Quantum Information Processing (2022) 21:373
https://doi.org/10.1007/s11128-022-03714-3

Parallel quantum addition for Korean block ciphers

Kyungbae Jang[1] · Gyeongju Song[1] · Hyunjun Kim[1] · Hyeokdong Kwon[1] · Hyunji Kim[1] · Hwajeong Seo[1]

Received: 15 January 2022 / Accepted: 18 October 2022

Abstract
Malicious users using quantum computers can employ quantum attacks on modern cryptography algorithms. Grover's search algorithm, a well-known quantum algorithm, can reduce the search complexity of $O(2^n)$ to $\sqrt{2^n}$ for symmetric key cryptography with an n-bit key. To apply the Grover search algorithm, the target encryption process must be implemented in a quantum circuit. In this paper, we present optimized quantum circuits for Korean block ciphers based on ARX architectures. We adopt the optimal quantum adder and design it in parallel way. Compared to previous implementations, we provide performance improvements of 78%, 85%, and 70% in terms of circuit depth for LEA, HIGHT, and CHAM, respectively, while keeping the number of qubits and quantum gates minimum. The depth of a circuit is an important factor

This work was partly supported by Institute for Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (<Q|Crypton>, No.2019-0-00033, Study on Quantum Security Evaluation of Cryptography based on Computational Quantum Complexity, 50%), and this work was 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, 25%), and this work of Kyungbae Jang was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (2022R1A6A3A13062701, 25%).

✉ Hwajeong Seo
hwajeong84@gmail.com

Kyungbae Jang
starj1023@gmail.com

Gyeongju Song
thdrudwn98@gmail.com

Hyunjun Kim
khj930704@gmail.com

Hyeokdong Kwon
korlethean@gmail.com

Hyunji Kim
khj1594012@gmail.com

[1] Division of IT Convergence Engineering, Hansung University, Seoul, South Korea

Published online: 09 November 2022

related to its execution time. Finally, we estimate the cost of the Grover key search for Korean block ciphers and evaluate the post-quantum security based on the criteria presented by NIST.

Keywords Parallel quantum addition · Korean block ciphers · ARX architecture · Grover algorithm

1 Introduction

International companies, such as IBM, Google, Microsoft, and Amazon, are advancing the development of large-scale quantum computers. If a large-scale quantum computer that can operate quantum algorithms is developed, the safety of cryptography, which is widely used today, will be vulnerable by these quantum attacks. Shor's algorithm has been proven to solve the mathematical hardness of RSA and elliptic curve cryptography (ECC) [33] on quantum computers [8, 14, 17, 18, 32]. National Institute of Standards and Technology (NIST) is currently conducting a post-quantum cryptography (PQC) competition in preparation for the security collapse of modern public key cryptography.

Another quantum algorithm, Grover's search algorithm, is well known for reducing the security level of symmetric key cryptography [16]. Grover search algorithm reduces n-bit block ciphers with n-bit security to $\frac{n}{2}$-bit security. Since the security of symmetric key ciphers from a quantum attack is reduced, it is important to estimate quantum resources needed for the quantum attack. NIST presents the cost of Grover key search, which is the most effective quantum attack against symmetric key cryptography, as a post-quantum security strength. In other words, as the cost of Grover key search increases, the strength of post-quantum security increases. With this motivation, estimating the cost of Grover key search for symmetric key cryptography is an active research field in recent years [1, 3, 5, 6, 11, 15, 21, 23–26, 28, 30].

In this paper, we focus on quantum cryptanalysis of Korean block ciphers. For this, we firstly optimize quantum circuits for Korean block ciphers LEA [19], HIGHT [29], and CHAM [20]. In the previous work [21], Jang et al. estimated the cost required to implement Korean block ciphers (LEA, HIGHT, and CHAM) as quantum circuits. We present improved performance with proposed implementation techniques and perform tight quantum cryptanalysis on Korean block ciphers.

These three block ciphers are designed with ARX architecture. ARX architecture means that it consists of *Addition*, *Rotation*, and *XOR* operations. Actually, round functions and key schedules of LEA, HIGHT, and CHAM are designed only with addition, rotation, and XOR operations. Addition is not a high-cost operation on classical computers. In contrast, addition on a quantum computer corresponds to a relatively high-cost operation. Thus, various implementation techniques have been proposed to efficiently implement a quantum adder. We adopt an optimal quantum adder [12] to optimize the ARX architecture. This quantum adder uses one ancilla qubit in the proposed implementation, but this can reduce Toffoli gates and circuit depth. In addition, we present a technique to implement the parallel addition by allocating more qubits. As a result, we reduce the number of Toffoli gates and provide high-performance improvements in terms of circuit depth.

In [21], only costs of quantum circuits for Korean block ciphers are estimated. On the other hand, we tightly estimate costs of Grover key search based on proposed quantum circuits. Finally, we evaluate the post-quantum security level of Korean block ciphers based on the criteria presented by NIST [31]. Our quantum circuit implementations and reports are simulated and estimated on a classical computer with a quantum programming tool. We use ProjectQ [34], a quantum programming tool provided by ETH Zurich and IBM for validation of proposed implementation results. The use of simulators is common in many quantum-related studies [4, 24–26, 28] as access to real quantum computers is difficult and large-scale quantum computers have not yet been developed.

In quantum analysis of symmetric key cryptography, it is important to efficiently implement quantum circuits for target ciphers. Source codes of proposed implementation will be available as a public domain for the reproduction of proposed methods.

1.1 Our contribution

The contribution of this paper can be summarized as follows:

1. **Optimization of ARX architecture in quantum circuits with parallel additions** `LEA`, `HIGHT`, and `CHAM` block ciphers have ranges that allow the parallel addition. We optimize the round function and key schedule structure by using the parallel addition.
2. **Optimized quantum circuits for Korean block ciphers of ARX architecture using an optimal quantum adder** We adopt an optimal quantum adder that uses few qubits and design it as a parallel addition architecture. We provide performance improvements in terms of circuit depth, while keeping the number of qubits and quantum gates minimum.
3. **Tight quantum cryptanalysis of Korean block ciphers** Quantum costs of Grover key search are estimated based on optimized quantum circuits. Then, we evaluate the post-quantum security level for Korean block ciphers compared to costs presented by NIST as security strength.

Source code of proposed method is available as open-source[1]. We validate proposed implementations by simulating it on classical computer, as long as it is practicable.

1.2 Previous works

For the analysis of symmetric key cryptography, researches implemented the block cipher in quantum circuits and estimated quantum resources required for Grover's search. Starting with `AES` [1, 15, 24, 30], the research field is expanding to `SIMON` [5], `SPECK` [25], `GIFT` [22], `PRESENT` [26], and `PIPO` [27]. In quantum circuit implementations, `SIMON` [5] uses fewer quantum resources than `SPECK` [25] because `SIMON` uses AND operation rather than addition, which requires a lot of quantum

[1] https://github.com/starj1023/Korea-ARX-QC/.

resources. In the SPN structure, when the SBox is implemented efficiently, the overall resource used is reduced [22, 26, 27]. For the efficient implementation of quantum circuits for resource reduction, in [13], authors presented LIGHTER-R, a tool to optimize a 4-bit Sbox on a quantum circuit.

For AEADs, authors of [6] presented quantum implementation GRAIN-128-AEAD and TINYJAMBU. In [7], authors presented quantum implementations and analysis of KNOT-AEAD.

2 Background

2.1 LEA

LEA is working on the 128-bit plaintext (X) organized in a 32×4 array and uses 128, 192, or 256-bit key (K). LEA requires constants (δ) for generating round keys as follows. LEA-128, 192, and 256 use $\delta_{0\sim3}$, $\delta_{0\sim5}$, and $\delta_{0\sim7}$, respectively.

$$\begin{aligned}
\delta_0 &= \texttt{0xc3efe9db}, \quad \delta_1 = \texttt{0x44626b02},\\
\delta_2 &= \texttt{0x79e27c8a}, \quad \delta_3 = \texttt{0x78df30ec},\\
\delta_4 &= \texttt{0x715ea49e}, \quad \delta_5 = \texttt{0xc785da0a},\\
\delta_6 &= \texttt{0xe04ef22a}, \quad \delta_7 = \texttt{0xe5c40957}
\end{aligned}$$

The key schedule of LEA-128, LEA-192, and LEA-256 is similar to each other and is as follows (LEA-128, LEA-192, and LEA-256 in that order). The ranges of i (round number) for LEA-128, LEA-192, and LEA-256 are $(0 \le i \le 23)$, $(0 \le i \le 27)$, and $(0 \le i \le 31)$, respectively. Notation ($\boxplus$) indicates modular addition, and notation ($\lll$) represents left rotation.

$$\begin{aligned}
K[0] &= (K[0] \boxplus (\delta_{i \bmod 4} \lll i)) \lll 1\\
K[1] &= (K[1] \boxplus (\delta_{i \bmod 4} \lll (i+1))) \lll 3\\
K[2] &= (K[2] \boxplus (\delta_{i \bmod 4} \lll (i+2))) \lll 6\\
K[3] &= (K[3] \boxplus (\delta_{i \bmod 4} \lll (i+3))) \lll 11\\
RK_i &= (K[0], K[1], K[2], K[1], K[3], K[1])\\
K[0] &= (K[0] \boxplus (\delta_{i \bmod 6} \lll i)) \lll 1\\
K[1] &= (K[1] \boxplus (\delta_{i \bmod 6} \lll (i+1))) \lll 3\\
K[2] &= (K[2] \boxplus (\delta_{i \bmod 6} \lll (i+2))) \lll 6\\
K[3] &= (K[3] \boxplus (\delta_{i \bmod 6} \lll (i+3))) \lll 11\\
K[4] &= (K[4] \boxplus (\delta_{i \bmod 6} \lll (i+4))) \lll 13\\
K[5] &= (K[5] \boxplus (\delta_{i \bmod 6} \lll (i+5))) \lll 17\\
RK_i &= (K[0], K[1], K[2], K[3], K[4], K[5])
\end{aligned}$$

$$
\begin{aligned}
&K[6i \bmod 8] = (K[6i \bmod 8] \boxplus (\delta_{i \bmod 8} \lll i)) \lll 1\\
&K[(6i+1) \bmod 8] = (K[(6i+1) \bmod 8] \boxplus\\
&(\delta_{i \bmod 8} \lll (i+1))) \lll 3\\
&K[(6i+2) \bmod 8] = (K[(6i+2) \bmod 8] \boxplus\\
&(\delta_{i \bmod 8} \lll (i+2))) \lll 6\\
&K[(6i+3) \bmod 8] = (K[(6i+3) \bmod 8] \boxplus\\
&(\delta_{i \bmod 8} \lll (i+3))) \lll 11\\
&K[(6i+4) \bmod 8] = (K[(6i+4) \bmod 8] \boxplus\\
&(\delta_{i \bmod 8} \lll (i+4))) \lll 13\\
&K[(6i+5) \bmod 8] = (K[(6i+5) \bmod 8] \boxplus\\
&(\delta_{i \bmod 8} \lll (i+5))) \lll 17\\
&RK_i = (K[6i \bmod 8], K[(6i+1) \bmod 8],\\
&K[(6i+2) \bmod 8], K[(6i+3) \bmod 8],\\
&K[(6i+4) \bmod 8], K[(6i+5) \bmod 8])
\end{aligned}
$$

`LEA-128`, `192`, and `256` use the same round function except for the number of rounds i. The round function, regardless of its parameters, is given by (notation $(\oplus)$ indicates XOR operation):

$$
\begin{aligned}
X_{i+1}[0] &= ((X_i[0] \oplus RK_i[0]) \boxplus (X_i[1] \oplus RK_i[1])) \lll 9\\
X_{i+1}[1] &= ((X_i[1] \oplus RK_i[2]) \boxplus (X_i[2] \oplus RK_i[3])) \lll 5\\
X_{i+1}[2] &= ((X_i[2] \oplus RK_i[4]) \boxplus (X_i[3] \oplus RK_i[5])) \lll 3\\
X_{i+1}[3] &= X_i[0]
\end{aligned}
$$

2.2 HIGHT

`HIGHT` is working with 64-bit plaintext(X) organized in a 8×8 array and uses 128-bit key(K) in 8×16 array. In `HIGHT` block cipher, the key schedule to generate round keys(RK) using δ is as follows:

$$
\begin{aligned}
s_{i+6} &= s_{i+2} \oplus s_{i-1}\\
\delta_i &= (s_{i+6}, s_{i+5}, s_{i+4}, s_{i+3}, s_{i+2}, s_{i+1}, s_i)\\
&\text{for } i = 0 \text{ to } 7:\\
&\quad \text{for } j = 0 \text{ to } 7:\\
&\qquad RK[16 \cdot i + j] = K[j - i \bmod 8] \boxplus \delta_{16 \cdot i + j}\\
&\quad \text{for } j = 0 \text{ to } 7:\\
&\qquad RK[16 \cdot i + j + 8] = K[(j - i \bmod 8) + 8] \boxplus \delta_{16 \cdot i + j + 8}
\end{aligned}
$$

The round function is given by ($1 \le i \le 31$):

$$
\begin{aligned}
X_i[j] &= X_{i-1}[j-1], \quad j = 1, 3, 5, 7\\
X_i[0] &= X_{i-1}[7] \oplus (F_0(X_{i-1}[6]) \boxplus RK[4i-1])\\
X_i[2] &= X_{i-1}[1] \boxplus (F_1(X_{i-1}[0]) \oplus RK[4i-4])\\
X_i[4] &= X_{i-1}[3] \oplus (F_0(X_{i-1}[2]) \boxplus RK[4i-3])\\
X_i[6] &= X_{i-1}[5] \boxplus (F_1(X_{i-1}[4]) \oplus RK[4i-2])
\end{aligned}
$$

The last round function (i.e., $i = 32$) is slightly different and given by:

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

F_0 and F_1 functions used in round function are given by:

$$\begin{aligned}
F_0(X[k]) &= (X[k] \lll 1) \oplus (X[k] \lll 2) \oplus (X[k] \lll 7)\\
F_1(X[k]) &= (X[k] \lll 3) \oplus (X[k] \lll 4) \oplus (X[k] \lll 6)
\end{aligned}$$

2.3 CHAM

CHAM is working with 64(16×4) or 128(32×4)-bit plaintext(X) and uses 128- or 256-bit key(K). In the key schedule of CHAM, input key K is divided into $K = K[0]||K[1]||\ldots||K[w]$ ($w = 8$ for CHAM-64/128 and 12-8/256, $w = 4$ for CHAM-128/128). The key schedule is given by:

$$\begin{aligned}
RK[i] &= K[i] \oplus (K[i] \lll 1) \oplus (K[i] \lll 8)\\
RK[i + w] &= K[i] \oplus (K[i] \lll 1) \oplus (K[i] \lll 8)
\end{aligned}$$

The round function of CHAM depends on the number of rounds i (the diagram can be seen in Fig. 4). If the number of rounds i is odd, the round function is given by:

$$\begin{aligned}
&X_{i+1}[3] = ((X_i[0] \oplus i) \boxplus (X_i[1] \lll 1) \oplus\\
&(RK[i \bmod w)) \lll 8\\
&X_{i+1}[j] = X_i[j + 1], (0 \le j \le 2)
\end{aligned}$$

Otherwise (i.e., $i = 0$ or even), given by:

$$\begin{aligned}
&X_{i+1}[3] = ((X_i[0] \oplus i) \boxplus (X_i[1] \lll 8) \oplus\\
&(RK[i \bmod w)) \lll 1\\
&X_{i+1}[j] = X_i[j + 1], (0 \le j \le 2)
\end{aligned}$$

2.4 Quantum gates

Quantum gates used in quantum computers are reversible for all changes during computations. In other words, in quantum computing, it is possible to return to the initial state using only the output state. There are quantum gates with reversible properties that can replace classical gates used in cryptography algorithms. Figure 1 shows representative quantum gates frequently used in cryptography implementations.

The X (NOT) gate (Fig. 1a) inverts the state of the input qubit. The CNOT gate (Fig. 1b) inverts the state of y only if x is 1. The Toffoli (CCNOT) gate (Fig. 1c)

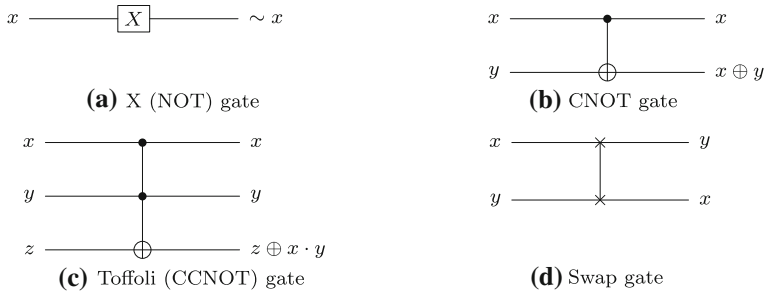

Fig. 1 Detailed descriptions of quantum gates

replaces the AND operation, inverting the state of z only if x and y are 1. The Swap gate (Fig. 1d) changes the state of two input qubits to each other.

2.5 Quantum brute force attack using Grover's search algorithm

A brute force attack on symmetric key cryptography involves finding a key that satisfies a specific plaintext–ciphertext pair. For symmetric key cryptography using an n-bit key, $O(2^n)$ searches are required for a brute force attack. Grover search algorithm is a quantum algorithm that is optimal for brute force attacks on symmetric key cryptography [16]. Compared to a classic computer that requires $O(2^n)$ searches, Grover search algorithm recovers the key with a high probability in $\sqrt{2^n}$ searches. The procedure for Grover key search is as follows.

1. n-qubit key is prepared in superposition $|\psi\rangle$ by applying Hadamard gates. All states of qubits have the same amplitude.

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

2. The symmetric key cryptography is implemented as a quantum circuit and placed in oracle. In oracle $f(x)$, the plaintext is encrypted with the key k in the superposition state. As a result, ciphertexts for all key values are generated. The sign of the solution key is changed to a negative by comparing it with the known ciphertext c. When $f(x) = 1$ changes the sign to negative and applies to all states.

$$f(x) = \begin{cases} 1 \text{ if } \mathrm{Enc}(k) = c \\ 0 \text{ if } \mathrm{Enc}(k) \neq c \end{cases}$$

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

3. Lastly, the diffusion operator amplifies the amplitude of the negative sign state.

Grover algorithm iterates phases 2 and 3 to sufficiently increase the amplitude of the solution and observes it at the end of stage. For an n-bit key, the optimal number of iterations of Grover search algorithm is $\lfloor \frac{\pi}{4}\sqrt{2^n} \rfloor$, which is about $\sqrt{2^n}$. A brute force attack that requires 2^n searches in a classic computer is reduced to $\sqrt{2^n}$ searches in a quantum computer. The efficient implementation of symmetric key cryptography as a quantum circuit is the most important.

The diffusion operator in Grover's search does not require any special technique to be implemented and has negligible overhead. The diffusion operator is included in the Grover iteration, but it is mostly excluded from the cost estimation [3, 6, 7, 15, 30]. The cost of Grover's key search depends on the complexity of oracle.

2.6 Quantum adder

A quantum adder is implemented as a configuration of X, CNOT, and Toffoli gates. In previous implementations of Korean block ciphers [21], a quantum adder based on the ripple-carry approach is used [12]. The quantum adder in previous works utilizes one ancilla qubit, $(2n-2)$ Toffoli gates, $(4n-2)$ CNOT gates, and the circuit depth is $(5n-3)$. In another implementation of the ARX architecture, SPECK [3], Anand. R et al. adopted a different quantum adder [35]. The quantum adder in their works utilizes $(2n-2)$ Toffoli gates, $(5n-6)$ CNOT gates, and the circuit depth is $(5n-5)$ without ancilla qubit. Compared to the quantum adder used in [21], it saves one qubit, but it does not improve the performance in terms of quantum gates or circuit depth.

We use an improved quantum adder based on the ripple-carry approach [12], which is called the CDKM adder. This quantum adder uses one ancilla, but this can reduce Toffoli gates and circuit depth. When $n \geq 4$ in n-qubit addition, an improved quantum adder can be implemented. Since 8-bit addition of HIGHT block cipher is the smallest unit in Korean block ciphers, it can be applied to all. In the case of modular addition, one ancilla qubit can be saved (generic addition uses two ancilla), and the quantum gates and circuit depth can be reduced. Finally, the quantum adder we adopted uses one ancilla qubit, $(2n-3)$ Toffoli gates, $(5n-7)$ CNOT gates, $(2n-6)$ X gates, and the circuit depth is $(2n+3)$. Details of the implementation are found in Algorithm 1 of Sect. 3.

3 Quantum circuit design methodology

In this section, we present optimized quantum circuit techniques for Korean block ciphers. In our understanding, the most important factor to efficiently implement the cipher of the ARX architecture as a quantum circuit is how to design quantum addition. We use an optimal quantum adder and design a parallel addition structure with only small trade-offs between quantum resources. For a clear understanding of the proposed method, it is recommended to cross-check Figures 13 (LEA), 11 (HIGHT), and 12 (CHAM) in the previous work [21] with Figs. 2 (LEA), 3 (HIGHT), and 4 (CHAM) in this paper.

Proposed implementations provide a 78% improvement in LEA, 85% in HIGHT and 70% in CHAM, compared to previous results in terms of depth, respectively. Furthermore, proposed implementations reduce the use of the Toffoli gate, which is the highest cost in NCT (NOT, CNOT, and Toffoli) gates. The quantum circuit implementation for the optimal CDKM adder used in the proposed method is described in Algorithm 1. In CNOT (x, y), x and y are inputs of the CNOT gate and $x \oplus y$ is stored in y. In Toffoli (x, y, z), $x \cdot y$ is stored in z (i.e., $z = z \oplus (x \cdot y)$).

Algorithm 1 Quantum circuit for optimal CDKM adder (n-bit, $n \geq 6$).

Input: n-qubit operands a, b, Carry qubit $c(0)$
Output: $a = a, b = a + b, c = 0$
for $i = 0$ to $n - 3$ **do**
  $b[i+1] \leftarrow \text{CNOT}(a[i+1], b[i+1])$
end for
$c \leftarrow \text{CNOT}(a[1], c)$
$c \leftarrow \text{Toffoli}(a[0], b[0], c)$
$a[1] \leftarrow \text{CNOT}(a[2], a[1])$
$a[1] \leftarrow \text{Toffoli}(c, b[1], a[1])$
$a[2] \leftarrow \text{CNOT}(a[3], a[2])$
for $i = 0$ to $n - 6$ **do**
  $a[i+2] \leftarrow \text{Toffoli}(a[i+1], b[i+2], a[i+2])$
  $a[i+3] \leftarrow \text{CNOT}(a[i+4], a[i+3])$
end for
$a[n-3] \leftarrow \text{Toffoli}(a[n-4], b[n-3], a[n-3])$
$b[n-1] \leftarrow \text{CNOT}(a[n-2], b[n-1])$
$b[n-1] \leftarrow \text{CNOT}(a[n-1], b[n-1])$
$b[n-1] \leftarrow \text{Toffoli}(a[n-3], b[n-2], b[n-1])$
for $i = 0$ to $n - 4$ **do**
  $b[i+1] \leftarrow \text{X}(b[i+1])$
end for
$b[1] \leftarrow \text{CNOT}(c, b[1])$
for $i = 0$ to $n - 4$ **do**
  $b[i+2] \leftarrow \text{CNOT}(a[i+1], b[i+2])$
end for
$a[n-3] \leftarrow \text{Toffoli}(a[n-4], b[n-3], a[n-3])$
for $i = 0$ to $n - 6$ **do**
  $a[n-4-i] \leftarrow \text{Toffoli}(a[n-5-i], b[n-4-i], a[n-4-i])$
  $a[n-3-i] \leftarrow \text{CNOT}(a[n-2-i], a[n-3-i])$
  $b[n-3-i] \leftarrow \text{X}(b[n-3-i])$
end for
$a[1] \leftarrow \text{Toffoli}(c, b[1], a[1])$
$a[2] \leftarrow \text{CNOT}(a[3], a[2])$
$b[2] \leftarrow \text{X}(b[2])$
$c \leftarrow \text{Toffoli}(a[0], b[0], c)$
$a[1] \leftarrow \text{CNOT}(a[2], a[1])$
$b[1] \leftarrow \text{X}(b[1])$
$c \leftarrow \text{CNOT}(a[1], c)$
for $i = 0$ to $n - 2$ **do**
  $b[i] \leftarrow \text{CNOT}(a[i], b[i])$
end for
return a, b, c

3.1 LEA

This section describes the quantum circuit implementation of the LEA. Compared to the previous work [21], which is a quantum implementation for LEA, we reduce the depth by improving the key schedule. For LEA-256, the number of qubits is also reduced.

3.1.1 Key schedule

In the previous implementation [21], $\delta_{0\sim3}$ are initially allocated for the key schedule (LEA-128), and $\delta_{i \bmod 4}$ is used to generate the i-th round key. δ_0 is used to generate the first round key. δ_0 is added to the initial key $K(K[0] \sim [3])$, and the result is used as a round key. The result of addition (i.e., $\delta_0 + K[0]$) is updated in $K[0]$ qubits, while δ_0 is kept in unchanged states. δ_0 is sequentially added to $K[1]$, $K[2]$, and $K[3]$ [21, Figure 13]. On the other hand, we take a different approach by initially setting four δ_0s.

First, we set four δ_0 (not $\delta_0, \delta_1, \delta_2, \delta_3$). In the key schedule of LEA-128, $K[0]$, $K[1]$, $K[2]$, and $K[3]$ are independent of each other. They can be operated in a parallel. For the parallel addition, four same values of $\delta_0[0] \sim [3]$ are required, and $\delta_0[j]$ is added to $K[j]$ $(j = 0, 1, 2, 3)$. As in the previous implementation, if one δ_0 is used in an addition, it cannot be added in a parallel. δ_0 is returned when the addition is completed. However, in the process of addition with $K[0] \sim [3]$, the value of δ_0 is changed by each $K[0] \sim [3]$. It is a design feature that a ripple-carry quantum adder for the addition $(x + y)$, the result is stored in y, x is changed to x' during the process, and returned to x when the addition is completed [12].

Second, we use four carry qubits (c_0, c_1, c_2, c_3). In the ripple-carry modular adder, one carry qubit c_0 is used, which is initialized to 0 after the addition. Taking the advantage of saving qubits, the previous implementation reuses one carry qubit c_0 for all additions. However, reusing initialized c_0 makes the parallel addition infeasible. For the parallel addition with K_0, K_1, K_2, and K_3, we additionally allocate 3 qubits (c_1, c_2, and c_3). As a result, we provide a high performance improvement in terms of depth.

Lastly, we change four δ_0s to four δ_1s for the next key schedule. Since 0xc3efe9db (δ_0) and 0x44626b02 (δ_1) are known values in advance, only X gates are used to change values of qubits. The first round key generation in the proposed quantum circuit design for the key schedule of LEA-128 is described in Algorithm 2. The detailed process for changing δ is described in Algorithm 3.

The key schedule structures of LEA-128, LEA-192 and LEA-256 are similar (LEA-192 uses $\delta_{0\sim5}$ and LEA-25-6 uses $\delta_{0\sim7}$, respectively). However, for LEA-256, we can save qubits compared to the previous implementation [21]. Remember that the key schedule of LEA-256 uses $\delta_{0\sim7}$. Our implementation (LEA-192 and LEA-256) starts by allocating $\delta_0[0 \sim 5]$ at the beginning. This is to allocate 5 δ_0s with the same value for parallel addition. And in the next key schedule, $\delta_0[0 \sim 5]$ is changed to $\delta_1[0 \sim 5]$ without additional qubits (only X gates are used).

In this way, we use only five δs by changing the values. On the other hand, in [21], $\delta_{0\sim7}$ (i.e., seven δs with different values) are initially allocated and adopted according

Algorithm 2 Quantum circuit for `LEA-128` key schedule (first round key generation).

Input: Initial key $K[0] \sim [3]$, $\delta_0[0] \sim [3]$, $c_{0\sim3}$
Output: Round key RK_0, $\delta_1[0] \sim [3]$
1: $\delta_0[0] \leftarrow \delta_0[0] \lll 0$
2: $K[0] \leftarrow ADD(\delta_0[0], K[0], c_0) \lll 1$
3: $\delta_0[1] \leftarrow \delta_0[1] \lll 1$
4: $K[1] \leftarrow ADD(\delta_0[1], K[1], c_1) \lll 3$
5: $\delta_0[2] \leftarrow \delta_0[2] \lll 2$
6: $K[2] \leftarrow ADD(\delta_0[2], K[2], c_2) \lll 6$
7: $\delta_0[3] \leftarrow \delta_0[3] \lll 3$
8: $K[3] \leftarrow ADD(\delta_0[3], K[3], c_3) \lll 11$

// Reverse
9: $\delta_0[0] \leftarrow \delta_0[0] \ggg 0$
10: $\delta_0[1] \leftarrow \delta_0[1] \ggg 1$
11: $\delta_0[2] \leftarrow \delta_0[2] \ggg 2$
12: $\delta_0[3] \leftarrow \delta_0[3] \ggg 3$
13: $\delta_1[0] \sim [3] \leftarrow$ Change $\delta(\delta_0[0] \sim [3], \delta_1[0] \sim [3])$
14: **return** $RK_0(K_0, K_1, K_2, K_1, K_3, K_1), \delta_1[0] \sim [3]$

Algorithm 3 Change δ.

Input: $\delta[0] \sim [3]$, current δ, next δ
Output: $\delta[0] \sim [3]$
1: current $\delta \leftarrow$ current $\delta \oplus$ next δ
2: **for** $i = 0$ to 31 **do**
3: **if** (current $\delta \gg i$) & 1 **then**
4: $\delta[0][i] \leftarrow$X($\delta[0][i]$)
5: $\delta[1][i] \leftarrow$X($\delta[1][i]$)
6: $\delta[2][i] \leftarrow$X($\delta[2][i]$)
7: $\delta[3][i] \leftarrow$X($\delta[3][i]$)
8: **end if**
9: **end for**
10: **return** $\delta[0]$, $\delta[1]$, $\delta[2]$, $\delta[3]$

to the number of rounds i. Thus, we benefit from a reduction in the number of qubits for two deltas (total 32×2 qubits). This is why our quantum implementation of `LEA-256` uses fewer qubits than the previous work. The first round key generation of the proposed quantum circuit design for `LEA-192` and `LEA-256` is same and is described in Algorithm 4.

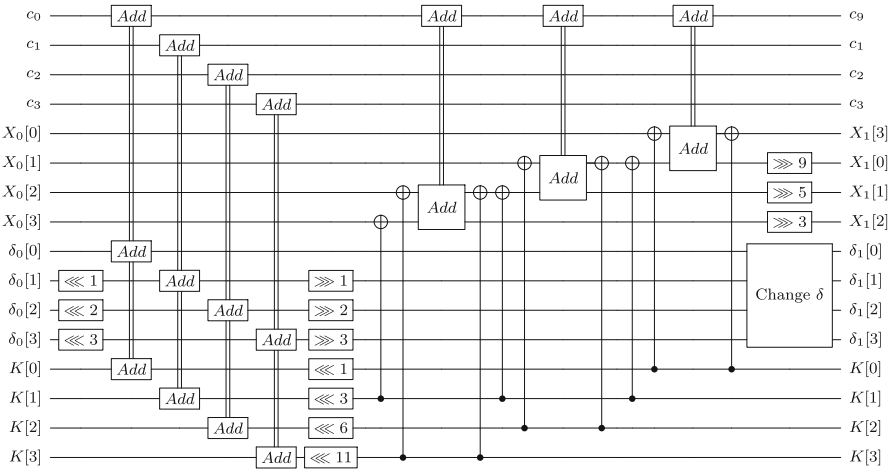

Fig. 2 Quantum circuit for `LEA-128`

Algorithm 4 Quantum circuit for key schedule of `LEA-192` and `LEA-256` (first round key generation).

Input: Initial key $K[0] \sim [5]$, $\delta_0[0] \sim [5]$, $c_{0\sim5}$
Output: Round key RK_0, $\delta_1[0] \sim [5]$
1: $\delta_0[0] \leftarrow \delta_0[0] \lll 0$
2: $K[0] \leftarrow ADD(\delta_0[0], K[0], c_0) \lll 1$
3: $\delta_0[1] \leftarrow \delta_0[1] \lll 1$
4: $K[1] \leftarrow ADD(\delta_0[1], K[1], c_1) \lll 3$
5: $\delta_0[2] \leftarrow \delta_0[2] \lll 2$
6: $K[2] \leftarrow ADD(\delta_0[2], K[2], c_2) \lll 6$
7: $\delta_0[3] \leftarrow \delta_0[3] \lll 3$
8: $K[3] \leftarrow ADD(\delta_0[3], K[3], c_3) \lll 11$
9: $\delta_0[4] \leftarrow \delta_0[4] \lll 4$
10: $K[4] \leftarrow ADD(\delta_0[4], K[4], c_4) \lll 13$
11: $\delta_0[5] \leftarrow \delta_0[5] \lll 5$
12: $K[5] \leftarrow ADD(\delta_0[5], K[5], c_5) \lll 17$

// Reverse
13: $\delta_0[0] \leftarrow \delta_0[0] \ggg 0$
14: $\delta_0[1] \leftarrow \delta_0[1] \ggg 1$
15: $\delta_0[2] \leftarrow \delta_0[2] \ggg 2$
16: $\delta_0[3] \leftarrow \delta_0[3] \ggg 3$
17: $\delta_0[4] \leftarrow \delta_0[4] \ggg 4$
18: $\delta_0[5] \leftarrow \delta_0[5] \ggg 5$
19: $\delta_1[0] \sim [5] \leftarrow Change\ \delta(\delta_0[0] \sim [5], \delta_1[0] \sim [5])$
20: **return** $RK_0(K_0, K_1, K_2, K_3, K_4, K_5), \delta_1[0] \sim [5]$

3.1.2 Round function

The round function of `LEA` is performed on 128-qubit plaintext ($X[0], X[1], X[2], X[3]$). Algorithm 5 describes the quantum circuit for the round function of

LEA block cipher. There is no point of parallelism because the X variables are not independent each other.

Additions that update $X[3]$ and $X[2]$ (lines 3 and 7 of Algorithm 5) cannot be done in a parallel way because we use $X[2]$ to update $X[3]$. The notation CNOT32 means CNOT gate operation in 32-qubit units. Figure 2 shows the overall quantum circuit structure for the first round of LEA-128. The four additions performed in the key schedule are performed in parallel because they do not affect each other. To save qubits, an on-the-fly approach is utilized, which updates the round key one by one and uses it in the round function.

Algorithm 5 Quantum circuit for round function of LEA.

Input: $X[0] \sim [3]$, $RK[0] \sim [5]$, c_0
Output: $X[0] \sim [3]$

//Update $X[3]$
1: $X[3] \leftarrow \text{CNOT32}(RK[5], X[3])$
2: $X[2] \leftarrow \text{CNOT32}(RK[4], X[2])$
3: $X[3] \leftarrow ADD(X[2], X[3], c_0) \ggg 3$
4: $X[2] \leftarrow \text{CNOT32}(RK[4], X[2])$ //Reverse

//Update $X[2]$
5: $X[2] \leftarrow \text{CNOT32}(RK[3], X[2])$
6: $X[1] \leftarrow \text{CNOT32}(RK[2], X[1])$
7: $X[2] \leftarrow ADD(X[1], X[2], c_0) \ggg 5$
8: $X[1] \leftarrow \text{CNOT32}(RK[2], X[1])$ //Reverse

//Update $X[1]$
9: $X[1] \leftarrow \text{CNOT32}(RK[1], X[1])$
10: $X[0] \leftarrow \text{CNOT32}(RK[0], X[0])$
11: $X[1] \leftarrow ADD(X[0], X[1], c_0) \ggg 9$
12: $X[0] \leftarrow \text{CNOT32}(RK[0], X[0])$ //Reverse
13: **return** $X[1], X[2], X[3], X[0]$

3.2 HIGHT

Compared with the previous work, we achieve the highest improvement (85%) for depth in the quantum implementation of HIGHT. This is because improvements are obtained in both round function and key schedule. Details are now explained.

3.2.1 Key schedule and round function

In the previous implementation [21], an on-the-fly approach is used to reduce qubits. Initial key qubits are updated and used as round keys. To reduce the use of qubits, δ_0, which is required for the round key generation, is set and updated to the next $\delta_i (1 \leq i \leq 127)$. Since four additions are independent of each other in the round function, the parallel addition is available. If a round key is generated by updating the initially set δ_0, following additions cannot be performed in a parallel way [21,

Figure 11]. Thus, we initially set $\delta_0, \delta_1, \delta_2, \delta_3$ for the parallel addition. The next round uses $\delta_0, \delta_1, \delta_2, \delta_3 \rightarrow \delta_4, \delta_5, \delta_6, \delta_7$ updated by 4. Updating δ by 1 can be implemented simply with a few CNOT gates and logical swap, which is described in Algorithm 6. Logical swap does not use quantum swap gates by changing the index of qubits. Algorithm 7 describes updating δ by 4 for the parallel addition.

Algorithm 6 Update δ_i.

Input: $\delta_i\,(s_{0\sim7})$
Output: δ_{i+1}
1: $s[0] \leftarrow \text{CNOT}(s[3], s[0])$
//Logical swap
2: **return** $s[1], s[2], s[3], s[4], s[5], s[6], s[0], s[7]$

Algorithm 7 Update $\delta_{i\sim i+3}$ by 4.

Input: $\delta_{i\sim i+3}$
Output: $\delta_{i+4\sim i+7}$
1: **for** $j = 0$ to 3 **do**
2: $\quad\delta_i \leftarrow$ Update δ_i
3: $\quad\delta_{i+1} \leftarrow$ Update δ_{i+1}
4: $\quad\delta_{i+2} \leftarrow$ Update δ_{i+2}
5: $\quad\delta_{i+3} \leftarrow$ Update δ_{i+3}
6: **end for**
7: **return** $\delta_{i\sim i+3}$

The proposed round function quantum circuit of HIGHT is described in Algorithm 10, including key schedule. The last round function of HIGHT is slightly different, but since the structure is very similar, we omit the quantum circuit for it. Quantum circuits for the F_0 and F_1 functions are the same as the previous implementation [21]. These are implemented efficiently without additional qubits through internal mixing operations using only CNOT gates. Algorithms 8 and 9 describe quantum circuits for F_0 and F_1 functions.

Reverse operations performed at the end of Algorithm 10 return values of $X[0], X[2], X[4], X[6]$ and also return values of $K[0] \sim [3]$ for the following key schedule. In HIGHT, the parallel addition can be performed in both round function and key schedule. This is the reason for the highest performance improvement (i.e., 85%) in HIGHT. Figure 3 shows the overall quantum circuit structure for the first round of HIGHT. The notation † indicates the reverse operation, $Add^\dagger$ for K is for the next round key update, and $F^\dagger$ returns values of $X_0[0], X_0[2], X_0[4]$ and $X_0[6]$ (for $X_1[1], X_1[3], X_1[5]$ and $X_1[7]$). Additions that do not share $c(c_0, c_1, c_2, c_3)$ are performed in parallel.

Algorithm 8 Quantum circuit for $F_0(X[k])$

Input: $X[k]$
Output: $X[k]$
1: CNOT($X[k][4]$, $X[k][5]$), CNOT($X[k][3]$, $X[k][4]$)
2: CNOT($X[k][2]$, $X[k][3]$), CNOT($X[k][2]$, $X[k][0]$)
3: CNOT($X[k][4]$, $X[k][2]$), CNOT($X[k][5]$, $X[k][2]$)
4: CNOT($X[k][7]$, $X[k][5]$), CNOT($X[k][6]$, $X[k][4]$)
5: CNOT($X[k][0]$, $X[k][3]$), CNOT($X[k][1]$, $X[k][3]$)
6: CNOT($X[k][6]$, $X[k][1]$), CNOT($X[k][7]$, $X[k][1]$)
7: CNOT($X[k][7]$, $X[k][0]$), CNOT($X[k][5]$, $X[k][6]$)
8: CNOT($X[k][4]$, $X[k][6]$), CNOT($X[k][3]$, $X[k][6]$)
9: CNOT($X[k][1]$, $X[k][6]$), CNOT($X[k][6]$, $X[k][7]$)
10: CNOT($X[k][5]$, $X[k][7]$), CNOT($X[k][1]$, $X[k][7]$)
11: CNOT($X[k][0]$, $X[k][7]$)
12: **return** ($X[k][1]$, $X[k][0]$, $X[k][3]$, $X[k][7]$, $X[k][2]$,
13: $X[k][4]$, $X[k][5]$, $X[k][6]$)

Algorithm 9 Quantum circuit for $F_1(X[k])$

Input: $X[k]$
Output: $X[k]$
1: CNOT($X[3]$, $X[4]$), CNOT($X[1]$, $X[4]$)
2: CNOT($X[2]$, $X[3]$), CNOT($X[0]$, $X[3]$)
3: CNOT($X[1]$, $X[2]$), CNOT($X[7]$, $X[2]$)
4: CNOT($X[0]$, $X[1]$), CNOT($X[6]$, $X[1]$)
5: CNOT($X[7]$, $X[0]$), CNOT($X[5]$, $X[0]$)
6: CNOT($X[6]$, $X[7]$), CNOT($X[4]$, $X[7]$)
7: CNOT($X[3]$, $X[7]$), CNOT($X[2]$, $X[7]$)
8: CNOT($X[0]$, $X[7]$), CNOT($X[5]$, $X[7]$)
9: CNOT($X[0]$, $X[6]$), CNOT($X[7]$, $X[6]$)
10: CNOT($X[4]$, $X[6]$), CNOT($X[1]$, $X[6]$)
11: CNOT($X[7]$, $X[5]$), CNOT($X[6]$, $X[5]$)
12: CNOT($X[3]$, $X[5]$), CNOT($X[0]$, $X[5]$)
13: **return** ($X[k][5]$, $X[k][6]$, $X[k][7]$, $X[k][0]$, $X[k][1]$,
14: $X[k][2]$, $X[k][3]$, $X[k][4]$)

3.3 CHAM

In the circuit depth, LEA and HIGHT provide performance improvements of 78 % and 85 %, and CHAM provides 70 % improvement. In this section, we describe parallel addition in CHAM. We discuss the performance differences between CHAM, LEA, and HIGHT in Sect. 4. Figure 4 shows round functions of CHAM and proposed technique at the same time.

3.3.1 Round function

We focus on four round functions ($i = 0, 1, 2, 3$) to describe the parallel point. We perform the parallel addition for three rounds ($i = 0, 1, 2$). In the blue box (i.e., $i = 3$)

Algorithm 10 Quantum circuit for round function of HIGHT (first round).

Input: $X[0] \sim [7]$, $K[0] \sim [3]$, $\delta_{0\sim3}$, $c_{0\sim3}$
Output: $X[0] \sim [7]$, $\delta_{4\sim7}$
1: **Generate** $RK[0]$ **and transform** $X[0]$**:**
2: $RK[0] \leftarrow ADD(\delta_0, K[0], c_0)$
3: $X[0] \leftarrow F_1(X[0])$
4: $X[0] \leftarrow \text{CNOT8}(RK[0], X[0])$
5: $X[1] \leftarrow ADD(X[0], X[1], c_0)$

6: **Generate** $RK[1]$ **and transform** $X[2]$**:**
7: $RK[1] \leftarrow ADD(\delta_1, K[1], c_1)$
8: $X[2] \leftarrow F_0(X[2])$
9: $X[2] \leftarrow ADD(RK[1], X[2], c1)$
10: $X[3] \leftarrow \text{CNOT8}(X[2], X[3])$

11: **Generate** $RK[2]$ **and transform** $X[4]$**:**
12: $RK[2] \leftarrow ADD(\delta_2, K[2], c_2)$
13: $X[4] \leftarrow F_1(X[4])$
14: $X[4] \leftarrow \text{CNOT8}(RK[2], X[4])$
15: $X[5] \leftarrow ADD(X[4], X[5], c_2)$

16: **Generate** $RK[3]$ **and transform** $X[6]$**:**
17: $RK[3] \leftarrow ADD(\delta_3, K[3], c_3)$
18: $X[6] \leftarrow F_0(X[6])$
19: $X[6] \leftarrow ADD(RK[3], X[6], c_3)$
20: $X[7] \leftarrow \text{CNOT8}(X[6], X[7])$

21: Reverse(**generate** $RK[0]$ **and transform** $X[0]$)
22: Reverse(**generate** $RK[1]$ **and transform** $X[2]$)
23: Reverse(**generate** $RK[2]$ **and transform** $X[4]$)
24: Reverse(**generate** $RK[3]$ **and transform** $X[6]$)
25: $\delta_{4\sim7} \leftarrow$ Update $\delta_{0\sim3}$ by 4
26: **return** $X[7]$, $X[0]$, $X[1]$, $X[2]$, $X[3]$, $X[4]$, $X[5]$, $X[6]$, $\delta_{4\sim7}$

of Fig. 4, we need the addition result ($X'[0]$) of the round function when the condition is $i = 0$. This is reason that the parallel addition is only available in three rounds. Afterward, we prepare values for the parallel addition.

First, we allocate three additional qubits (c_0, c_1, and c_2) for the parallel addition. In the red box, RK is XORed to X. Since X is required in the next addition, we XOR X to RK using CNOT gates. In the previous implementation, RK is XORed to X, changing the value of X, making parallel addition infeasible [21, Figure 12]. Since the proposed method does not change the value of X, parallel addition is available. The rotation operation on X reverts back after updating RK. Lastly, we need three round keys. In the previous implementation, t additional qubits are allocated for round key generation ($t = 3$ for 64-bit plaintext, $t = 11$ for 128-bit plaintext). We allocate $(3 \times t)$ qubits to generate three round keys. In this way, the parallel addition is performed in three units.

Algorithm 11 describes a quantum circuit for a round function in CHAM-64/128 when i is odd. Since the number of rotation changes depending on whether i is odd or not (see Fig. 4), the quantum circuit is omitted when the condition is $i \neq odd$. Algorithm 12 describes a quantum circuit for round functions of CHAM-128/128,

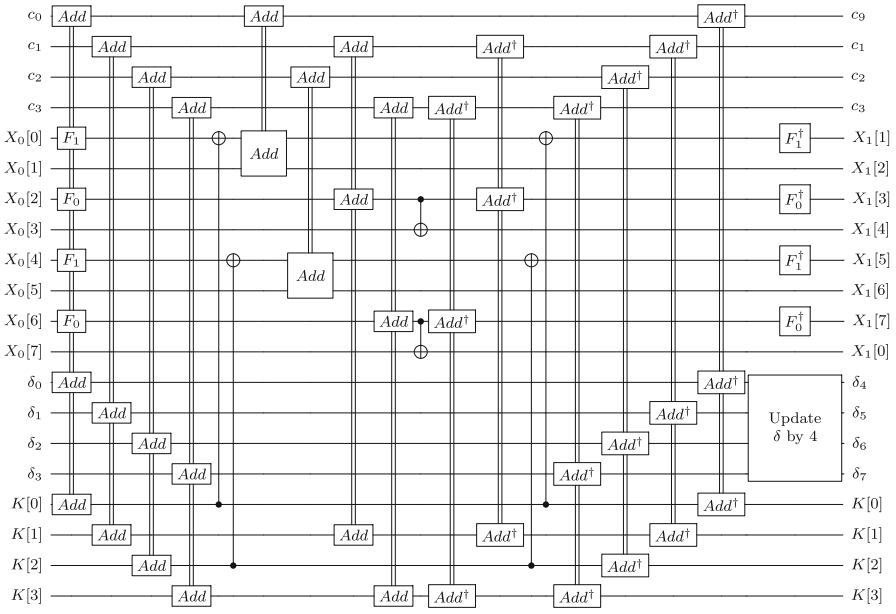

Fig. 3 Quantum circuit for HIGHT

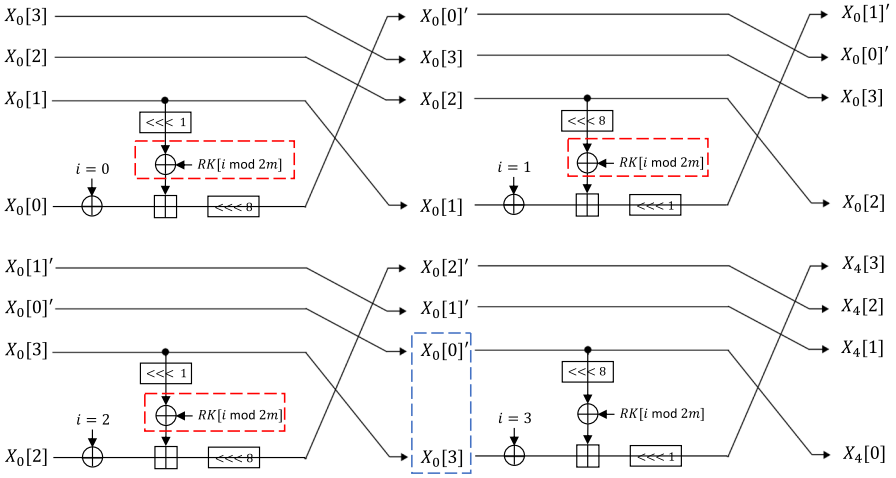

Fig. 4 Four round functions of CHAM ($i = 0, 1, 2, 3$)

128/256. The input of CAHM-64/128 is 64-qubit $X(X[0] \sim [3])$, and the input of CHAM-128/128 and 128/256 is 128-qubit $X(X[0] \sim [3])$. The difference between Algorithms 11 and 12 is that it increases from 16-qubit unit operation (i.e., CNOT, ADD) to 32-qubit unit operation.

Algorithm 11 Quantum circuit for round function of CHAM-64/128 (i is odd).

Input: $X[0] \sim [3], RK, c$
Output: $X[0] \sim [3]$
//Add round constant(i)
1: **for** $j = 0$ to 7 **do**
2: **if** $(i \gg j)$ & 1 **then**
3: $X[0] \leftarrow$X($X[0][j]$)
4: **end if**
5: **end for**

//Preparation for parallel addition
6: $X[1] \leftarrow X[1] \lll 8$
7: $RK \leftarrow$ CNOT16($X[1], RK$)
8: $X[1] \leftarrow X[1] \ggg 8$

//Parallel addition part
9: $X[0] \leftarrow ADD(RK, X[0], c)$

//Reverse
10: $X[1] \leftarrow X[1] \lll 8$
11: $RK \leftarrow$ CNOT16($X[1], RK$)
12: $X[1] \leftarrow X[1] \ggg 8$
13: **return** $X[1], X[2], X[3], X[0] \lll 1$

3.3.2 Key schedule

The quantum circuit for the key schedule is the same as [21] and is similar to the F_0 and F_1 functions of HIGHT. The key schedule of CHAM is also performed as an internal mixing operation using only CNOT gates. Since the generation of $RK[i]$ and $RK[i+w]$ is similar (w is 8 or 4), only the key schedule of CHAM-64/128 for $RK[i]$ is described in Algorithm 13. After K is used as the round key $RK[i]$, it is returned to K by reverse operation. This is because it must be used again when generating $RK[(i+w) \oplus 1]$.

Algorithm 12 Quantum circuit for round function of `CHAM-128/128`, `128/256` (i is odd).

Input: $X[0] \sim [3]$, RK, c
Output: $X[0] \sim [3]$
//Add round constant(i)
1: **for** $j = 0$ to 7 **do**
2: **if** $(i \gg j)$ & 1 **then**
3: $X[0] \leftarrow$X($X[0][j]$)
4: **end if**
5: **end for**

//Preparation for parallel addition
6: $X[1] \leftarrow X[1] \lll 8$
7: $RK \leftarrow$ CNOT32($X[1]$, RK)
8: $X[1] \leftarrow X[1] \ggg 8$

//Parallel addition part
9: $X[0] \leftarrow ADD(RK, X[0], c)$

//Reverse
10: $X[1] \leftarrow X[1] \lll 8$
11: $RK \leftarrow$ CNOT32($X[1]$, RK)
12: $X[1] \leftarrow X[1] \ggg 8$
13: **return** $X[1]$, $X[2]$, $X[3]$, $X[0] \lll 1$

Algorithm 13 Quantum circuit for key schedule of `CHAM-64/128`($RK[i]$)

Input: $K[i]$
Output: $RK[i]$
1: CNOT($K[i][0]$, $K[i][8]$), CNOT($K[i][1]$, $K[i][9]$)
2: CNOT($K[i][2]$, $K[i][10]$), CNOT($K[i][3]$, $K[i][11]$)
3: CNOT($K[i][4]$, $K[i][12]$), CNOT($K[i][5]$, $K[i][13]$)
4: CNOT($K[i][6]$, $K[i][14]$), CNOT($K[i][15]$, $K[i][6]$
5: **for** $j = 0$ to 5 **do**
6: CNOT($K[i][j + 9]$, $K[i][j]$)
7: **end for**
8: CNOT($K[i][7]$, $K[i][6]$), CNOT($K[i][8]$, $K[i][15]$)
9: CNOT($K[i][8]$, $K[i][7]$)
10: **for** $j = 0$ to 6 **do**
11: CNOT($K[i][j]$, $K[i][j + 8]$)
12: **end for**
13: **return** $K[i][15]$, $K[i][0]$, ..., $K[i][14]$

4 Evaluation

In this section, we discuss performance improvements of Korean block ciphers on quantum circuits. We estimate costs for Grover's key search based on the proposed quantum circuits for Korean block ciphers. We utilize the quantum programming tool ProjectQ (on a classic computer) to implement quantum circuits for `LEA`, `HIGHT`, and `CHAM`. `ClassicalSimulator`, an internal library, is used to simulate quantum circuits to verify implementation suitability. Using this library, we can check test vectors for ciphers. Another library, `ResourceCounter`, is used to analyze quantum resources. Finally, we evaluate the post-quantum security strength. We apply post-

Table 1 Quantum resources required for previous implementations [21]

Cipher		Qubits	Toffoli gates	CNOT gates	X gates	Depth
LEA	128/128	385	10,416	28,080	68	26,328
	128/192	513	15,624	39,816	100	39,452
	128/256	641	17,856	45,504	130	45,057
HIGHT	64/128	201	6272	20,523	4	16,447
CHAM	64/128	196	2400	12,285	240	7,807
	128/128	268	4960	26,885	240	19,880
	128/256	396	5952	32,277	304	23,856

Table 2 Quantum resources required for proposed implementations

Cipher		Qubits	Toffoli gates	CNOT gates	X gates	Depth
LEA (this work)	128/128	388	10,248	32,616	11,152	6505
	128/192	518	15,372	46,620	17,004	7589
	128/256	582	17,568	53,280	19,494	8580
HIGHT (this work)	64/128	228	5824	22,614	4496	2479
CHAM (this work)	64/128	204	2320	13,200	2320	2615
	128/128	292	4880	28,760	4880	5307
	128/256	420	5856	34,944	5872	6594

quantum security strength estimation for symmetric key cryptography presented by NIST [31].

4.1 Comparison of quantum circuit implementations for Korean block ciphers

Table 1 shows quantum resources required for the Korean block ciphers implemented in [21]. Table 2 shows quantum resources required for the implementations presented in this paper. The design of quantum circuits in [21] is focused on saving qubits and does not take into account the parallelism of additions. On the other hand, by adopting the optimal quantum adder and performing parallel addition, optimized implementation gives 78 % improvement in LEA, 85 % in HIGHT and 70 % in CHAM compared to previous implementations in terms of depth, respectively. The circuit depth is estimated from the beginning of the circuit to the end, and it is related to the execution time [9].

For LEA-128/128, four additions are performed in a parallel way in the key schedule, and for LEA-128/192 and LEA-128/256, six additions are performed in a parallel way. In CHAM, three additions are performed in a parallel way in the round function. Therefore, the performance improvement is higher in the case of LEA than that of CHAM. HIGHT provides the highest performance improvement, as the four additions are performed in parallel in both the round function and key schedule. In CHAM, for every three consecutive round functions, three additions are performed in parallel.

Table 3 Quantum resources required for Grover oracle

Cipher		r	Qubits	T gates	Clifford gates	Total gates	Depth
LEA	128/128	1	389	147,484	251,504	398,988	13,011
	128/192	2	1037	438,524	746,400	1,184,924	15,179
	128/256	2	1165	500,012	853,272	1,353,284	17,161
HIGHT	64/128	2	457	167,084	294,808	461,892	4,959
CHAM	64/128	2	409	68,972	136,320	205,292	5,231
	128/128	1	293	72,332	145,360	217,692	10,615
	128/256	2	841	172,076	350,656	522,732	13,189

In the resource evaluation, the number of qubits is also an important factor. At the current level of quantum computer development, there are not enough qubits to execute proposed circuits. For this reason, the number of qubits is tied to when the circuit can actually work. The proposed implementation reduces the depth while keeping the number of qubits low. For the LEA-128/256 case, the number of qubits is reduced. The number of Toffoli gates, which are high-cost quantum gates, is reduced due to improvements in the quantum adder itself.

4.2 Cost estimation for Grover oracle

We estimate the cost of the oracle of Grover's search algorithm. The proposed quantum circuit is located in the oracle, and the plaintext is encrypted with the key in the superposition state. The generated ciphertext is compared to the known ciphertext and reverse operations are performed.

In [15], Grassl et al. estimated the cost of the Grover key search for AES and suggested that r plaintext–ciphertext pairs are required to recover a unique key ($r = 3, 4, 5$ for AES-128, 192, 256). Later, Langenberg et al. suggested that $r = \lceil k/n \rceil$ (i.e., key size/block size) is sufficient to recover a unique key [30]. Based on the approach in [30], we assume that $r = \lceil k/n \rceil$ and estimate the cost of oracle. Encryptions (r) can be performed in parallel.

Finally, for oracle, ($2 \times r \times$ Table 2) quantum gates and ($r \times$ Table 2+1) qubits are used and the depth is ($2 \times$ Table 2+1). For the analysis at the Clifford + T level, we decompose the Toffoli gate into seven T gates + eight Clifford gates and a T depth of 4 following the decomposition in [2]. X gates and CNOT gates are counted as Clifford gates. For the n-bit ciphertext, an $n \cdot r$ multi-controlled NOT gate is used to check whether it matches the known ciphertext. It is decomposed into $(32 \cdot n \cdot r - 84)$ T gates [36]. Multi-controlled NOT gates also use one additional qubit, which is flipped if the ciphertext matches. Table 3 shows the quantum resources required for Grover oracle. In Oracle, when $r \geq 2$, plaintexts(r) are encrypted with the same key. There is a room for optimization of qubits and quantum gates. However, for simplicity of estimation and following Grassl's approach [15], this optimization is not taken into account.

Table 4 Cost estimation for Grover key search

Cipher		Qubits	Total gates	Total depth	Cost	NIST security
LEA	128/128	389	$1.195 \cdot 2^{82}$	$1.247 \cdot 2^{77}$	$1.491 \cdot 2^{159}$	Not achieved
	128/192	1037	$1.775 \cdot 2^{115}$	$1.455 \cdot 2^{109}$	$1.292 \cdot 2^{225}$	Level 1
	128/256	1165	$1.014 \cdot 2^{148}$	$1.645 \cdot 2^{141}$	$1.668 \cdot 2^{289}$	Level 3
HIGHT	64/128	457	$1.384 \cdot 2^{82}$	$1.901 \cdot 2^{75}$	$1.316 \cdot 2^{158}$	Not achieved
CHAM	64/128	409	$1.23 \cdot 2^{81}$	$1.003 \cdot 2^{76}$	$1.234 \cdot 2^{157}$	Not achieved
	128/128	293	$1.304 \cdot 2^{81}$	$1.018 \cdot 2^{77}$	$1.328 \cdot 2^{158}$	Not achieved
	128/256	841	$1.566 \cdot 2^{146}$	$1.264 \cdot 2^{141}$	$1.98 \cdot 2^{287}$	Level 3

Level 1 : 2^{170}, Level 3 : 2^{233}, Level 5 : 2^{298}

4.3 Cost estimation for Grover key search

Grover search algorithm is well known for reducing complexity $O(2^k)$ to $\sqrt{2^k}$. After Grover's algorithm is published, M. Boyer et al. provided a tight analysis of Grover's search algorithm and suggested $\lfloor \frac{\pi}{4}\sqrt{2^k} \rfloor$ instead of $\sqrt{2^k}$ as the optimal number of iterations [10]. Thus, we estimate Table 3 $\times \lfloor \frac{\pi}{4}\sqrt{2^k} \rfloor$ excluding qubits as the cost of Grover key search and is shown in Table 4.

Costs estimated in Table 4 consider only Oracle, not the diffusion operator. As mentioned earlier (Sect. 2.5), the diffusion operator occupies less resources in Grover's search. Most of the researches also ignore the cost of the diffusion operator [3, 6, 7, 15, 30], and our estimation method is as follows.

4.4 Evaluation of post-quantum security strength

- **Level 1**: Any attack that breaks the relevant security definition must require computational resources comparable to or greater than those required for key search on a block cipher with a 128-bit key (e.g., AES-128)
- **Level 3**: Any attack that breaks the relevant security definition must require computational resources comparable to or greater than those required for key search on a block cipher with a 192-bit key (e.g., AES-192)
- **Level 5**: Any attack that breaks the relevant security definition must require computational resources comparable to or greater than those required for key search on a block cipher with a 256-bit key (e.g., AES-256)

NIST presents the following post-quantum security strengths based on the cost of Grover key search for AES [31] estimated by Grassl et al [15]. NIST estimates the cost for AES-128, 192 and 256 as 2^{170}, 2^{233} and 2^{298} which are (total gates $\times$ total depth) in Grassl et al's implementations [15]. We compare the cost of Grover key search for Korean block ciphers with the post-quantum security strength presented by NIST. The cost of key search for LEA-128/128, HIGHT, CHAM-64/128, CHAM-128/128 using a 128-bit key is less than the cost of AES-128, which is Level 1 (2^{170}). Therefore, no security level is achieved. In the post-quantum era, increas-

ing the key size for symmetric key cryptography is a well-known countermeasure. LEA-128/192 using a 192-bit key requires $\lfloor \frac{\pi}{4}\sqrt{2^{192}} \rfloor$ searches, which increases the cost of key search. Compared to AES-192(2^{233}), which has the same key size, LEA-128/192 is exposed to attack at a lower cost and achieves only Level 1. In the case of LEA-128/256 and CHAM-128/256 using a 256-bit key, the security level also increases according to the key size, but only achieves Level 3 because it is exposed to attack at a lower cost than AES-256(2^{298}).

5 Conclusion and future work

In this paper, we present optimized quantum circuit implementations for Korean block ciphers. The parallel addition is applied by analyzing the structure of Korean block ciphers, which are ARX architectures. This provides a high performance improvement for quantum circuit depth reduction. Further, we estimate the Grover search cost and evaluate the post-quantum security level of Korean block ciphers by following NIST standards.

Proposed implementations contribute to quantum circuit optimization technology and quantum cryptanalysis for Korean block ciphers. Future work is to apply Grover search algorithm to cryptanalysis rather than exhaustive key search. One such prominent candidate would be differential or linear cryptanalysis. Symmetric key cryptography from attacks by quantum computers can be effective. In [3], authors do a resource analysis of quantum differential cryptanalysis for SPECK and show that it is more powerful than quantum brute force attack. Even in a quantum differential attack, it is important to implement an efficient quantum circuit for the target cipher, and the proposed work can be leveraged.

Data Availability Source codes of proposed implementations are available in https://github.com/starj1023/Korea-ARX-QC/.

References

1. Almazrooie, M., Samsudin, A., Abdullah, R., Mutter, K.: Quantum reversible circuit of AES-128. Quantum Inf. Process. **17**(03), 1–30 (2018)
2. Amy, M., Maslov, D., Mosca, M., Roetteler, M.: A meet-in-the-middle algorithm for fast synthesis of depth-optimal quantum circuits. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **32**(6), 818–830 (2013)
3. Anand, R., Maitra, A., Mukhopadhyay, S.: Evaluation of quantum cryptanalysis on speck. In: Bhargavan, K., Oswald, E., Prabhakaran, M. (eds.) Progress in Cryptology—INDOCRYPT 2020, pp. 395–413. Springer, Cham (2020)
4. Anand, R., Maitra, A., Mukhopadhyay, S.: Evaluation of quantum cryptanalysis on SPECK. In: Bhargavan, K., Oswald, E., Prabhakaran, M. (eds.) Progress in Cryptology—INDOCRYPT 2020, pp. 395–413. Springer, Cham (2020)
5. Anand, R., Maitra, A., Mukhopadhyay, S.: Grover on SIMON. Quantum Inf. Process. **19**, 340 (2020)
6. Anand, R., Maitra, S., Maitra, A., Mukherjee, C.S., Mukhopadhyay, S.: Resource estimation of Grovers-kind quantum cryptanalysis against FSR based symmetric ciphers. Cryptology ePrint Archive, Report 2020/1438 (2020) https://ia.cr/2020/1438
7. Baksi, A., Jang, K.B., Song, G., Seo, H., Xiang, Z.: Quantum implementation and resource estimates for rectangle and knot. Quantum Inf. Process. **20**, 395 (2021)

8. Banegas, G., Bernstein, D.J., van Hoof, I., Lange, T.: Concrete quantum cryptanalysis of binary elliptic curves. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2021**(1), 451–472 (2020)
9. Bhattacharjee, D., Chattopadhyay, A.: Depth-optimal quantum circuit placement for arbitrary topologies. CoRR abs/1703.08540 (2017)
10. Boyer, M., Brassard, G., Häyer, P., Tapp, A.: Tight bounds on quantum searching. Fortschr. Phys. **46**(4–5), 493–505 (1998)
11. Chauhan, A., Sanadhya, S.: Quantum resource estimates of Grover's key search on ARIA. In: International Conference on Security, Privacy, and Applied Cryptography Engineering, pp. 238–258. Springer, Cham (2020)
12. Cuccaro, S., Draper, T., Kutin, S., Moulton, D.: A new quantum ripple-carry addition circuit. 11 (2004)
13. Dasu, V.A., Baksi, A., Sarkar, S., Chattopadhyay, A.: LIGHTER-R: optimized reversible circuit implementation for sboxes. In: 32nd IEEE International System-on-Chip Conference, SOCC 2019, Singapore, September 3–6, 2019, pp. 260–265 (2019)
14. Gidney, C.: Factoring with $n + 2$ clean qubits and n-1 dirty qubits (2018)
15. Grassl, M., Langenberg, B., Roetteler, M., Steinwandt, R.: Applying Grover's algorithm to AES: quantum resource estimates. In: Post-Quantum Cryptography, pp. 29–43. Springer, Cham (2016)
16. Grover, L.K.: A fast quantum mechanical algorithm for database search. In: Proceedings of the twenty-eighth annual ACM symposium on theory of computing, pp. 212–219 (1996)
17. Häner, T., Jaques, S., Naehrig, M., Roetteler, M., Soeken, M.: Improved quantum circuits for elliptic curve discrete logarithms. In: International Conference on Post-Quantum Cryptography, pp. 425–444. Springer, Cham (2020)
18. Häner, T., Roetteler, M., Svore, K. M.: Factoring using $2n + 2$ qubits with Toffoli based modular multiplication. Preprint at arXiv:1611.07995 (2016)
19. 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: Kim, Y., Lee, H., Perrig, A. (eds.) Information Security Applications, pp. 3–27. Springer, Cham (2014)
20. Hong, D., Sung, J., Hong, S., Lim, J., Lee, S., Koo, B.S., Lee, C., Chang, D., Lee, J., Jeong, K., Kim, H., Kim, J., Chee, S.: Hight: a new block cipher suitable for low-resource device. In: Goubin, L., Matsui, M. (eds.) Cryptographic Hardware and Embedded Systems—CHES 2006, pp. 46–59. Springer, Berlin (2006)
21. Jang, K., Choi, S., Kwon, H., Kim, H., Park, J., Seo, H.: Grover on Korean block ciphers. Appl. Sci. **10**(18), 6407 (2020)
22. Jang, K., Kim, H., Eum, S., Seo, H.: Grover on GIFT. Cryptology ePrint Archive, Report 2020/1405 (2020) https://eprint.iacr.org/2020/1405
23. Jang, K., Baksi, A., Breier, J., Seo, H., Chattopadhyay, A.: Quantum implementation and analysis of default. Cryptology ePrint Archive (2022)
24. Jang, K., Baksi, A., Song, G., Kim, H., Seo, H., Chattopadhyay, A.: Quantum analysis of aes. Cryptology ePrint Archive (2022)
25. Jang, K., Choi, S., Kwon, H., Seo, H.: Grover on SPECK: Quantum resource estimates. Cryptology ePrint Archive, Report 2020/640 (2020) https://ia.cr/2020/640
26. Jang, K., Song, G., Kim, H., Kwon, H., Kim, H., Seo, H.: Efficient implementation of present and gift on quantum computers. Appl. Sci. **11**(11), 4776 (2021)
27. Jang, K., Song, G., Kwon, H., Uhm, S., Kim, H., Lee, W.K., Seo, H.: Grover on pipo. Electronics **10**(10), 1194 (2021)
28. 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, pp. 280–310. Springer, Cham (2020)
29. 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: Kim, H., Kim, D.C. (eds.) Information Security and Cryptology—ICISC 2017, pp. 3–25. Springer, Cham (2018)
30. Langenberg, B., Pham, H., Steinwandt, R.: Reducing the cost of implementing the advanced encryption standard as a quantum circuit. IEEE Trans. Quantum Eng. **1**, 1–12 (2020)
31. NIST.: Submission requirements and evaluation criteria for the post-quantum cryptography standardization process (2016) https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/call-for-proposals-final-dec-2016.pdf

32. Roetteler, M., Naehrig, M., Svore, K. M., Lauter, K.: Quantum resource estimates for computing elliptic curve discrete logarithms. In: International Conference on the Theory and Application of Cryptology and Information Security, pp. 241–270. Springer, Cham (2017)
33. Shor, P.: Algorithms for quantum computation: discrete logarithms and factoring. In: Proceedings 35th Annual Symposium on Foundations of Computer Science, pp. 124–134 (1994)
34. Steiger, D.S., Häner, T., Troyer, M.: ProjectQ: an open source software framework for quantum computing. Quantum **2**, 49 (2018)
35. Takahashi, Y., Tani, S., Kunihiro, N.: Quantum addition circuits and unbounded fan-out. Preprint at arXiv:0910.2530 (2009)
36. Wiebe, N., Roetteler, M.: Quantum arithmetic and numerical analysis using Repeat-Until-Success circuits. arXiv:1406.2040 (2014)