applied sciences

MDPI

Article

Grover on Korean Block Ciphers

Kyoungbae Jang, Seungju Choi, Hyeokdong Kwon, Hyunji Kim, Jaehoon Park and Hwajeong Seo *

Division of IT Convergence Engineering, Hansung University, Seoul 02876, Korea; starj1234@hansung.ac.kr (K.J.); bookingstore3@hansung.ac.kr (S.C.); hyeok@hansung.ac.kr (H.K.); 1594012@hansung.ac.kr (H.K.); 20213201@hansung.ac.kr (J.P.)

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

Received: 1 August 2020; Accepted: 11 September 2020; Published: 14 September 2020

Abstract: The Grover search algorithm reduces the security level of symmetric key cryptography with n-bit security level to $O(2^{n/2})$. In order to evaluate the Grover search algorithm, the target block cipher should be efficiently implemented in quantum circuits. Recently, many research works evaluated required quantum resources of AES block ciphers by optimizing the expensive substitute layer. However, few works were devoted to the lightweight block ciphers, even though it is an active research area, nowadays. In this paper, we present optimized implementations of every Korean made lightweight block ciphers for quantum computers, which include HIGHT, CHAM, and LEA, and NSA made lightweight block ciphers, namely SPECK. Primitive operations for block ciphers, including addition, rotation, and exclusive-or, are finely optimized to achieve the optimal quantum circuit, in terms of qubits, Toffoli gate, CNOT gate, and X gate. To the best of our knowledge, this is the first implementation of ARX-based Korean lightweight block ciphers in quantum circuits.

Keywords: Grover search algorithm; block ciphers; quantum circuits; CHAM; HIGHT; LEA; SPECK

1. Introduction

As the Internet of Things (IoT) technology gets developed, a number of wearable and smart devices are gradually spreading through people's life [1]. Between these IoT devices, abundant data, from simple sensor data to even sensitive personal data, are being exchanged and processed. In order to protect these sensitive data, the exchange process of the data must be secured properly. To achieve the security, cryptographic algorithms must be applied to the data. However, applying the cryptographic algorithm requires resources, including computational power and memory. Most of the IoT devices do not have enough resources in order to apply cryptographic algorithms of conventional computers since most of them are equipped with low computational power and small memory footprint.

In order to resolve this hard condition, lightweight cryptography has been actively studied [2]. Unlike classical cryptography, a lightweight cryptography is designed for low-end devices. Most lightweight cryptography focuses on using resources efficiently to operate in the resource-constrained devices. However, as a cryptography, it is still important to maintain security even from the potential attacks using not only conventional computers, but also the upcoming quantum computers.

The advent of quantum computers and quantum algorithms has dramatically changed the cryptography community. The most well-known modern public key cryptography, such as RSA and Elliptic Curve Cryptography (ECC), are based on the difficulties of factorization and discrete algebra problem [3]. However, Shor's algorithm can perform prime number factorization efficiently, making RSA and ECC vulnerable [4].

In the field of symmetric key cryptography, the impact of quantum computers is not as critical as the case of the public key cryptography. The Grover search algorithm can be used to find the n-bit

secret key at the speed of $\sqrt{n}$, which is the most effective quantum attack method for block ciphers [5]. Applying the Grover search algorithm to block ciphers is the most efficient way to measure the security level of block ciphers against attacks from quantum computers. For this reason, not only the Grover's algorithm but also the cryptography must be implemented with a quantum circuit for cryptanalysis. Since the development of quantum computer is a rudimentary stage, finding the optimal quantum resource for the target algorithm is one of the most important research fields.

In order to estimate the quantum resources, a number of block cipher implementations have been investigated [6–9]. Grassl et al. estimated the quantum resource required for AES block cipher to apply the Grover search algorithm [6]. Afterward, Langenberg et al. and Jaques et al. found more optimal substitute layer design in quantum circuits than Grassl et al. [7,8]. Recently, Anand et al. implemented the SIMON in quantum gates [9], which is a block cipher developed by the National Security Agency (NSA) to support the hardware environment [10].

In this paper, we implemented SPECK and every Korean lightweight block cipher for the quantum resource estimation. To optimize the quantum circuit, functions, such as the key scheduling and the round functions, were optimized by simultaneously calculating results. In addition, the number of qubits was reduced by efficiently arranging the order of operations and the order of objects. Target ciphers are ARX-based structures designed with Addition, Rotation, and XOR operations.

1.1. Contribution

1.1.1. Optimized Implementation of ARX-Based Block Ciphers in Quantum Gates

Quantum gates for AES block ciphers have been actively studied. However, only a few works were devoted to lightweight ARX-based block ciphers. In this paper, we implemented four different ARX-based block ciphers and presented optimized implementations to reduce required qubits.

1.1.2. First Quantum Implementation of ARX-Based All Korean Block Ciphers and In-Depth Analysis

Korean block ciphers, such as LEA, HIGHT, and CHAM are efficiently implemented in quantum gates. This is the first implementation of Korean block ciphers in quantum gates. We analyzed the implementations and show the security against the quantum computer and their features.

1.1.3. Quantum Resource Estimation between Software-Oriented and Hardware-Oriented Block Ciphers

The architecture of block cipher is largely divided into software-oriented (i.e., SPECK) and hardware-oriented (i.e., SIMON). We compared both block ciphers in quantum gates and estimated the quantum security.

The remainder of this paper is organized as follows. In Section 2, target block ciphers, quantum implementations, and quantum algorithms are given. In Section 3, proposed quantum implementations for block ciphers are given. In Section 4, the evaluation of proposed quantum implementation is presented. Finally, Section 5 concludes the paper.

2. Related Works

2.1. Notation

In this paper, the following notations are commonly used, and Table 1 explains its meaning.

2.2. Target Block Ciphers

2.2.1. Hight

In CHES'06, ultra lightweight block cipher HIGHT was presented [11]. HIGHT is a 64-bit block cipher with 128-bit keys. Eight whitening keys and 128 subkeys, 8-bit respectively, are

generated from the 128-bit key. HIGHT block cipher performs 8-bit wise ARX operations and the encryption/decryption operation consists of 32 rounds. In each round, a 64-bit round key is required, and, in total, 2048-bit of round keys are needed to process a 64-bit block. The parameters for the HIGHT block cipher are shown in Table 2.

Table 1. Notations.

Notation	Meaning
K	Initial keywords
RK	Round key
$\oplus$	XOR operation
$\boxplus$	Modular addition operation
ROL_i	Rotation left operation (i-bit)
ROR_i	Rotation right operation (i-bit)

Table 2. Parameters of HIGHT.

Cipher	Block Size (bits)	Key Size (bits)	Word Size (bits)	Keywords	Rounds
HIGHT	64	128	8	16	32

The key schedule of HIGHT uses the initial key words K to generate whitening keys WK and round keys RK. The whitening key is used for initial conversion of plain text and final conversion for cipher text. The whitening key is generated as follows:

$$\begin{aligned} WK[i] &= K[i+12],\ 0 \le i \le 3 \\ WK[i] &= K[i-4],\quad 4 \le i \le 7 \end{aligned} \tag{1}$$

The connection polynomial of LFSR h is $x^7 + x^3 + 1$. The initial internal state value of h is $\delta_0 = (s_6, s_5, s_4, s_3, s_2, s_1, s_0) = (1, 0, 1, 1, 0, 1, 0)$. For $1 \le i \le 127$, δ_i is generated as follows and used for round key generation:

$$\begin{aligned} s_{i+6} &= s_{i+2} \oplus s_{i-1} \\ \delta_i &= (s_{i+6}, s_{i+5}, s_{i+4}, s_{i+3}, s_{i+2}, s_{i+1}, s_i) \end{aligned} \tag{2}$$

Round key RK is generated as follows:

$$\begin{aligned} &\text{for } i = 0 \text{ to } 7: \\ &\quad \text{for } j = 0 \text{ to } 7: \\ &\qquad RK[16 \cdot i + j] = K[j - i \bmod 8] \boxplus \delta_{16 \cdot i + j} \\ &\quad \text{for } j = 0 \text{ to } 7: \\ &\qquad RK[16 \cdot i + j + 8] = K[(j - i \bmod 8) + 8] \boxplus \delta_{16 \cdot i + j + 8} \end{aligned} \tag{3}$$

Before performing the round function, the initial conversion of plaintext T using four whitening keys WK is performed in the encryption process, as follows:

$$\begin{aligned} X_0[i] &= T[i], i = 1, 3, 5, 7 \\ X_0[0] &= T[0] \boxplus WK[0] \\ X_0[2] &= T[2] \oplus WK[1] \\ X_0[4] &= T[4] \boxplus WK[2] \\ X_0[6] &= T[6] \oplus WK[3] \end{aligned} \tag{4}$$

In the round function of HIGHT, auxiliary functions $F_0(X)$ and $F_1(X)$ using left rotation operation are used, and, for $1 \le i < 32$, the round function is shown in Equation (6):

$$\begin{aligned} F_0(X) &= \mathrm{ROL}_1(X) \oplus \mathrm{ROL}_2(X) \oplus \mathrm{ROL}_7(X) \\ F_1(X) &= \mathrm{ROL}_3(X) \oplus \mathrm{ROL}_4(X) \oplus \mathrm{ROL}_6(X) \end{aligned} \tag{5}$$

$$\begin{aligned} X_i[j] &= X_{i-1}[j-1], \quad j = 1,3,5,7 \\ X_i[0] &= X_{i-1}[7] \oplus (F_0(X_{i-1}[6]) \boxplus RK[4i-1]) \\ X_i[2] &= X_{i-1}[1] \boxplus (F_0(X_{i-1}[0]) \oplus RK[4i-4]) \\ X_i[4] &= X_{i-1}[3] \oplus (F_0(X_{i-1}[2]) \boxplus RK[4i-3]) \\ X_i[6] &= X_{i-1}[5] \boxplus (F_0(X_{i-1}[4]) \oplus RK[4i-2]) \end{aligned} \tag{6}$$

In the last round $i = 32$, the $F_1(X)$ is also used, and the output positions of the input values are not changed as follows:

$$\begin{aligned} X_{32}[i] &= X_{31}[i], \quad i = 0,2,4,6 \\ X_{32}[1] &= X_{31}[1] \boxplus (F_1(X_{31}[0]) \oplus RK[124]) \\ X_{32}[3] &= X_{31}[3] \oplus (F_0(X_{31}[2]) \boxplus RK[125]) \\ X_{32}[5] &= X_{31}[5] \boxplus (F_1(X_{31}[4]) \oplus RK[126]) \\ X_{32}[7] &= X_{31}[7] \oplus (F_0(X_{31}[6]) \boxplus RK[127]) \end{aligned} \tag{7}$$

After executing all round functions, the ciphertext C is generated by the final conversion using the whitening key, and the final conversion is shown in Equation (8):

$$\begin{aligned} C[i] &= X_{32}[i], i = 1,3,5,7 \\ C[0] &= X_{32}[0] \boxplus WK[4] \\ C[2] &= X_{32}[2] \oplus WK[5] \\ C[4] &= X_{32}[4] \boxplus WK[6] \\ C[6] &= X_{32}[6] \oplus WK[7] \end{aligned} \tag{8}$$

Key schedule and round function structure of HIGHT are shown in Figure 1.

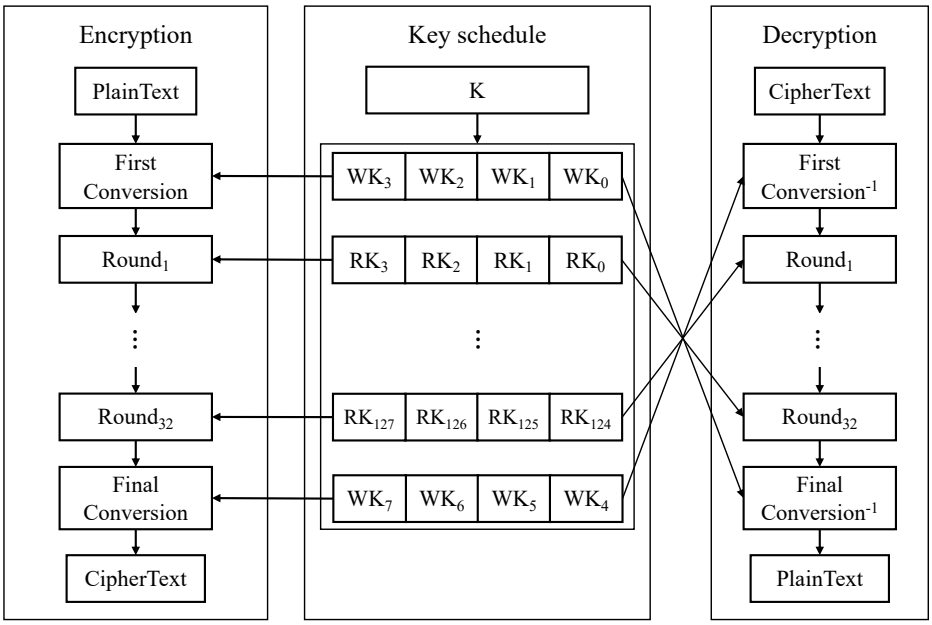

Figure 1. Key schedule and round function of HIGHT.

2.2.2. CHAM

In ICISC'17, a family of lightweight block ciphers CHAM was announced by the Attached Institute of ETRI [12]. The family consists of three ciphers: including CHAM-64/128, CHAM-128/128, and CHAM-128/256. The CHAM block ciphers are of the generalized 4-branch Feistel structure based on ARX operations.

In ICISC'19, the revised version of CHAM block cipher was presented [13]. In order to prevent new related-key differential characteristics and differentials of CHAM using a SAT solver, the numbers of rounds of CHAM-64/128, CHAM-128/128, and CHAM-128/256 are increased from 80 to 88, 80 to 112, and 96 to 120, respectively. Table 3 shows the parameters of the block ciphers CHAM.

Table 3. Parameters of CHAM.

Cipher	Block Size (bits)	Key Size (bits)	Word Size (bits)	Keywords	Rounds
CHAM-64/128	64	128	16	8	80
CHAM-128/128	128	128	32	4	80
CHAM-128/256	128	256	32	8	96

By performing the key schedule on the initial key words $K = \{K[0], K[1], \ldots, K[m-1]\}$, generates round key $RK = \{RK[0], RK[1], \ldots, RK[2m-1]\}$ for use in the i-th round for $0 \leq i < r$. The key schedule is shown in Equation (9) below, where $0 \leq i < m$:

$$\begin{aligned} RK[i] &= K[i] \oplus \mathrm{ROL}_1(K[i]) \oplus \mathrm{ROL}_8(K[i]) \\ RK[(i+m) \oplus 1] &= K[i] \oplus \mathrm{ROL}_1(K[i]) \oplus \mathrm{ROL}_{11}(K[i]) \end{aligned} \tag{9}$$

If the round key RK is generated through key scheduling, the plaintext $T = \{X_0[0], X_0[1], X_0[2], X_0[3]\}$ is encrypted by repeating the round function in the following two cases.

If i from the i-th round for $0 \leq i < r$ is odd, the upper function is called. If the i is even, then the lower function is called:

$$\begin{aligned} X_{i+1}[j] &= X_i[j+1], (0 \leq j \leq 2) \\ X_{i+1}[3] &= \mathrm{ROL}_8((X_i[0] \oplus i) \boxplus (\mathrm{ROL}_1(X_i[1]) \oplus RK[i \bmod 2m])) \end{aligned} \tag{10}$$

In the rest of the cases, the following function applies:

$$\begin{aligned} X_{i+1}[j] &= X_i[j+1], (0 \leq j \leq 2) \\ X_{i+1}[3] &= \mathrm{ROL}_1((X_i[0] \oplus i) \boxplus (\mathrm{ROL}_8(X_i[1]) \oplus RK[i \bmod 2m])) \end{aligned} \tag{11}$$

Key schedule and round function structure of CHAM are shown in Figures 2 and 3.

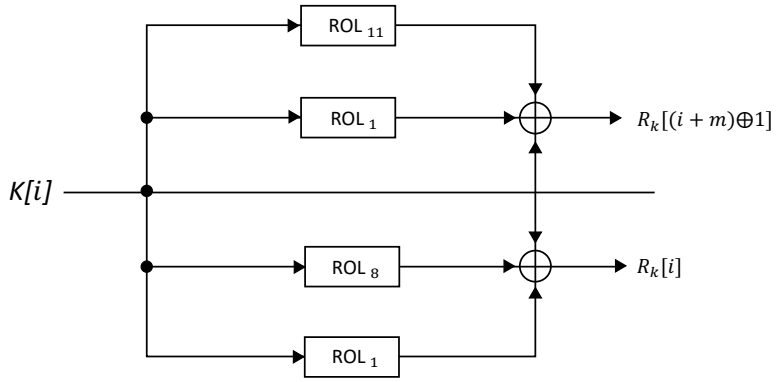

Figure 2. Key schedule of CHAM.

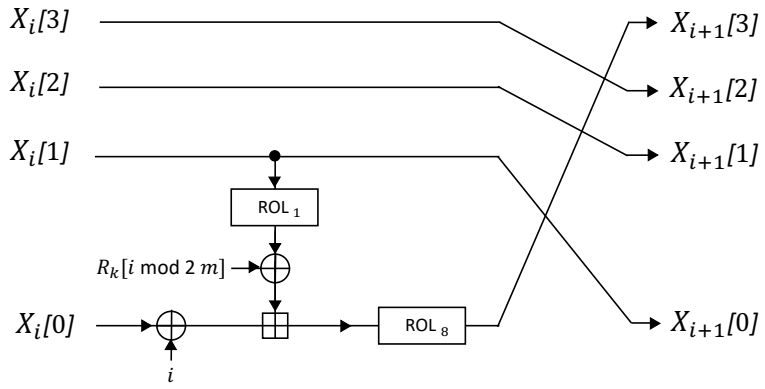

Figure 3. Round function of CHAM.

2.2.3. LEA

In WISA'13, a lightweight block cipher LEA was presented [14]. LEA is a 128-bit block cipher supporting three key lengths, i.e., 128-bit, 192-bit, and 256-bit. LEA–128, LEA-192, and LEA-256 require 24, 28, and 32 rounds, respectively. In each round, 192-bit round keys are required. The word size is 32-bit and primitive operations are 32-bit wise addition, rotation, and exclusive-or. The parameters for the LEA block cipher are shown in Table 4.

Table 4. Parameters of LEA.

Cipher	Block Size (bits)	Key Size (bits)	Word Size (bits)	Keywords	Rounds
LEA-128	128	128	32	4	24
LEA-192	128	192	32	6	28
LEA-256	128	256	32	8	32

The key schedule of the LEA uses constant value δ representing the ASCII codes of 'L,' 'E,' and 'A', and the key schedule is slightly different depending on the parameters. The key schedule of LEA-128 for $0 \leq i < 24$ is as follows:

$$\begin{aligned} \delta[0] &= 0xc3efe9db, \quad \delta[1] = 0x44626b02 \\ \delta[2] &= 0x79e27c8a, \quad \delta[3] = 0x78df30ec \\ \delta[4] &= 0x715ea49e, \quad \delta[5] = 0xc785da0a \\ \delta[6] &= 0xe04ef22a, \quad \delta[7] = 0xe5c40957 \end{aligned} \tag{12}$$

$$\begin{aligned} K[0] &= \mathrm{ROL}_1(K[0] \boxplus \mathrm{ROL}_i(\delta[i \bmod 4])) \\ K[1] &= \mathrm{ROL}_3(K[1] \boxplus \mathrm{ROL}_{i+1}(\delta[i \bmod 4])) \\ K[2] &= \mathrm{ROL}_6(K[2] \boxplus \mathrm{ROL}_{i+2}(\delta[i \bmod 4])) \\ K[3] &= \mathrm{ROL}_{11}(K[3] \boxplus \mathrm{ROL}_{i+3}(\delta[i \bmod 4])) \\ RK_i &= (K[0], K[1], K[2], K[1], K[3], K[1]) \end{aligned} \tag{13}$$

Unlike the key schedule, the round function is performed as follows for $0 \leq i < r$ regardless of the parameter for the plaintext $P = \{X_0[0], X_0[1], X_0[2], X_0[3]\}$:

$$\begin{aligned} X_{i+1}[0] &= \mathrm{ROL}_9((X_i[0] \oplus RK_i[0]) \boxplus (X_i[1] \oplus RK_i[1])) \\ X_{i+1}[1] &= \mathrm{ROL}_5((X_i[1] \oplus RK_i[2]) \boxplus (X_i[2] \oplus RK_i[3])) \\ X_{i+1}[2] &= \mathrm{ROL}_3((X_i[2] \oplus RK_i[4]) \boxplus (X_i[3] \oplus RK_i[5])) \\ X_{i+1}[3] &= X_i[0] \end{aligned} \tag{14}$$

Key schedule and round function structure of LEA are shown in Figures 4 and 5.

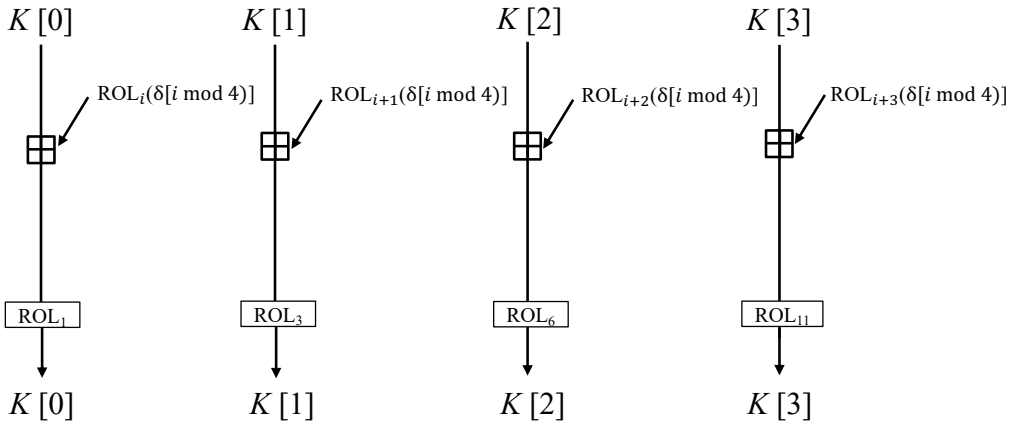

Figure 4. Key schedule of LEA-128.

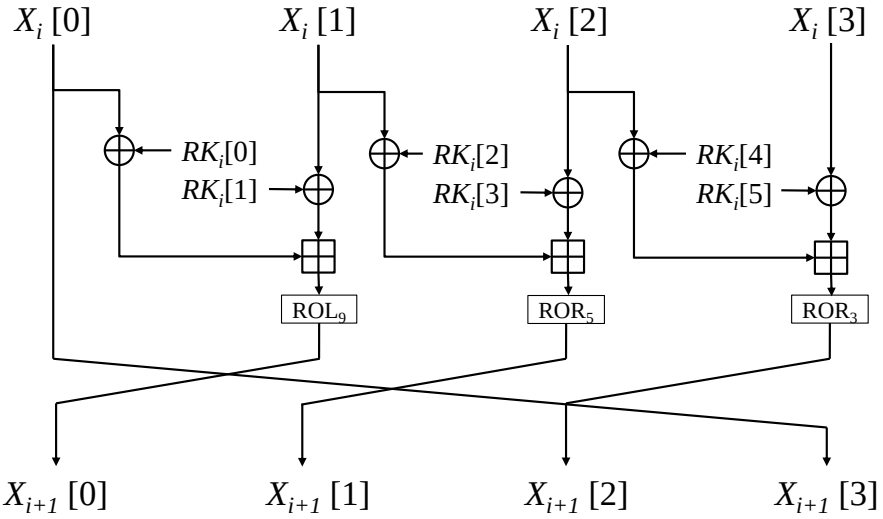

Figure 5. i-th round function of LEA.

2.2.4. SPECK

In 2013, NSA released two lightweight block ciphers, SPECK and SIMON, for the low-end devices [10]. SPECK has been implemented in order to perform efficiently in the software environment. The SIMON, on the other hand, has been implemented and optimized for hardware. Although these two ciphers are considered sisters, SIMON has already been designed as quantum circuit [9]. For this reason, this paper only covers quantum circuit implementation and optimization for SPECK.

SPECK takes the form of block cipher and is implemented with an ARX structure that consists of simple operations such as addition, rotation, and bit-wise exclusive operation. SPECK is also implemented in various parameters to provide different levels of security, which are shown in Table 5.

Table 5. Parameters of SPECK.

Block Size (bits)	Key Size (bits)	Word Size (bits)	Keywords	Rounds
32	64	16	4	22
48	72, 96	24	3, 4	22, 23
64	96, 128	32	3, 4	26, 27
96	96, 144	48	2, 3	28, 29
128	128, 192, 256	64	2, 3, 4	32, 33, 34

A key schedule gets processed in order to generate a round key RK, which will be used in each round. The initial key words are $K = \{K[0], l[0], \ldots, l[m-2]\}$, and the following key schedule is performed for $0 \leq i < r$, and the new key value $RK = \{K[0], K[1], \ldots, K[r-1]\}$ is used as the round key. α and β are 7 and 2 when the size of the block is 32-bit, respectively. For the rest of the block size, α and β are 8 and 3, respectively:

$$\begin{aligned} l[i+m-1] &= (K[i] \boxplus \mathrm{ROR}_\alpha(l[i])) \oplus i \\ K[i+1] &= \mathrm{ROL}_\beta(K[i]) \oplus l[i+m-1] \end{aligned} \tag{15}$$

During the round function of SPECK with the block size of $2n$, for $0 \leq i < r$, a $2n$-bit ciphertext word is generated using an n-bit round key for a $2n$-bit ($X[0]$, $X[1]$) input word:

$$\begin{aligned} X[1] &= (\mathrm{ROR}_\alpha(X[1]) \boxplus X[0]) \oplus RK[i] \\ X[0] &= \mathrm{ROL}_\beta(X[0]) \oplus X[1] \end{aligned} \tag{16}$$

Key schedule and encryption of SPECK in case of m = 2 are shown in Figure 6.

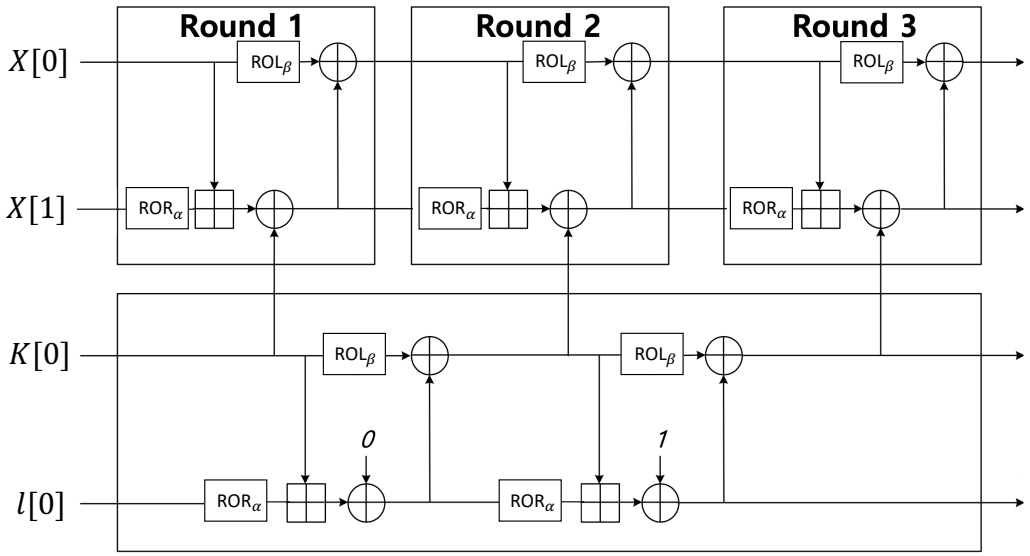

Figure 6. Overview of SPECK($m = 2$) block cipher: key schedule and encryption.

2.3. Quantum Implementations and Algorithms

2.3.1. Quantum Gates

Quantum computers have several gates that can emulate the classical gates. The two most representative gates are CNOT and Toffoli gates. The CNOT gate performs a NOT gate operation on the second qubit when the first input qubit of the two input qubits is set as one. The NOT gate (i.e., X gate) inverts the state of the qubit. This gate performs the same role as the addition operation on the binary field. The circuit configuration is shown on the left side of Figure 7. The Toffoli gate takes three qubits as input. When the first and second qubits are set to one, the gate performs a NOT gate operation on the last qubit. This serves as an AND operation on the binary field. The circuit configuration for Toffoli gate is shown on the right side of Figure 7.

A lot of works have been done on the addition operation in quantum computers [15–18]. Among them, Cuccaro et al. suggested the riple-carry addition circuit to achieve the most optimal design by using only one ancillary qubit [15]. For n-bit addition operation, $2n + 2$ qubits, $2n$ Toffoli gates, and $4n$ CNOT gates are used.

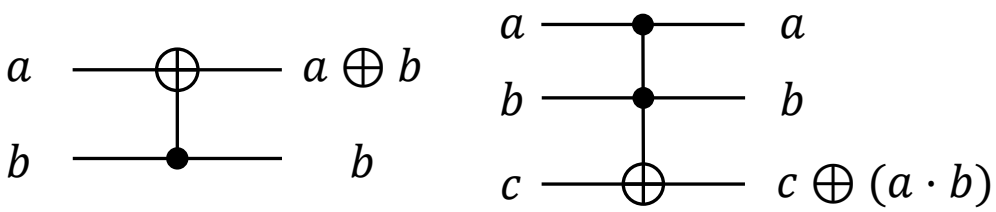

Figure 7. Circuit configuration of the (left) CNOT and (right) Toffoli gate.

2.3.2. Grover Search Algorithm

The Grover search algorithm is a quantum algorithm that finds specific data for n unsorted data. The classic method requires $O(2^n)$ searches in brute force attack. However, this can be found within $O(2^{n/2})$ times with the Grover search algorithm. The Grover search algorithm consists of an oracle function and a diffusion operator, as shown in Figure 8.

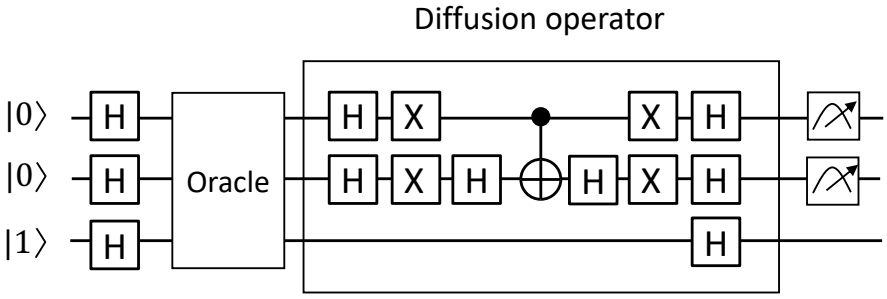

Figure 8. Grover search algorithm.

The oracle function $f(x)$ returns 1 if input x is the solution to the search. Otherwise, it returns 0. When $f(x) = 1$, the sign of the state x is flipped. It then proceeds to the diffusion operator step, which increases the amplitude of the solution. The searching step is as follows: First, the average amplitude is calculated for all data. Second, the difference between the amplitude and the average amplitude of each data are calculated. If the answer to find in the 2-bit input is 10, the status after these two steps is as shown in Figure 9.

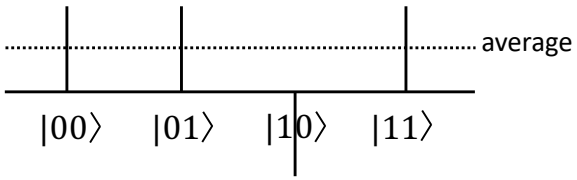

Figure 9. Condition after the oracle of Grover algorithm.

After performing the oracle function, the amplitude of the solution has a different sign from other amplitudes. The difference from the average amplitude increases and the difference between the non-answer amplitudes decreases. The Grover search algorithm increases the amplitude probability of the solution by repeating the oracle function and diffusion operators. The status after diffusion operations is given in Figure 10.

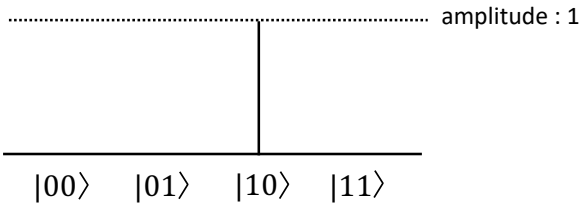

Figure 10. Condition after diffusion of the Grover algorithm.

2.3.3. Previous Quantum Implementations

Advanced Encryption Standard (AES) is the international block cipher standard [19] and the block cipher is widely used in practice, such as database encryption and network security. For this reason, a number of works focused on quantum resource estimation for the AES block cipher.

Grassl et al. have presented first quantum circuits to implement an exhaustive key search for the AES and analyzed the quantum resources required to carry out such attacks [6]. All three variants of AES (key size 128, 192, and 256 bit) are implemented for the Grover's quantum algorithm with optimal operations for each layer.

Langenberg et al. presented new quantum circuits for all three AES key lengths. For AES-128, the number of Toffoli gates can be reduced by more than 88% compared to Grassl et al.'s estimates [6], while simultaneously reducing the number of qubits [20].

Previous works of AES have derived the full gate cost by analyzing quantum circuits for the cipher but focused on minimizing the number of qubits. In [8], they have studied the cost of quantum key search attacks under a depth restriction and introduced techniques that reduce the oracle depth, even if it requires more qubits. As part of this work, they released Q# implementations of the full Grover oracle for AES-128, AES-192, and AES-256, including unit tests and code to reproduce their quantum resource estimates.

In [9], the reversible implementation of SIMON block cipher was presented. This block cipher is hardware-oriented suggested by NSA. They have successfully implemented the Grover oracle and Grover diffusion for key search. They have estimated the resources in terms of CNOT, Toffoli, and X gates, which are required to carry out the quantum attack on the block cipher.

3. Proposed Method

In this chapter, we describe the optimized quantum circuit for applying the Grover's algorithm to HIGHT, CHAM, LEA, and SPECK block ciphers.

All block ciphers implemented in this paper are composed of modular addition, rotation, and XOR operations based on ARX. In the quantum circuit, the rotation operation can be designed without using any gates by changing the order of qubits. Given $a = \{a_0, a_1, a_2, a_3\}$ and $b = \{b_0, b_1, b_2, b_3\}$, the operation $a \ggg 1$ is performed. Then, the XOR operation $b = b \oplus a$ is performed on the elements of b and the elements of a changed in order by rotation. In this case, since a is $\{a_1, a_2, a_3, a_0\}$ after $a \ggg 1$, it is completed by performing CNOT(a_1, b_0), CNOT(a_2, b_1), CNOT(a_3, b_2), and CNOT(a_0, b_3). In this way, there is no cost for the rotation operation, and the XOR operation can be performed simply with a CNOT gate.

However, for addition operation, Toffoli gates are required, so a more complex circuit is needed than the previous two cases. We used the method of Cuccaro et. al's ripple carry addition [15]. In order to use the ripple-carry addition circuit, not only the qubits of the calculation target, but also two additional qubits for carry calculation are required. Since the addition of block cipher is the modular addition, the value of highest carry qubit can be ignored. Therefore, the addition operation can be performed only with the initial carry qubit of c_0. Since this c_0 qubit is initialized to 0 after addition operation, it can be recycled at any time.

The block cipher is largely divided into two steps, including key schedule and round function. In order to reduce the required qubit, an on-the-fly approach is utilized.

3.1. HIGHT

In this section, we describe the proposed HIGHT block cipher in the quantum circuit to apply the Grover search algorithm.

3.1.1. Key Schedule

In order to reduce the use of qubits in HIGHT, rather than generating all the round keys in advance, the key schedule is performed simultaneously during the round. In the key schedule, the $\delta_0 = \{s_6, s_5, s_4, s_3, s_2, s_1, s_0\}$ is used first, and then a new δ_i must be generated for $1 \leq i \leq 127$. At this time, the number of qubits can be saved by recycling the existing qubits. For example, when creating a δ_1 in Equation (2), a new s_7 is generated, and the s_0 is not included in new δ_i. Therefore, we do not allocate a new qubit for s_7, but operate on the existing s_0 and treat it as a new value. The detailed process for this is shown in Algorithm 1.

Algorithm 1 Generate δ_i of HIGHT

Input: $\delta_{i-1} = \{s_{(i+5)\%7}, s_{(i+4)\%7}, s_{(i+3)\%7}, s_{(i+2)\%7}, s_{(i+1)\%7}, s_{i\%7}, s_{(i-1)\%7}\}$
Output: δ_i
1: $s_{(i-1)\%7} \leftarrow \text{CNOT}(s_{(i+2)\%7}, s_{(i-1)\%7})$
2: **return** $\delta_i = \{s_{(i-1)\%7}, s_{(i+5)\%7}, s_{(i+4)\%7}, s_{(i+3)\%7}, s_{(i+2)\%7}, s_{(i+1)\%7}, s_{i\%7}\}$

As mentioned earlier, to reduce the number of qubits, the key schedule is performed simultaneously with the function of the round. Rather than generating all round keys at once, the key schedule is performed in units of 16 keywords. Instead of generating a round key in new qubits, the operation is performed on the initial key $K = \{K[0], K[1], \ldots, K[15]\}$ to convert it to round key $RK = \{RK[0], RK[1], \ldots, RK[15]\}$. The detailed process for this is described in Algorithm 2.

Algorithm 2 i-th key schedule of HIGHT

Input: $\delta_{16\cdot(i-1)}$, K
Output: K
1: **for** $j = 0$ to 7 **do**
2: Generate $\delta_{16\cdot(i-1)+j}$
3: $RK[16 \cdot (i-1) + j] \leftarrow \text{ADD}(\delta_{16\cdot(i-1)+j}, K[j - i + 1 \bmod 8])$
4: Use $RK[16 \cdot (i-1) + j]$ in round function
5: Reverse : $\text{ADD}(\delta_{16\cdot(i-1)+j}, K[j - i + 1 \bmod 8])$
6: **end for**
7: **for** $j = 0$ to 7 **do**
8: Generate $\delta_{16\cdot(i-1)+j+8}$
9: $RK[16 \cdot (i-1) + j + 8] \leftarrow \text{ADD}(\delta_{16\cdot(i-1)+j+8}, K[(j - i + 1 \bmod 8) + 8])$
10: Use $RK[16 \cdot (i-1) + j + 8]$ in round function
11: Reverse : $\text{ADD}(\delta_{16\cdot(i-1)+j+8}, K[(j - i + 1 \bmod 8) + 8])$
12: **end for**
13: **return** $K = \{K[0], K[1], \ldots, K[15]\}$

The RK of 16 keywords is used for 4 rounds. After use, RK is initialized to K by reverse operation. In addition, again, the key schedule is performed to generate the round key $RK = \{RK[16], RK[17], \ldots, RK[31]\}$ to be used for the next four rounds.

3.1.2. Round Function

In HIGHT, before proceeding to the round function, the plaintext T is initially converted using the whitening key as shown in Equation (4). The quantum circuit design for this is in Algorithm 3.

Algorithm 3 Initial conversion of HIGHT

Input: T, K
Output: X_0
1: $X_0[i] \leftarrow T[i], \quad i = 1, 3, 5, 7$
2: $X_0[0] \leftarrow$ADD$(K[12], T[0])$
3: $X_0[2] \leftarrow$CNOT$(K[13], T[2])$
4: $X_0[4] \leftarrow$ADD$(K[14], T[4])$
5: $X_0[6] \leftarrow$CNOT$(K[15], T[6])$
6: **return** $X = \{X_0[0], X_0[1], \ldots, X_0[7]\}$

We optimized the round function by properly specifying the operation target and order in the round function. The detailed process for this is described in Algorithm 4 and explained in detail.

Algorithm 4 Round function of HIGHT

Input: Round i, X_{i-1}, RK
Output: X_i
First, operation:
    $X_{i-1}[0] \leftarrow F_0(X_{i-1}[0])$
    $X_{i-1}[0] \leftarrow$ CNOT$(RK[4i-4], X_{i-1}[0])$
    $X_i[2] \leftarrow$ ADD$(X_{i-1}[0], X_{i-1}[1])$
    $X_{i-1}[0] \leftarrow$ CNOT$(RK[4i-4], X_{i-1}[0])$ (reverse)
    $X_{i-1}[0] \leftarrow F_0$_reverse$(X_{i-1}[0])$
Second operation:
    $X_{i-1}[2] \leftarrow F_0(X_{i-1}[2])$
    $X_{i-1}[2] \leftarrow$ ADD$(RK[4i-3], X_{i-1}[2])$
    $X_i[4] \leftarrow$ CNOT$(X_{i-1}[2], X_{i-1}[3])$
    $X_{i-1}[2] \leftarrow$ ADD$(RK[4i-3], X_{i-1}[2])$ (reverse)
    $X_{i-1}[2] \leftarrow F_0$_reverse$(X_{i-1}[2])$
Third operation:
    $X_{i-1}[4] \leftarrow F_0(X_{i-1}[4])$
    $X_{i-1}[4] \leftarrow$ CNOT$(RK[4i-2], X_{i-1}[4])$
    $X_i[6] \leftarrow$ ADD$(X_{i-1}[4], X_{i-1}[5])$
    $X_{i-1}[4] \leftarrow$ CNOT$(RK[4i-2], X_{i-1}[4])$ (reverse)
    $X_{i-1}[4] \leftarrow F_0$_reverse$(X_{i-1}[4])$
Fourth operation:
    $X_{i-1}[6] \leftarrow F_0(X_{i-1}[6])$
    $X_{i-1}[6] \leftarrow$ ADD$(RK[4i-1], X_{i-1}[6])$
    $X_i[0] \leftarrow$ CNOT$(X_{i-1}[6], X_{i-1}[7])$
    $X_{i-1}[6] \leftarrow$ ADD$(RK[4i-1], X_{i-1}[6])$ (reverse)
    $X_{i-1}[6] \leftarrow F_0$_reverse$(X_{i-1}[6])$
Last operation:
    $X_i[j] = X_{i-1}[j-1], \quad j = 1, 3, 5, 7$
return $X_i = \{X_i[0], X_i[1], \ldots, X_i[7]\}$

In particular, through the optimization of the auxiliary function F_0 and F_1, no additional qubit was used, and the use of the CNOT gate was also reduced. First, since the RK must be used immediately after generation, and then returned to the initial K by a reverse operation, the operation of the round function proceeds in the order of generation of the RK.

Second, in order not to use additional qubits, the new X_i must operate on the existing X_{i-1}. However, if you look at the first and third lines of Equation (6) and second line of Algorithm 4, you can see that the value of $X_{(i-1)}[0]$, which is first calculated in the round function, is used again. Thus, $X_{i-1}[0]$ value should not be changed. However, $X_{i-1}[1]$ is no longer used after calculation. Therefore, by calculating the value of $X_i[2]$ in $X_{i-1}[1]$, no additional qubits were required. The rest of the round function follows the similar procedure. The specific process is presented in Algorithm 4.

The auxiliary functions F_0 and F_1 rotate the input value and then XOR-ing them all. Since input value is 8-bit, if qubits for the generated value are allocated and then XOR-ing the rotation value of input, this requires $8 \times 3 = 24$ CNOT gates and 8 new qubits. We efficiently optimized this auxiliary function. The desired output was generated by mixing the input values without allocating the qubits for the outputs of F_0 and F_1. As a result, 21 CNOT gates in F_0 and 24 CNOT gates in F_1 were implemented without additional qubits. The detailed process for F_0 and F_1 functions is presented in Algorithms 5 and 6.

Algorithm 5 $F_0(X[i])$ of HIGHT

Input: $X[i]$
Output: $X_{new}[i]$
1: CNOT($X[i][4], X[i][5]$), CNOT($X[i][3], X[i][4]$)
2: CNOT($X[i][2], X[i][3]$), CNOT($X[i][2], X[i][0]$)
3: CNOT($X[i][4], X[i][2]$), CNOT($X[i][5], X[i][2]$)
4: $X_{new}[i][4] \leftarrow X[i][2]$
5: CNOT($X[i][7], X[i][5]$)
6: $X_{new}[i][6] \leftarrow X[i][5]$
7: CNOT($X[i][6], X[i][4]$)
8: $X_{new}[i][5] \leftarrow X[i][4]$
9: CNOT($X[i][0], X[i][3]$), CNOT($X[i][1], X[i][3]$)
10: $X_{new}[i][2] \leftarrow X[i][3]$
11: CNOT($X[i][6], X[i][1]$), CNOT($X[i][7], X[i][1]$)
12: $X_{new}[i][0] \leftarrow X[i][1]$
13: CNOT($X[i][7], X[i][0]$)
14: $X_{new}[i][1] \leftarrow X[i][0]$
15: CNOT($X[i][5], X[i][6]$), CNOT($X[i][4], X[i][6]$)
16: CNOT($X[i][3], X[i][6]$), CNOT($X[i][1], X[i][6]$)
17: $X_{new}[i][7] \leftarrow X[i][6]$
18: CNOT($X[i][6], X[i][7]$), CNOT($X[i][5], X[i][7]$)
19: CNOT($X[i][1], X[i][7]$), CNOT($X[i][0], X[i][7]$)
20: $X_{new}[i][3] \leftarrow X[i][7]$
21: **return** $X_{new} = \{X_{new}[0], X_{new}[1], \ldots, X_{new}[7]\}$

In the round function, after using F_0 and F_1, it should be returned to the original input value. For this, the F_0_reverse and F_1_reverse functions should be executed in reverse order of calculation of the F_0 and F_1 functions.

Algorithm 6 $F_1(X)$ of HIGHT

Input: X

Output: X_{new}

1: CNOT($X[3], X[4]$), CNOT($X[1], X[4]$)
2: $X_{new}[7] \leftarrow X[4]$
3: CNOT($X[2], X[3]$), CNOT($X[0], X[3]$)
4: $X_{new}[6] \leftarrow X[3]$
5: CNOT($X[1], X[2]$), CNOT($X[7], X[2]$)
6: $X_{new}[5] \leftarrow X[2]$
7: CNOT($X[0], X[1]$), CNOT($X[6], X[1]$)
8: $X_{new}[4] \leftarrow X[1]$
9: CNOT($X[7], X[0]$), CNOT($X[5], X[0]$)
10: $X_{new}[3] \leftarrow X[0]$
11: CNOT($X[6], X[7]$), CNOT($X[4], X[7]$)
12: CNOT($X[3], X[7]$), CNOT($X[2], X[7]$)
13: CNOT($X[0], X[7]$), CNOT($X[5], X[7]$)
14: $X_{new}[2] \leftarrow X[7]$
15: CNOT($X[0], X[6]$), CNOT($X[7], X[6]$)
16: CNOT($X[4], X[6]$), CNOT($X[1], X[6]$)
17: $X_{new}[1] \leftarrow X[6]$
18: CNOT($X[7], X[5]$), CNOT($X[6], X[5]$)
19: CNOT($X[3], X[5]$), CNOT($X[0], X[5]$)
20: $X_{new}[0] \leftarrow X[5]$
21: **return** $X_{new} = \{X_{new}[0], X_{new}[1], \ldots, X_{new}[7]\}$

In the last 32nd round, the round function from Equation (7) is performed. Compared to the previous round functions, shown in Algorithm 4, the biggest difference is that the last round uses auxiliary function, F_1, while calculating. Therefore, we have omitted the quantum circuit design for the last round since the overall structure is almost identical.

After all the round functions have been performed, the final conversion from Equation (8) gets performed and encryption gets completed. The quantum circuit design for final conversion is also omitted because it is almost identical to the initial conversion from Algorithm 3. The overall quantum circuit for HIGHT is shown in Figure 11.

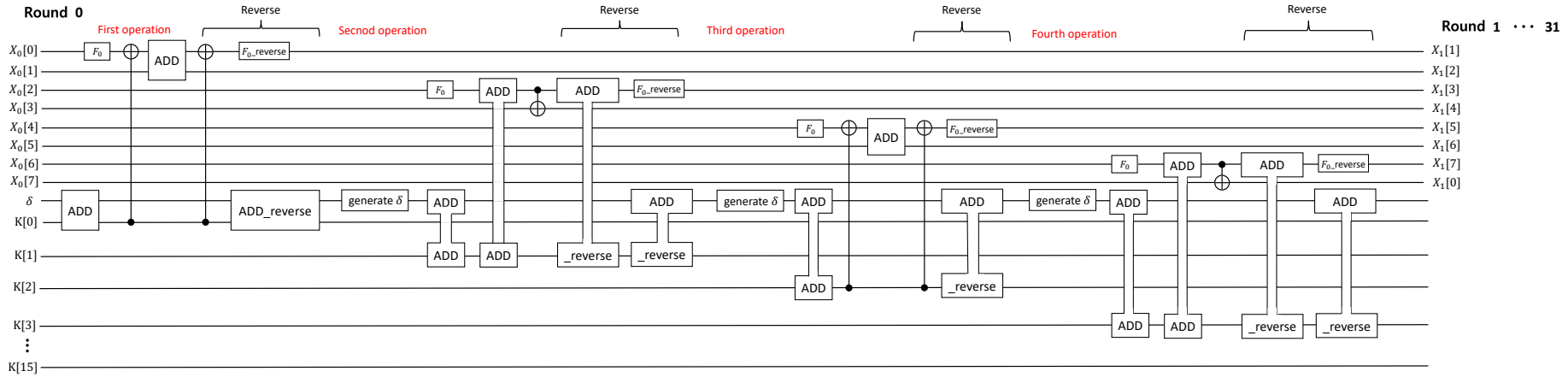

Figure 11. Quantum circuit for HIGHT.

3.2. CHAM

In this section, we describe the proposed CHAM block cipher in the quantum circuit to apply the Grover search algorithm.

3.2.1. Key Schedule

The key schedule of CHAM has the same structure as the auxiliary functions F_0 and F_1 of HIGHT. It performs ROL_1, ROL_8 or ROL_1, ROL_{11} operations on initial keywords K, and XOR-ing them all on the K. Therefore, the method of optimizing the auxiliary function of HIGHT was applied to the key schedule of CHAM. The CNOT gate is performed between the qubits storing K to generate an output in which both the K and rotation values of K are XOR-ed. In the key schedule from Equation (9), the first line operation is well optimized without any additional qubits. The detailed process of the first line in Equation (9) is described in Algorithm 7. Not as much as first line operation, the second line operation in Equation (9) has been optimized for using three temporary qubits. The detailed process of key schedule is described in Algorithm 8.

After performing Algorithms 7 or 8 to use RK generated from K, K is required again when generating another round key RK. Therefore, after using the round key, it is necessary to perform a reverse key schedule operation as done in HIGHT.

Algorithm 7 $RK[i]$ key schedule of CHAM

Input: $K[i]$
Output: $RK[i]$

1: $\mathrm{CNOT}(K[i][0], K[i][8]), \mathrm{CNOT}(K[i][1], K[i][9])$
2: $\mathrm{CNOT}(K[i][2], K[i][10]), \mathrm{CNOT}(K[i][3], K[i][11])$
3: $\mathrm{CNOT}(K[i][4], K[i][12]), \mathrm{CNOT}(K[i][5], K[i][13])$
4: $\mathrm{CNOT}(K[i][6], K[i][14]), \mathrm{CNOT}(K[i][15], K[i][6]$
5: **for** $j = 0$ to 5 **do**
6: $\quad \mathrm{CNOT}(K[i][j+9], K[i][j]), RK[i][j+1] \leftarrow K[i][j]$
7: **end for**
8: $\mathrm{CNOT}(K[i][7], K[i][6]), RK[i][7] \leftarrow K[i][6]$
9: $\mathrm{CNOT}(K[i][8], K[i][15]), RK[i][0] \leftarrow K[i][15]$
10: $\mathrm{CNOT}(K[i][8], K[i][7]), RK[i][8] \leftarrow K[i][7]$
11: **for** $j = 0$ to 6 **do**
12: $\quad \mathrm{CNOT}(K[i][j], K[i][j+8]), RK[i][j+9] \leftarrow K[i][j+8]$
13: **end for**
14: **return** $RK[i] = \{RK[i][0], RK[i][1], \ldots, RK[i][15]\}$

Algorithm 8 $RK[i+8\oplus 1]$ key schedule of CHAM

Input: $K[i]$
Output: $RK[i+8\oplus 1]$
1: CNOT($K[i][15]$,temp$[0]$), CNOT($K[i][9]$,temp$[0]$)
2: CNOT($K[i][5]$,temp$[1]$), CNOT($K[i][15]$,temp$[1]$)
3: CNOT($K[i][4]$,temp$[2]$), CNOT($K[i][10]$,temp$[2]$)
4: CNOT($K[i][14], K[i][15]$), CNOT($K[i][4], K[i][15]$
5: CNOT($K[i][9], K[i][4]$), CNOT($K[i][3], K[i][4]$
6: $RK[i+8\oplus 1][15] \leftarrow K[i][15]$, $RK[i+8\oplus 1][4] \leftarrow K[i][4]$
7: CNOT($K[i][14], K[i][9]$), CNOT($K[i][8], K[i][9]$
8: CNOT($K[i][13], K[i][14]$), CNOT($K[i][3], K[i][14]$
9: $RK[i+8\oplus 1][9] \leftarrow K[i][9]$, $RK[i+8\oplus 1][14] \leftarrow K[i][14]$
10: CNOT($K[i][8], K[i][3]$), CNOT($K[i][2], K[i][3]$
11: CNOT($K[i][13], K[i][8]$), CNOT($K[i][7], K[i][8]$
12: $RK[i+8\oplus 1][3] \leftarrow K[i][3]$, $RK[i+8\oplus 1][8] \leftarrow K[i][8]$
13: CNOT($K[i][12], K[i][13]$), CNOT($K[i][2], K[i][13]$
14: CNOT($K[i][7], K[i][2]$), CNOT($K[i][1], K[i][2]$
15: $RK[i+8\oplus 1][13] \leftarrow K[i][13]$, $RK[i+8\oplus 1][2] \leftarrow K[i][2]$
16: CNOT($K[i][12], K[i][7]$), CNOT($K[i][6], K[i][7]$
17: CNOT($K[i][1], K[i][12]$), CNOT($K[i][1], K[i][12]$
18: $RK[i+8\oplus 1][7] \leftarrow K[i][7]$, $RK[i+8\oplus 1][12] \leftarrow K[i][12]$
19: CNOT($K[i][6], K[i][1]$), CNOT($K[i][0], K[i][1]$
20: CNOT($K[i][11], K[i][6]$), CNOT($K[i][5], K[i][6]$
21: $RK[i+8\oplus 1][1] \leftarrow K[i][1]$, $RK[i+8\oplus 1][6] \leftarrow K[i][6]$
22: CNOT($K[i][10], K[i][11]$), CNOT($K[i][0], K[i][11]$
23: $RK[i+8\oplus 1][11] \leftarrow K[i][11]$
24: CNOT(temp$[0], K[i][10]$), CNOT(temp$[1], K[i][0]$)
25: $RK[i+8\oplus 1][10] \leftarrow K[i][10]$, $RK[i][0] \leftarrow K[i][0]$
26: CNOT(temp$[2], K[i][5]$)
27: $RK[i+8\oplus 1][5] \leftarrow K[i][5]$
28: **return** $RK[i+8\oplus 1] = \{RK[i+8\oplus 1][0], RK[i+8\oplus 1][1], \ldots, RK[i+8\oplus 1][15]\}$

3.2.2. Round Function

In the second line of the round function from Equation (10), among the input X_i values, the values of $X_i[1]$, $X_i[2]$, and $X_i[3]$ are used as new values of X_{i+1}, but $X_i[0]$ is not used after calculation. Therefore, the value of $X_i[0]$ in the second line can be used by computing $X_{i+1}[3]$ to save the qubits for a new value.

In the second line of the round function, the constant i is XOR-ed to $X_i[0]$. This is done by performing the X gate on the qubit according to the position where the bit of i is 1 in $X_i[0]$. For example, if $i = 3 = (1, 1, 0, 0, \ldots, 0)$, X gate is performed on the qubits $X_3[0][0]$ and $X_3[0][1]$.

The round function of CHAM is expressed in Equations (10) and (11). The detailed circuit design process is shown in Algorithm 9, and the overall quantum circuit for CHAM-64/128 is shown in Figure 12.

Algorithm 9 Round function of CHAM

Input: RK, X_i, Round i
Output: X_{i+1}

if i is odd **then**
  $X_i[1] \leftarrow X_i[1] \lll 1$
  $X_i[1] \leftarrow$CNOT$(RK[i \bmod 2m], X_i[1])$
  $X_i[0] \leftarrow$X gate$(i, X_i[0])$
  $X_i[0] \leftarrow$ADD$(X_i[1], X_i[0])$
  $X_i[1] \leftarrow$CNOT$(RK[i \bmod 2m], X_i[1])$ (reverse)
  $X_i[1] \leftarrow X_i[1] \ggg 1$ (reverse)
  $X_{i+1}[3] \leftarrow X_i[0] \lll 8$
  $X_{i+1}[j] \leftarrow X_i[j+1], \quad j = 0, 1, 2$
else
  $X_i[1] \leftarrow X_i[1] \lll 8$
  $X_i[1] \leftarrow$CNOT$(RK[i \bmod 2m], X_i[1])$
  $X_i[0] \leftarrow$X gate$(i, X_i[0])$
  $X_i[0] \leftarrow$ADD$(X_i[1], X_i[0])$
  $X_i[1] \leftarrow$CNOT$(RK[i \bmod 2m], X_i[1])$ (reverse)
  $X_i[1] \leftarrow X_i[1] \ggg 8$ (reverse)
  $X_{i+1}[3] \leftarrow X_i[0] \lll 1$
  $X_{i+1}[j] \leftarrow X_i[j+1], \quad j = 0, 1, 2$
end if
return $X_{i+1} = \{X_{i+1}[0], X_{i+1}[1], X_{i+1}[2], X_{i+1}[3]\}$

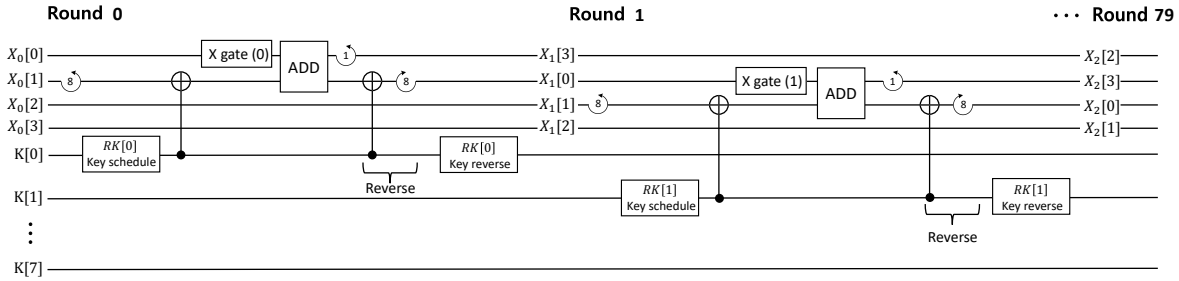

Figure 12. Quantum circuit for CHAM-64/128.

3.3. LEA

In this section, we describe the proposed LEA block cipher in the quantum circuit to apply the Grover search algorithm.

3.3.1. Key Schedule

In the LEA-128 key schedule from Equation (13), the first round key RK_0 is generated using the initial keywords K and a constant value δ. In addition, the next round key RK_1 generation requires RK_0 and δ. Therefore, in order not to use additional qubits in the key schedule of the LEA, the key schedule and the round function are performed simultaneously by using the round key in the corresponding round function and generating the next round key. The detailed process for this is in Algorithm 10.

Algorithm 10 i-th key schedule of LEA-128

Input: δ, $K(i=1)$ or $RK_{i-2}(i>1)$
Output: RK_{i-1}
1: $K[0] \leftarrow \text{ADD}(\delta[(i-1) \bmod 4] \lll (i-1), K[0])$
2: $K[0] \leftarrow K[0] \lll 1$
3: $K[1] \leftarrow \text{ADD}(\delta[(i-1) \bmod 4] \lll (i), K[1])$
4: $K[1] \leftarrow K[1] \lll 3$
5: $K[2] \leftarrow \text{ADD}(\delta[(i-1) \bmod 4] \lll (i+1), K[2])$
6: $K[2] \leftarrow K[2] \lll 6$
7: $K[3] \leftarrow \text{ADD}(\delta[(i-1) \bmod 4] \lll (i+2), K[3])$
8: $K[3] \leftarrow K[3] \lll 11$
9: **return** $RK_{i-1} = \{K[0], K[1], K[2], K[1], K[3], K[1]\}$

3.3.2. Round Function

In the round function, additional qubits were not used by appropriately specifying the operation order and target. New qubits for X_{i+1} from Equation (14) are not allocated, and it is calculated in the X_i and treated as a new value X_{i+1}. The operation starts from the third line of Equation (14). $X_i[3]$ is used as the value of $X_{i+1}[2]$ because $X_i[3]$ is no longer used. After the third line operation is finished, the value of $X_i[2]$ in the second line is no longer used, so it is used as the new value $X_{i+1}[1]$. The detailed process for this is in Algorithm 11. The overall quantum circuit for LEA-128 is shown in Figure 13.

Algorithm 11 Round function of LEA-128

Input: X_i, RK, Round i
Output: X_{i+1}
1: $X_i[3] \leftarrow \text{CNOT}(RK_i[5], X_i[3])$
2: $X_i[2] \leftarrow \text{CNOT}(RK_i[4], X_i[2])$
3: $X_i[3] \leftarrow \text{ADD}(X_i[2], X_i[3])$
4: $X_i[2] \leftarrow \text{CNOT}(RK_i[4], X_i[2])$ (reverse)
5: $X_{i+1}[2] \leftarrow X_i[3] \lll 3$
6: $X_i[2] \leftarrow \text{CNOT}(RK_i[3], X_i[2])$
7: $X_i[1] \leftarrow \text{CNOT}(RK_i[2], X_i[1])$
8: $X_i[2] \leftarrow \text{ADD}(X_i[1], X_i[2])$
9: $X_i[1] \leftarrow \text{CNOT}(RK_i[2], X_i[1])$ (reverse)
10: $X_{i+1}[1] \leftarrow X_i[2] \lll 5$
11: $X_i[1] \leftarrow \text{CNOT}(RK_i[1], X_i[1])$
12: $X_i[0] \leftarrow \text{CNOT}(RK_i[0], X_i[0])$
13: $X_i[1] \leftarrow \text{ADD}(X_i[0], X_i[1])$
14: $X_i[0] \leftarrow \text{CNOT}(RK_i[0], X_i[0])$ (reverse)
15: $X_{i+1}[0] \leftarrow X_i[1] \lll 9$
16: $X_{i+1}[3] \leftarrow X_i[0]$
17: **return** $X_{i+1} = \{X_{i+1}[0], X_{i+1}[1], X_{i+1}[2], X_{i+1}[3]\}$

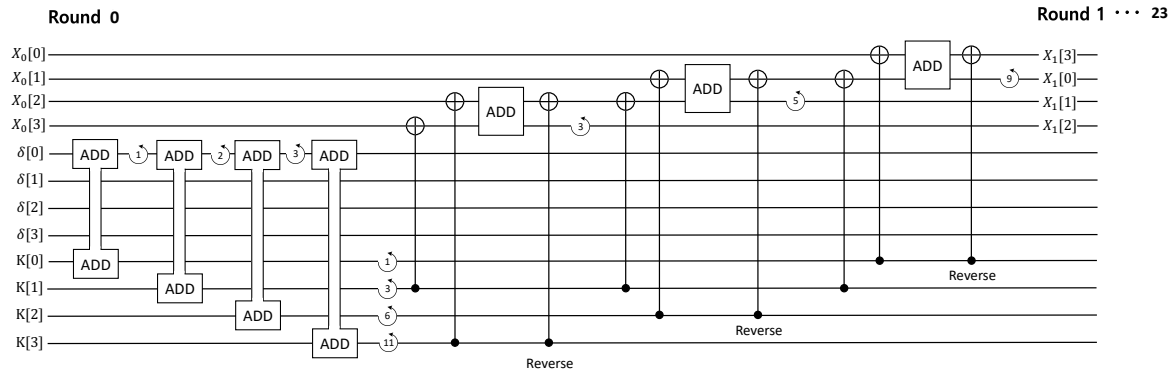

Figure 13. Quantum circuit for LEA-128.

3.4. SPECK

In this section, we describe the proposed SPECK block cipher in the quantum circuit to apply the Grover search algorithm.

3.4.1. Key Schedule

Looking at Equation (15), key words $K = \{K[0], l[0], \ldots, l[m-2]\}$ are extended to key $K = \{K[0], K[1], \ldots, K[r-1]\}$ through key schedule. To save the use of qubits, we update the qubits of $K[0]$ used in the first round and use it as the next round key instead of generating all round key RK. As a result, $K[0]$ is used as a round key from start to finish. Qubits assigned to the initial keywords K, an element of the key schedule for updating the round key $K[0]$, are reused to the end. By recycling the qubits in this way, all round functions can be completed with only the initial keyword qubits. If a round key is generated in $K[0]$, the next round key must be generated after executing the round function. For $1 \leq i < r$, the detailed process for the key schedule is in Algorithm 12.

Algorithm 12 RK[i] key schedule of SPECK.

Input: $K = \{K[0], l[0], \ldots, l[m-2]\}$
Output: $RK[i]$
1: $l[(i-1)\%(m-1)] \leftarrow l[(i-1)\%(m-1)] \ggg \alpha$
2: $l[(i-1)\%(m-1)] \leftarrow \mathrm{ADD}(K[0], l[(i-1)\%(m-1)])$
3: $l[(i-1)\%(m-1)] \leftarrow \mathrm{X\ gate}((i-1), l[(i-1)\%(m-1)])$
4: $K[0] \leftarrow K[0] \lll \beta$
5: $K[0] \leftarrow \mathrm{CNOT}(l[(i-1)\%(m-1)], K[0])$
6: $RK[i] \leftarrow K[0])$
7: **return** $RK[i]$

3.4.2. Round Function

Round function of SPECK from Equation (16) uses only input plaintext X and round key RK. It proceeds by updating the input X, and encryption is completed when all round functions are performed. The detailed process for this is described in Algorithm 13. The overall quantum circuit for SPECK-32/64 is shown in Figure 14.

Algorithm 13 Round function of SPECK.

Input: X, RK, Round i
Output: X_{i+1}
1: $X_i[1] \leftarrow X_i[1] \ggg \alpha$
2: $X_i[1] \leftarrow \text{ADD}(X_i[0], X_i[1])$
3: $X_{i+1}[1] \leftarrow \text{CNOT}(RK_i[i], X_i[1])$
4: $X_i[0] \leftarrow X_i[0] \lll \beta$
5: $X_{i+1}[0] \leftarrow \text{CNOT}(X_{i+1}[1], X_i[0])$
6: **return** $X_{i+1} = \{X_{i+1}[0], X_{i+1}[1]\}$

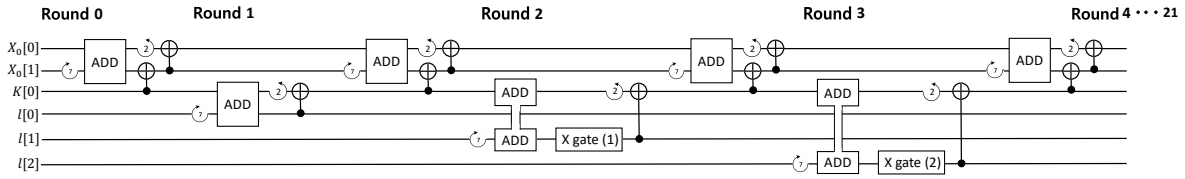

Figure 14. Quantum circuit for SPECK-32/64.

4. Evaluation

In order to evaluate quantum gates, we utilized the quantum computer emulator. We utilized the well-known IBM's ProjectQ framework for evaluation of the proposed method (https://github.com/ProjectQ-Framework/ProjectQ). The framework provides quantum computer compiler and quantum resource estimator. This is useful for accurate evaluation. Proposed quantum gates are written in Python and follow the ProjectQ grammar.

4.1. Cost of Quantum Gates

The most expensive of the quantum gates used in the implementation is the Toffoli gate. Toffoli gate consists of six CNOT gates + seven T gates. In other words, Toffoli gate costs more than six CNOT gates. Therefore, the number of Toffoli gates is the most important factor in evaluating quantum resources. On the other hand, the cost of an X gate is the cheapest and the number used for implementation is overall negligibly low.

4.2. Korean Block Ciphers in Quantum Gates

Korean block ciphers, such as CHAM, LEA, and HIGHT, are based on the ARX architecture. However, the primitive operations for each block cipher are totally different, in terms of word size and the number of operations. In Table 6, the quantum resource estimates for each block cipher, and the higher cost is shown in red and the lower cost is shown in green. The detailed comparison is as follows.

4.2.1. Case 1: 64-Bit Plaintext & 128-Bit Key

First, we compared CHAM-64/128 and HIGHT-64/128 block ciphers in terms of quantum resources. The number of qubit, Toffoli gate, and CNOT gate for CHAM-64/128 is lower than HIGHT-64/128 by 5, 3872, and 8203, respectively. In CHAM-64/128, it is all about performing a 16-bit Addition operation once in the round function. However, in HIGHT, 8-bit addition operations are performed 14 times in the round function, including the reverse operation. As a result, the circuit complexity of HIGHT is higher, even though CHAM performs more round functions. Only the number of X gates for HIGHT-64/128 is lower than CHAM-64/128 by 236. This is because CHAM performs an operation that XOR-ing constant i.

4.2.2. Case 2: 128-Bit Plaintext & 128-Bit Key

Second, we compared CHAM-128/128 and LEA-128/128 block ciphers in terms of quantum resources. The number of qubit, Toffoli gate, and CNOT gate for CHAM-128/128 is lower than LEA-128/128 by 117, 5456, and 1120, respectively. From the qubit point of view, LEA adds the constant δ to the key value in the key schedule. If it was an XOR operation, it can be solved only by an X gate without an additional qubit like CHAM. However, because it is an addition operation, setting the value of δ requires 128 qubits, so LEA uses more qubits. As you can see intuitively, the key schedule and round function of the LEA consist mostly of addition. Therefore, LEA has the high complexity among the block ciphers in this paper. Only the number of X gates for LEA-128/128 is lower than CHAM-128/128 by 172. This is because, unlike CHAM, LEA does not use the X gate during the round function, but uses the X gate only when initially setting the δ value. The number of X gates is equal to the δ's hamming weight.

4.2.3. Case 3: 128-Bit Plaintext & 256-Bit Key

Third, we compared CHAM-128/256 and LEA-128/256 block ciphers. The number of qubit, Toffoli gate, and CNOT gate for CHAM-128/256 is lower than LEA-128/256 by 245, 11,904, and 13,152, respectively. Only the number of X gate for LEA-128/256 is lower than CHAM-128/256 by 174. This result also shows that, even in large-scale parameters, CHAM has lower complexity in quantum gates than LEA.

4.2.4. CASE 4: Addition of LEA and HIGHT

Lastly, we compared addition of LEA and HIGHT. LEA-128 performs addition operations four times in the key schedule, three times in the round function, a total of seven, whereas HIGHT performs addition operations four times in the key schedule, four times in the round function, a total of eight. Unlike a classic computer, on a quantum computer, initializing qubits to its previous value requires a reverse operation that iterates over what was done. Since the result of the addition operation of the LEA gets used continuously, the reverse operation is not required. On the other hand, in HIGHT, in order to minimize the usage of the qubit, it is necessary to perform the reverse operation after using the addition result. Therefore, a total of six reverse operations are added, resulting in increased circuit complexity.

Overall, the CHAM block cipher requires the lowest quantum resources. This result introduces the fact that classical security level is not directly related to quantum attack efforts. The lesson is that modern lightweight block ciphers are not good options against quantum computers.

Table 6. Comparison between Korean block ciphers in terms of quantum resources.

Block Cipher	Plaintext (bit)	Key (bit)	Qubits Used	Toffoli Gates	CNOT Gates	X Gates
CHAM	64	128	196	2400	12,285	240
	128	128	268	4960	26,885	240
	128	256	396	5952	32,277	304
HIGHT	64	128	201	6272	20,523	4
LEA	128	128	385	10,416	28,080	68
	128	192	513	15,624	39,816	100
	128	256	641	17,856	45,504	130

4.3. Software-Oriented and Hardware-Oriented Block Ciphers

We also evaluated the quantum resources for software-oriented block cipher (i.e., SPECK) and hardware-oriented block cipher (i.e., SIMON). SPECK consists of addition, XOR, and rotation, while SIMON consists of logical-and, XOR, and rotation. The main difference is addition (i.e., SPECK) and logical-and operation (i.e., SIMON). In Table 7, the comparison between SPECK and SIMON in terms of quantum resources is given. The result shows that the number of qubit, toffoli gate, and CNOT gate

for SIMON is lower than SPECK. Only the number of X gate for SPECK is lower than SIMON. For this reason, we can conclude that hardware oriented block cipher is more vulnerable than software-oriented block cipher in terms of Grover search algorithm.

Table 7. Comparison between SPECK and SIMON in terms of quantum resources.

Plaintext (bit)	Key (bit)	Qubits Used		Toffoli Gates		CNOT Gates		X Gates	
		SPECK	SIMON [9]	SPECK	SIMON [9]	SPECK	SIMON [9]	SPECK	SIMON [9]
32	64	97	96	1,290	512	3,706	2,816	42	448
48	72	121	120	1,982	864	5,606	3,312	42	792
48	96	145	144	2,074	864	5,866	4,800	45	768
64	96	161	160	3,162	1,344	8,890	5,184	54	1,248
64	128	193	192	3,286	1,408	9,238	7,396	57	1,216
96	96	193	192	5,172	2,496	14,436	9,792	60	2,400
96	144	241	240	5,360	2,592	14,960	10,080	64	2,448
128	128	257	256	7,942	4,352	22,086	17,152	75	4,224
128	192	321	320	8,192	4,416	22,784	17,472	80	4,224
128	256	385	384	8,444	4,608	23,484	26,624	81	4,352

4.4. SPN and ARX Based Block Ciphers

AES is based on SPN structure composed of AddRoundkey, ShiftRows, SubBytes, and MixColumn.

AddRoundkey is an operation that XOR-ing a key value to data, and can be implemented relatively simple by using only CNOT gates. ShiftRows is free because it uses only rotation operations.

On the other hand, MixColumn requires a (complicated) multiplication circuit, which uses more Toffoli gate than addition circuit, and generic n-bit multiplication requires at least an additional n qubits. Some multiplication works using the Karatsuba algorithm have been performed, which can reduce the number of Toffoli gates, but this requires extra qubits. SubBytes is an operation for substituting data in 8-bit units, and the actual operation is an operation for affine conversion after obtaining an inverse on $GF(2^8)$ for one byte. For inversion operation, squaring and multiplication are performed together. Squaring can be performed with only 19 CNOT gates on an 8-bit basis without additional qubits, but, as mentioned earlier, inversion is a complex circuit because of multiplication. That is, in the SPN structure, the complexity of the circuit is greatly increased compared to the ARX structure due to SubBytes of substitution and MixCoulmn of permutation.

Several studies with quantum resource estimation required for AES block ciphers are presented in Table 8. The implementation of Langenberg et al. [7] is relatively well optimized because the S-box operation is optimized efficiently.

Table 8. Quantum resources to implement AES-128.

Block Cipher	Qubits Used	Toffoli Gates	CNOT Gates	X Gates
Grassl et al. [6]	984	151,552	166,548	1456
Almazrooie et al. [21]	976	150,528	192,832	1370
Langenberg et al. [20]	864	16,940	107,960	1507

5. Conclusions

In this paper, we presented the optimized implementation of ARX-based Korean block ciphers in quantum gates. The proposed implementation achieved the optimal number of qubit and quantum gates by carefully designing the adder circuit and its reversible gates. This is the first implementation of Korean block ciphers in quantum gates. The result shows that the CHAM block cipher requires the lowest quantum resources, which means the CHAM block cipher is the most vulnerable towards the Grover search algorithm. We also evaluated the SPECK and SIMON on quantum gates. The result shows that hardware-oriented block cipher (i.e., SIMON) shows the lower quantum resources than SPECK.

Author Contributions: Investigation, H.K. (Hyeokdong Kwon), H.K. (Hyunji Park) and J.P.; Software, K.J. and H.S.; Supervision, H.S.; Writing—original draft, K.J.; Writing—review and editing, S.C. All authors have read and agreed to the published version of the manuscript.

Funding: This work was 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)

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*; Technical Report 2017/511, Cryptology ePrint Archive; University of Luxembourg: Luxembourg, 2017.
3. Hankerson, D.; Menezes, A.J.; Vanstone, S. *Guide to Elliptic Curve Cryptography*; Springer Science & Business Media: Berlin/Heidelberg, Germany, 2006.
4. Shor, P.W. Algorithms for quantum computation: Discrete logarithms and factoring. In Proceedings of the 35th IEEE Annual Symposium on Foundations of Computer Science, Santa Fe, NM, USA, 20–22 November 1994; pp. 124–134.
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 2019/854, Cryptology ePrint Archive; University of Luxembourg: Luxembourg, 2019.
8. Jaques, S.; Naehrig, M.; Roetteler, M.; Virdia, F. Implementing Grover oracles for quantum key search on AES and LowMC. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*; Springer: Berlin/Heidelberg, Germany, 2020; pp. 280–310.
9. Anand, R.; Maitra, A.; Mukhopadhyay, S. Grover on SIMON. *arXiv* **2020**, arXiv:2004.10686.
10. Beaulieu, R.; Shors, D.; Smith, J.; Treatman-Clark, S.; Weeks, B.; Wingers, L. The SIMON and SPECK Families of Lightweight Block Ciphers. *IACR Cryptol. ePrint Arch.* **2013**, *2013*, 404–449.
11. Hong, D.; Sung, J.; Hong, S.; Lim, J.; Lee, S.; Koo, B.S.; Lee, C.; Chang, D.; Lee, J.; Jeong, K.; et al. HIGHT: A new block cipher suitable for low-resource device. In *International Workshop on Cryptographic Hardware and Embedded Systems*; Springer: Berlin/Heidelberg, Germany, 2006; pp. 46–59.
12. Koo, B.; Roh, D.; Kim, H.; Jung, Y.; Lee, D.G.; Kwon, D. CHAM: A Family of Lightweight Block Ciphers for Resource-Constrained Devices. In *International Conference on Information Security and Cryptology (ICISC'17)*; Springer: Cham, Switzerland, 2017.
13. Roh, D.; Koo, B.; Jung, Y.; Jeong, I.W.; Lee, D.G.; Kwon, D.; Kim, W.H. Revised Version of Block Cipher CHAM. In *International Conference on Information Security and Cryptology*; Springer: Cham, Switzerland, 2019; pp. 1–19.
14. Hong, D.; Lee, J.K.; Kim, D.C.; Kwon, D.; Ryu, K.H.; Lee, D.G. LEA: A 128-bit block cipher for fast encryption on common processors. In *International Workshop on Information Security Applications*; Springer: Cham, Switzerland, 2013, pp. 3–27.
15. Cuccaro, S.A.; Draper, T.G.; Kutin, S.A.; Moulton, D.P. A new quantum ripple-carry addition circuit. *arXiv* **2004**, arXiv:quant-ph/0410184.
16. Draper, T.G. Addition on a quantum computer. *arXiv* **2000**, arXiv:quant-ph/0008033.
17. Draper, T.G.; Kutin, S.A.; Rains, E.M.; Svore, K.M. A logarithmic-depth quantum carry-lookahead adder. *arXiv* **2004**, arXiv:quant-ph/0406142.
18. Vedral, V.; Barenco, A.; Ekert, A. Quantum networks for elementary arithmetic operations. *Phys. Rev. A* **1996**, *54*, 147.
19. Daemen, J.; Rijmen, V. *AES Proposal: Rijndael*; NIST AES Competition; Springer: Berlin/Heidelberg, Germany, 1999; pp. 1–45.

20. 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.
21. Mishal Almazrooie, Azman Samsudin, R.A.; Mutter, K.N. Quantum reversible circuit of AES-128. *Quantum Inf. Process.* **2018**, *17*, 112–177.