applied sciences

MDPI

Article

A Parallel Quantum Circuit Implementations of LSH Hash Function for Use with Grover's Algorithm

Gyeongju Song, Kyungbae Jang, Hyunji Kim and Hwajeong Seo *

Division of IT Convergence Engineering, Hansung University, Seoul 02876, Korea
* Correspondence: hwajeong@hansung.ac.kr; Tel.: +82-760-8033

Abstract: Grover's search algorithm accelerates the key search on the symmetric key cipher and the pre-image attack on the hash function. To conduct Grover's search algorithm, the target cipher algorithm should be efficiently implemented in a quantum circuit. Currently, small quantum computers are difficult to operate with large quantum circuits due to limited performance. Therefore, if a large quantum computer that can operate Grover's algorithm appears, it is expected that a cipher attack will be possible. In this paper, we propose a parallel structure quantum circuit for the Korean hash function standard (i.e., LSH). The proposed quantum circuit designed a parallel operation structure for the message expansion (i.e., MsgExp) function and the mix function, which are the internal structures of the LSH hash function. This approach shows an efficient result for quantum circuit implementation in terms of quantum resources by reducing the depth of the quantum circuit by about 96% through the trade-off of appropriate quantum resources compared to previous work. This result can be a reference for the implementation of a parallel quantum circuit in the future and is expected to advance the attack timing of the search algorithm for Grover's LSH hash function.

Keywords: quantum computing; LSH hash function; Grover's algorithm; quantum circuit

Citation: Song, G.; Jang, K.; Kim, H.; Seo, H. A Parallel Quantum Circuit Implementations of LSH Hash Function for Use with Grover's Algorithm. *Appl. Sci.* **2022**, *12*, 10891. https://doi.org/10.3390/app122110891

Academic Editors: Konstantinos Rantos, Konstantinos Demertzis and George Drosatos

Received: 19 September 2022
Accepted: 24 October 2022
Published: 27 October 2022

1. Introduction

Quantum computers can solve specific problems much faster than classic computers. Shor's algorithm [1] and Grover's algorithm [2], known as quantum algorithms, can threaten the current cryptosystem. Shor's algorithm poses a threat to the security of Rivest–Shamir–Adleman (RSA) and Elliptic Curve Cryptography (ECC), which are public-key cryptography. The integer factorization and discrete logarithm problems used in the safety of RSA and ECC are difficult problems on classical computers. However, using Shor's algorithm, this problem can be solved in polynomial time. Grover's algorithm accelerates the speed of finding specific data in unsorted data. Thus, Grover's algorithm poses a threat by accelerating brute-force attacks to find the key to symmetric cryptography. If $O(n)$ queries have to be performed for a brute-force attack in a classic computer, a quantum computer can be performed in $O(\sqrt{n})$ queries. To prevent such quantum algorithm attacks, the Institute of Standards and Technology (NIST) is working on standardization of post-quantum cryptography. Recently, research has been conducted actively to optimize symmetric key cryptography [3–17] and hash functions [18,19] as quantum circuits.

Current quantum computers are difficult to operate due to performance limitations such as the number of available qubits and errors. The depth of a quantum circuit is connected to the time step (i.e., time complexity) required for quantum operations executed in quantum hardware [20–25]. Therefore, much research has proposed the implementation of a quantum circuit that reduces the number and error (i.e., depth) of qubits in the implementation of the quantum circuit.

With this research motivation, we propose a parallel-structured quantum circuit for the LSH [26] hash function, a Korean national standard hash function designed in Korea. We implemented a parallel structure using the parallel adder [27] for the message

expansion function(MsgExp) and the mix function, which are internal functions of the LSH hash function. The proposed parallel quantum circuit is designed to reduce circuit depth through efficient trade-offs between quantum resources. We improved the sequential LSH quantum circuit in previous work [19], greatly reducing the depth of the quantum circuit. The sequential LSH quantum circuit of the previous work showed about 210,000 depth based on LSH-256-n (i.e., n = 224 or 256) and about 420,000 depth based on LSH-512-m (i.e., m = 224, 256, 384, or 512). On the other hand, the depth of the proposed parallel quantum circuit is 6879 based on LSH-256-n. The proposed work reduced the depth by about 96.73% compared to the previous work. As a result of resource estimation, the CNOT and X gates increased, but the Toffoli gates, which are more expensive than the CNOT and X gates, decreased. Therefore, we argue that this is an efficient trade-off. The structure of this paper is as follows: Section 2, background knowledge about LSH hash function, quantum computing, and Grover's algorithm is explained. Section 3 describes the proposed LSH quantum circuit, and Section 4 evaluates the quantum resources estimation result.

2. Background

2.1. LSH Hash Function

LSH is Korean national standard (KSM X 3262) hash function designed in Korea and approved by the Korean Cryptographic Module Verification Program (KCMVP). LSH operates in units of words (w = 32, 64) and has an n-bit output value. For this, we denote it as LSH-8w-n.(family: LSH-256-224, LSH-256-256, LSH-512-224, LSH-512-256, LSH-512-384, and LSH-512-512). The LSH hash function consists of initialization, compression, and final, as shown in Figure 1.

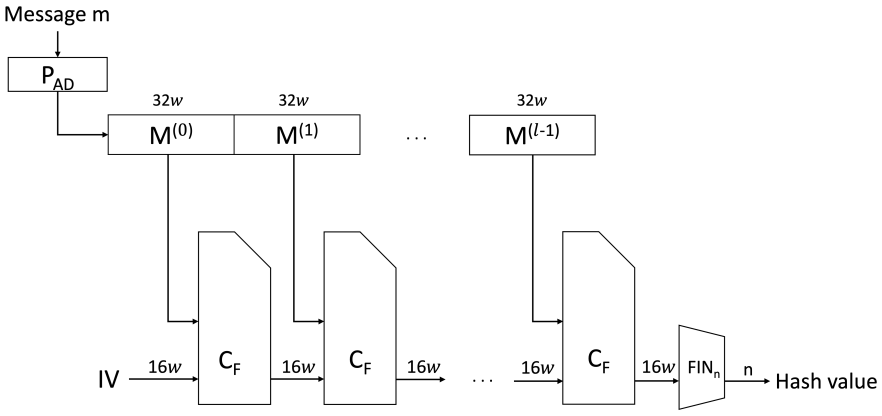

Figure 1. Overview of LSH hash function.

In the initialization, the input message is padded in word (w) units and divided into word-sized message blocks. For message m, "1" is appended to the end of m and padded with "0" to make it $32wt$-bits in length ($t = \lceil (|m| + 1)/32w \rceil$). The padded message $m_p = m_0||m_1||\cdots||m_{32wt-1}$ can be represented as a $4wt$ byte array $m_a = (m[0], \cdots, m[4wt-1])$. The $4wt$ byte array m_a is converted to $M = (M[0], \cdots, M[32t-1])$ of the $32t$ word array through Equation (1)

$$M[s] \leftarrow m[ws/8 + (w/8 - 1)]|| \cdots ||m[ws/8 + 1]||m[ws/8], \quad (0 \le s \le (32t - 1)) \tag{1}$$

The word array converted through Equation (1) is divided into t message blocks $M^{(0)}, M^{(0)}, \cdots m^{t-1}$ according to Equation (2).

$$M^{(i)} \leftarrow (M[32i], M[32i + 1], \cdots, M[32i + 31]), \quad (0 \le i \le (t - 1)) \tag{2}$$

Concatenated variable (CV) is initialized with an initialization vector (IV). The initialization vectors (IV) for LSH-256-224 and LSH-512-224 are shown in Tables 1 and 2. The data

format is hexadecimal. LSH-256 uses 512-bit IV and LSH-512 uses 1024-bit IV to initialize CV, respectively.

Table 1. Initialization vector (IV) for LSH-256-224 in hexadecimal.

IV[0]	IV[1]	IV[2]	IV[3]
068608D3	62D8F7A7	D76652AB	4C600A43
IV[4]	IV[5]	IV[6]	IV[7]
BDC40AA8	1ECA0B68	DA1A89BE	3147D354
IV[8]	IV[9]	IV[10]	IV[11]
707EB4F9	F65B3862	6B0B2ABE	56B8EC0A
IV[12]	IV[13]	IV[14]	IV[15]
CF237286	EE0D1727	33636595	8BB8D05F

Table 2. Initialization vector (IV) for LSH-512-224 in hexadecimal.

IV[0]	IV[1]	IV[2]	IV[3]
0C401E9FE8813A55	4A5F446268FD3D35	FF13E452334F612A	F8227661037E354A
IV[4]	IV[5]	IV[6]	IV[7]
A5F223723C9CA29D	95D965A11AED3979	01E23835B9AB02CC	52D49CBAD5B30616
IV[8]	IV[9]	IV[10]	IV[11]
9E5C2027773F4ED3	66A5C8801925B701	22BBC85B4C6779D9	C13171A42C559C23
IV[12]	IV[13]	IV[14]	IV[15]
31E2B67D25BE3813	D522C4DEED8E4D83	A79F5509B43FBAFE	E00D2CD88B4B6C6A

In the compression function (CF), the connection variable (CV) is updated using the expanded message and the initial connection variable (CV). The t message blocks generated in the initialization step are used sequentially as input to the CF: $W^{16} \times W^{32} \rightarrow W^{16}$. The compression function proceeds in four steps. (1) Message expansion (MsgExp): $W^{32} \rightarrow W^{16(N_s+1)}$, (2) Message addition (MsgAdd): $W^{16} \times W^{16} \rightarrow W^{16}$, (3) Message mix (Mix): $W^{32} \rightarrow W^{16}$, (4) Word permutation (WordPerm): $W^{16} \rightarrow W^{16}$. Figure 2 shows the process of the compression function.

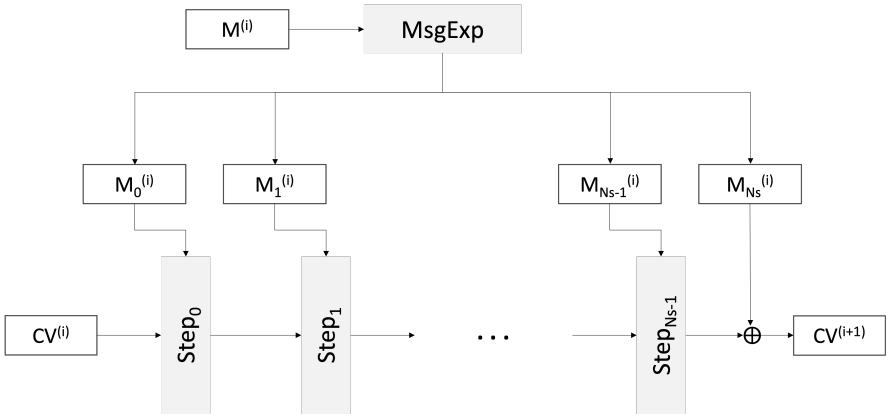

Figure 2. Compression Function (CF) of LSH.

First, the message block $M^{(i)}$ entered in the compression function is expanded via MsgExp into $(N_s + 1)$ 16-word arrays $M_j^{(i)} (0 \leq j \leq N_s)$. If w = 32 then N_s = 26, if w = 64 then N_s = 28. Set the initial value of $T (T = T[0], \ldots, T[15])$ with $CV^{(i)}$ and T is updated

using the message in the *Step* function. The step function $step : W^{16} \times W^{16} \rightarrow W^{16}$ in the compression function that processes the message works as follows:

$$Step_j := WordPerm \circ Mix_j \circ MsgAdd, \quad (0 \leq j \leq (N_s - 1))$$

The message expansion function (MsgExp) generates $(N_s + 1)$ word arrays $M_j^{(i)}$ $(0 \leq j \leq N_s)$ from the i-th message block $M^{(i)} = (M^{(i)}[0], \cdots, M^{(i)}[31])$, which is the input of the compression function. The message generation method is as in Equation (4). In the MsgAdd function, an XOR operation is performed on the same index of two 16-word arrays X and Y: $\text{MsgAdd}(X, Y) := (X[0] \oplus Y[0], \cdots, X[15] \oplus Y[15])$. The mix function updates T with two word pairs $T[l], T[l + 8] (0 \leq l \leq 7)$ for $T = (T[0], \cdots, T[15])$. The operation of the Mix function is as follows:

$$X \leftarrow X \boxplus Y; \quad X \leftarrow X^{\lll \alpha_j}; \quad X \leftarrow X \oplus SC_j[l];$$

$$Y \leftarrow X \boxplus Y; \quad Y \leftarrow Y^{\lll \beta_j}; \quad X \leftarrow X \boxplus Y; \quad Y \leftarrow Y^{\lll \gamma_l}$$

The bit rotation amount used in the mix function is shown in Table 3. The bit rotation amount varies depending on the word(w), even/odd steps.

Table 3. Bit rotation amount in $Mix_{j,l}$.

w	j	α	β_j	γ_0	γ_1	γ_2	γ_3	γ_4	γ_5	γ_6	γ_7
32	Even	29	1	0	8	16	24	24	16	8	0
	Odd	5	17								
64	Even	23	59	0	16	32	48	8	24	40	56
	Odd	7	3								

The final function ($FIN_n : W_{16} \rightarrow 0, 1_n$) generates the final hash value using $CV^{(t)} = CV^{(t)}[0], \cdots, CV^{(t)}[15]$ updated from compression. For the 8-word array $H = H[0], \cdots, H[7]$ and the w-byte array $h_b = h_b[0], \cdots, h_b[w\text{-}1]$, the completion function FIN_n performs Equation (3) to output the final hash value.

$$\begin{aligned} &H[i] = CV^{(t)}[i] \oplus CV^{(t)}[i + 8], \ (0 \leq i \leq 7), \\ &h_b[s] = H[(8s/w)^{\ggg (8s \bmod w)}][7:0], \ (0 \leq s \leq (w - 1)), \\ &h = (h_b[0] \,||\, \ldots \,||\, h_b[w - 1])_{[0:n-1]} \end{aligned} \tag{3}$$

2.2. Quantum Computing

Quantum computers use the quantum mechanical properties of qubits: superposition and entanglement to perform computations. A classic computer uses bits that have one of the states 0 and 1, but a quantum computer uses qubits that can have both 0 and 1 at the same time. Since qubits have both 0 and 1 probabilistically, 2^n values can be expressed with n-qubits and calculated at once.

Figure 3 shows the X, CNOT, Toffoli, and SWAP gate among the quantum gates that control qubits.

The X gate performs like the NOT gate of a digital logic gate. A single qubit is used as an input and the state of the input qubit is inversed. The CNOT gate has an entangled state in which one qubit affects another. The two input qubits are divided into control qubit and target qubit, respectively. When the state of one control qubit is one, the state of the target qubit is inversed. In the Tofffoli gate, the state of two qubits affects the state of one qubit. That is, three qubits operate in an entangled state. The three input qubits are divided into two control qubits and one target qubit. When the state of both control qubits is one, the state of the target qubit is inversed. The SWAP gate changes the position of two qubits. Therefore, it does not use quantum resource cost. Current small-scale quantum

computers have small usable quantum resources and small operable quantum circuits, so it is important to implement efficient quantum circuits.

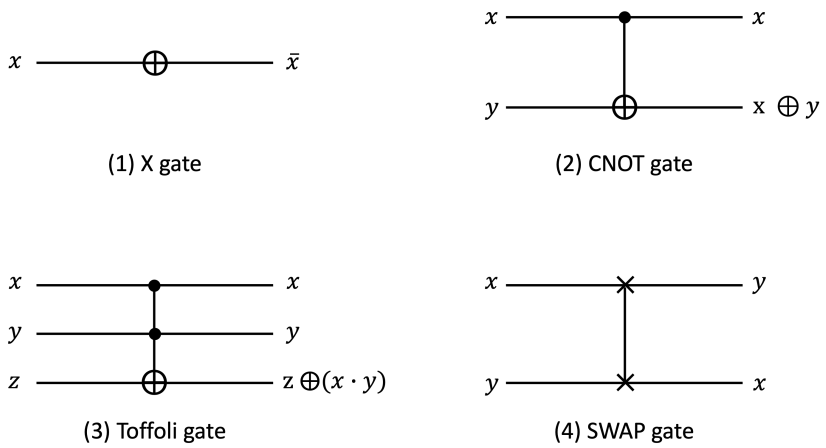

Figure 3. Quantum gates: X gate, CNOT gate, Toffoli gate, and SWAP gate.

2.3. Grover's Algorithm

Grover's algorithm [2] is a quantum algorithm that searches a space with n elements to find the input data that generate the output of a particular function. Searching n unsorted databases on a classic computer would require n searches. In quantum computers, Grover's search algorithm can find specific data through $\sqrt{n}$ searches. So the time complexity is reduced from $O(n)$ to $O(\sqrt{n})$. As a result, Grover's algorithm threatens symmetric key cryptography because it reduces the time required for brute-force attacks.

Grover's algorithm works with the Oracle and Diffusion operator, and the order is as follows. First, Hadamard gates are all applied to the qubits of the data we want to find. Second, the Oracle function $f(x)$ returns 1 when x is the answer, and it revers the phase of qubits representing the answer. Third, the diffusion operator increases the probability of an answer by amplifying the amplitude of the correct answer qubits reversed through the oracle. Through repetition of the oracle and diffusion process, the probability of the answer exceeds the threshold, and as a result, x that exceeds the threshold becomes the correct answer. Figure 4 shows the overall structure of Grover's algorithm when $x = 11$ is the correct answer. The two input qubits have a superposition state through the Hadamard gate, and the state of the correct answer qubit is reversed in Oracle. The state of the reversed qubits in the oracle is amplified by the Diffusion operator.

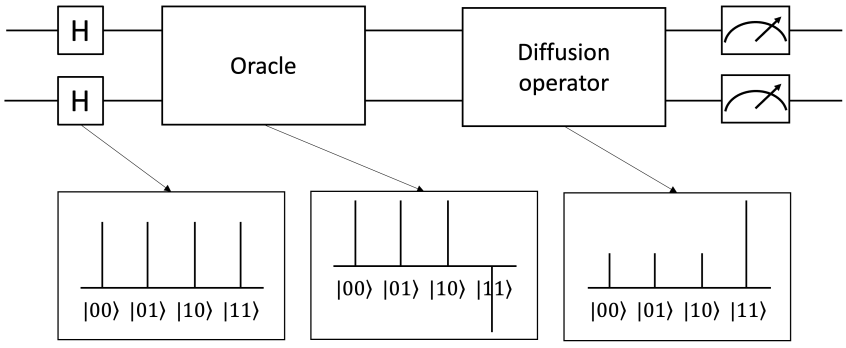

Figure 4. Grover's algorithm (answer $x = 11$).

The brute-force attack on block ciphers with Grover's algorithm is as follows. This is a known-plaintext attack (KPA) that can be performed when the plaintext-ciphertext pair of the block cipher is known. The n-bit key used in the cipher is targeted to a brute-force

attack. The operation of Grover's algorithm requires a quantum circuit for the target cipher, and the quantum circuit operates inside Oracle. The quantum circuit in Oracle performs encryption with plaintext and n-bit key as inputs. The plaintext is set to a known plaintext using the X gate, and the Hadamard gate is applied to the n-bit key to making a superposition state. Oracle uses superposition keys for encryption so it can have the ciphertext for all keys in a single query. At the end of Oracle, set a known ciphertext to find the key state that generates the same ciphertext as the known ciphertext. In Oracle, the sign of the correct key is inverted, and the diffusion operator increases the probability of measuring the correct key. It is known that the correct key can be found in about $\lfloor \frac{\pi}{4}\sqrt{2^n} \rfloor$ iterations of the Grover algorithm.

A pre-image attack on a hash function using Grover's algorithm is similar to the case of the block cipher. The hash function performs a Hadamard gate on the message, followed by Oracle and the Diffusion operator. In Oracle, since the hash function is operated using the superposition message, it is possible to have a hash value for all plaintexts with a single query. At the end of Oracle, a known hash value is set to find a message that outputs a known hash value. In Oracle, the sign of the correct message is inverted, and the diffusion operator increases the probability of measuring the message. It is known that the correct message can be found in about $\lfloor \frac{\pi}{4}\sqrt{2^n} \rfloor$ iterations of the Grover algorithm.

3. Proposed Method

3.1. LSH Quantum Circuit

This section describes the parallel quantum circuit for LSH. In this paper, the parallel operation is designed for the independently operable part of the LSH hash function. This is designed to reduce the quantum circuit depth of LSH through an efficient quantum resource trade-off. We implemented the message expansion(MsgExp) function and the message mix(Mix) function in parallel. As a result, the parallel quantum circuit shows about a 96% reduction in the depth of the quantum circuit compared to the previous work [19]. In message expansion(MsgExp) and message mix(Mix), which are internal functions of the LSH hash function, each message is independently calculated in units of words. Therefore, since they do not affect each other's results, it shows that the depth of the circuit can be greatly reduced by processing the operation of each message word in parallel. That is, both functions can significantly reduce the depth of the quantum circuit by processing the operation of each message word in parallel. We design the parallel operation in the LSH using the parallel adder proposed by [27]. In a previous work, Song et al. [19] used a sequential adder [27] in the LSH quantum circuit. The quantum adder is performed by reusing 1-ancilla qubits. However, since the ancilla qubits used in the quantum adder are reused, sequential operations must be performed even if parallel operations are possible. The sequential adder uses $(2n-2)$ Toffoli gates, $4n$ CNOT gates, and $(6n-2)$ depth; (n: bit length). Figure 5 shows the sequential addition operation in MsgExp. In this adder, the message block pairs M_j, M_{j-1} are calculated sequentially. Since 16 additions are performed one by one, the quantum circuit has a depth of 16 $\times$ ($6n-2$). The sequential adder of the LSH quantum circuit is inefficient in terms of quantum circuit depth because it greatly increases the depth. As a result, the sequential quantum circuit of the previous work was implemented at a depth of hundreds of thousands (#LSH-256-n : about 210,050; #LSH-512-m : about 421,850).

We propose a method to utilize the adder in [27] as an efficient parallel quantum adder in LSH. We design a parallel addition structure that uses an optimal quantum adder for the LSH quantum circuit and has an efficient trade-off between quantum resources. The parallel quantum adder uses $(2n-3)$ Toffoli gates, $(5n-7)$ CNOT gates, and $(2n-6)$ X gates, and has a depth of $(2n+3)$. Figure 6 shows the parallel addition operation in MsgExp. In this adder, the message block pairs M_j, M_{j-1} are calculated in parallel. Since 16 additions are performed at once, the depth of the quantum circuit is only ($2n+3$). The parallel adder increases the number of CNOT and X gates. However, it reduces the number of Toffoli gates, which is a more expensive resource than CNOT, X gates, and significantly reduces the depth

of the quantum circuit. Consequently, we saw this as a very efficient trade-off. We describe quantum circuits based on LSH-256-n. In fact, we show the result of reducing the total depth by about 96% compared to the previous work by implementing the parallel structure. The trade-off results for quantum resources are described in detail in Section 4.

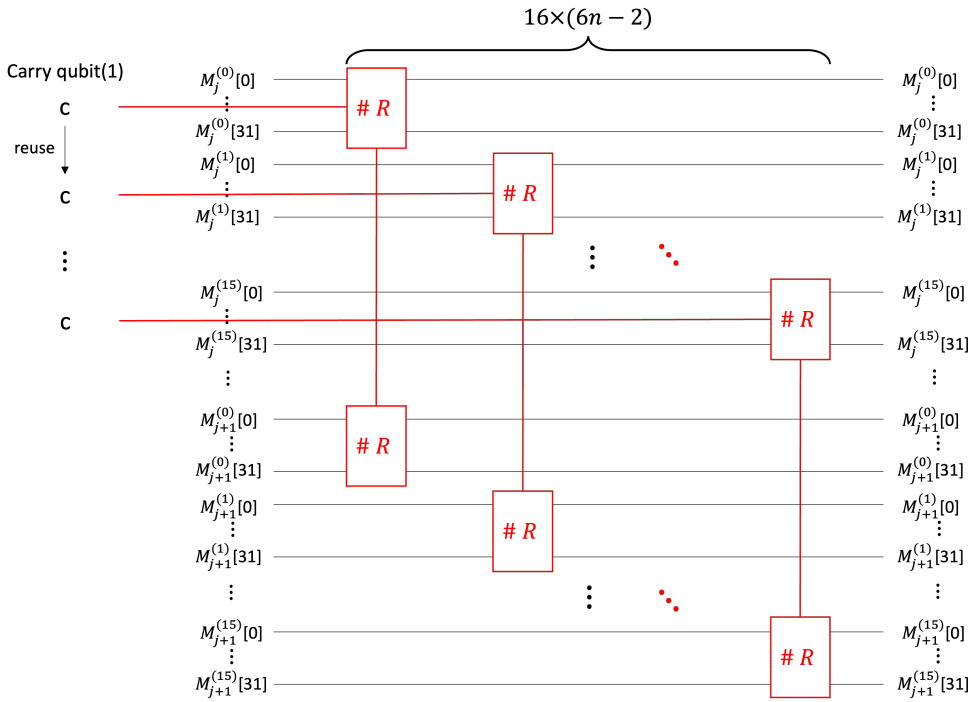

Figure 5. Sequential addition in MsgExp function. (Depth: 16 × (6n−2)).

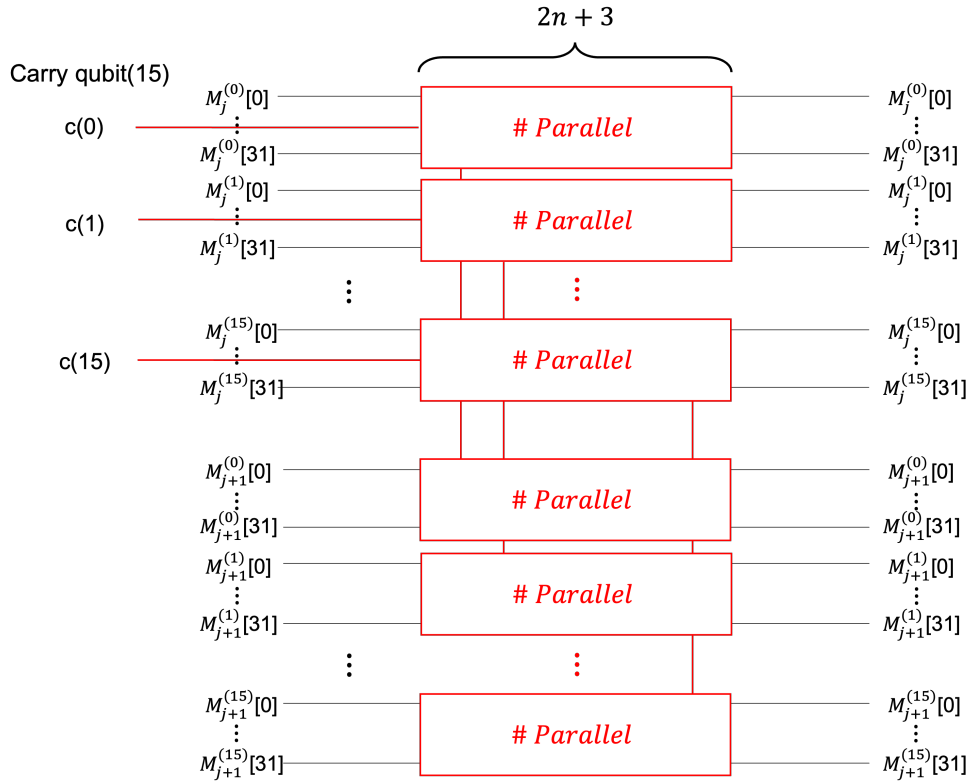

Figure 6. Parallel addition in MsgExp function. (Depth: only 2n+3).

3.2. Parallel Quantum Circuit for LSH

In LSH, the addition is used for message expansion (MsgExp) and message mix (Mix), and it has a characteristic that can be processed as a parallel adder. Each addition operation unit in MsgExp and the mix does not affect the results of each other. Due to this characteristic, the depth of the circuit can be significantly reduced using the parallel adder. In LSH-256-n, 32 bits are processed in units of 1 word. In the LSH-256-n quantum circuit, 1024 qubits are used in the padded plaintext M, 512 qubits for the connection variable (CV), and 16 carry qubits are used in the parallel adder. In LSH-512-m, 64 bits are processed in units of 1 word. In the LSH-512-m quantum circuit, 2048 qubits for the padded plaintext M, 1024 qubits for the connection variable (CV), and 15 carry qubits are used in the parallel adder. The overall operation process of LSH-256-n and LSH-512-m is the same, but the output of each operation bit unit, step constant, and the final hash value is different. Source codes of the proposed parallel structure LSH quantum circuit are available in https://github.com/kyungzzu/Grover-on-SM3-and-LSH (accessed on 18 September 2022).

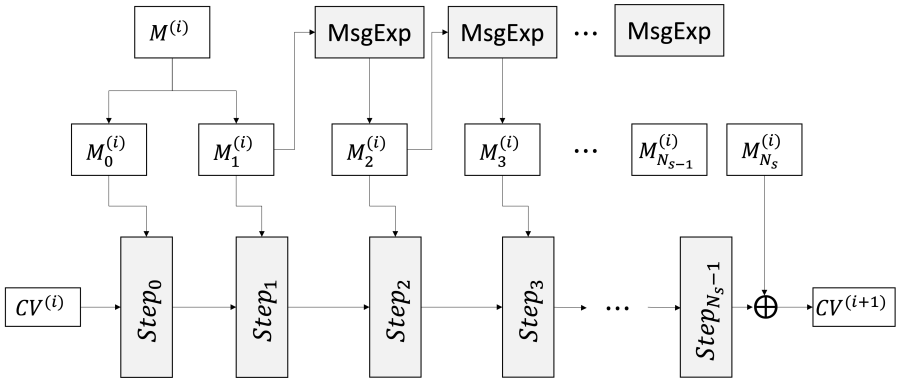

Figure 7. LSH quantum circuit for MsgExp and *step* function.

Figure 2 shows the progress of the MsgExp function and *step* function of the original LSH hash function. That is, after expanding all messages through the MsgExp function, the expanded message is used in the *step* function. This method is very inefficient in terms of quantum resources because it requires qubits to store the entire expanded message. Therefore, in the quantum circuit, the MsgExp function and *step* functions are iteratively performed, as shown in Figure 7 to reduce the temporary qubits used for message expansion. For example, by the message expansion equation $M_j^{(i)}[l] \leftarrow M_{j-1}^{(i)}[l] \boxplus M_{j-2}^{(i)}[\tau(l)]$ in Equation (4), the third message block M_2 is expanded by the addition operation of M_1 and M_0. $\tau(l)$ is the value substituted by the permutation in Table 4. If the MsgExp and *step* functions are performed in units of one message block when the M_2 message block is used, M_0 and M_1 have already been used, so the result of the expansion of M_2 can be calculated in M_0. In LSH-256-n, a 1024 bit message is divided into $M_{k-1}^i, M_{k-2}^i (0 \leq i \leq 15)$ of 1 word(1 word = 32 bit) each to perform the *step* function. In LSH-512-m, a 2048-bit message is divided into $M_{k-1}^i, M_{k-2}^i (0 \leq i \leq 15)$ of 1 word(1 word = 64 bits) each to perform *step* function.

In summary, the proposed technique does not allocate qubits to store the updated M. Instead, it saves qubits by generating new values for M used in the previous round. The connection variable $T[i], T[i+8], (0 \leq i \leq 7)$ updates the value by performing the MsgExp, Mix, and WordPerm functions, and finally obtains a hash value through the Final function with the updated value. MsgExp generates 16 word message block $M_j^{(i)} (0 \leq j \leq N_s)$ for message block $M^{(i)} = M^{(i)}[0], \ldots, M^{(i)}[31]$ (32 bits) by using Equation (4). The adder used to generate the next message is performed after bit permutation, where the bit permutation $\tau(l)$ is shown in Table 4.

$$\begin{aligned} M_0^{(i)} &\leftarrow M^{(i)}[0], M^{(i)}[1], \cdots, M^{(i)}[15] , \\ M_1^{(i)} &\leftarrow M^{(i)}[16], M^{(i)}[17], \cdots, M^{(i)}[31] , \\ M_j^{(i)}[l] &\leftarrow M_{j-1}^{(i)}[l] \boxplus M_{j-2}^{(i)}[\tau(l)] , \ (0 \leq l \leq 15) \end{aligned} \tag{4}$$

Table 4. Permutation table for the LSH expansion function.

l	0	1	2	3	4	5	6	7
$\tau(l)$	3	2	0	1	7	4	5	6
l	8	9	10	11	12	13	14	15
$\tau(l)$	11	10	8	9	15	12	13	14

In MsgExp, the addition operations of message block pairs (i.e., $M_{j-1}, M_{j-2},\ 2 \leq j \leq N_s$) are all independent, so the adders can be designed in parallel. Algorithm 1 shows the computation for parallel addition in MsgExp. This adder uses 16 ancilla qubits c to store carry values per message pair. Since the adder uses ancilla qubits c_j individually, it can perform parallel additions on pairs of input messages. As a result, the Algorithm 1 is run concurrently for the number of message pairs.

Algorithm 1 Parallel quantum adder of LSH.

Input: M_k and M_{k-1} pair, ancilla c_k ($1 \leq k \leq 16$)

for i = 0 to 29 **do**
  $M_{k-1}[i+1] \leftarrow$ CNOT($M_k[i+1]$, $M_{k-1}[i+1]$)
end for
$c_k \leftarrow$ CNOT($M_k[1]$, c_k)
$c_k \leftarrow$ Toffoli($M_k[0]$, $M_{k-1}[0]$, c_k)
$M_k[1] \leftarrow$ CNOT($M_k[2]$, $M_k[1]$)
$M_k[1] \leftarrow$ Toffoli(c_k, $M_{k-1}[1]$, $M_k[1]$)
$M_k[2] \leftarrow$ CNOT($M_k[3]$, $M_k[2]$)
for i = 0 to 26 **do**
  $M_k[i+2] \leftarrow$ Toffoli($M_k[i+1]$, $M_{k-1}[i+2]$, $M_k[i+2]$)
  $M_k[i+3] \leftarrow$ CNOT($M_k[i+4]$, $M_k[i+3]$)
end for
$M_k[29] \leftarrow$ Toffoli($M_k[28]$, $M_{k-1}[29]$, $M_k[29]$)
$M_{k-1}[31] \leftarrow$ CNOT($M_k[30]$, $M_{k-1}[31]$)
$M_{k-1}[31] \leftarrow$ CNOT($M_k[31]$, $M_{k-1}[31]$)
$M_{k-1}[31] \leftarrow$ Toffoli($M_k[29]$, $M_{k-1}[30]$, $M_{k-1}[31]$)
for i = 0 to 28 **do**
  X($M_{k-1}[i+1]$)
end for
$M_{k-1}[1] \leftarrow$ CNOT(c_k, $M_{k-1}[1]$)
for i=0 to 28 **do**
  $M_{k-1}[i+2] \leftarrow$ CNOT($M_k[i+1]$, $M_{k-1}[i+2]$)
end for
$M_k[29] \leftarrow$ Toffoli($M_k[28]$, $M_{k-1}[29]$, $M_k[29]$)

Algorithm 1 *Cont.*

for i=0 to 26 **do**
  M_k[28-i] ← Toffoli(M_k[27-i], M_{k-1}[28-i], M_k[28-i])
  M_k[29-i] ← CNOT(M_k[30-i], M_k[29-i])
  X(M_{k-1}[29-i])
end for
M_k[1] ← Toffoli(c_k, M_{k-1}[1], M_k[1])
M_k[2] ← CNOT(M_k[3], M_k[2])
X(M_{k-1}[2])
c_k ← Toffoli(M_k[0], M_{k-1}[0], c_k)
M_k[1] ← CNOT(M_k[2], M_k[1])
X(M_{k-1}[1]
c_k ← CNOT(M_k[1], c_k)
for i = 0 to 30 **do**
  M_{k-1}[i] ← CNOT(M_k[i], M_{k-1}[i])
end for

In Mix, adders operate in parallel for $T[i]$ and $T[i+8]$ $(0 \leq i \leq 7)$ pairs, respectively. The result of the addition operation is stored in $T[i]$. Since addition operations of $T[i]$ and $T[i+8]$ do not affect each other, the parallel operation is possible. The adder used in Mix is the same as Algorithm 1, and message block pairs (i.e., M_{j-1}, M_{j-2}, $2 \leq j \leq N_s$) are changed to $T[i]$, $T[i+8]$ $(0 \leq i \leq 7)$ pairs at the input. Algorithm 2 shows the quantum circuit implementation of the Mix function. One Mix function is performed with two word pairs $T[i], T[i+8], (0 \leq i \leq 7)$ and a total of eight Mix functions are operated per round. In the Mix function quantum circuit, the a_rotation, b_rotation, and c_rotation functions of lines 2, line 5, and line 7 perform index rotation. The rotation value is determined according to the number of words (32-bit or 64-bit) and the j value of the step function $Step_j$. Since only the swap gate is used in the rotation operation, additional quantum resources are not used.

Algorithm 2 Quantum circuit of the Mix function.

Input: $T[i]$, $T[i+8]$, $SC[i]$, $(0 \leq i \leq 7)$
1: $T[i+8]$ ← **Parallel_adder**($T[i]$, $T[i+8]$)
2: **a_rotation**($T[i]$)
3:
4: **Applying X gate to** $T[i]$ **according to** $SC[i]$
5: $T[i+8]$ ← **Parallel_adder**($T[i]$, $T[i+8]$)
6: **b_rotation**($T[i+8]$)
7: $T[i]$ ← **Parallel_adder**($T[i+8]$, $T[i]$)
8: **c_rotation**($T[i+8]$)

4. Evaluation

The proposed LSH quantum circuit was evaluated using a quantum emulator (i.e., IBM ProjectQ). Among various compilers provided by IBM, the ProjectQ quantum compiler can estimate the resources of implemented quantum circuits. It measures the number of Toffoli gates, CNOT gates, X gates, and qubits used in a quantum circuit. One of the important elements of a quantum circuit is making it work with minimal resources and depth. Currently, the number of qubits available in quantum computer technology is limited, and it is efficient to reduce the quantum resource cost. The depth of a quantum circuit is related to the time complexity required for quantum operations performed on quantum hardware [20–25]. Therefore, many studies are being conducted to reduce the

depth associated with errors on a noisy quantum computer. With this research motivation, we have worked to reduce the number of quantum gates and qubits for the implementation of quantum circuits. Further, we devised a reduction of the depth of the quantum circuit with an efficient trade-off between quantum resources. We designed a parallel LSH structure using an optimal quantum adder for the LSH quantum circuit. As a result, the parallel quantum circuit significantly reduces depth with an efficient trade-off in terms of quantum resources. The adder in the previous work [19] performed addition sequentially to reduce the ancilla qubits used. However, since LSH can be operated in parallel, such sequential operation is inefficient because it greatly increases the depth of the LSH quantum circuit. Table 5 shows the quantum resources of our previous work [19], and Table 6 shows the quantum resources of the parallel LSH proposed in this paper. Compared with the previous work, the proposed circuit uses an additional 15-qubit and uses more CNOT gates and X gates, but uses fewer Toffoli gates. However, the Toffoli gate is a more expensive resource than the X gate and CNOT gate. Due to this trade-off of quantum resources, the circuit depth is greatly reduced. As a result, we argue for an efficient quantum circuit implementation with a slight quantum resource trade-off. The depth of the proposed LSH quantum circuit was reduced by about 96% compared to previous work.

Table 7 shows the resources required to perform a Grover attack on LSH using the proposed quantum circuit. To get the correct result for the attack, the Grover's algorithm $\lfloor \frac{\pi}{4}\sqrt{2^n} \rfloor$ times. LSH-256/512-n with n-bit hash length repeats the Grover's algorithm $\lfloor \frac{\pi}{4}\sqrt{2^n} \rfloor$ times. Therefore, the quantum resource required for Grover's algorithm attack is calculated as ($2 \times$ Table 6 $\times \lfloor \frac{\pi}{4}\sqrt{2^n} \rfloor$). The operation of the current small-scale quantum computer is unclear for the proposed LSH quantum circuit. According to the quantum computer development roadmap announced by IBM, it aims to develop more than 1000 qubits by 2023 and 1K-1M qubits after 2024. We predict that an attack on the LSH hash function will be possible after 2024 when the available resources of a large-scale quantum computer reach the resources required for Grover's algorithm.

Table 5. Quantum resource estimation results for the sequential LSH quantum circuit proposed in [19].

Algorithm	Qubits	Toffoli Gates	CNOT Gates	X Gates	Depth
LSH-256-224	1537	63,488	145,152	1536	210,051
LSH-256-256	1537	63,488	145,152	3492	210,049
LSH-512-224	3073	139,104	312,832	7663	421,851
LSH-512-256	3073	139,104	312,832	7696	421,851
LSH-512-384	3073	139,104	312,832	7668	421,850
LSH-512-512	3073	139,104	312,832	7680	421,852

Table 6. Quantum resource estimation results for our LSH parallel quantum circuit.

Algorithm	Qubits	Toffoli Gates	CNOT Gates	X Gates	Depth
LSH-256-224	1552	62,464	170,752	59,392	6879
LSH-256-256	1552	62,464	170,752	59,392	6879
LSH-512-224	3088	138,000	375,760	134,688	14,517
LSH-512-256	3088	138,000	375,760	134,688	14,517
LSH-512-384	3088	138,000	375,760	134,688	14,517
LSH-512-512	3088	138,000	375,760	134,688	14,517

Table 7. Quantum resource estimation result required for Grover's algorithm of parallel LSH quantum circuit.

Algorithm	Toffoli Gates	CNOT Gates	X Gates	Depth
LSH-256-224	1.91×2^{128}	1.3×2^{130}	1.81×2^{128}	1.68×2^{125}
LSH-256-256	1.91×2^{144}	1.3×2^{146}	1.81×2^{144}	1.68×2^{141}
LSH-512-224	1.05×2^{130}	1.43×2^{131}	1.02×2^{130}	1.77×2^{126}
LSH-512-256	1.05×2^{146}	1.43×2^{147}	1.02×2^{146}	1.77×2^{142}
LSH-512-384	1.05×2^{210}	1.43×2^{211}	1.02×2^{210}	1.77×2^{206}
LSH-512-512	1.05×2^{274}	1.43×2^{275}	1.02×2^{274}	1.77×2^{270}

5. Conclusions

In this paper, we proposed a parallel LSH quantum circuit that improved the previous sequential LSH quantum circuit [19]. We compare the results of the quantum resource estimation for the parallel quantum circuit of LSH with previous work and show the result of reducing the depth of the quantum circuit by about 96% through an efficient trade-off of quantum resources. Quantum resources required for a quantum preimage attack using Grover's search algorithm are determined according to the quantum circuit of the target hash function. Therefore, the results of this paper are expected to advance the timing of Grover's search algorithm attack on the LSH hash function in the future.

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

Funding: This research was financially supported by Hansung University.

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

References

1. Shor, P.W. Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer. *SIAM J. Comput.* **1997**, *26*, 1484–1509. [CrossRef]
2. Grover, L.K. A fast quantum mechanical algorithm for database search. In Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing, Philadelphia, PA, USA, 22–24 May 1996; pp. 212–219.
3. Grassl, M.; Langenberg, B.; Roetteler, M.; Steinwandt, R. Applying Grover's algorithm to AES: quantum resource estimates. In *Proceedings of the Post-Quantum Cryptography*; Springer: Berlin/Heidelberg, Germany, 2016; pp. 29–43.
4. Langenberg, B.; Pham, H.; Steinwandt, R. *Reducing the Cost of Implementing AES as a Quantum Circuit*; Technical Report; Cryptology ePrint Archive, Report 2019/854; 2019. Available online: https://eprint.iacr.org/2019/854 (accessed on 18 September 2022).
5. Jaques, S.; Naehrig, M.; Roetteler, M.; Virdia, F. Implementing Grover oracles for quantum key search on AES and LowMC. In Proceedings of the Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, 10–14 May 2020; pp. 280–310.
6. Anand, R.; Maitra, A.; Mukhopadhyay, S. Grover on SIMON. *Quantum Inf. Process.* **2020**, *19*, 1–17. [CrossRef]
7. Jang, K.; Choi, S.; Kwon, H.; Seo, H. *Grover on SPECK: Quantum Resource Estimates*; Cryptology ePrint Archive, Report 2020/640; 2020. Available online: https://eprint.iacr.org/2020/640 (accessed on 18 September 2022).
8. Jang, K.; Kim, H.; Eum, S.; Seo, H. *Grover on GIFT*; Cryptology ePrint Archive, Report 2020/1405; 2020. Available online: https://eprint.iacr.org/2020/1405 (accessed on 18 September 2022).
9. Schlieper, L. In-place implementation of Quantum-Gimli. *arXiv* **2020**, arXiv:2007.06319.
10. Jang, K.; Choi, S.; Kwon, H.; Kim, H.; Park, J.; Seo, H. Grover on Korean Block Ciphers. *Appl. Sci.* **2020**, *10*, 6407. [CrossRef]
11. Jang, K.; Song, G.; Kim, H.; Kwon, H.; Kim, H.; Seo, H. Efficient Implementation of PRESENT and GIFT on Quantum Computers. *Appl. Sci.* **2021**, *11*, 4776. [CrossRef]
12. Song, G.; Jang, K.; Kim, H.; Lee, W.K.; Seo, H. Grover on Caesar and Vigenère Ciphers. *IACR Cryptol. ePrint Arch.* **2021**, *2021*, 554.
13. Jang, K.; Song, G.; Kwon, H.; Uhm, S.; Kim, H.; Lee, W.K.; Seo, H. Grover on PIPO. *Electronics* **2021**, *10*, 1194. [CrossRef]
14. 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 18 September 2022).
15. Baksi, A.; Jang, K.; Song, G.; Seo, H.; Xiang, Z. Quantum implementation and resource estimates for rectangle and knot. *Quantum Inf. Process.* **2021**, *20*, 1–24. [CrossRef]

16. Song, G.; Jang, K.; Kim, H.; Eum, S.; Sim, M.; Kim, H.; Lee, W.; Seo, H. SPEEDY Quantum Circuit for Grover's Algorithm. *Appl. Sci.* **2022**, *12*, 6870. [CrossRef]
17. Huang, Z.; Sun, S. *Synthesizing Quantum Circuits of AES with Lower T-depth and Less Qubits*; Cryptology ePrint Archive, Paper 2022/620; 2022. Available online: https://eprint.iacr.org/2022/620 (accessed on 18 September 2022).
18. Amy, M.; Matteo, O.D.; Gheorghiu, V.; Mosca, M.; Parent, A.; Schanck, J. Estimating the Cost of Generic Quantum Pre-Image Attacks on SHA-2 and SHA-3. 2016. Available online: http://xxx.lanl.gov/abs/1603.09383 (accessed on 18 September 2022).
19. Song, G.j.; Jang, K.b.; Seo, H.j. Resource Eestimation of Grover Algorithm through Hash Function LSH Quantum Circuit Optimization. *J. Korea Inst. Inf. Secur. Cryptol.* **2021**, *31*, 323–330.
20. Debnath, S.; Linke, N.M.; Figgatt, C.; Landsman, K.A.; Wright, K.; Monroe, C. Demonstration of a small programmable quantum computer with atomic qubits. *Nature* **2016**, *536*, 63–66. [CrossRef] [PubMed]
21. Ofek, N.; Petrenko, A.; Heeres, R.; Reinhold, P.; Leghtas, Z.; Vlastakis, B.; Liu, Y.; Frunzio, L.; Girvin, S.; Jiang, L.; et al. Extending the lifetime of a quantum bit with error correction in superconducting circuits. *Nature* **2016**, *536*, 441–445. [CrossRef] [PubMed]
22. Kielpinski, D.; Monroe, C.; Wineland, D.J. Architecture for a large-scale ion-trap quantum computer. *Nature* **2002**, *417*, 709–711. [CrossRef] [PubMed]
23. Farhi, E.; Goldstone, J.; Gutmann, S. A quantum approximate optimization algorithm. *arXiv* **2014**, arXiv:1411.4028.
24. Farhi, E.; Goldstone, J.; Gutmann, S.; Neven, H. Quantum algorithms for fixed qubit architectures. *arXiv* **2017**, arXiv:1703.06199.
25. Barends, R.; Kelly, J.; Megrant, A.; Veitia, A.; Sank, D.; Jeffrey, E.; White, T.C.; Mutus, J.; Fowler, A.G.; Campbell, B.; et al. Superconducting quantum circuits at the surface code threshold for fault tolerance. *Nature* **2014**, *508*, 500–503. [CrossRef] [PubMed]
26. Kim, D.C.; Hong, D.; Lee, J.K.; Kim, W.H.; Kwon, D. LSH: A new fast secure hash function family. In Proceedings of the International Conference on Information Security and Cryptology, Latin America Florianópolis, Brazil, 17–19 September 2014; pp. 286–313.
27. Cuccaro, S.A.; Draper, T.G.; Kutin, S.A.; Moulton, D.P. A new quantum ripple-carry addition circuit. *arXiv* **2004**, arXiv:quant-ph/0410184.