



Article

Grover on Scrypt

Gyeongju Song  and Hwajeong Seo 

Information Computer Engineering, Hansung University, Seoul 02876, Republic of Korea;
thdrudwn98@hansung.ac.kr

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

Abstract: This paper presents an optimized quantum circuit for the *scrypt* cryptographic algorithm. We applied various optimization techniques to reduce the DW cost, which is the product of the time and space complexity of quantum circuits. In our proposed method, the number of ancilla qubits was significantly reduced through the use of optimized inverse operations, while the depth was minimized by implementing parallel structures. For the SHA-256, we devised a structure that achieves a substantial reduction in the number of ancilla qubits with only a slight increase in quantum circuit depth. By cleaning the dirty ancilla qubits used in the previous round through inverse operations, we enabled their reuse in each subsequent round. Specifically, we reduced the number of 8128 ancilla qubits, achieving this with an increase of only 6 in the full depth of the quantum circuit. Additionally, within Salsa20/8 in SMix, we reused qubits through inverse operations and performed some operations in parallel to reduce both the number of qubits and the overall quantum circuit depth. Finally, our quantum circuit for *scrypt* demonstrates a significant reduction in the width (the number of qubits) with only a minimal increase in the full quantum circuit depth.

Keywords: quantum implementation; grover algorithm; scrypt



Citation: Song, G.; Seo, H. Grover on Scrypt. *Electronics* **2024**, *13*, 3167.
<https://doi.org/10.3390/electronics13163167>

Academic Editors: Ilias K. Savvas and Apostolis Xenakis

Received: 15 July 2024

Revised: 1 August 2024

Accepted: 8 August 2024

Published: 10 August 2024



Copyright: © 2024 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

Quantum computers leverage the properties of quantum mechanics to utilize qubits that can exist in both 0 and 1 states simultaneously, providing powerful computational capabilities for specific problems. Shor's algorithm efficiently solves the integer factorization problem for large numbers [1], threatening the security of public-key cryptography such as RSA. Grover's algorithm efficiently addresses the problem of finding a desired item in an unsorted database. While classical algorithms take $O(N)$ time complexity, Grover's algorithm solves it in $O(\sqrt{N})$ time complexity [2]. This greatly accelerates brute force attacks for finding keys in symmetric key algorithms (e.g., AES). The advent of quantum computers significantly threatens existing cryptographic systems, including public-key cryptography and symmetric-key cryptography. In response, NIST has launched the Post-Quantum Cryptography Standardization competition to find replacements for current cryptographic standards. NIST has also proposed the cost of Grover's algorithm attack, using the maximum depth (MAXDEPTH) of AES as a metric to assess the quantum security strength of block ciphers. To estimate Grover's algorithm attack cost for a target cipher and verify the post-quantum security levels proposed by NIST, it is necessary to implement the cipher as a quantum circuit. Quantum circuits are required inside the oracle of Grover's algorithm. Optimized quantum circuits allow for a more accurate determination of post-quantum security levels. Motivated by this, many prior studies have conducted optimized implementations of various ciphers as quantum circuits, such as LSH [3,4], PIPO [5], SPEEDY [6], and so on [7–18].

This paper presents the implementation of a quantum circuit for *scrypt*. We implement the *scrypt* algorithm on a quantum computer with low circuit complexity and provide a detailed explanation of the various optimization techniques applied to achieve this. We propose a method to ancilla qubits to clean $|0\rangle$ through inverse operations and to reuse them in

the next iteration. During this process, we perform inverse operations in parallel with subsequent operations to minimize the increase in depth. Additionally, we follow a qubit update method, where if the pre-update value of an updated qubit is needed, we use the updated value in the next operation and then apply the inverse operation to restore the pre-update value. Using this method, we significantly reduce the number of ancilla qubits while only slightly increasing the overall depth. Consequently, we reduce the circuit complexity, calculated as the product of time and space complexity ($DW\text{-cost} = \text{qubit} \times T\text{-depth}$), for *scrypt*. We provide the estimated resources for the optimized quantum circuit. To the best of our knowledge, this is the first quantum circuit implementation for *scrypt*. Therefore, it is difficult to compare the optimization results of our quantum circuit with previous *scrypt* results. We perform quantum circuit optimization based on the original reference implementation of *scrypt*. This *scrypt* quantum circuit can be used to operate Grover's algorithm to estimate resources. Therefore, this paper will serve as the foundation for future implementations of the *scrypt* algorithm in quantum circuits. Further optimization of future *scrypt* quantum circuits will help us more accurately assess the post-quantum security of the algorithm. The structure of this paper is as follows: in Section 2, related research on NIST's PQC Standards Competition and quantum computing is written to help understand the paper. Section 3 explains the implementation of the proposed *scrypt* quantum circuit. This section describes the optimization techniques applied to each *scrypt* function in detail. Section 4 estimates and analyzes the resources required for the proposed quantum circuit. We use the projectQ [19] tool to analyze resources and verify implementation results. The DW cost is calculated and written based on the estimated resources. Finally, Section 5 concludes the paper.

1.1. Our Contribution

1.1.1. First Quantum Circuit for *Scrypt*

To the best of our knowledge, this is the first quantum circuit for *scrypt*. The *scrypt* algorithm has been implemented in various optimized implementations across different hardware and software environments [20–22]. However, there has been no quantum circuit result suitable for a quantum computer environment. Although optimizations such as memory optimization have been studied for *scrypt* [23], we implemented the quantum circuit based on the original reference code. We present an optimization direction for the *scrypt* quantum circuit and establish a basis for assessing initial post-quantum security. These research results will serve as a foundation for future implementations of *scrypt* quantum circuits.

1.1.2. Qubit Reuse through Inverse Computation

We propose a method where, after using ancilla qubits in a clean $|0\rangle$ state, the dirty qubit is returned to a clean $|0\rangle$ state through inverse operations, allowing them to be reused in subsequent iterations. In SHA-256, dirty ancilla qubits that have been used can be reset to a clean $|0\rangle$ state through inverse operations, enabling reuse in all loops. Using this method, we reduced the number of qubits by 8128 in SHA-256 within $PBKDF2_{SHA256}$ with a slight increase in depth (approximately depth 6). In the Salsa function within SMix, we avoided using ancilla qubits for intermediate value storage during the 'Operation on columns' and 'Operate on rows' steps. Instead, we updated the intermediate values directly into the inputs. When the pre-update values of the updated qubits were needed for subsequent operations, we used the updated values in the following operations and then applied inverse operations to restore the pre-update values.

1.1.3. Parallel Structures in SHA-256 and Salsa20/8

We designed the internal operations to be as parallel as possible to reduce the depth of the quantum circuit. In SHA-256, we reset the used ancilla qubits to a clean state $|0\rangle$ through inverse operations and designed the circuit to reuse them in all loops. However, adding inverse operations increases the depth of the quantum circuit. To minimize the impact of

these inverse operations on the circuit depth, we implemented them to operate in parallel with subsequent operations. Similarly, within the salsa function, the inverse operations that restore the updated qubits to their pre-update state and the internal additions operate partially in parallel with subsequent operations. Therefore, while inverse operations were added to reduce the number of qubits, they do not significantly affect the overall depth.

1.1.4. Optimized Quantum Circuit with Low Complexity

We significantly reduced the number of ancilla qubits. To achieve this, we implemented additional operations in parallel so that the quantum circuit depth is minimally affected. We efficiently implemented *script* into a quantum circuit by reducing the ancilla qubits and depth. These results demonstrate an optimized circuit complexity (DW cost) calculated as the product of time and space complexity.

2. Background

2.1. NIST's PQC Standards Competition

The National Institute of Standards and Technology (NIST) initiated the Post-Quantum Cryptography (PQC) Standards competition to address the imminent threat quantum computing poses to classical cryptographic systems. This project, officially announced in 2016, aims to identify and standardize cryptographic algorithms resistant to quantum attacks. As part of this competition, NIST established the post-quantum security level categories shown in Table 1 to classify algorithms [24–26]. These categories are defined based on the complexity of the Grover attack that can be quantified in terms of circuit size. The three proposed security levels (security level 1, 3, and 5) are defined based on the security of different variants of AES, considering attacks on quantum computers. Considering that NIST established security categories based on the depth of quantum circuits, optimizing the depth is crucial in quantum cryptographic implementations to determine the security level of the cryptographic algorithm accurately.

Table 1. NIST post-quantum security level categories.

Level	Cipher	Category
1	AES 128	2^{157} / MAXDEPTH quantum gates
3	AES 192	2^{221} / MAXDEPTH quantum gates
5	AES 256	2^{285} / MAXDEPTH quantum gates

2.2. Quantum Computing

Quantum computing is a technology that performs calculations by leveraging the principles of quantum mechanics [27]. Unlike classical computers, which process information using bits, quantum computers use qubits. Qubits can exist in a superposition state, allowing them to represent both 0 and 1 simultaneously, thereby enabling parallel computation. Additionally, through the phenomenon of entanglement, interactions between distant qubits can be maximized. These characteristics enable quantum computers to solve certain computational problems much faster than classical computers.

Grover's quantum algorithm [2] can significantly speed up database searches by a factor of a square root, drastically reducing the time required to find keys in symmetric-key cryptographic systems (e.g., AES). This implies that quantum computers could easily attack many current symmetric-key cryptographic systems. Consequently, advancements in quantum computing are driving the development of new cryptography, such as post-quantum cryptography (PQC) [28].

Quantum Circuit

A quantum circuit performs quantum operations using qubits, transforming and interacting with qubit states through quantum gates. The depth of a circuit refers to the maximum number of computational steps involving quantum gates, while the width of a

circuit indicates the number of qubits being processed. Thus, a quantum circuit is composed of width, depth, and quantum gates. Major quantum gates include the X gate, CNOT gate, and Toffoli gate, as shown in Figure 1:

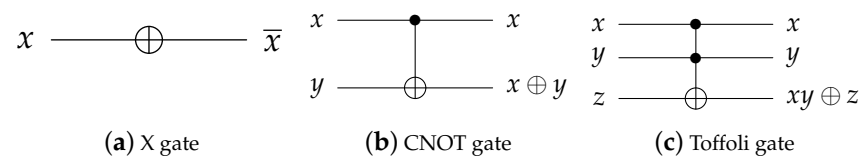


Figure 1. Quantum gates.

2.3. *Scrypt*

Scrypt is a password-based key derivation function developed by Colin Percival in 2009 [29]. It was designed to be computationally intensive and memory-hard, making it resistant to hardware attacks, particularly those using ASICs (application-specific integrated circuits) and GPUs (graphics processing units). This makes *scrypt* an ideal choice for highly secure applications such as password hashing and cryptocurrencies. The detailed parameters of *scrypt* are available in Table 2.

Table 2. Parameters in *scrypt*.

Parameters	Meaning	Remarks
passwd	Password entered by user	-
passwdlen; len (pwd)	Length of the password	-
salt	A unique, randomly generated value	A positive integer
saltlen; len (salt)	Length of the salt	-
N	CPU/memory cost parameter	A power of two
r	Block size parameter	A positive integer
p	Parallelization parameter	A positive integer satisfying

Algorithm 1 presents the pseudocode for the operation of *scrypt*. The PBKDF2 function generates an initial derived key B using a password (passwd) and a salt. This function typically uses the SHA-256 hash function internally. The SMix function is a core component of the *scrypt* algorithm, responsible for performing memory-hard operations to enhance security against brute-force attacks. The function takes the following parameters: B, the input, which is the initial derived key block and must be 128r bytes, and N, which determines the computational cost and must be to the power of 2.

Algorithm 2 illustrates the $BLOCKMIX_{salsa8}$ and SMix functions used in the *scrypt* algorithm. The $PBKDF2_{SHA256}$ function generates a 128rp byte string. This generated string is divided into p blocks of equal length, and the SMix function is called for each block. The results from the SMix function are then concatenated and used as the salt in the final $PBKDF2_{SHA256}$ operation. In this final $PBKDF2_{SHA256}$ operation, the original password and the new salt are used to generate the final dkLen byte output key.

The SMix function is a core component of the *scrypt* algorithm responsible for its memory-hard properties. According to the *scrypt* RFC [29], the recommended block size parameter is $r = 8$. With this parameter, the initial input block for SMix is only 1 kB, making it suitable for the cache. However, the SMix function expands this 1 kB block into an array of N blocks. These blocks are then accessed iteratively in a pseudorandom order determined by the contents of previously accessed blocks. If N is sufficiently large, the SMix function becomes memory-bound, resulting in significant computational costs for running *scrypt*.

Algorithm 1 *Scrypt* algorithm**Input:** passwd, passwdlen, salt, saltlen, N, r, p

```

//  $(B_0 \dots B_{p-1}) \leftarrow \text{PBKDF2}(P, S, 1, p \times \text{MFLen})$ 
1:  $\text{PBKDF2}_{\text{SHA256}}(\text{passwd}, \text{passwdlen}, \text{salt}, \text{saltlen}, 1, B, p \times 128 \times r)$ 

2: for (i = 0 to i < p) :
3:    $\text{SMix}(B_i, N)$  //  $B_i \leftarrow \text{MF}(B_i, N)$ 

//  $\text{DK} \leftarrow \text{PBKDF2}(P, B, 1, \text{dkLen})$ 
4:  $\text{PBKDF2}_{\text{SHA256}}(\text{passwd}, \text{passwdlen}, B, p \times 128 \times r, 1, \text{buf}, \text{buflen})$ 

```

Algorithm 2 $\text{BLOCKMIX}_{\text{salsa8}}$ and SMix algorithms

```

1: function  $\text{SMix}(B, N)$ 
2:  $X \leftarrow B$ 
3: for (i=0 to i<N) :
4:    $V_i \leftarrow X$ 
5:    $X \leftarrow \text{BLOCKMIX}_{\text{salsa8}}(X)$ 

6: for (i = 0 to i < N) :
7:    $j \leftarrow \text{Integerify}(X) \bmod N$ 
8:    $X \leftarrow \text{BLOCKMIX}_{\text{salsa8}}(X \oplus V_j)$ 

9: function  $\text{BLOCKMIX}_{\text{salsa8}}$ 
10:  $X \leftarrow B_{2r-1}$ 
11: for (i = 0 to i <  $2 \times r$ ) :
12:  $X \leftarrow \text{salsa20}_8(X \oplus B_i)$ 
13:  $Y_i \leftarrow X$ 

```

3. Quantum Circuit on Scrypt

In this section, we describe the implementation of *scrypt* in a quantum circuit. The key operations of *scrypt* include SMix and $\text{PBKDF2}_{\text{SHA256}}$. Within the SMix function, the Salsa20/8 (Salsa20/8 is the 8-round version of Salsa20 [30]) algorithm is utilized, and the SHA-256 algorithm is employed within the $\text{PBKDF2}_{\text{SHA256}}$ function.

The ultimate goal of our *scrypt* quantum circuit implementation is to reduce the circuit complexity, i.e., the product of time and space complexity (DW cost; width \times T-depth), by adjusting the depth-width balance of the *scrypt* quantum circuit to be used in the Grover Oracle. To achieve this, we introduced parallel structures and optimization methods into the *scrypt* quantum circuit. We designed a structure for SHA-256 that significantly reduces the number of ancilla qubits with only a slight increase in depth. To minimize the number of ancilla qubits used in each round, we cleaned and reused the dirty ancilla qubits from the previous round through inverse operations, allowing their continuous use in each round. In the salsa20_8 function within SMix , we avoided using ancilla qubits for storing intermediate values during the ‘Operation on columns’ and ‘Operate on rows’ steps. Instead, we updated the intermediate values directly in the inputs. When the pre-update values were needed for subsequent operations, we used the updated values and then applied inverse operations to restore the pre-update values. Thus, we did not use ancilla qubits to store the updated results. This approach allowed the inverse operations to restore the updated qubits to their pre-update state, and the internal additions operated partially in parallel with subsequent operations, thereby reducing the overall depth. As a result, we added the inverse operation to reduce the number of qubits, but it did not significantly affect the full depth.

3.1. Notation

In this paper, qubits are represented as arrays. For example, a 32-length qubit x is written as $x[i]$, where $0 \leq i \leq 31$. An array declared with a certain data type, such as an unsigned 32-bit integer array $X[n]$, is represented as an $n \times 32$ array. To avoid confusion, we distinguish between one-dimensional and two-dimensional qubit arrays using lowercase and uppercase letters, respectively (lowercase: one-dimensional qubit arrays, uppercase: two-dimensional qubit arrays). We use mixing-XOR (hereafter called m-XOR) in the internal optimization process. m-XOR, introduced in [13], is a method that transforms the creating operations of CNOT gates into updating operations to minimize qubit usage. Here, a creating operation refers to an out-of-place operation that stores the result of $a \oplus b$ in c ($c = a \oplus b$), while an updating operation refers to an in-place operation that stores the result of $a \oplus b$ in qubit b ($b = a \oplus b$).

3.2. Addition in Scrypt

SHA-256 in PBKDF2 and Salsa20/8 in the SMix function use the $(2n + 3)$ -depth quantum adder proposed in [31]. The quantum adder uses n -bit a and b as inputs along with a 1-bit ancilla c , performing an in-place addition: $ADD(a, b, c) = (a, a \oplus b, c)$. Here, a retains its original value, while b is updated to $a \oplus b$. The ancilla c is input in a clean $|0\rangle$ state and remains clean after the operation, allowing it to be reused in subsequent addition operations. Consequently, this quantum circuit allows for addition using only a single qubit as the ancilla.

3.3. SHA-256 in PBKDF2_{SHA256}

Within the password-based key derivation function 2 (PBKDF2_{SHA256}), SHA-256 operates. We implemented the SHA-256 quantum circuit in PBKDF2 by significantly reducing ancilla qubits while slightly increasing the depth. The operations performed in SHA-256 are as follows:

$Ch(x, y, z) = ((x \& (y \wedge z)) \wedge z);$
 $Maj(x, y, z) = ((x \& (y \mid z)) \mid (y \& z));$
 $SHR(x, n) = (x \gg n);$
 $ROTR(x, n) = ((x \gg n) \mid (x \ll (32 - n)));$
 $S0(x) = (ROTR(x, 2) \wedge ROTR(x, 13) \wedge ROTR(x, 22));$
 $S1(x) = (ROTR(x, 6) \wedge ROTR(x, 11) \wedge ROTR(x, 25));$
 $s0(x) = (ROTR(x, 7) \wedge ROTR(x, 18) \wedge SHR(x, 3));$
 $s1(x) = (ROTR(x, 17) \wedge ROTR(x, 19) \wedge SHR(x, 10)).$

Algorithms 3–5 illustrate the quantum circuits for $S0$, $S1$, and Maj functions in PBKDF2_{SHA256}, respectively. We propose a method to reuse ancilla qubits ($S0_{anc}$, $S1_{anc}$, and $Maj_{anc1,3}$ in the algorithms) by returning them to a clean $|0\rangle$ state through inverse operations, allowing them to be reused in each iteration of the functions. Thus, inverse operations can reset ancilla qubits to $|0\rangle$, enabling their reuse in all loops.

We reduce the full depth of the quantum circuit by ensuring that the $S1^\dagger$, $S0^\dagger$, and Maj^\dagger operations run in parallel with subsequent operations. Figure 2 illustrates the parallel processing of the RND operations within SHA-256. $S1^\dagger$ and $S0^\dagger$ run in parallel with the subsequent operations Ch and Maj , respectively, while Maj^\dagger operates in parallel with the Add operation of the next RNDr function. Using this method, we reduced the number of qubits by 8128 with only a slight increase in depth (approximately 6) when SHA-256 operates once. As a result, the $S0_{anc}$, $S1_{anc}$, and $Maj_{anc1,3}$ qubits can continue to be reused in the next round of the RNDr function in PBKDF2_{SHA256}.

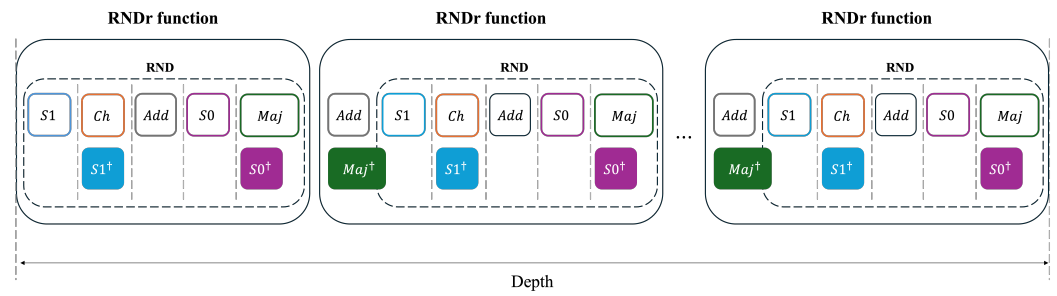


Figure 2. RNDr function of $PBKDF2_{SHA256}$ with inverse functions ($S1^\dagger$, $S0^\dagger$, Maj^\dagger) operating in parallel.

Algorithm 3 $S0$ quantum circuit in SHA-256

Input: x , h , $S0_{anc}$

- 1: \diamond **Point 1: Start inverse operation**
- 2: **for** ($i = 0$ to 32) :
- 3: $S0_{anc}[i] \leftarrow CNOT(x[(i + 2)\%32], S0_{anc}[i])$
- 4: $S0_{anc}[i] \leftarrow CNOT(x[(i + 13)\%32], S0_{anc}[i])$
- 5: $S0_{anc}[i] \leftarrow CNOT(x[(i + 22)\%32], S0_{anc}[i])$
- 6: \diamond **Point 2: End inverse operation**

7: $h \leftarrow Add(S0_{anc}, h)$

8: \diamond **Start inverse operation from Point 1 to Point 2.**

Algorithm 4 $S1$ quantum circuit in SHA-256

Input: x , h , $S1_{anc}$

- 1: \diamond **Point 1: Start inverse operation**
- 2: **for** ($i = 0$ to 32) :
- 3: $S1_{anc}[i] \leftarrow CNOT(x[(i + 6)\%32], S1_{anc}[i])$
- 4: $S1_{anc}[i] \leftarrow CNOT(x[(i + 11)\%32], S1_{anc}[i])$
- 5: $S1_{anc}[i] \leftarrow CNOT(x[(i + 25)\%32], S1_{anc}[i])$
- 6: \diamond **Point 2: End inverse operation**
- 7: $h \leftarrow Add(S1_{anc}, h)$

8: // Reset $S1_{anc}$ to clean $|0\rangle$

9: \diamond **Start inverse operation from Point 1 to Point 2.**

The quantum circuits of $s0$ and $s1$ are shown in Algorithms 6 and 7. In $s0$ and $s1$, the SHR operation performs an n -bit right shift, filling the n -bit portion with zeros, so CNOT does not operate in this n -bit region. Consequently, we omitted the shift operation by not applying CNOT in the n -bit portion. To exclude the CNOT operation for n bits, the number of loop iterations is set to $32 - n$ (where $n = 3, 10$) in line 4 of Algorithms 6 and 7. This approach can reduce the use of CNOT gates. Additionally, through the m-XOR method, we modified the result of $CNOT(ROTR(x, 17), ancilla)$ and $CNOT(ROTR(x, 18), ancilla)$ for subsequent unused h to $CNOT(ROTR(x, 17), h)$ and $CNOT(ROTR(x, 19), h)$, thereby reducing the number of ancilla qubits. ROTR performs an n -bit right rotation, but in a quantum computer that executes bitwise operations, the rotation operation can be omitted by adjusting the indices of the control qubits.

Algorithm 5 *Maj* quantum circuit in SHA-256**Input:** $x, h, Maj_{anc1}, Maj_{anc2}, Maj_{anc3}$

```

1: for ( $i = 0$  to 32) :
2:    $Maj_{anc1}[i] \leftarrow Toffoli(y[i], z[i], Maj_{anc1}[i])$ 

3: //OR-gate operation
4: for ( $i = 0$  to 32) :
5:    $y[i] \leftarrow X(y[i])$ 
6:    $z[i] \leftarrow X(z[i])$ 
7:    $Maj_{anc2}[i] \leftarrow Toffoli(y[i], z[i], Maj_{anc2}[i])$ 
8:    $y[i] \leftarrow X(y[i])$ 
9:    $z[i] \leftarrow X(z[i])$ 
10:   $Maj_{anc2}[i] \leftarrow X(Maj_{anc2}[i])$ 

11: for ( $i = 0$  to 32) :
12:   $Maj_{anc3}[i] \leftarrow Toffoli(x[i], Maj_{anc2}[i], Maj_{anc3}[i])$ 

13: //OR-gate operation (Reset  $Maj_{anc2}$ )
14: for ( $i = 0$  to 32) :
15:   $y[i] \leftarrow X(y[i])$ 
16:   $z[i] \leftarrow X(z[i])$ 
17:   $Maj_{anc2}[i] \leftarrow Toffoli(y[i], z[i], Maj_{anc2}[i])$ 
18:   $y[i] \leftarrow X(y[i])$ 
19:   $z[i] \leftarrow X(z[i])$ 
20:   $Maj_{anc2}[i] \leftarrow X(Maj_{anc2}[i])$ 

21: //OR-gate operation.
22: for ( $i = 0$  to 32) :
23:   $Maj_{anc1}[i] \leftarrow X(Maj_{anc1}[i])$ 
24:   $Maj_{anc3}[i] \leftarrow X(z[i])$ 
25:   $Maj_{anc2}[i] \leftarrow Toffoli(Maj_{anc1}[i], Maj_{anc3}[i], Maj_{anc2}[i])$ 
26:   $Maj_{anc1}[i] \leftarrow X(Maj_{anc1}[i])$ 
27:   $Maj_{anc3}[i] \leftarrow X(Maj_{anc3}[i])$ 
28:   $Maj_{anc2}[i] \leftarrow X(Maj_{anc2}[i])$ 

29:  $h \leftarrow Add(Maj_{anc2}, h)$ 

30: for ( $i = 0$  to 32) :
31:   $Maj_{anc3}[i] \leftarrow Toffoli(y[i], z[i], Maj_{anc3}[i])$  //Reset  $Maj_{anc3}$ 
32:   $Maj_{anc1}[i] \leftarrow Toffoli(y[i], z[i], Maj_{anc1}[i])$  //Reset  $Maj_{anc1}$ 

```

Algorithm 6 *s0* quantum circuit in SHA-256**Input:** x, h

```

1: for ( $i = 0$  to 32) :
2:    $h[i] \leftarrow CNOT(x[(i + 7)\%32], h[i])$ 
3:    $h[i] \leftarrow CNOT(x[(i + 18)\%32], h[i])$ 

4: //  $h$  update
5: for ( $i = 0$  to 32 - 3) :
6:    $h[i] \leftarrow CNOT(x[i + 3], h[i])$ 

```


Algorithm 7 s1 quantum circuit in SHA-256**Input:** x, h

```

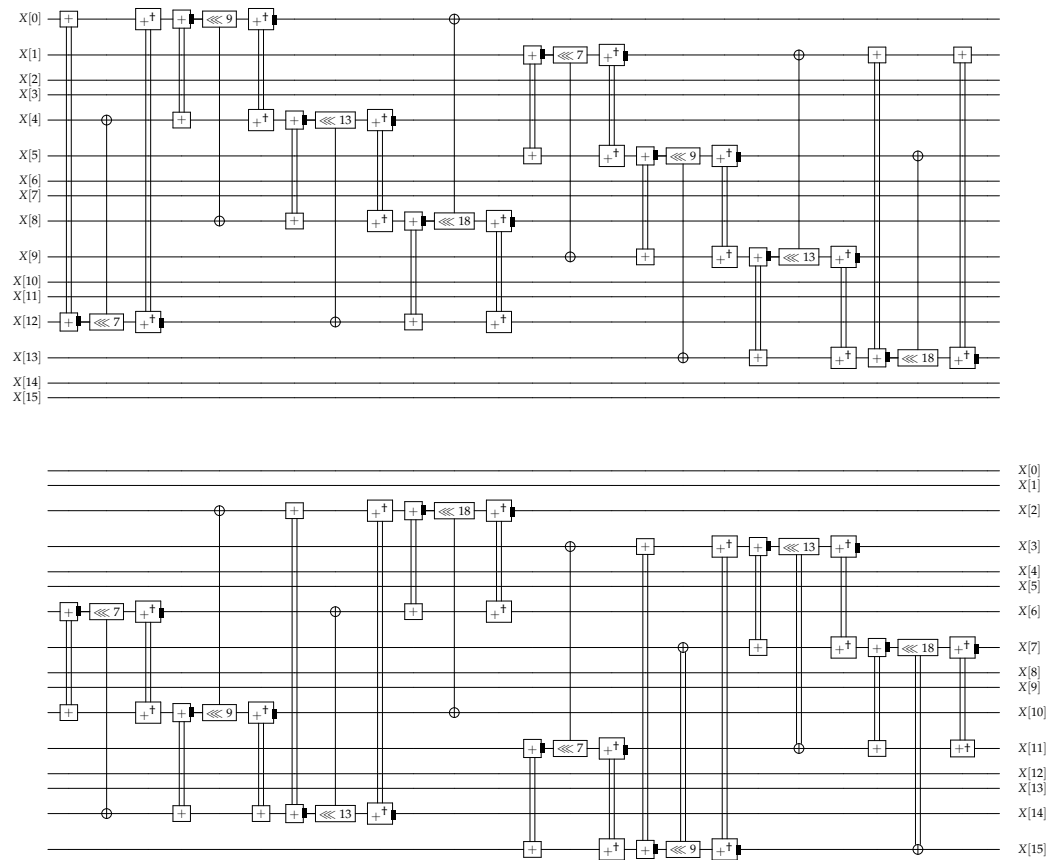
1: for ( $i = 0$  to 32) :
2:    $h[i] \leftarrow \text{CNOT}(x[(i + 17)\%32], h[i])$ 
3:    $h[i] \leftarrow \text{CNOT}(x[(i + 19)\%32], h[i])$ 

4: //  $h$  update
5: for ( $i = 0$  to 32 − 10) :
6:    $h[i] \leftarrow \text{CNOT}(x[i + 10], h[i])$ 

```

3.4. Salsa20/8 in SMix

Salsa20/8 is the eight-round version of Salsa20 and is used within the SMix function of *script*. Figures 3 and 4 illustrate the operations in the columns and rows of Salsa20/8, respectively. In the Figure, $+$ represents the in-place addition of two qubits, while $+\dagger$ performs the inverse addition. Each CNOT operation performs a right shift of n bits ($\lll n$) on the control qubit. Instead of separately performing the shift operation on the control qubit, we adjusted the index to operate. We will refer to this method as index-rotating, which allows us to omit the shift operation on the qubits. X is a 16×32 array of qubits and is denoted as $X[i][j]$ (where $0 \leq i \leq 16$ and $0 \leq j \leq 32$). Therefore, $X[0], X[1], \dots, X[15]$ represent an array of 32 qubits. The CNOT gate for a two-dimensional array X , i.e., $\text{CNOT}(X[i_1], X[i_2])$, involves XOR operations between $X[i_1][j]$ and $X[i_2][j]$, where j ranges from 0 to 31. Thus, 32 CNOT gates are used. The Add operation utilizes the adder described in Section 3.2.

**Figure 3.** Operation on columns in Salsa20/8.

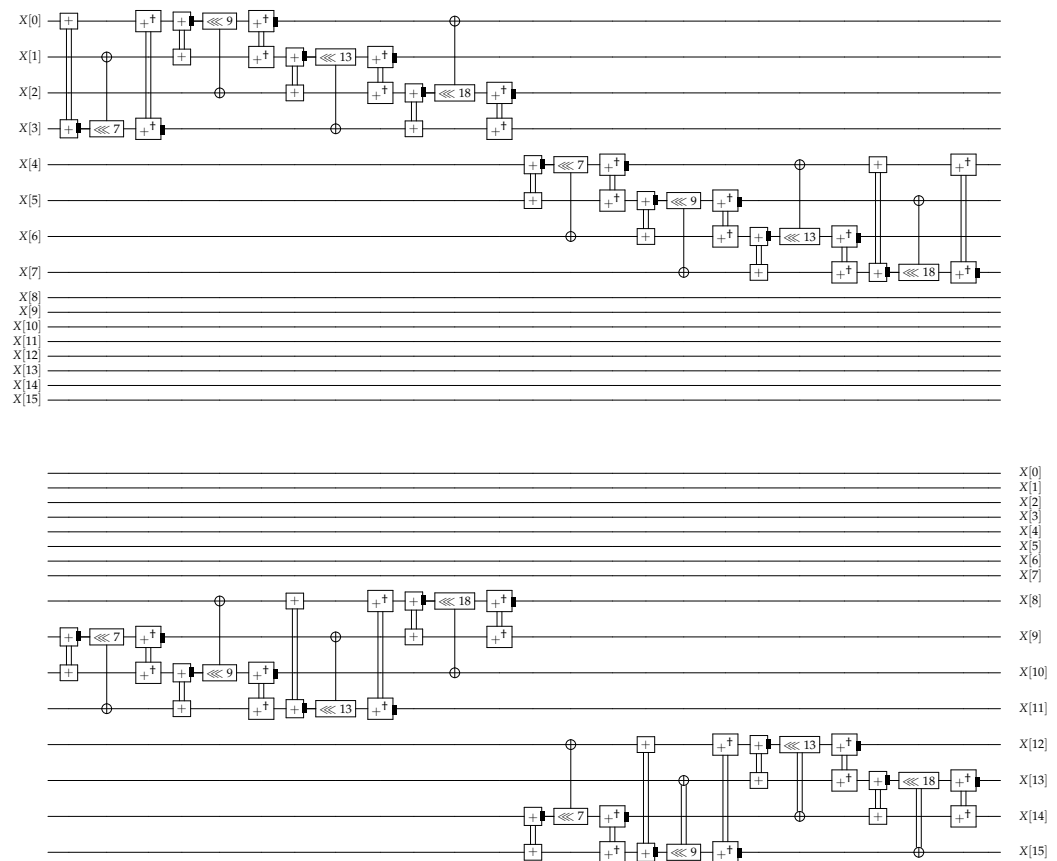


Figure 4. Operation on rows in Salsa20/8.

4. Evaluation

In this paper, we optimized the quantum circuit for *script* by adjusting the width–depth balance. To operate Grover’s algorithm, the quantum circuit of the target cipher, *script* in this case, must be placed inside the oracle. This allows us to estimate the quantum resources needed for Grover’s key search. We presented a quantum circuit for *script*. Tables 3 and 4 show the estimated quantum resources for the internal functions of *script*. The quantum resources needed for these quantum circuits were estimated using the ProjectQ tool [19]. We designed a structure for SHA-256 that significantly reduces the number of ancilla qubits with only a slight increase in depth. To minimize the number of ancilla qubits allocated in each round, we cleaned the dirty ancilla qubits used in the previous round through inverse operations, allowing continuous use in each round. In the S0 function, 2048 ancilla qubits were used (64 loops \times 32), and in the S1 function, 2048 ancilla qubits were used (64 loops \times 32). In the Maj function, 6144 ancilla qubits were used (64 loops \times 96). However, through inverse operations, we reduced the number of ancilla qubits to 32 in the S0 function, 32 in the S1 function, and 2048 in the Maj function (64 loops \times 32), reducing a total of 8128 qubits. The inverse operations were designed to run in parallel with subsequent operations, resulting in an increase of only 6 in the total depth of the quantum circuit used for inverse operations. Furthermore, in Salsa20/8 within SMix, we reused qubits through inverse operations and performed some parallel operations to reduce both the number of qubits and the full depth, while the m-XOR and index-rotating techniques were used to reduce the number of quantum gates. As a result, our quantum circuit ultimately showed a significant reduction in width (number of qubits) with almost no increase in full depth.

Table 3. Quantum resource estimation for the *script* quantum circuit.

Function	Qubit	Quantum Gates		
		Toffoli	CNOT	X
SHA-256 in PBKDF2	17,100	58,448	137,888	63,231
Salsa20/8 in SMix	1040	16,592	145,776	16,060

Table 4. Quantum resource (decomposed quantum gate) estimation for the *script* quantum circuit and DW cost.

Function	Quantum Gates		T-Depth	Full Depth	DW-Cost
	T	T ⁺			
SHA-256 in PBKDF2	179,230	225,774	292,240	138,358	1.1×2^{31}
Salsa20/8 in SMix	57,448	58,424	82,960	35,050	1.28×2^{26}

5. Conclusions

In this paper, we applied optimization techniques to reduce the DW cost in a *script* quantum circuit. By implementing inverse operations, we reduced the number of qubits used, and by designing a parallel structure, we minimized the circuit's depth. We also estimated the quantum resources required for this implementation. To the best of our knowledge, this is the first quantum circuit implementation of *script*. Therefore, these findings will serve as a foundation for future *script* quantum circuit implementations and can be used to assess the quantum security level.

Author Contributions: Software, G.S.; Investigation, G.S.; Writing—original draft, G.S.; Writing—review & editing, H.S.; 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.

Data Availability Statement: Data are contained within the article.

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

References

- Shor, P.W. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Rev.* **1999**, *41*, 303–332. [\[CrossRef\]](#)
- 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.
- Song, G.J.; Jang, K.B.; Seo, H.J. Resource Estimation of Grover Algorithm through Hash Function LSH Quantum Circuit Optimization. *J. Korea Inst. Inf. Secur. Cryptol.* **2021**, *31*, 323–330.
- 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. [\[CrossRef\]](#)
- Jang, K.; Song, G.; Kwon, H.; Uhm, S.; Kim, H.; Lee, W.K.; Seo, H. Grover on PIPO. *Electronics* **2021**, *10*, 1194. [\[CrossRef\]](#)
- 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. [\[CrossRef\]](#)
- Song, G.; Jang, K.; Seo, H. Improved Low-Depth SHA3 Quantum Circuit for Fault-Tolerant Quantum Computers. *Appl. Sci.* **2023**, *13*, 3558. [\[CrossRef\]](#)
- Jang, K.; Baksi, A.; Breier, J.; Seo, H.; Chattopadhyay, A. Quantum implementation and analysis of default. *Cryptogr. Commun.* **2023**, 1–17. [\[CrossRef\]](#)
- Chauhan, A.K.; Sanadhya, S.K. Quantum resource estimates of grover's key search on aria. In *Proceedings 10, Proceedings of the Security, Privacy, and Applied Cryptography Engineering: 10th International Conference, SPACE 2020, Kolkata, India, 17–21 December 2020*; Springer: Berlin/Heidelberg, Germany, 2020; pp. 238–258.
- Anand, R.; Maitra, A.; Mukhopadhyay, S. Grover on SIMON. *Quantum Inf. Process.* **2020**, *19*, 340. [\[CrossRef\]](#)
- Rahman, M.; Paul, G. Grover on KATAN: Quantum resource estimation. *IEEE Trans. Quantum Eng.* **2022**, *3*, 1–9. [\[CrossRef\]](#)
- Jang, K.; Song, G.; Kim, H.; Kwon, H.; Kim, H.; Seo, H. Parallel quantum addition for Korean block ciphers. *Quantum Inf. Process.* **2022**, *21*, 373. [\[CrossRef\]](#)

13. Liu, Q.; Preneel, B.; Zhao, Z.; Wang, M. Improved Quantum Circuits for AES: Reducing the Depth and the Number of Qubits. In *Proceedings of the International Conference on the Theory and Application of Cryptology and Information Security*; Springer: Singapore, 2023; pp. 67–98.
14. Zou, J.; Li, L.; Wei, Z.; Luo, Y.; Liu, Q.; Wu, W. New quantum circuit implementations of SM4 and SM3. *Quantum Inf. Process.* **2022**, *21*, 181. [\[CrossRef\]](#)
15. Song, G.; Jang, K.; Kim, H.; Lee, W.K.; Hu, Z.; Seo, H. Grover on SM3. In *Proceedings of the Information Security and Cryptology–ICISC 2021: 24th International Conference, Seoul, Republic of Korea, 1–3 December 2021*; Revised Selected Papers; Springer: Cham, Switzerland, 2022; pp. 421–433.
16. Grassl, M.; Langenberg, B.; Roetteler, M.; Steinwandt, R. Applying Grover’s algorithm to AES: Quantum resource estimates. In *Proceedings of the International Workshop on Post-Quantum Cryptography*; Springer: Cham, Switzerland, 2016; pp. 29–43.
17. Huang, Z.; Sun, S. Synthesizing quantum circuits of AES with lower t-depth and less qubits. In *Proceedings of the International Conference on the Theory and Application of Cryptology and Information Security*; Springer: Cham, Switzerland, 2022; pp. 614–644.
18. Shi, H.; Feng, X. Quantum Circuits of AES with a Low-depth Linear Layer and a New Structure. *Cryptol. ePrint Arch.* **2024**.
19. Steiger, D.S.; Häner, T.; Troyer, M. ProjectQ: An open source software framework for quantum computing. *Quantum* **2018**, *2*, 49. [\[CrossRef\]](#)
20. Rollmeister. veriumMiner: Further Aarch64/Armv8 Optimizations. 2023. Available online: <https://github.com/rollmeister/veriumMiner> (accessed on 31 July 2024).
21. Novators. EnScript: EnScript, a Script Based Password Hashing Library. 2023. Available online: <https://github.com/Novators/EnScript> (accessed on 31 July 2024).
22. Daninet. Hash-Wasm: Lightning Fast Hash Functions Using Hand-Tuned WebAssembly Binaries. 2023. Available online: <https://github.com/Daninet/hash-wasm> (accessed on 31 July 2024).
23. Watkins, D. Script mining with ASICS. *arXiv* **2022**, arXiv:2208.02160.
24. National Institute of Standards and Technology. *Module-Lattice-Based Key-Encapsulation Mechanism Standard*; Federal Information Processing Standards Publication 203; National Institute of Standards and Technology: Gaithersburg, MD, USA, 2023.
25. National Institute of Standards and Technology. *Module-Lattice-Based Digital Signature Standard*; Federal Information Processing Standards Publication 204; National Institute of Standards and Technology: Gaithersburg, MD, USA, 2023.
26. National Institute of Standards and Technology. *Stateless Hash-Based Digital Signature Standard*; Federal Information Processing Standards Publication 205; National Institute of Standards and Technology: Gaithersburg, MD, USA, 2023.
27. Nielsen, M.A.; Chuang, I.L. *Quantum Computation and Quantum Information*; Cambridge University Press: Cambridge, UK, 2010.
28. Bernstein, D.J.; Lange, T. Post-quantum cryptography. *Nature* **2017**, *549*, 188–194. [\[CrossRef\]](#) [\[PubMed\]](#)
29. Percival, C. Stronger Key Derivation via Sequential Memory-Hard Functions. 2009. Available online: https://www.bsdcn.org/2009/schedule/attachments/87_scrypt.pdf (accessed on 10 August 2024.).
30. Bernstein, D.J. The Salsa20 family of stream ciphers. In *New Stream Cipher Designs: The eSTREAM Finalists*; Springer: Berlin/Heidelberg, Germany, 2008; pp. 84–97.
31. 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.

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.