applied sciences

MDPI

Article

Optimized Implementation and Analysis of CHAM in Quantum Computing

Yujin Yang [1], Kyungbae Jang [1], Anubhab Baksi [2] and Hwajeong Seo [1,*]

[1] Division of IT Convergence Engineering, Hansung University, Seoul 02876, Republic of Korea
[2] School of Computer Science and Engineering, Nanyang Technological University, Singapore 639798, Singapore
* Correspondence: hwajeong@hansung.ac.kr; Tel.: +82-760-8033

Abstract: A quantum computer capable of running the Grover search algorithm, which reduces the complexity of brute-force attacks by a square root, has the potential to undermine the security strength of symmetric-key cryptography and hash functions. Recently, studies on quantum approaches have proposed analyzing potential quantum attacks using the Grover search algorithm in conjunction with optimized quantum circuit implementations for symmetric-key cryptography and hash functions. Analyzing quantum attacks on a cipher (i.e., quantum cryptanalysis) and estimating the necessary quantum resources are related to evaluating post-quantum security for the target cryptography algorithms. In this paper, we revisit quantum implementations of CHAM block cipher, an ultra lightweight cipher, with a focus on optimizing the linear operations in its key schedule. We optimized the linear equations of CHAM as matrices by applying novel optimized decomposition techniques. Using the improved CHAM quantum circuits, we estimate the cost of Grover's key search and evaluate the post-quantum security strength with further reduced costs.

Keywords: quantum computer; Grover's algorithm; lightweight block cipher; CHAM; linear layer optimization

Citation: Yang, Y.; Jang, K.; Baksi, A.; Seo, H. Optimized Implementation and Analysis of CHAM in Quantum Computing. *Appl. Sci.* **2023**, *13*, 5156. https://doi.org/10.3390/app13085156

Academic Editors: Juan A. Gómez-Pulido and Hai Jiang

Received: 9 February 2023
Revised: 12 April 2023
Accepted: 18 April 2023
Published: 20 April 2023

1. Introduction

In the recent years, we have seen a massive interest in research and development in quantum computing technology. There are many major players in the field, including international conglomerates such as IBM and Google, as well as universities and national research centers. They have been putting a lot of effort into the development of quantum computing technology. As a result, Google's research team, which had achieved quantum supremacy with its superconducting processor Sycamore in 2019 [1], recently demonstrated experimentally that errors can be reduced by increasing the number of logical qubits in their superconducting system [2]. In addition, the USTC research team demonstrated a quantum computational advantage with a 60-qubit superconducting quantum processor [3], and the Xanadu's team reported quantum computational advantages by showing that Gaussian Boson Sampling (GBS) tasks can be performed via a programmable photonic processor, Borealis [4]. However, the quantum computers—if they become functional—will massively disrupt a lot of the cryptographic systems that are presently deployed in various industrial applications. There are many facets, but in particular, the Grover's search algorithm [5] can be applied against any symmetric key ciphers [6] to reduce the security claim to the square root bound of what is expected against a classical computer. In addition to the Grover's algorithm, there are various quantum or search optimization algorithms such as Shor [7], Simon [8], and Particle Swarm Optimization algorithm [9,10]. Other algorithms related to quantum computers include Classical-Quantum hybrid algorithms [11], which are mainly applied to quantum machine learning and quantum chemistry, unlike the quantum algorithms (i.e., Shor, Simon, and Grover) mentioned above. As such, although

there are various algorithms related to quantum computers, it is common to use Grover's algorithm in the symmetric-key cryptosystem. While it is acknowledged that a functional quantum computer has yet to see the light of the day, the researchers have been preemptively working against this emerging threat. There has also been a standardization effort by National Institute of Standards and Technology (NIST), a security organization run by the US government, to classify the quantum security level with respect of Advanced Encryption Standard (AES) for encryption and Secure Hash Algorithm (SHA) for hash [12]. In July 2022, four selected Post-Quantum Cryptography (PQC) algorithms have announced and created a new draft standard for the corresponding algorithms currently in progress [13]. Now, in order to apply the Grover's search algorithm, one needs to implement the cipher in quantum logic—the search complexity varies based on the quantum cost of the cipher implementation. Naturally, researchers have noted this, and have been working on finding an optimized quantum implementation of symmetric key ciphers (including [14,15], etc.). In order to reduce the quantum cost, while the previous optimization research has focused only on reducing the number of qubits [16,17], more recent research has further been considering reducing the overall depth and T-depth of quantum circuits [18–20]. Most of these studies also deal with estimating the attack cost of recovering the secret key of a symmetric key cipher via Grover's algorithm to assess security strength.

At the same time, we have also witnessed a new design in the symmetric key ciphers, referred to as lightweight ciphers, which are specialized to work on tightly constrained devices [21]. Since lightweight block ciphers are essential to reduce costs with similar performance, methods of optimizing and implementing symmetric keys in a linear layer for more efficient use in lightweight environments have been steadily discussed. This includes studies [22,23] that optimize methods using matrix decomposition, such as PLU factorization and Gaussian–Jordan elimination, by applying them to symmetric keys. Thus, applying various matrix decomposition methods to lightweight block cipher quantum circuit implementations could greatly help circuit optimization and the final cost savings of a key recovery attack using Grover's algorithm.

In this paper, we optimize the quantum circuits of CHAM, one of the Korean lightweight block ciphers, by applying two linear layer optimization techniques such as matrix decomposition to its key-schedule. Then, we compare and evaluate the estimated costs of Grover's algorithm attack for the existing research [24] and each of the methods.

Contribution

Our contributions in this work are manifold and can be summarized as follows:

1. Optimization to the linear layer of the CHAM block cipher: To the best of our knowledge/understanding, this type of optimization has never been applied to this cipher before. We show how the optimization affects the quantum footprint, by comparing the benchmarks of the naïve versus the optimized implementation.
2. Optimization of CHAM's quantum circuit: Unlike classical computers, quantum computers have limited resources. For this reason, it is essential to optimize the number of qubits in optimizing quantum circuits. In addition, since the depth of a quantum circuit is related to speed and time, not only the number of qubits, but also the depth of the circuit is considered an important factor in optimizing the circuit. We apply two linear layer optimization techniques to CHAM's key schedule to reduce common unnecessary auxiliary qubits used for round key generation in existing structures. Moreover, we also reduce the number of CNOT gates and the depth of the circuit using the second improvement method, and so satisfied two important optimization factors.
3. Evaluation of post-quantum security about improved quantum circuit of CHAM: NIST provides criteria for evaluating the post-quantum security strength of existing codes in preparation for quantum computers. In order to evaluate the security strength, it is necessary to implement the cipher as a quantum circuit and estimate the cost of a Grover's algorithm attack. We estimate the cost of a Grover's algorithm attack

about an improved quantum circuit of CHAM, compare it with the results of the previous one, and then evaluate the post-quantum security strength. Additionally, circuits are implemented for Revised CHAM, and security strength evaluations are conducted together.

The organization of this paper is as follows. Section 2 presents the background of the CHAM cipher, quantum computers and quantum gates, linear layer optimization techniques, and the Grover's algorithm. In Section 3, the proposed quantum implementation is presented. In Section 4, we evaluate the proposed quantum circuit for CHAM. Section 5 estimates the cost of Grover's algorithm for the proposed CHAM quantum circuit, and evaluates the NIST post-quantum security level for CHAM based on the estimated cost. Finally, Section 6 concludes this article.

2. Background

2.1. Description of CHAM

CHAM [25] is a Korean lightweight block cipher that uses a stateless-based round key that targets low-end IoT platforms. It was announced at ICISC'17 and uses a stateless-based round key. This cipher uses the ARX operations (Addition, Rotation, eXclusive-or) and has advantages in a variety of resource-constrained environments. The CHAM family includes three ciphers: CHAM-64/128, CHAM-128/128, and CHAM-128/256 (here, the first parameter indicates the block size n, and the second parameter indicates the key size k). In 2019, a paper that revised CHAM [26] by increasing the number of rounds was proposed to secure a sufficient security margin for CHAM. The revised CHAM is identical to the original CHAM except for the number of rounds. The cipher is also added to the Cryptography libraries https://www.cryptopp.com/wiki/CHAM (accessed on 10 April 2023). In Table 1, the CHAM parameters (n, k, w, r) are shown. The branch(word)'s bit length and the number of rounds are denoted, respectively, by w and r. The first value of r in Table 1 represents the number of rounds in the original CHAM, while the second value represents the number of rounds in the revised CHAM.

Table 1. CHAM's parameters.

Cipher	n	k	w	r
CHAM-64/128	64	128	16	80, 88
CHAM-128/128	128	128	32	80, 112
CHAM-128/256	128	256	32	96, 120

The round key $RK = \{RK_0, RK_1, \ldots, RK_{2k/w-1}\}$ used in the i-th round of the key schedule is generated using the initial keyword K_i, where $(0 \leq i < k/w)$. k/w is the key size k divided by word size w. In 64/128 and 128/256, (k, w) is (128, 16), (256, 32), respectively, so k/w is 8, and 128/128 has (128, 32), so k/w is 4. In order to generate the round key RK of the key schedule, the following equation is used:

$$\begin{aligned} RK_i &= K_i \oplus (K_i \lll 1) \oplus (K_i \lll 8), \\ RK_{(i+k/w)\oplus 1} &= K_i \oplus (K_i \lll 1) \oplus (K_i \lll 11). \end{aligned} \tag{1}$$

In Equation (1), $\lll$ indicates a left-rotation operator. The expression $\lll 8$ performs an operation where bits are shifted to the left by 8 bits, and the bits that fall off at the left edge are put back on the right edge.

The n-bit block plaintext P is divided into four w-bit words and encrypted by repeating the number of rounds corresponding to the parameters in the Table 1. In this case, the operation is different depending on whether r is odd or even. More precisely, the overall structure is the same, but the number of rotations varies depending on whether it is odd or even.

$$\begin{aligned} X_{i+1}[j] &= X_i[j+1], (0 \le j \le 2), \\ X_{i+1}[3] &= ((X_i[0] \oplus i) \boxplus ((X_i[1] \lll 1) \oplus RK_{i \bmod 2m})) \lll a. \end{aligned} \tag{2}$$

The round function is shown in Equation (2) for $0 \le i < r$, and if i is even, a is 8; otherwise, a is 1. In Equation (2), $x \boxplus y$ means addition of x, y modulo 2^w.

Jang et al. [24] optimized quantum circuits for the Korean lightweight block ciphers CHAM, LEA, and HIGHT. In that paper [24], an improved quantum adder [27] is applied, and parallel structure is used by allocating additional (ancilla) qubits in order to reduce in terms of circuit depth. Compared to the results of a previous paper [28], which first implemented CHAM as a quantum circuit, the performance improved by 70%. Although optimized by changing the structure of CHAM to a parallel structure, the round key generation part is the same as in the previous paper [28].

2.2. Quantum Gates

This section briefly mentions what is related to quantum computing and explains the quantum gates required to implement the cryptographic operation of ciphers. In a quantum computer environment, a quantum bit (qubit) is the unit of quantum information, and has the property of being able to have both 0 and 1 at the same time. This property is called quantum superposition, and it exponentially increases the computational space of quantum computers. Gates must be used for computation, but in a quantum computer environment, logic gates such as AND, OR, and XOR used in classical computers cannot be used, so quantum gates are used as a means to replace logic gates.

There are many quantum gates, but a single-qubit gate and controlled gate are generally considered basic. Single-qubit gates that cause the spin to revolve around some axis [29] include X (NOT), H, S gates, etc. Controlled gates including CNOT, Toffoli gate, etc., affect a target qubit through one or more control qubits. Among the quantum gates, the following four quantum gates are often used to implement quantum circuits in cryptography. The four quantum gates are shown in Figure 1.

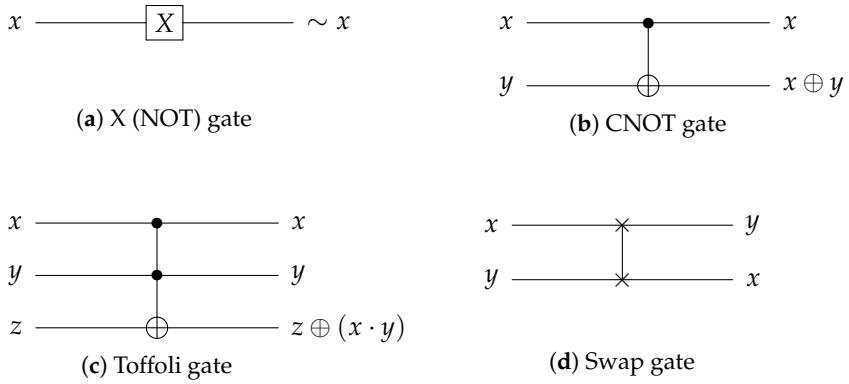
The X gate shown in Figure 1a is the simplest reversible flip gate among the basic gates. The X gate performs an operation that inverts a single qubit that comes as an input. It has the same operation as a classical bit flip and can be matched with a NOT gate among logic gates.

The CNOT gate shown in Figure 1b stands for a Controlled-NOT gate. It inverts the target qubit only when the state of the control qubit is $|1\rangle$. CNOT gates are often used when performing an in-place XOR operation between two inputs.

The Toffoli gate shown in Figure 1c stands for a Controlled–Controlled–NOT gate and is also called CCNOT. It can be seen as a CNOT gate with two control qubits, and the target qubit can be inverted only when both control qubits are $|1\rangle$. The Toffoli gate is not a universal quantum gate, and it can be implemented by a combination of Clifford $+T$ gates. Here, Clifford $+T$ gates are a set of universal quantum gates consisting of Clifford, which includes H gate, S gate, and CNOT gate, and a single T gate, which is non-Clifford. By fixing this set of gates, we can estimate the number of quantum gates, circuit depth, cost, etc. We adopt the method of decomposing Toffoli gates into 8 Clifford $+ 7T$ gates from [30]. In this case, the full depth is 8 and the$T$-depth is 4. The Toffoli gate has the highest quantum cost among the four basic quantum gates of Figure 1.

The Swap Gate shown in Figure 1d exchanges the states of two qubits that come in as inputs. Unlike the previous gates, the SWAP gate does not use quantum resources. Except for the SWAP gate, the quantum resource cost of other gates increases in the mentioned order.

While NAND or NOR gates are universal gates in classical computing, universal quantum gate sets include Clifford gates (X, CNOT, H, S) $+T$ gate, rotation gates, and others. By using these universal quantum gate sets, it is possible to perform any quantum operation. In this paper, we estimate the complexity of total quantum gates at the Clifford + T level.



(**a**) X (NOT) gate (**b**) CNOT gate (**c**) Toffoli gate (**d**) Swap gate

Figure 1. Quantum gates.

2.3. Linear Layer Optimization

Linear equations can be expressed as matrices.

$$\begin{aligned} ax_1 + bx_2 &= y_1 \\ cx_1 + dx_2 &= y_2 \end{aligned} \tag{3}$$

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} \tag{4}$$

For example, when there are two linear equations such as Equation (3), converting them into matrices can be expressed as Equation (4).

2.3.1. PLU Factorization

The PLU factorization allows us an easy way to implement a binary matrix. Given a square matrix M; the algorithm funds a permutation matrix P, a lower triangular matrix L, and an upper triangular matrix U; such that $M = PLU$. The matrix L has a value of 0 except for the lower triangular matrix, and on the contrary, matrix U has 0 except for the upper triangular matrix.

$$M = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \end{bmatrix} = P \cdot L \cdot U = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{5}$$

In Equation (5), the first matrix is arbitrary square matrix M, the second one is matrix P, the third one is matrix L, and the last one is matrix U. In our case, the PLU matrices were obtained utilizing SageMath 9.8 https://www.sagemath.org/ (accessed on 10 April 2023), which is an open-source programming tool for multiple Python-based mathematics. PLU factorization can be used to find the inverse of a non-singular matrix or a solution to a row-linear equation. Applying this method to ciphers can reduce the number of XOR2s in a general circuit and CNOT gates in a quantum circuit.

In the past, PLU factorization was used in various contexts. For instance, the 32×32 binary matrix of AES (the `MixColumn` matrix) [16,20,22,31] or the implementation of an elliptic curve [32].

2.3.2. FSE'20 (Xiang et al. [33])

The work by Xiang et al. [33] was published in the FSE'20, where the authors proposed an in-place algorithm to optimize the implementation of binary matrices. To the best of our finding, this is the first-of-its-kind.

Because of the in-place nature of the implementation (i.e., $x_i \leftarrow x_i \oplus x_j$), it becomes possible to update the same qubits (thus, we do not need ancilla/garbage qubits). At the same time, the algorithm is quite efficient, as the state-of-the-art results for in-place

implementation of several binary matrices (including that of the 32×32 `MixColumn` matrix of AES, which takes 92 XOR2/CNOT gates) are reported in this paper.

2.4. Grover's Algorithm (for Key Search)

Grover's algorithm [5] is a quantum search algorithm that derives the desired answer using the superposition and entanglement principles of quantum mechanics. When the function domain size is n, using the Grover's algorithm reduces the computational complexity to $\sqrt{n}$.

The Grover's algorithm consists of an oracle and a diffusion operator. Among these, the oracle operator is a key component in the algorithm as it has a direct relationship with the overall performance of the algorithm. Namely, it is essential to implement oracle efficiently to optimize the performance of the algorithm.

1. In order to make the key of n-qubit superposition states $|\psi\rangle$, the Hadamard gates are applied to k of n-qubit. This ensures that all states of the qubits have the same amplitude:

$$|\psi\rangle = H^{\otimes n}|0\rangle^{\otimes n} = \left(\frac{|0\rangle + |1\rangle}{\sqrt{2}}\right) = \frac{1}{2^{n/2}} \sum_{k=0}^{2^n-1} |k\rangle \tag{6}$$

2. The attack-target cipher is implemented as a quantum circuit and arranged in oracle $f(k)$. Encrypting the known plaintexts of the superpositioned key produces ciphertexts c for all key values. Afterwards, if the ciphertext matches the known ciphertext, $f(k) = 1$ and $(-1)^{f(k)}|k\rangle|-\rangle$, so the sign of the solution key value is reversed.

$$f(k) = \begin{cases} 1 \text{ if } Enc(key) = c \\ 0 \text{ if } Enc(key) \neq c \end{cases} \tag{7}$$

$$U_f(|\psi\rangle|-\rangle) = \frac{1}{2^{n/2}} \sum_{k=0}^{2^n-1} (-1)^{f(k)}|k\rangle|-\rangle \tag{8}$$

3. In the final step, the diffusion operator amplifies the amplitude of the negative-signed solution key returned by the oracle. Usually, diffusion operators do not require special skills to implement because their design is general.

3. Quantum Implementation of CHAM

In our implementation, the key-schedule of [24] is improved to optimize the quantum circuit of CHAM. Since the calculation processes between the parameters are similar, CHAM 64/128 will be used as an example.

3.1. Binary Matrix in Key Schedule

The round key RK matrix of CHAM is generated using Equation (1) of Section 2.1. The i range of CHAM 64-128 is $(0 \leq i < r)$, so CHAM 64-128 exists from RK_0 to RK_{15}. For RK, indices 0 to 7 use the first formula in Equation (1) and 8 to 15 use the second formula, resulting in a total of two RK matrices. In the case of CHAM-64/128, since the word size w is 16, the matrix obtained through the above process has a size of 16×16. The following is the $RK_{0\sim7}$ matrix of CAHM 64-128:

$$\begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1
\end{bmatrix} \quad (9)$$

The row of the matrix RK shows the result of each rotation operation when i is applied to the key-schedule equation. For example, the second row of $RK_{0\sim7}$ is composed of $(1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0)$, according to Equation (9). When K_i is in column 2, 1 in column 1 is a value of RoL_1, and 1 in column 10 is a value of ROL_8. The matrices of the remaining parameters including CHAM-64/128 are generated using sage math tool.

3.2. PLU Factorization

In order to generate RK using PLU factorization [23], P, L, and U matrices are to be obtained, respectively, as mentioned in Section 2.3. This can also be easily obtained by using the `LU()` https://doc.sagemath.org/html/en/reference/matrices/sage/matrix/matrix2.html#sage.matrix.matrix2.Matrix.LU (accessed on 10 April 2023) function of SageMath.

Algorithm 1 is a quantum circuit implementation of applying U, L. In each matrix, column n represents $k[n-1]$. One thing to note is that the P, L, and U matrices are constants, which classifies Algorithm 1 as a classical-quantum implementation. The condition values (U, L) used in the if statements in Algorithm 1 are all classical. Thus, the if statements used in Algorithm 1 are for designing the quantum circuit (i.e., programming related), not for internal if statements based on the measurement values of qubits within the quantum circuit. Conceptually, the role of the if statements is similar to that of a for the loop.

In the algorithms, not all matrices consist of qubits.The U process proceeds from 1 to 16 rows of the matrix U (i.e., from top to bottom). In a row, it is meaningful because only the upper terms of the diagonal term in the matrix U have non-zero values based on the main diagonal line due to the characteristics of the upper triangular matrix. Therefore, if it searches in the right direction based on the main diagonal line, and if the value is 1, the corresponding element $k[1+i+j]$ is set as the control qubit and the main diagonal element $k[i]$ is set as the target qubit and then CNOT gates are applied.

The L process as opposed to U proceeds in the first row direction in row 16 of the matrix L, i.e., from bottom to top. L is also meaningful only for the left value of the main diagonal due to the characteristics of the lower triangular matrix. Similar to U, we search the left side of the main diagonal reference, and if the value is 1, CNOT gates are applied to the corresponding element $k[w-1-j]$ and the main diagonal element $k[w-1-i]$.

In the P process, the permutation is performed according to the matrix P. Only CNOT gates are used for U and L processes, whereas only SWAP gates are used for P processes, which rearrange the order. In the LU factorization, only the input matrix is different according to the RK index (Section 2.1), but the algorithm is the same, so one operation is used. On the other hand, in the permutation step, it is not easy to define an algorithm for the P matrix for each RK index, so different SWAP gates are applied each time according to the P matrix.

Algorithm 1 Quantum implementation of LU factorization.

Input: Word size w, w-qubit Key $k_{0\sim7}$, Upper triangular matrix U, Lower triangular matrix L
Output: w-qubit key $k_{0\sim7}$

```
// Applying U
for i = 0 to w − 2 do
    for j = 0 to w − i − 2 do
        if U[(i ∗ w) + 1 + i + j]==1 then
            k[i] ← CNOT(k[i + j + 1], k[i])
        end if
    end for
end for

// Applying L
for i = 0 to w − 2 do
    for j = w − 1 to i + 1 step −1 do
        if L[w ∗ (w − 1 − i) + w − 1 − j]==1 then
            k[w − 1 − i] ← CNOT(k[w − 1 − j], k[w − 1 − i])
        end if
    end for
end for
return k_{0∼7}
```

Algorithm 2 describes a quantum implementation of the CHAM-64/128 Key schedule applying PLU factorization. In order to recycle $k_{0\sim7}$, a reverse operation is performed on quantum gates included in the Transform range at Reverse (transform $k_{0\sim7}$). This part is implemented using ProjectQ's `Compute()` and `Uncompute()` commands. Transform corresponds to `Compute()` and Reverse corresponds to `Uncompute()`. Since this paper is about key-schedule optimization, only the location where the encryption step is performed is mentioned as a comment, and the implementation algorithm is omitted.

Figures 2 and 3 represent quantum circuit diagrams of CHAM-64/128 key schedule with PLU factorization for $RK_{0\sim7}$. These diagrams aim to illustrate the quantum operations actually performed when Apply L, U, and permutations are executed. These circuits are only valid for $RK_{0\sim7}$, and the circuit diagram differs for $RK_{8\sim15}$ due to the use of different applied matrices.

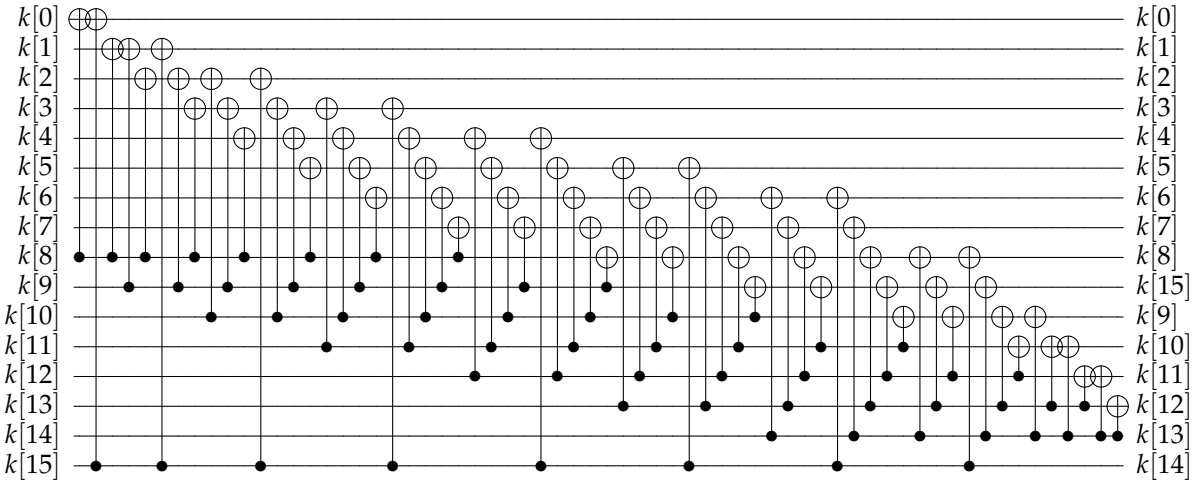


Figure 2. Quantum circuit of the CHAM-64/128 key schedule with PLU factorization (Apply_U), white circle indicates XOR operation and black dot indicates connection.

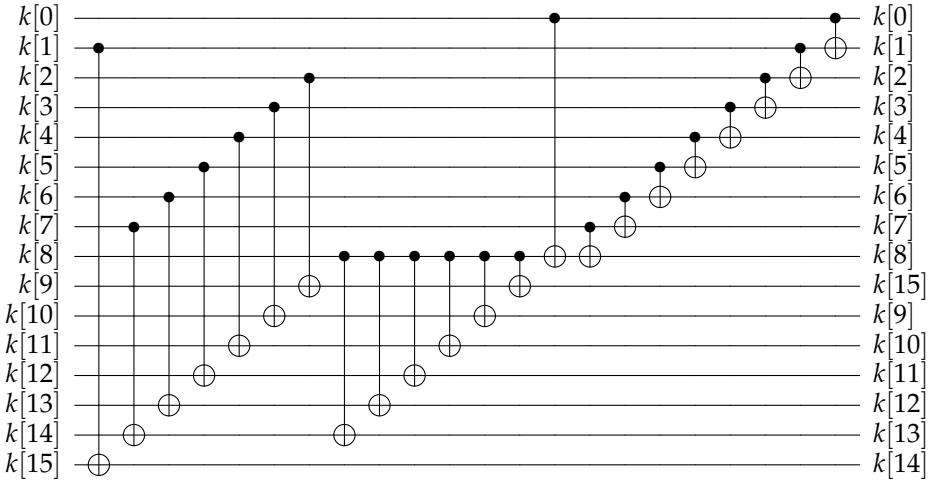


Figure 3. Quantum circuit of the CHAM-64/128 key schedule with PLU factorization (Apply_L), white circle indicates XOR operation and black dot indicates connection.

Algorithm 2 Quantum implementation of CHAM-64/128 key-schedule with PLU factorization.

Input: 16-qubit Key $k_{0\sim7}$, Upper triangular matrices $U_{1,2}$, Lower triangular matrices $L_{1,2}$
Output: Round key $RK_{0\sim15}$

Transform $k_{0\sim7}$:
  for $i = 0$ to 7 **do**
    $k_i \leftarrow$ Apply LU(k_i, 16, U_1, L_1)
    $(k_i[12], k_i[11]) \leftarrow$ SWAP($k_i[11]$, $k_i[12]$)
    $(k_i[11], k_i[13]) \leftarrow$ SWAP($k_i[13]$, $k_i[11]$)
    $(k_i[11], k_i[10]) \leftarrow$ SWAP($k_i[10]$, $k_i[11]$)
    $(k_i[10], k_i[14]) \leftarrow$ SWAP($k_i[14]$, $k_i[10]$)
    $(k_i[10], k_i[9]) \leftarrow$ SWAP($k_i[9]$, $k_i[10]$)
    $(k_i[9], k_i[15]) \leftarrow$ SWAP($k_i[15]$, $k_i[9]$)
  end for
return $k_{0\sim7}$ ▷ $RK_{0\sim7}$
▷ After performing encryption
Reverse (**transform** $k_{0\sim7}$)

Transform $k_{0\sim7}$:
  for $i = 0$ to 7 **do**
    $k_i \leftarrow$ Apply LU(k_i, 16, U_2, L_2)
    $(k_i[13], k_i[11]) \leftarrow$ SWAP($k_i[11]$, $k_i[13]$)
    $(k_i[10], k_i[9]) \leftarrow$ SWAP($k_i[9]$, $k_i[10]$)
    $(k_i[8], k_i[7]) \leftarrow$ SWAP($k_i[7]$, $k_i[8]$)
    $(k_i[6], k_i[5]) \leftarrow$ SWAP($k_i[5]$, $k_i[6]$)
  end for
return $k_{0\sim7}$ ▷ $RK_{8\sim15}$
▷ After performing encryption
Reverse (**transform** $k_{0\sim7}$)
return $k_{0\sim7}$

3.3. FSE'20

The second improvement method is the application of FSE'20 to the key-schedule. The method of applying PLU factorization uses CNOT and SWAP gates, while the method of applying FSE'20 uses only CNOT gates. The SWAP part uses a logical swap method that can include values to Python's list in the order you want them to. Since the process is too complicated to use the SWAP gate, a logical swap is used to intuitively know. Since

the logical swap method does not use quantum gates, the reverse operation is not applied immediately. Therefore, it is necessary to make an operation that performs the opposite process of the logical swap so that it can be put back in the original order in the list.

The generation of round keys $RK_{0\sim7}$ with FSE'20 is described in Algorithm 3. If the reverse operation is performed in Algorithm 3, the reverse function can be applied to lines 1 to 14 in the same way as PLU factorization. Then, k has the original value if the same operation of line 16 to 19 is performed one more time. The reverse operation proceeds in the same way as the aforementioned PLU algorithm.

Algorithm 3 Quantum implementation of CHAM-64/128 Key Schedule applying FSE'20.

Input: 16-qubit Key $k_{0\sim7}$
Output: Round key$RK_{0\sim7}$
1: **Transform** $k_{0\sim7}$:
2: **for** $i = 0$ to 7 **do**
3: $k_i[5] \leftarrow \text{CNOT16}(k_i[13], k_i[5])$, $k_i[2] \leftarrow \text{CNOT16}(k_i[10], k_i[2])$
4: $k_i[15] \leftarrow \text{CNOT16}([k7], k_i[15])$, $k_i[8] \leftarrow \text{CNOT16}(k_i[0], k_i[8])$
5: $k_i[7] \leftarrow \text{CNOT16}(k_i[6], k_i[7])$, $k_i[7] \leftarrow \text{CNOT16}(k_i[14], k_i[7])$
6: $k_i[4] \leftarrow \text{CNOT16}(k_i[12], k_i[4])$, $k_i[14] \leftarrow \text{CNOT16}(k_i[5], k_i[14])$
7: $k_i[12] \leftarrow \text{CNOT16}(k_i[11], k_i[12])$, $k_i[10] \leftarrow \text{CNOT16}(k_i[9], k_i[10])$
8: $k_i[13] \leftarrow \text{CNOT16}(k_i[4], k_i[13])$, $k_i[10] \leftarrow \text{CNOT16}(k_i[1], k_i[10])$
9: $k_i[12] \leftarrow \text{CNOT16}(k_i[3], k_i[12])$, $k_i[9] \leftarrow \text{CNOT16}(k_i[8], k_i[9])$
10: $k_i[1] \leftarrow \text{CNOT16}(k_i[8], k_i[1])$, $k_i[0] \leftarrow \text{CNOT16}(k_i[15], k_i[0])$
11: $k_i[8] \leftarrow \text{CNOT16}(k_i[0], k_i[8])$, $k_i[3] \leftarrow \text{CNOT16}(k_i[2], k_i[3])$
12: $k_i[6] \leftarrow \text{CNOT16}(k_i[5], k_i[6])$, $k_i[4] \leftarrow \text{CNOT16}(k_i[12], k_i[4])$
13: $k_i[11] \leftarrow \text{CNOT16}(k_i[2], k_i[11])$, $k_i[15] \leftarrow \text{CNOT16}(k_i[7], k_i[15])$
14: $k_i[2] \leftarrow \text{CNOT16}(k_i[10], k_i[2])$, $k_i[5] \leftarrow \text{CNOT16}(k_i[13], k_i[5])$

▷ logical swap

15: $RK_i[0] \leftarrow k_i[0]$
16: **for** $j = 15$ to 1 step -1 **do**
17: $RK_i[j - (\lceil j/2 \rceil - 1) * 2] \leftarrow k_i[j]$
18: **end for**
19: **end for**
20: **return** $RK_{0\sim7}$

▷ After performing encryption

21: Reverse (**transform** $k_{0\sim7}$)
22: **return** $k_{0\sim7}$

4. Performance

In this section, we evaluate and compare the performance of proposed CHAM quantum circuits by estimating the quantum resources. However, the proposed quantum circuits cannot yet run in current quantum computers because the size of our circuits are too large. We use ProjectQ, a quantum programming tool provided by IBM, on classical computers to implement and simulate quantum circuits. Thus, the quantum circuits in our work are implemented and simulated at an error-free environment (i.e., logical level). A large number of qubits can be simulated using ProjectQ's own library, `ClassicalSimulator`, which is limited to boolean quantum gates such as X, CNOT, and Toffoli. Quantum gates that affect the phase such as S and H gates cannot be used. `ClassicalSimulator` allows us to validate the implementation by classical calculation of the output of the implemented quantum circuit. To verify the proposed quantum circuits, the test vector provided in [25] (Appendix A) is used for the plaintext, ciphertext, and secret key. If the output of running our quantum circuits in `ClassicalSimulator` matches the ciphertext of the test-vector, we can be confident that the circuits have been implemented correctly. To estimate the detailed quantum resources, we use another internal library called `ResourceCounter`. Unlike the

`ClassicalSimulator`, the `ResourceCounter` does not compute values that are affected by quantum gates. It only calculates the depth of the circuit, and the number of allocated quantum gates and qubits.

The quantum resources needed to implement CHAM quantum circuits at the NCT (NOT(X), CNOT, and Toffoli) level are shown in Tables 2 and 3.

Table 2. Quantum resources required for previous CHAM implementation (JSKKKS'20).

Cipher	#CNOT	#X	#Toffoli	#qubits	Depth
CHAM-64/128	13,200	2320	2320	204	2615
CHAM-128/128	28,760	4880	4880	292	5307
CHAM-128/256	34,944	5872	5856	420	6594

Table 3. Quantum resources required for proposed CHAM implementations.

Method	Cipher	#CNOT	#X	#Toffoli	#qubits	Depth
PLU [23] (this work)	CHAM-64/128	21,360	2320	2320	195	2673
	CHAM-128/128	52,000	4880	4880	259	6431
	CHAM-128/256	62,400	5872	5856	387	6356
FSE'20 [33] (this work)	CHAM-64/128	13,040	2320	2320	195	2612
	CHAM-128/128	28,800	4880	4880	259	5240
	CHAM-128/256	34,560	5872	5856	387	6249

As mentioned in Section 2.2, the Toffoli gate consists of a combination of Clifford and T gates. We decompose one Toffoli gate into 7 Tgates, 8 Clifford gates with T depth 4, full depth 8 using the [30] method, which is widely used in related to quantum studies. Table 4 shows a comparison of the detailed quantum resources required for quantum circuits of the previous and the proposed structure at the Clifford + T level.

Table 4. CHAM quantum resources comparison by methods applied to key-schedule in detail.

CHAM	Methods	#CNOT	#1qCliff	#T	T-Depth	#qubits	Full Depth
64/128	Previous [24]	27,120	6960	16,240	9280	204	17,035
	PLU [23] (this work)	35,280				195	17,092
	FSE'20 [33] (this work)	29,960				195	17,031
128/128	Previous [24]	58,040	14,640	34,160	19,520	292	37,766
	PLU [23] (this work)	81,280				259	37,878
	FSE'20 [33] (this work)	58,080				259	37,768
128/256	Previous [24]	70,080	17,584	40,992	23,424	420	45,252
	PLU [23] (this work)	97,536				387	45,014
	FSE'20 [33] (this work)	69,696				387	44,904

Compared to the existing CHAM implementation, Jang et al. [24] used auxiliary qubits in the key-schedule process for generating a round key. These dedicated qubits are used solely for generating a round key and can be sufficiently saved through optimizing. Both PLU factorization and FSE'20 eliminate the need for additional auxiliary qubits by

implementing the key-schedule circuit in-place, resulting in a reduction in the number of qubits.

In the case of PLU factorization, the circuit depth of parameters 64/128 and 128/128 increase, but the depth of 128/256 decreases. Since the overall change in the depth of the circuit is insignificant, this modulation can be ignored, and it can be considered that the number of qubits is saved while maintaining the circuit depth to some extent. However, the number of CNOT gates increased by about 40%, which can be seen as a trade-off with qubits.

FSE'20 overall reduce the number of qubits, CNOT gates, and full depth of the circuit. Comparing Tables 2 and 3, FSE'20 shows improvements in qubits by 7.9%, depth by 2.2%, and number of CNOT gates by 0.7%. When considering the maintenance of the number of #lqCliffod, #T, and T-depth, it can be observed that the FSE'20 method significantly conserves quantum resources. Comparing the two methods, PLU factorization shows an increase in the values of the rest of the indicators except for qubits, while most of the values in FSE'20 decrease, and shows improvement. Based on these results, FSE'20 can be considered as the most optimal method. Since circuit depth is related to speed and time, the reduction in circuit depth in FSE'20 suggests an improvement in the execution speed of CHAM. In addition, considering a low-power environment with limited quantum computers and resources, reducing the number of gates and qubits through FSE'20 increases the feasibility of the circuit by correspondingly reducing the demand on resources. However, FSE'20 has a limitation that it cannot be used for linear layer matrices larger than 32×32. For CHAM, the matrix for the largest instances (i.e., CHAM-128/256) is 32×32, so we could have used this method, but we cannot use this method for ciphers with matrices larger than this.

T-counts and T-depth are related to the Toffoli gate used in the quantum circuit of CHAM for the adder. The adder is not used in the key-schedule step that we optimized; therefore, all three methods have the same values for T-counts and T-depth in Table 4. Furthermore, #lqCliff also has the same value for all three methods as only the CNOT gate is utilized in the key-schedule phase.

5. Cost Estimate for Grover Key Search

The two components of Grover's algorithm are the oracle, which finds a solution, and the diffusion operator, which increases the amplitude of the solution. The diffusion operator has a very low overhead and doesn't require any special implementation methods. So, the cost for the Grover's algorithm is estimated using just the quantum resources needed for the oracle in the majority of cases [14,16,20,22,34]. The efficiency of the implemented quantum circuit determines the quantum resources of the oracle.

The oracle operates encryption and reverse operations sequentially. Since the CHAM quantum circuit is applied to each operation, the cost of using the oracle once is (Table 4 $\times$ 2) when $R = \lceil k/n \rceil$ (key size/block size) is 1 except for qubits. If R is 2, since there are 2 encryption and reverse operation pairs, there are 4 operations in total, so it is multiplied by 4. The effectiveness of the quantum circuit determines the performance of the Grover key search. Although it is well known that the Grover's algorithm needs to be iterated $\sqrt{2^n}$ times to recover the n-bit key, a detailed analysis of [35] suggests that it is optimal to repeat the Grover's algorithm $\lfloor \frac{\pi}{4} \cdot \sqrt{2^n} \rfloor$ times. Thus, the cost of the Grover key search is calculated as (Table 4$\times$ 2 $\times$ $\lfloor \frac{\pi}{4} \cdot \sqrt{2^n} \rfloor$) without the number of qubits. This means repeating Oracle by $\lfloor \frac{\pi}{4} \cdot \sqrt{2^n} \rfloor$. Table 5 shows the quantum resources required for Grover key search for CHAM with various methods applied. That is, the costs shown in Table 5 represent the resources needed for a Grover's key search attack, and this resource requirement depends on the complexity of the oracle. The costs in Table 5 are used to determine how secure a CHAM cipher is in a quantum computing environment.

NIST security is based on the post-quantum security strength specified by NIST, and the Grover attack costs of AES-128, -192, and -256 are $2^{170}, 2^{233}$, and 2^{298}, respectively. The lowest security level for encryption algorithms is 1 (corresponds to that of AES-128), and the level increases as the key size increases. According to the post-quantum security

strength criteria provided by NIST [12], 128/256 satisfies Level 3, but 64/128 and 128/128 do not exceed Level 1. NIST's post-quantum security requirements specify that the cipher should be similar to or higher than the Grover attack cost for AES to ensure security on quantum computers. Accordingly, it may be considered that 64/128 and 128/128 do not achieve an appropriate security level.

However, a document provided by NIST [12] states that the estimated costs should be conservatively evaluated if a quantum attack with a significantly reduced attack cost later appears. The AES attack cost estimated by NIST is cited in a 2016 paper by Grassl et al. [16], and implementations that further optimize AES quantum circuits have been recently proposed. A recent Grover attack cost estimation study on AES [20] shows that since Level 1 is 2^{157}, it can be seen that parameters with key size k =128 also satisfy the security strength.

Table 5. Comparison of cost of Grover's key search by methods applied to key-schedule for CHAM.

CHAM	Methods	Total Gates	Total Depth	Cost	Security [12]
64/128	PLU	1.456×2^{81}	1.025×2^{76}	1.462×2^{157}	2^{170} (Level 1)
	FSE'20	1.226×2^{81}	1.002×2^{76}	1.228×2^{157}	
	PLU (revision)	1.568×2^{81}	1.119×2^{76}	1.755×2^{157}	
	FSE'20 (revision)	1.345×2^{81}	1.094×2^{76}	1.471×2^{157}	
128/128	PLU	1.583×2^{81}	1.233×2^{77}	1.952×2^{158}	2^{170} (Level 1)
	FSE'20	1.305×2^{81}	1.005×2^{77}	1.311×2^{158}	
	PLU (revision)	1.103×2^{82}	1.716×2^{77}	1.894×2^{159}	
	FSE'20 (revision)	1.818×2^{81}	1.400×2^{77}	1.273×2^{159}	
128/256	PLU	1.895×2^{146}	1.219×2^{141}	1.155×2^{288}	2^{333} (Level 3)
	FSE'20	1.562×2^{146}	1.198×2^{141}	1.871×2^{287}	
	PLU (revision)	1.179×2^{147}	1.511×2^{141}	1.781×2^{288}	
	FSE'20 (revision)	1.946×2^{146}	1.491×2^{141}	1.450×2^{288}	

6. Conclusions

In this paper, we present optimizing quantum circuit implementations of the CHAM ciphers. We apply linear layer optimization techniques such as PLU factorization and FSE'20 to CHAM's key-schedule to eliminate additional qubits, saving the number of qubits without significantly increasing the depth of the circuit. In particular, in the case of FSE'20, not only the number of qubits but also the number of CNOT gates and the depth of the circuit are reduced. However, the PLU factorization method has limitations as it leads to an increase in both the number of CNOT gates and the depth of the circuit. While the change in depth is trivial, less than 1%, the increase in the number of CNOT gates is not negligible, ranging from 3 to 40%. In our work, we considered this as a trade-off with qubit. In the case of the FSE'20 method, we concluded that it is the most optimal method in terms of reducing the number of qubits, the number of gates, and the depth. Currently, according to Xiang et al. [33], this method has a limitation that it can only be used for linear layer matrices of size 32 or less.

The performance of future quantum computers is not yet known accurately, and for this reason, it is difficult to predict how secure the existing cryptography will be in the quantum computing environment. Accordingly, NIST proposed to measure security strength through computation resource estimation. The value provided as a security level in this standard is the cost estimated to be incurred when attacking AES through a Grover attack. Evaluating what level of security the ciphers has in the post-quantum environment through this criterion and whether it can be used is a very important task as it prevents attacks that may occur in the future. So, we estimate the quantum circuit cost required for Grover attacks and evaluate the security level. As a result, it can be considered that all methods satisfy the security strength by satisfying the cost presented in the recently proposed paper on optimized quantum circuits in AES [20].

In future studies, in addition to PLU factorization and FSE'20, we will find a way to optimize the circuit, apply it to lightweight block ciphers such as CHAM, and compare these and then try to find an optimal method. In addition, if the two techniques applied in the thesis are applied to other ciphers (e.g., SPECK, SIMON, and LEA block ciphers), a new optimization study will be able to proceed.

Author Contributions: Software, Y.Y., K.J. and A.B.; Writing—original draft, Y.Y.; Supervision, H.S. All authors have read and agreed to the published version of the manuscript.

Funding: This research was financially supported by Hansung University.

Acknowledgments: We thank Soham Roy (Indian Institute of Technology Madras, India) and Da Lin (Hubei University, Wuhan, China) for the kind help.

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

References

1. Arute, F.; Arya, K.; Babbush, R.; Bacon, D.; Bardin, J.C.; Barends, R.; Biswas, R.; Boixo, S.; Brandao, F.G.; Buell, D.A.; et al. Quantum supremacy using a programmable superconducting processor. *Nature* **2019**, *574*, 505–510. [CrossRef] [PubMed]
2. Suppressing quantum errors by scaling a surface code logical qubit. *Nature* **2023**, *614*, 676–681. [CrossRef]
3. Zhu, Q.; Cao, S.; Chen, F.; Chen, M.C.; Chen, X.; Chung, T.H.; Deng, H.; Du, Y.; Fan, D.; Gong, M.; et al. Quantum computational advantage via 60-qubit 24-cycle random circuit sampling. *Sci. Bull.* **2022**, *67*, 240–245. [CrossRef] [PubMed]
4. Madsen, L.S.; Laudenbach, F.; Askarani, M.F.; Rortais, F.; Vincent, T.; Bulmer, J.F.; Miatto, F.M.; Neuhaus, L.; Helt, L.G.; Collins, M.J.; et al. Quantum computational advantage with a programmable photonic processor. *Nature* **2022**, *606*, 75–81. [CrossRef] [PubMed]
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. Delfs, H.; Knebl, H.; Delfs, H.; Knebl, H. Symmetric-key encryption. In *Introduction to Cryptography: Principles and Applications*; Springer: Berlin/Heidelberg, Germany, 2007; pp. 11–31.
7. Shor, P.W. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Rev.* **1999**, *41*, 303–332. [CrossRef]
8. Simon, D.R. On the power of quantum computation. *SIAM J. Comput.* **1997**, *26*, 1474–1483. [CrossRef]
9. Nayak, J.; Swapnarekha, H.; Naik, B.; Dhiman, G.; Vimal, S. 25 Years of Particle Swarm Optimization: Flourishing Voyage of Two Decades. *Arch. Comput. Methods Eng.* **2022**, *30*, 1663–1725. [CrossRef]
10. Liu, G.; Chen, W.; Chen, H.; Xie, J. A quantum particle swarm optimization algorithm with teamwork evolutionary strategy. *Math. Probl. Eng.* **2019**, *2019*, 1805198. [CrossRef]
11. Bergholm, V.; Izaac, J.; Schuld, M.; Gogolin, C.; Ahmed, S.; Ajith, V.; Alam, M.S.; Alonso-Linaje, G.; AkashNarayanan, B.; Asadi, A.; et al. Pennylane: Automatic differentiation of hybrid quantum-classical computations. *arXiv* **2018**, arXiv:1811.04968.
12. NIST. Submission Requirements and Evaluation Criteria for the Post-Quantum Cryptography Standardization Process. 2016. Available online: https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/call-for-proposals-final-dec-2016.pdf (accessed on 10 April 2023).
13. Alagic, G.; Apon, D.; Cooper, D.; Dang, Q.; Dang, T.; Kelsey, J.; Lichtinger, J.; Liu, Y.-K.; Miller, C.A.; Moody, D.; et al. *Status Report on the Third Round of the NIST Post-Quantum Cryptography Standardization Process*; NIST Interagency/Internal Report (NISTIR); National Institute of Standards and Technology: Gaithersburg, MD, USA, 2022.
14. Baksi, A.; Jang, K.; Song, G.; Seo, H.; Xiang, Z. Quantum Implementation and Resource Estimates for Rectangle and Knot. *Quantum Inf. Process.* **2021**, *20*, 395. [CrossRef]
15. Jang, K.; Baksi, A.; Breier, J.; Seo, H.; Chattopadhyay, A. Quantum Implementation and Analysis of DEFAULT. Cryptology ePrint Archive, Paper 2022/647, 2022. Available online: https://eprint.iacr.org/2022/647 (accessed on 10 April 2023).
16. Grassl, M.; Langenberg, B.; Roetteler, M.; Steinwandt, R. Applying Grover's Algorithm to AES: Quantum Resource Estimates. In *Post-Quantum Cryptography, Proceedings of the PQCrypto 2016, Fukuoka, Japan, 24–26 February 2016*; Takagi, T., Ed.; Springer: Cham, Switzerland, 2016; pp. 29–43. [CrossRef]
17. 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*, 2689–1808 . [CrossRef]
18. Zhu, C.; Huang, Z. Optimizing the depth of quantum implementations of linear layers. In Proceedings of the International Conference on Information Security and Cryptology, Istanbul, Turkey, 16–17 October 2023; pp. 129–147.
19. Huang, Z.; Sun, S. Synthesizing quantum circuits of AES with lower t-depth and less qubits. In Proceedings of the Advances in Cryptology—ASIACRYPT 2022: 28th International Conference on the Theory and Application of Cryptology and Information Security, Taipei, Taiwan, 5–9 December 2022; Proceedings, Part III; Springer: Berlin/Heidelberg, Germany, 2023; pp. 614–644.
20. Jang, K.; Baksi, A.; Song, G.; Kim, H.; Seo, H.; Chattopadhyay, A. Quantum Analysis of AES. Cryptology ePrint Archive, Paper 2022/683, 2022. Available online: https://eprint.iacr.org/2022/683 (accessed on 10 April 2023).

21. Hatzivasilis, G.; Fysarakis, K.; Papaefstathiou, I.; Manifavas, C. A review of lightweight block ciphers. *J. Cryptogr. Eng.* **2018**, *8*, 141–184. [CrossRef]
22. Jaques, S.; Naehrig, M.; Roetteler, M.; Virdia, F. Implementing Grover Oracles for Quantum Key Search on AES and LowMC. In *Lecture Notes in Computer Science, Proceedings of the Advances in Cryptology—EUROCRYPT 2020—39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, 10–14 May 2020*; Canteaut, A., Ishai, Y., Eds.; Springer: Berlin/Heidelberg, Germany, 2020; Volume 12106, pp. 280–310. [CrossRef]
23. Van Hoof, I. Space-efficient quantum multiplication of polynomials for binary finite fields with sub-quadratic Toffoli gate count. *arXiv* **2019**, arXiv:1910.02849.
24. Jang, K.; Song, G.; Kim, H.; Kwon, H.; Kim, H.; Seo, H. Parallel quantum addition for Korean block ciphers. *Quantum Inf. Process.* **2022**, *21*, 1–25. [CrossRef]
25. 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 Proceedings of the International Conference on Information Security and Cryptology, Seoul, Republic of Korea, 29 November–1 December 2017; pp. 3–25.
26. Roh, D.; Koo, B.; Jung, Y.; Jeong, I.W.; Lee, D.G.; Kwon, D.; Kim, W.H. Revised version of block cipher CHAM. In Proceedings of the Information Security and Cryptology—ICISC 2019: 22nd International Conference, Seoul, Republic of Korea, 4–6 December 2019; Springer: Berlin/Heidelberg, Germany, 2020; pp. 1–19.
27. Cuccaro, S.A.; Draper, T.G.; Kutin, S.A.; Moulton, D.P. A new quantum ripple-carry addition circuit. *arXiv* **2004**. [CrossRef]
28. Jang, K.; Choi, S.; Kwon, H.; Kim, H.; Park, J.; Seo, H. Grover on Korean block ciphers. *Appl. Sci.* **2020**, *10*, 6407. [CrossRef]
29. Jones, J. Nuclear magnetic resonance quantum computation. In *Les Houches*; Elsevier: Amsterdam, The Netherlands, 2004; Volume 79, pp. 357–400.
30. Amy, M.; Maslov, D.; Mosca, M.; Roetteler, M.; Roetteler, M. A Meet-in-the-Middle Algorithm for Fast Synthesis of Depth-Optimal Quantum Circuits. *IEEE Trans. -Comput.-Aided Des. Integr. Circuits Syst.* **2013**, *32*, 818–830. [CrossRef]
31. Zou, J.; Wei, Z.; Sun, S.; Liu, X.; Wu, W. Quantum Circuit Implementations of AES with Fewer Qubits. In Proceedings of the Advances in Cryptology—ASIACRYPT 2020, Online Event, 7–11 December 2020; Moriai, S., Wang, H., Eds.; Springer: Cham, Switzerland, 2020; pp. 697–726. [CrossRef]
32. Banegas, G.; Bernstein, D.J.; van Hoof, I.; Lange, T. Concrete Quantum Cryptanalysis of Binary Elliptic Curves. Cryptology ePrint Archive, Paper2020/1296, 2020. Available online: https://eprint.iacr.org/2020/1296 (accessed on 1 April 2023).
33. Xiang, Z.; Zeng, X.; Lin, D.; Bao, Z.; Zhang, S. Optimizing implementations of linear layers. *IACR Trans. Symmetric Cryptol.* **2020**, *2022*, 120–145. [CrossRef]
34. Bijwe, S.; Chauhan, A.K.; Sanadhya, S.K. Quantum Search for Lightweight Block Ciphers: GIFT, SKINNY, SATURNIN. Cryptology ePrint Archive, Paper 2020/1485, 2020. Available online: https://eprint.iacr.org/2020/1485 (accessed on 10 April 2023).
35. Boyer, M.; Brassard, G.; Høyer, P.; Tapp, A. Tight Bounds on Quantum Searching. *Fortschritte Phys.* **1998**, *46*, 493–505. [CrossRef]