Quantum Circuit Implementation and Grover's Search on the Lightweight Block Cipher KLEIN Family

Indranil Mukherjee[†], Ranit Dutta[†], Bhupendra Singh[*],
Lexy Alexandar[*], Bimal Mandal[†]

[†]Department of Mathematics, Indian Institute of Technology Jodhpur,
Karwar–342030, India;
{p25ma0005, p22ma207, bimalmandal}@iitj.ac.in

[*]Centre for Artificial Intelligence & Robotics, DRDO, Bangalore-560093;
{bhusinghdrdo, lexyalexander}@gmail.com

June 16, 2026

Abstract

The continued advancement of quantum computing is expected to transform the landscape of cryptographic security, making many classical algorithms vulnerable to quantum attacks such as Grover's exhaustive key search. In this study, we present an efficient quantum circuit implementation of the lightweight block cipher KLEIN for all variants. Each functional component of the cipher, such as key addition, substitution, RotateNibbles, MixNibbles, and key scheduling, is implemented. The complete quantum design involves gates such as CCNOT, CNOT, and Pauli-X. Furthermore, we provide a comprehensive resource estimate for executing Grover's search algorithm on the proposed quantum circuits, highlighting their resilience and practicality in post-quantum cryptographic contexts.

Keywords: Lightweight block cipher, KLEIN, Quantum circuit, Quantum cost, Grover's search algorithm

1 Introduction

The development of quantum computers is now revolutionizing the field of computation by enabling efficient solutions to certain computationally difficult problems. Recent studies have found that symmetric-key cryptographic algorithms are generally considered more resistant to quantum attacks than public-key cryptographic algorithms and have sufficient security levels to resist quantum attacks. Thus, designing quantum circuits for symmetric key cryptographic algorithms is an emerging field of research. Lightweight cryptography is an important field for providing security to devices that are resource-constrained. Recently, designing quantum circuits for lightweight ciphers has attracted serious consideration from researchers. Regarding symmetric-key cryptography, the effect of the existence of quantum computers and the application of Grover's algorithm [5] decreases the security strength of a secret key of n-bit to $\mathcal{O}(2^{n/2})$. The

application of Grover's search algorithm to block ciphers is a common method to calculate the security against a quantum attack. The optimal use of quantum resources in a target algorithm is one of the most critical challenges due to the infancy of quantum computing. Grassl et al. [6] estimated the quantum resources for the application of Grover's search to the AES block cipher. Later, more optimal substitute layers for AES quantum circuits were proposed by Langenberg et al. and Jaques et al. in [2]. More recently, studies on lightweight block ciphers have also been carried out. Anand et al. in [9] estimated the quantum resources for the SIMON cipher, and Jang et al. in [12] estimated the quantum resources for the SPECK cipher. The quantum resource requirements for Korean block ciphers, namely the HIGHT, LEA, and CHAM block ciphers, were given in [12]. Also, recently quantum implementation on DEFAULT cipher has been done [7]. Gong et al. introduced a new family of lightweight ciphers, termed KLEIN ciphers, by [1]. This is in view of the low hardware cost requirement. Hence, in this paper, a quantum circuit design for the KLEIN lightweight cipher is proposed, and the design is based on the Toffoli family of quantum gates. Additionally, the application of Grover's search algorithm to the proposed quantum circuit is carried out, and the quantum resource requirements for exhaustive search of the keys are estimated.

1.1 *Contributions and organization:*

In this paper, we develop a quantum circuit realization of the lightweight block cipher KLEIN. We present a detailed resource analysis of each component of the cipher, and a complete quantum architecture is constructed using gates from the Toffoli family. The work presented in this paper is mainly divided into the following two parts.

- Quantum implementation of the KLEIN block cipher: To the best of our knowledge, this work presents the first quantum circuit implementation of the KLEIN lightweight block cipher. It also provides an initial estimate for the quantum resources required to perform an exhaustive key search on KLEIN using the Grover's algorithm.

- In-depth analysis of quantum resource estimation for lightweight block cipher KLEIN: A comprehensive quantum resource analysis of the proposed KLEIN implementation is conducted, including detailed counts of qubits, Toffoli gates, CNOT gates, Pauli-X gates, and overall circuit depth. This enables a precise understanding of the quantum cost of implementing KLEIN and applying Grover's search algorithm in a quantum setting to know about post quantum security components of this cipher. We implemented the proposed circuits using *ProjectQ*, and the implementation is available in the following repository. `https://github.com/Indranil-0807/klein_implementation.git`

The rest of the paper is organized as follows. In section 2, we describe the KLEIN ciphers, the fundamentals of quantum gates, and Grover's search algorithm. In section 3, we discuss the quantum implementation of each operation of KLEIN cipher using basic reversible quantum logic gates. In section 4, we estimate the quantum cost for the KLEIN family. In section 5, we apply Grover's quantum search algorithm to perform an exhaustive key search over the cipher's key space. Section 6 concludes the paper.

2 Preliminaries

Let $\mathbb{F}_2 = \{0, 1\}$ denote the binary field, where addition and multiplication are performed modulo 2. For a positive integer n, the set $\mathbb{F}_2^n$ denotes the n-dimensional vector space over $\mathbb{F}_2$. Its elements are binary strings

$$\mathbf{x} = (x_{n-1}, x_{n-2}, \ldots, x_1, x_0), \qquad x_i \in \mathbb{F}_2.$$

Throughout the paper, bold symbols such as $\mathbf{x}, \mathbf{y}, \mathbf{a}, \mathbf{b}$ denote binary vectors, while non-bold symbols such as x_i denote individual bits. When a binary vector is interpreted as an integer, we use the convention

$$\mathrm{int}(\mathbf{x}) = \sum_{i=0}^{n-1} x_i 2^i.$$

Thus, for a nibble $\mathbf{x} = (x_3, x_2, x_1, x_0) \in \mathbb{F}_2^4$, the corresponding integer value is

$$\mathrm{int}(\mathbf{x}) = 8x_3 + 4x_2 + 2x_1 + x_0.$$

This convention is used in the S-box tables, where the entries $0, 1, \ldots, 15$ represent the corresponding elements of $\mathbb{F}_2^4$.

For quantum states, the computational basis state corresponding to $\mathbf{x} \in \mathbb{F}_2^n$ is denoted by $|\mathbf{x}\rangle$. Equivalently, if $\mathrm{int}(\mathbf{x}) = j$, then $|\mathbf{x}\rangle$ may also be written as $|j\rangle$.

In the MixNibbles operation, two consecutive nibbles are concatenated to form one byte. We write

$$X_j = c_{2j} \| c_{2j+1} \in \mathbb{F}_2^8,$$

where $\|$ denotes concatenation. Each byte is identified with an element of the finite field $\mathbb{F}_{2^8}$. Following the Rijndael/AES convention, we represent a byte

$$X = (x_7, x_6, \ldots, x_0)$$

as the polynomial

$$X(z) = x_7 z^7 + x_6 z^6 + \cdots + x_1 z + x_0$$

over $\mathbb{F}_2$, with multiplication performed modulo the irreducible polynomial

$$m(z) = z^8 + z^4 + z^3 + z + 1.$$

Therefore, the constants $01, 02, 03, 09, 0B, 0D, 0E$ appearing in the MixNibbles matrix are interpreted as elements of $\mathbb{F}_{2^8}$.

The KLEIN family of lightweight block ciphers follows an SPN structure and consists of three variants, including KLEIN-64/64, KLEIN-64/80 and KLEIN-64/96. KLEIN-64/64 uses a 64-bit plaintext, 64-bit initial key and consists of 12 rounds. KLEIN-64/80 uses a 64-bit plaintext, 80-bit initial key and consists of 16 rounds and KLEIN-64/96 uses a 64-bit plaintext, 96-bit initial key and consists of 20 rounds. Below, we provide a detailed description of the round functions employed in the KLEIN cipher.

AddRoundKey step: The round key is XORed with the current state:

$$\text{State} \longleftarrow \text{State} \oplus \text{RoundKey}$$

$\mathbf{x}$	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
S($\mathbf{x}$)	7	4	A	9	1	F	B	0	C	3	2	6	8	E	D	5

SubNibbles step: An S-box is applied to each 4-bit nibble of the state. The S-box is a non-linear substitution that maps a 4-bit input to a 4-bit output. The S-box used in KLEIN is designed to provide high non-linearity and resistance to linear and differential cryptanalysis. It is a 4×4 involution permutation $S : \mathbb{F}_2^4 \longrightarrow \mathbb{F}_2^4$ defined as follows:

Since the SubNibbles step is the only non-linear layer in KLEIN, an optimal resistance against linear and differential cryptanalyses is required. Therefore, the choice of the S-box (S) fulfills the following conditions:

- The S-box satisfies $S(S(\mathbf{x})) = \mathbf{x}$, for all $\mathbf{x} \in \mathbb{F}_2^4$. Thus, it can be used both in encryption and decryption processes.
- The S-box has no fixed points, i.e., $S(\mathbf{x}) \neq \mathbf{x}$, for all $\mathbf{x} \in \mathbb{F}_2^4$.
- The KLEIN S-box has optimal algebraic degree 3.

RotateNibbles: After substitution, the nibbles are rotated to introduce diffusion in the cipher. This step rotates the 16 nibbles of the state by two bytes to the left in each round. The rotation ensures that the changes in the state spread out over multiple rounds. Mathematically, in the ith round, where $i \in \{1, 2, \ldots, 12\}$, nibbles of the state are denoted as $(b_0^i, b_1^i, \ldots, b_{15}^i)$, the rotation can be expressed as follows (Figure 1).

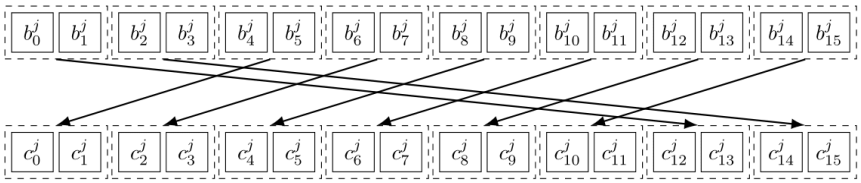


Figure 1: The RotateNibbles step of KLEIN [1]

MixNibbles: Let $(c_0^i, c_1^i, \ldots, c_{15}^i)$ be the 16 nibbles at the input of the MixNibbles layer in round i, where each $c_j^i \in \mathbb{F}_2^4$. We group every two consecutive nibbles into one byte:

$$X_j^i = c_{2j}^i \| c_{2j+1}^i \in \mathbb{F}_{2^8}, \qquad 0 \leq j \leq 7.$$

The eight bytes are then divided into two four-byte columns. Each column is multiplied by the KLEIN MDS matrix over $\mathbb{F}_{2^8}$.

$$\begin{bmatrix} Y_0^{i+1} \\ Y_1^{i+1} \\ Y_2^{i+1} \\ Y_3^{i+1} \end{bmatrix} = \begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix} \begin{bmatrix} X_0^i \\ X_1^i \\ X_2^i \\ X_3^i \end{bmatrix},$$

and

$$\begin{bmatrix} Y_4^{i+1} \\ Y_5^{i+1} \\ Y_6^{i+1} \\ Y_7^{i+1} \end{bmatrix} = \begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix} \begin{bmatrix} X_4^i \\ X_5^i \\ X_6^i \\ X_7^i \end{bmatrix}.$$

Here

$$Y_j^{i+1} = s_{2j}^{i+1} \| s_{2j+1}^{i+1}$$

denotes the output byte of the MixNibbles layer. This operation ensures that the nibbles in the state are mixed for increased diffusion. These four steps, AddRoundKey, SubNibbles, RotateNibbles and MixNibbles, are repeated 12 times in the KLEIN-64/64 cipher, followed by a final AddRoundKey operation, to ensure both confusion and diffusion, which are critical for cryptographic security.

Key schedule: The key schedule for KLEIN-64/64 is a simple linear transformation applied to the 64-bit key, producing round keys for each round. The round key for each round r is derived from the original key using rotation and XOR operations to ensure that different keys are used in each round . The key schedule of KLEIN-64/64 is designed as follows:

- Input: A 64-bit master key, mk, for KLEIN-64/64.
- Key scheduling: Let i be the round counter of KLEIN-64/64. In the first round so that $i = 1$, the initial subkey $sk^1 = mk = sk_0^1 \| sk_1^1 \| \cdots \| sk_7^1$. For KLEIN-64/64, the $(i+1)$th subkey sk^{i+1} can be derived from the ith subkey sk^i as follows.
  - Divide the ith subkey sk^i into two byte-oriented tuples, such that

$$\mathbf{a} = (sk_0^i, sk_1^i, sk_2^i, sk_3^i)$$
$$\mathbf{b} = (sk_4^i, sk_5^i, sk_6^i, sk_7^i).$$

  - Cycling left shift one byte position in (a, b), obtain

$$\mathbf{a}' = (sk_1^i, sk_2^i, sk_3^i, sk_0^i)$$
$$\mathbf{b}' = (sk_5^i, sk_6^i, sk_7^i, sk_4^i).$$

  - Swap the tuple $(\mathbf{a}', \mathbf{b}')$ with a Feistel-like structure such that $\mathbf{a}'' = \mathbf{b}'$ becomes the left tuple, whilst $\mathbf{b}'' = \mathbf{a}' \oplus \mathbf{b}'$ becomes the right tuple.
  - XOR round counter i with the third byte in the left tuple $\mathbf{a}''$, and substitute the second and third bytes of the right tuple $\mathbf{b}''$ using the KLEIN S-box.
- Output: Iteratively execute the above steps and get the subkey sk^i for the ith round transformation.

Now we present the key schedule algorithm of KLEIN-64/64. To save memory for storing intermediate values, the KLEIN subkeys can be generated during each round transformation. Here, the Feistel-like structure provides more complexities to resist weak key attacks, which have been found in some block cipher recently.

Algorithm 1 KeySchedule Procedure

Input: Master key mk
Output: Round keys $sk^1, sk^2, \ldots, sk^{13}$
$sk^1 \leftarrow mk$
for $r = 2$ to $12 + 1$ **do**
    for $i = 0$ to 3 **do**
        $sk_i^r \leftarrow sk_{(i+1) \bmod 3+3}^{r-1}$
        $sk_{i+4}^r \leftarrow sk_{(i+1) \bmod 4+3}^{r-1} \oplus sk_{(i+1) \bmod 4}^{r-1}$
    end for
    $sk_2^r \leftarrow sk_2^r \oplus (r-1)$
    $sk_5^r \leftarrow S[sk_5^r]$
    $sk_6^r \leftarrow S[sk_6^r]$
end for

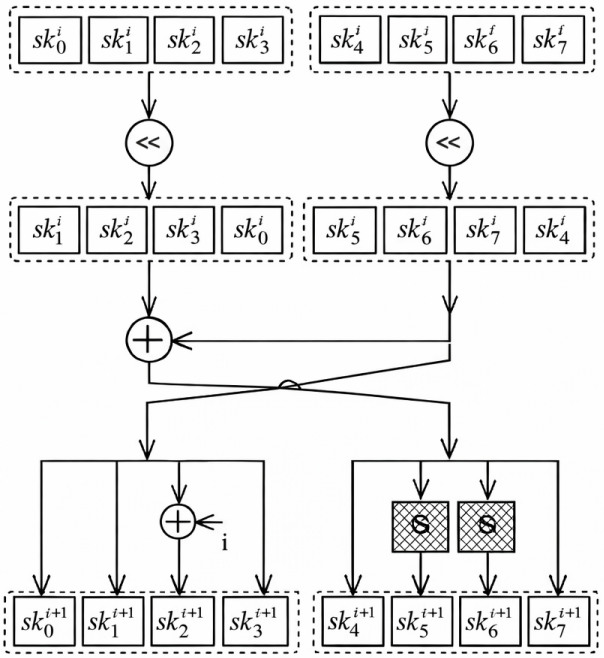


Figure 2: Key schedule algorithm of KLEIN-64/64 [1]

2.1 Fundamental quantum gates

Quantum gates are the building blocks of quantum circuits, analogous to classical logic gates. Unlike classical gates, quantum gates are reversible and are represented by unitary matrices.

- NOT gate (X gate): The quantum NOT gate flips the state of a single qubit. It is equivalent to the classical NOT gate and is represented by the Pauli-X matrix:

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

$$X|0\rangle = |1\rangle, \quad X|1\rangle = |0\rangle$$

- Hadamard gate (H gate): The Hadamard gate creates superposition. Maps the basis states as:

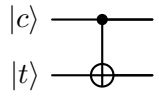
$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

$$H|0\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle), \quad H|1\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$$

- CNOT gate (Controlled-NOT): The CNOT gate flips the target qubit if the control qubit is $|1\rangle$. It operates on two qubits and is represented as:

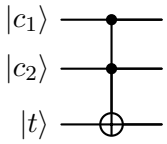
$$\text{CNOT} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

$$\begin{aligned} \text{CNOT}(|0\rangle|0\rangle) &= |0\rangle|0\rangle, \quad \text{CNOT}(|0\rangle|1\rangle) = |0\rangle|1\rangle, \\ \text{CNOT}(|1\rangle|0\rangle) &= |1\rangle|1\rangle, \quad \text{CNOT}(|1\rangle|1\rangle) = |1\rangle|0\rangle \end{aligned}$$



- Toffoli gate (CCNOT): The Toffoli gate (Controlled-Controlled-NOT) flips the target qubit if both control qubits are $|1\rangle$. It is a 3-qubit gate.

$$\text{Toffoli}(|c_1, c_2, t\rangle) = \begin{cases} |c_1, c_2, t \oplus 1\rangle, & \text{if } c_1 = c_2 = 1 \\ |c_1, c_2, t\rangle, & \text{otherwise} \end{cases}$$



2.2 Grover's search algorithm

Grover's algorithm is a quantum algorithm that finds a marked item in an unsorted database of N elements in $O(\sqrt{N})$ time. The problem statment of this algorithm is as follows: Given a function $f : \{0,1\}^n \to \{0,1\}$, where there exists a unique input $\mathbf{x}_0 \in \{0,1\}^n$ such that $f(\mathbf{x}_0) = 1$, the goal is to determine the marked element $\mathbf{x}_0$ with high probability using the Grover search algorithm.

- Steps of the Grover's algorithm

  i. Prepare an n-qubit quantum register in the state: $|0\rangle^{\otimes n}$

  ii. Apply Hadamard gates $H^{\otimes n}$ to all qubits to obtain an equal superposition:

$$|\psi\rangle = \frac{1}{\sqrt{2^n}} \sum_{x=0}^{2^n-1} |x\rangle$$

  iii. Repeat the following steps approximately $\left\lfloor \frac{\pi}{4}\sqrt{2^n} \right\rfloor$ times:

  a. Flip the sign of the amplitude corresponding to the solution:

$$O_f |\mathbf{x}\rangle = \begin{cases} -|\mathbf{x}\rangle, & \text{if } \mathbf{x} = \mathbf{x}_0 \\ |\mathbf{x}\rangle, & \text{otherwise} \end{cases}$$

  b. Reflect the state vector about the average amplitude to amplify the probability of measuring $\mathbf{x}_0$.

  iv. Measure the quantum register. With high probability, the result will be the marked item $\mathbf{x}_0$.

- Visual explanation of oracle and diffusion

  i. Oracle: Flips the sign of the amplitude of the marked state.

  ii. Diffusion operator: Reflects all amplitudes about the average to amplify the marked one.

3 Proposed quantum circuit for the KLEIN cipher

This work presents the quantum realization of the key operations in the KLEIN lightweight block cipher, namely *AddRoundKey*, *SubNibbles*, *RotateNibbles*, *MixNibbles* and the *Key Schedule*. Each operation is implemented using basic reversible quantum logic gates. Where feasible, quantum gate reuse is employed to minimize both quantum cost and circuit depth.

3.1 Quantum implementation of AddRoundKey

The *AddRoundKey* operation in the KLEIN cipher involves bitwise XOR between a 64-bit plaintext block and a selected 64-bit round key, extracted from the 64-bit master key. In the first encryption round, this XOR is performed directly between the input data and the key. In the quantum circuit, this XOR functionality is realized using

Controlled-NOT (CNOT) gates, which are the standard quantum equivalent of the classical XOR. Each CNOT gate performs the following:

$$\text{CNOT}(|\mathbf{a}\rangle\,|\mathbf{b}\rangle) = |\mathbf{a}\rangle\,|\mathbf{a}\oplus\mathbf{b}\rangle$$

To complete one round of the *AddRoundKey* operation, 64 parallel CNOT gates are required, one for each bit in the 64-bit state. The output from this step is fed directly into the *SubNibbles* transformation.

3.2 Quantum circuit for SubNibbles

Each S-layer consists of 16 parallel S-boxes and the S-box used in KLEIN is designed to provide high non-linearity and resistance to linear and differential cryptanalysis. KLEIN S-box representation given in Preliminaries section. For that, we first write the algebraic normal form (ANF) of each coordinate function of KLEIN S-box. Then we reduce the CCNOT gate by reusing ancilla qubit.

3.2.1 Circuit implementation using ANF

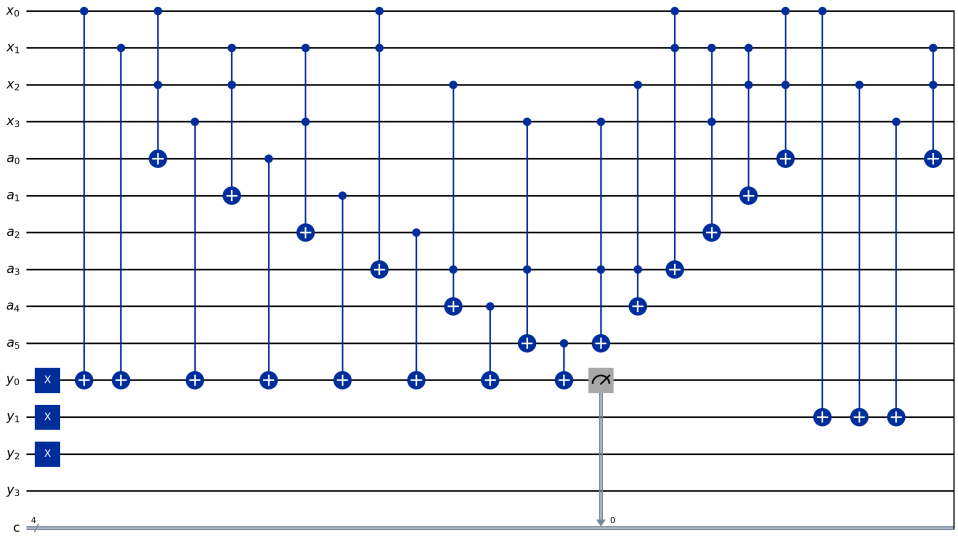
In this section, we present a circuit implementation of KLEIN S-box. The algebraic normal form (ANF) of KLEIN S-box is defined as follows. Let $\mathbf{x} = (x_3, x_2, x_1, x_0) \in \mathbb{F}_2^4$ be any 4-bit input to the KLEIN S-box S, and let $S(\mathbf{x}) = (y_3, y_2, y_1, y_0)$, where each coordinate y_i is represented using ANF as follows.

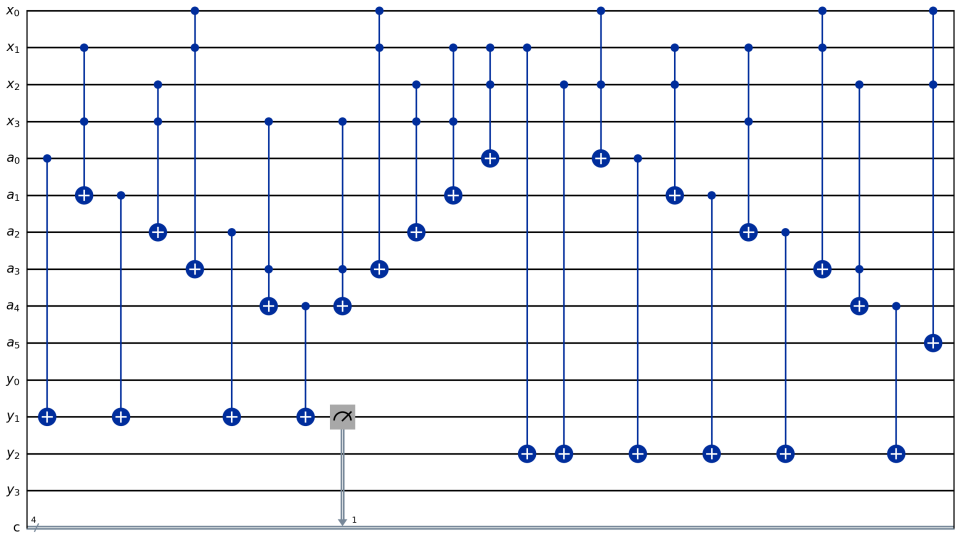
$$\begin{aligned}
y_0 &= 1 \oplus x_0 \oplus x_1 \oplus x_3 \oplus x_0x_2 \oplus x_1x_2 \oplus x_1x_3 \oplus x_0x_1x_2 \oplus x_0x_1x_3,\\
y_1 &= 1 \oplus x_0 \oplus x_2 \oplus x_3 \oplus x_1x_2 \oplus x_1x_3 \oplus x_2x_3 \oplus x_0x_1x_3,\\
y_2 &= 1 \oplus x_1 \oplus x_2 \oplus x_0x_2 \oplus x_1x_2 \oplus x_0x_3 \oplus x_0x_1x_2 \oplus x_0x_2x_3 \oplus x_1x_2x_3,\\
y_3 &= x_1 \oplus x_3 \oplus x_0x_2 \oplus x_0x_3 \oplus x_0x_1x_3 \oplus x_1x_2x_3.
\end{aligned}$$

In Figure 3, we present the quantum circuit corresponding to the algebraic normal form (ANF) representation of the round function, constructed using CCNOT, CNOT, and Pauli-X gates. Because each output bit y_i contains algebraic terms composed of three input variables, the resulting implementation requires a relatively larger number of CCNOT gates compared to other lightweight ciphers in this category.

To obtain the final output state, a measurement operation is applied to the respective output register on each wire. As shown for y_0 in Figure 3, reproducing the correct output as dictated by its ANF expression involves performing XOR operations over all intermediate values generated along that wire. The same procedure is adopted for the remaining output components y_1, y_2, and y_3, ensuring that the circuit output is consistent with the intended ANF formulation.

We have implemented the naive circuit for the KLEIN S-box by reusing the ancilla qubits and obtain the circuit diagram as shown in Figure 3.


x_0
x_1
x_2
x_3
a_0
a_1
a_2
a_3
a_4
a_5
y_0
y_1
y_2
y_3
c
X
X
X
4
0


x_0
x_1
x_2
x_3
a_0
a_1
a_2
a_3
a_4
a_5
y_0
y_1
y_2
y_3
c
4
1

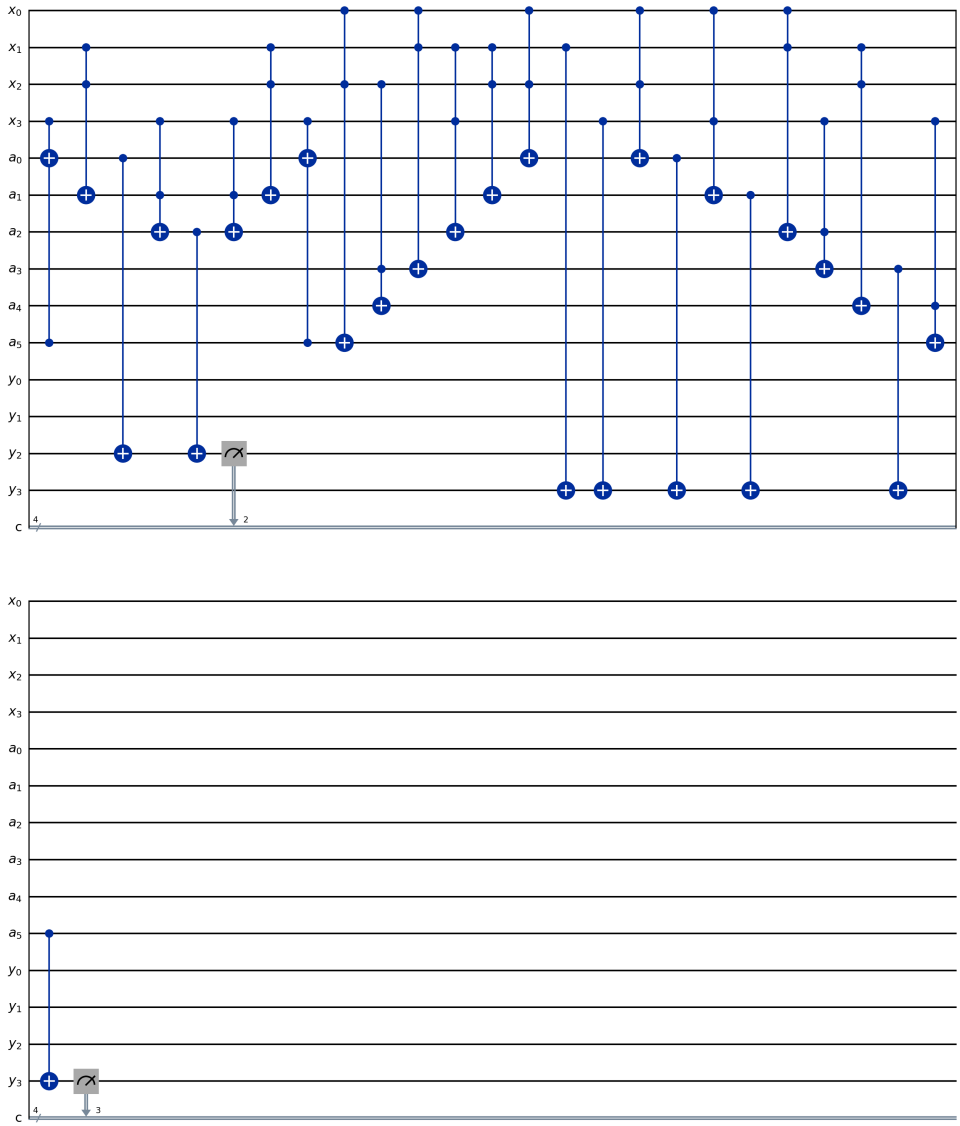


Figure 3: Circuit diagram of KLEIN S-box using ANF

We list our output data for the implementation of a single KLEIN S-box in the table 1. We count the number of X-gate, CNOT, CCNOT needed for this naive implementation.

X-gate	**CNOT**	**CCNOT**	**Ancilla qubit**
3	29	46	6

Table 1: Gate resource summary for implementing a single S-box with reduced CCNOT

3.2.2 Implementation of KLEIN S-box with Lighter-r in-place bit slice method

In this implementation, the gate count decreases significantly. Moreover, the CCNOT count is considerably reduced compared with the naive implementation described above. Table 2 gives the resource count for implementing a single KLEIN S-box. Also we have used Lighter-r with NCT-qc library. For the in-place bit-sliced implementation, this S-box is used 16 times in parallel to perform the 64-bit SubNibbles operation.

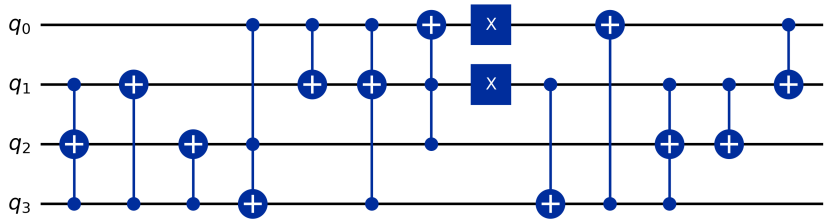


Figure 4: Lighter-r circuit for KLEIN S-box

X-gate	CNOT	CCNOT	qubit	Full depth
2	7	5	4	14

Table 2: Gate resource summary for implementing a single S-box using Lighter-r

In Table 3 we have decomposed the CCNOT gate into $7T + 8Clifford$ gates with T-depth 4 and total depth 8. This decomposition is applied for the entire circuit implementation of the CCNOT gate.

X-gate	CNOT	T	qubit	Full depth
2	37	35	4	60

Table 3: Implementing a single S-box using Lighter-r with T gate decomposition

3.2.3 T-depth and T-count

In the context of fault-tolerant quantum computation, the dominant cost metric is associated with non-Clifford operations, particularly the T gate. Clifford gates (H, CNOT) can be implemented relatively efficiently, whereas the T gate requires resource-intensive procedures such as magic-state distillation. For this reason, the complexity of a quantum circuit is commonly evaluated using two parameters: *T-count* and *T-depth*.

- T-count denotes the total number of T and $T^\dagger$ gates appearing in the Clifford+T decomposition of a circuit. It represents the total non-Clifford cost and is a direct indicator of fault-tolerant resource consumption.
- T-depth measures the number of sequential layers of T or $T^\dagger$ gates that cannot be parallelized. Even if a circuit contains a large number of non-Clifford gates,

a low T-depth indicates that many of these operations can be executed concurrently, leading to significantly reduced execution time on a fault-tolerant quantum architecture.

3.3 Quantum circuit for RotateNibbles

The RotatNibbles operation involves rearranging the nibbles produced by the S-box outputs through simple transposition. In this work, the RotateNibbles step is implemented using a series of SWAP gates, which efficiently realign the positions of the nibbles to prepare for the next round of computation. Since SWAP gates are considered to have negligible quantum cost, a detailed cost evaluation for this operation is not necessary. The result of the round function is directly derived from the output of this RotateNibbles transformation.

3.4 Quantum circuit for MixNibbles

The initial implementation models the 64-bit KLEIN state as 8 bytes, each represented using 8 qubits. These are grouped into two 4-byte columns, consistent with the matrix multiplication in the MixNibbles step. Since each 4-byte column $\begin{bmatrix}\mathbf{x}_0 & \mathbf{x}_1 & \mathbf{x}_2 & \mathbf{x}_3\end{bmatrix}^T$ is transformed using the MDS matrix

$$\begin{bmatrix}2 & 3 & 1 & 1\\ 1 & 2 & 3 & 1\\ 1 & 1 & 2 & 3\\ 3 & 1 & 1 & 2\end{bmatrix}$$

Then the output is

$$\begin{aligned}\mathbf{y}_0 &= 2\mathbf{x}_0 \oplus 3\mathbf{x}_1 \oplus \mathbf{x}_2 \oplus \mathbf{x}_3\\ \mathbf{y}_1 &= \mathbf{x}_0 \oplus 2\mathbf{x}_1 \oplus 3\mathbf{x}_2 \oplus \mathbf{x}_3\\ \mathbf{y}_2 &= \mathbf{x}_0 \oplus \mathbf{x}_1 \oplus 2\mathbf{x}_2 \oplus 3\mathbf{x}_3\\ \mathbf{y}_3 &= 3\mathbf{x}_0 \oplus \mathbf{x}_1 \oplus \mathbf{x}_2 \oplus 2\mathbf{x}_3\end{aligned}$$

Now we calculate the resources that is needed to implement the matrix multiplication. The implementation invokes multiplication subroutines on the fly for each x_i contribution, using CNOT gates:

- No ancilla is used; each multiplication is recomputed independently.
- A total of 624 CNOT gates with total depth 156 and a total 64 qubits are required for the 1 round MixNibbles implementation.

3.5 Quantum circuit for the key schedule algorithm

The KLEIN key schedule updates the round key using byte rotations, a Feistel-like XOR operation, the addition of a round constant, and S-box applications. In the quantum implementation, these operations are realized using reversible gates in a way that is consistent with the encryption circuit.

The byte rotations and swaps appearing in the key schedule only permute the positions of qubits. Hence, they can be implemented by wire relabelling or SWAP

gates. Since these operations do not contribute to the non-linear part of the circuit, we do not include them in the quantum cost calculation.

The Feistel-like update contains bytewise XOR operations. These are implemented using CNOT gates, since a CNOT gate maps

$$|a\rangle |b\rangle \longmapsto |a\rangle |a \oplus b\rangle .$$

For a bytewise XOR, eight CNOT gates are required. Therefore, the XOR part of the key schedule contributes a linear number of CNOT gates.

The round constant is a classical value. Its addition to the corresponding key byte is implemented by applying Pauli-X gates to those qubit positions where the round constant has bit value 1. Hence, this part contributes only X gates and does not require any ancilla qubits.

The non-linear part of the key schedule comes from the S-box applications. In KLEIN-64/64, the S-box is applied to the second and third bytes of the right tuple. Since each byte consists of two nibbles, this corresponds to four parallel applications of the 4-bit KLEIN S-box. Using the optimized in-place S-box implementation described in Subsection 3, each S-box requires 2 X gates, 7 CNOT gates, and 5 CCNOT gates. Thus, the four S-box applications contribute

$$4 \times 2 = 8 \quad \text{X gates}, \quad 4 \times 7 = 28 \quad \text{CNOT gates}, \quad \text{and} \quad 4 \times 5 = 20 \quad \text{CCNOT gates}.$$

Together with the XOR operations and the round-constant addition, the full key schedule circuit for KLEIN-64/64 requires the resources shown in Table 4. For the KLEIN-64/80 and KLEIN-64/96 variants, the same type of operations are used, but the larger key registers lead to additional CNOT gates in the key update. These variant-wise costs are included in Table 5.

X-gate	**CNOT**	**CCNOT**
9	60	20

Table 4: Quantum gate resource summary for one KLEIN-64/64 key schedule update

4 Quantum cost estimation

In this section, we estimate the quantum resources required for the proposed quantum implementation of the KLEIN family. The main resources considered in our analysis are the number of quantum gates, the quantum cost, and the number of qubits. The implemented circuits are constructed using gates from the Toffoli family, namely CCNOT, CNOT, and Pauli-X gates. We use the standard cost convention in which a CCNOT gate has quantum cost 5, while each CNOT and Pauli-X gate has quantum cost 1. Therefore, for a circuit containing #CCNOT CCNOT gates, #CNOT CNOT gates, and $\#X$ Pauli-X gates, the total quantum cost is computed as

$$\text{Quantum Cost} = 5 \times \#\text{CCNOT} + \#\text{CNOT} + \#X.$$

Table 5 gives the quantum resource requirements of the main building blocks of one round of KLEIN. The AddRoundKey operation is a bitwise XOR between the state

and the round key. Hence, it is implemented using 64 CNOT gates and has quantum cost 64. The SubNibbles layer consists of 16 parallel applications of the 4-bit KLEIN S-box. Since the optimized implementation in-place of one KLEIN S-box requires 5 CCNOT gates, 7 CNOT gates and 2 Pauli-X gates, the complete SubNibbles layer requires $16 \times 5 = 80$ CCNOT gates, $16 \times 7 = 112$ CNOT gates and $16 \times 2 = 32$ Pauli-X gates. Thus, the total quantum cost of the SubNibbles layer is

$$5 \times 80 + 112 + 32 = 544.$$

The RotateNibbles operation only permutes the positions of nibbles. In the quantum circuit, this operation can be implemented by wire rearrangement or SWAP gates. Under the cost convention adopted in this work, this operation is not included in the total quantum cost calculation. The MixNibbles layer is a linear transformation over $\mathbb{F}_{2^8}$. Therefore, it is implemented using only CNOT gates. As described in Section 3, one MixNibbles layer requires 624 CNOT gates and therefore has quantum cost 624.

The cost of the key schedule depends on the key size. For KLEIN-64/64, the key scheduler requires 20 CCNOT gates, 60 CNOT gates, and 9 Pauli-X gates, giving a total quantum cost of

$$5 \times 20 + 60 + 9 = 169.$$

For KLEIN-64/80 and KLEIN-64/96, the same number of CCNOT and Pauli-X gates is required, but the larger key registers lead to additional CNOT gates in the key-update process. Hence, the key scheduler costs for KLEIN-64/80 and KLEIN-64/96 are 177 and 185, respectively.

Building Blocks	**Number of Gates**			**Quantum Cost**			
	CCNOT	**CNOT**	**X**	**CCNOT**	**CNOT**	**X**	**Total**
AddRoundKey	0	64	0	0	64	0	64
SubNibbles	80	112	32	400	112	32	544
RotateNibbles	0	0	0	0	0	0	0
Key scheduler-64	20	60	9	100	60	9	169
Key scheduler-80	20	68	9	100	68	9	177
Key scheduler-96	20	76	9	100	76	9	185
MixNibbles	0	624	0	0	624	0	624

Table 5: Quantum cost estimation for the KLEIN cipher building blocks

Using the component-wise estimates from Table 5, we now estimate the total resources required for the full-round implementations of KLEIN-64/64, KLEIN-64/80, and KLEIN-64/96. These variants have 12, 16, and 20 rounds, respectively. In addition to the round operations, the final AddRoundKey operation is also included in the total CNOT count.

For KLEIN-64/64, the total number of CCNOT gates is $12 \times (80 + 20) = 1200$. The total number of CNOT gates is

$$12 \times (64 + 112 + 624 + 60) + 64 = 10384,$$

where the final 64 CNOT gates correspond to the final AddRoundKey operation. Similarly, the total number of Pauli-X gates is

$$12 \times (32 + 9) = 492.$$

The same method is used for KLEIN-64/80 and KLEIN-64/96, with the appropriate number of rounds and the corresponding key scheduler cost. The qubit count is computed from the 64-bit state register together with the key register. Thus, KLEIN-64/64, KLEIN-64/80, and KLEIN-64/96 require 128, 144, and 160 qubits, respectively.

Ciphers	CCNOT	CNOT	Pauli-X	Total Gates	Quantum Cost	Qubits
KLEIN-64/64 (proposed)	1200	10384	492	12076	16876	128
KLEIN-64/80 (proposed)	1600	13952	656	16208	22608	144
KLEIN-64/96 (proposed)	2000	17584	820	20404	28404	160

Table 6: Resource estimates for full-round KLEIN ciphers

For fault-tolerant quantum computation, it is also useful to express the resource estimates after decomposing the CCNOT gates into Clifford+T gates. In this work, each CCNOT gate is decomposed into 7 T gates together with Clifford gates. Hence, the T count is obtained by multiplying the number of CCNOT gates in Table 6 by 7. The CNOT and one-qubit Clifford counts are updated according to the chosen CCNOT decomposition. The resulting resource estimates are given in Table 7. This form of presentation gives a more realistic view of the implementation cost in a fault-tolerant quantum setting. These values are particularly important because non-Clifford resources, such as T count and T depth, are usually dominant in fault-tolerant quantum implementations.

Ciphers	T	CNOT	1qCliff	Total Gates	Total Depth	Qubits
KLEIN-64/64 (proposed)	8400	17584	2892	28876	2516	128
KLEIN-64/80 (proposed)	11200	23552	3856	38608	3352	144
KLEIN-64/96 (proposed)	14000	29584	4820	48404	4188	160

Table 7: Resource estimates for full-round KLEIN ciphers with CCNOT decomposition

Finally, we compare the proposed quantum implementation of KLEIN with existing quantum implementations of other block ciphers. Table 8 lists the number of CCNOT gates, CNOT gates, Pauli-X gates, total gates, quantum cost, and qubit requirements for several lightweight block ciphers. Since different ciphers have different block sizes, key sizes, round numbers, and internal structures, this table should be interpreted as a resource-level comparison rather than a direct security comparison. Such a comparison is useful for understanding the implementation overhead and practical feasibility of different lightweight designs in a quantum setting. Nevertheless, the table shows that the proposed quantum implementation of KLEIN has competitive resource requirements among lightweight block ciphers.

Ciphers	CCNOT	CNOT	Pauli-X	Total Gates	Quantum Cost	Qubits
GIFT-64/128 [4]	1792	1792	3261	6845	14013	192
GIFT-128/128 [4]	6144	6144	10953	23241	47817	256
CHAM-64/128 [12]	2400	12285	240	14925	24525	196
CHAM-128/128 [12]	4960	26885	240	32085	51925	268
CHAM-128/256 [12]	5952	32277	304	38533	62341	396
HIGHT-64/128 [12]	6272	20523	4	26799	51887	201
LEA-128/128 [12]	10416	28080	68	38564	80228	385
LEA-128/192 [12]	15624	39816	100	55540	118036	513
LEA-128/256 [12]	17856	45504	130	63490	134914	641
SPECK-64/128 [12]	3286	9238	57	12581	25725	193
SPECK-128/128 [12]	7942	22086	75	30103	61871	257
PRESENT-64/80 [13]	2108	4683	1118	7909	16333	144
PRESENT-64/128 [13]	2232	4838	1164	8234	17162	192
SIMON-64/128 [9]	1408	7396	1216	10020	15652	192
SIMON-128/128 [9]	4352	17152	4224	25728	43136	256

Table 8: Comparison of quantum resources with other block ciphers

5 Application of Grover's algorithm to the KLEIN cipher

This study employs Grover's quantum search algorithm to perform an exhaustive key search over the cipher's key space. Compared to classical brute-force methods, Grover's approach achieves a quadratic speedup in time complexity. The estimation of quantum cost and resource utilization following the application of Grover's algorithm is carried out based on the methodology outlined in [2, 4, 6]. The required number of qubits is computed using the expression:

$$\text{Total qubits} = r \times q + 1,$$

where $r = \frac{\text{key size}}{\text{block size}}$, and q denotes the number of qubits necessary for implementing one full instance of the KLEIN cipher.

- For KLEIN-64/64 configuration, the value of r is 1, leading to a total qubit count of $q+1$. We assume that $r_k = \lceil \frac{k}{64} \rceil$ known plaintext–ciphertext pairs are sufficient to avoid false positives in an exhaustive key search for KLEIN-64/64. Implementing Grover's oracle circuit incurs substantial quantum resource costs, primarily due to the requirement of 2 parallel instances of the cipher. Additionally, to enable efficient parallel key search, an extra $2(r-1) \times$ key size CNOT gates are incorporated. Here $r = 1$ so there do not require extra CNOT gate.

- For KLEIN-64/80 configuration, the value of r is 2, leading to a total qubit count of $2q + 1$. We assume that $r_k = \lceil \frac{k}{64} \rceil$ known plaintext–ciphertext pairs are sufficient to avoid false positives in an exhaustive key search for KLEIN-64/80. Implementing Grover's oracle circuit incurs substantial quantum resource costs, primarily due to the requirement of 4 parallel instances of the cipher. Additionally, to enable efficient parallel key search, an additional $2(r-1) \times$ key size CNOT gates are incorporated. Here $r = 2$ so we need 160 CNOT gate.

- For KLEIN-64/96 configuration, the value of r is 2, leading to a total qubit count of $2q + 1$. We assume that $r_k = \lceil \frac{k}{64} \rceil$ known plaintext–ciphertext pairs are sufficient to avoid false positives in an exhaustive key search for KLEIN-64/96. Implementing Grover's oracle circuit incurs substantial quantum resource costs, primarily due to the requirement of 4 parallel instances of the cipher. Additionally, to enable efficient parallel key search, an extra $2(r-1) \times$ key size CNOT gates are incorporated. Here $r = 2$ so we need 192 CNOT gate.

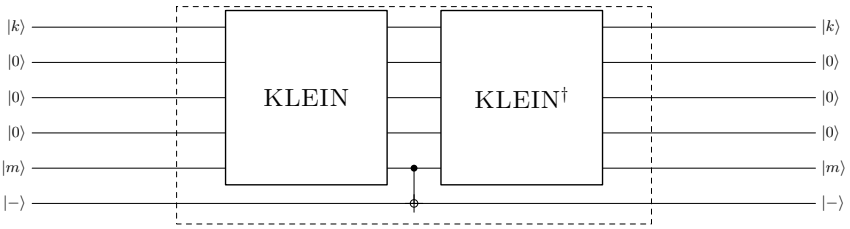


Figure 5: Grover's oracle circuit for KLEIN

A detailed summary of the total quantum resources required after integrating Grover's algorithm is provided in Table 9.

Ciphers	**CCNOT**	**CNOT**	**Pauli-X**	**Total Gates**	**Qubits**
KLEIN-64/64 (proposed)	2400	20768	984	24152	129
KLEIN-64/80 (proposed)	6400	55808	2624	64832	289
KLEIN-64/96 (proposed)	8000	70336	3280	81616	321

Table 9: Resources estimation for Grover's search on KLEIN

Ciphers	**T**	**CNOT**	**1qcliff**	**Total Gates**	**Total depth**	**Qubits**
KLEIN-64/64 (proposed)	16800	35168	5784	57752	2516	129
KLEIN-64/80 (proposed)	44800	94368	15424	154592	3352	289
KLEIN-64/96 (proposed)	56000	118528	19280	193808	4188	321

Table 10: Grover's search estimation on full round KLEIN ciphers with CCNOT decomposition

The cost of Grover's key search is calculated from the estimated oracle cost. For an n-bit search space, Grover's algorithm requires on the order of $2^{n/2}$ iterations. In [6], the authors showed that the optimal number of iterations for an n-bit search space is

$$\left\lfloor \frac{\pi}{4} \sqrt{2^n} \right\rfloor .$$

Therefore, the total cost of Grover's key search is estimated as in Table 11 by multiplying the oracle cost by

$$\left\lfloor \frac{\pi}{4}\sqrt{2^{64}} \right\rfloor, \quad \left\lfloor \frac{\pi}{4}\sqrt{2^{80}} \right\rfloor, \quad \text{and} \quad \left\lfloor \frac{\pi}{4}\sqrt{2^{96}} \right\rfloor$$

respectively for each varients, where the cost of the diffusion operator is not counted.

Ciphers	**Total gates**	**Total depth**	**Cost**
KLEIN-64/64 (proposed)	1.3842×2^{47}	1.9297×2^{42}	1.33×2^{90}
KLEIN-64/80 (proposed)	1.85×2^{56}	1.28×2^{51}	1.184×2^{108}
KLEIN-64/96 (proposed)	1.16×2^{65}	1.6060×2^{59}	1.86×2^{124}

Table 11: Grover's search estimation

NIST's post-quantum security analysis highlights that Grover search is inherently iterative. In Grover's algorithm, the same oracle, together with the diffusion operator, is applied approximately $O(2^{n/2})$ times. Therefore, the main cost arises from resources that grow with the number of iterations, such as the total gate count and the overall circuit depth. In particular, non-Clifford resources, including the T-count and T-depth obtained after decomposing Toffoli gates into Clifford+T gates, are often especially important.

By comparison, the number of qubits represents a fixed initial requirement. Once the oracle registers and the necessary workspace or ancilla qubits are allocated, the same registers are reused in every Grover iteration, so the circuit width does not grow with the search. In other words, gates and depth accumulate over repeated iterations, whereas the qubit count remains unchanged. This interpretation is consistent with NIST-style resource estimates, where time-related costs, such as gate complexity and depth, mainly determine the cost of Grover-based key search, while the qubit requirement is considered separately as an iteration-independent resource.

6 Conclusion

In this paper, we presented a quantum circuit implementation of the KLEIN lightweight block cipher family. The proposed design implements the main components of KLEIN, including AddRoundKey, SubNibbles, RotateNibbles, MixNibbles, and the Key Schedule, using reversible quantum gates. We also estimated the quantum resources required for the full-round implementations of KLEIN-64/64, KLEIN-64/80, and KLEIN-64/96. These estimates include gate counts, quantum cost, and qubit requirements, which are important parameters for evaluating the feasibility of quantum implementations.

Furthermore, we analyzed the cost of applying Grover's key search algorithm to the proposed circuits. This provides a clearer idea of the resources needed for quantum exhaustive key search against the KLEIN variants. We also discussed the Clifford+T decomposition cost, since the T count is an important metric in fault-tolerant quantum computation. To the best of our knowledge, this is the first quantum implementation and Grover-based resource estimation for the full KLEIN family. As future work, similar quantum cost analyses may be performed for other lightweight block ciphers, such as TWINE and Piccolo, and compared with the results obtained for KLEIN.

References

[1] Gong, Z., Nikova, S., Law, Y. W.: KLEIN: A New Family of Lightweight Block Ciphers. RFID: Security and Privacy 2011, LNCS 7055 1–18 (2012).

[2] Langenberg, B., Pham, H., Steinwandt, R.: Reducing the Cost of Implementing the Advanced Encryption Standard as a Quantum Circuit. IEEE Transactions on Quantum Engineering 1 1–12 (2020).

[3] Saravanan, P., Kalpana, P.: Novel Reversible Design of Advanced Encryption Standard Cryptographic Algorithms for Wireless Sensor Networks. Wireless Personal Communications 100(4) 1427–1458 (2018).

[4] Jang, K., Kim, H., Eum, S., Seo, H.: Grover on GIFT. IACR Cryptology ePrint Archive 2020/1405 (2020).

[5] Grover, L. K.: A Fast Quantum Mechanical Algorithm for Database Search. Proceedings of the 28th Annual ACM Symposium on Theory of Computing (STOC), 212–219 (1996).

[6] Grassl, M., Langenberg, B., Roetteler, M., Steinwandt, R.: Applying Grover's Algorithm to AES: Quantum Resource Estimates. PQCrypto 2016, LNSC 9606 29–43 (2016).

[7] Jang, K., Baksi, A., Breier, J., Seo, H., Chattopadhyay, A.: Quantum Implementation and Analysis of DEFAULT. Cryptography and Communications 17 359–375 (2025).

[8] Dasu, V. A., Baksi, A., Sarkar, S., Chattopadhyay, A.: LIGHTER-R: Optimized Reversible Circuit Implementation for S-boxes. IEEE International System-on-Chip Conference (SOCC 2019) 260–265 (2019).

[9] Anand, R., Maitra, A., Mukhopadhyay, S.: Grover on SIMON. Quantum Information Processing 19 (2020).

[10] Bijwe, S., Chauhan, A. K., Sanadhya, S. K.: Quantum Search for Lightweight Block Ciphers: GIFT, SKINNY, SATURNIN. IACR Cryptology ePrint Archive 2020/1485 (2020).

[11] Chauhan, A. K., Sanadhya, S. K.: Quantum Resource Estimates of Grover's Key Search on ARIA. SPACE 2020, LNCS 12586 238–258 (2020).

[12] Jang, K., Choi, S., Kwon, H., Kim, H., Park, J., Seo, H.: Grover on Korean Block Ciphers. Applied Sciences 10(18) (2020).

[13] Jang, K., Song, G., Kim, H., Kwon, H., Kim, H., Seo, H.: Efficient Implementation of PRESENT and GIFT on Quantum Computers. Applied Sciences 11(11) (2021).

[14] Baksi, A., Jang, K., Song, G., Seo, H., Xiang, Z.: Quantum Implementation and Resource Estimates for RECTANGLE and KNOT. Quantum Information Processing 20 (2021).

[15] Dasu, V. A., Baksi, A., Sarkar, S., Chattopadhyay, A.: LIGHTER-R: Optimized Reversible Circuit Implementation for S-boxes. IEEE International System-on-Chip Conference (SOCC), Singapore 260–265 (2019).

[16] Jean, J., Peyrin, T., Sim, S. M., Tourteaux, J.: Optimizing Implementations of Lightweight Building Blocks. IACR Transactions on Symmetric Cryptology 2017(4) 130–168 (2017).

[17] Anand, R., Maitra, S., Maitra, A., Mukherjee, C. S., Mukhopadhyay, S.: Quantum Resource Estimation for FSR Based Symmetric Ciphers and Related Grover's Attacks. INDOCRYPT 2021, LNSC 13143 179–198 (2021).

[18] Saravanan, P., Jenitha, J., Aasish, S. R., Sanjana, S.: Quantum Circuit Design of RECTANGLE Lightweight Cipher. International Symposium on VLSI Design and Test (VDAT 2021), Surat, India (2021).

[19] Huang, Z., Sun, S.: Synthesizing Quantum Circuits of AES with Lower T-depth and Less Qubits. ASIACRYPT 2022, LNCS 13793 614–644 (2023).

[20] Jang, K., Song, G., Kwon, H., Uhm, S., Kim, H., Lee, W. K., Seo, H.: Grover on PIPO. Electronics 10(10) (2021).

[21] Xiang, Z., Zeng, X., Lin, D., Bao, Z., Zhang, S.: Optimizing Implementations of Linear Layers. IACR Transactions on Symmetric Cryptology 2020(2) 120–145 (2020).

[22] Zou, J., Wei, Z., Sun, S., Liu, X., Wu, W.: Quantum Circuit Implementations of AES with Fewer Qubits. ASIACRYPT 2020, LNCS 12491 697–726 (2020).

[23] Rahman, M., Paul, G.: Grover on KATAN: Quantum Resource Estimation. IEEE Transactions on Quantum Engineering 3 1–9 (2022).

[24] Shor, P. W.: Algorithms for Quantum Computation: Discrete Logarithms and Factoring. In: Proceedings of the 35th Annual Symposium on Foundations of Computer Science 124–134 (1994).

[25] Jaques, S., Naehrig, M., Roetteler, M., Virdia, F.: Implementing Grover Oracles for Quantum Key Search on AES and LowMC. EUROCRYPT 2020, LNCS 12106 280–310 (2020).

[26] Amy, M., Maslov, D., Mosca, M., Roetteler, M.: 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 32(6) 818–830 (2013).