IACR Transactions on Symmetric Cryptology
ISSN 2519-173X, Vol. 2024, No. 2, pp. 322–347. DOI:10.46586/tosc.v2024.i2.322-347

A Framework to Improve the Implementations of Linear Layers

Yufei Yuan[1,2](✉), Wenling Wu[1,2](✉), Tairong Shi[1,3], Lei Zhang[1,4] and Yu Zhang[1,2]

[1] Trusted Computing and Information Assurance Laboratory, Institute of Software Chinese Academy of Science, Beijing, 100190, China
{yufei2021,wenling,tairong2018,zhanglei,zhangyu2021}@iscas.ac.cn
[2] University of Chinese Academy of Sciences, Beijing, 100049, China
[3] PLA SSF Information and Engineering University, Zhengzhou, 450001,Henan
[4] State Key Laboratory of Cryptology, P.O. Box 5159, Beijing 100878, China

Abstract. This paper presents a novel approach to optimizing the linear layer of block ciphers using the matrix decomposition framework. It is observed that the reduction properties proposed by Xiang et al. (in FSE 2020) need to be improved. To address these limitations, we propose a new reduction framework with a complete reduction algorithm and swapping algorithm. Our approach formulates matrix decomposition as a new framework with an adaptive objective function and converts the problem to a Graph Isomorphism problem (GI problem). Using the new reduction algorithm, we were able to achieve lower XOR counts and depths of quantum implementations under the s-XOR metric. Our results outperform previous works for many linear layers of block ciphers and hash functions; some of them are better than the current g-XOR implementation. For the AES MixColumn operation, we get two implementations with 91 XOR counts and depth 13 of in-place quantum implementation, respectively.

Keywords: Linear Layer · Implementation · XOR Counts · Quantum Circuit · AES.

1 Introduction

In recent years, lightweight cryptography has gained significant attention with the advent of the Internet of Things (IoTs) and Radio-Frequency Identification (RFID) technologies. These new rapidly developing applications demand high device performance and energy efficiency. As a result, there has been a growing interest in reducing the complexity and energy consumption of hardware implementation while ensuring the security of the cryptographic system [DEMS16, BJK+20].

To make the implementation of symmetric ciphers as lightweight as possible, researchers have focused on optimizing some metrics, including gate equivalents, latency, circuit size, energy consumption, and so on. Since the linear components of lightweight cipher are often equivalent to a series of XOR operations, the most intuitive requirement is to minimize the XOR counts [Paa97]. Similarly, in quantum circuit design, the corresponding metrics involve the depth, the width (the number of qubits), and the gate count (the number of quantum gates). The quantum logic gates contain Pauli gates, Hadamard gates, CNOT gates, and others. For the linear layer, the depth of the quantum circuit, the number of qubits, and the number of CNOT gates are essential metrics to estimate the complexities in the standard quantum circuit model [NC01].

In designing the primitives of lightweight cryptography, many researchers concentrate on constructing a low-cost linear matrix while ensuring security and meeting software

Received: 2023-11-23 Revised: 2024-03-01 Accepted: 2024-05-01 Published: 2024-06-18

and hardware performance requirements. Various works such as [SKOP15], [KLSW17], [CTG16], and [LS16] have explored the design of matrices containing special structures like circulant, Hadamard, Toeplitz, or involution matrices, aiming to reduce the number of XOR operations. Moreover, Ascon [DEMS16], the winner of the NIST lightweight cryptography cipher competition, employs the permutation which costs two binary XOR operations per bit. Recently in CRYPTO 2023, Solane et al. [EHDRM23] constructed a new linear layer called the "*Twin column parity mixer*", which requires only 3.2 XOR operations per bit and has a bitwise differential branch number of 12 (4 for linear branch number).

Meanwhile, it has been another focus to optimize the implementation of existing linear layers. To further delve into reducing the XOR counts, researchers have established three essential measurement tools in linear layer optimization to formulate the implementation: d-XOR, g-XOR, and s-XOR counts. The g-XOR and s-XOR counts also correspond to two different ways to implement the linear layer, and their definitions will be introduced in detail in the following sections. Many researchers have designed various heuristic algorithms to optimize the implementation based on these metrics.

Furthermore, finding an optimal implementation with g-XOR counts is equivalent to solving the problem Shortest Linear Program (SLP), which is NP-hard [BMP13]. Boyar and Peralta proposed a heuristic algorithm (named BP algorithm) that minimizes the *distance vector* [BP10]. Their algorithm defines a *base set* and updates the distance vector according to the current base set. In the process of updating, the smaller the distance vector is, the closer the current base set is to the aimed matrix. The works on [BFI19, ME19, TP20] have focused on improving the efficiency in different cases, such as dense matrices, by modifying the tie-breaking phase in the BP algorithm. Lin et al. [LXZZ21] introduced a new framework that optimizes the implementation, achieving 91 XOR counts for AES under the g-XOR metric. Their framework mainly relies on several reduction rules and iteratively searches for the subsequence in the implementation that satisfies the rules. Qun et al. [LWF+22] introduced the strategy of backward search based on the Constant Matrix Multiplication (CMM) problem to minimize the depth in implementing the linear layer. In optimizing the s-XOR counts, Xiang et al. [XZL+20]. proposed a reduction algorithm based on reduction properties, achieving the best result of 92 XORs under the s-XOR metric for AES. Under the s-XOR metric, the implementation can be immediately transformed into an in-place quantum circuit. Moreover, some researchers take other potential metrics into their counts instead of only considering the XOR counts [LWF+22, HS22].

Regarding quantum circuits, Grover's algorithm can provide at most a quadratic speedup on finding the key for a symmetric cipher instead of a classical exhaustive key search [Gro96]. Thus, many works focus on improving the quantum implementations of some block ciphers. In [GLRS16], a quantum circuit on AES requiring around 3000 to 7000 qubits was proposed for Grover's attack. Furthermore, the works on [LPS20, ZWS+20, JNRV20] have improved by reducing the number of qubits and lowering the circuit depth. Huang and Sun [HS22] proposed a new framework for generating a quantum circuit for the round function of block ciphers, and the linear layer is taken from the result of Xiang et al. The work in [Max19] achieved an out-of-place quantum implementation with 92 XOR counts, a depth 22, and a circuit width 318. Zhu and Huang [ZH22] extended the ideas from Xiang et al. to quantum circuits by utilizing the properties of move-equivalence and exchange-equivalence and obtained results in the in-place quantum implementation of the AES MixColumn operation with depth 28 and keeping 92 XOR counts. Recently, at ASIACRYPT 2023 [LPZW23], Liu et al. proposed some generic techniques to improve AES quantum implementations, including searching for a new AES MixColumn in-place implementation with 98 XOR counts and depth 16 without ancilla.

Despite many efforts dedicated to optimizing the XOR counts and the quantum depth of linear layer circuits [DBBV+21, MZ22], the circuit implementations of linear components still can be better optimized.

1.1 Our Contribution

Our research presents a more comprehensive framework for implementing the linear layer of a block cipher using heuristic search. This framework mainly employs the method of elementary matrix decomposition, which is able to convert the sequence directly into a specific in-place implementation of XOR operations without auxiliary registers.

We first identify a gap in the work by Xiang et al. and propose several definitions for describing the reduction of type-3 sequences. Moreover, our approach introduces the concept of equivalent sequences, which allows us to describe the original reduction properties introduced by Xiang et al. within a novel framework. Using the equivalent relationship, we model the problem of optimizing type-3 sequences as a GI problem, which avoids the limitations of previous work and enables the framework to reduce type-3 sequences. The framework also introduces the concept of the objective function and prime sequence, which make the framework flexible enough to optimize the implementation from different intentions, e.g., reducing the XOR counts and the quantum circuit's depth. Furthermore, we propose a low-complexity algorithm to determine whether elements in the sequence can be swapped to adjacent positions based on the pairwise exchange nature of the sequence. The new techniques enable the search for implementations with lower XOR counts and lower depths of quantum circuits.

We achieved superior results in many reversible matrices using our newly proposed reduction algorithm. In particular, we have obtained a new implementation of the AES linear layer with 91 XORs under the s-XOR counts metric, which is equivalent to the result obtained under the g-XOR metric [LXZZ21]. Additionally, we have applied our search algorithm to reduce the quantum circuit depth for linear layers by setting the objective function. As a result, we have obtained a new AES MixColumn in-place quantum implementation with 98 XOR counts and depth 13 without ancilla, representing the best result for our known. These results can be applied to the newest AES quantum implementations in [HS22] and [LPZW23].

The implementations can be verified at https://github.com/YuanYufeiISCAS/LinearLayerOptFramework.

1.2 Organization

The paper is structured as follows. Section 2 introduces the notation used in this paper and offers some relevant knowledge and background. Section 3 describes the method of matrix decomposition and points out the incompleteness of the method. Our new framework to optimize the s-XOR sequence is introduced in Section 4. Section 5 shows the detailed results including achieving lower XOR counts and lower depth of quantum implementation of some linear layers. At last, Section 6 makes a conclusion and discusses future work.

2 Preliminaries

In this part, we introduce the notation in this work and offer some relevant knowledge about algebra and graph theory.

2.1 Notation

$\mathbb{F}_2$	binary finite field

$GL(n, \mathbb{F}_2)$	the general linear group of invertible $n \times n$ matrices over the finite field $\mathbb{F}_2$
E_i^1	the i-th type-1 elementary matrix in sequence
E_i^3	the i-th type-3 elementary matrix in sequence
$E(i \leftrightarrow j)$	the specific type-1 matrix is equal to performing row exchange between the i-th row and the j-th row
$E(i + j)$	the specific type-3 matrix is equal to performing row addition over $\mathbb{F}_2$ from the j-th row to the i-th row
$A_{m\times n}$	a matrix with m rows and n columns with entries from $\mathbb{F}_2$
$A_{i,j}$	the entry in position (i, j) of matrix A
I	the identity matrix over $GL(n, \mathbb{F}_2)$
s	a product sequence of type-1 or type-3 matrix
$\|s\|$	the length of product sequence
$\mathcal{S}^{l,n}$	the set of all type-3 sequences with length l on $GL(n, \mathbb{F}_2)$
$\mathcal{P}_f^n$	the set of prime sequences under objective function f on $GL(n, \mathbb{F}_2)$
$\bar{\mathcal{S}}^{l,n}$	the set of representatives of all equivalence classes of $\mathcal{S}^{l,n}$
$\bar{\mathcal{P}}_f^n$	the set of representatives of all equivalence classes of $\mathcal{P}_f^n$
G_A	the bipartite graph corresponding to the relation matrix A
$\overline{G_A}$	the bipartite graph G_A after coloring the vertices

2.2 Relevant Knowledge

`Linear Layer` The linear operation in a substitution-permutation network block cipher can be written as a matrix multiplication like the following form:

$$L_{m\times n} \cdot U_{n\times 1} = O_{m\times 1}$$

L is the matrix form of a linear operation, U is the input bit vector and O is the output bit vector. This work only considers L as a non-singular square matrix.

`Elementary Matrix`

Definition 1. An $n \times n$ square matrix of type-1, type-2, and type-3 is a matrix obtained from identity matrix $I_{n\times n}$ by performing an elementary row (or column) operation of row swapping, row multiplication with non-zero constant, and row addition respectively.

Note that the method of Gauss-Jordan elimination can be used to transform any non-singular matrix into an identity matrix by performing a series of elementary operations. Also, every elementary transformation can be described as the multiplication of the elementary matrix in Definition 1. Formally, we have the Theorem 1:

Theorem 1. *[Art11] A is non-singular if and only if A is the product of elementary matrices.*

Because we are dealing with matrices over the binary field, we can just consider the multiplication of type-1 and type-3 matrices. The general matrix multiplication is not commutative, however, for multiplication of type-1 and type-3 elementary matrices the following property (Property 1) holds.

Property 1. [XZL+20] The commutative transfer properties.

1. $E(i + j)E(k \leftrightarrow l) = E(k \leftrightarrow l)E(f_{k,l}(i) + f_{k,l}(j))$
2. $E(k \leftrightarrow l)E(i + j) = E(f_{k,l}(i) + f_{k,l}(j))E(k \leftrightarrow l)$

where

$$f_{k,l}(x) = \begin{cases} k, & \text{if } x = l, \\ l, & \text{if } x = k, \\ x, & \text{else.} \end{cases}$$

According to Theorem 1 and Property 1, we can decompose a non-singular matrix into a product sequence of elementary matrices, where the preceding part is the product of type-1 matrices and the subsequent part is the product of type-3 matrices. Since row additions correspond to XOR operations in the implementation of the linear function, reducing the number of type-3 matrices in multiplications is the key to optimizing the implementation. Next, we will introduce three metrics in XOR counts to measure the cost of implementing a specific matrix.

Relevant metrics

Definition 2. d-XOR counts. [KPPY14]
The d-XOR counts of a matrix M in $GL(n, \mathbb{F}_2)$, denoted by $wt_d(M)$ is

$$wt_d(M) = \omega(M) - n$$

where $\omega(M)$ denotes the number of ones in the matrix M.

We only use $wt_d(M)$ to estimate the cost of implementation. And the following g-XOR counts and s-XOR counts correspond to the optimal number of XOR operations for two different implementations.

Definition 3. g-XOR counts. [XZL+20]
Consider a matrix $M_{n\times n}$ over $\mathbb{F}_2$, the implementation of M can be viewed as a XOR sequence made of $x_i = x_{j_1} \oplus x_{j_2}$, where $0 \le j_1, j_2 \le i$ for $i = n, n+1, ..., n+t-1$ (t is the length of the corresponding XOR sequence). In the implementation, $x_0, x_1, ..., x_{n-1}$ are the input bits, and the output bits are a subset of $x_i (i > n-1)$. The minimal number of such operation sequences that compute the n bits output is defined as g-XOR counts.

Definition 4. s-XOR counts. [JPST17]
For a non-singular matrix $M \in GL(n, \mathbb{F}_2)$, the minimal number of t is called s-XOR counts (sequential XOR-count) of M, such that

$$M = P \prod_{k=1}^{t} E_k^3$$

where P is a permutation matrix and E_k^3 is a type-3 elementary matrix.

The difference between g-XOR and s-XOR is that operation under the s-XOR metric stores the result within the input lines. The s-XOR sequence expression directly corresponds to a series of XOR operations, where the k-th operation is $x_i = x_i \oplus x_j$ when $E_k^3 = E(i+j)$. Consequently, s-XOR representation offers significant advantages in quantum circuit implementation, such as avoiding additional auxiliary qubits. Moreover, in-place implementation tends to result in a lower T-depth [Max19]. In classic circuits, we only need one instruction to execute $x_i = x_i \oplus x_j$, while the g-XOR sequence requires an extra copy instruction and an additional register on platforms that has invariably 2-operand instructions, typically for some micro-controllers of RISC architectures.

While implementing a linear layer using the s-XOR matrix provides numerous benefits, the s-XOR counts are consistently higher than the g-XOR counts [XZL+20]. Regarding the relationship between d-XOR and g-XOR counts, s-XOR is not always less than d-XOR. In the work of [Köl19], a matrix in $GL(7, \mathbb{F}_2)$ is presented, which exhibits its s-XOR that is larger than the d-XOR. Furthermore, researchers employ diverse approaches when studying these metrics, and the results from both perspectives often complement each other. In our heuristic search work, we have also achieved fewer counts as existing results under the g-XOR metric in many matrices.

After introducing some basic metrics and the matrix decomposition theorems, it is possible to shift our focus to another field of mathematics, namely graph theory. What unites these two domains is the profound interaction between algebraic structures, represented by matrices, and the combinatorial structures of graphs. For instance, matrices derived from graphs, such as adjacency matrices or incidence matrices, can be subjected to various decomposition methods, providing valuable structural insights into the underlying graphs. Furthermore, graph isomorphism enables us to explore matrix similarity, thereby unifying these two areas under the broad umbrella of mathematical structure analysis. The concepts of graph, bipartite graph, and graph isomorphism problem are as follows.

Undirected Graph and Bipartite Graph. An undirected graph is a concept in graph theory [W+01] that consists of a set of vertices and the edges that connect those vertices. In an undirected graph, the edges have no direction, meaning that all the edges are bidirectional and that each edge can be traversed from two connected vertices.

An undirected graph G can be presented as a composition of vertices V and a set of edges E. The vertex set V can be represented as $V = \{v(1), v(2), ..., v(n)\}$, where n is the total number of vertices. The edge set E can be represented as $E = \{(v(i), v(j)) | v(i), v(j) \in V\}$, indicating that for element $(v(i), v(j)) \in E$ there is an edge connecting vertices $v(i)$ and $v(j)$.

Another way of representing a graph is by using its adjacency matrix. An adjacency matrix is an $n \times n$ matrix, where n is the total number of vertices. If there exists an edge between vertex $v(i)$ and $v(j)$, the elements in the i-th row, j-th column, and j-th row, i-th column of the adjacency matrix are 1; otherwise, they are 0. The adjacency matrix can be expressed as:

$$A_{i,j} = \begin{cases} 1, & \text{if there is an edge between vertices } v(i) \text{ and } v(j), \\ 0, & \text{otherwise.} \end{cases}$$

In a bipartite graph, all vertices can be divided into two disjoint sets: V_1 and V_2. Every edge in the graph connects a vertex in V_1 to a vertex in V_2. The bipartite graph has no edges between vertices within the same part. Formally, a graph $G = (V, E)$ is said to be bipartite if there exists a partition $V = V_1 \cup V_2$ such that every edge $(v(u), v(w))$ in E satisfies $v(u) \in V_1$ and $v(w) \in V_2$. It is important to note that in the following discussion, we consider the bipartite graph as a special case of an undirected graph. For convenience, we denote the vertices set V of a bipartite graph G and i-th vertex as $V(G) = \{v(i) | i = 1, 2, ..., m_1 + m_2\}$ where m_1 and m_2 represent the sizes of the two vertex sets respectively. By exchanging the order of the vertices, without loss of generality, we can ensure that the first m_1 vertices are the vertices in V_1, and the last m_2 vertices are the vertices in V_2. When we consider the bipartite graph with $m_1 = m_2 = m$, we can use square matrix of dimension m to describe the graph. To distiguinsh it from adjacency matrix, the matrix is called relation matrix and it can be expressed as:

$$A_{i,j} = \begin{cases} 1, & \text{if there is an edge between vertices } v(i) \text{ and } v(j+m), \\ 0, & \text{otherwise.} \end{cases}$$

Graph Isomorphism Problem For undirected graphs, an isomorphism is a bijective mapping between the vertex sets of two graphs that preserves both the adjacency and the direction of the edges. Formally, if we have two finite undirected graphs $G = (V(G), E(G))$ and $H = (V(H), E(H))$, we say G and H are isomorphic if there exists a bijection

$f : V(G) \to V(H)$ such that for any two vertices $u, v \in V(G)$ are adjacent if and only if $f(u), f(v) \in V(H)$ are adjacent. In other words, isomorphisms are adjacency-preseving bijections between the sets of vertices, and the graph isomorphism problem asks to determine whether two given graphs are isomorphic [Bab18]. Despite being in the NP class, the graph isomorphism problem is not known to be NP-complete or in P class [For96]. However, many powerful quasi-polynomial algorithms are sufficient to solve our problem [HBD17]. In our work, we employ **Nauty and Traces**[1] [MP14] to solve the GI problem, which is well-suited to handle the vertex-colored graph.

3 Method of Matrix Decomposition

In this section, we first commence with a detailed presentation of Xiang et al.'s methods to set the stage for further discussions. For further details on the matrix decomposition method to optimize the implementation of the linear layer, we refer the reader to [XZL+20].

The work of Xiang et al. presents several ingenious methods to decompose a matrix to a sequence of type-1 or type-3 elementary matrix multiplication s :

1. Gauss-Jordan method shows that there exists a sequence of elementary row transformations that converts the original invertible linear matrix L to identity matrix I such that $s \cdot M = I$, then we have $M = s^{-1}$. Type-1 and type-3 matrices are involutional matrices, so attaining a sequence decomposition of M is easy. The method is referred to as strategy-1.

2. The method described above is based on row swapping and row addition. Similarly, we can decompose the matrix L through column swapping and column addition to another sequence s'. Strategy-2 is the designation for this method.

3. The strategy-3 in their work is motivated by reducing the number of type-3 elementary matrices in the decomposition sequence. They first reduce the number of ones in matrix L through row addition as much as possible. If there are multiple local optima, it will choose one operation randomly. When performing row addition of any two rows that cannot reduce the number of "1" in the matrix, the program is turned to strategy-1 or strategy-2 to avoid an infinite loop. Combining strategy-1 or strategy-2 with strategy-3 results in strategy-3-1 and strategy-3-2, respectively.

Assume we get a sequence s through the strategy-3-1 or strategy-3-2 has the form:

$$s = E^1_{m-1} \cdot \ldots \cdot E^1_1 \cdot E^1_0 \cdot E^3_{n-1} \cdot \ldots \cdot E^3_1 \cdot E^3_0$$

In order to identify potential reductions, the method exhaustively examines all possible combinations of two and three type-3 matrices in the given sequence. If these matrices can be swappped to adjacent positions, the algorithm checks whether they satisfy seven distinct reduction properties. Only if both conditions are met, a reduction will be made accordingly.

To introduce the properties, the notation $E(i + j)$ is employed to denote the type-3 elementary matrix with an additional "1" in position (i, j) as compared to the identity matrix. Similarly, the notation $E(i \leftrightarrow j)$ is used to represent the type-1 elementary matrix, which exchanges the i-th row and the j-th row of the identity matrix. Then we have:

Property 2. [XZL+20](Reduction Properties)

- R1: $E(k + i)E(k + j)E(i + j) = E(i + j)E(k + i)$
- R2: $E(i + k)E(k + j)E(i + j) = E(k + j)E(i + k)$

[1] https://pallini.di.uniroma1.it/.

- R3: $E(i+k)E(j+k)E(i+j) = E(i+j)E(j+k)$
- R4: $E(j+k)E(i+k)E(i+j) = E(i+j)E(j+k)$
- R5: $E(k+j)E(k+i)E(i+j) = E(i+j)E(k+i)$
- R6: $E(k+j)E(i+k)E(i+j) = E(i+k)E(k+j)$
- R7: $E(j+i)E(i+j) = E(i \leftrightarrow j)E(j+i)$

Among the properties, i, j, and k are all integers and are different from each other.

Property 3. [XZL+20](Swapping Properties)

- $E(k+l)E(i+j) = E(i+j)E(k+l)$
- $E(i+j)E(k+j) = E(k+j)E(i+j)$
- $E(i+j)E(i+k) = E(i+k)E(i+j)$

Among the properties, i, j, k, l are all integers and are different from each other.

We found that in addition to the reduction properties proposed by Xiang et al., there are more reduction properties. For example, the simplest one:

$$E(i+j)E(i+j) = I$$

And the extra properties which are similar to Property 2:

$$E(i+j)E(k+i)E(i+j) = E(k+i)E(k+j) = E(k+j)E(k+i)$$

$$E(i+j)E(j+k)E(i+j) = E(i+k)E(j+k) = E(j+k)E(i+k)$$

The more complicated properties (noticed that none of them can be deduced from the existing properties):

$$E(k+i)E(i+j)E(j+k)E(k+i)E(i+j) = E(i \leftrightarrow k)E(j \leftrightarrow k)E(k+i)E(j+k)$$

$$E(j+k)E(i+j)E(k+i)E(j+k)E(i+j) = E(i \leftrightarrow j)E(j \leftrightarrow k)E(i+j)E(k+i)$$

When the number of parameters in the reduction property is much greater than 3 (e.g., introduce variables i, j, k, l to describe the property when the number equals 4), it will create more than 50 reduction properties like above, which means the Property 2 are incomplete. Furthermore, the properties proposed by Xiang et al. have other problems besides completeness; that is, they have redundant properties. Consider two equations, R3 and R4, in Property 2, the right-hand formulas are equal, and the left-hand formulas can be obtained by exchanging $E(i+k)$ and $E(j+k)$ through swapping properties. As a result, we need to use a new mathematical method to revisit these properties so that there is no redundancy and a complete description of the features that can be reduced.

Apart from obtaining a comprehensive reduction algorithm, it is crucial to determine whether non-adjacent type-3 matrices in one sequence can be swapped to adjacent positions. The work of Xiang et al. did not propose an explicit algorithm, and the judgment logic in their code is as follows: for E_x^3 and $E_y^3 (x < y)$ in sequence, judging whether they can reach the adjacent position only judges whether E_x^3 can move left one by one until E_{y-1}^3 or E_y^3 can move right one by one until E_{x+1}^3.

However, this logic needs to be more balanced with the complexity of the sequence and account for the potential difficulties in pairwise exchanges. This problem is more complicated when the width of the circuit increases, as the following example shows:

Example 1. Suppose the sequence s consists of two subsequences s_1 and s_2, i.e., $s = s_1 \cdot s_2$. The subsequences $s_1 = E(y+x)E(y+t_0)...E(y+t_m)$, $s_2 = E(u_0+y)E(u_1+y)...E(u_n+y)E(x+y)$ such that t_i, u_i, x, y are different integers. In this situation, we can find $E(y+x)$ and $E(x+y)$ satisfy the R7 in reduction properties. However, the sequence s cannot be reduced through the algorithm proposed by Xiang et al. We only need to move the $E(y+x)$ to the tail of s_1 and move the $E(x+y)$ into the head of s_2, and then it can apply the reduction properties.

To overcome this limitation, we design a new algorithm that accurately determines whether a given matrix can be pairwise exchanged to adjacent positions. This algorithm provides a more reliable approach to address the challenge of swapping properties.

By incorporating this new algorithm, we can enhance the overall effectiveness and robustness of the reducing process. This ensures that the identified properties can not only be reduced but also seamlessly transferred to adjoining positions. In Section 4 of our research, we present an efficient algorithm that can completely solve the pairwise exchange problem in $O(|s|^2)$ time complexity.

4 A Framework to Optimize s-XOR Sequence

In this section, we attempt to redefine the problem, including the definition of matrix decomposition. Starting from a new definition, we derive a complete reduction algorithm within the framework of matrix decomposition.

Definition 5. Let M be an invertible matrix in $GL(n, \mathbb{F}_2)$. One *Matrix Decomposed Sequence* of M is defined as $S = \prod_{i=0}^{k} E_i$ such that $S \cdot I = M$ where matrix E_i in the sequence is either a type-1 or a type-3 matrix. Applying the Property 1, the sequence S can be transformed to *Matrix Decomposed Ordered Sequence*, which has the following form:

$$s = E^1_{m-1}E^1_{m-2}...E^1_0 \cdot E^3_{n-1}E^3_{n-2}...E^3_0$$

Such that:

$$M = S \cdot I = s \cdot I = E^1_{m-1}E^1_{m-2}...E^1_0 E^3_{n-1}E^3_{n-2}...E^3_0 \cdot I$$

If a sequence only contains type-1 elementary matrices, it is called a type-1 sequence. Similarly, if a sequence only contains type-3 elementary matrices, it is referred to as a type-3 sequence. We note the front part of s as type-1 sequence of s, written as $s_0 = E^1_{m-1}E^1_{m-2}...E^1_0$ while the counterpart is $s_1 = E^3_{n-1}E^3_{n-2}...E^3_0$ which means the type-3 sequence of s. In case the sequence is empty (or $|s| = 0$), it is replaced by the identity matrix I.

Based on the above analysis, the E^1 elementary matrix in the circuit corresponds to the permute operation, while E^3 means the XOR operation of two lines of the circuit. The order of type-3 sequence and XOR operation in the circuit are the same. However, since the cost of permutation in the circuit is negligible and the aim is to reduce the number of type-3 matrices in decomposed sequences, the research mainly focuses on type-3 sequences and ignores the details of type-1 sequences. In the following section, the notation $|s|$ is employed to denote the length of type-3 sequences in the matrix decomposed ordered sequences s.

Definition 6. For two type-3 sequences s_1 and s_2, they are *equivalent* if and only if there exist two permutation matrices p_L and p_R, such that the following equation holds:

$$p_L \cdot s_1 \cdot p_R = s_2$$

At the same time, we use $s_1 \sim s_2$ to describe the equivalence relationship.

The permutation matrix can be written as a type-1 sequence, giving rise to an alternative way of expressing the equation: $s_l \cdot s_1 \cdot s_r = s_2$. Specifically, the type-1 sequence s_l (s_r) equals permutation matrix p_L (p_R) corresponding respectively.

It is easy to check that the above relation is reflexive ($I \cdot s \cdot I = s$), symmetric (if $p_L \cdot s_1 \cdot p_R = s_2$ then $p_L^{-1} \cdot s_2 \cdot p_L^{-1} = s_1$) and transitive (since if $p_L \cdot s_1 \cdot p_R = s_2$ and $p'_L \cdot s_2 \cdot p'_R = s_3$, it holds that $(p'_L \cdot p_L) \cdot s_1 \cdot (p_R \cdot p'_R) = s_3$) and therefore it is an equivalence relation. Consequently, we can explore its properties from the perspectives of graph or set theory. For two equivalent sequences s_1, s_2, we assume their values equal to matrix $A, B \in GL(m, \mathbb{F}_2)$ respectively. Let $A = \{a_{i,j}\}_{m\times m}$ and $B = \{b_{i,j}\}_{m\times m}$ be the relation matrices of the bipartite graph G_A and G_B. Under these definitions, we introduce the following lemma:

Lemma 1. *Given two type-3 sequences s_1 and s_2, if s_1 and s_2 are equivalent then the bipartite graph G_A and G_B are isomorphic where $A = s_1 \cdot I$ and $B = s_2 \cdot I$.*

Proof. Assume the s_1 and s_2 are the type-3 sequences on $GF(m, \mathbb{F}_2)$, then matrix A and B are square matrices with size m. It is clear that if s_1 and s_2 are equivalent, there exists vertices permutation σ_L and σ_R such that $p_L \cdot A \cdot p_R = B$ where p_L and p_R are the matrix forms of the permutation. The permutation σ_L can be obtained from p_L through $\sigma_L(i) = j$ when the element in p_L with position (i, j) equals 1.

Then it holds that $B_{i,j} = A_{\sigma_R(i),\sigma_L(j)}$ for $i, j = 1, 2, ...m$. According to the definition of bipartite graph mentioned above, it means for all $B_{i,j} = 1$, then $(v(i), v(j + m)) \in E(G_A)$ and $(v(\sigma_R(i)), v(\sigma_L(j) + m) \in E(G_B)$. In other words, we are able to construct a bijective mapping f from $V(G_A)$ to $V(G_B)$:

$$f(v(i)) = \begin{cases} v(\sigma_R(i)), & \text{if } i \le \text{m}, \\ v(\sigma_L(i-m)+m), & \text{else.} \end{cases}$$

It holds that for any two vertices $v(i)$ and $v(j)$ in G_A, they are adjacent if and only if their image under permutation f, i.e., $f(v(i))$ and $f(v(j))$ are adjacent in G_B. □

However, this lemma does not hold conversely because the isomorphic mapping between bipartite graph G_A and G_B may cross the vertex partition set V_1 and V_2. Then, we cannot transform this situation to the form of Definition 6. We show a simple example:

Example 2. Given two type-3 sequences s_1, s_2, the corresponding values of s_1 and s_2 matrices are as follows. Figure 1 shows the bipartite graphs G_A and G_B generated by s_1 and s_2.

$$A = s_1 \cdot I = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}, \qquad B = s_2 \cdot I = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

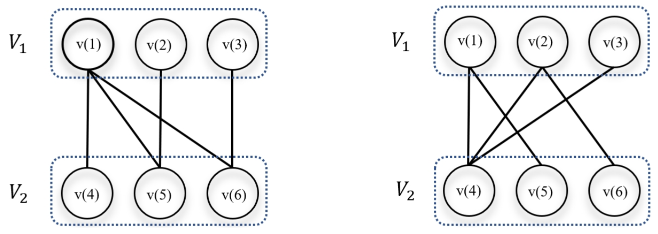

Figure 1: G_A and G_B

There are no permutations p_L and p_R such that $p_L \cdot s_1 \cdot p_R = s_2$ holds. Because the number of ones in each row in matrix A is not in one-to-one correspondence with matrix

B. While the graph G_A and G_B are isomorphic since exist the permutation σ can keep the relationship of edges within G_A:

$$\sigma = \begin{pmatrix} v(1) & v(2) & v(3) & v(4) & v(5) & v(6) \\ v(5) & v(6) & v(4) & v(1) & v(3) & v(2) \end{pmatrix}$$

Actually, it exist p_L and p_R such that $p_L \cdot s_1 \cdot p_R = (s_2 \cdot I)^T$. Furthermore, in many other situations, the isomorphic mapping will map vertices in V_1 to V_2, and also map vertices in V_1 to V_1.

We can use two techniques to handle this situation: first, using a directed graph instead of an undirected one. When we only consider transferring the invertible matrix to the directed graph, which means if matrix A has elements $a_{i,j} = 1$, then the directed bipartite graph has an edge from $i \in V_2$ to $j \in V_1$. The directed edge will restrict the mapping within V_1 and V_2 since the in-degree of the vertices in V_2 is zero where greater than zero in V_1.

The second technique, which has been actually applied in our work, is to color the graph. In graph theory, vertex-colored graphs will consider the color of a vertex as an extra variable. Formally, a graph with $|V|$ nodes is said to be *colored* if each node in the graph is labeled with a positive integer not greater than $|V|$ [JKMT03]. Going back to example 2, we can color the points in V_1 as 1 (blue) and, simultaneously, color the points in V_2 as 2 (red.) The newly generated colored undirected graphs are not isomorphic anymore. Because for two vertex-colored graphs, the isomorphic mapping should preserve the edge and the vertex colors at the same time. Figure 2 shows vertex-colored in Example 1. We note the colored graphs as $\overline{G_A} = C(G_A)$ and $\overline{G_B} = C(G_B)$, C coloring the V_1 with colour 1 (blue), and coloring the V_2 with colour 2 (red). After the above discussion, we can get the following theorem:

Theorem 2. *For two sequence s_1 and s_2, $s_1 \sim s_2$ iff $\overline{G_A}$ is isomorphic to $\overline{G_B}$ where $A = s_1 \cdot I$ and $B = s_2 \cdot I$.*

Proof. According to Lemma 1, we only need to prove the necessity of the theorem. Assume $\overline{G_A}$ is isomorphic to $\overline{G_B}$, then it is able to construct permutations σ_L and σ_R from a bijective mapping $f : V(\overline{G_A}) \to V(\overline{G_B})$. Due to the vertex-coloured graphs, f can be separated to two part now, one's preimage and image are $v(i)$ where $i \leq m$, the other is the opposite. Then this theorem can be proved similarly by the method of Lemma 1. □

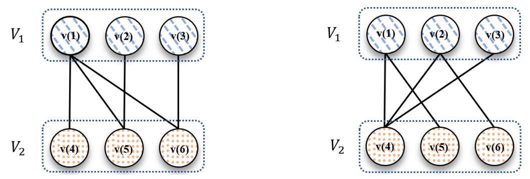

Figure 2: $\overline{G_A}$ and $\overline{G_B}$ where the vertices filled with diagonal dashed lines serve as a representative of "blue" vertices while the vertices filled with dots serve as a representative of "red" vertices.

Therefore, we transform the process of determining the equivalence of two sequences into the process of determining the isomorphism of two colored graphs. The latter problem can be easily solved with the help of Nauty and Traces [MP14].

Definition 7. A type-3 sequence s, s is a *composite sequence* if there exists another type-3 sequence s' such that $s \sim s'$ and $f(s') < f(s)$. If not, we define s as a *prime sequence*. f is the objective function we want to minimize which is mapping the sequence s to $a \in \mathbb{R}$.

In order to better explain the meaning of prime sequence and objective function, we take the search for the minimum number of XOR in implementing a reversible linear matrix as an example up to Section 5.2. In this situation, the objective function $f(s) = |s|$, where $|s|$ is the number of type-3 matrices in the sequence s.

Equivalence relations enable the creation of equivalence classes in collections. To study what kind of sequence can be optimized, we introduce the concept of prime sequence and composite sequence. If we denote $\mathcal{S}^{l,n}$ is the set of all sequences with a length of l and square matrix size of n (n also equals the width of the circuit). And for given objective function f, we denote $\mathcal{P}_f^n$ as the set of all prime sequence with square matrix size of n. Under the same objective function f and square matrix size n, the subscript and superscipt can be omitted without ambiguity. In the given setting, the notations $\mathcal{P}$ and $\mathcal{P}_f^n$ are interchangeable.

In the case of optimized s-XOR counts, it is evident that for sequence s such that $|s| = l$ cannot be decomposed to a sequence s' with shorter length if and only if $s \in \mathcal{P}$. Moreover, we can design a trivial exhaustive algorithm to search all $\mathcal{P}$. However, it is important to note that the size of $\mathcal{P}$ can become excessively large as the variable scales up. In such cases, the use of equivalence classes can be a more effective solution. The set $\mathcal{P}$ is able to divided into $\mathcal{P} = \bar{\mathcal{P}}_0 \cup \bar{\mathcal{P}}_1 \cup ... \cup \bar{\mathcal{P}}_t$, where the set's elements $\bar{\mathcal{P}}_i$ are equivalence classes by the relationship we defined. So, we can only store one element of every equivalence class as a representative, and the designated set is noted as $\bar{\mathcal{P}}_f^n$ ($\bar{\mathcal{P}}$ for short).

Algorithm 1 is a trivial algorithm for an exhaustive search for $\bar{\mathcal{P}}_f^n$ within set $\mathcal{S}^{1,n} \cup \mathcal{S}^{2,n} \cup ... \cup \mathcal{S}^{K,n}$ where K is a big enough integral number to cover all prime sequences. Due to the size of $\mathcal{S}^{K,n}$ is $n^K(n-1)^K$, and the complexity of practical isomorphism algorithms on the worst-case is $O(2^n)$. Therefore the complexity of Algorithm 1 can be estimated as $O(n^{2K} \cdot 2^n \cdot |\bar{\mathcal{P}}|)$. In theory, to find the equivalent sequence that has smaller length, we can exhaustively search $\bar{\mathcal{P}}$ with objective function $f(s) = |s|$. And for any $s \in \mathcal{S}^{l,n}$ we can check whether exists $p \in \mathcal{P}$ such that $s \sim p$. However, it is unrealistic to search the whole $\bar{\mathcal{P}}$ in $GL(n, \mathbb{F}_2)$ for the large n, since the size of $\bar{\mathcal{P}}$ can be roughly estimated by the following corollary.

Algorithm 1 Exhaustive search for $\bar{\mathcal{P}}_f^n$

Input: The size of general linear group n; the objective function f.
Output: $\bar{\mathcal{P}}$

```
1: initialize $\bar{\mathcal{P}} := \varnothing$;
2: generate $S := \mathcal{S}^{1,n} \cup \mathcal{S}^{2,n} \cup ... \cup \mathcal{S}^{K,n}$, ($K$ is a big integral)
3: for all $s \in S$ do
4:     $\mathsf{flag}_{\mathsf{isomorphic}} := \mathit{false}$
5:     for $p \in \bar{\mathcal{P}}$ do
6:         if $s \sim p$ then
7:             $\mathsf{flag}_{\mathsf{isomorphic}} := \mathit{true}$
8:             if $f(s) < f(p)$ then
9:                 $\bar{\mathcal{P}}$ erase $p$
10:                $\bar{\mathcal{P}}$ append $s$
11:            end if
12:            break;
13:        end if
14:    end for
15:    if $\mathsf{flag}_{\mathsf{isomorphic}} == \mathit{false}$ then
16:        $\bar{\mathcal{P}}$ append $s$
17:    end if
18: end for
```

Corollary 1. *The size of $\bar{\mathcal{P}}$ is bounded by the following inequality:*

$$\left\lceil \frac{|GL(n,\mathbb{F}_2)|}{(n!)^2} \right\rceil \leq |\bar{\mathcal{P}}| \leq \frac{|GL(n,\mathbb{F}_2)|}{n!} = \frac{1}{n!}\prod_{k=0}^{n-1}(2^n - 2^k)$$

Proof. The work in [Köl19] proved a similar theory. The difference is that we consider two permutation matrices simultaneously.

Essentially, $\forall p \in \mathcal{P}$ is the optimal type-3 sequence decomposition for corresponding reversible matrix $M' \in GL(n,\mathbb{F}_2)$ under objective function f. Assume M' has two matrix decomposed sequences s_1, s_2 and exist two prime sequence p_1 and p_2 such that $s_1 \sim p_1, s_2 \sim p_2$ for $p_1, p_2 \in \mathcal{P}$. According to the definition of prime sequence, we have $p_L \cdot p_1 \cdot p_R \cdot I = s_1 \cdot I = M'$ and $p'_L \cdot p_2 \cdot p'_R \cdot I = s_2 \cdot I = M'$ which holds $p_1 \sim p_2$. So exists one and only one equivalence class such that the element in this class is equivalent to any matrix decomposed sequences of a certain invertible matrix. In other words, we can find one and only one equivalence class representing a specific reversible matrix.

Then for any two different invertible matrices M_1 and M_2, $M_p \cdot M_1 = M_2$ where M_p is a permutation matrix. Assume p_1, p_2 are the prime sequences representing M_1 and M_2, respectively. It exists the permutation matrices p_L, p_R, p'_L, p'_R satisfy that $p_L \cdot p_1 \cdot p_R = M_1$ and $p'_L \cdot p_2 \cdot p'_R = M_2$. Thus it has $M_p \cdot p_L \cdot p_1 \cdot p_R = p'_L \cdot p_2 \cdot p'_R$ which holds $p_1 \sim p_2$. In other words, we can use the same equivalence class to represent the matrices M_1 and M_2. For one invertible matrix M_1, we can get $n!$ different invertible matrix M_2 by performing $n!$ different row permutation, which are belonged to the same equivalence class. It suggests that the upper bound of the number of distinct equivalence classes can be estimated by dividing the total number of invertible matrices by $n!$. When we consider the row permutation and column permutation simultaneously, it will create some reversible matrix repeatedly. As the result, we have the property $\lceil \frac{|GL(n,\mathbb{F}_2)|}{(n!)^2} \rceil \leq |\bar{\mathcal{P}}| \leq \frac{|GL(n,\mathbb{F}_2)|}{n!}$. $\square$

The corollary demonstrates that if we want to get the complete $\bar{\mathcal{P}}$ which are the optimial sequence of matrix in $GL(n,\mathbb{F}_2)$, the data size is enormous when $n = 16$ or $n = 32$. To tackle this challenge, instead of focusing on finding the global optimum, we need to focus on searching for local optima. We search for reductions which are optimal for every subsequence with length l_m (where l_m is a parameter denoting maximum considered length of a subsequence that can be optimized), which can be swapped to an adjacent position.

The crucial ingredient of the algorithm is the function $\mathsf{QueryBetterSequence}(s)$, $\mathsf{QBS}(s)$ for short. It takes a sequence s of type-3 elementary matrix multiplications from $GL(n,\mathbb{F}_2)$ of length is equal to l_m as input and returns a quadruple $(\mathsf{flag}_{\mathsf{reduce}}, p, p_L, p_R)$. If $\mathsf{flag}_{\mathsf{reduce}}$ is true, the function could reduce s to p such that $f(p) < f(s)$ and $s = p_L \cdot p \cdot p_R$. Otherwise, there was no such sequence p satifies that $f(p) < f(s)$.

In Algorithm 2, we rely on an query function $\mathsf{QBS}()$ and swapping check algorithm $\mathsf{SwapCheck}()$, which we will elaborate on subsequently. Essentially, the algorithm involves iterating over all feasible discontinuous subsequences of length l with pointer array ptr and the function $\mathsf{MovePointer}()$ to update the pointer array. Specifically, the function $\mathsf{MovePointer}(\mathsf{ptr})$ attempts to move the last element of the pointer array ptr. If the current pointer cannot move (that is, reaches the boundary or equals the value of last pointer minus 1), it tries to move the next pointer, and so on until all pointers cannot be moved. If the subsequence can be swapped to continuous and can be reduced to a new sequence with smaller length, the original sequence s is updated accordingly.

In order to create the function $\mathsf{QBS}()$ in the program, it is crucial to leverage the property of equivalence and the algorithm for solving GI problem. For given length l_m, after generating the $\bar{\mathcal{P}}$ through Algorithm 1 then the given sequence $s \in \mathcal{S}^{l_m,n}$ can be optimized by the following steps. If the sequence s is equivalent to $p \in \bar{\mathcal{P}}$ and $f(p) < f(s)$, we can

Algorithm 2 Reduction algorithm for $s \in \mathcal{S}^{l,n}$

```
Input: The sequence $s$ to be reduced; function QBS();the length $l_m$ of
    subsequence in iteration;
Output: reduced sequence from $s$
 1: initialize ptr := $\{0, 1, 2, ..., l_m - 1\}$
 2: initialize loop := $true$
 3: while loop do
 4:     $s'$ := $s$[ptr[0]] || $s$[ptr[1]] || ... || $s$[ptr[$l_m - 1$]]
 5:     (flag$_{\mathsf{swap}}$, pos$_{\mathsf{insert}}$) := SwapCheck(ptr, $s$)
 6:     (flag$_{\mathsf{reduce}}$, $p, p_L, p_R$) := QBS($s'$)
 7:     if flag$_{\mathsf{reduce}}$ and flag$_{\mathsf{swap}}$ then
 8:         Remove items of $s$ with index in ptr
 9:         Insert $p$ at pos$_{\mathsf{insert}}$
10:         Update $s$ based on $p_L$ and $p_R$
11:         Reset ptr := $\{0, 1, 2, ..., l_m - 1\}$
12:     end if
13:     (loop, ptr) := MovePointer(ptr)
14: end while
15: return $s$
```

optimize sequence s by using sequence p instead since we have the following equation:

$$p_L \cdot p \cdot p_R = s$$

As a result, we can build the query function assumed above by precomputing the GI problem to obtain p_L and p_R. We use an example in Appdendix A to demonstrate the process.

The function SwapCheck(ptr, s) in Algorithm 2 is able to judge the specific elements in the sequence whether can be swapped to adjacent locations through Property 3. And if the swapping flag is true, it will also return the indices of the elements after swapping. However, we have to claim that the efficiency of this part of the algorithm dramatically affects the efficiency of our entire reduction algorithm. Although the exchange algorithm used by Xiang et al. is incomplete, its efficiency is fully guaranteed. For type-3 sequence $s = E_0^3 E_1^3 ... E_n^3$, the pointer array is ptr $= \{i_0, i_1, ... i_{l_m-1}\}$ such that $i_0 \leq i_1 ... \leq i_{l_m-1}$ and insert position index is pos$_{\mathsf{insert}}$. In our actual use, the exchange algorithm can be divided into three methods:

Method 1. Judge if exist pos$_{\mathsf{insert}}$ $\in$ ptr, $E_{i_k}^3 (i_k \in$ ptr$)$ can be swapped to the index pos$_{\mathsf{insert}} + k$ in sequence s.

Method 2. Judge if exist pos$_{\mathsf{insert}}$ $\in \{i_0, i_0 + 1, ..., i_{l_m-1}\}$, $E_{i_k}^3 (i_k \in$ ptr$)$ can be swapped to the index pos$_{\mathsf{insert}} + k$ in sequence s.

Method 3. The generalized swapping algorithm (Algorithm 3).

To solve the problem that for given sequence s and target element index array a (assume the size is m), we only need to consider the element among in index array a which are not our targets. It means we need only consider the elements in the interval $s' = s[i_0, ..., i_{l'}]$.

Subsequently, a pointer is moved in an orderly fashion, with the current element being marked as "t". If t happens to be an element in ptr, no further action is required. Otherwise, there are only two possible options: move it to the left outside or right outside the s'. One essential guideline is to keep the left side of s' as all target elements and try to move t to the left first. If this fails, the next step is to move t to the right. When moving t to the right, we only need to determine whether all elements on the right side can be exchanged. If not, we must continue the comparison based on the depth-first traversal method. In the worst-case scenario where each element is to be moved, at most $|s'|$ comparisons are made; hence, the complexity of the overall algorithm can be estimated as $O(|s|^2)$. The Algorithm 3 pseudocode can be found in Appendix B.

We must point out that when applying Method 3, overly complex matrices could have efficiency issues (when the matrix size is 32 bits or more). The reason is that the worst case in Algorithm 2 is that we always perform one optimization when we traverse all possible pointer values. Assume the complexity of the SwapCheck() algorithm is t_0 and t_1 times reduction has been made, then the complexity of the algorithm be estimated as $O(l^{l_m} \cdot t_0 \cdot t_1)$. Thus, striking a balance and using different methods based on different matrices is imperative.

In conclusion, we have proposed a more general reduction framework for optimizing the objective function of the type-3 matrix sequence by solving GI problem. This framework precomputes the function QueryBetterSequence(s) to judge whether the type-3 sequence with length l_m can be reduced. It applies the elimination strategy-3-1 or strategy-3-2 raised by Xiang et al., which continuously creates sequences. Moreover, it will check all subsequences which can be swapped to adjacent positions for each sequence. This approach ensures that all subsequences of length $l'(l' \leq l_m)$ remain optimal, thereby avoiding the limitations of Xiang et al.'s original algorithm highlighted in Section 3, including insufficiency and redundancy.

5 Applications

In this section, we apply our improved algorithm to a variety of invertible matrices and get enhanced implementations under XOR counts and the quantum circuit depth, respectively. For optimizing XOR counts, we conduct a comprehensive comparison with existing algorithms, including those proposed by Xiang et al., Paar, and Boyar. Our new algorithm has achieved the best results we know so far in most of the matrices provided, including 16 × 16 and 32 × 32 matrices. Notably, we improve the AES MixColumn implementation with 91 XORs (Table 4) from the result of 92 [XZL+20], which is equal to the implementation under g-XOR counts [LXZZ21].

In the previous section, we introduced the definition of the objective function, which we regarded as the length of the type-3 sequence for ease of understanding. However, the definition of the objective function is very flexible. For example, we can change the objective function to the quantum circuit depth corresponding to the type-3 sequence. We apply our algorithm to many ciphers' linear layers as well, and Table 5 provides the second AES MixColumn implementation with a depth of 13, which is a considerable improvement from the previous implementation of 28 [ZH22]. While considering some complex matrices, our algorithm still faces efficiency issues, including most of the 64 × 64 matrices and high d-XOR counts matrices in the size of 32 × 32.

5.1 Low XOR counts implementation under s-XOR metric

We applied the algorithm described in Section 4 to various matrices, including the AES linear layer, to search for an implementation of the linear layer with fewer XOR operations. For each matrix, we obtain the corresponding result by running the process on a 64-core computer for over five days. It is evident that, in most cases, there was a significant improvement compared to the original results. However, it is essential to note that while this method offers a more comprehensive approach than the one proposed by Xiang et al., it does have efficiency limitations. Mainly, when the initial sequence generated by [XZL+20] involves approximately 500 or more XOR operations, the efficiency gap between the two algorithms becomes pronounced. In such cases, when the matrix size exceeds 16×16, we must switch from the swapping strategy described in Section 4's method 3 to method 2. Moreover, the parameter l we set here is 4.

In order to compare efficiency, we have included results from recent studies in Table 2. To distinguish between the two methods of implementation, namely s-XOR and g-XOR implementations, we have divided them into separate sub-tables for comparison.

Table 2: Implementation cost of cipher linear layers under different methods.

Linear Layer	[KLSW17]	[BFI19]	[TP20]	[LXZZ21]	[XZL+20]	This paper[1]
	g-XOR				s-XOR	
AES [JV02]	97	95	94	91	92	**91***
Clefia M0 [SSA+07]	106	102	103	96	98	**97**
Clefia M1 [SSA+07]	111	110	108	108	103	**103**
Fox mu4 [JV05]	137	131	135	130	136	**130***
Twofish [SKW+98]	129	125	122	121	111	**110***
Joltik [JNP15]	48	47	44	43	44	**44**
SmallScale AES [CMR05]	47	45	45	43	43	**42***
Whirlwind M0 [BNN+10]	212	210	-	183	183	**173***
Whirlwind M1 [BNN+10]	235	234	-	180	190	**181**
4×4 matrices over $GL(4, \mathbb{F}_2)$						
[SKOP15]	48	46	46	44	44	**44**
[LS16]	44	44	43	43	44	**44**
[LW16]	44	44	43	43	44	**44**
[CTG16]	43	42	41	40	41	**41**
[JPST17]	47	46	40	40	41	**41**
4×4 matrices over $GL(8, \mathbb{F}_2)$						
[SKOP15]	98	94	92	91	90	**89***
[LS16]	112	110	108	107	121	**115**
[LW16]	102	102	99	99	104	**104**
[CTG16]	110	108	106	105	114	**112**
[SS16]	107	104	102	100	114	**113**
[JPST17]	86	86	92	80	82	**82**
[SKOP15](Involutory)	100	94	92	89	91	**90**
[LW16](Had., Involutory)	91	90	88	86	87	**86***
[LW16](Circ., Involutory)	97	-	-	-	86	**86**
[SS16](Involutory)	100	98	97	92	93	**92***
[JPST17](Involutory)	91	92	86	-	83	**82***
8×8 matrices over $GL(4, \mathbb{F}_2)$						
[SKOP15]	209	194	-	172	170	**169***
[SS17]	201	203	-	177	183	**183**
[SKOP15](Involutory)	217	212	-	172	185	**182**

[1] Results with (*) indicates that they are also the state-of-the-art known to us under g-XOR counting.

5.2 Optimizing the depth of quantum implementations of linear layers

The method in section 4 is a universal optimization framework that enables the adaptation of the objective function to search for low-depth implementations of the linear layer in quantum circuits. This is achieved by setting $f(s)$, the objective function, to be equal to the depth of sequence s, then the $\bar{\mathcal{P}}$ store the type-3 sequence in which the elements are the lowest depth of sequences to get the corresponding matrices. In the 5.2 section, we set the $f(s) = M \cdot depth(s) + |s|$ where M is a big number that can consider the depth of the circuit first, and XOR counts second. Specifically, $M = 10^{\lceil \log_{10} |s| \rceil + 2}$ in our experiment.

In the case of a type-3 sequence, it corresponds to an XOR operation sequence. While the order of this operation sequence should be maintained, it is not absolute. The only requirement is to keep the values of x_i and x_j on which this operation depends the same. From this perspective, a type-3 sequence can be viewed as a graph, where each operation represents a node. These nodes contain x_i and x_j and are connected to the previous nodes that contain either x_i or x_j. The nodes in each layer share the same quantum circuit depth, and our objective function is transformed into the depth of this graph. The following example shows how the function f works on a quantum circuit:

Example 3. Consider a type-3 sequence: $E(4+5)E(7+5)E(1+2)E(2+4)E(1+3)E(0+4)E(1+2)$. We initiate a counter array with zeros to store the depth. We then traverse the entire sequence from the beginning. The first operation, $E(1+2)$, checks that the depths stored in counter[1] and counter[2] are both 0. Consequently, the $E(1+2)$ depth is set to 1. The same procedure is followed for $E(0+4)$. However, when we reach $E(1+3)$, we find that counter[1] is 1 and counter[3] is 0. In this case, we take the maximum value and increment the depth of $E(1+3)$ by one, resulting in depth 2. By following this process, we can determine the depth of the entire sequence, which in this example is 3. Figure 3 shows how it works.

We apply this approach to the AES cipher's linear layer and obtain a quantum circuit with depth 13, which is in Table 5 and the other linear layer results are in Table 3. A comparison with the method proposed by [ZH22] reveals that our implementation exhibits high parallelism. In most layers, the number of included parallel operations exceeds 5, whereas in Zhu et al.'s method, a considerable portion of the circuit layers only includes less than three parallel operations. However, we have to claim that the result in Table 3 does not keep the best XOR counts, and the implementations might use several extra XORs to achieve high parallelism. As mentioned in the introduction, many works use the result of the s-XOR sequence to construct an AES quantum circuit since the s-XOR is an in-place implementation of a linear layer and does not need ancilla qubits. It always leads to lower width and full depth in the circuit. Taking the work in [HS22] as an example, they proposed low-depth quantum circuits for AES, which applied the linear layer raised by Xiang et al.[XZL+20]. They used #Q estimator to get the implementation depth to 30. If taking our new implementation, it can decrease the full depth from 2198 (with T-depth-4 S-box) to 2072 and from 2312 (with T-depth-3 S-box) to 2186 while using extra 216 XOR operations. The number of additional CNOT gates is less than 0.1% of the total number of CNOT gates in the original quantum circuit.

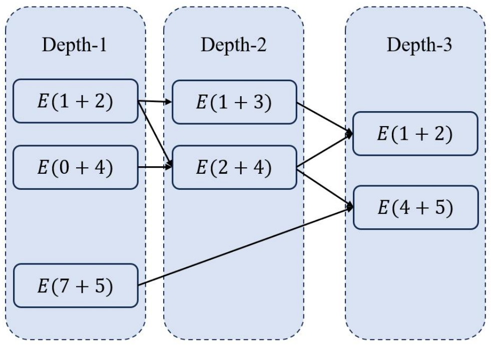

Figure 3: Conversion of quantum circuits from a type-3 sequence

Table 3: Quantum circuit depth comparison for implementing cipher linear layers under different methods

Linear Layer	size	[XZL+20]		[ZH22]		This paper	
		XOR	Depth	XOR	Depth	XOR	Depth
AES [JV02]	32	92	41	92	28	98	**13**
ANUBIS [Bar00]	32	98	40	98	20	102	**14**
Clefia M0 [SSA+07]	32	98	41	98	27	105	**15**
Clefia M1 [SSA+07]	32	103	41	103	16	106	**13**
Fox Mu4 [JV05]	32	136	75	136	48	161	**31**
QARMA128 [Ava17]	32	48	12	48	5	48	**5**
Twofish [SKW+98]	32	111	53	111	29	113	**22**
Whirlwind M0 [BNN+10]	32	183	93	183	51	249	**46**
Whirlwind M1 [BNN+10]	32	190	90	190	54	264	**50**
Joltik [JNP15]	16	44	23	44	17	48	**9**
MIDORI [BBI+15]	16	24	9	24	3	24	**4**
SmallScale AES [CMR05]	16	43	29	43	19	46	**10**
4×4 matrices over $GL(4, \mathbb{F}_2)$							
[SKOP15]	16	44	30	44	22	47	**11**
[LS16]	16	44	30	44	25	49	**12**
[LW16]	16	44	29	44	27	48	**12**
[CTG16]	16	41	27	41	21	43	**10**
[JPST17]	16	41	28	41	18	45	**9**
4×4 matrices over $GL(8, \mathbb{F}_2)$							
[SKOP15]	32	90	42	90	20	96	**14**
[LS16]	32	121	79	121	54	136	**29**
[LW16]	32	104	69	104	42	129	**27**
[CTG16]	32	114	72	114	47	132	**26**
[SS16]	32	114	58	114	40	147	**26**
[JPST17]	32	82	43	82	22	86	**13**
[SKOP15](Involutory)	32	91	39	91	16	94	**13**
[LW16](Had., Involutory)	32	87	39	87	19	96	**11**
[SS16](Involutory)	32	93	42	93	18	98	**13**
[JPST17](Involutory)	32	83	34	83	14	85	**11**
8×8 matrices over $GL(4, \mathbb{F}_2)$							
[SKOP15]	32	170	89	170	49	249	**48**
[SS17]	32	183	83	183	44	247	**49**
[SKOP15](Involutory)	32	185	85	185	37	248	**46**

Table 4: The implementation of AES MixColumn with 91 XORs.

No.	Operation	No.	Operation	No.	Operation
1	$x[15] = x[15] + x[23]$	32	$x[12] = x[12] + x[19]$	63	$x[18] = x[18] + x[9]$
2	$x[25] = x[25] + x[1]$	33	$x[19] = x[19] + x[27]$	64	$x[27] = x[27] + x[11] \quad y[11]$
3	$x[10] = x[10] + x[18]$	34	$x[11] = x[11] + x[10]$	65	$x[24] = x[24] + x[16] \quad y[8]$
4	$x[5] = x[5] + x[13]$	35	$x[27] = x[27] + x[3]$	66	$x[1] = x[1] + x[16]$
5	$x[3] = x[3] + x[19]$	36	$x[3] = x[3] + x[2]$	67	$x[23] = x[23] + x[22] \quad y[15]$
6	$x[4] = x[4] + x[28]$	37	$x[2] = x[2] + x[10]$	68	$x[16] = x[16] + x[7]$
7	$x[16] = x[16] + x[0]$	38	$x[10] = x[10] + x[26]$	69	$x[16] = x[16] + x[0] \quad y[16]$
8	$x[1] = x[1] + x[9]$	39	$x[3] = x[3] + x[26]$	70	$x[22] = x[22] + x[6]$
9	$x[23] = x[23] + x[7]$	40	$x[26] = x[26] + x[25]$	71	$x[30] = x[30] + x[22] \quad y[6]$
10	$x[7] = x[7] + x[31]$	41	$x[2] = x[2] + x[25] \quad y[26]$	72	$x[22] = x[22] + x[13] \quad y[30]$
11	$x[28] = x[28] + x[20]$	42	$x[25] = x[25] + x[17]$	73	$x[13] = x[13] + x[20]$
12	$x[22] = x[22] + x[14]$	43	$x[9] = x[9] + x[25]$	74	$x[20] = x[20] + x[12] \quad y[12]$
13	$x[8] = x[8] + x[16]$	44	$x[17] = x[17] + x[8]$	75	$x[26] = x[26] + x[18] \quad y[10]$
14	$x[0] = x[0] + x[8]$	45	$x[8] = x[8] + x[7] \quad y[24]$	76	$x[11] = x[11] + x[19] \quad y[3]$
15	$x[14] = x[14] + x[30]$	46	$x[19] = x[19] + x[23]$	77	$x[6] = x[6] + x[14] \quad y[22]$
16	$x[30] = x[30] + x[6]$	47	$x[10] = x[10] + x[1] \quad y[2]$	78	$x[7] = x[7] + x[31] \quad y[31]$
17	$x[24] = x[24] + x[15]$	48	$x[25] = x[25] + x[15]$	79	$x[19] = x[19] + x[3] \quad y[19]$
18	$x[13] = x[13] + x[29]$	49	$x[3] = x[3] + x[7] \quad y[27]$	80	$x[12] = x[12] + x[28] \quad y[4]$
19	$x[6] = x[6] + x[5]$	50	$x[4] = x[4] + x[7]$	81	$x[18] = x[18] + x[10] \quad y[18]$
20	$x[5] = x[5] + x[21]$	51	$x[7] = x[7] + x[15]$	82	$x[17] = x[17] + x[16]$
21	$x[21] = x[21] + x[28]$	52	$x[15] = x[15] + x[14]$	83	$x[9] = x[9] + x[17] \quad y[17]$
22	$x[28] = x[28] + x[23]$	53	$x[25] = x[25] + x[0] \quad y[9]$	84	$x[17] = x[17] + x[1] \quad y[25]$
23	$x[29] = x[29] + x[5]$	54	$x[0] = x[0] + x[23]$	85	$x[28] = x[28] + x[4] \quad y[28]$
24	$x[5] = x[5] + x[4] \quad y[29]$	55	$x[1] = x[1] + x[23]$	86	$x[21] = x[21] + x[13] \quad y[5]$
25	$x[18] = x[18] + x[2]$	56	$x[23] = x[23] + x[31]$	87	$x[13] = x[13] + x[5] \quad y[13]$
26	$x[28] = x[28] + x[3]$	57	$x[31] = x[31] + x[30]$	88	$x[31] = x[31] + x[15] \quad y[7]$
27	$x[3] = x[3] + x[11]$	58	$x[0] = x[0] + x[24] \quad y[0]$	89	$x[1] = x[1] + x[25] \quad y[1]$
28	$x[11] = x[11] + x[15]$	59	$x[14] = x[14] + x[29]$	90	$x[15] = x[15] + x[23] \quad y[23]$
29	$x[4] = x[4] + x[12]$	60	$x[29] = x[29] + x[21] \quad y[21]$	91	$x[14] = x[14] + x[22] \quad y[14]$
30	$x[20] = x[20] + x[4]$	61	$x[4] = x[4] + x[19] \quad y[20]$		
31	$x[12] = x[12] + x[11]$	62	$x[19] = x[19] + x[18]$		

Table 5: The quantum implementation of AES MixColumn with depth of 13 and 98 XORs

Depth	Operation	Depth	Operation	Depth	Operation
1	$x[18] = x[18] + x[26]$	5	$x[2] = x[2] + x[1]$	10	$x[10] = x[10] + x[2]$
1	$x[20] = x[20] + x[12]$	5	$x[24] = x[24] + x[0]$	11	$x[14] = x[14] + x[13]$
1	$x[21] = x[21] + x[5]$	6	$x[10] = x[10] + x[18]$	11	$x[31] = x[31] + x[30]$
1	$x[23] = x[23] + x[31]$	6	$x[3] = x[3] + x[11] \quad y[19]$	11	$x[25] = x[25] + x[17] \quad y[1]$
1	$x[11] = x[11] + x[27]$	6	$x[21] = x[21] + x[28]$	11	$x[0] = x[0] + x[8] \quad y[16]$
1	$x[8] = x[8] + x[24]$	6	$x[27] = x[27] + x[26]$	11	$x[2] = x[2] + x[18]$
1	$x[22] = x[22] + x[6]$	6	$x[0] = x[0] + x[15]$	11	$x[15] = x[15] + x[7] \quad y[7]$
1	$x[25] = x[25] + x[1]$	6	$x[24] = x[24] + x[16]$	11	$x[27] = x[27] + x[11] \quad y[27]$
2	$x[31] = x[31] + x[15]$	6	$x[12] = x[12] + x[4]$	12	$x[13] = x[13] + x[5] \quad y[21]$
2	$x[4] = x[4] + x[20]$	6	$x[20] = x[20] + x[31]$	12	$x[14] = x[14] + x[29] \quad y[14]$
2	$x[5] = x[5] + x[29]$	7	$x[18] = x[18] + x[9]$	12	$x[16] = x[16] + x[8] \quad y[0]$
2	$x[27] = x[27] + x[3]$	7	$x[26] = x[26] + x[25]$	12	$x[7] = x[7] + x[31] \quad y[23]$
2	$x[6] = x[6] + x[14]$	7	$x[15] = x[15] + x[23]$	12	$x[30] = x[30] + x[6]$
2	$x[19] = x[19] + x[18]$	7	$x[12] = x[12] + x[19]$	12	$x[1] = x[1] + x[25] \quad y[9]$
2	$x[12] = x[12] + x[28]$	7	$x[1] = x[1] + x[0]$	12	$x[2] = x[2] + x[26] \quad y[18]$
2	$x[1] = x[1] + x[9]$	7	$x[24] = x[24] + x[31] \quad y[8]$	13	$x[31] = x[31] + x[23] \quad y[31]$
3	$x[30] = x[30] + x[14]$	7	$x[28] = x[28] + x[20]$	13	$x[5] = x[5] + x[21] \quad y[5]$
3	$x[15] = x[15] + x[7]$	7	$x[21] = x[21] + x[13] \quad y[29]$	13	$x[30] = x[30] + x[22] \quad y[22]$
3	$x[3] = x[3] + x[18]$	8	$x[9] = x[9] + x[25]$	13	$x[8] = x[8] + x[24] \quad y[24]$
3	$x[29] = x[29] + x[21]$	8	$x[23] = x[23] + x[22]$	13	$x[6] = x[6] + x[14] \quad y[6]$
3	$x[13] = x[13] + x[20]$	8	$x[11] = x[11] + x[19]$		
3	$x[27] = x[27] + x[31]$	8	$x[26] = x[26] + x[10] \quad y[26]$		
3	$x[28] = x[28] + x[4]$	8	$x[1] = x[1] + x[31]$		
3	$x[9] = x[9] + x[24]$	8	$x[18] = x[18] + x[17] \quad y[10]$		
4	$x[14] = x[14] + x[5]$	8	$x[12] = x[12] + x[3] \quad y[12]$		
4	$x[7] = x[7] + x[6]$	8	$x[28] = x[28] + x[4] \quad y[4]$		
4	$x[18] = x[18] + x[2]$	9	$x[9] = x[9] + x[16] \quad y[17]$		
4	$x[3] = x[3] + x[23]$	9	$x[22] = x[22] + x[14] \quad y[30]$		
4	$x[4] = x[4] + x[27]$	9	$x[25] = x[25] + x[8]$		
4	$x[19] = x[19] + x[15]$	9	$x[23] = x[23] + x[7] \quad y[15]$		
4	$x[17] = x[17] + x[24]$	9	$x[11] = x[11] + x[10] \quad y[3]$		
4	$x[29] = x[29] + x[20] \quad y[13]$	9	$x[19] = x[19] + x[15]$		
4	$x[30] = x[30] + x[22]$	9	$x[17] = x[17] + x[1] \quad y[25]$		
5	$x[6] = x[6] + x[21]$	9	$x[20] = x[20] + x[12] \quad y[20]$		
5	$x[16] = x[16] + x[23]$	10	$x[25] = x[25] + x[31]$		
5	$x[5] = x[5] + x[12]$	10	$x[19] = x[19] + x[27] \quad y[11]$		
5	$x[4] = x[4] + x[15] \quad y[28]$	10	$x[14] = x[14] + x[30]$		
5	$x[27] = x[27] + x[10]$	10	$x[1] = x[1] + x[16]$		
5	$x[20] = x[20] + x[11]$	10	$x[8] = x[8] + x[15]$		

6 Conclusion

In this work, we present a new flexible framework of heuristic search for optimizing the XOR counts under s-XOR and the depth of the quantum circuit. With the new definition and introduction of the new equivalent relationship, we solve the reducing problem by solving the GI problem instead of using reducing properties. The new approach avoids the previous limitations and achieves superior results in many reversible matrices. For AES MixColumn, we achieve an implementation with 91 XORs which equals the best implementation under the g-XOR metric. Furthermore, compared with previous work, the depth of quantum depth has been reduced from depth 16 to depth 13 without ancilla. The results show the new approach is effective and has the potential to improve the existing quantum circuits.

As a future work, it would be interesting to study the more effective way to create original type-3 sequences instead of using strategy-3-1 or strategy-3-2. Moreover, the research about solving SLP or SLPD problems can be potentially inspired by this work.

Acknowledgments

The authors would like to thank the anonymous reviewers for their sightful comments and patient suggestions. This work is supported by the National Natural Science Foundation of China (No. 62072445 and No. 62202493) and the CAS Project for Young Scientists in Basic Research (Grant No. YSBR-035).

References

[Art11] Michael Artin. *Algebra (Second Edition)*. Pearson Prentice Hall, 2011.

[Ava17] Roberto Avanzi. The QARMA Block Cipher Family. Almost MDS Matrices Over Rings with Zero Divisors, Nearly Symmetric Even-mansour Constructions with Non-involutory Central Rounds, and Search Heuristics for Low-latency S-boxes. *IACR Transactions on Symmetric Cryptology*, pages 4–44, 2017.

[Bab18] László Babai. Group, graphs, algorithms: the graph isomorphism problem. In *Proceedings of the International Congress of Mathematicians: Rio de Janeiro 2018*, pages 3319–3336. World Scientific, 2018.

[Bar00] Paulo SLM Barreto. The Anubis Block Cipher. *NESSIE*, 2000.

[BBI$^+$15] Subhadeep Banik, Andrey Bogdanov, Takanori Isobe, Kyoji Shibutani, Harunaga Hiwatari, Toru Akishita, and Francesco Regazzoni. Midori: A Block Cipher for Low Energy. In *Advances in Cryptology–ASIACRYPT 2015: 21st International Conference on the Theory and Application of Cryptology and Information Security, Auckland, New Zealand, November 29–December 3, 2015, Proceedings, Part II 21*, pages 411–436, 2015.

[BFI19] Subhadeep Banik, Yuki Funabiki, and Takanori Isobe. More Results on Shortest Linear Programs. In *Advances in Information and Computer Security: 14th International Workshop on Security, IWSEC 2019, Tokyo, Japan, August 28–30, 2019, Proceedings 14*, pages 109–128, 2019.

[BJK$^+$20] Christof Beierle, Jérémy Jean, Stefan Kölbl, Gregor Leander, Amir Moradi, Thomas Peyrin, Yu Sasaki, Pascal Sasdrich, and Siang Meng Sim. Skinny-aead and Skinny-hash. *IACR Transactions on Symmetric Cryptology*, pages 88–131, 2020.

[BMP13] Joan Boyar, Philip Matthews, and René Peralta. Logic Minimization Techniques with Applications to Cryptology. *Journal of Cryptology*, 26:280–312, 2013.

[BNN$^+$10] Paulo Barreto, Ventzislav Nikov, Svetla Nikova, Vincent Rijmen, and Elmar Tischhauser. Whirlwind: a New Cryptographic Hash Function. *Designs, codes and cryptography*, 56:141–162, 2010.

[BP10] Joan Boyar and René Peralta. A New Combinational Logic Minimization Technique with Applications to Cryptology. In *Experimental Algorithms: 9th International Symposium, SEA 2010, Ischia Island, Naples, Italy, May 20-22, 2010. Proceedings 9*, pages 178–189, 2010.

[CMR05] Carlos Cid, Sean Murphy, and Matthew JB Robshaw. Small Scale Variants of the AES. In *International Workshop on Fast Software Encryption*, pages 145–162, 2005.

[CTG16] Beierle Christof, Kranz Thorsten, and Leander Gregor. Lightweight Multiplication in GF(2n) with Applications to MDS Matrices; CRYPTO 2016. LNCS 9814, 2016.

[DBBV$^+$21] Timothee Goubault De Brugiere, Marc Baboulin, Benoît Valiron, Simon Martiel, and Cyril Allouche. Reducing the Depth of Linear Reversible Quantum Circuits. *IEEE Transactions on Quantum Engineering*, 2:1–22, 2021.

[DEMS16] Christoph Dobraunig, Maria Eichlseder, Florian Mendel, and Martin Schläffer. Ascon V1. 2. *Submission to the CAESAR Competition*, 5(6):7, 2016.

[EHDRM23] Solane El Hirch, Joan Daemen, Raghvendra Rohit, and Rusydi H Makarim. Twin Column Parity Mixers and Gaston-A New Mixing Layer and Permutation. *Cryptology ePrint Archive*, 2023.

[For96] Scott Fortin. The Graph Isomorphism Problem. 1996.

[GLRS16] Markus Grassl, Brandon Langenberg, Martin Roetteler, and Rainer Steinwandt. Applying Grover's Algorithm to AES: Quantum Resource Estimates. In *International Workshop on Post-Quantum Cryptography*, pages 29–43, 2016.

[Gro96] Lov K Grover. A Fast Quantum Mechanical Algorithm for Database Search. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 212–219, 1996.

[HBD17] Harald Andrés Helfgott, Jitendra Bajpai, and Daniele Dona. Graph Isomorphisms in Quasi-polynomial Time. *arXiv preprint arXiv:1710.04574*, 2017.

[HS22] Zhenyu Huang and Siwei Sun. Synthesizing Quantum Circuits of AES with Lower T-depth and Less Qubits. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 614–644, 2022.

[JKMT03] Birgit Jenner, Johannes Köbler, Pierre McKenzie, and Jacobo Torán. Completeness results for graph isomorphism. *Journal of Computer and System Sciences*, 66(3):549–566, 2003.

[JNP15] Jérémy Jean, Ivica Nikolić, and Thomas Peyrin. Joltik V1. 3. *CAESAR Round*, 2, 2015.

[JNRV20] Samuel Jaques, Michael Naehrig, Martin Roetteler, and Fernando Virdia. Implementing Grover Oracles for Quantum Key Search on AES and LowMC. In *Advances in Cryptology–EUROCRYPT 2020: 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10–14, 2020, Proceedings, Part II 30*, pages 280–310, 2020.

[JPST17] Jérémy Jean, Thomas Peyrin, Siang Meng Sim, and Jade Tourteaux. Optimizing Implementations of Lightweight Building Blocks. *Cryptology ePrint Archive*, 2017.

[JV02] Daemen Joan and Rijmen Vincent. The Design of Rijndael: AES-the Advanced Encryption Standard. *Information Security and Cryptography*, 2002.

[JV05] Pascal Junod and Serge Vaudenay. FOX: a New Family of Block Ciphers. In *Selected Areas in Cryptography: 11th International Workshop, SAC 2004, Waterloo, Canada, August 9-10, 2004, Revised Selected Papers 11*, pages 114–129, 2005.

[KLSW17] Thorsten Kranz, Gregor Leander, Ko Stoffelen, and Friedrich Wiemer. Shorter Linear Straight-line Programs for MDS Matrices. *IACR Transactions on Symmetric Cryptology*, pages 188–211, 2017.

[Köl19] Lukas Kölsch. XOR-counts and Lightweight Multiplication with Fixed Elements in Binary Finite Fields. In *Advances in Cryptology–EUROCRYPT 2019: 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19–23, 2019, Proceedings, Part I 38*, pages 285–312, 2019.

[KPPY14] Khoongming Khoo, Thomas Peyrin, Axel Y Poschmann, and Huihui Yap. FOAM: Searching for Hardware-optimal SPN Structures and Components with a Fair Comparison. In *Cryptographic Hardware and Embedded Systems–CHES 2014: 16th International Workshop, Busan, South Korea, September 23-26, 2014. Proceedings 16*, pages 433–450, 2014.

[LPS20] Brandon Langenberg, Hai Pham, and Rainer Steinwandt. Reducing the Cost of Implementing the Advanced Encryption Standard As a Quantum Circuit. *IEEE Transactions on Quantum Engineering*, 1:1–12, 2020.

[LPZW23] Qun Liu, Bart Preneel, Zheng Zhao, and Meiqin Wang. Improved Quantum Circuits for AES: Reducing the Depth and the Number of Qubits. *Cryptology ePrint Archive*, 2023.

[LS16] Meicheng Liu and Siang Meng Sim. Lightweight MDS Generalized Circulant Matrices. In *International Conference on Fast Software Encryption*, pages 101–120, 2016.

[LW16] Yongqiang Li and Mingsheng Wang. On the Construction of Lightweight Circulant Involutory MDS Matrices. In *International Conference on Fast Software Encryption*, pages 121–139, 2016.

[LWF$^+$22] Qun Liu, Weijia Wang, Yanhong Fan, Lixuan Wu, Ling Sun, and Meiqin Wang. Towards Low-latency Implementation of Linear Layers. *Cryptology ePrint Archive*, 2022.

[LXZZ21] Da Lin, Zejun Xiang, Xiangyong Zeng, and Shasha Zhang. A Framework to Optimize Implementations of Matrices. In *Cryptographers' Track at the RSA Conference*, pages 609–632, 2021.

[Max19] Alexander Maximov. AES MixColumn with 92 XOR Gates. *Cryptology ePrint Archive*, 2019.

[ME19] Alexander Maximov and Patrik Ekdahl. New Circuit Minimization Techniques for Smaller and Faster AES SBoxes. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 91–125, 2019.

[MP14] Brendan D McKay and Adolfo Piperno. Practical Graph Isomorphism, II. *Journal of symbolic computation*, 60:94–112, 2014.

[MZ22] Dmitri Maslov and Ben Zindorf. Depth Optimization of CZ, CNOT, and Clifford Circuits. *IEEE Transactions on Quantum Engineering*, 3:1–8, 2022.

[NC01] Michael A Nielsen and Isaac L Chuang. Quantum Computation and Quantum Information. *Phys. Today*, 54(2):60, 2001.

[Paa97] Christof Paar. Optimized Arithmetic for Reed-Solomon Encoders. In *Proceedings of IEEE international symposium on information theory*, page 250, 1997.

[SKOP15] Siang Meng Sim, Khoongming Khoo, Frédérique Oggier, and Thomas Peyrin. Lightweight MDS Involution Matrices. In *Fast Software Encryption: 22nd International Workshop, FSE 2015, Istanbul, Turkey, March 8-11, 2015, Revised Selected Papers 22*, pages 471–493, 2015.

[SKW+98] Bruce Schneier, John Kelsey, Doug Whiting, David Wagner, Chris Hall, and Niels Ferguson. Twofish: A 128-bit Block Cipher. *NIST AES Proposal*, 15(1):23–91, 1998.

[SS16] Sumanta Sarkar and Habeeb Syed. Lightweight Diffusion Layer: Importance of Toeplitz Matrices. *Cryptology ePrint Archive*, 2016.

[SS17] Sumanta Sarkar and Habeeb Syed. Analysis of Toeplitz MDS Matrices. In *Australasian Conference on Information Security and Privacy*, pages 3–18, 2017.

[SSA+07] Taizo Shirai, Kyoji Shibutani, Toru Akishita, Shiho Moriai, and Tetsu Iwata. The 128-bit Blockcipher CLEFIA. In *Fast Software Encryption: 14th International Workshop, FSE 2007, Luxembourg, Luxembourg, March 26-28, 2007, Revised Selected Papers 14*, pages 181–195, 2007.

[TP20] Quan Quan Tan and Thomas Peyrin. Improved Heuristics for Short Linear Programs. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 203–230, 2020.

[W+01] Douglas Brent West et al. *Introduction to Graph Theory*, volume 2. 2001.

[XZL+20] Zejun Xiang, Xiangyoung Zeng, Da Lin, Zhenzhen Bao, and Shasha Zhang. Optimizing Implementations of Linear Layers. *IACR Transactions on Symmetric Cryptology*, pages 120–145, 2020.

[ZH22] Chengkai Zhu and Zhenyu Huang. Optimizing the Depth of Quantum Implementations of Linear Layers. In *International Conference on Information Security and Cryptology*, pages 129–147, 2022.

[ZWS+20] Jian Zou, Zihao Wei, Siwei Sun, Ximeng Liu, and Wenling Wu. Quantum Circuit Implementations of AES with Fewer Qubits. In *Advances in Cryptology–ASIACRYPT 2020: 26th International Conference on the Theory and Application of Cryptology and Information Security, Daejeon, South Korea, December 7–11, 2020, Proceedings, Part II 26*, pages 697–726, 2020.

A An example in optimizing sequence

To better explain how to optimize the sequence through solving GI problem, we take a matrix $M \in GL(4, \mathbb{F}_2)$ and try to optimize the number of XOR counts in implementation. Assume the value of M is:

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

Moreover, we get a Matrix Decomposed Ordered Sequence of M through strategy 3-1 or strategy 3-2 in Section 3:

$$s = E(0 \leftrightarrow 1)E(1 \leftrightarrow 2)E(3+0)E(0+2)E(2+1)E(0+2).$$

We first consider using the reduction properties proposed by Xiang et al. to optimize the sequence s. As discussed above, we can ignore the type-1 matrices that consume zero XOR operation. For type-3 subsequence $s' = E(3+0)E(0+2)E(2+1)E(0+2)$ in s, the algorithm iterates all subsequences in s' with length of 3 and length of 2. Furthermore, we have Table 6, which shows that the sequence s' cannot be reduced:

Table 6: All cases of subsequences in s' with length of 3 and length of 2

Case	Problem
$E(0+2)E(2+1)E(0+2)$ $E(3+0)E(0+2)$ $E(0+2)E(2+1)$ $E(2+1)E(0+2)$	Cannot be reduced through Property 2
other cases	Cannot be swapped to adjacent position

Now, we use the method in Section 4 to analyze the sequence s'. In the setting of optimizing the sequence length, we will first let the objective function $f(s) = |s|$ and parameter $n = 4$. Then, the framework will generate all equavalence classes on $GL(4, \mathbb{F}_2)$ with Algorithm 1. Assume we obtain one prime sequence $p = E(1+2)E(0+2)$, and its corresponding value $M_p \in GL(4, \mathbb{F}_2)$:

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

When the algorithm iterates all subsequences in s' with length of 3 and length of 2, it will calculate the corresponding matrix (the first case in Table 6):

$$M_1 = E(0+2)E(2+1)E(0+2) \cdot I = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

It is found that the bipartite graphs $\overline{G_{M_p}}$ and $\overline{G_{M_1}}$ are isomorphic. And with the help of Theorem 2 and GI problem solver, we can get two permutation matrices p_L and p_R such that $M_s = p_L \cdot M_p \cdot p_R$, where:

$$p_L = E(1 \leftrightarrow 2), p_R = E(1 \leftrightarrow 2)$$

Then we have:

$$\begin{aligned} M_1 &= E(0+2)E(2+1)E(0+2) = p_L \cdot p \cdot p_R = E(1 \leftrightarrow 2)E(1+2)E(0+2)E(1 \leftrightarrow 2) \\ &= E(2+1)E(0+1) \end{aligned}$$

After substituting the new M_s sequence with length of 2 into original sequence s, we will get:

$$s = E(0 \leftrightarrow 1)E(1 \leftrightarrow 2)E(3+0)E(2+1)E(0+1).$$

Notice that the optimization process does not use any reduction property.

B The general swapping algorithm

Algorithm 3 Swapping check algorithm $s \in \mathcal{S}^{l,n}$

Input: $indexArray$: the index array we aim to exchange the corresponding element to adjacent location; s: the original sequence;

Output: $\mathsf{flag}_{\mathsf{swap}}$: Whether the element can be swapped together; $index$: The location after swap successfully.

```
 1: initialize nowSeqInterval := 0
 2: initialize $i := indexArray[0]$
 3: initialize leftNum := 0
 4: initialize targetElement := $\{s[indexArray[0]], s[indexArray[1]], ...\}$
 5: initialize elementRight := []
 6: while $i < indexArray[-1]$ do
 7:     if $i \in indexArray$ then
 8:         continue
 9:     end if
10:     # Move $s[i]$ to the left by default
11:     moveDirection := left
12:     for $j \in$ targetElement do
13:         if $s[i]$ cannot swap with $j$ then
14:             moveDirection = right; break
15:         end if
16:     end for
17:     if moveDirection == left then
18:         swap $s[i]$ to left and update
19:         leftNum+ = 1; $i+ = 1$; continue
20:     end if
21:     if moveDirection == right then
22:         if $s[i]$ can swap to right then
23:             $i+ = 1$; continue
24:             append $s[i]$ to elementRight
25:         end if
26:         if $s[i]$ cannot swap to right then
27:             return false
28:         end if
29:     end if
30: end while
31: update $s$ through elementRight
32: index = leftNum + $indexArray[0]$
33: return true
```