IACR Transactions on Symmetric Cryptology
ISSN 2519-173X, Vol. 2020, No. 2, pp. 120–145. DOI:10.13154/tosc.v2020.i2.120-145

Optimizing Implementations of Linear Layers

Zejun Xiang[1], Xiangyong Zeng[1], Da Lin[1], Zhenzhen Bao[2] and Shasha Zhang[1]

[1] Faculty of Mathematics and Statistics, Hubei Key Laboratory of Applied Mathematics, Hubei University, Wuhan, China.
{xiangzejun,xzeng}@hubu.edu.cn,linda@stu.hubu.edu.cn,amushasha@163.com
[2] Division of Mathematical Sciences, School of Physical and Mathematical Sciences, Nanyang Technological University, Singapore, Singapore.
zzbao@ntu.edu.sg

Abstract. In this paper, we propose a new heuristic algorithm to search efficient implementations (in terms of Xor count) of linear layers used in symmetric-key cryptography. It is observed that the implementation cost of an invertible matrix is related to its matrix decomposition if sequential-Xor (s-Xor) metric is considered, thus reducing the implementation cost is equivalent to constructing an optimized matrix decomposition. The basic idea of this work is to find various matrix decompositions for a given matrix and optimize those decompositions to pick the best implementation. In order to optimize matrix decompositions, we present several matrix multiplication rules over $\mathbb{F}_2$, which are proved to be very powerful in reducing the implementation cost. We illustrate this heuristic by searching implementations of several matrices proposed recently and matrices already used in block ciphers and Hash functions, and the results show that our heuristic performs equally good or outperforms Paar's and Boyar-Peralta's heuristics in most cases.

Keywords: Linear Layer · Implementation · Xor Count · AES

1 Introduction

Lightweight cryptography has become one of the main focuses in cryptographic community as the rapid development of lightweight applications, such as Radio-Frequency IDentification (RFID) tags and Internet of Things (IoTs). Generally, lightweight cryptography means low-cost implementable cryptography, where low-cost covers circuit size, energy consumption, latency and so on. Among which the circuit size of a cryptographic algorithm depends on the number of required gates to implement it. In this sense, lightweight cryptography means a cryptographic algorithm with small block and key size and with very efficiently implementable building blocks (e.g., small nonlinear components, and compact linear layer requiring very few or even no Xor gates).

There is already a lot of work focusing on the design of lightweight symmetric-key primitives [BKL+07, GPPR11, BBI+15]. To this end, the designer would favor lightweight components to reduce the cost. For instance, a recursive Maximum Distance Separable (MDS) matrix [DR02] is first adopted in LED [GPPR11] block cipher, which recursively multiply a simpler matrix for generating an MDS matrix; A near MDS matrix with all nonzero elements being 1 is adopted in MIDORI [BBI+15].

The design of lightweight cryptography has inspired the study of lightweight building blocks in the past several years. The Sbox is one of the most important nonlinear components of symmetric-key cryptography, and 4-bit Sbox is a popular choice in the design of lightweight cryptography since its compact implementation in both hardware

Received: 2020-03-02, Accepted: 2020-05-01, Published: 2020-07-24

and software [LP07, Saa11, ZBRL15, JPST17]. Besides, the design of lightweight linear components has also got intensive studies [SKOP15, BKL16, LS16, LW16, LW17, SS16, GLG+17, ZWS18, LSL+19], with special attention paid on MDS matrices since an MDS matrix has an optimal branch number [Dae95] and provides a decent lower bound on the minimal number of active S-boxes within a few rounds. Another line of work focuses on optimizing implementations of linear matrices over $\mathbb{F}_2$ [JPST17, BMP13, Paa97, TP20, BFI19, ME19]. As the goal of this line of work is to find a smaller number of XOR gates needed to implement a linear layer, this is of more practical significance. For example, the Advanced Encryption Standard (AES) [DR02] has been widely used in practice, and its round function has been frequently used in the design of other cryptographic primitives (such as AEGIS [WP13] and ForkAES [ARVV18]), thus an implementation of its linear layer with a smaller XOR count will directly reduce the cost of deploying AES and the primitives that uses its round function. This paper follows this line of work and offers an alternative heuristic to search optimized implementations of linear matrices.

Since the linear layer of a symmetric-key primitive can be represented as a binary matrix, the implementation of a linear layer is a sequence of XOR operations. Thus, the implementation cost of a linear layer can be estimated by the number of XOR operations required to implement the corresponding binary matrix. There are generally three metrics discussed in the literature, i.e., d-XOR, s-XOR, and what we called g-XOR in this paper. Generally, the direct-XOR (d-XOR) metric counts the number of 1's in a matrix, and a lightweight matrix is thus a sparse matrix under this metric. The sequential-XOR (s-XOR) metric counts the number of XOR instructions $x_i = x_i \oplus x_j$ needed to transform the inputs to the outputs. In this paper, we refer to general-XOR (g-XOR) as a metric counting the number of operations $t_i = t_j \oplus t_k$ that required to compute the outputs from the given inputs. Note that the difference between s-XOR and g-XOR is that s-XOR counts self-update XOR operations, i.e., the first input bit will be rewritten as the XOR of the two input bits, while the g-XOR uses another variable to store the result. Though d-XOR is intuitive and easy to compute, it is not sufficient to use d-XOR to measure the implementation cost of a matrix [JPST17, KLSW17, DL18], since implementing a linear matrix under d-XOR metric may compute the same intermediate value several times. However, determining the optimal implementation under s-XOR or g-XOR metric is a hard problem.

In fact, finding the minimal g-XOR count corresponds to the Shortest Linear Program (SLP) problem which has been proved by Boyar *et al.* to be an NP-hard problem. As pointed out in [Köl19], determining the optimal s-XOR count is related to the problem of optimal pivoting in Gauss-Jordan elimination, as the number of additions in an optimal elimination process is an upper bound of the s-XOR count. Currently, there are no efficient algorithms to compute the optimal g-XOR or s-XOR count for a given matrix with a large size, say 32×32 matrices. However, a lot of work has been focusing on this topic and tried to reduce this gap. Paar's [Paa97] and Boyar-Peralta's [BP10] heuristics are the two most discussed methods to search optimized implementations of linear matrices under g-XOR metric. In [JPST17], the authors presented an exhaustive search algorithm to determine the optimal s-XOR count of small-scale matrices. Following this line of work, we will present in this paper a new heuristic to search optimized implementations under s-XOR metric for reasonable large matrices.

Our Contribution. In this paper, we propose a new heuristic which can be used to find efficient implementations of binary matrices of size up to 32 under s-XOR metric. The s-XOR count measures the number of *in-place* [JPST17] operations $x_i = x_i \oplus x_j$ required to transform the inputs to the outputs, and each operation is equivalent to performing a row addition of the corresponding matrix [Köl19] which will result in an identity (or a permutation) matrix eventually. As each row addition can be simulated by a left-multiplication of an elementary matrix, the problem of determining a smaller s-XOR

count of an invertible matrix M is reduced to deriving a matrix decomposition of M with as fewer type-3 elementary matrices as possible.

To this end, we present seven rules of elementary matrix multiplications which can help reduce the number of type-3 elementary matrices for a given matrix decomposition, with the restriction that all matrices are over $\mathbb{F}_2$. These rules are built on the fact that XORing the same two bits will cancel each other, and we could save one XOR operation. Among all the seven rules, six of which specify the cases when three consecutive XOR operations can be reduced to two XOR operations, and one of which specifies the case when two consecutive XOR operations can be reduced to one XOR operation.

With the help of these rules, a new heuristic is designed to search optimized implementations of matrices. This heuristic can be roughly divided into two steps. The first step is to decompose the given matrix into a product of elementary matrices, and the second step is to build a lot of equivalent decompositions to be optimized by iteratively applying the seven rules.

To compare our heuristic with the previous ones, we implemented our heuristic and experimented on several matrices. The results show that our heuristic can find equally good or better implementations in most cases compared with those obtained using Paar's and Boyar-Peralta's heuristics. For instance, we find an implementation of M_{AES} (the MixColumns matrix of AES) using 92 s-XOR operations, which performs equally good as the previous best result of M_{AES} under g-XOR metric [Max19]. Moreover, an implementation under s-XOR metric is a sequence of *in-place* instructions. Therefore, on platforms equipped with invariably 2-operand instructions, our implementation is quite straightforward for compiling since they need no extra auxiliary registers and additional copy instructions. Thus, our heuristic enjoys an extra advantage that it can be very friendly for bit-sliced implementations of linear layers on software. Similar reasoning deduces the advantage of our *in-place* instructions for the quantum implementations of linear layers in terms of the number of qubits and gates needed. Also, since an invertible matrix and its inverse share the same matrix decomposition length (in terms of the number of type-3 elementary matrices) [BKL16, Köl19], they have the same implementation cost under s-XOR metric. As a direct application, the inverse MixColumns matrix used in AES can be implemented with 92 XOR's.

All the source code of this paper is available at https://github.com/xiangzejun/Optimizing_Implementations_of_Linear_Layers.

Structure of the Paper. The paper is structured as follows. Section 2 presents some backgrounds and introduces related work. Strategies to obtain matrix decompositions adopted in our heuristic are presented in Sect. 3. Section 4 proposes several rules to help optimize a matrix decomposition. Section 5 describes our heuristic search algorithm in detail, and the applications of this heuristic to a large set of matrices. At last, Section 6 discusses and concludes the paper.

2 Preliminaries

We first present some notations used throughout the paper.

2.1 Backgrounds

Elementary Operations. There are three types of operations that are called elementary operations in matrix theory, which are

1. Interchange two rows (columns), which will be referred to as type-1 elementary operations.

$\mathbb{F}_2$	the finite field with two elements 1 and 0.
$\mathbb{F}_{2^s}$	the finite field with 2^s elements.
$\mathbb{F}_2^l$	the vector space of all l-dimensional vectors over $\mathbb{F}_2$.
$\mathbb{F}_{2^s}^l$	the vector space of all l-dimensional vectors over $\mathbb{F}_{2^s}$.
$\mathrm{GL}(n, \mathbb{F}_2)$	all n-by-n invertible matrices over $\mathbb{F}_2$.
$\mathrm{GL}(n, \mathbb{F}_2)^{l\times l}$	all l-by-l matrices with their elements being matrices in $\mathrm{GL}(n, \mathbb{F}_2)$.
I_n	the n-by-n identity matrix over $\mathbb{F}_2$.
I	the identity matrix over $\mathbb{F}_2$ if the order is clear from the context.
$E(i \leftrightarrow j)$	the resulting matrix by exchanging the ith and jth row of an identity matrix (a type-1 elementary matrix).
$E(i + j)$	the resulting matrix by adding the jth row to the ith row of an identity matrix (a type-3 elementary matrix).

2. Multiply a row (column) with a nonzero number, which will be referred to as type-2 elementary operations.
3. Add a row (column) to another one multiplied by a nonzero number, which will be referred to as type-3 elementary operations.

Thus, concerning both row and column operations, there are in total six types of elementary operations.

Elementary Matrix and Matrix Decomposition. If an elementary operation is performed on an identity matrix, the resulting matrix is called an elementary matrix. Thus, there are three types of elementary matrices, referred to as type-1, type-2 and type-3, which correspond to the three types of elementary operations. Since elementary operations are invertible transformations, all elementary matrices are invertible matrices. Moreover, the inverse of an elementary matrix is also an elementary matrix. Note that performing an elementary operation on matrix M can be simulated by left-multiplying or right-multiplying the corresponding elementary matrix. For example, adding the jth row to the ith row of M can be simulated as left-multiplying M by $E(i + j)$ and adding the ith column of M to the jth column can be simulated as right-multiplying M by $E(i + j)$.

Theorem 1 (Theorem 1.2.16 in [Art11])**.** *Any invertible matrix can be transformed into an identity matrix using elementary row and/or column operations. Thus, any invertible matrix can be decomposed as a product of elementary matrices.*

Linear Algebra in $\mathbb{F}_2$. Since the only nonzero element in $\mathbb{F}_2$ is 1, if we consider a matrix in $\mathrm{GL}(n, \mathbb{F}_2)$, type-2 elementary operations make the matrix unchanged and type-3 elementary operations are equivalent to adding a row (or a column) to another. Therefore, the type-2 elementary operations are not concerned in the rest of the paper.

Corollary 1. *Any matrix in $GL(n, \mathbb{F}_2)$ can be transformed into an identity matrix by applying a series of type-1 and type-3 elementary row and/or column operations. Thus, any matrix in $GL(n, \mathbb{F}_2)$ can be decomposed as a product of type-1 and type-3 elementary matrices.*

It is clear that the matrix multiplication does not satisfy the commutative law, i.e., $M_1M_2 \neq M_2M_1$ for two general matrices M_1 and M_2. However, we could still study commutative properties of some special elementary matrices over $\mathbb{F}_2$ to help find a matrix decomposition in the following sections. Given two elementary matrices $E(i + j)$ (type-3) and $E(k \leftrightarrow l)$ (type-1), we have the following property.

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

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

Note that Property 1 was presented in [Köl19] in a slightly different form (Equation (1) of [Köl19]). Property 1 reveals that the multiplication between a type-1 elementary matrix and a type-3 elementary matrix in $\mathrm{GL}(n, \mathbb{F}_2)$ is commutative up to a modification on the type-3 elementary matrix. According to Corollary 1, any matrix M in $\mathrm{GL}(n, \mathbb{F}_2)$ can be decomposed as a product of type-1 and type-3 elementary matrices. Moreover, according to Property 1, we can always rearrange the order of elementary matrices (up to modifications on the type-3 elementary matrices). Accordingly, we have the following theorem.

Theorem 2. *Any matrix M in $GL(n, \mathbb{F}_2)$ can be decomposed as:*

$$M = E(i_t + j_t) \cdots E(i_1 + j_1)E(i'_s \leftrightarrow j'_s) \cdots E(i'_1 \leftrightarrow j'_1). \tag{1}$$

Xor metrics. We present three metrics used in the literature to evaluate the implementation cost of a linear matrix. The d-Xor metric was first introduced in [KPPY14]. It counts the number of 1's in a matrix as a measurement of lightweight diffusion layer.

Definition 1 (d-Xor [KPPY14])**.** Given an $m \times n$ matrix $M_{m\times n}$ over $\mathbb{F}_2$, the d-Xor count is defined as $\mathrm{wt}(M_{m\times n}) - m$, where $\mathrm{wt}(M_{m\times n})$ is the Hamming weight of $M_{m\times n}$, i.e., the number of 1's in M.

A lightweight matrix is a matrix with very sparse nonzero elements under d-Xor metric. Since this metric is intuitive and easy to compute, it has been adopted in the design of new lightweight diffusion layers [SKOP15, LS16, LW16, SS16]. However, the d-Xor metric corresponds to the very naive implementation of a linear matrix, i.e., implementing the linear matrix according to its algebraic expressions. This may compute an intermediate value several times, thus resulting in an overestimation of the Xor operations needed.

Definition 2 (g-Xor)**.** Given a linear matrix $M_{m\times n}$ over $\mathbb{F}_2$, each row of M corresponds with a linear expression defined on the n inputs. The implementation of M can be viewed as a sequence of Xor operations $x_i = x_{j_1} \oplus x_{j_2}$, where $0 \le j_1, j_2 < i$ and $i = n, n+1, ..., t-1$. $x_0, x_2, ..., x_{n-1}$ are the n inputs and the m outputs are a subset of all x_i's. The g-Xor count is defined as the minimal number of such operations $x_i = x_{j_1} \oplus x_{j_2}$ that completely compute the m outputs.

Note that the g-Xor metric has been used in several heuristics [Paa97, BMP13, LSL+19]. Finding the optimal implementation of a linear matrix under g-Xor metric has been proved to be an NP-hard problem, i.e., the Shortest Linear Program (SLP) problem by Boyar *et al.* in [BMP13]. As a result, there are no efficient algorithms to find the optimal implementation under this metric for large linear matrices (e.g., 16×16 and 32×32 matrices) used in symmetric-key cryptography. Another metric used in the literature is the s-Xor count which was introduced in [JPST17].

Definition 3 (s-Xor [JPST17])**.** Let $M \in \mathrm{GL}(n, \mathbb{F}_2)$ be an invertible matrix. Assume $x_0, x_1, ..., x_{n-1}$ are the n input bits of M. It is always possible to perform a sequence of Xor instructions $x_i = x_i \oplus x_j$ with $0 \le i, j \le n-1$, such that the n input bits are updated to the n output bits. The s-Xor count of M is defined as the minimal number of Xor instructions to update the inputs to the outputs.

Clearly, the d-XOR count of a matrix is always larger than or equal to its g-XOR count. Similarly, the s-XOR count is larger than or equal to the g-XOR count, as we can easily transform the *in-place* XOR instruction $x_i = x_i \oplus x_j$ to an *out-of-place* instruction $t_k = x_i \oplus x_j$ by introducing a new variable t_k. It was conjectured that the s-XOR count is smaller than or equal to the d-XOR count in [JPST17]. However, Lukas disproved this conjecture by presenting a counterexample in [Köl19]. Although, the g-XOR count is the smallest one among the three metrics theoretically, it is not always easy to find the best implementations under the g-XOR metric. Our heuristic reveals that it is possible to find better implementations under the s-XOR metric, and this may attribute to the properties of s-XOR detected in this paper. Moreover, the *in-place* feature of the XOR instructions makes the s-XOR metric enjoy an extra advantage that it never uses temporary registers to store intermediate values, and this feature may be friendly when considering a bit-sliced software implementation. Concretely, for the instruction set architecture (ISA) of many platforms, typically the micro-controllers of RISC architectures that have 16-bit instructions, such as Atmel AVR, TI MSP430, and some versions of ARM Thumb, the CPU has invariably 2-operand instructions. For 2-operand instructions, the destination register is one of the source registers, and in a format of '`operator destination/source`$_1$, `source`$_2$'. Thus, a logic computation, e.g., $t = x_1 \oplus x_2$, needs at least two instructions and three registers. That is, first '`move` t, x_1'; then '`xor` t, x_2'. Therefore, on these platforms, *in-place* implementation, e.g., $x_1 = x_1 \oplus x_2$, is quite straightforward for compiling and requires no additional `move` instructions and extra registers. Besides, *in-place* implementations can be more friendly in terms of the G-cost metric for implementing the algorithm using quantum circuits, where G-cost measures the total number of gates required. Apart from G-cost, the considered metrics for quantum implementation includes the circuit depth and width (DW-cost). For example, for the quantum implementation of the MixColumns of AES in [JNRV19], Jaques *et al.* translated an *in-place* implementation into a quantum circuit with 1108 controlled-NOT (CNOT) gates, full circuit depth of 111, and a circuit width of 128. Whereas, the authors translated the 92-XOR-gate out-of-place implementation presented in [Max19] into a quantum circuit with 1248 CNOT gates, full circuit depth of 22, and a circuit width of 318. From these experiments, the authors concluded that for a depth restricted setting, Maximov's low-depth version seems better. However, without the depth restriction, it is advantageous to use the *in-place* version to minimize both G-cost and DW-cost metrics.

2.2 Related Work

We give an overview of several heuristics for searching optimized implementations in this subsection.

Since it is impractical to always find the optimal implementations under g-XOR and/or s-XOR metrics for large linear layers, a lot of efforts have been paid on optimizing building blocks of linear layers. In the case of an MDS matrix $M_{l\times l}$ whose entries are over $\mathbb{F}_{2^s}$ or $\mathrm{GF}(2, \mathbb{F}_2)$, computing each element of the output vector $y = Mx$ requires the computations of l multiplications and $(l-1)$ additions of s-tuples (as each entry in an MDS matrix is nonzero). As $(l-1)$ additions of s-tuple require $(l-1)s$ XOR operations, this part of the cost is treated as fixed [KPPY14], and spare efforts are paid to optimize the multiplication when designing lightweight linear diffusion layer [SKOP15, LS16, LW16, BKL16, SS16, LW17, JPST17, LSL+19]. To optimize the implementation of the multiplication, one can try to use a different basis if the matrix entries are finite filed elements, or reuse the intermediate values. All these efforts are the so-called local optimization and they treat the costs of additions of s-tuples as fixed. For instance, if we consider the matrix M_{AES}, the fixed cost is $4 \times 3 \times 8 = 96$ XOR operations. However, the best-known result only needs 92 XOR operations [Max19] after being globally optimized. This implies if only local optimization is performed on M_{AES}, the fixed cost (96 XOR operations) is larger than the

globally optimized cost (92 XOR operations). This is indeed the motivation of [KLSW17] where Kranz *et al.* applied several existing global optimization heuristics to optimize the implementations of a large set of already known and newly designed matrices.

One of the global optimization heuristics discussed in [KLSW17] was proposed by Paar in [Paa97], where the author presented two algorithms to optimize any matrix over $\mathbb{F}_2$. Given a matrix $M_{m\times n}$ over $\mathbb{F}_2$ and an input vector $x = (x_0, x_1, ..., x_{n-1})$, the output $y = Mx$ contains m linear functions. Paar's algorithm first computes the number of occurrence of all $x_i + x_j (i \neq j = 0, 1, ..., n-1)$ in all the m linear functions and replace the maximum occurrence by a new variable x_n. After this step, those m linear functions are defined on $n+1$ variables, and the problem is reduced to implementing an $m \times (n+1)$ matrix. This new $m \times (n+1)$ matrix is processed similarly in the next steps until the m linear forms are completely computed. The core parts of Paar's two algorithms are the same, with the only difference being that they apply different strategies when there are multiple $(x_i + x_j)'s$ that occur the same maximum time. Paar's first algorithm which is denoted as Paar1 chooses the maximum occurrence that appears first, while the second algorithm which is denoted as Paar2 checks all possibilities. Thus, the algorithm Paar2 is more time-consuming with the advantage that it may find better implementations. Notably, Paar's algorithms are cancellation-free, i.e., the cancellation of two same input bits is out of the range of Paar's heuristic and this somehow limits the use case of the two algorithms.

Another discussed heuristic was proposed by Boyar and Peralta in [BP10], where the authors keeps a record of a so-called distance vector whose ith coordinate is defined as the minimal number of XOR operations required to compute the ith linear function based on a base of known functions. Initially, the n input variables are set as the initial base and the ith coordinate of the distance vector equals to the Hamming weight of the ith row of M minus one. Then Boyar-Peralta's heuristic proceeds by XORing two existing linear functions that minimize the sum of the new distance vector. Boyar-Peralta's heuristic has lots of variants [BFI19, TP20, ME19] that differ in tie-breaking phase, i.e., when there are multiple choices that minimize the sum of the new distance vectors at the same time. Boyar-Peralta's heuristic is not quite efficient on dense matrix since the step choosing two existing linear functions must try all possibilities to compute the new distance vectors. Later in [VSP18], Visconti *et al.* extended Boyar-Peralta's heuristic to the applications of dense matrices based on an observation that $\bar{M}$ (the complement of M) is sparse if M is dense. Thus, if we can efficiently implement $\bar{M}$, M can be implemented by XORing each row of $\bar{M}$ with a common bit which is the sum of all input bits.

Note that both Paar's and Boyar-Peralta's (and its variants) heuristics are based on g-XOR metric. In [JPST17], the authors proposed to use s-XOR count to measure the cost of implementing linear transformations over $\mathbb{F}_2^s$. The *in-place* update used in s-XOR is an invertible operation, and this has facilitated the meet-in-the-middle (MITM) search algorithm presented in [JPST17] to find optimal implementations of small invertible matrices. The MITM search algorithm starts by enumerating all possible operations $x_i = x_i \oplus x_j$ and storing the resulting linear functions as nodes of a graph with original inputs being its root node. At the same time, the search algorithm enumerates all possible operations $y_k = y_k \oplus y_l$ and store the resulting linear functions as nodes of another graph with original outputs being its root node. These two graphs are expanded in a breadth-first way until the shortest path connecting the two roots through a matching is found. This algorithm is essentially an exhaustive search, and it indeed finds the optimal implementation of a matrix under s-XOR metric. However, this search algorithm can only be applied to small linear matrices due to its large memory consumption. Thus, this algorithm is only used to optimize finite field multiplications with constants (since the corresponding matrices are always small-scale in practical applications), and based on these optimized finite field multiplications, the authors can find some new lighter MDS matrices and improve the implementations of several previous MDS matrices.

Bernstein proposed in [JB09] an algorithm to optimize linear maps modulo 2. His heuristic is to recursively eliminate the largest row in reverse lexicographic order of a matrix by the second largest row until each column of the matrix contains at most one nonzero element, then this whole elimination process is converted to a sequence of instructions. Bernstein's elimination is essentially performing elementary operations on the target matrix. Motivated by Bernstein's and Jean *et al.*'s work, we propose a new heuristic under s-XOR metric that can be used to optimize linear matrices with size up to 32.

3 Matrix Decomposition in $\mathbf{GL}(n, \mathbb{F}_2)$

As showed in Sect. 2.1, any matrix in $\mathrm{GL}(n, \mathbb{F}_2)$ can be decomposed as a product of type-1 and type-3 elementary matrices. In this section, we present three methods to decompose any given matrix in $\mathrm{GL}(n, \mathbb{F}_2)$.

3.1 Elementary Row (Column) Operation Based Matrix Decomposition

Note that any invertible matrix in $\mathrm{GL}(n, \mathbb{F}_2)$ can be transformed into an identity matrix by a row reduction (see Sect. 1.2 of [Art11]), which can be simulated as left-multiplying the corresponding elementary matrices. Thus, we can find a series of type-1 and type-3 elementary matrices $E_1 E_2 \cdots E_{s+t}$ for any given matrix M in $\mathrm{GL}(n, \mathbb{F}_2)$, such that $E_1 E_2 \cdots E_{s+t} M = I$. Where E_k is a type-1 or type-3 elementary matrix, and there are s type-1 elementary matrices and t type-3 elementary matrices among $E_1 E_2 \cdots E_{s+t}$. Moreover, we can rearrange the order of $E_1 E_2 \cdots E_{s+t}$ according to Property 1 and get

$$E(i'_1 \leftrightarrow j'_1) \cdots E(i'_s \leftrightarrow j'_s) E(i_1 + j_1) \cdots E(i_t + j_t) M = I.$$

Thus,

$$M = E(i_t + j_t)^{-1} \cdots E(i_1 + j_1)^{-1} E(i'_s \leftrightarrow j'_s)^{-1} \cdots E(i'_1 \leftrightarrow j'_1)^{-1}. \tag{2}$$

Note that elementary matrices in $\mathrm{GL}(n, \mathbb{F}_2)$ are involutions, i.e., $E(j + i)^{-1} = E(j + i)$ and $E(i \leftrightarrow j)^{-1} = E(i \leftrightarrow j)$. Therefore, Equation (2) can be simplified as

$$M = E(i_t + j_t) \cdots E(i_1 + j_1) E(i'_s \leftrightarrow j'_s) \cdots E(i'_1 \leftrightarrow j'_1),$$

which is consistent with the matrix decomposition form presented in Sect. 2.1. In the rest of this paper, we will refer to this matrix decomposition method as **Strategy 1**.

Similar to the elementary row operation based matrix decomposition, any matrix in $\mathrm{GL}(n, \mathbb{F}_2)$ can be transformed into an identity matrix by a column reduction (i.e., elementary column operations), and we will refer to the method using elementary column operations as **Strategy 2**.

3.2 Hybrid Elementary Operation Based Matrix Decomposition

We present in this subsection a matrix decomposition method that uses both elementary row and column operations. The motivation of this strategy lies in that the implementation cost of a matrix depends only on the number of type-3 elementary matrices in its matrix decomposition (this will be explained in Sect. 4). Note that the main role of type-3 elementary matrices is to eliminate 1's in a given matrix such that it can be transformed into an identity matrix in the end. Thus, we try to reduce as many 1's as possible each time when performing an elementary operation. Specifically, we consider all possible type-3 elementary row and column operations in each step, and choose the one that reduces the most number of 1's in the resulting matrix. However, this may cause two problems.

1. In each step, there may exist multiple choices of elementary operations that all minimize the number of 1's in the resulting matrices. When this happens, we randomly pick one of the best choices rather than choosing the one that appears first. This strategy may help us find better implementations since we can run the procedure multiple times and choose the best one. Note that traversing all candidate elementary operations in each step will cause the procedure prohibitively slow, and deterministically choosing one candidate will prohibit any chance to find better solutions. By allowing randomness, we obtain more flexibility to make the trade-off between the expandable time and the quality of the solutions.

2. In an intermediate step, it is possible that performing every elementary operation will result in a matrix of which the number of 1's is not reduced. In that case, without additional strategy, the procedure may fall into an infinite loop (see Example 1). We adopt two strategies to solve this problem. The first strategy is to turn to apply **Strategy 1** and the second one is to turn to apply **Strategy 2** for the following procedure. In the rest of this paper, we will refer to these two strategies as **Strategy 3-1** and **Strategy 3-2** respectively.

Example 1. The following 6×6 matrix M will trigger the second problem. The number of 1's contained in this matrix is 14, and no matter which elementary row or column opration is performed on M, the number of 1's in the resulting matrix will be greater than or equal to 14, i.e., the best choice in this step can reduce zero 1's.

$$M = \begin{pmatrix} 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 \end{pmatrix}.$$

Assuming that the fourth (the rows being numbered from 1 to 6) row is added to the third row in this step, we will get M_1 containing 14 1's as listed in the following.

$$M_1 = \begin{pmatrix} 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 \end{pmatrix}.$$

Again, no matter which elementary operation is performed on M_1, the resulting matrix contains greater than or equal to 14 1's. If we happen to choose adding the fourth row to the third row in this step, M_1 will be transformed back to M, thus leading to an infinite loop.

Since both elementary row and column operations are used to process the matrix in Strategy 3-1 and Strategy 3-2, we can find the corresponding elementary matrices $E_1 E_2 \cdots E_k$ and $E'_1 E'_2 \cdots E'_l$ such that

$$E_1 E_2 \cdots E_k M E'_1 E'_2 \cdots E'_l = I.$$

Thus,

$$\begin{aligned} M &= E_k^{-1} \cdots E_2^{-1} E_1^{-1} E_l'^{-1} \cdots E_2'^{-1} E_1'^{-1} \\ &= E_k \cdots E_2 E_1 E'_l \cdots E'_2 E'_1 \end{aligned}$$

After rearranging the order of $E_k \cdots E_2 E_1 E'_l \cdots E'_2 E'_1$, we can get that

$$M = E(i_t + j_t) \cdots E(i_1 + j_1) E(i'_s \leftrightarrow j'_s) \cdots E(i'_1 \leftrightarrow j'_1),$$

where $k + l = s + t$. Appendix A presents a small example to illustrate the process of this strategy.

4 Reducing the Implementation Cost of Linear Matrices

In this section, we first introduce how to convert a decomposition to an implementation for a given matrix, then we present several properties of elementary matrix multiplication in $\mathrm{GL}(n, \mathbb{F}_2)$ which are exploited to reduce the implementation cost.

4.1 Converting Matrix Decomposition to Matrix Implementation

Let $M \in \mathrm{GL}(n, \mathbb{F}_2)$ be a given invertible matrix. According to Sect. 3, M can be decomposed as

$$M = E(i_t + j_t) \cdots E(i_1 + j_1) E(i'_s \leftrightarrow j'_s) \cdots E(i'_1 \leftrightarrow j'_1).$$

In order to implement the linear transformation defined by M, we have to design a circuit to compute the output $y = Mx$ for any given input $x \in \mathbb{F}_2^n$. Thus, it is equivalent to implement $y = E(i_t + j_t) \cdots E(i_1 + j_1) E(i'_s \leftrightarrow j'_s) \cdots E(i'_1 \leftrightarrow j'_1)x$. Note that the result of the matrix-vector multiplication $E(i'_1 \leftrightarrow j'_1)x$ is just an exchange of the two bits $x_{i'_1}, x_{j'_1}$ of x. Exchanging two bits is a hard-wired circuit in hardware implementation which has a negligible cost. However, this is not entirely free in software implementation. But we can hard code type-1 elementary operations, then a shuffle among variables is appended. For large matrices, this final shuffle should take only a small fraction of the total cost (also much lower than the hidden cost in out-of-place implementation). Thus, the implementation cost of $E(i'_s \leftrightarrow j'_s) \cdots E(i'_1 \leftrightarrow j'_1)x$ is omitted. Let $x' = E(i'_s \leftrightarrow j'_s) \cdots E(i'_1 \leftrightarrow j'_1)x$, and the overall cost of M comes from the implementation of $E(i_t + j_t) \cdots E(i_1 + j_1)x'$.

Recall that $E(i + j)$ is a type-3 elementary matrix. Let's consider the case of matrix-vector multiplication $E(i_1 + j_1)x'$. This will result in a vector of which the i_1th bit is the XORing of the i_1th and j_2th bits of x' and other bits are the same as x''s. Therefore, implementing $E(i_1 + j_1)x'$ requires one XOR operation, and $E(i_t + j_t) \cdots E(i_1 + j_1)x'$ needs an overall cost of t XOR operations. From this point of view, the overall implementing cost of M depends on the number of type-3 elementary matrices in the matrix decomposition of M, and optimizing the implementation of a matrix is equivalent to search a matrix decomposition with as fewer type-3 elementary matrices as possible.

Since $E(i'_s \leftrightarrow j'_s) \cdots E(i'_1 \leftrightarrow j'_1)$ has a negligible implementation cost, we will assume in the rest of this paper that M can be decomposed as $M = E(i_t + j_t) \cdots E(i_1 + j_1)$, i.e., the type-1 elementary matrices are omitted for the sake of simplicity.

4.2 Reducing Implementing cost

In this subsection, we propose several rules of matrix multiplication in $\mathrm{GL}(n, \mathbb{F}_2)$ which can be used to efficiently reduce the number of type-3 elementary matrices in the matrix decomposition.

Property 2. Let $E(i \leftrightarrow j)$ and $E(i + j)$ denote a type-1 and type-3 elementary matrices in $\mathrm{GL}(n, \mathbb{F}_2)$ respectively, then the following equations hold.

R1 $E(k + i)E(k + j)E(i + j) = E(i + j)E(k + i)$,

R2 $E(i + k)E(k + j)E(i + j) = E(k + j)E(i + k)$,

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

The proof of Property 2 is presented in Appendix B. All the above seven rules are built on the fact that XORing the same two bits will cancel each other, thus we can remove one type-3 elementary matrix when two bits canceling each other. Each of the seven rules can reduce one type-3 elementary matrix, thus reduce the implementation cost of linear layers. This inspires us to identify as many patterns matching Property 2 as possible for a given matrix decomposition. However, it requires to identify three (R1-R6) or two (R7) consecutive elementary matrices in the matrix decomposition to match the patterns presented in Property 2, which limits the optimization of the whole matrix in the fact that consecutive elementary matrices satisfying those patterns are far from enough to derive a good implementation.

However, there exist a lot of matrices satisfying those patterns in Property 2 which are separated by some other elementary matrices and thus not adjacent. As is known that matrix multiplication does not satisfy the commutative law generally, we can not simply interchange the positions of elementary matrices to make them adjacent. However, we can still find some special cases when the commutative law holds for the elementary matrix multiplication in $(n, \mathbb{F}_2)$.

Property 3. Let i, j, k, l be integers and $i \neq j \neq k \neq l$, then we have

1. $E(k+l)E(i+j) = E(i+j)E(k+l)$,
2. $E(i+j)E(k+j) = E(k+j)E(i+j)$,
3. $E(i+j)E(i+k) = E(i+k)E(i+j)$.

Property 3 presents three cases when the multiplications of elementary matrices are commutative. Thus, if the identified match patterns are not adjacent, Property 3 can be used to interchange the positions of elementary matrices and try to make the match pattern adjacent, and thus are exploited to reduce the implementation cost. The whole procedure to reduce the number of type-3 elementary matrices using the rules listed in Property 2 is presented in Algorithm 1. An example is also listed here to illustrate the usage of Property 2.

Example 2. Let $M = E(3{+}2)E(3{+}4)E(3{+}1)E(2{+}1)$, i.e., M is decomposed as a product of four elementary matrices. Each of the three consecutive matrices $E(3{+}2)E(3{+}4)E(3{+}1)$ and $E(3{+}4)E(3{+}1)E(2{+}1)$ does not match the patterns presented in Property 2. However, it can be observed that the multiplication between $E(3+2)$ and $E(3+4)$ is commutable according to Property 3. Thus, M can be rewritten as $M = E(3{+}4)E(3{+}2)E(3{+}1)E(2{+}1)$ and the last three consecutive matrices satisfy the pattern R1 presented in Property 2. As a result, M can be decomposed as $M = E(3+4)E(2+1)E(3+2)$, based on which the implementation cost of M can be reduced by one XOR operation.

5 Search Algorithm and Applications

5.1 Heuristic Search Algorithm

For any given matrix M in $\mathrm{GL}(n, \mathbb{F}_2)$, an initial matrix decomposition can be obtained using one of the four strategies (Strategy 1, 2, 3-1 and 3-2) introduced in Sect. 3. Then this

Algorithm 1 Reduce Matrix Decomposition

Input: The matrix decomposition $seq = E(i_t + j_t) \cdots E(i_1 + j_1) \triangleq E_t \cdots E_1$ for a given matrix M;

Output: Reduced Decomposition **Reduce**(seq) of M;

$flag \leftarrow True$;
$l = t$;
while $flag$ **do**
    $flag \leftarrow False$;
    for all possible combinations E_a, E_b, E_c with $l \geq a > b > c \geq 1$ in seq **do**
        if $E_aE_bE_c$ matches one of the patterns R1-R6 in Property 2 **then**
            if E_a, E_b, E_c can be adjacent by swapping according to Property 3 **then**
                $seq \leftarrow$ rewrite the decomposition to let E_a, E_b, E_c be adjacent;
                reduce seq according to R1-R6, and update seq;
                $l \leftarrow l - 1$;
                $flag \leftarrow True$;
                Break;
            end if
        end if
    end for
    for all possible combinations E_a, E_b with $l \geq a > b \geq 1$ in seq **do**
        if E_aE_b matches the pattern R7 in Property 2 **then**
            if E_a, E_b can be adjacent by swapping according to Property 3 **then**
                $seq \leftarrow$ rewrite the decomposition to let E_a, E_b be adjacent;
                reduce seq according to R1-R6, and update seq;
                $l \leftarrow l - 1$;
                $flag \leftarrow True$;
                Break;
            end if
        end if
    end for
end while
return seq;

initial decomposition can be optimized by Algorithm 1. Clearly, different strategies result in different initial decompositions, which further lead to different optimized implementations. Our experiment results show that the hybrid strategies, i.e., Strategy 3-1 and Strategy 3-2, are better (in terms of the XOR count of the optimized implementation) than Strategy 1 and Strategy 2 in most cases, and this might attribute to the shorter initial decompositions of Strategy 3-1 and Strategy 3-2. Thus, Strategy 3-1 and Strategy 3-2 will be adopted in our search algorithm. Moreover, the differences between these four strategies reveal that the specific form of a matrix decomposition has a great impact on the resulting optimized implementation. This has motivated the design of a heuristic search algorithm presented in Algorithm 2.

Intuitively, there are only four matrix decomposition strategies presented in Sect. 3 which result in four equivalent matrix decompositions. Thus, the best implementation is the one with fewest type-3 elementary matrices after being reduced by Algorithm 1. However, we present a method here to generate far more than four equivalent matrix decompositions, such that the best implementation is chosen from a large set of reduced matrix decompositions rather than four. This has the advantage that a better implementation may be obtained. The core idea of this method is to pick a segment of matrices from the decomposition of a given matrix, then replace this segment by another equivalent segment, thus a new decomposition is obtained.

Specifically, we denote the decomposition of M as $M = E(i_t + j_t) \cdots E(i_1 + j_1) \triangleq E_t E_{t-1} \cdots E_1$, where E_i is the abbreviation of a type-3 elementary matrix. Our heuristic search algorithm starts by choosing g (g is set to t initially) consecutive elementary matrices from $E_t E_{t-1} \cdots E_1$ which is denoted as $E_{g+u}, E_{g-1+u}, \cdots, E_{1+u}$ $(0 \leq u \leq t-g)$, thus $M = E_t \cdots E_{g+u+1} E_{g+u} \cdots E_{1+u} E_u \cdots E_1$. Then a new matrix M' is computed as $M' = E_{g+u} E_{g-1+u} \cdots E_{1+u}$ and we decompose M' using Strategy 3-1 or 3-2 as $M' = E'_v E'_{v-1} \cdots E'_1$. Therefore, we can get an equivalent matrix decomposition of M as

$$M = E_t \cdots E_{g+u+1} E'_v \cdots E'_1 E_u \cdots E_1. \tag{3}$$

Then, this new decomposition is reduced by Algorithm 1. If the reduced matrix decomposition has t' type-3 elementary matrices with $t' \geq t$, we turn to choose another g consecutive elementary matrices from $M = E_t E_{t-1} \cdots E_1$. If all choices of possible g consecutive elementary matrices are tried and all get worse results, we set $g = g - 1$ and repeat the above procedure. If instead $t' < t$, we update the matrix decomposition of M by this new reduced result and repeat the above procedure by setting $g = t'$.

The most time-consuming part of this heuristic is to pick out a segment of matrices from the decomposition of M and replace this part by another segment. Since Algorithm 2 adopts Strategy 3-1 or Strategy 3-2 for matrix decompositions and these two strategies randomly pick an elementary operation if there are multiple choices that are equally good, the search algorithm will output different results each time when it is called. Thus, we can run the procedure several times and return the best result.

5.2 Applications

In this subsection, we apply our heuristic to a large set of matrices. To present a thorough comparison with Paar's and Boyar-Peralta's heuristics, we have searched implementations of those matrices that were optimized in [KLSW17] using Paar's and Boyar-Peralta's heuristics. The results are listed in Table 1 and Table 2. Experiment results reveal that our heuristic performs equally good or gains improvement compared with Paar's and Boyar-Peralta's heuristics on 16×16 matrices. This implies that even in the simplest use case of 16×16 matrices, there is still room for improvements. In the case of 32×32 matrices, our heuristic can find better implementations in most cases. Particularly, we can also find an implementation of the matrix used in AES MixColumns using only 92 XOR

Algorithm 2 Search Optimized Matrix Decomposition

Input: $M \in \mathrm{GL}(n, \mathbb{F}_2)$;
Output: Optimized Decomposition of M;

Decompose M as $E_t E_{t-1} \cdots E_1$; ▷ Strategy 3-1 or 3-2
$seq \leftarrow E_t, E_{t-1}, \cdots, E_1$;
$g \leftarrow t + 1$;
while $g \geq 2$ **do**
  $g = g - 1$;
  for $i = 0, t - g$ **do**
    $seq_1 = E_t, \cdots, E_{i+g-1}$;
    $seq_2 = E_{i+g}, \cdots, E_{i+1}$;
    $seq_3 = E_i, \cdots, E_1$;
    $M' = E_{i+g} \cdots E_{i+1}$;
    Decompose M' as $E'_v \cdots E'_1$; ▷ Strategy 3-1 or 3-2
    $seq'_2 = E'_v, \cdots, E'_1$; ▷ Decompose M'
    $seq' = seq_1 + seq'_2 + seq_3$;
    $seq* \leftarrow$ **Reduce**(seq') ▷ Algorithm 1
    if $|seq| > |seq*|$ **then**
      $seq = seq*$;
      $g = |seq*| + 1$;
      break;
    end if
  end for
end while
  return seq;

operations (see in Table 3), which is equally good concerning the number of required Xor operations compared with previous best results.

With regard to 64×64 matrices, it seems that the matrix size is too large such that our procedure can not always stop in a reasonable time. However, if Algorithm 2 can return implementations for 64×64 matrices, they are always much better than previous results.

5.3 On Inverse Matrices

As pointed in [BKL16], the s-Xor count is invariant under taking the inverse. Moreover, the implementation of a matrix can be easily converted to the implementation of its inverse and they share the same cost. In the design of efficiently implementable linear layers, a sparse matrix is more desirable to reach this goal, such as the linear layer used in AES which is designed with this idea in mind. However, the inverse matrix of AES MixColumns is much more complex, and it is often expected to be less efficient. Note that Paar's and Boyar-Peralta's heuristics deal with a matrix and its inverse independently, and this will result in different implementations. However, our heuristic is based on matrix decomposition, and the inverse of a matrix M can be decomposed according to the decomposition of M. That is, when M can be decomposed as $M = E_0 E_1 \cdots E_{n-1}$, M^{-1} can be decomposed as $M^{-1} = E_{n-1}^{-1} \cdots E_1^{-1} E_0^{-1}$. This implies the implementation cost of M^{-1} equals the cost of M obtained using our heuristic and s-Xor metric. A direct application of this property is that the AES inverse MixColumns can be implemented using only 92 Xor operations (see Table 4). This also sheds some light on the design of linear layers since the community used to design lightweight linear layers using extremely sparse matrices. This property may inspire us to design linear layers from the space of all possible matrices rather than focusing on sparse matrices.

Table 1: Implementation cost of cipher matrices under different optimization heuristics.

Cipher	Size	[KLSW17][1]	[KLSW17][2]	[KLSW17][3]	[BFI19]	This paper
Fox mu8 [JV04]	64	611	-	594	592	-
Grostl [GKM+09]	64	493	-	475	460	-
Khazad [BR00b]	64	488	-	507	492	366
Whirlpool [BR00c]	64	481	-	465	464	-
Aes[4] [DR02]	32	108	108	97	95	92
Anubis [BR00a]	32	122	121	113	102	99
Clefia M_0 [SSA+07]	32	121	121	106	102	98
Clefia M_1 [SSA+07]	32	121	121	111	110	103
Fox mu4 [JV04]	32	144	143	137	131	136
Twofish [SKW+98]	32	151	149	129	125	111
Joltik [JNP14]	16	52	48	48	47	44
SmallScale Aes [CMR05]	16	54	54	47	45	43
Whirlwind M_0 [BNN+10]	32	218	218	212	210	183
Whirlwind M_1 [BNN+10]	32	246	244	235	234	190
Qarma128 [Ava17]	32	48	48	48	48	48
Midori [BBI+15]	16	24	24	24	24	24
Prince M_0, M_1 [BCG+12]	16	24	24	24	24	24
Pride$L_0 - L_3$ [ADK+14]	16	24	24	24	24	24
Qarma64 [Ava17]	16	24	24	24	24	24
Skinny64 [BJK+16]	16	12	12	12	12	12

[1] This column presents the results in [KLSW17] obtained by Paar1's heuristic.
[2] This column presents the results in [KLSW17] obtained by Paar2's heuristic.
[3] This column presents the results in [KLSW17] obtained by Boyar-Peralta's heuristic.
[4] [Max19] presents a 92-Xor implementation of AES.

Table 2: Implementation cost of newly designed matrices under different optimization heuristics

Matrix	[KLSW17][1]	[KLSW17][2]	[KLSW17][3]	[BFI19]	This paper
4 x 4 matrices over GL(4,F_2)					
[SKOP15]	50	48	48	46	44
[LS16]	49	46	44	44	44
[LW16]	48	47	44	44	44
[BKL16]	48	47	42	42	41
[SS16]	46	45	43	42	41
[JPST17]	48	47	43	42	41
[SKOP15](Involutory)	52	48	48	47	44
[LW16](Involutory)	51	48	48	46	44
[SS16] (Involutory)	50	48	42	40	38
[JPST17](Involutory)	51	47	47	46	41
4 × 4 matrices over GL(8,F_2)					
[SKOP15]	100	98	100	94	90
[LS16]	116	116	112	110	121
[LW16]	102	102	102	102	110
[BKL16]	116	112	110	108	114
[SS16]	110	108	107	104	114
[JPST17]	96	95	86	86	82
[SKOP15] (Involutory)	104	101	100	94	91
[LW16] (Involutory)	101	97	91	90	87
[SS16] (Involutory)	110	109	100	98	93
[JPST17] (Involutory)	102	100	91	92	83
8 × 8 matrices over GL(4,F_2)					
[SKOP15]	210	209	194	192	170
[SS17]	212	212	204	203	182
[SKOP15] (Involutory)	222	222	217	212	185
8 × 8 matrices over GL(8,F_2)					
[SKOP15]	474	-	467	460	-
[LS16]	464	-	447	443	-
[BKL16]	506	-	498	497	-
[SS17]	447	-	438	436	-
[SKOP15] (Involutory)	430	-	428	419	348
[JPST17] (Involutory)	620	-	599	591	-

Table 3: An implementation of AES MixColumns with 92 XOR operations.

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

$x_0, x_1, ..., x_{31}$ are the 32 input bits, and $y_0, y_1, ..., y_{31}$ are the 32 output bits. $x_0||\cdots||x_7$ is the fist byte with x_7 being the most significant bit, i.e., the coefficient of the degree-7 term of the finite field element in polynomial representation.

Table 4: An implementation of AES inverse MixColumns with 92 XOR operations.

No.	Operation	No.	Operation	No.	Operation
0	$u_{27}=u_{27}+u_3$	31	$u_{14}=u_{14}+u_{22}$	62	$u_{28}=u_{28}+u_{12}$
1	$u_1=u_1+u_{24}$	32	$u_{23}=u_{23}+u_{30}$	63	$u_4=u_4+u_{23}$
2	$u_{30}=u_{30}+u_{22}$	33	$u_{23}=u_{23}+u_{15}$	64	$u_{11}=u_{11}+u_{31}[v_{19}]$
3	$u_{26}=u_{26}+u_{10}$	34	$u_{31}=u_{31}+u_{22}$	65	$u_5=u_5+u_{28}$
4	$u_{31}=u_{31}+u_{23}$	35	$u_8=u_8+u_{31}$	66	$u_3=u_3+u_{11}$
5	$u_9=u_9+u_1$	36	$u_{24}=u_{24}+u_{23}$	67	$u_4=u_4+u_3$
6	$u_{22}=u_{22}+u_6$	37	$u_8=u_8+u_{24}$	68	$u_{13}=u_{13}+u_4[v_{29}]$
7	$u_{15}=u_{15}+u_{31}$	38	$u_1=u_1+u_8$	69	$u_{29}=u_{29}+u_5[v_5]$
8	$u_7=u_7+u_{15}$	39	$u_7=u_7+u_{31}$	70	$u_5=u_5+u_{13}$
9	$u_{17}=u_{17}+u_9$	40	$u_{16}=u_{16}+u_{24}$	71	$u_{14}=u_{14}+u_5[v_{30}]$
10	$u_{18}=u_{18}+u_{26}$	41	$u_3=u_3+u_7$	72	$u_{16}=u_{16}+u_{31}[v_8]$
11	$u_{25}=u_{25}+u_{24}$	42	$u_0=u_0+u_7$	73	$u_{26}=u_{26}+u_2[v_{26}]$
12	$u_{24}=u_{24}+u_0$	43	$u_9=u_9+u_{16}$	74	$u_{30}=u_{30}+u_{14}[v_{22}]$
13	$u_{25}=u_{25}+u_{17}$	44	$u_{17}=u_{17}+u_0[v_{25}]$	75	$u_{10}=u_{10}+u_{26}[v_{18}]$
14	$u_{11}=u_{11}+u_{27}$	45	$u_{25}=u_{25}+u_1[v_1]$	76	$u_0=u_0+u_{16}$
15	$u_{19}=u_{19}+u_{11}$	46	$u_1=u_1+u_{17}$	77	$u_8=u_8+u_{23}[v_{16}]$
16	$u_4=u_4+u_{28}$	47	$u_{10}=u_{10}+u_1$	78	$u_{22}=u_{22}+u_{30}[v_6]$
17	$u_6=u_6+u_{14}$	48	$u_{18}=u_{18}+u_9[v_2]$	79	$u_0=u_0+u_8[v_{24}]$
18	$u_{26}=u_{26}+u_{25}$	49	$u_2=u_2+u_9$	80	$u_6=u_6+u_{22}[v_{14}]$
19	$u_{12}=u_{12}+u_4$	50	$u_{10}=u_{10}+u_{18}$	81	$u_9=u_9+u_{25}[v_9]$
20	$u_{20}=u_{20}+u_{12}$	51	$u_2=u_2+u_{10}[v_{10}]$	82	$u_7=u_7+u_{15}[v_{15}]$
21	$u_{27}=u_{27}+u_{26}$	52	$u_3=u_3+u_2$	83	$u_{24}=u_{24}+u_0[v_0]$
22	$u_{28}=u_{28}+u_{27}$	53	$u_{11}=u_{11}+u_{10}$	84	$u_{21}=u_{21}+u_{29}[v_{21}]$
23	$u_{21}=u_{21}+u_5$	54	$u_3=u_3+u_{18}$	85	$u_4=u_4+u_{20}[v_4]$
24	$u_{13}=u_{13}+u_{21}$	55	$u_{19}=u_{19}+u_3[v_3]$	86	$u_3=u_3+u_{27}[v_{11}]$
25	$u_{21}=u_{21}+u_{20}$	56	$u_{27}=u_{27}+u_{19}$	87	$u_1=u_1+u_9[v_{17}]$
26	$u_{29}=u_{29}+u_{13}$	57	$u_{28}=u_{28}+u_7$	88	$u_5=u_5+u_{21}[v_{13}]$
27	$u_{30}=u_{30}+u_{21}$	58	$u_{27}=u_{27}+u_{23}[v_{27}]$	89	$u_{28}=u_{28}+u_4[v_{12}]$
28	$u_{22}=u_{22}+u_{29}$	59	$u_{12}=u_{12}+u_{11}$	90	$u_{23}=u_{23}+u_7[v_{31}]$
29	$u_6=u_6+u_{21}$	60	$u_{12}=u_{12}+u_{27}[v_{20}]$	91	$u_{31}=u_{31}+u_{23}[v_{23}]$
30	$u_{15}=u_{15}+u_6[v_7]$	61	$u_{20}=u_{20}+u_{28}[v_{28}]$		

$u_0, u_1, ..., u_{31}$ are the 32 input bits, and $v_0, v_1, ..., v_{31}$ are the 32 output bits.

6 Summary

In this paper, we introduced a new heuristic search algorithm to globally optimize the implementation of linear matrices, which is built on the decomposition theory of invertible matrices. We observed that a type-3 elementary matrix corresponds to an Xor operation in the implementation of matrices. Thus we reduced the problem of optimizing implementations of matrices to the problem of optimizing matrix decompositions. First of all, we presented four matrix decomposition strategies which are based on different elementary operations performed on a matrix. To optimize the matrix implementation, we constructed a large number of equivalent matrix decompositions and proposed seven matrix multiplication rules that can be fed to the heuristic to reduce the cost. As applications, we applied our heuristic to a large set of newly designed and known matrices for symmetric-key ciphers. The results show that our heuristic is quite powerful in reducing the implementation cost of 16×16 and 32×32 matrices. Moreover, our heuristic enjoys an extra advantage that a matrix and its inverse share the same cost.

Though our heuristic can find better implementations in most cases, the circuit depth of the resulting implementation is out of the consideration of this paper, and we leave it as future work.

Acknowledgments

The authors would like to thank the anonymous reviewers for their helpful comments and suggestions. This work was supported by the National Natural Science Foundation of China (Grant No.61802119). Xiangyong Zeng was supported by Major Technological Innovation Special Project of Hubei Province (Grant No. 2019ACA144). Zhenzhen Bao was partially supported by Nanyang Technological University in Singapore under Grant 04INS000397C230, and Singapore's Ministry of Education under Grants RG18/19 and MOE2019-T2-1-060.

References

[ADK+14] Martin R. Albrecht, Benedikt Driessen, Elif Bilge Kavun, Gregor Leander, Christof Paar, and Tolga Yalçin. Block ciphers - focus on the linear layer (feat. PRIDE). In *Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part I*, pages 57–76, 2014.

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

[ARVV18] Elena Andreeva, Reza Reyhanitabar, Kerem Varici, and Damian Vizár. Forking a blockcipher for authenticated encryption of very short messages. *IACR Cryptology ePrint Archive*, 2018:916, 2018.

[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 Trans. Symmetric Cryptol.*, 2017(1):4–44, 2017.

[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*, pages 411–436, 2015.

[BCG+12] Julia Borghoff, Anne Canteaut, Tim Güneysu, Elif Bilge Kavun, Miroslav Knezevic, Lars R. Knudsen, Gregor Leander, Ventzislav Nikov, Christof Paar, Christian Rechberger, Peter Rombouts, Søren S. Thomsen, and Tolga Yalçin. PRINCE - A low-latency block cipher for pervasive computing applications - extended abstract. In Xiaoyun Wang and Kazue Sako, editors, *Advances in Cryptology - ASIACRYPT 2012 - 18th International Conference on the Theory and Application of Cryptology and Information Security, Beijing, China, December 2-6, 2012. Proceedings*, volume 7658 of *Lecture Notes in Computer Science*, pages 208–225. Springer, 2012.

[BFI19] Subhadeep Banik, Yuki Funabiki, and Takanori Isobe. More results on shortest linear programs. In Nuttapong Attrapadung and Takeshi Yagi, editors, *Advances in Information and Computer Security - 14th International Workshop on Security, IWSEC 2019, Tokyo, Japan, August 28-30, 2019, Proceedings*, volume 11689 of *Lecture Notes in Computer Science*, pages 109–128. Springer, 2019.

[BJK+16] Christof Beierle, Jérémy Jean, Stefan Kölbl, Gregor Leander, Amir Moradi, Thomas Peyrin, Yu Sasaki, Pascal Sasdrich, and Siang Meng Sim. The SKINNY family of block ciphers and its low-latency variant MANTIS. In *Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part II*, pages 123–153, 2016.

[BKL+07] Andrey Bogdanov, Lars R. Knudsen, Gregor Leander, Christof Paar, Axel Poschmann, Matthew J. B. Robshaw, Yannick Seurin, and C. Vikkelsoe. PRESENT: an ultra-lightweight block cipher. In *Cryptographic Hardware and Embedded Systems - CHES 2007, 9th International Workshop, Vienna, Austria, September 10-13, 2007, Proceedings*, pages 450–466, 2007.

[BKL16] Christof Beierle, Thorsten Kranz, and Gregor Leander. Lightweight multiplication in gf(2^n) with applications to MDS matrices. In *Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part I*, pages 625–653, 2016.

[BMP13] Joan Boyar, Philip Matthews, and René Peralta. Logic minimization techniques with applications to cryptology. *J. Cryptology*, 26(2):280–312, 2013.

[BNN+10] Paulo S. L. M. Barreto, Ventzislav Nikov, Svetla Nikova, Vincent Rijmen, and Elmar Tischhauser. Whirlwind: a new cryptographic hash function. *Des. Codes Cryptogr.*, 56(2-3):141–162, 2010.

[BP10] Joan Boyar and René Peralta. A new combinational logic minimization technique with applications to cryptology. In Paola Festa, editor, *Experimental Algorithms, 9th International Symposium, SEA 2010, Ischia Island, Naples, Italy, May 20-22, 2010. Proceedings*, volume 6049 of *Lecture Notes in Computer Science*, pages 178–189. Springer, 2010.

[BR00a] Paulo S.L.M. Barreto and Vincent Rijmen. The ANUBIS Block Cipher. *First Open NESSIE Workshop*, 2000.

[BR00b] Paulo S.L.M. Barreto and Vincent Rijmen. The Khazad legacy-level Block Cipher. *First Open NESSIE Workshop*, 2000.

[BR00c] Paulo S.L.M. Barreto and Vincent Rijmen. The WHIRLPOOL Hashing Function. *First Open NESSIE Workshop*, 2000.

[CMR05] Carlos Cid, Sean Murphy, and Matthew J. B. Robshaw. Small scale variants of the AES. In *Fast Software Encryption: 12th International Workshop, FSE 2005, Paris, France, February 21-23, 2005, Revised Selected Papers*, pages 145–162, 2005.

[Dae95] Joan Daemen. *Cipher and Hash Function Design Strategies based on linear and differential cryptanalysis.* PhD thesis, Doctoral Dissertation, March 1995, KU Leuven, 1995.

[DL18] Sébastien Duval and Gaëtan Leurent. MDS matrices with lightweight circuits. *IACR Trans. Symmetric Cryptol.*, 2018(2):48–78, 2018.

[DR02] Joan Daemen and Vincent Rijmen. *The Design of Rijndael: AES - The Advanced Encryption Standard.* Information Security and Cryptography. Springer, 2002.

[GKM+09] Praveen Gauravaram, Lars R. Knudsen, Krystian Matusiewicz, Florian Mendel, Christian Rechberger, Martin Schläffer, and Søren S. Thomsen. Grøstl - a SHA-3 candidate. In *Symmetric Cryptography, 11.01. - 16.01.2009*, 2009.

[GLG+17] Zhiyuan Guo, Renzhang Liu, Si Gao, Wenling Wu, and Dongdai Lin. Direct construction of optimal rotational-xor diffusion primitives. *IACR Trans. Symmetric Cryptol.*, 2017(4):169–187, 2017.

[GPPR11] Jian Guo, Thomas Peyrin, Axel Poschmann, and Matthew J. B. Robshaw. The LED block cipher. In *Cryptographic Hardware and Embedded Systems - CHES 2011 - 13th International Workshop, Nara, Japan, September 28 - October 1, 2011. Proceedings*, pages 326–341, 2011.

[JB09] Daniel J Bernstein. Optimizing linear maps modulo 2. *Workshop Record of SPEED-CC: Software Performance Enhancement for Encryption and Decryption and Cryptographic Compilers*, 2009.

[JNP14] Jérémy Jean, Ivica Nikolić, and Thomas Peyrin. Joltik. *Submission to the CAESAR competition*, 2014.

[JNRV19] Samuel Jaques, Michael Naehrig, Martin Roetteler, and Fernando Virdia. Implementing grover oracles for quantum key search on AES and lowmc. *IACR Cryptology ePrint Archive*, 2019:1146, 2019.

[JPST17] Jérémy Jean, Thomas Peyrin, Siang Meng Sim, and Jade Tourteaux. Optimizing implementations of lightweight building blocks. *IACR Trans. Symmetric Cryptol.*, 2017(4):130–168, 2017.

[JV04] 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*, pages 114–129, 2004.

[KLSW17] Thorsten Kranz, Gregor Leander, Ko Stoffelen, and Friedrich Wiemer. Shorter linear straight-line programs for MDS matrices. *IACR Trans. Symmetric Cryptol.*, 2017(4):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*, pages 285–312, 2019.

[KPPY14] Khoongming Khoo, Thomas Peyrin, Axel York 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*, pages 433–450, 2014.

[LP07] Gregor Leander and Axel Poschmann. On the classification of 4 bit s-boxes. In *Arithmetic of Finite Fields, First International Workshop, WAIFI 2007, Madrid, Spain, June 21-22, 2007, Proceedings*, pages 159–176, 2007.

[LS16] Meicheng Liu and Siang Meng Sim. Lightweight MDS generalized circulant matrices. In *Fast Software Encryption - 23rd International Conference, FSE 2016, Bochum, Germany, March 20-23, 2016, Revised Selected Papers*, pages 101–120, 2016.

[LSL+19] Shun Li, Siwei Sun, Chaoyun Li, Zihao Wei, and Lei Hu. Constructing low-latency involutory MDS matrices with lightweight circuits. *IACR Trans. Symmetric Cryptol.*, 2019(1):84–117, 2019.

[LW16] Yongqiang Li and Mingsheng Wang. On the construction of lightweight circulant involutory MDS matrices. In *Fast Software Encryption - 23rd International Conference, FSE 2016, Bochum, Germany, March 20-23, 2016, Revised Selected Papers*, pages 121–139, 2016.

[LW17] Chaoyun Li and Qingju Wang. Design of lightweight linear diffusion layers from near-mds matrices. *IACR Trans. Symmetric Cryptol.*, 2017(1):129–155, 2017.

[Max19] Alexander Maximov. AES mixcolumn with 92 XOR gates. *IACR Cryptology ePrint Archive*, 2019:833, 2019.

[ME19] Alexander Maximov and Patrik Ekdahl. New circuit minimization techniques for smaller and faster AES sboxes. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2019(4):91–125, 2019.

[Paa97] C. Paar. Optimized arithmetic for reed-solomon encoders. In *Proceedings of IEEE International Symposium on Information Theory*, pages 250–250, June 1997.

[Saa11] Markku-Juhani O. Saarinen. Cryptographic analysis of all 4×4-bit s-boxes. In *Selected Areas in Cryptography - 18th International Workshop, SAC 2011, Toronto, ON, Canada, August 11-12, 2011, Revised Selected Papers*, pages 118–133, 2011.

[SKOP15] Siang Meng Sim, Khoongming Khoo, Frédérique E. 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*, 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, 1998.

[SS16] Sumanta Sarkar and Habeeb Syed. Lightweight diffusion layer: Importance of toeplitz matrices. *IACR Trans. Symmetric Cryptol.*, 2016(1):95–113, 2016.

[SS17] Sumanta Sarkar and Habeeb Syed. Analysis of toeplitz MDS matrices. In *Information Security and Privacy - 22nd Australasian Conference, ACISP 2017, Auckland, New Zealand, July 3-5, 2017, Proceedings, Part II*, pages 3–18, 2017.

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

[TP20] Quan Quan Tan and Thomas Peyrin. Improved heuristics for short linear programs. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2020(1):203–230, 2020.

[VSP18] Andrea Visconti, Chiara Valentina Schiavo, and René Peralta. Improved upper bounds for the expected circuit complexity of dense systems of linear equations over GF(2). *Inf. Process. Lett.*, 137:1–5, 2018.

[WP13] Hongjun Wu and Bart Preneel. AEGIS: A fast authenticated encryption algorithm. In Tanja Lange, Kristin E. Lauter, and Petr Lisonek, editors, *Selected Areas in Cryptography - SAC 2013 - 20th International Conference, Burnaby, BC, Canada, August 14-16, 2013, Revised Selected Papers*, volume 8282 of *Lecture Notes in Computer Science*, pages 185–201. Springer, 2013.

[ZBRL15] Wentao Zhang, Zhenzhen Bao, Vincent Rijmen, and Meicheng Liu. A new classification of 4-bit optimal s-boxes and its application to present, RECTANGLE and SPONGENT. In *Fast Software Encryption - 22nd International Workshop, FSE 2015, Istanbul, Turkey, March 8-11, 2015, Revised Selected Papers*, pages 494–515, 2015.

[ZWS18] Lijing Zhou, Licheng Wang, and Yiru Sun. On efficient constructions of lightweight MDS matrices. *IACR Trans. Symmetric Cryptol.*, 2018(1):180–200, 2018.

A Rational of Strategy 3-1 (and 3-2)

We will use a small example to illustrate the process of Strategy 3-1. Assuming that

$$M = \begin{pmatrix} 0 & 1 & 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 \end{pmatrix},$$

the rows and columns will be numbered from 1 to 6 in the following. We first compute all possible $r_i = r_i \oplus r_j$, where r_i denotes the ith row of M and $1 \leq i, j \leq 6$, $i \neq j$. We save the candidates that reduce the most 1's in the resulting matrix. For example, $r_1 = r_1 \oplus r_3$ will transform the first row from (0,1,1,0,1,1) to (0,0,1,1,0,0), and other rows remain unchanged. The first row contains four and two 1's before and after the transformation. Thus, this transformation will reduce two 1's in the resulting matrix. After checking all possibilities, we can find that $r_1 = r_1 \oplus r_3$, $r_1 = r_1 \oplus r_4$, $r_3 = r_3 \oplus r_1$, $r_4 = r_4 \oplus r_1$ and $r_4 = r_4 \oplus r_3$ will make the matrix reduce two 1's and no candidates can make the matrix reduce more than two 1's. Similarly, we check all possibilities of $c_i = c_i \oplus c_j$, where c_i denotes the ith column of M, and we can find that $c_5 = c_5 \oplus c_3$, $c_5 = c_5 \oplus c_6$, $c_6 = c_6 \oplus c_3$ and $c_6 = c_6 \oplus c_5$ makes

the matrix reduce two 1's and no more candidates can make it reduce more than two 1's. Combining all possible row and column transformations, we have 9 choices for this step. We randomly pick one of the choices when using Strategy 3-1 and 3-2. Assuming that we choose the first candidate, i.e., $r_1 = r_1 \oplus r_3$, the matrix after transformation is

$$M_1 = \begin{pmatrix} 0&0&1&1&0&0\\ 1&0&0&0&0&1\\ 0&1&0&1&1&1\\ 1&0&1&0&1&1\\ 1&1&0&1&0&0\\ 1&0&0&1&1&0 \end{pmatrix} = \begin{pmatrix} 1&0&1&0&0&0\\ 0&1&0&0&0&0\\ 0&0&1&0&0&0\\ 0&0&0&1&0&0\\ 0&0&0&0&1&0\\ 0&0&0&0&0&1 \end{pmatrix} \begin{pmatrix} 0&1&1&0&1&1\\ 1&0&0&0&0&1\\ 0&1&0&1&1&1\\ 1&0&1&0&1&1\\ 1&1&0&1&0&0\\ 1&0&0&1&1&0 \end{pmatrix} = E(1{+}3)M.$$

Next, we perform similar procedure on M_1 iteratively. We list in the following the two sequential steps.

$$M_2 = \begin{pmatrix} 0&0&1&1&0&0\\ 1&0&0&0&0&1\\ 0&1&0&0&1&1\\ 1&0&1&0&1&1\\ 1&1&0&0&0&0\\ 1&0&0&1&1&0 \end{pmatrix} = \begin{pmatrix} 0&0&1&1&0&0\\ 1&0&0&0&0&1\\ 0&1&0&1&1&1\\ 1&0&1&0&1&1\\ 1&1&0&1&0&0\\ 1&0&0&1&1&0 \end{pmatrix} \begin{pmatrix} 1&0&0&0&0&0\\ 0&1&0&1&0&0\\ 0&0&1&0&0&0\\ 0&0&0&1&0&0\\ 0&0&0&0&1&0\\ 0&0&0&0&0&1 \end{pmatrix} = M_1E(2{+}4),$$

$$M_3 = \begin{pmatrix} 0&0&1&1&0&0\\ 1&0&0&0&0&1\\ 0&1&0&0&1&1\\ 0&0&1&0&1&0\\ 1&1&0&0&0&0\\ 1&0&0&1&1&0 \end{pmatrix} = \begin{pmatrix} 1&0&0&0&0&0\\ 0&1&0&0&0&0\\ 0&0&1&0&0&0\\ 0&1&0&1&0&0\\ 0&0&0&0&1&0\\ 0&0&0&0&0&1 \end{pmatrix} \begin{pmatrix} 0&0&1&1&0&0\\ 1&0&0&0&0&1\\ 0&1&0&0&1&1\\ 1&0&1&0&1&1\\ 1&1&0&0&0&0\\ 1&0&0&1&1&0 \end{pmatrix} = E(4{+}2)M_2.$$

It can be verified that the number of 1's in the matrix is reduced by two for each of the above three steps. However, we can see that no matter which row or column elementary transformation is performed on M_3, the number of 1's contained in M_3 cannot be reduced. This explains the second problem presented in Section 3.2. In this case we turn to Strategy 1 or Strategy 2 for the following process of M_3. For illustration purpose, we adopt Strategy 1 to process M_3, i.e., using Gaussian Elimination to transform M_3 to an identity matrix. Specifically,

$$\begin{aligned} &E(4 \leftrightarrow 5)E(3 \leftrightarrow 4)E(2 \leftrightarrow 5)E(1 \leftrightarrow 2)E(5+6)E(2+6)E(4+3)E(1+3)E(6+1)\\ &E(1+4)E(3+5)E(6+2)E(5+2)M_3 = E. \end{aligned}$$

That is to say, we have to perform nine type-3 and four type-1 elementary row transformations to get an identity matrix. Combining with the first three elementary transformation, we can get

$$\begin{aligned} &E(4 \leftrightarrow 5)E(3 \leftrightarrow 4)E(2 \leftrightarrow 5)E(1 \leftrightarrow 2)E(5+6)E(2+6)E(4+3)E(1+3)E(6+1)\\ &E(1+4)E(3+5)E(6+2)E(5+2)E(4+2)E(1+3)ME(2+4) = E. \end{aligned}$$

Thus,

$$\begin{aligned} M =& E(1+3)^{-1}E(4+2)^{-1}E(5+2)^{-1}E(6+2)^{-1}E(3+5)^{-1}E(1+4)^{-1}\\ & E(6+1)^{-1}E(1+3)^{-1}E(4+3)^{-1}E(2+6)^{-1}E(5+6)^{-1}E(1 \leftrightarrow 2)^{-1}\\ & E(2 \leftrightarrow 5)^{-1}E(3 \leftrightarrow 4)^{-1}E(4 \leftrightarrow 5)^{-1}E(2+4)^{-1}. \end{aligned}$$

Moreover, type-1 and type-3 elementary matrices are involutory matrices, thus

$$\begin{aligned} M =& E(1+3)E(4+2)E(5+2)E(6+2)E(3+5)E(1+4)E(6+1) \\ & E(1+3)E(4+3)E(2+6)E(5+6)E(1 \leftrightarrow 2)E(2 \leftrightarrow 5)E(3 \leftrightarrow 4) \\ & E(4 \leftrightarrow 5)E(2+4). \end{aligned}$$

Using Property 1 presented in the paper, type-1 elementary matrices are moved to the right of the decomposition, thus

$$\begin{aligned} M =& E(1+3)E(4+2)E(5+2)E(6+2)E(3+5)E(1+4)E(6+1) \\ & E(1+3)E(4+3)E(2+6)E(5+6)E(5+1)E(1 \leftrightarrow 2)E(2 \leftrightarrow 5) \\ & E(3 \leftrightarrow 4)E(4 \leftrightarrow 5). \end{aligned}$$

Therefore, we successfully get a matrix decomposition as required.

B Proof of Rule 1-7

Let I denote the identity matrix, and $I_{i,j}$ denote the matrix with a single entry at the ith row and jth column taking a value of 1 and all other entries being 0. Clearly, $E(i+j) = I + I_{i,j}$.

Lemma 1.

$$I_{i,j}I_{u,v} = \begin{cases} 0, & \text{if } j \neq u \\ I_{i,v}, & \text{if } j = u \end{cases}, \text{ where 0 denotes the zero matrix.}$$

Proof. Since all rows except the ith row of $I_{i,j}$ are zero row vectors, and all columns except the vth column of $I_{u,v}$ are zero column vectors. It can be easily deduced that all rows except the ith row of $I_{i,j}I_{u,v}$ are zero row vectors, and all columns except the vth column of $I_{i,j}I_{u,v}$ are zero column vectors, which means the only possible nonzero element in $I_{i,j}I_{u,v}$ is $I_{i,j}I_{u,v}[i][v]$, which denotes the element at the ith row and vth column of $I_{i,j}I_{u,v}$. Since $I_{i,j}I_{u,v}[i][v] = \oplus_k I_{i,j}[i][k] \times I_{u,v}[k][v]$, it follows that $I_{i,j}I_{u,v}[i][v]$ equals 1 if $j = u$ and 0 otherwise. □

Proof of Rule 1. $E(k+i)E(k+j)E(i+j) = E(i+j)E(k+i)$, where $i \neq j \neq k$.

Proof.

$$\begin{aligned} & E(k+i)E(k+j)E(i+j) = (I + I_{k,i})(I + I_{k,j})(I + I_{i,j}) \\ & = (I + I_{k,i} + I_{k,j} + I_{k,i}I_{k,j})(I + I_{i,j}) \\ & = (I + I_{k,i} + I_{k,j})(I + I_{i,j}) \\ & = I + I_{k,i} + I_{k,j} + I_{i,j} + I_{k,i}I_{i,j} + I_{k,j}I_{i,j} \\ & = I + I_{k,i} + I_{k,j} + I_{i,j} + I_{k,j} \\ & = I + I_{k,i} + I_{i,j}. \end{aligned} \tag{4}$$

$$\begin{aligned} & E(i+j)E(k+i) = (I + I_{i,j})(I + I_{k,i}) \\ & = I + I_{i,j} + I_{k,i} + I_{i,j}I_{k,i} \\ & = I + I_{i,j} + I_{k,i}. \end{aligned} \tag{5}$$

(4) equals (5) which completes the proof. □

Rule 2-6 can be proved similarly.

Proof of Rule 7. $E(j+i)E(i+j) = E(i \leftrightarrow j)E(j+i)$, where $i \neq j$.

Proof. Clearly, $E(i \leftrightarrow j) = I + I_{i,j} + I_{j,i} + I_{i,i} + I_{j,j}$.

$$\begin{aligned} E(j+i)E(i+j) &= (I + I_{j,i})(I + I_{i,j}) \\ &= I + I_{i,j} + I_{j,i} + I_{j,i}I_{i,j} \\ &= I + I_{i,j} + I_{j,i} + I_{j,j}. \end{aligned} \tag{6}$$

$$\begin{aligned} E(i \leftrightarrow j)E(j+i) &= (I + I_{i,j} + I_{j,i} + I_{i,i} + I_{j,j})(I + I_{j,i}) \\ &= I + I_{i,j} + I_{j,i} + I_{i,i} + I_{j,j} + I_{j,i} + I_{i,j}I_{j,i} + I_{j,i}I_{j,i} + I_{i,i}I_{j,i} + I_{j,j}I_{j,i} \\ &= I + I_{i,j} + I_{j,i} + I_{i,i} + I_{j,j} + I_{j,i} + I_{i,i} + I_{j,i} \\ &= I + I_{i,j} + I_{j,i} + I_{j,j}. \end{aligned} \tag{7}$$

(6) equals (7) which completes the proof. □