


Article

SPEEDY Quantum Circuit for Grover's Algorithm

Gyeongju Song ¹, Kyoungbae Jang ¹, Hyunjun Kim ¹, Siwoo Eum ¹, Minjoo Sim ¹, Hyunji Kim ¹, Waikong Lee ² and Hwajeong Seo ^{1,*} 

¹ Division of IT Convergence Engineering, Hansung University, Seoul 02876, Korea; thdrudwn98@hansung.ac.kr (G.S.); starj1024@hansung.ac.kr (K.J.); 20213201@hansung.ac.kr (H.K.); 21213203@hansung.ac.kr (S.E.); alswntla@hansung.ac.kr (M.S.); 1594012@hansung.ac.kr (H.K.)

² Department of Computer Engineering, Gachon University, Seongnam 13306, Korea; waikonglee@gachon.ac.kr

* Correspondence: hwajeong@hansung.ac.kr; Tel.: +82-760-8033

Abstract: In this paper, we propose a quantum circuit for the SPEEDY block cipher for the first time and estimate its security strength based on the post-quantum security strength presented by NIST. The strength of post-quantum security for symmetric key cryptography is estimated at the cost of the Grover key retrieval algorithm. Grover's algorithm in quantum computers reduces the n -bit security of block ciphers to $\frac{n}{2}$ bits. The implementation of a quantum circuit is required to estimate the Grover's algorithm cost for the target cipher. We estimate the quantum resource required for Grover's algorithm by implementing a quantum circuit for SPEEDY in an optimized way and show that SPEEDY provides either 128-bit security (i.e., NIST security level 1) or 192-bit security (i.e., NIST security level 3) depending on the number of rounds. Based on our estimated cost, increasing the number of rounds is insufficient to satisfy the security against quantum attacks on quantum computers.

Keywords: Grover's algorithm; quantum computing; SPEEDY block cipher; post-quantum



Citation: Song, G.; Jang, K.; Kim, H.; Eum, S.; Sim, M.; Kim, H.; Lee, W.; Seo, H. SPEEDY Quantum Circuit for Grover's Algorithm. *Appl. Sci.* **2022**, *12*, 6870. <https://doi.org/10.3390/app12146870>

Academic Editor: Arcangelo Castiglione

Received: 3 June 2022

Accepted: 5 July 2022

Published: 7 July 2022

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.



Copyright: © 2022 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

With the development of quantum computers, public key cryptography and symmetric key cryptography are vulnerable against quantum algorithms. It is expected that cryptography will no longer be secure when large-scale quantum computers that have reached the quantum resources required for target cryptography attacks are released [1]. Grover's search algorithm is a well-known quantum algorithm that can accelerate the exhaustive key search against symmetric key cryptography [2]. Grover's algorithm can reduce the computational complexity from $O(N)$ to $O(\sqrt{N})$ for symmetric key cryptography using an n -bit key (i.e., $N = 2^n$) in a quantum computer.

The National Institute of Standards and Technology (NIST) held a competition on post-quantum cryptography with the goal of setting standards for post-quantum cryptography to prepare for the post-quantum era, and presented an estimate of the strength of security for symmetric key cryptography [3]. Block ciphers are not guaranteed to be secure in quantum computers by Grover's algorithm as well. In order to evaluate the safety of the target cipher in the post-quantum era, it is necessary to estimate the required quantum resources by implementing the target cipher as a quantum circuit. As a result, NIST presented the cost of key retrieval using Grover's algorithm as an indicator of security strength in the post-quantum era. As a result, estimating the Grover key retrieval cost of symmetric key cryptography is an interesting area of research [4–20].

SPEEDY is a block cipher proposed by CHES'21 [21] that operates with the block size, key length, and the number of rounds as variables. Since SPEEDY targets 6-bit S-boxes and 64-bit CPUs, the least common multiple of 6 and 64 (i.e., 192) is used as the default block size and key length. SPEEDY for a length of 192 bits and rounds of r is called SPEEDY- r -192.

SPEEDY provides 128 bits of security when $r = 6$ and achieves full security of 192 bits at $r = 7$.

In this paper, we implement the SPEEDY block cipher as a quantum circuit and check the post-quantum security strength by estimating the resources to be applied to Grover's algorithm. To the best of our knowledge, this is the first implementation of SPEEDY in quantum circuits. Grover's algorithm operates by repeating oracle and diffusion operations, and the quantum circuit of the target cipher is essential in oracle. We used the proposed quantum circuit to estimate the quantum resources needed for Grover's algorithm. Furthermore, we decomposed the estimated resource into the lower-level *Clifford + T* gate to check the post-quantum security strength. We estimated the quantum resource by increasing r to see how the number of rounds affects the post-quantum security strength. SPEEDY-7-192 provided 192-bit security on classic computers, but showed a security strength of level 1 (i.e., the AES-128 level) on quantum computers. The post-quantum security strength did not increase even after a larger increase in rounds. In other words, estimating the quantum cost according to various rounds, it was confirmed that SPEEDY provided level 1 (AES-128) post-quantum security, and that the increase in rounds did not significantly affect the quantum security strength. Our results show that increasing the number of rounds can improve security for classical computers, but it is not enough for quantum computers. Finally, it was confirmed through comparison with other lightweight ciphers (i.e., LEA, CHAM, HIGHT, and PIPO) that the increase in rounds did not significantly affect the post-quantum security strength and that increasing the key length affects the security strength. We used IBM's ProjectQ platform to implement and simulate quantum circuits.

Contributions of This Paper

- Implementation of the first quantum circuit for SPEEDY block cipher: To the best of our knowledge, this is the first quantum circuit implementation of SPEEDY. In the S-box implementation, an efficient Algebraic Normal Form (ANF) S-box was adopted in terms of quantum resources to reduce quantum resources, and separate quantum resources were not used through logical swap in ShiftColumns and Key Schedule.
- Estimating the cost of Grover key search for SPEEDY: We estimated the Grover algorithm's quantum resource for the SPEEDY block cipher. Estimated quantum resources are decomposed into a lower-level *Clifford + T* gate to lay the foundation for quantum security level analysis. Finally, we confirmed the post-quantum security strength through quantum cost calculation.
- Post-quantum security evaluation and analysis of SPEEDY block cipher: We evaluated post-quantum security for SPEEDY based on the Grover key retrieval cost presented by NIST. Additionally, we noted the change in security strength with increasing rounds of cipher. Based on these attempts, we discussed the differences in cipher security between classical and quantum computers.

2. Related Work

2.1. Quantum Background

A quantum computer uses qubit, similar to the bit used in classic computer operations [22]. While bits have fixed values of 0 and 1, qubit can have values of 0 and 1 at the same time [23]. The calculation can be performed quickly. Due to the nature of these qubits, a 2^n time brute-force attack in a classic computer can be performed only $\lfloor \frac{\pi}{4} 2^{\frac{n}{2}} \rfloor$ times on a quantum computer. In quantum computing, all changes except measurements must be reversible. It is possible to return to the initial value only with the result value without additional information.

2.1.1. Quantum Gates

Quantum gates exploit the quantum entanglement and superposition states of qubits [24,25]. In quantum computing, the state of a qubit is changed with a quantum gate that can perform reversible operations. Figure 1 shows some of the quantum gates.

1. *NOT/X – gate*, $X(x) = \bar{x}$: The X gate inverts the state of a single qubit.
2. *CNOT gate*, $CNOT(x, y) = (x, x \oplus y)$: One of the two input qubits becomes the control qubit, and the other becomes the target qubit. When the control bit is set to one, the state of the target qubit is inverted. If the control qubit x is one, target qubit y is inverted.
3. *Toffoli gate*, $Toffoli(x, y, z) = (x, y, x \cdot y \oplus z)$: Two of the three input qubits become the control qubits, and the other becomes the target qubit. When all control bits are one, the state of the target qubit is inverted. If both control qubits (x and y) are one, the target qubit z is inverted.
4. *SWAP gate*, $SWAP(x, y) = (y, x)$: This changes the state of two qubits.

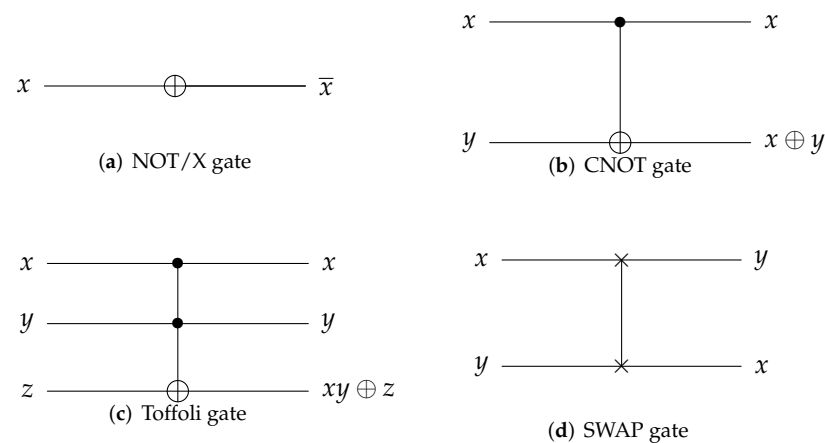


Figure 1. Quantum gates.

2.1.2. Grover's Algorithm for Key Search

An exhaustive key search using Grover's algorithm described in Figure 2 can recover an n -bit key with only $2^{n/2}$ searches. The classical key search performs 2^n searches in the worst case (i.e., $O(2^n)$). Grover key search always repeats $2^{n/2}$. The Grover's algorithm operates with oracle and diffusion operation and increases the probability of finding the correct key through repetition. The oracle in Grover's algorithm finds the correct key. The quantum circuit of the target cipher is used. The diffusion operation in Grover's algorithm operates to increase the probability of measuring the correct key. The following describes the operation process for Grover's algorithm:

1. n -qubits are prepared to find a key of length n .
 $|0\rangle^{\otimes n} = |0\rangle_0 \otimes |0\rangle_1 \otimes \cdots \otimes |0\rangle_n$.
2. All n qubits are placed in a superposition state by a Hadamard gate.
 $|\psi\rangle^{\otimes n} = |\psi\rangle_0 \otimes |\psi\rangle_1 \otimes \cdots \otimes |\psi\rangle_n$.
3. If the ciphertext generated by the input n -qubits (i.e., key) matches the known ciphertext, the sign of the key in that state is inverted.
4. The amplitude of the solution key is amplified through the diffusion operator.
5. Steps 3 and 4 are repeated $\sqrt{2^n}$ times to increase the key search probability.

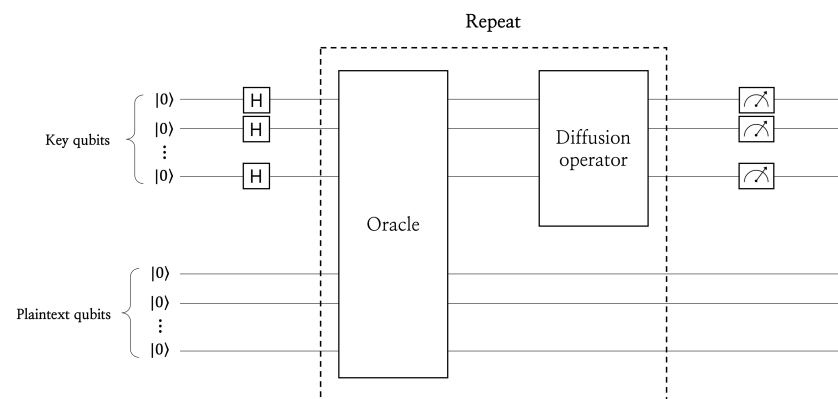


Figure 2. Exhaustive key search in Grover's algorithm.

2.1.3. Grover's Algorithm for SPEEDY

The oracle in Grover's algorithm performs a known plaintext attack (KPA), and the attack is possible when it knows one plaintext–ciphertext pair. Figure 3 shows the oracle in Grover's algorithm. To perform KPA in oracle, a quantum circuit for the target cipher is required, and our SPEEDY quantum circuit performs encryption within oracle. For the SPEEDY quantum circuit, it performs encryption using a superposition key and finds the key value when the encryption result is the same as the known ciphertext. When these conditions are satisfied, the key is correct. In the SPEEDY block cipher, a known plaintext of length $6l$ and a 192-bit superposition key is input to perform the encryption function Enc . After storing the encryption result for the SPEEDY quantum circuit in the plaintext qubit, it is compared with the known ciphertext to find the correct key. Thus, this increases the observation probability of the correct key through the diffusion operation. Since the Grover's algorithm repeats this operation, the encrypted plaintext state is returned to the previous state through the decryption (i.e., inverse) function Enc^\dagger .

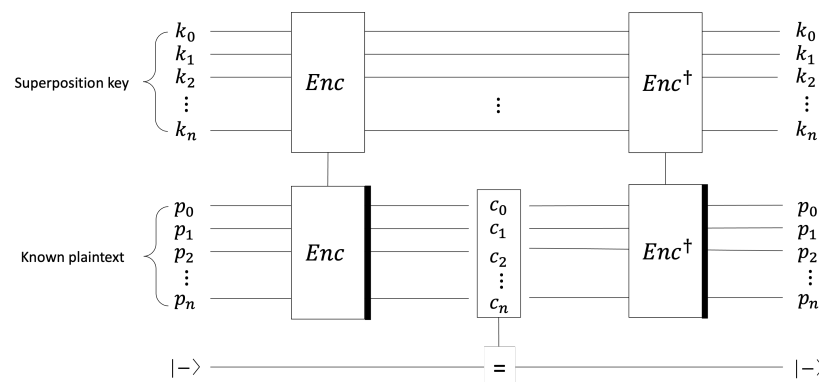


Figure 3. Oracle in Grover's algorithm.

2.2. SPEEDY: Family of Block Ciphers

The SPEEDY block cipher is a family of ultra-low-latency block ciphers proposed at the CHES'21 [21]. It can use different block sizes and key lengths, and the number of rounds determines the level of security. SPEEDY is also an ultra-low-latency block cipher suite dedicated to the design of integrated circuits based on standard cells developed for very high execution speeds in CMOS hardware. SPEEDY aims to be a secure architecture for CPUs that require very-low-latency encryption, such as secure cache, dedicated hardware expansion, memory encryption, and pointer authentication. SPEEDY is noted as SPEEDY- $r-6l$ for block size $6 \times l$ and number of rounds r . The internal state is represented as a $l \times 6$ array. Since the SPEEDY block cipher targets 6-bit S-boxes and 64-bit high-end CPUs, it uses the least common multiple of 6 and 64 (i.e., SPEEDY- $r-192$) as a default

block size and key length. Therefore, in this paper, SPEEDY is described based on the SPEEDY- r -192 representation. The operations in SPEEDY- r -192 work on a 32×6 array. In SPEEDY, it works with functions such as round function (R), S-box (SB), ShiftColumns (SC), MixColumns (MC), AddRoundKey (A_{k_r}), and AddRoundConstant (A_{c_r}). Each function of SPEEDY operates in the following order, except for the last round: A_{k_r} , SB, SC, MC, A_{c_r} , KeySchedule. The last round is an exception and operates in the following order: A_{k_r} , SB, SC, SB, KeySchedule, A_{k_r} .

2.2.1. S-Box (SB)

The S-box in the SPEEDY block cipher is a 6-to-6-bit box with a 6-bit output (y_0 to y_5) for a 6-bit input (x_0 to x_5). It operates as a combination of NOT gate and NAND gate, as shown in Equation (1).

$$\begin{aligned} y_0 &= (x_3 \wedge \bar{x}_5) \vee (x_3 \wedge x_4 \wedge x_2) \vee (\bar{x}_3 \wedge x_1 \wedge x_0) \vee (x_5 \wedge x_4 \wedge x_1) \\ y_1 &= (x_5 \wedge x_3 \wedge \bar{x}_2) \vee (\bar{x}_5 \wedge x_3 \wedge \bar{x}_4) \vee (x_5 \wedge x_2 \wedge x_0) \vee (\bar{x}_3 \wedge \bar{x}_0 \wedge x_1) \\ y_2 &= (\bar{x}_3 \wedge x_0 \wedge x_4) \vee (x_3 \wedge x_0 \wedge x_1) \vee (\bar{x}_3 \wedge \bar{x}_4 \wedge x_2) \vee (\bar{x}_0 \wedge \bar{x}_2 \wedge \bar{x}_5) \\ y_3 &= (\bar{x}_0 \wedge x_2 \wedge \bar{x}_3) \vee (x_0 \wedge x_2 \wedge x_4) \vee (x_0 \wedge \bar{x}_2 \wedge x_5) \vee (\bar{x}_0 \wedge x_3 \wedge x_1) \\ y_4 &= (x_0 \wedge \bar{x}_3) \vee (x_0 \wedge \bar{x}_4 \wedge \bar{x}_2) \vee (\bar{x}_0 \wedge x_4 \wedge x_5) \vee (\bar{x}_4 \wedge \bar{x}_2 \wedge x_1) \\ y_5 &= (x_2 \wedge x_5) \vee (\bar{x}_2 \wedge \bar{x}_1 \wedge x_4) \vee (x_2 \wedge x_1 \wedge x_0) \vee (\bar{x}_1 \wedge x_0 \wedge x_3) \end{aligned} \quad (1)$$

2.2.2. ShiftColumns (SC)

In ShiftColumns(SC), the j -th column of the state is rotated upside by j bits. The process is shown in Equation (2).

$$y_{[i,j]} = x_{[i+j,j]} \quad (0 \leq i < l, 0 \leq j < 6) \quad (2)$$

2.2.3. MixColumns (MC)

MixColumns performs a CNOT operation with a shift in a column. The shift follows the order of the given constant $\alpha = [\alpha_0, \alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_5]$. In Equation (3) of MixColumn, i and j are rows and columns.

$$y_{i,j} = x_{[i,j]} \oplus x_{[i+\alpha_1,j]} \oplus x_{[i+\alpha_2,j]} \oplus x_{[i+\alpha_3,j]} \oplus x_{[i+\alpha_4,j]} \oplus x_{[i+\alpha_5,j]} \oplus x_{[i+\alpha_6,j]} \quad 0 \leq i < l, 0 \leq j < 6 \quad (3)$$

2.2.4. AddRoundKey (A_{k_r})

The length of the key k_r is equal to the length of $6 \cdot l$, and k_r performs an XOR operation with x on the same bit position. The AddRoundKey (A_{k_r}) operation is as follows:

$$y_{i,j} = x_{[i,j]} \oplus k_{r[i,j]}, \quad \forall i, j \quad (4)$$

2.2.5. AddRoundConstant (A_{c_r})

The constant c_r of $6l$ bits operates XOR with x on the same bit position. The round constants are chosen as the binary number of $\pi - 3 = 0.1415 \dots$. The AddRoundConstant (A_{c_r}) operation is as follows:

$$y_{i,j} = x_{[i,j]} \oplus c_{r[i,j]} \quad \forall i, j \quad (5)$$

2.2.6. KeySchedule

In KeySchedule, the 0-th round key k_0 is initialized to a specific value. Then, r round key k_r is computed as in Equation (6). The k_r uses the permutation P to change the bit position.

$$\begin{aligned} k_{r+1}[i',j'] &= k_r[i,j] \\ (i',j') &:= P(i,j) \quad \text{with} \quad (6 \cdot i' + j') \equiv (\beta \cdot (6 \cdot i + j) + \gamma) \bmod 6 \end{aligned} \quad (6)$$

2.2.7. Round Function

The SPEEDY block cipher repeats the round to proceed with encryption. In r -round encryption, operations are performed in the same way from 0 to $r - 1$. In the last round, MixColumn (MC) and ShiftColumn (SC) are performed once each. The operation of the round function R follows Equation (7):

$$R_n = \begin{cases} A_{c_n} \circ MC \circ SC \circ SB \circ A_{k_n} & (0 < n < r - 2) \\ A_{k_{n+1}} \circ SB \circ SC \circ SB \circ A_{k_n} & (n = r - 1) \end{cases} \quad (7)$$

3. Quantum Circuit for SPEEDY

In this section, we describe our proposed SPEEDY quantum circuit. The quantum circuit is designed based on SPEEDY-7-192. It is used to estimate the resources required for Grover's algorithm. The overall quantum circuit operation sequence is shown in Figure 4. As shown in Figure 5, a 32×6 array (i.e., $x_{[i][j]}$, $0 \leq i < 6$, $0 \leq j < 32$) in a classical computer is implemented by a 1×192 array (i.e., $x_{[i]}$, $0 \leq i < 192$) in a quantum computer. We note the quantum circuits for the main algorithms of SPEEDY: S-box (SB), ShiftColumns (SC), MixColumns (MC), AddRoundKey (A_{k_r}), and AddRoundConstant (A_{c_r}). The SPEEDY quantum circuit operates for rounds 0 to $r - 1$, and the operation is different only in the last round. In the quantum circuit, rounds 0 to $r - 2$ operate in the following order: AddRoundKey, S-box, ShiftColumn, MixColumn, AddRoundConstant, KeySchedule. The last round $r - 1$ operates in the order of AddRoundKey, S-box, ShiftColumn, S-box, KeySchedule, AddRoundKey.

KeySchedule \circ AddRoundConstant \circ MixColumn \circ ShiftColumn \circ S-box \circ AddRoundKey $\quad (0 \leq R \leq r - 2)$

AddRoundKey \circ KeySchedule \circ S-box \circ ShiftColumn \circ S-box \circ AddRoundKey $\quad (R = r - 1)$

Figure 4. Operation sequence for SPEEDY quantum circuit.

The SPEEDY quantum circuit uses the quantum gates described in Section 2.2.1 and additionally uses a multi-controlled X gate. The multi-controlled X gates used in the SPEEDY quantum circuit are represented as follows:

- $CCCX(x_0, x_1, x_2, y_0) = (x_0, x_1, x_2, (x_0 \cdot x_1 \cdot x_2) \oplus y_0)$: x_0, x_1 , and x_2 are the control qubits and y_0 is a target qubit. When all control qubits are 1, the X gate is used to y_0 .
- $CCCCX(x_0, x_1, x_2, x_3, y_0) = (x_0, x_1, x_2, x_3, (x_0 \cdot x_1 \cdot x_2 \cdot x_3) \oplus y_0)$: x_0, x_1, x_2 , and x_3 are the control qubits and y_0 is a target qubit. When all control qubits are 1, the X gate is used to y_0 .
- $CCCCCX(x_0, x_1, x_2, x_3, x_4, y_0) = (x_0, x_1, x_2, x_3, (x_0 \cdot x_1 \cdot x_2 \cdot x_3 \cdot x_4) \oplus y_0)$: x_0, x_1, x_2, x_3 , and x_4 are the control qubits and y_0 is a target qubit. When all control qubits are 1, the NOT gate is used to y_0 .

$x_{[0,0]}$	$x_{[0,1]}$	$x_{[0,2]}$	$x_{[0,3]}$	$x_{[0,4]}$	$x_{[0,5]}$
$x_{[1,0]}$	$x_{[1,1]}$	$x_{[1,2]}$	$x_{[1,3]}$	$x_{[1,4]}$	$x_{[1,5]}$
$x_{[2,0]}$	$x_{[2,1]}$	$x_{[2,2]}$	$x_{[2,3]}$	$x_{[2,4]}$	$x_{[2,5]}$
\vdots	\vdots	\vdots	\vdots	\vdots	\vdots
$x_{[31,0]}$	$x_{[31,1]}$	$x_{[31,2]}$	$x_{[31,3]}$	$x_{[31,4]}$	$x_{[31,5]}$

Bit array (32x6)

$x_{[0]}$	$x_{[1]}$	$x_{[2]}$	$x_{[3]}$	$x_{[4]}$	$x_{[5]}$
$x_{[6]}$	$x_{[7]}$	$x_{[8]}$	$x_{[9]}$	$x_{[10]}$	$x_{[11]}$
$x_{[12]}$	$x_{[13]}$	$x_{[14]}$	$x_{[15]}$	$x_{[16]}$	$x_{[17]}$
\vdots	\vdots	\vdots	\vdots	\vdots	\vdots
$x_{[186]}$	$x_{[187]}$	$x_{[188]}$	$x_{[189]}$	$x_{[190]}$	$x_{[191]}$

Qubit array (1x192)

Figure 5. Bit array on classical computer and qubit array on quantum computer.

3.1. S-Box (SB)

The SPEEDY S-box uses NAND and OAI gates best suited for ultra-low latency. Therefore, the operation of the S-box follows Equation (1), expressed in disjunctive normal form (DNF). However, DNF is inefficient in terms of resources in the quantum circuit. In quantum circuits, NAND and OAI operations must allocate as many qubits as the number of operations to store intermediate values. To solve this problem, we reduced the quantum resources by using Algebraic Normal Form (ANF), which is performed as XOR gates. ANF is expressed using a combination of XOR and AND. The equation of the S-box expressed as ANF can be found in detail in [21]. Algorithm 1 shows our S-box quantum circuit implemented using CNOT and multi-controlled X gates. Furthermore, we have schematically shown the operation of Algorithm 1 as a quantum circuit in Figure 6. Here, we reduce the quantum resource by omitting the extra qubits for intermediate values. Since the SPEEDY S-box uses a lot of multi-controlled X gates, the gate cost is the highest part of the overall operation. In the S-box, the results of inputs x_0 to x_5 are output in ancilla y_0 to y_5 . At the input ancilla, qubit y should initially be set to zero, and at the end of the circuit, it stores the 6-bit result of the S-box. Input x is the result of ShiftColumn and is the target of the S-box operation. That is, the S-box execution result of x is stored in y .

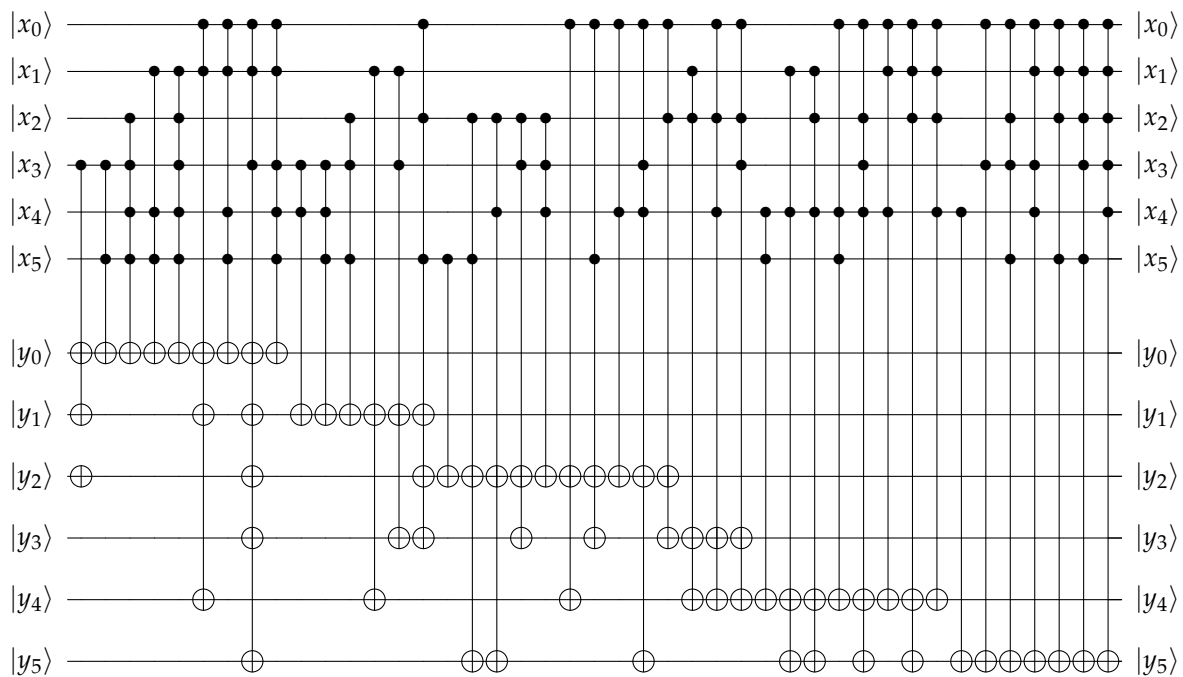


Figure 6. Quantum circuit for S-box.

Algorithm 1 Quantum circuit of S-box (SB)

Input: $x_0, x_1, x_2, x_3, x_4, x_5$ **Output:** $y_0, y_1, y_2, y_3, y_4, y_5$

```

1:  $y_0 \leftarrow \text{CNOT}(x_3, y_0)$ 
2:   Toffoli( $x_5, x_3, y_0$ )
3:   CCCCX( $x_5, x_4, x_3, x_2, y_0$ )
4:   CCCX( $x_5, x_4, x_1, y_0$ )
5:   CCCCCX( $x_5, x_4, x_3, x_2, x_1, y_0$ )
6:   Toffoli( $x_1, x_0, y_0$ )
7:   CCCCCX( $x_5, x_4, x_1, x_0, y_0$ )
8:   CCCX( $x_3, x_1, x_0, y_0$ )
9:   CCCCCX( $x_5, x_4, x_3, x_1, x_0, y_0$ )

10:  $y_1 \leftarrow \text{CNOT}(x_3, y_1)$ 
11:   Toffoli( $x_4, x_3, y_1$ )
12:   CCCX( $x_5, x_4, x_3, y_1$ )
13:   CCCX( $x_5, x_3, x_2, y_1$ )
14:   CNOT( $x_1, y_1$ )
15:   Toffoli( $x_3, x_1, y_1$ )
16:   CCCX( $x_5, x_2, x_0, y_1$ )
17:   Toffoli( $x_1, x_0, y_1$ )
18:   CCCX( $x_3, x_1, x_0, y_1$ )

19:  $y_2 \leftarrow \text{NOT}(y_2)$ 
20:   CNOT( $x_5, y_2$ )
21:   Toffoli( $x_5, x_2, y_2$ )
22:   Toffoli( $x_4, x_2, y_2$ )
23:   Toffoli( $x_3, x_2, y_2$ )
24:   CCCX( $x_4, x_3, x_2, y_2$ )
25:   CNOT( $x_0, y_2$ )
26:   Toffoli( $x_5, x_0, y_2$ )
27:   Toffoli( $x_4, x_0, y_2$ )
28:   CCCX( $x_4, x_3, x_0, y_2$ )
29:   Toffoli( $x_2, x_0, y_2$ )
30:   CCCX( $x_5, x_2, x_0, y_2$ )
31:   CCCX( $x_3, x_1, x_0, y_2$ )

32:  $y_3 \leftarrow \text{CNOT}(x_2, y_3)$ 
33:   Toffoli( $x_3, x_2, y_3$ )
34:   Toffoli( $x_3, x_1, y_3$ )

35:   Toffoli( $x_5, x_0, y_3$ )
36:   Toffoli( $x_2, x_0, y_3$ )
37:   CCCX( $x_5, x_2, x_0, y_3$ )
38:   CCCX( $x_4, x_2, x_0, y_3$ )
39:   CCCX( $x_3, x_2, x_0, y_3$ )
40:   CCCX( $x_3, x_1, x_0, y_3$ )

41:  $y_4 \leftarrow \text{Toffoli}(x_5, x_4, y_4)$ 
42:   CNOT( $x_1, y_4$ )
43:   Toffoli( $x_4, x_1, y_4$ )
44:   Toffoli( $x_2, x_1, y_4$ )
45:   CCCX( $x_4, x_2, x_1, y_4$ )
46:   CNOT( $x_0, y_4$ )
47:   CCCX( $x_5, x_4, x_0, y_4$ )
48:   CCCX( $x_4, x_3, x_0, y_4$ )
49:   CCCX( $x_3, x_2, x_0, y_4$ )
50:   CCCCCX( $x_4, x_3, x_2, x_0, y_4$ )
51:   Toffoli( $x_1, x_0, y_4$ )
52:   CCCX( $x_4, x_1, x_0, y_4$ )
53:   CCCX( $x_2, x_1, x_0, y_4$ )
54:   CCCCCX( $x_4, x_2, x_1, x_0, y_4$ )

55:  $y_5 \leftarrow \text{CNOT}(x_4, y_5)$ 
56:   Toffoli( $x_5, x_2, y_5$ )
57:   Toffoli( $x_4, x_2, y_5$ )
58:   Toffoli( $x_4, x_1, y_5$ )
59:   CCCX( $x_4, x_2, x_1, y_5$ )
60:   Toffoli( $x_3, x_0, y_5$ )
61:   CCCX( $x_4, x_3, x_0, y_5$ )
62:   CCCCCX( $x_5, x_3, x_2, x_0, y_5$ )
63:   CCCCCX( $x_4, x_3, x_2, x_0, y_5$ )
64:   CCCX( $x_3, x_1, x_0, y_5$ )
65:   CCCCCX( $x_4, x_3, x_1, x_0, y_5$ )
66:   CCCX( $x_2, x_1, x_0, y_5$ )
67:   CCCCCX( $x_5, x_2, x_1, x_0, y_5$ )
68:   CCCCCX( $x_5, x_3, x_2, x_1, x_0, y_5$ )
69:   CCCCCX( $x_4, x_3, x_2, x_1, x_0, y_5$ )
70: return  $y_0, \dots, y_5$ 

```

3.2. ShiftColumns (SC)

ShiftColumns in the quantum circuit perform column shifts. It is implemented assuming that 6 qubits are arranged in 32 rows for a 1×192 qubit array. Assuming that the qubits are arranged as in Figure 7, each column of qubits shifts in the order $\delta = 0, 1, 2, 3, 4, 5$. That is, a shift of 1 in a column is a shift of index 6 in a qubit array. Here, we used logical swap to rotate the columns. In quantum circuits, swap gate only changes the position of the qubit. As a result, we do not use additional quantum resources in ShiftColumn (SC). Algorithm 2 shows the quantum circuit operation of ShiftColumn. It rearranges the input into *new_array* according to the operation. Then, it changes the index of the input as arranged in *new_array*.

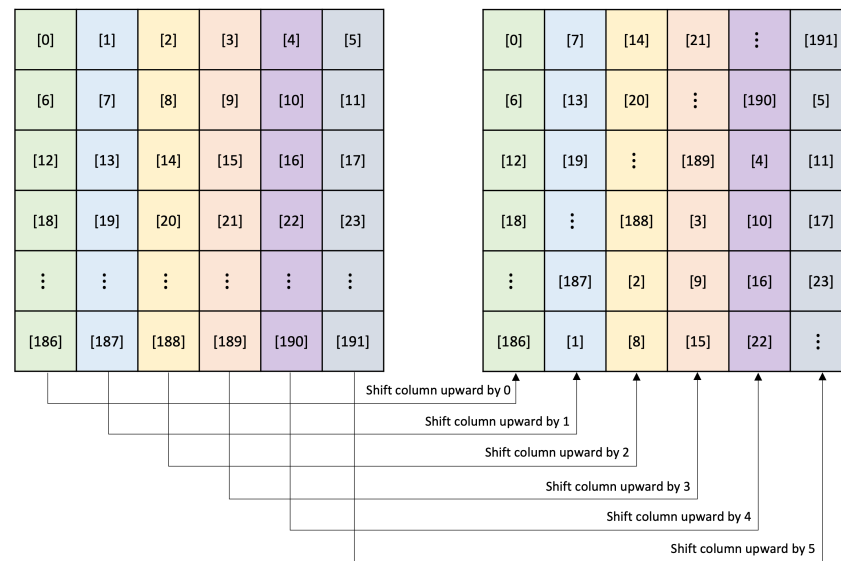


Figure 7. ShiftColumns (SC) operation process.

Algorithm 2 Quantum circuit of ShiftColumns (SC)

Input: 192-qubit array = $[x_0, x_1, \dots, x_{192}]$

Output: 192-qubit array = $[x_0, x_7, x_{14}, \dots, x_{29}]$

```

1: new_array = [ ]
2: for  $i = 0$  to 31 do
3:   for  $j = 0$  to 5 do
4:     new_array [ $6 \cdot i + j$ ]  $\leftarrow x_{(6 \cdot (i+j) + j)}$ 
5:   end for
6: end for
7: return new_array

```

3.3. MixColumns (MC)

MixColumns repeats the XOR operation by shifting the index of the qubits. The MixColumn quantum circuit works with Algorithm 3. In Algorithm 3, the result is stored in the input qubit x_k , and $temp_k$ is used as the temporary storage qubit. First, we use the CNOT gate to store the original x in $temp_k$. Then, the CNOT gate for $temp$ and x is performed during the shift in the index of $temp$. The standard of shift follows the order of

α ($\alpha = 1, 5, 9, 15, 21, 26$). The operation is stored in x , so no additional qubits are needed to store the result.

Algorithm 3 Quantum circuit of MixColumns (MC)

Input: $x_k, temp_k$ ($k = 0, \dots, 191$), $\alpha = [1, 5, 9, 15, 21, 26]$

Output: x_k ($k = 0, \dots, 191$)

```

1: for  $i = 0$  to 191 do
2:    $temp_i \leftarrow \text{CNOT}(x_i, temp_i)$  // copy target qubit( $x$ ) to temporary qubit( $temp$ ).
3: end for

4: for  $i = 0$  to 31 do
5:   for  $j = 0$  to 5 do
6:      $x_{6 \cdot i + j} \leftarrow \text{CNOT}(temp_{6 \cdot (i + \alpha[0]) + j}, x_{6 \cdot i + j})$  //  $x_{6 \cdot i + j} = x_{6 \cdot i + j} \oplus x_{6 \cdot (i + \alpha[0]) + j}$ 
7:      $x_{6 \cdot i + j} \leftarrow \text{CNOT}(temp_{6 \cdot (i + \alpha[1]) + j}, x_{6 \cdot i + j})$  //  $x_{6 \cdot i + j} = \text{line } 6 \oplus x_{6 \cdot (i + \alpha[1]) + j}$ 
8:      $x_{6 \cdot i + j} \leftarrow \text{CNOT}(temp_{6 \cdot (i + \alpha[2]) + j}, x_{6 \cdot i + j})$  //  $x_{6 \cdot i + j} = \text{line } 7 \oplus x_{6 \cdot (i + \alpha[2]) + j}$ 
9:      $x_{6 \cdot i + j} \leftarrow \text{CNOT}(temp_{6 \cdot (i + \alpha[3]) + j}, x_{6 \cdot i + j})$  //  $x_{6 \cdot i + j} = \text{line } 8 \oplus x_{6 \cdot (i + \alpha[3]) + j}$ 
10:     $x_{6 \cdot i + j} \leftarrow \text{CNOT}(temp_{6 \cdot (i + \alpha[4]) + j}, x_{6 \cdot i + j})$  //  $x_{6 \cdot i + j} = \text{line } 9 \oplus x_{6 \cdot (i + \alpha[4]) + j}$ 
11:     $x_{6 \cdot i + j} \leftarrow \text{CNOT}(temp_{6 \cdot (i + \alpha[5]) + j}, x_{6 \cdot i + j})$  //  $x_{6 \cdot i + j} = \text{line } 10 \oplus x_{6 \cdot (i + \alpha[5]) + j}$ 
12:   end for
13: end for

14: return  $x_0, \dots, x_{191}$ 

```

3.4. AddRoundKey (A_{k_r})

AddRoundKey(A_{k_r}) in the quantum circuit is assigned a qubit k (i.e., key) of length equal to the input length. In qubit k , the key value is stored in advance. The input qubit x operates the CNOT gate with k of the same index. We performed the XOR operation according to Equation (4). Since the constant is already known, there is no need to allocate qubits for it.

3.5. AddRoundConstant (A_{c_r})

In AddRoundConstant (A_{c_r}), XOR uses the input x and a constant value. Since the constant is already known, there is no need to allocate qubits for it. Therefore, the X gate shifts to x at the position where the constant value is one, without using the CNOT gate. An X gate operating with a single qubit has a lower gate cost than a CNOT gate operating with two qubits. Therefore, our choice is efficient in terms of quantum resources, saving the gate cost. The X gate is used only where the value in the constant is one, so the X gate is used as much as the Hamming weight. Algorithm 4 shows the operation for AddRoundConstant (A_{c_r}).

Algorithm 4 Quantum circuit of AddRoundConstant (A_{c_r})**Input:** $constant, x_0, \dots, x_{191}$ **Output:** x_0, \dots, x_{191}

```

1: for  $i = 0$  to 191 do
2:   if ( $constant \& 1 = 1$ )
3:      $x_i \leftarrow X(x_i)$ 
4:    $constant = constant \gg 1$ 
5: end for
6: return  $x_0, \dots, x_{191}$ 

```

4. Evaluation

In this section, we estimate the resources of Grover's algorithm for a SPEEDY block cipher implemented as a quantum circuit. Resources estimated by the quantum simulator are used to evaluate the security strength in the quantum computer. The cost is calculated as (total gates \times total depth) and the total gate is the sum of T and *Clifford* gate. We decompose the non-Clifford gates into $T + Clifford$ gates to obtain the total gate [26]. Finally, we show that SPEEDY-7-192, which achieved a security strength of 192 length in the classic computer, does not achieve a security strength in the quantum computer. The block cipher security strength is evaluated based on the estimate of the post-quantum security strength presented by NIST [3].

4.1. Resource Estimation

Table 1 shows the quantum circuit resource estimation results for SPEEDY encryption. We estimated resources for rounds 6, 14, and 28 other than round 7 to evaluate the strength of security for each round. Based on SPEEDY-7-192, 4224 qubits, 1792 CCCCX gates, 3584 CCCCX gates, 10752 CCCX gates, 10,304 Toffoli gates, 13,632 CNOT gates, and 2118 X gates were used. The estimated quantum resource is proportional to the number of rounds.

Table 1. Quantum resources for SPEEDY.

Cipher	r	Qubits	Gates						Depth
			CCCCCX	CCCCX	CCCX	Toffoli	CNOT	X	
SPEEDY- r -192	6	3648	1536	3072	9216	8832	11,520	855	859
	7	4224	1792	3584	10,752	10,304	13,632	1018	1002
	14	8256	3584	7168	21,504	20,608	28,416	2118	2011
	28	16,320	7168	14,336	43,008	41,216	57,984	4346	4029

4.2. Security Strength Analysis for SPEEDY

The post-quantum security strength for SPEEDY is evaluated based on the security strength category presented by NIST [3]. We calculate the cost with the same calculation as Grassl et al. [4]. That is, the cost is calculated to (total gate \times total depth). In NIST, the cost of AES-128, 196, and 256, which are security strength standards, was estimated as AES-128: 2^{170} , AES-196: 2^{233} , AES-256: 2^{298} . The following are security strength categories presented by NIST based on AES-128, 196, 256:

- Level 1: Block ciphers using a 128-bit key (e.g., AES 128) require computational resources that are greater than or comparable to those required for key search.
- Level 3: Block ciphers using a 192-bit key (e.g., AES 192) require computational resources that are greater than or comparable to those required for key search.
- Level 5: Block ciphers using a 256-bit key (e.g., AES 256) require computational resources that are greater than or comparable to those required for key search.

Grover's algorithm can reduce the computational complexity from $O(N)$ to $O(\sqrt{N})$ for symmetric key cryptography with an n -bit key (i.e., $N = 2^n$). This search algorithm increases the probability of finding the right key by repeating oracle and diffusion, and the quantum circuit is used in the oracle operation for encryption and decryption. Since the quantum circuit is a reversible circuit, decryption can be performed by reverse operation (i.e., encryption resource = decryption resource). Therefore, the total quantum resource used in the Grover's algorithm is calculated as $\lceil \text{key size/block size} \rceil \times (\text{Encryption} + \text{Decryption}) \times \text{number of iterations (i.e., } R \times 2 \times \text{Table 1} \times \lfloor \frac{\pi}{4} 2^{\frac{n}{2}} \rfloor)$.

The Grover's algorithm resource for SPEEDY is shown in Table 2. We compute the Grover's key search cost by decomposing the non-*Clifford* gate into the T + *Clifford* gate from the estimated resource. Since the X gate and CNOT gate are the *Clifford* gates, only the Toffoli gate and multi-controlled X gate are decomposed into T + *Clifford* gates. One Toffoli gate is decomposed into 7 T gates and 8 *Clifford* gates, and multi-controlled X gates are decomposed into $(32 \times C - 84)$ T gates (C : number of control qubits).

Table 2. Cost estimation for Grover's algorithm.

Cipher	r	Gates		Total Gates	Total Depth	Cost	Security
		T	<i>Clifford</i>				
SPEEDY- r -192	6	1.27×2^{115}	1.99×2^{112}	1.51×2^{115}	1.31×2^{106}	1.97×2^{221}	Level 1
	7	1.48×2^{115}	1.16×2^{113}	1.77×2^{115}	1.53×2^{106}	1.38×2^{222}	Level 1
	14	1.48×2^{116}	1.16×2^{114}	1.77×2^{116}	1.54×2^{107}	1.36×2^{224}	Level 1
	28	1.48×2^{117}	1.17×2^{115}	1.77×2^{117}	1.54×2^{108}	1.36×2^{226}	Level 1
LEA-128/128 [27]	-	1.13×2^{81}	1.92×2^{81}	1.195×2^{82}	1.247×2^{77}	1.491×2^{159}	Not achieved
LEA-128/192 [27]	-	1.67×2^{115}	1.42×2^{115}	1.775×2^{115}	1.455×2^{109}	1.292×2^{225}	Level 1
LEA-128/256 [27]	-	1.91×2^{147}	1.63×2^{148}	1.014×2^{148}	1.645×2^{141}	1.668×2^{289}	Level 3
CHAM-64/128 [27]	-	1.05×2^{81}	1.04×2^{82}	1.23×2^{81}	1.003×2^{76}	1.234×2^{157}	Not achieved
CHAM-128/128 [27]	-	1.10×2^{80}	1.11×2^{81}	1.304×2^{81}	1.018×2^{77}	1.328×2^{158}	Not achieved
CHAM-128/256 [27]	-	1.31×2^{146}	1.34×2^{146}	1.566×2^{146}	1.264×2^{146}	1.98×2^{287}	Level 3
HIGHT-64/128 [27]	-	1.05×2^{81}	1.04×2^{82}	1.384×2^{82}	1.901×2^{75}	1.316×2^{158}	Not achieved
PIPIO-64/128 [8]	-	1.68×2^{78}	1.31×2^{79}	1.07×2^{80}	1.52×2^{73}	1.62×2^{153}	Not achieved
PIPIO-64/256 [8]	-	1.09×2^{144}	1.72×2^{144}	1.40×2^{145}	1.99×2^{138}	1.39×2^{284}	Level 3

In a classic computer, SPEEDY- r -192 provides 128-bit security when $r = 6$ and 192-bit security when $r = 7$. However, both SPEEDY-6-192 and SPEEDY-7-192 provided 128-bit security in a quantum computer. In response, we performed encryption with more rounds r to check the strength of security in quantum computers. However, even if the number of rounds of r was increased as in Table 2, security was maintained at level 1. Contrary to expectations, increasing the number of rounds did not provide higher security. In other words, it can be confirmed that it is difficult to increase the security strength in quantum computers by increasing the number of rounds of encryption. It is also very inefficient because the number of rounds r must increase exponentially to enhance security in quantum computers. Simply put, the classic method of increasing security by increasing the number of rounds does not apply to quantum computers.

On the other hand, looking at the LEA, CHAM, and PIPO ciphers evaluated by [8,27] in Table 2, security strength was not achieved with a 64-bit key, but was achieved with a 256-bit key length. HIGHT ciphers that only work with 64-bit keys do not achieve security strength. For the LEA cipher, LEA-128/128 (using 128-bit key) did not achieve the security strength, but LEA-128/192 (using 192-bit key) and LEA-128/256 (using 256-bit key) achieved level 1 and level 3, respectively. In the case of the CHAM cipher, CHAM-64/128 and CHAM-128/128 (using 128-bit key) did not achieve the security strength, but CHAM-64/256 (using 256-bit key) achieved level 3. In the case of the PIPO cipher, PIPO-64/128 (using 128-bit key) did not achieve the security strength, but PIPO-64/256 (using 256-bit key) achieved level 3.

From the above results, it was confirmed that it is difficult to increase the quantum security strength by increasing the number of rounds and the block length, but it can be increased through the key length. Therefore, in order to strengthen the security in quantum computers, it is necessary to consider measures to increase the number of iterations exponentially by increasing the key length.

5. Conclusions

In this paper, a quantum circuit for the SPEEDY block cipher is presented. We estimated the resources required to perform a key search attack based on SPEEDY- r -192 and obtained the cost required to evaluate the security strength. As a result, SPEEDY-7-192 provided 192-bit security in classic computers, but showed a security strength of level 1 (i.e., AES-128 level) in quantum computers. In other words, encryption that is secure in classic computers cannot be considered secure in quantum computers. We increased the number of rounds as a way to strengthen the security in the quantum computer, but it did not increase the security strength significantly. Based on the results in this paper, we propose a method to increase the key length to ensure the security of the target cipher (SPEEDY in this paper) in a quantum computer.

Author Contributions: Formal analysis, H.K. (Hyunjun Kim), K.J. and G.S.; Investigation, G.S.; Software, S.E. and G.S.; Writing—original draft, G.S.; Writing—review and editing, G.S., M.S., H.K. (Hyunji Kim), W.L. and H.S. All authors have read and agreed to the published version of the manuscript.

Funding: 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, 50%) 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, 40%). This research was financially supported by Hansung University for Hwajeong Seo.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Mosca, M. Cybersecurity in an era with quantum computers: Will we be ready? *IEEE Secur. Priv.* **2018**, *16*, 38–41. [\[CrossRef\]](#)
2. Grover, L.K. A fast quantum mechanical algorithm for database search. In Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing, Philadelphia, PA, USA, 22–24 May 1996; pp. 212–219.
3. NIST. *Submission Requirements and Evaluation Criteria for the Post-Quantum Cryptography Standardization Process*; NIST: Gaithersburg, MD, USA, 2016.
4. Grassl, M.; Langenberg, B.; Roetteler, M.; Steinwandt, R. Applying Grover’s algorithm to AES: Quantum resource estimates. In *Post-Quantum Cryptography*; Springer: Berlin/Heidelberg, Germany, 2016; pp. 29–43.
5. Langenberg, B.; Pham, H.; Steinwandt, R. Reducing the Cost of Implementing the Advanced Encryption Standard as a Quantum Circuit. *IEEE Trans. Quantum Eng.* **2020**, *1*, 1–12. doi: 10.1109/TQE.2020.2965697. [\[CrossRef\]](#)
6. Jang, K.; Choi, S.; Kwon, H.; Seo, H. Grover on SPECK: Quantum Resource Estimates. *IACR Cryptol. ePrint Arch.* **2020**, *2020*, 640.
7. Chauhan, A.; Sanadhya, S. Quantum Resource Estimates of Grover’s Key Search on ARIA. In Proceedings of the International Conference on Security, Privacy, and Applied Cryptography Engineering, Kolkata, India, 17–21 December 2020; Springer: Cham, Switzerland, 2020; pp. 238–258. [\[CrossRef\]](#)

8. Jang, K.; Song, G.; Kwon, H.; Uhm, S.; Kim, H.; Lee, W.K.; Seo, H. Grover on PIPO. *Electronics* **2021**, *10*, 1194. [[CrossRef](#)]
9. Jang, K.B.; Kim, H.J.; Park, J.H.; Song, G.J.; Seo, H.J. Optimization of LEA Quantum Circuits to Apply Grover's Algorithm. *KIPS Trans. Comput. Commun. Syst.* **2021**, *10*, 101–106.
10. Jang, K.; Kim, H.; Eum, S.; Seo, H. Grover on GIFT. *IACR Cryptol. ePrint Arch.* **2020**, *2020*, 1405.
11. Song, G.; Jang, K.; Kim, H.; Lee, W.K.; Hu, Z.; Seo, H. Grover on SM3. *Cryptology ePrint Archive*. 2021. Available online: <https://eprint.iacr.org/2021/668> (accessed on 4 July 2022).
12. Amy, M.; Matteo, O.D.; Gheorghiu, V.; Mosca, M.; Parent, A.; Schanck, J. Estimating the cost of generic quantum pre-image attacks on SHA-2 and SHA-3. In Proceedings of the International Conference on Selected Areas in Cryptography, St. John's, NL, Canada, 10–12 August 2016; pp. 317–337.
13. Zou, J.; Li, L.; Wei, Z.; Luo, Y.; Liu, Q.; Wu, W. New quantum circuit implementations of SM4 and SM3. *Quantum Inf. Process.* **2022**, *21*, 1–38. [[CrossRef](#)]
14. Anand, R.; Maitra, A.; Mukhopadhyay, S. Grover on SIMON. *Quantum Inf. Process.* **2020**, *19*, 340. [[CrossRef](#)]
15. Huang, Z.; Sun, S. Synthesizing Quantum Circuits of AES with Lower T-depth and Less Qubits. *Cryptology ePrint Archive*. 2022. Available online: <https://eprint.iacr.org/2022/620> (accessed on 4 July 2022).
16. Baksi, A.; Jang, K.; Song, G.; Seo, H.; Xiang, Z. Quantum implementation and resource estimates for RECTANGLE and KNOT. *Quantum Inf. Process.* **2021**, *20*, 395. [[CrossRef](#)]
17. Jang, K.; Baksi, A.; Song, G.; Kim, H.; Seo, H.; Chattopadhyay, A. Quantum Analysis of AES. *Cryptology ePrint Archive*. 2022. Available online: <https://eprint.iacr.org/2022/683> (accessed on 4 July 2022).
18. Jang, K.; Baksi, A.; Breier, J.; Seo, H.; Chattopadhyay, A. Quantum Implementation and Analysis of DEFAULT. *Cryptology ePrint Archive*. 2022. Available online: <https://eprint.iacr.org/2022/647> (accessed on 4 July 2022).
19. Jang, K.B.; Song, G.J.; Kim, H.J.; Seo, H.J. Grover on Simplified AES. In Proceedings of the 2021 IEEE International Conference on Consumer Electronics-Asia (ICCE-Asia), Gangwon, Korea, 1–3 November 2021; pp. 1–4.
20. Song, G.; Jang, K.; Kim, H.; Lee, W.K.; Seo, H. Grover on Caesar and Vigenere ciphers. *Cryptology ePrint Archive*. 2021. Available online: <https://eprint.iacr.org/2021/554> (accessed on 4 July 2022).
21. Leander, G.; Moos, T.; Moradi, A.; Rasoolzadeh, S. The SPEEDY Family of Block Ciphers: Engineering an Ultra Low-Latency Cipher from Gate Level for Secure Processor Architectures. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* **2021**, *4*, 510–545. [[CrossRef](#)]
22. Mermin, N.D. *Quantum Computer Science: An Introduction*; Cambridge University Press: Cambridge, UK, 2007.
23. Steane, A. Quantum computing. *Rep. Prog. Phys.* **1998**, *61*, 117. [[CrossRef](#)]
24. DiVincenzo, D.P. Quantum gates and circuits. *Proc. R. Soc. London. Ser. Math. Phys. Eng. Sci.* **1998**, *454*, 261–276. [[CrossRef](#)]
25. Brylinski, J.L.; Brylinski, R. Universal quantum gates. *Math. Quantum Comput.* **2002**, *79*, 117–134.
26. Amy, M.; Maslov, D.; Mosca, M.; Roetteler, M. A Meet-in-the-Middle Algorithm for Fast Synthesis of Depth-Optimal Quantum Circuits. *IEEE Trans. -Comput.-Aided Des. Integr. Circuits Syst.* **2013**, *32*, 818–830. [[CrossRef](#)]
27. Jang, K.; Song, G.; Kim, H.; Kwon, H.; Kim, H.; Seo, H. Parallel Quantum Addition for Korean Block Cipher. *Cryptology ePrint Archive*. 2021. Available online: <https://eprint.iacr.org/2021/1507> (accessed on 4 July 2022).