applied sciences

MDPI

Article

Efficient Implementation of PRESENT and GIFT on Quantum Computers

Kyungbae Jang, Gyeongju Song, Hyunjun Kim, Hyeokdong Kwon, Hyunji Kim and Hwajeong Seo *

Division of IT Convergence Engineering, Hansung University, Seoul 02876, Korea; starj1234@hansung.ac.kr (K.J.); thdrudwn98@hansung.ac.kr (G.S.); amdjd0704@hansung.ac.kr (H.K.); hyeok@hansung.ac.kr (H.K.); 1594012@hansung.ac.kr (H.K.)

* Correspondence: hwajeong@hansung.ac.kr; Tel.:+82-2-760-8033

Abstract: Grover search algorithm is the most representative quantum attack method that threatens the security of symmetric key cryptography. If the Grover search algorithm is applied to symmetric key cryptography, the security level of target symmetric key cryptography can be lowered from n-bit to $\frac{n}{2}$-bit. When applying Grover's search algorithm to the block cipher that is the target of potential quantum attacks, the target block cipher must be implemented as quantum circuits. Starting with the AES block cipher, a number of works have been conducted to optimize and implement target block ciphers into quantum circuits. Recently, many studies have been published to implement lightweight block ciphers as quantum circuits. In this paper, we present optimal quantum circuit designs of symmetric key cryptography, including PRESENT and GIFT block ciphers. The proposed method optimized PRESENT and GIFT block ciphers by minimizing qubits, quantum gates, and circuit depth. We compare proposed PRESENT and GIFT quantum circuits with other results of lightweight block cipher implementations in quantum circuits. Finally, quantum resources of PRESENT and GIFT block ciphers required for the oracle of the Grover search algorithm were estimated.

Keywords: Grover search algorithm; quantum circuits; PRESENT block cipher; GIFT block cipher

Citation: Jang, K; Song, G.; Kim, H.; Kwon, H.; Kim, H.; Seo, H. Efficient Implementation of PRESENT and GIFT on Quantum Computers. *Appl. Sci.* **2021**, *11*, 4776. https://doi.org/10.3390/app11114776

Academic Editor: José Luis Rojo-Álvarez

Received: 22 February 2021
Accepted: 21 May 2021
Published: 23 May 2021

1. Introduction

With the development of embedded technology, the use of many wearable devices and smart devices has increased [1]. IoT devices exchange abundant network packets with each other, including personal information. This needs to keep the privacy. To prevent leakage of data and information, we need to send and receive data securely without information leakages. For this reason, cryptographic algorithms are required to protect the security of data transmitted and received between devices. However, many IoT devices have low computing power, low memory, and low computing power, which can make it difficult to apply cryptographic algorithms to these devices.

Under these circumstances, the lightweight cryptography targeting low-end devices has been actively researched [2]. Lightweight cryptography algorithms are designed to use resources efficiently. They are working on devices with limited performances. In CHES'07, the lightweight block cipher, namely PRESENT block cipher, was proposed [3]. It was designed with a substitution–permutation–network structure. In CHES'17, the lightweight block cipher, namely GIFT block cipher, was proposed, which improved the PRESENT block cipher, with improved performance and security level [4].

Quantum computers using the Grover search algorithm can reduce the security of the block cipher with 2^n-bit security level to $O(2^{\frac{n}{2}})$ [5]. Block ciphers can be attacked by quantum computers using the Grover's search algorithm. As the development of large-scale quantum computers is still underway, it is very important to minimize quantum resources required for the target block cipher algorithm. With this motivation, research has been conducted to optimize the AES block cipher into quantum circuits [6–9]. Grassl et al.

implemented the AES block cipher as a quantum circuit to evaluate the quantum resources required by the Grover's search algorithm[6]. Later, Jaques et al. and Langenberg et al. implemented more compact quantum circuits than the Grassl's implementation result [7,8]. This is the result obtained by optimizing the Sbox operation that previously used a large amount of quantum resources. In addition to the AES block cipher, many studies on lightweight block ciphers have been conducted. Anand et al. implemented the SIMON block cipher and evaluated quantum resources required by the Grover search algorithm [9,10]. Jang et al. implemented the SPECK block cipher as a quantum circuit to evaluated quantum gates for applying the Grover search algorithm. In [11], the author implemented the Gimli cipher, as a quantum circuit to estimate the required quantum resources. In [12], Jang et al. implemented the Korean lightweight block ciphers, including CAHM, LEA, and HIGHT as quantum circuits and estimated quantum resources.

In this paper, we efficiently implemented PRESSINT and GIFT block ciphers as quantum circuits. Both PRESENT and GIFT block ciphers use Sbox to convert input values to other values. Classical computers can use a predefined Sbox that directly matches the output value depending on the input value. In quantum computers, all input values should coexist and it is impossible to use a predefined Sbox. Therefore, the operation of Sbox should be designed as quantum gates. In order to optimize the quantum circuit, it is important to optimize the Sbox operation. In order to implement the Sbox operation of PRESENT block cipher, the LIGHTER-R tool [13] was used to optimize the operation. The LIGHTER-R tool is described in Section 2. For GIFT block cipher, we chose the hardware-friendly Sbox to optimize the Sbox operation that requires a number of quantum resources. As a result of comparing the quantum circuit of SIMON (i.e., hardware friendly block cipher) and SPECK (i.e., software friendly block cipher) [9,10], we confirmed that the hardware-friendly operation is also optimal for quantum computers. We saved a lot of qubits by implementing a hardware-friendly Sbox. In addition to the Sbox operation of PRESENT and GIFT block ciphers, AddRoundkey and Keyschedule were also optimized.

Finally, we compare proposed PRESNET and GIFT quantum circuit implementations with other lightweight block cipher quantum circuit implementations with similar parameters and security levels. We estimated resources for applying the oracle of Grover's algorithm to PRESENT and GIFT symmetric key cryptography based on the proposed method.

Contribution

- First design of quantum gates for PRESENT and GIFT symmetric key cryptography. As far as we know, we firstly implemented PRESENT and GIFT block ciphers as quantum circuits. We present a method of implementing PRESENT and GIFT block ciphers as quantum circuits for application to the oracle of the Grover search algorithm.
- Quantum circuits with optimized qubits for PRESENT and GIFT block ciphers. One of the most important factors when evaluating quantum circuits is optimizing the number of qubits. When designing quantum circuits, new qubits are allocated to temporal storage or new values. However, we use an on-the-fly approach to recycle the initially allocated qubits until the encryption is finished. By using efficient Sbox quantum implementation, we did not allocate qubits except for the initial key and plaintext.
- Quantum gates and circuit depth analysis for PRESENT and GIFT block ciphers. Proposed PRESENT and GIFT implementations are evaluated using the IBM ProjectQ framework, a quantum computer emulator (https://github.com/ProjectQ-Framework/ProjectQ, accessed on 10 May 2021) [14]. IBM ProjectQ uses a variety of quantum compilers that allow us to simulate quantum computers or draw quantum circuits. Among them, the resource counter compiler, which is a quantum resource estimator, measures quantum resources by analyzing qubits, quantum gates, and circuit depth. Compared with quantum implementation results of other block ciphers, we implemented low-cost PRESENT and GIFT quantum circuits.

2. Related Work

2.1. PRESENT Block Cipher

The compact symmetric key cryptography, namely PRESENT block cipher, was presented in CHES'07 [15]. The PRESENT is a block cipher using the Substitution Permutation Network (SPN) method and consists of 31 rounds. The PRESENT has a block size of 64 bits and supports 80-bit and 128-bit key sizes. In the PRESENT block cipher, each round consists of three steps: AddRoundKey, Sbox, and Permutation. The encryption algorithm of PRESENT block cipher is described in Figure 1. Each round consists of AddRoundkey, Sbox, and Permutation in the order. When the plain text is entered, the round is repeated by 31 times. In the end, AddRoundkey is performed.

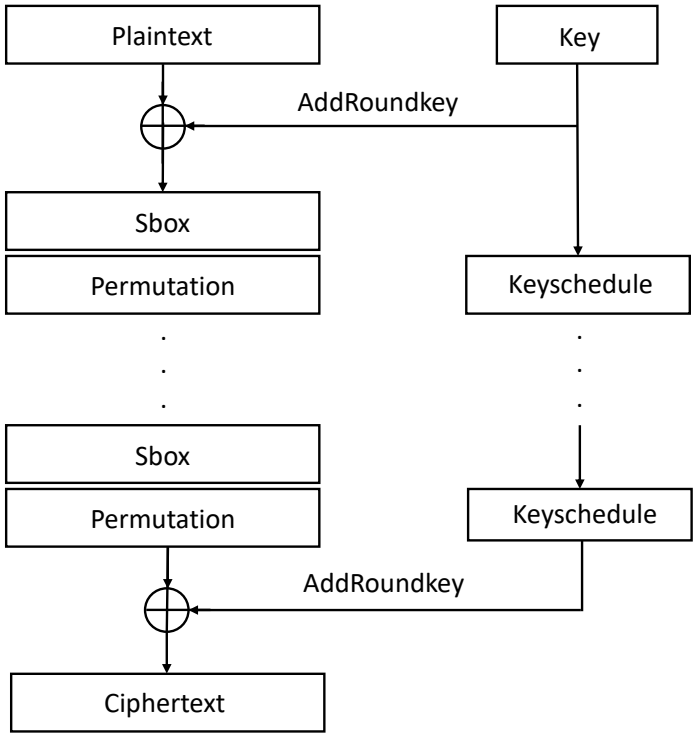

Figure 1. Encryption process of PRESENT block cipher.

2.1.1. AddRoundkey of PRESENT Block Cipher

Given the 64-bit round key $RK_i = rk_{63}, ..., rk_0$ is exclusive-ored to the 64-bit block $B_i = b_{63}, ..., b_0$ for $1 \leq i \leq 32$. The notation $\oplus$ means XOR operation.

$$b_j \leftarrow b_j \oplus rk_j, \quad j = 0, ..., 63 \tag{1}$$

2.1.2. Sbox of PRESENT Block Cipher

The 64-bit block is split into 4 bits and becomes the input value of the 4-bit Sbox. The Sbox of PRESENT block cipher is given in Table 1.

Table 1. Sbox of PRESENT block cipher.

x	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
Sbox(x)	c	5	6	b	9	0	a	d	3	e	f	8	4	7	1	2

2.1.3. Permutation of PRESENT Block Cipher

In the PRESENT block cipher, the permutation replaces the $P_{64}(i)$-th bit of block B with the i-th bit of block B. Details on the permutation of PRESENT block cipherare shown in Table 2.

Four bits (0, 21, 42, and 63), which are colored in red, do not change their positions. On the other hand, three bits (1, 4, and 16), which are colored in blue, exchange positions with each other. This is explained in detail in Section 3.

Table 2. Bit permutation of PRESENT block cipher.

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$P_{64}(i)$	0	16	32	48	1	17	33	49	2	18	34	50	3	19	35	51
i	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
$P_{64}(i)$	4	20	36	52	5	21	37	53	6	22	38	54	7	23	39	55
i	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
$P_{64}(i)$	8	24	40	56	9	25	41	57	10	26	42	58	11	27	43	59
i	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
$P_{64}(i)$	12	28	44	60	13	29	45	61	14	30	46	62	15	31	47	63

2.1.4. Keyschedule of PRESENT Block Cipher

In the version using the 80-bit key, round key RK is the leftmost 64 bits of the 80-bit key.

$$RK = rk_{63}, ...rk_0 = k_{79}, ..., k_{16} \tag{2}$$

In the version using 128-bit key, round key RK is the leftmost 64 bits of 128-bit key.

$$RK = rk_{63}, ...rk_0 = k_{127}, ..., k_{64} \tag{3}$$

After extracting the round key RK, keys ($K = k_{79}, ..., k_0$ for 80-bit security level or $K = k_{127}, ..., k_0$ 128-bit security levels) are updated with Rotation, Sbox, and XOR operations. Round key generation of 80-bit security level and 128-bit security level are given in Equations (4) and (5), respectively.

$$\begin{aligned} &k_{79}, k_{78}, ...k_1, k_0 \leftarrow k_{18}, k_{17}, ...k_{20}, k_{19} \\ &k_{79}, k_{78}, k_{77}, k_{76} \leftarrow \text{Sbox}(k_{79}, k_{78}, k_{77}, k_{76}) \\ &k_{19}, k_{18}, k_{17}, k_{16}, k_{15} \leftarrow k_{19}, k_{18}, k_{17}, k_{16}, k_{15} \oplus \text{round } i \end{aligned} \tag{4}$$

$$\begin{aligned} &k_{127}, k_{126}, ...k_1, k_0 \leftarrow k_{66}, k_{65}, ...k_{68}, k_{67} \\ &k_{127}, k_{126}, k_{125}, k_{124} \leftarrow \text{Sbox}(k_{127}, k_{126}, k_{125}, k_{124}) \\ &k_{123}, k_{122}, k_{121}, k_{120} \leftarrow \text{Sbox}(k_{123}, k_{122}, k_{121}, k_{120}) \\ &k_{66}, k_{65}, k_{64}, k_{63}, k_{62} \leftarrow k_{66}, k_{65}, k_{64}, k_{63}, k_{62} \oplus \text{round } i \end{aligned} \tag{5}$$

2.2. GIFT Block Cipher

The GIFT block cipher is a symmetric key cryptography using the Substitution Permutation Network (SPN) method. There are GIFT-64/128 (64-bit block and 128-bit key) and GIFT-128/128 (128-bit block and 128-bit key). In the GIFT block cipher, each round performs four steps: Sbox, Permutation, AddRoundKey and Constant XOR. The encryption operation of GIFT block cipher is described in Figure 2.

2.2.1. Sbox of GIFT Block Cipher

The n-bit block ($n = 64, 128$) is split into 4 bits and becomes the input value of the 4-bit Sbox. The Sbox of GIFT block cipher is given in Table 3.

Table 3. Sbox of GIFT block cipher.

x	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
Sbox(x)	1	a	4	c	6	f	3	9	2	d	b	7	5	0	8	e

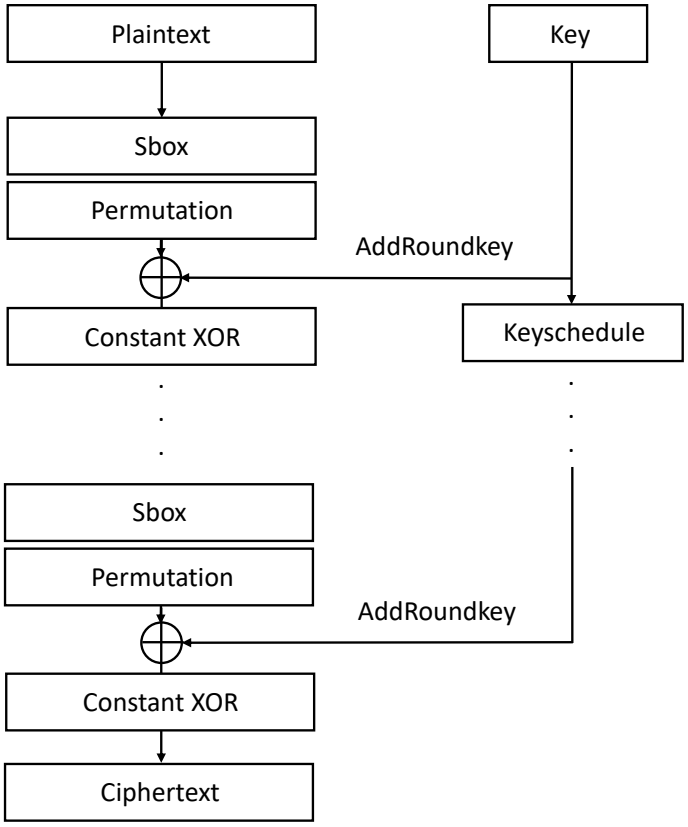

Figure 2. Encryption process of GIFT block cipher.

2.2.2. Permutation of GIFT Block Cipher

In the permutation, GIFT-64/128 replaces the $P_{64}(i)$-th bit of block B with the i-th bit of block B. Details on the permutation of GIFT-64/128 are shown in Table 4. In this paper, detailed Table on permutation of GIFT-128/128 is omitted. Permutation Table of GIFT-128/128 can be found in [4].

Table 4. Permutation of GIFT-64 bit.

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$P_{64}(i)$	0	17	34	51	48	1	18	35	23	49	2	19	16	33	50	3
i	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
$P_{64}(i)$	4	21	38	55	52	5	22	39	36	53	6	23	20	37	54	7
i	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
$P_{64}(i)$	8	25	42	59	56	9	26	43	40	57	10	27	24	41	58	11
i	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
$P_{64}(i)$	12	29	46	63	60	13	30	47	44	61	14	31	28	45	62	15

2.2.3. AddRoundkey of GIFT Block Cipher

In the GIFT-64/128 block cipher, k_0 and k_1 (32-bit total) are selected from the key $(K = k_7, ..., k_0)$. k_0 and k_1 are used as U and V of the round key as follows, $RK = U||V = u_{15}...u_0||v_{15}...v_0$ $(U = k_1, V = k_0)$. The round key is exclusive-ored with the block B, where U is XORed to b_{4i+1} and V is XORed to b_{4i}.

$$b_{4i+1} \leftarrow b_{4i+1} \oplus u_i, \ b_{4i} \leftarrow b_{4i} \oplus v_i, \ i = 0, ..., 15 \tag{6}$$

In the GIFT-128/128 block cipher, k_0, k_1, k_4, and k_5 (64-bit in a total) are selected from the key K. k_0, k_1, k_4 and k_5 are used as U and V of the round key as follows,

$RK = U||V = u_{31}...u_0||v_{31}...v_0$ ($U = k_5||k_4, V = k_1||k_0$). The round key is XORed to the block B, where U is XORed to b_{4i+2} and V is XORed to b_{4i+1}.

$$b_{4i+2} \leftarrow b_{4i+2} \oplus u_i,\ b_{4i+1} \leftarrow b_{4i+1} \oplus v_i,\ i = 0, ..., 31 \tag{7}$$

2.2.4. Constant XOR of GIFT Block Cipher

Round constants C given in Table 5 are used in GIFT-64/128 and GIFT-128/128 block ciphers. Single bit and round constants ($C = c_5c_4c_3c_2c_1c_0$) are XORed to block B as in Equation (8).

$$\begin{aligned} &b_{n-1} \leftarrow b_{n-1} \oplus 1, \\ &b_{23} \leftarrow b_{23} \oplus c_5,\ b_{19} \leftarrow b_{19} \oplus c_4,\ b_{15} \leftarrow b_{15} \oplus c_3, \\ &b_{11} \leftarrow b_{11} \oplus c_2,\ b_7 \leftarrow b_7 \oplus c_1,\ b_3 \leftarrow b_3 \oplus c_0. \end{aligned} \tag{8}$$

Table 5. Round constants C.

Rounds	**Constants *C***															
1 to 16	01	03	07	0F	1F	3E	3D	3B	37	2F	1E	3C	39	33	27	0E
17 to 32	1D	3A	35	2B	16	2C	18	30	21	02	05	0B	17	2E	1C	38
33 to 48	31	23	06	0D	1B	36	2D	1A	34	29	12	24	08	11	22	04

2.2.5. Keyschedule of GIFT Block Cipher

In GIFT-64/128 and GIFT-128/128 block ciphers, the Keyschedule updates key ($K = k_7, ..., k_0$) and extracts the round key from the updated key K. The Keyschedule is shown in Equation (9). The notation ($\ggg i$) denotes a right rotation operation (i-bit).

$$k_7||k_6||...||k_1||k_0 \leftarrow k_1 \ggg 2||k_0 \ggg 12||...||k_3||k_2, \tag{9}$$

2.3. Quantum Gates and Algorithm

2.3.1. Quantum Gates

To perform the work of classical gates, quantum gates should be implemented. CNOT gate and Toffoli gate are the most commonly used in quantum circuits. The CNOT gate receives two qubits and XOR the first qubit to the second qubit (i.e., CNOT $(a, b) \rightarrow a = a,\ b = a \oplus b$). This gate stores the XOR result of the two input qubits in the second qubit. The quantum circuit of CNOT gate is shown in Figure 3 (left). The Toffoli gate is more expensive and complex than the CNOT gate. Three qubits are input to the Toffoli gate, and the AND result of the first and second qubits is XORed to the third qubit (i.e., Toffoli (a, b, c) $\rightarrow a = a,\ b = b,\ c = c \oplus (a \cdot b)$. The quantum circuit of Toffoli gate is shown in Figure 3 (right). The notation $(\cdot)$ indicates AND operation.

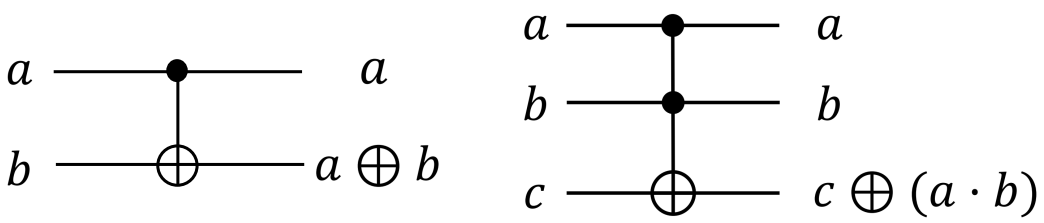

Figure 3. CNOT gate (left) and Toffoli gate (right).

The logical-OR quantum gate is composed of a combination of of Toffoli gate and X gate, as shown in Figure 4. The X gate operates on a single qubit and performs a NOT operation. In the logical OR quantum gate, a and b (input qubits) are changed (0 to 1, 1 to 0) by the X gate (i.e., Quantum OR $(a, b, c) \rightarrow a = \sim a,\ b = \sim b,\ c = c \oplus (a \vee b)$. To return to the original a or b, reversible gate must be performed by executing the X gate once more. The OR quantum gate (reversible) is shown in Figure 4. The notation $\vee$ represents logical-OR operation.

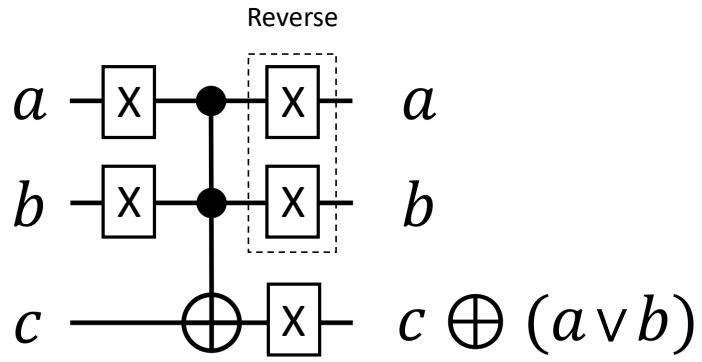

Figure 4. Logical-OR quantum gate.

2.3.2. Grover's Search Algorithm

The Grover's search algorithm finds unique data in the database. If a brute force attack requires $O(2^n)$ searches, this can be found in $O(2^{\frac{n}{2}})$ searches using the Grover search algorithm. There are two core modules in the Grover's search algorithm (i.e., oracle and diffusion operator). The circuit structure of Grover's search algorithm is shown in Figure 5.

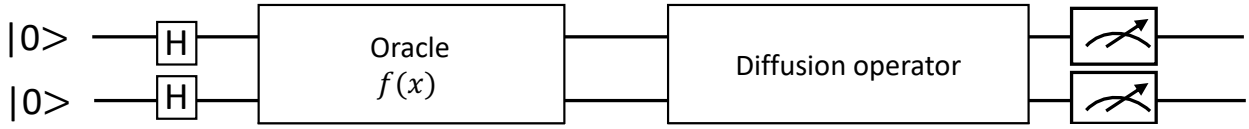

Figure 5. Grover search algorithm.

If the x constitutes the data to be found in the database, the oracle $f(x)$ returns 1. In oracle, if $f(x)$ returns 1, it flips the sign of the state x. When $f(10) = 1$, the state after oracle is shown in Figure 6.

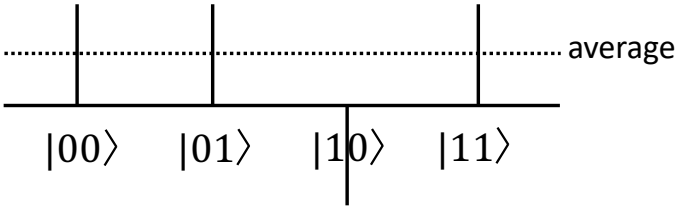

Figure 6. State after oracle.

Steps of the diffusion operator are as follows. First, we calculate the average of amplitudes of all data. Then, we calculate the gap between the average obtained earlier and the amplitude of each data (i.e., average amplitude—(each amplitude—average amplitude)). Therefore, the probability of the answer data x increases, and the probability of non-answers decreases. As shown in Figure 7, when operating with 2 qubits, it is possible to find a solution with 100% probability just by performing diffusion operator once. The Grover's search algorithm iterates over the oracle and the diffusion operator to measure the answer data x with a high probability.

Figure 8 shows the overall key search quantum circuit for a block cipher using the Grover search algorithm. First, we applied Hadamard gates to key qubits to make them superposition states. Second, an encryption quantum circuit is implemented in the oracle to encrypt plain text. Then, it checks whether the known ciphertext matches the generated ciphertext. If they match, the sign of the key state is inverted. Finally, the diffusion operator operates only on key qubits to amplify the amplitude of the solution. Afterward, we iterate the oracle and diffusion operator to increase the amplitude of the solution key, and then measure the key qubits.

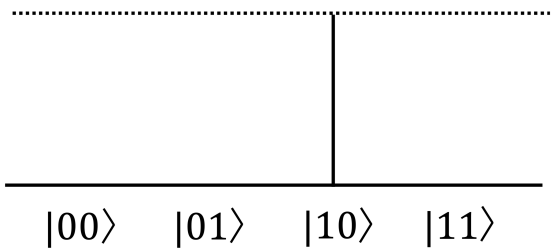

Figure 7. State after diffusion operator.

The most important part in quantum attack using the Grover's search algorithm is how to implement the oracle and how to optimize the quantum circuit for the encryption part.

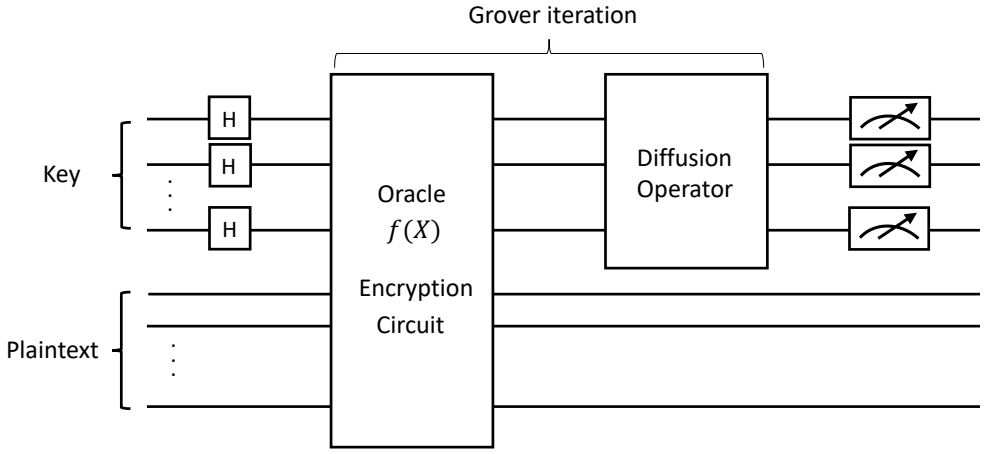

Figure 8. Grover key search quantum circuit for block cipher.

2.4. LIGHTER-R

In [13], authors presented the 4-bit Sbox implementation generator for quantum computers, namely LIGHTER-R. LIGHTER-R is an extension of LIGHTER developed for classical computers, targeting quantum computers [16]. The LIGHTER uses the Meet In The Middle approach to design the compact result of 4-bit Sbox for classical computers. On the other hand, the LIGHTER-R which extends the LIGHTER can implement a 4-bit Sbox optimized for quantum computers using various versions of reversible logic libraries. Details of the LIGHTER-R are described in [13]. Using the LIGHTER-R tool, we can implement an optimized Sbox quantum circuit.

3. Proposed Method

3.1. Quantum circuit for PRESENT Block Cipher

In the proposed PRESENT quantum circuit, only qubits for plaintext and key are allocated. Therefore, no additional qubits were used. All operations, including AddRoundkey, Sbox, Permutation, and Keyschedule, were optimized in terms of qubits and quantum gates.

3.1.1. AddRoundkey of PRESENT Block Cipher

In the AddRoundkey operation, the leftmost 64 qubits of master key are used as the round key. The round key is exclusive-ored with the 64-qubit block ($B(b_{63}, ..., b_0)$), which becomes the ciphertext. The XOR operation can be performed using the CNOT gate. At this time, the qubit whose result value changes should be B. The implementation of PRESENT AddRoundKey quantum circuit is shown in Algorithm 1.

Algorithm 1 Quantum circuit for AddRoundKey of PRESENT block cipher

Input: 64-qubit block $B(b_{63}, ..., b_0)$, 64-qubit round key $RK(rk_{63}, ..., rk_0)$
Output: 64-qubit block $B(b_{63}, ...b_0)$ after AddRoundKey
1: **for** $i = 0$ to 63 **do**
2: $b_i \leftarrow$ CNOT (rk_i, b_i)
3: **end for**
4: **return** $B(b_{63}, ..., b_0)$

3.1.2. Sbox of PRESENT Block Cipher

Classical computers can utilize a predefined Sbox that directly matches the output value according to the input value(i.e., Table 1). However, the predefined Sbox cannot be used in quantum computers where multiple values exist as probabilities due to the qubit superposition. For quantum computers, we have to implement the Sbox equation as a quantum circuit with different outputs depending on the input. The Sbox operation of PRESENT block cipher is shown in Equation (10). The notation $x_0x_1x_2'$ indicates x_0 AND x_1 AND (NOT x_2).

To implement this in a quantum circuit, additional qubits and expensive quantum gates should be used. An additional 4 qubits must be allocated to store the result value of Sbox (i.e., Sbox(x_0), Sbox(x_1), Sbox(x_2), and Sbox(x_3)) and many AND operations in Equation (10) increase the use of Toffoli gate, which is expensive quantum gates.

$$\begin{aligned}
\text{Sbox}(x_0) &\leftarrow x_3'x_2'x_0 \oplus x_3'x_1x_0 \oplus x_3x_2'x_0' \oplus x_3x_1x_0' \oplus x_3x_2x_1'x_0 \oplus x_3'x_2x_1'x_0' \\
\text{Sbox}(x_1) &\leftarrow x_3'x_2'x_1 \oplus x_3x_2x_0 \oplus x_3'x_1x_0' \oplus x_3x_2'x_1' \oplus x_3x_2'x_0' \\
\text{Sbox}(x_2) &\leftarrow x_3x_2x_1' \oplus x_2'x_1x_0' \oplus x_3'x_2x_1x_0 \oplus x_3'x_2'x_1' \oplus x_2'x_1'x_0 \\
\text{Sbox}(x_3) &\leftarrow x_3x_2'x_1 \oplus x_3x_2'x_0 \oplus x_3'x_1'x_0' \oplus x_3'x_2x_1 \oplus x_3'x_1x_0
\end{aligned} \tag{10}$$

In order to avoid such an inefficient Sbox implementation of Equation (10), we implemented the optimal Sbox quantum circuit using the LIGHTER-R tool. The LIGHTER-R tool generates quantum circuits using graph-based MITM search algorithm according to the input and output values. Input qubits of the quantum circuit are used as the output qubits after the Sbox operation is completed. Therefore, there is no need to allocate additional qubits, and the graph-based MITM search algorithm matches output values more simply than the Equation (10). By using the LIGHTER-R tool, we can implement an optimized PRESENT Sbox quantum circuit with no additional qubits and low gate cost. The optimized implementation of PRESENT Sbox quantum circuit using the LIGHTER-R is shown in Algorithm 2.

Algorithm 2 Quantum circuit using LIGHTER-R for Sbox of PRESENT block cipher

Input: 4-qubit input $x(x_3, x_2, x_1, x_0)$ (before entering Sbox).
Output: 4-qubit output $x(x_3, x_2, x_1, x_0)$ (after performing Sbox).
1: $x_1 \leftarrow$ CNOT (x_2, x_1)
2: $x_3 \leftarrow$ Toffoli (x_1, x_2, x_3)
3: $x_2 \leftarrow$ Toffoli (x_3, x_1, x_2)
4: $x_1 \leftarrow$ Toffoli (x_0, x_2, x_1)
5: $x_2 \leftarrow$ CNOT (x_3, x_2)
6: $x_3 \leftarrow$ X (x_3)
7: $x_2 \leftarrow$ CNOT (x_1, x_2)
8: $x_0 \leftarrow$ CNOT (x_3, x_0)
9: $x_1 \leftarrow$ CNOT (x_0, x_1)
10: $x_0 \leftarrow$ X (x_0)
11: $x_3 \leftarrow$ Toffoli (x_1, x_2, x_3)
12: **return** $x(x_1, x_3, x_2, x_0)$

The arrangement of input quibts and output qubits was changed in Algorithm 2. It can be performed with Swap gates on x_1, x_3 and x_1, x_2. However, this can be done by qubit relabeling to treat x_1 as x_2, x_2 as x_3, and x_3 as x_1 without Swap gates. Swap gates are used in the implementation, but not counted as resources. Figure 9 shows PRESENT Sbox quantum circuit.

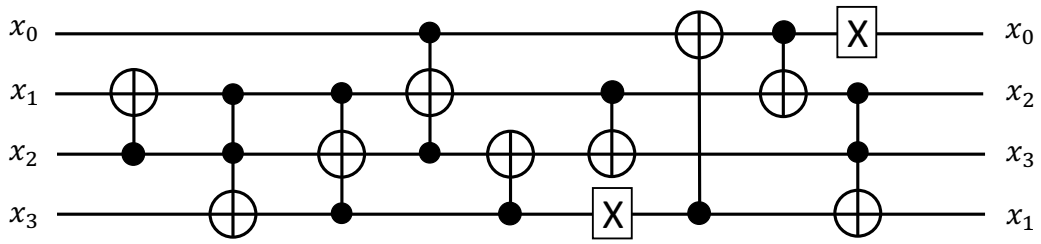

Figure 9. Quantum circuits for Sbox of PRESENT block cipher.

3.1.3. Permutation of PRESENT Block Cipher

The permutation operation changes the bit order as shown in Table 2. It can be implemented using only Swap gates. In Table 2, 4 bits colored in red (0, 21, 42, and 63) do not change the position, but all 60 bits except 4 bits colored in red change the position. Moreover, 60 bits are grouped into 20 3-bit (e.g., blue) to exchange bit positions with each other. This can be done with two Swap gates as follows.

$$\text{Swap}(b_1, b_4),\ \text{Swap}(b_4, b_{16}) \tag{11}$$

The permutation is completed by performing this step by 19 more times for the remaining 57 bits. Finally, 40 Swap gates are used. Otherwise, this can be implemented with qubits relabeling. Therefore, Swap gates are not counted as gate cost. Therefore, the quantum cost for Permutation of PRESENT block cipher is zero.

3.1.4. Keyschedule of PRESENT Block Cipher

The PRESENT block cipher can use 80-bit key or 128-bit key. Since they are similar, this paper only describes the 80-bit key schedule version.

The key is rotated by 19 bits to the right and it can be done using only Swap gates. After the rotation operation, the Sbox of Algorithm 2 is performed on the leftmost 4 qubits (k_{79}, k_{78}, k_{77}, and k_{76}). Finally, the round i is exclusive-ored to $k_{19}, k_{18}, k_{17}, k_{16}$, and k_{15} and the leftmost 64 qubits are extracted and used as round key. XORing round i can be done with X gates at the position, where the i-th bit is set to 1. For example, when $i = 1$, an X gate is performed on k_{15}, when $i = 2$, an X gate is performed on k_{16}, and when $i = 3$, an X gate is performed on k_{15} and k_{16}. The detailed process for Keyschedule of PRESENT-80 is given in Algorithm 3.

Algorithm 3 Quantum circuits for Keyschedule of PRESENT-80

Input: 80-qubit key $K(k_0, ..., k_{79})$.
Output: 64-qubit round key $RK(rk_{63}, ..., rk_0)$.
1: $k \leftarrow k \ggg 19$ using Swap gates
2: $k_{79}, k_{78}, k_{77}, k_{76} \leftarrow$Sbox $(k_{79}, k_{78}, k_{77}, k_{76})$
3: $k_{19}, k_{18}, k_{17}, k_{16}, k_{15} \leftarrow$X $(k_{19}, k_{18}, k_{17}, k_{16}, k_{15})$ according to round i
4: **return** $K(k_{79}, ..., k_{16})$

3.2. Quantum Circuit for GIFT Block Cipher

In the presented GIFT$-n/128$ quantum circuit, only (n+128)-qubits are allocated respectively to assign plaintext (n-bit) and key (128-bit). Therefore, it is optimized without additional qubits. All operations, including AddRoundkey, Sbox, Permutation, Constant XOR, and Keyschedule, were optimized in terms of quantum resources.

3.2.1. Sbox of GIFT Block Cipher

In [4], the author of GIFT block cipher presented two versions of Sbox. One is optimized for implementation in software and the other is optimized for implementation in hardware. Detailed processes are given in Algorithms 4 and 5 for software-oriented and hardware-oriented, respectively. Comparing these two Sboxes, the hardware-friendly Sbox offers more advantages over the software-friendly Sbox when it comes to quantum circuits.

Algorithm 4 Software-oriented implementation of GIFT Sbox

Input: 4-bit input $x(x_3, x_2, x_1, x_0)$ (before entering Sbox).
Output: 4-bit output $x(x_3, x_2, x_1, x_0)$ (after performing Sbox).
1: $x_1 \leftarrow x_1$ XOR (x_0 AND x_2)
2: t $\leftarrow x_0$ XOR (x_1 AND x_3)
3: $x_2 \leftarrow x_2$ XOR (t OR x_1)
4: $x_0 \leftarrow x_3$ XOR x_2
5: $x_1 \leftarrow x_1$ XOR x_3
6: $x_0 \leftarrow$ NOT x_0
7: $x_2 \leftarrow x_2$ XOR (t AND x_1)
8: $x_3 \leftarrow$ t
9: **return** $x(x_3, x_2, x_1, x_0)$

In the software-friendly Sbox operation (see Algorithm 4), the input and output of operations are different (e.g., $x_0 = x_3$ XOR x_2). Qubits in quantum computers must be initialized to zero to overwrite the new value. In order to initialize a qubit to zero, the same value must exist in another qubit. The Algorithm 4 is designed as a quantum circuit Since the new value cannot be overwritten (e.g., $x_0 \leftarrow x_3$ XOR x_2, we have to allocate a new qubit and also allocate an additional qubit for temporary storage. On the other hand, we can see that in the hardware-oriented Sbox design of Algorithm 5, the input and output of the operation are always the same.

Algorithm 5 Hardware-oriented implementation of GIFT Sbox

Input: 4-bit input $x(x_3, x_2, x_1, x_0)$ (before entering Sbox).
Output: 4-bit output $x(x_3, x_2, x_1, x_0)$ (after performing Sbox).
1: $x_1 \leftarrow x_1$ XNOR (x_0 NAND x_2)
2: $x_0 \leftarrow x_0$ XNOR (x_1 NAND x_3)
3: $x_2 \leftarrow x_2$ XNOR (x_0 NOR x_1)
4: $x_3 \leftarrow x_3$ XNOR x_2
5: $x_1 \leftarrow x_1$ XNOR x_3
6: $x_2 \leftarrow x_2$ XNOR (x_0 NAND x_1)
7: **return** $x(x_0, x_2, x_1, x_3)$

Therefore, we were able to optimize the quantum circuit by choosing a hardware-friendly Sbox. The resulting value can be stored in qubits that are entered into the operation. For example, the quantum circuit for line 4 of Algorithm 5 corresponds to lines 9 and 10 of Algorithm 6. The operation continues on x_3 without allocating additional qubits.

Therefore, no additional qubits are used and an optimized 4-qubit Sbox quantum circuit can be implemented. The implementation of GIFT Sbox quantum circuit is described in Algorithm 6.

The NOT operation is performed twice on lines 1, 2, 3 and 6 of the Algorithm 5. Two NOT operations cancel each other. The arrangement of input and output qubits is altered in Algorithm 6. It can be performed with one Swap gate on x_0, x_3. As mentioned earlier, Swap gates are not considered quantum resources. Quantum circuit for Sbox of GIFT block cipher is described in Figure 10.

Algorithm 6 Quantum circuits for Sbox of GIFT block cipher

Input: 4-qubit input $x(x_3, x_2, x_1, x_0)$ (before entering Sbox).
Output: 4-qubit output $x(x_3, x_2, x_1, x_0)$ (after performing Sbox).
1: $x_1 \leftarrow$ Toffoli (x_0, x_2, x_1)
2: $x_0 \leftarrow$ Toffoli (x_1, x_3, x_0)
3: $x_0 \leftarrow$ X (x_0)
4: $x_1 \leftarrow$ X (x_1)
5: $x_2 \leftarrow$ Toffoli (x_0, x_1, x_2)
6: $x_2 \leftarrow$ X (x_2)
7: $x_0 \leftarrow$ X (x_0) (reverse)
8: $x_1 \leftarrow$ X (x_1) (reverse)
9: $x_3 \leftarrow$ CNOT (x_2, x_3)
10: $x_3 \leftarrow$ X (x_3)
11: $x_1 \leftarrow$ CNOT (x_3, x_1)
12: $x_1 \leftarrow$ X (x_1)
13: $x_2 \leftarrow$ Toffoli (x_0, x_1, x_2)
14: **return** $x(x_0, x_2, x_1, x_3)$

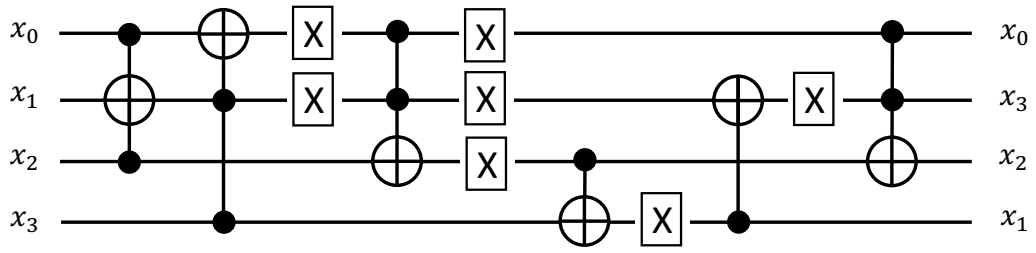

Figure 10. Quantum circuit for Sbox of GIFT block cipher.

3.2.2. Permutaiton of GIFT Block Cipher

After the Sbox operation, the permutation of Table 4 is performed. Similar to the permutation of PRESENT block cipher, bit position changes can be made using Swap gates and are not counted as quantum resources. Therefore, by relabeling the qubits, the permutation of GIFT block cipher can be done without quantum resources.

3.2.3. AddRoundkey of GIFT Block Cipher

In the AddRoundkey operation, the round key RK ($n/2$-bit) is XORed to the block B ($n/2$-bit). The XOR operation can be done with the CNOT gate. At this time, the result of qubit should be B. The AddRoundkey of GIFT-64/128 and GIFT-128/128 block ciphers are similar, only the number of bits is different. In the GIFT-128/128 block cipher, double the CNOT gates are used compared to the GIFT-64/128 block cipher. Quantum circuits for Addroundkey of GIFT-64/128 and GIFT-128/128 block ciphers are shown in Algorithm 7 and Algorithm 8, respectively.

Algorithm 7 Quantum circuit for AddRoundkey of GIFT-64/128 block cipher

Input: 64-qubit block $B(b_{63}, ..., b_0)$, 32-qubit round key $RK(rk_{31}, ..., rk_0)$.
Output: 64-qubit block $B(b_{63}, ..., b_0)$ after AddRoundKey.
1: **for** $i = 0$ to 15 **do**
2: $b_{4i} \leftarrow$ CNOT (rk_i, b_{4i})
3: $b_{4i+1} \leftarrow$ CNOT (rk_{i+16}, b_{4i+1})
4: **end for**
5: **return** $B(b_{63}, ..., b_0)$

Algorithm 8 Quantum circuits for AddRoundkey of GIFT-128/128 block cipher

Input: 128-qubit block $B(b_{127}, ..., b_0)$, 64-qubit round key $RK(rk_{63}, ..., rk_0)$.
Output: 128-qubit block $B(b_{127}, ..., b_0)$ after AddRoundKey.
1: **for** $i = 0$ to 31 **do**
2: $b_{4i+1} \leftarrow$ CNOT (rk_i, b_{4i+1})
3: $b_{4i+2} \leftarrow$ CNOT (rk_{i+32}, b_{4i+2})
4: **end for**
5: **return** $B(b_{127}, ..., b_0)$

3.2.4. Constant XOR of GIFT Block Cipher

The round constant C in Table 5 and the single bit are XORed to block B. Since the constant C for each round is already set, we performed X gates on $b_{23}, b_{19}, b_{15}, b_{11}, b_7$, and b_3 only for positions where bit of C is one. When the round constant C is 3 in round 2, c_0 and c_1 are 1. Therefore, the X gate (b_3) and X gate (b_7) are performed. For a single bit, an X gate is always performed on b_{n-1}. In this way, no qubits are used for Constant XOR. Moreover, CNOT gates are not used; only X gates are used. The implementation of the GIFT-n/128 constant XOR quantum circuit is described in Algorithm 9.

Algorithm 9 Quantum circuits for constant XOR of GIFT-n/128 block cipher

Input: $b_{n-1}, b_{23}, b_{19}, b_{15}, b_{11}, b_7, b_3$ of n-bit block B.
Output: $b_{n-1}, b_{23}, b_{19}, b_{15}, b_{11}, b_7, b_3$ of n-bit block B after Constant XOR.
1: $b_{23}, b_{19}, b_{15}, b_{11}, b_7, b_3 \leftarrow$X $(b_{23}, b_{19}, b_{15}, b_{11}, b_7, b_3)$ according to round constant C $(c_5, c_4, c_3, c_2, c_1, c_0)$
2: $b_{n-1} \leftarrow$ X (b_{n-1})
3: **return** $b_{n-1}, b_{23}, b_{19}, b_{15}, b_{11}, b_7, b_3$

3.2.5. Keyschedule of GIFT Block Cipher

In GIFT, the state of the key is updated after the round key is used, which is shown in Equation (9). The Keyschedule of PRESENT block cipher uses Sbox and round i, while the GIFT only changes the bit positions of key K. It can be done using only Swap gates, and using the method of relabeling qubits does not require quantum resources.

4. Evaluation

Proposed PRESENT and GIFT implementations are evaluated by using the quantum computer emulator IBM ProjectQ. IBM ProjectQ offers a variety of quantum computer compilers and can estimate quantum resources of quantum circuits. One of the quantum compilers, the resource counter, analyzes CNOT gates, Toffoli gates, X gates, and the qubits used in the quantum circuit and estimates the circuit depth. Proposed implementations focused on optimizing the number of quantum gates, circuit depth, and qubits.

In Table 6, quantum resources to implement PRESENT and GIFT symmetric key cryptography as quantum gates are evaluated. In addition, the results of quantum implementation of other block ciphers, which have been studied recently, are shown. We compare proposed PRESNET and GIFT quantum circuit implementations with the SIMON, SPECK, and CHAM quantum circuit implementations.

First, the optimal number of qubits was achieved, because qubits were only used to allocate plaintext and key. We generate a round key for each round, directly. By using this method, we use round key and update it as the next round key. Therefore, we allocate qubits for only the first key and recycle them to the end. In the implemented quantum circuit, no qubits were used until the completion of the final round. Since large-scale quantum computers have not yet been developed, optimizing the number of qubits required for quantum circuits should be considered.

Second, in terms of quantum gates, PRESENT and GIFT belong to the low-cost group along with SIMON which is designed to be hardware friendly. When we implement

quantum circuits, a relatively small number of quantum gates were used to perform the rounds. Particularly, we were able to save a lot of quantum gate cost by using the LIGHTER-R tool for PRESENT and hardware-friendly Sbox for the GIFT block cipher. The cost of quantum gates required for Sbox could be reduced, and as a result, overall quantum circuit optimization was achieved.

Lastly, in terms of circuit depth, PRESENT and GIFT block ciphers are highly optimized compared to other block ciphers. The exact depth of SPECK and CHAM block ciphers has not been estimated, but it is higher than that of SIMON block cipher. In the case of AddRoundKey and Constant XOR of GIFT and PRESENT block ciphers, qubits do not interact with each other. Each can be performed with one parallel gate operation (depth of 1). The operation that takes up the most depth is Sbox, which is 9 for PRESENT Sbox and 10 for GIFT Sbox. The GIFT block cipher only performs Sbox for block B of each round, whereas the PRESENT block cipher uses Sbox for Keyschedule as well as block B. In the PRESENT block cipher, Sbox operation of Keyschedule and Sbox operation of block B are independent each other. These operations are executed in parallel way. Therefore, the depth is the same as that when Sbox was executed once. With these features, GIFT and PRESENT block ciphers are highly optimized in terms of depth using a small depth in each round. Reducing the depth of the quantum circuit is important for optimization, because the overall execution time is shortened [17]. The reason that PRESENT-64/80 and PRESENT-64/128 block ciphers have the same depth is because the number of rounds is the same (31 rounds).

Table 6. Comparison of quantum resources to implement PRESENT, GIFT, and other block ciphers.

Quantum Circuit	Qubits	Toffoli Gates	CNOT Gates	X Gates	Circuit Depth
PRESENT-64/80 (This work)	144	2108	4683	1118	311
PRESENT-64/128 (This work)	192	2232	4838	1164	311
GIFT-64/128 (This work)	192	1792	1792	3261	308
GIFT-128/128 (This work)	256	6144	6144	10,953	528
SIMON-64/128 [9]	192	1408	7396	1216	2643
SIMON-128/128 [9]	256	4352	17,152	4224	8427
SPECK-64/128 [12]	193	3286	9238	57	-
SPECK-128/128 [12]	257	7942	22,086	75	-
CHAM-64/128 [12]	196	2400	12,285	240	-
CHAM-128/128 [12]	268	4960	26,885	240	-

According to [18], to attack block ciphers with Grover's search algorithm, known plain text and ciphertext pairs are required. In detail, r = (key length/block size) pairs are used. In [19], they estimated the quantum gates required to apply Grover's search algorithm in parallel to AES quantum circuit implementations. To apply the proposed PRESENT and GIFT quantum circuits to the Grover search algorithm, $(1 + r \cdot q)$ qubits are used. q is qubits of Table 6.

The PRESENT block cipher requires 4 instances since r = 2, therefore the number of gates is four times the Table 6 result. Since the r of GIFT-64/128 block cipher is 2, 4 instances are required, therefore the number of gates is four times the Table 6 result. Since r of GIFT-64/128 block cipher is 1, 2 instances are required, therefore the number of gates is two times the Table 6 result. This is why GIFT-64/128 block cipher in Table 7 requires more qubits than GIFT-128/128 block cipher. For parallel search, 2·(key length)·$(r - 1)$ CNOT gates are additionally used. In Table 7, quantum resources to apply PRESENT and GIFT algorithms to the oracle of Grover's search are shown.

Table 7. Quantum resources to apply PRESENT and GIFT algorithms to the oracle of Grover's search algorithm.

Symmetric Key Cryptography	Qubits	Toffoli Gates	CNOT Gates	X Gates
PRESENT-64/80	289	8432	18,892	4472
PRESENT-64/128	385	8928	19,608	4656
GIFT-64/128	385	7168	7424	13,044
GIFT-128/128	257	12,288	12,288	21,906

5. Conclusions

We designed efficient implementations of PRESENT and GIFT block ciphers on quantum computers. Our quantum circuits for PRESNET and GIFT block ciphers achieved an optimal number by minimizing qubits, quantum gates, and circuit depth. Based on the proposed method, we estimated quantum resources to apply the Grover search algorithm.

Future work is to implement another block cipher as a quantum circuit to evaluate quantum resources for the Grover search algorithm. It seems very meaningful to estimate quantum resources for implementing candidate block ciphers in lightweight cryptography competition hosted by NIST (https://csrc.nist.gov/projects/lightweight-cryptography, accessed on 10 May 2021). Many block ciphers have been proposed, and estimating the quantum resources for these block ciphers would make for interesting research. The result of FELICS competition (https://www.cryptolux.org/index.php/FELICS, accessed on 10 May 2021) [20] is another candidate. In this competition, a lot of lightweight symmetric key cryptography algorithms were evaluated on embedded processors. Comparing the performance of whether there is a relationship between quantum computers and embedded processors is an attractive research opportunity.

Author Contributions: Data curation, K.J.; Investigation, G.S., H.K. (Hyunjun Kim), H.K. (Hyeokdong Kwon) and S.U.; Software, K.J. and G.S.; Supervision, H.S.; Writing—original draft, K.J.; Writing—review and editing, K.J., H.K. (Hyunji Kim) 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, 90%) 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, 10%).

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

References

1. Atzori, L.; Iera, A.; Morabito, G. The Internet of Things: A survey. *Comput. Netw.* **2010**, *54*, 2787–2805.
2. Biryukov, A.; Perrin, L.P. State of the Art in Lightweight Symmetric Cryptography. 2017. Available online: https://eprint.iacr.org/2017/511 (accessed on 10 May 2021).
3. Bogdanov, A.; Knudsen, L.; Leander, G.; Paar, C.; Poschmann, A.; Robshaw, M.; Seurin, Y.; Vikkelsoe, C. PRESENT: An ultra-lightweight block cipher. In *International Workshop on Cryptographic Hardware and Embedded Systems*; Springer: Berlin/Heidelberg, Germany, 2007; Volume 4727, pp. 450–466. doi:10.1007/978-3-540-74735-2_31.
4. Banik, S.; Peyrin, T.; Sasaki, Y.; Sim, S.M.; Todo, Y. GIFT: A Small Present. In Proceedings of the International Conference on Cryptographic Hardware and Embedded Systems, Taipei, Taiwan, 25–28 September 2017; pp. 321–345. doi:10.1007/978-3-319-66787-4_16.
5. 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.
6. 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.
7. Langenberg, B.; Pham, H.; Steinwandt, R. Reducing the Cost of Implementing AES as a Quantum Circuit; Technical Report; Cryptology ePrint Archive, Report 2019/854; 2019. Available online: https://eprint.iacr.org/2019/854 (accessed on 10 May 2021).

8. Jaques, S.; Naehrig, M.; Roetteler, M.; Virdia, F. Implementing Grover oracles for quantum key search on AES and LowMC. In Proceedings of the Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, 10–14 May 2020; Springer: Berlin/Heidelberg, Germany 2020; pp. 280–310.
9. Anand, R.; Maitra, A.; Mukhopadhyay, S. Grover on SIMON. *Quantum Inf. Process.* **2020**, *19*, 1–17.
10. Jang, K.; Choi, S.; Kwon, H.; Seo, H. Grover on SPECK: Quantum Resource Estimates. Cryptology ePrint Archive, Report 2020/640. 2020. Available online: https://eprint.iacr.org/2020/640 (accessed on 10 May 2021).
11. Schlieper, L. In-place implementation of Quantum-Gimli. *arXiv* **2020**, arXiv:2007.06319.
12. Jang, K.; Choi, S.; Kwon, H.; Kim, H.; Park, J.; Seo, H. Grover on Korean Block Ciphers. *Appl. Sci.* **2020**, *10*, 6407.
13. Dasu, V.A.; Baksi, A.; Sarkar, S.; Chattopadhyay, A. LIGHTER-R: Optimized Reversible Circuit Implementation For SBoxes. In Proceedings of the 2019 32nd IEEE International System-on-Chip Conference (SOCC), Singapore, 3–6 September 2019; pp. 260–265. doi:10.1109/SOCC46988.2019.1570548320.
14. Steiger, D.S.; Häner, T.; Troyer, M. ProjectQ: An open source software framework for quantum computing. *Quantum* **2018**, *2*, 49.
15. Yang, G.; Zhu, B.; Suder, V.; Aagaard, M.; Gong, G. The SIMECK Family of Lightweight Block Ciphers. In *International Workshop on Cryptographic Hardware and Embedded Systems*; Springer: Berlin/Heidelberg, Germany, 2015; pp. 307–329. doi:10.1007/978-3-662-48324-4_16.
16. Jean, J.; Peyrin, T.; Sim, S.M.; Tourteaux, J. Optimizing Implementations of Lightweight Building Blocks. *IACR Trans. Symmetric Cryptol.* **2017**, *2017*, 130–168.
17. Bhattacharjee, D.; Chattopadhyay, A. Depth-Optimal Quantum Circuit Placement for Arbitrary Topologies. *arXiv* **2017**, arXiv:1703.08540.
18. Amento-Adelmann, B.; Grassl, M.; Langenberg, B.; Liu, Y.K.; Schoute, E.; Steinwandt, R. Quantum cryptanalysis of block ciphers: A case study. In Proceedings of the Poster at Quantum Information Processing QIP, Delft, The Netherlands, 15–19 January 2018.
19. 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.
20. Dinu, D.; Biryukov, A.; Großschädl, J.; Khovratovich, D.; Le Corre, Y.; Perrin, L. FELICS—Fair evaluation of lightweight cryptographic systems. In Proceedings of the NIST Workshop on Lightweight Cryptography, Gaithersburg, MD, USA, 20–21 July 2015; Volume 128.