Grover on SPEEDY

Gyeongju Song[1], Kyungbae Jang[1], Hyunjun Kim[1],
Siwoo Eum[1], Minjoo Sim[1], Hyunji Kim[1],
Wai-Kong Lee[2], and Hwajeong Seo[1][0000−0003−0069−9061]

[1]IT Department, Hansung University, Seoul (02876), South Korea,
{thdrudwn98, starj1023, khj930704,
shuraatum, minjoos9797, khj1594012,
hwajeong84}@gmail.com
[2]Department of Computer Engineering,
Gachon University, Seongnam, Incheon (13120), Korea,
waikonglee@gachon.ac.kr

Abstract. With the advent of quantum computers, revisiting the security of cryptography has been an active research area in recent years. In this paper, we estimate the cost of applying Grover's algorithm to SPEEDY block cipher. SPEEDY is a family of ultra-low-latency block ciphers presented in CHES'21. It is ensured that the key search equipped with Grover's algorithm reduces the n-bit security of the block cipher to $\frac{n}{2}$-bit. The issue is how many quantum resources are required for Grover's algorithm to work. NIST estimates the post-quantum security strength for symmetric key cryptography as the cost of Grover key search algorithm. SPEEDY provides 128-bit security or 192-bit security depending on the number of rounds. Based on our estimated cost, we present that increasing the number of rounds is insufficient to satisfy the security against attacks on quantum computers. To the best of our knowledge, this is the first implementation of SPEEDY as a quantum circuit.

Keywords: Grover's Search Algorithm · Quantum Computing · SPEEDY.

1 Introduction

Advances in quantum computers pose a threat to various public key cryptography and block ciphers. Grover's search algorithm is a well-known quantum algorithm that can accelerate exhaustive key search against symmetric key cryptography [1]. This quantum algorithm can reduce the computational complexity from $O(N)$ to $\sqrt{N}$ for symmetric key cryptography using an n-bit key (i.e. $N = 2^n$). Classic computers require 2^n queries for exhaustive key search to recover an n-bit key. Since qubits in quantum computers have states of 0 and 1 at the same time, using the Grover's search algorithm, a key recovery is possible with $\sqrt{2^n}$ queries. NIST presented a security strength estimation for symmetric key cryptography for the post-quantum era [2]. Block ciphers have not been verified for the safety with respect to the Grover search algorithm. In order to evaluate the safety of the target cipher even after the development of a quantum computer, the target block cipher must be implemented as a quantum circuit. NIST presented the cost of key search using Grover's algorithm as an indicator of security strength for the post-quantum era. According to this research motivation, it is an interesting research area to estimate the cost of Grover key search for symmetric key cryptography [3–12]. SPEEDY proposed in CHES'21 is a block cipher that is working with different block sizes, key lengths, and number of rounds [13]. In this paper, the SPEEDY block cipher is implemented as a quantum circuit by optimizing it in terms of quantum gates and the number of qubits. Then, we evaluate whether it is safe for quantum computers by estimating the resources to apply to the Grover's search algorithm. Since SPEEDY targets 6-bit S-box and 64-bit CPUs, the least common multiple of 6 and 64, (i.e. 192), is used as the default block size and key length. SPEEDY for 192 bits long and r rounds is called SPEEDY-r-192. r means the number of rounds. SPEEDY provides 128-bit security when $r = 6$ and full security of 192-bit is achieved at $r = 7$ where it requires one more round. We estimate the cost of Grover key search by increasing rounds r. Estimated costs are compared with the post-quantum security strength presented by NIST. As a result of cost estimation according to various r, SPEEDY provides Level-1 (AES-128) post-quantum security. Finally, our results show that increasing the number of rounds can enhance security against classical computers, but not for quantum computers. For quantum circuit implementation and simulation, IBM's ProjectQ platform is utilized.

1.1 Contributions of this paper:

- **Optimized quantum circuit implementation for SPEEDY:** To the best of our knowledge, this is the first quantum circuit implementation of SPEEDY. We adopt an efficient ANF (Algebraic Normal Form) S-box in terms of the required quantum gates and implement a ShiftColumns and Key Schedule using logical swap. As a result, a compact quantum circuit for SPEEDY is presented.
- **Estimating the cost of Grover key search for SPEEDY:** Quantum resources for applying Grover key search to SPEEDY are estimated. For detailed analysis, the resource estimation is performed at the low level of Clifford and T gates. With this, we lay the foundation for a quantum cryptanalysis of SPEEDY.
- **Post-quantum security evaluation and analysis of SPEEDY through quantum cryptanalysis:** Based on the cost of Grover key search and the criteria presented by NIST, the post-quantum security of SPEEDY is evaluated. We discuss differences between cryptonalysis in classic and quantum computers.

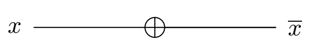
2 Related Work

2.1 Quantum background

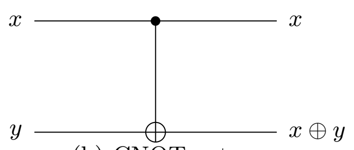
Similar to bits used in classic computer operations, quantum computers perform operations using qubits that have both 0 and 1 at the same time [14]. Due to the nature of these qubits, 2^n brute-force attack in classic computer can be performed only $\lfloor \frac{\pi}{4} 2^{\frac{n}{2}} \rfloor$ times on an quantum computer. In the quantum computing, all changes except measurements must be reversible. It is possible to return to the initial value only with the result value without additional information.

Quantum gates Quantum gates work using quantum entanglement and superposition of qubits. Quantum gates is utilizing quantum entanglement and superposition states. In the quantum computing, the state of a qubit is changed with a quantum gate that can perform reversible operations. Figure 1 shows some of the quantum gates.

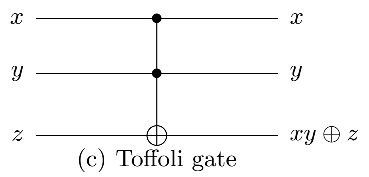
(a) **NOT/X-gate**, X(x)=$\overline{x}$: This inverts the state of a single qubit.
(b) **CNOT-gate**, CNOT(x, y)=(x, $x \oplus y$): One of the two input qubits becomes the control qubit. When the control bit is set to 1, the state of the target qubit is inverted. If the control qubit x is 1, y is inverted.
(c) **Toffoli-gate**, Toffoli(x, y, z)=(x, y, $x \cdot y \oplus z$): Two of the three input qubits become the control qubits. When all the control bits are 1, the state of the target qubit is inverted. If both control qubits (x and y) are 1, z is inverted.
(d) **SWAP-gate**, SWAP(x, y)=(y, x) : This changes the state of two input qubits.



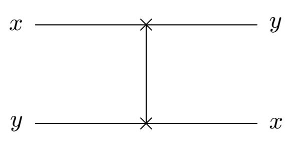
(a) NOT/X gate



(b) CNOT gate



(c) Toffoli gate



(d) SWAP gate

Fig. 1: Quantum gates.

Grover's search algorithm for key search An exhaustive key search using Grover's search algorithm can recover an n-bit key with only $2^{n/2}$ searches. First, we need to prepare key qubits in superposition state. This is achieved by applying Hadamard gates to the key qubits[1,2]. This allows an n-qubit key to have 2^n states with the equal probability. In the oracle, the key in superposition state and known plaintext are input to the encryption quantum circuit. If the generated ciphertext matches the known ciphertext, the sign of the state of that key is reversed[3]. If it is observed in this state, the solution key is returned with a probability of $1/2^n$. Grover key search repeats oracle and diffusion operator $2^{n/2}$ times and then observes the key qubits[4,5]. As a result, the solution key can be recovered with high probability. The classical key search performs 2^n searches in the worst case (i.e. $O(2^n)$), but Grover key search always repeats $2^{n/2}$.

The process of Grover's search algorithm is as follows (with Figure 2):

1. n-qubits to find the key are prepared. $|0\rangle^{\otimes n} = |0\rangle_0 \otimes |0\rangle_1 \otimes \cdots \otimes |0\rangle_n$.
2. All qubits are placed in superposition state using Hadamard gates.
3. If the ciphertext generated by the input n-qubits (i.e. key) matches the known ciphertext, the sign of the corresponding key state is inverted.
4. The amplitude of the solution key is amplified through the diffusion operator.
5. Steps 3 and 4 are repeated by $2^{n/2}$ times to increase the probability of key search.

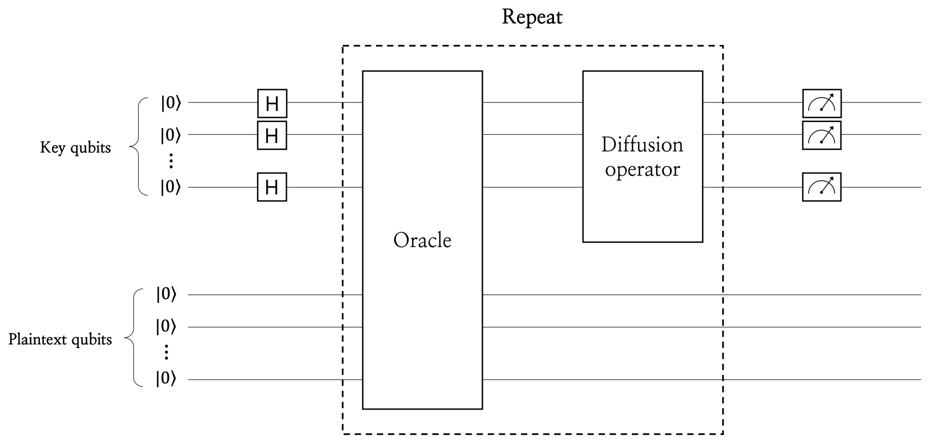


Fig. 2: Exhaustive key search using Grover's algorithm.

2.2 SPEEDY: family of block cipher

SPEEDY is a family of ultra low-latency block ciphers proposed at the CHES'21 [13]. Different block sizes and key lengths are available, and the number of rounds determines the degree of security. It is expressed as SPEEDY-r-$6l$ for the block size $6\times l$ and the number of rounds r. The internal state is represented as a $l\times 6$ array. Since SPEEDY targets 6-bit S-boxes and 64-bit high-end CPUs, it use the least common multiple of 6 and 64 (i.e. SPEEDY-r-192) as default block size and key length. Therefore, the description of SPEEDY in this paper is based on SPEEDY-r-192. In SPEEDY's round function (R), S-box (SB), ShiftColumns (SC), MixColumns (MC), AddRoundKey (A_{k_r}), AddRoundConstant (A_{c_r}) are performed, and SPEEDY-r-192 works as a 32×6 array.

S-box(SB) SPEEDY S-box is a 6-to-6-bit box with 6-bit output (y_0 to y_5) for 6-bit input (x_0 to x_5). It operates with NOT-gate and NAND-gate. Equation 1 shows the operation of S-box.

$$\begin{aligned}
y_0 &= (x_3 \wedge \overline{x}_5) \vee (x_3 \wedge x_4 \wedge x_2) \vee (\overline{x}_3 \wedge x_1 \wedge x_0) \vee (x_5 \wedge x_4 \wedge x_1)\\
y_1 &= (x_5 \wedge x_3 \wedge \overline{x}_2) \vee (\overline{x}_5 \wedge x_3 \wedge \overline{x}_4) \vee (x_5 \wedge x_2 \wedge x_0) \vee (\overline{x}_3 \wedge \overline{x}_0 \wedge x_1)\\
y_2 &= (\overline{x}_3 \wedge x_0 \wedge x_4) \vee (x_3 \wedge x_0 \wedge x_1) \vee (\overline{x}_3 \wedge \overline{x}_4 \wedge x_2) \vee (\overline{x}_0 \wedge \overline{x}_2 \wedge \overline{x}_5)\\
y_3 &= (\overline{x}_0 \wedge x_2 \wedge \overline{x}_3) \vee (x_0 \wedge x_2 \wedge x_4) \vee (x_0 \wedge \overline{x}_2 \wedge x_5) \vee (\overline{x}_0 \wedge x_3 \wedge x_1)\\
y_4 &= (x_0 \wedge \overline{x}_3) \vee (x_0 \wedge \overline{x}_4 \wedge \overline{x}_2) \vee (\overline{x}_0 \wedge x_4 \wedge x_5) \vee (\overline{x}_4 \wedge \overline{x}_2 \wedge x_1)\\
y_5 &= (x_2 \wedge x_5) \vee (\overline{x}_2 \wedge \overline{x}_1 \wedge x_4) \vee (x_2 \wedge x_1 \wedge x_0) \vee (\overline{x}_1 \wedge x_0 \wedge x_3)
\end{aligned} \tag{1}$$

ShiftColums (SC) In ShiftColumns(SC), rotates the j-th column of the state upside by j-bits. The process is shown in Equation 2.

$$y_{[i,j]} = x_{[i+j,j]} \qquad (0 \le i < l,\ 0 \le j < 6) \tag{2}$$

MixColumns(MC) MixColumns performs a CNOT operation with a shift in a column. The shift follows the order of the given constant α (α=α_0, α_1, α_2, α_3, α_4, α_5). In Equation 3 of MixColumn, i, j are rows and columns. ($0 \le i < l$, $0 \le j < 6$)

$$y_{i,j} = x_{[i,j]} \oplus x_{[i+\alpha_1,j]} \oplus x_{[i+\alpha_2,j]} \oplus x_{[i+\alpha_3,j]} \oplus x_{[i+\alpha_4,j]} \oplus x_{[i+\alpha_5,j]} \oplus x_{[i+\alpha_6,j]} \tag{3}$$

AddRoundKey(A_{k_r}) The length of the key k_r is equal to the length of the $6 \cdot l$. k_r is XORed to the whole of the state. The equation is as follows:

$$y_{i,j} = x_{[i,j]} \oplus k_{r[i,j]}, \qquad \forall i,j \tag{4}$$

AddRoundConstant(A_{c_r}) The constant c_r of $6l$-bits is XORed to the whole of the state. The round constants are chosen as the binary digits of the number $\pi - 3 = 0.1415...$ The equation is as follows:

$$y_{i,j} = x_{[i,j]} \oplus c_{r[i,j]}, \qquad \forall i,j \tag{5}$$

KeySchedule Initialize the zero-th round key k_0 initially. Then k_r is computed per round. k_r uses the permutation P to change the position of the bits. The equation is as follows:

$$\begin{aligned} &k_{r+1[i',j']} = k_{r[i,j]}, \\ &(i',j') := P(i,j) \quad with \quad (6 \cdot i' + j') \equiv (\beta \cdot (6 \cdot i + j) + \gamma)\, mod\, 6l \end{aligned} \tag{6}$$

Round function Encryption proceeds by repeating the round. For the round number r, the operation is performed in the same way from round 1 to r-1. The last round excludes MixColumn (MC) and one ShiftColumn (SC). The operation of the round function R follows Equation 7:

$$R_n = \begin{cases} A_{c_n} \circ MC \circ SC \circ SB \circ A_{k_n} & (0 < n < r-2) \\ A_{k_{n+1}} \circ SB \circ SC \circ SB \circ A_{k_n} & (n = r-1) \end{cases} \tag{7}$$

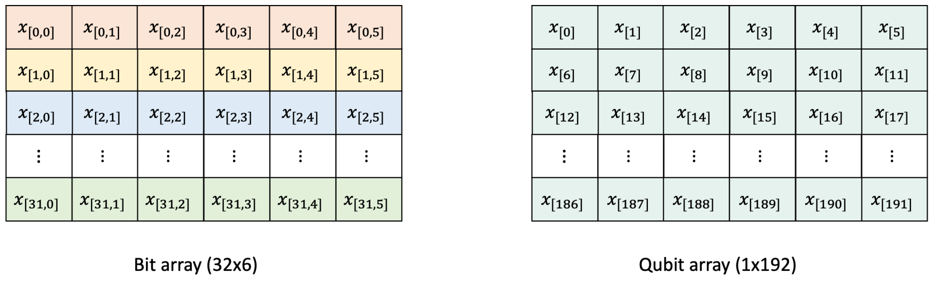
3 Quantum circuit for SPEEDY

This section describes the implementation of the quantum circuit of SPEEDY. The quantum circuit is designed based on SPEEDY-7-192. The implemented quantum circuit is used to estimate the resources required for the Grover's algorithm. As shown in Figure 3, a 32×6 array (i.e. $x_{[i][j],\, 0 \leq i < 6,\, 0 \leq j < 32}$) in a classical computer is used as a 1×192 array (i.e. $x_{[i],\, 0 \leq i < 192}$) in a quantum circuit. We describe quantum circuits for main algorithms of SPEEDY: S-box (SB), Shift-Columns, MixColumns, AddRoundKey, and AddRoundConstant. All quantum circuits are always in $modulo\, 192$ for the index. Multi-controlled X gates used in quantum circuits are represented as follows:

- **CCCX**(x_0, x_1, x_2, y_0)=(x_0, x_1, x_2, $(x_0 \cdot x_1 \cdot x_2) \oplus y_0$) **:** x_0, x_1, and x_2 are the control qubits and y_0 is the target qubits. When the control qubits are all 1, the NOT gate is used for y_0.
- **CCCCX**(x_0, x_1, x_2, x_3, y_0)=(x_0, x_1, x_2, x_3, $(x_0 \cdot x_1 \cdot x_2 \cdot x_3) \oplus y_0$) **:** x_0, x_1, x_2 and x_3 are the control qubits and y_0 is the target qubits. When the control qubits are all 1, the NOT gate is used for y_0.
- **CCCCCX**(x_0, x_1, x_2, x_3, x_4, y_0)=(x_0, x_1, x_2, x_3 $(x_0 \cdot x_1 \cdot x_2 \cdot x_3 \cdot x_4) \oplus y_0$) **:** x_0, x_1, x_2, x_3 and x_4 are the control qubits and y_0 is the target qubit. When the control qubits are all 1, the NOT gate is used for y_0.

3.1 S-box(SB)

S-box of SPEEDY uses NAND and OAI gates that are best-suited for ultra low-latency. The operation of S-box follows Equation 1 expressed in disjunctive normalform (DNF). However, DNF is inefficient to implement as a quantum circuit. For NAND and OAI operations in quantum circuits, it is necessary to



$x_{[0,0]}$	$x_{[0,1]}$	$x_{[0,2]}$	$x_{[0,3]}$	$x_{[0,4]}$	$x_{[0,5]}$
$x_{[1,0]}$	$x_{[1,1]}$	$x_{[1,2]}$	$x_{[1,3]}$	$x_{[1,4]}$	$x_{[1,5]}$
$x_{[2,0]}$	$x_{[2,1]}$	$x_{[2,2]}$	$x_{[2,3]}$	$x_{[2,4]}$	$x_{[2,5]}$
⋮	⋮	⋮	⋮	⋮	⋮
$x_{[31,0]}$	$x_{[31,1]}$	$x_{[31,2]}$	$x_{[31,3]}$	$x_{[31,4]}$	$x_{[31,5]}$

Bit array (32x6)

$x_{[0]}$	$x_{[1]}$	$x_{[2]}$	$x_{[3]}$	$x_{[4]}$	$x_{[5]}$
$x_{[6]}$	$x_{[7]}$	$x_{[8]}$	$x_{[9]}$	$x_{[10]}$	$x_{[11]}$
$x_{[12]}$	$x_{[13]}$	$x_{[14]}$	$x_{[15]}$	$x_{[16]}$	$x_{[17]}$
⋮	⋮	⋮	⋮	⋮	⋮
$x_{[186]}$	$x_{[187]}$	$x_{[188]}$	$x_{[189]}$	$x_{[190]}$	$x_{[191]}$

Qubit array (1x192)

Fig. 3: Representing bit-array as qubit-array

allocate as many qubits as the number of operations to store intermediate values. As the number of qubits increases, it is difficult for quantum computers to control errors. To resolve this issue, we use the Algebraic Normal Form (ANF) of the XOR operation suitable for quantum circuits. ANF is expressed using a combination of XOR and AND. We implement S-box as Algorithm 1 without qubits for intermediate values using CNOT and multi-controlled X gates. The equation of the S-box expressed as ANF can be found in details at [13]. In Algorithm 1, the result of input $x_0, \cdots, x_5$ is stored in $y_0, \cdots, y_5$. The S-box of the quantum circuit is characterized by high gate cost because it uses a lot of multi-controlled X gates.

3.2 ShiftColumns (SC)

ShiftColumns in quantum circuit performs column shifts. For 192 qubits, they are arranged in 6 columns of 32 qubits. In Figure 4, qubits are arranged like an array. Each column of qubits is rotated by $\delta = 0, 1, 2, 3, 4, 5$ in order. We used logical swap to rotate the columns. In quantum circuits, swap-gate only changes the position of the qubit. There is no need for gate cost.

3.3 MixColumns (MC)

MixColumns in quantum circuit performs XOR operation while shifting the index of the qubits. The proposed MixColumn operates with Algorithm 3. In the input, x_k is the qubit to be encrypted and $temp_k$ is the temporary storage qubit. First, we store x in $temp_k$ via the CNOT gate. CNOT operation of temp and x is performed while shifting the index of *temp*. Finally, the result of the operation is stored in x. Since the CNOT result is stored in x, there is no need for qubits to store the result. The standard of shift follows the order of α (α=1, 5, 9, 15, 21, 26).

3.4 AddRoundKey (A_{k_r})

The AddRoundKey (A_{k_r}) of the quantum circuit is assigned a qubit k (i.e. key) with a length equal to the length of the input. The input qubit x is XORed

Algorithm 1 Quantum circuit of S-box(SB).

```
Input: $x_0, x_1, x_2, x_3, x_4, x_5$
Output: $y_0, y_1, y_2, y_3, y_4, y_5$

 1: $y_0 \leftarrow$ CNOT($x_3$, $y_0$)
 2:     Toffoli($x_5$, $x_3$, $y_0$)
 3:     CCCCX($x_5$, $x_4$, $x_3$, $x_2$, $y_0$)
 4:     CCCX($x_5$, $x_4$, $x_1$, $y_0$)
 5:     CCCCCX($x_5$, $x_4$, $x_3$, $x_2$, $x_1$, $y_0$)
 6:     Toffoli($x_1$, $x_0$, $y_0$)
 7:     CCCCX($x_5$, $x_4$, $x_1$, $x_0$, $y_0$)
 8:     CCCX($x_3$, $x_1$, $x_0$, $y_0$)
 9:     CCCCCX($x_5$, $x_4$, $x_3$, $x_1$, $x_0$, $y_0$)

10: $y_1 \leftarrow$ CNOT($x_3$, $y_1$)
11:     Toffoli($x_4$, $x_3$, $y_1$)
12:     CCCX($x_5$, $x_4$, $x_3$, $y_1$)
13:     CCCX($x_5$, $x_3$, $x_2$, $y_1$)
14:     CNOT($x_1$, $y_1$)
15:     Toffoli($x_3$, $x_1$, $y_1$)
16:     CCCX($x_5$, $x_2$, $x_0$, $y_1$)
17:     Toffoli($x_1$, $x_0$, $y_1$)
18:     CCCX($x_3$, $x_1$, $x_0$, $y_1$)

19: $y_2 \leftarrow$ NOT($y_2$)
20:     CNOT($x_5$, $y_2$)
21:     Toffoli($x_5$, $x_2$, $y_2$)
22:     Toffoli($x_4$, $x_2$, $y_2$)
23:     Toffoli($x_3$, $x_2$, $y_2$)
24:     CCCX($x_4$, $x_3$, $x_2$, $y_2$)
25:     CNOT($x_0$, $y_2$)
26:     Toffoli($x_5$, $x_0$, $y_2$)
27:     Toffoli($x_4$, $x_0$, $y_2$)
28:     CCCX($x_4$, $x_3$, $x_0$, $y_2$)
29:     Toffoli($x_2$, $x_0$, $y_2$)
30:     CCCX($x_5$, $x_2$, $x_0$, $y_2$)
31:     CCCX($x_3$, $x_1$, $x_0$, $y_2$)

32: $y_3 \leftarrow$ CNOT($x_2$,$y_3$)
33:     Toffoli($x_3$, $x_2$, $y_3$)
34:     Toffoli($x_3$, $x_1$, $y_3$)
35:     Toffoli($x_5$, $x_0$, $y_3$)
36:     Toffoli($x_2$, $x_0$, $y_3$)
37:     CCCX($x_5$, $x_2$, $x_0$, $y_3$)
38:     CCCX($x_4$, $x_2$, $x_0$, $y_3$)
39:     CCCX($x_3$, $x_2$, $x_0$, $y_3$)
40:     CCCX($x_3$, $x_1$, $x_0$, $y_3$)

41: $y_4 \leftarrow$ Toffoli($x_5$, $x_4$, $y_4$)
42:     CNOT($x_1$, $y_4$)
43:     Toffoli($x_4$, $x_1$, $y_4$)
44:     Toffoli($x_2$, $x_1$, $y_4$)
45:     CCCX($x_4$, $x_2$, $x_1$, $y_4$)
46:     CNOT($x_0$, $y_4$)
47:     CCCX($x_5$, $x_4$, $x_0$, $y_4$)
48:     CCCX($x_4$, $x_3$, $x_0$, $y_4$)
49:     CCCX($x_3$, $x_2$, $x_0$, $y_4$)
50:     CCCCX($x_4$, $x_3$, $x_2$, $x_0$, $y_4$)
51:     Toffoli($x_1$, $x_0$, $y_4$)
52:     CCCX($x_4$, $x_1$, $x_0$, $y_4$)
53:     CCCX($x_2$, $x_1$, $x_0$, $y_4$)
54:     CCCCX($x_4$, $x_2$, $x_1$, $x_0$, $y_4$)

55: $y_5 \leftarrow$ CNOT($x_4$, $y_5$)
56:     Toffoli($x_5$, $x_2$, $y_5$)
57:     Toffoli($x_4$, $x_2$, $y_5$)
58:     Toffoli($x_4$, $x_1$, $y_5$)
59:     CCCX($x_4$, $x_2$, $x_1$, $y_5$)
60:     Toffoli($x_3$, $x_0$, $y_5$)
61:     CCCX($x_4$, $x_3$, $x_0$, $y_5$)
62:     CCCCX($x_5$, $x_3$, $x_2$, $x_0$, $y_5$)
63:     CCCCX($x_4$, $x_3$, $x_2$, $x_0$, $y_5$)
64:     CCCX($x_3$, $x_1$, $x_0$, $y_5$)
65:     CCCCX($x_4$, $x_3$, $x_1$, $x_0$, $y_5$)
66:     CCCX($x_2$, $x_1$, $x_0$, $y_5$)
67:     CCCCX($x_5$, $x_2$, $x_1$, $x_0$, $y_5$)
68:     CCCCCX($x_5$, $x_3$, $x_2$, $x_1$, $x_0$, $y_5$)
69:     CCCCCX($x_4$, $x_3$, $x_2$, $x_1$, $x_0$, $y_5$)
70: return $y_0, \cdots y_5$
```

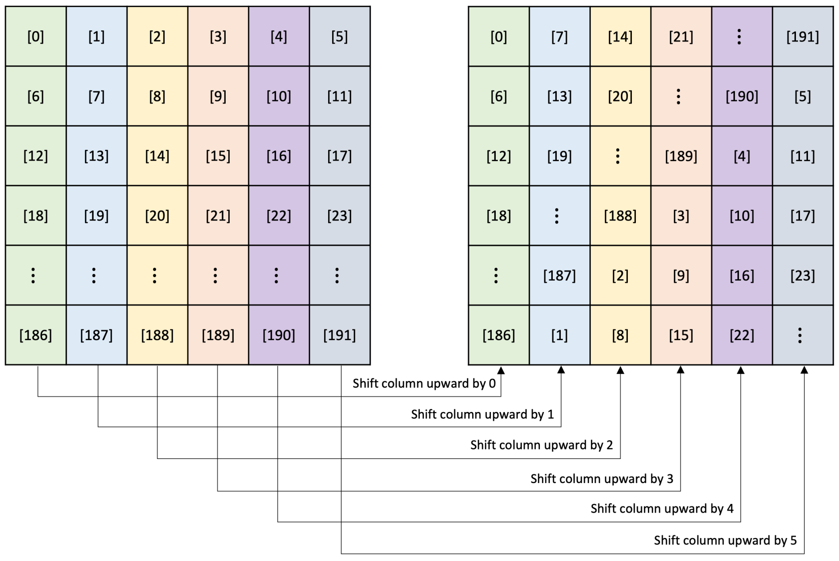


Fig. 4: ShiftColumns (SC) operation process

Algorithm 2 Quantum circuit of ShiftColumn (SC).

```
Input: 192-qubit array = $[x_0, x_1, \cdots, x_{192}]$
Output: 192-qubit array = $[x_0, x_7, x_{14}, \cdots, x_{29}]$
1: $new\_array = [\ ]$
2: for $i = 0$ to 31 do
3:     for $j = 0$ to 5 do
4:         $new\_array\ [6 \cdot i + j] \leftarrow x_{(6\cdot(i+j)+j)}$
5:     end for
6: end for
7: return $new\_array$
```

with the k of the same index. We performed the XOR operation according to Equation 4.

3.5 AddRoundConstant (A_{c_r})

In AddRoundConstant, XOR the input x and the constant. Since the constant is already known, there is no need to allocate qubits for it. We don't use CNOT-gate, we search where the index of a constant is 1, and we use X-gate for x of the same index. An X-gate operating with a single qubit has a lower cost of gate than a CNOT-gate operating with two qubits. The cost of the gate can be saved. Since X-gate is used by finding 1 in the index, only X-gate as much as Hamming weight is used. Algorithm 4 shows the operation of AddRoundConstant.

Algorithm 3 Quantum circuit of MixColumn (MC).

```
Input: $x_k, temp_k \quad (k = 0, ..., 191)$
Output: $x_k \quad (k = 0, ..., 191)$
 1: for $i = 0$ to 191 do
 2:     $temp_i \leftarrow$ CNOT($x_i$, $temp_i$) // copy target qubit($x$) to temporary qubit($temp$).
 3: end for
 4: for $i = 0$ to 31 do
 5:     for $j = 0$ to 5 do
 6:         $x_{6\cdot i+j} \leftarrow$ CNOT($temp_{6\cdot(i+1)+j}$,$x_{6\cdot i+j}$)  // $x_{6\cdot i+j} = x_{6*i+j} \oplus x_{6*(i+1)+j}$
 7:                    $\leftarrow$ CNOT($temp_{6\cdot(i+5)+j}$,$x_{6\cdot i+j}$)  // $x_{6\cdot i+j} = line.6 \oplus x_{6\cdot(i+5)+j}$
 8:                    $\leftarrow$ CNOT($temp_{6\cdot(i+9)+j}$,$x_{6\cdot i+j}$)  // $x_{6\cdot i+j} = line.7 \oplus x_{6\cdot(i+9)+j}$
 9:                    $\leftarrow$ CNOT($temp_{6\cdot(i+15)+j}$,$x_{6\cdot i+j}$) // $x_{6\cdot i+j} = line.8 \oplus x_{6\cdot(i+15)+j}$
10:                    $\leftarrow$ CNOT($temp_{6\cdot(i+21)+j}$,$x_{6\cdot i+j}$) // $x_{6\cdot i+j} = line.9 \oplus x_{6\cdot(i+21)+j}$
11:                    $\leftarrow$ CNOT($temp_{6\cdot(i+26)+j}$,$x_{6\cdot i+j}$) // $x_{6\cdot i+j} = line.10 \oplus x_{6\cdot(i+26)+j}$
12:     end for
13: end for
14: return $x_0, \cdots, x_{191}$
```

Algorithm 4 Quantum circuit of AddRoundConstant (A_{c_r}).

```
Input: $constant$, $x_0, \cdots, x_{191}$
Output: $x_0, \cdots, x_{191}$
 1: for $i = 0$ to 191 do
 2:     if( $constant$ & $1 = 1$ )
 3:         $x_i \leftarrow$ X($x_i$)
        $constant = constant \gg 1$
 4: end for
 5: return $x_0, \cdots, x_{191}$
```

4 Evaluation

In this session, resources of Grover's algorithm are estimated for SPEEDY implemented with quantum circuits. The estimated resources are used to evaluate the security strength in quantum computer. We use estimated resources to calculate the cost and evaluate the security strength in the quantum computer. The cost is calculated as (total gates × total depth). Total gates is the sum of T and Clifford gates. We decompose the non-Clifford gate resource into T+Clifford gates [15] to obtain Total gates. Finally, we show that SPEEDY-7-192, which achieved a security strength of 192 length in the classic computer, does not satisfy security strength in the quantum computer. The security strength of block ciphers is based on the estimation of the post-quantum security strength presented by NIST [2].

4.1 Quantum circuit resource estimation

Table 1 is the quantum circuit resource estimation result for SPEEDY. The SPEEDY of quantum circuits is used in oracles. We decompose the non-Clifford gate into T+Clifford to estimate the cost of the key search. Since NOT-gate and CNOT-gate are clifford-gates, only Toffoli-gate and multi-controlled X-gate are decomposed into T+Clifford. One Toffoli-gate is decomposed into 7 T-gates and 8 clifford-gates. Multi-controlled X-gates are decomposed into $(32 \times C - 84)$ T-gates (C: number of control qubits).

Table 1: Quantum resources for the proposed SPEEDY implementation.

Cipher	r	Qubits	T gates	Clifford gates	Total gates	Depth
SPEEDY-r-192	6	4,224	424,320	83,031	507,351	859
	7		495,040	97,082	592,122	1,002
	14		990,080	195,398	1,185,478	2,011
	28		1,980,160	392,058	2,372,218	4,029

4.2 Security strength analysis for SPEEDY

We evaluate the strength of security based on the security strength categories presented by NIST [2]. Implementation by Grassl et al.[3] computes the cost as (total gate $\times$ total depth). We calculate cost in the same way. The cost of AES-128, 196, and 256, which are the security strength standards presented by NIST, is estimated to be 2^{170}, 2^{233}, and 2^{298}. The following are security strength categories presented by NIST based on AES-128, 196, and 256:

– **Level 1:** Block ciphers using 128-bit key (e.g. AES 128) require computational resources that are greater than or comparable to those required for key search.
– **Level 3:** Block ciphers using 192-bit key (e.g. AES 192) require computational resources that are greater than or comparable to those required for key search.
– **Level 5:** Block ciphers using 256-bit key (e.g. AES 256) require computational resources that are greater than or comparable to those required for key search.

Grover's search algorithm can reduce the computational complexity from $O(N)$ to $\sqrt{N}$ for symmetric key cryptography using an n-bit key (i.e. $N = 2^n$). This search algorithm increases the probability of key search by iteration of oracle and diffusion. Oracle has encryption and decryption functions. Since the quantum circuit is a reversible circuit, decryption can be performed by reverse operation (i.e. encryption resource = decryption resource). As a result, the total number of resources required by Grover's algorithm is $2\times$ Table $1 \times \lfloor \frac{\pi}{4} 2^{\frac{n}{2}} \rfloor$.

Grover's algorithm repeats oracle and diffusion. Since oracle does most of the calculations, we calculated resources excluding diffusion. The Grover's algorithm resource for SPEEDY is shown in Table 2.

Table 2: Quantum resources for Grover's key search.

Cipher	r	Gates		Total gates	Total depth	Cost	Security
		T	Clifford				
SPEEDY r-192	6	$1.27 \cdot 2^{115}$	$1.99 \cdot 2^{112}$	$1.51 \cdot 2^{115}$	$1.31 \cdot 2^{106}$	$1.97 \cdot 2^{221}$	Level 1
	7	$1.48 \cdot 2^{115}$	$1.16 \cdot 2^{113}$	$1.77 \cdot 2^{115}$	$1.53 \cdot 2^{106}$	$1.38 \cdot 2^{222}$	Level 1
	14	$1.48 \cdot 2^{116}$	$1.16 \cdot 2^{114}$	$1.77 \cdot 2^{116}$	$1.54 \cdot 2^{107}$	$1.36 \cdot 2^{224}$	Level 1
	28	$1.48 \cdot 2^{117}$	$1.17 \cdot 2^{115}$	$1.77 \cdot 2^{117}$	$1.54 \cdot 2^{108}$	$1.36 \cdot 2^{226}$	Level 1

For classic computers, SPEEDY-r-192 provides 128-bit security when $r = 6$ and 192-bit security when $r = 7$. However, from quantum computers, increasing the number of rounds does not provide higher security. SPEEDY-6-192 provides post-quantum security Level 1. However, as shown in Table 2, even if the number of rounds r is increased, the security remains at Level 1. In order to increase security from quantum computers, it is very inefficient because the number of rounds r must increase exponentially. In a nut shell, classical methods of increasing security do not apply to quantum computers. Therefore, we need to apply a countermeasure for Grover's algorithm, which increases the number of iterations exponentially by increasing the key size.

5 Conclusion

We presented a quantum circuit for the block cipher SPEEDY. Based on SPEEDY-7-192, we estimated resources required to perform a key search attack and obtained the cost required to evaluate the NIST security strength. SPEEDY-7-192 showed the security strength of level 1 (i.e. the level of AES-128). We confirmed from the security strength results that SPEEDY-7-192 provided 192-bit security on the classic computer, but not quantum computer. Also, in quantum computers, SPEEDY does not provide higher security with increasing rounds. In order for SPEEDY to be secure in quantum computers, it is necessary to increase the key size to respond to Grover's algorithm.

References

1. L. K. Grover, "A fast quantum mechanical algorithm for database search," in *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pp. 212–219, 1996.

2. NIST., "Submission requirements and evaluation criteria for the post-quantum cryptography standardization process," 2016. `https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/call-for-proposals-final-dec-2016.pdf`.
3. M. Grassl, B. Langenberg, M. Roetteler, and R. Steinwandt, "Applying Grover's algorithm to AES: quantum resource estimates," in *Post-Quantum Cryptography*, pp. 29–43, Springer, 2016.
4. B. Langenberg, H. Pham, and R. Steinwandt, "Reducing the cost of implementing the advanced encryption standard as a quantum circuit," *IEEE Transactions on Quantum Engineering*, vol. 1, pp. 1–12, 2020.
5. K. Jang, S. Choi, H. Kwon, and H. Seo, "Grover on SPECK: Quantum resource estimates.," *IACR Cryptol. ePrint Arch.*, vol. 2020, p. 640, 2020.
6. A. Chauhan and S. Sanadhya, *Quantum Resource Estimates of Grover's Key Search on ARIA*, pp. 238–258. 12 2020.
7. K. Jang, G. Song, H. Kim, H. Kwon, H. Kim, and H. Seo, "Efficient implementation of PRESENT and GIFT on quantum computers," *Applied Sciences*, vol. 11, no. 11, 2021.
8. R. Anand, A. Maitra, and S. Mukhopadhyay, "Grover on SIMON," *Quantum Information Processing*, vol. 19, p. 340, 09 2020.
9. K. Jang, G. Song, H. Kwon, S. Uhm, H. Kim, W.-K. Lee, and H. Seo, "Grover on PIPO," *Electronics*, vol. 10, no. 10, p. 1194, 2021.
10. K. B. Jang, H. J. Kim, J. H. Park, G. J. Song, and H. J. Seo, "Optimization of LEA quantum circuits to apply Grover's algorithm," *KIPS Transactions on Computer and Communication Systems*, vol. 10, no. 4, pp. 101–106, 2021.
11. K. Jang, H. Kim, S. Eum, and H. Seo, "Grover on GIFT.," *IACR Cryptol. ePrint Arch.*, vol. 2020, p. 1405, 2020.
12. G. Song, K. Jang, H. Kim, W.-K. Lee, Z. Hu, and H. Seo, "Grover on SM3,"
13. G. Leander, T. Moos, A. Moradi, and S. Rasoolzadeh, "The speedy family of block ciphers: Engineering an ultra low-latency cipher from gate level for secure processor architectures," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2021, p. 510–545, Aug. 2021.
14. A. Steane, "Quantum computing," *Reports on Progress in Physics*, vol. 61, no. 2, p. 117, 1998.
15. M. Amy, D. Maslov, M. Mosca, and M. Roetteler, "A meet-in-the-middle algorithm for fast synthesis of depth-optimal quantum circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 32, p. 818–830, Jun 2013.