean block ciphers,” *Applied Sciences*, vol. 10, no. 18, p. 6407, 2020.
11. K. Jang, G. Song, H. Kim, H. Kwon, H. Kim, and H. Seo, “Efficient implementation of PRESENT and GIFT on quantum computers,” *Applied Sciences*, vol. 11, no. 11, p. 4776, 2021.
12. G. Song, K. Jang, H. Kim, W.-K. Lee, and H. Seo, “Grover on Caesar and Vigenère ciphers,” *IACR Cryptol. ePrint Arch.*, vol. 2021, p. 554, 2021.
13. K. Jang, G. Song, H. Kwon, S. Uhm, H. Kim, W.-K. Lee, and H. Seo, “Grover on PIPO,” *Electronics*, vol. 10, no. 10, p. 1194, 2021.

14. M. Amy, O. D. Matteo, V. Gheorghiu, M. Mosca, A. Parent, and J. Schanck, “Estimating the cost of generic quantum pre-image attacks on SHA-2 and SHA-3,” 2016.
15. V. Gheorghiu and M. Mosca, “Benchmarking the quantum cryptanalysis of symmetric, public-key and hash-based cryptographic schemes,” 2019.
16. D. S. Steiger, T. Häner, and M. Troyer, “ProjectQ: an open source software framework for quantum computing,” *Quantum*, vol. 2, p. 49, 2018.