More Accurate Division Property Propagations Based on Optimized Implementations of Linear Layers

Chunlei Hong, Shasha Zhang✉, Siwei Chen, Da Lin, and Zejun Xiang

Faculty of Mathematics and Statistics, Hubei Key Laboratory of Applied Mathematics, Hubei University, Wuhan, China
hongchunlei@stu.hubu.edu.cn, {amushasha,chensiwei_hubu}@163.com
linda@stu.hubu.edu.cn, xiangzejun@hubu.edu.cn

Abstract. As a generalized integral property, the division property can be used to search integral distinguishers of symmetric ciphers by taking the advantage of automatic tools, such as Mixed Integer Linear Programming (MILP) and Boolean Satisfiability Problem (SAT) solvers. In this case, the accuracy of corresponding models will influence the resulting distinguishers. In this paper, we present a new technique to characterize the division property propagation of linear layers. Firstly, we study the impact of a linear layer implementation on its division property propagations. We found that division trails derived from an optimized implementation of a linear layer can be more accurate than the $\mathcal{S}$ method, and different implementations can eliminate some different invalid division trails. Thus, we can eliminate a large number of invalid division trails by combining different implementations. As an application of our technique, we have searched distinguishers for Midori64, Skinny64 and LED. As a result, we can obtain the same longest distinguishers as the $\mathcal{ZR}$ method and the $\mathcal{HW}$ method, which are the exact modeling of linear layers. Moreover, our method can be used with both MILP and SAT, while the $\mathcal{HW}$ method can only work with SAT. In addition, the number of constraints with the $\mathcal{HW}$ method increases quadratically, however it increases linearly with our method.

Keywords: Division property · Linear Layer · Optimized implementation · Integral attack · Automatic tool

1 Introduction

Differential cryptanalysis [1], linear cryptanalysis [2] and integral cryptanalysis [3] are the most effective methods for attacking iterative block ciphers so far. In 1997, Daemen *et al.* proposed Square [4] block cipher and introduced a new method named Square attack to analyze the security of Square cipher. This attack method is the earliest form of integral attack. Afterwards, integral attack [3] was formally proposed by Knudsen and Wagner at FSE 2002. The core idea of integral attack is to find an integral distinguisher so that the adversary

can use this distinguisher to achieve a distinguishing attack or a key-recovery attack on the objective cipher. Thus, the most essential step in integral attacks is to construct effective integral distinguishers.

Currently, the most effective way to find a distinguisher is to study the evolution of integral properties in the encryption process and then judge whether there exists balanced bits in the corresponding output state. The division property (DP) [5], which was proposed as a generalized integral property by Todo at EUROCRYPT 2015, can be applied to search for longer integral distinguishers. With this technique, Todo constructed a new integral distinguisher and presented the full-round attack on MISTY1 in [6]. Afterwards, a more accurate DP called bit-based division property (BDP) [7] was introduced by Todo and Morii at FSE 2016. However, the BDP is only adapted to ciphers whose block sizes are upper bounded by 32 due to the huge time and memory complexities. In order to overstride this barrier, Xiang *et al.* [8] applied MILP method to search for integral distinguishers based on BDP at ASIACRYPT 2016, which allowed us to analyze primitives whose block sizes are larger than 32 using BDP. Thanks to this automatic method, what we need to do when searching for distinguishers is to construct an MILP model to characterize the propagation of BDP and use off-the-shelf solvers like Gurobi[1] to solve the model. Naturally, it is worth studying how to achieve the accurate description of the propagation of BDP in the MILP model, since the more accurate the corresponding MILP model is, the more balanced bits or longer distinguishers might be obtained.

The description of BDP propagations of non-linear layers (e.g., Sboxes [8], AND [8,9] and modular addition [10] operations) has been discussed extensively. As for linear layers, there are three main methods to constrain the BDP propagation.

- $\mathcal{S}$ **method [9].** It was proposed by Sun *et al.* for a matrix $M \in \mathbb{F}_2^{n\times n}$, and the core of this method is to decompose a complex matrix into a series of COPY and XOR operations and then model these basic operations with some auxiliary variables. The modeling rules of COPY and XOR have been already handled in [8]. The advantage of this method is that the total number of constraints is only $2n$ and it is generally applicable to all the types of linear layers, but its shortcoming is that it might introduce some invalid division trails, which will lead to an inaccurate model and cause the balance property of output bits to lose more quickly.
- $\mathcal{ZR}$ **method [11].** For an invertible matrix $M \in \mathbb{F}_{2^m}^{s\times s}$, Zhang and Rijmen constructed a one-to-one relation between a division trail of M and the invertibility of a sub-matrix of M, which is uniquely determined by this division trail. Specifically, a division trail is valid if and only if the corresponding sub-matrix is invertible. The number of constraints will be $m \cdot (2^s - 1)$ when constructing an MILP model. This method is completely accurate, but it cannot be applied to non-binary and non-invertible matrices as well as large matrices because of the huge scale of constraints.

[1] https://www.gurobi.com/

$\mathcal{HW}$ **method [12].** Inspired by the $\mathcal{ZR}$ method, Hu *et al.* noticed that a matrix $M \in \mathbb{F}_2^{n\times n}$ is invertible if and only if there is a matrix $M' \in \mathbb{F}_2^{n\times n}$ such that $M \times M'$ equals to an identity matrix. Thus, they introduced an auxiliary matrix and constrained the multiplication of this auxiliary matrix and the original matrix to be an identity matrix. If it is a solution, then the corresponding division trail is valid. This method is as accurate as the $\mathcal{ZR}$ method and is not limited by the invertibility of M. Note that the number of constraints is n^2, and it contains 4-degree constraints, thus it is only solvable for SMT/SAT solvers. In addition, when the scale of M is large, the model will be quite heavy, which may cause the infeasibility to solvers.

Recently, Elsheikh and Youssef [13] (ACISP 2021) proposed a method to optimize the precision of the BDP propagation of linear layers, which is based on the $\mathcal{ZR}$ method. In short, for a given input DP of linear layers, this method aims to search for the corresponding output DP such that the derived sub-matrix has full rank. Therefore, the first step is to determine the input DP and this process will take a lot of precomputations. The precomputations are time-consuming and hardly practical with the round increasing because of the huge amount of input DP. Thus, the authors in [13] only succeed in applying this technique to the first round. For more details, please refer to [13]. However, the above three methods and our method are all general descriptions of linear layers, without considering the specific input DP of linear layers.

1.1 Our Contributions

As mentioned above, the off-the-shelf methods to characterize the BDP propagation of complex linear layers have their advantages and limitations. In brief, the $\mathcal{S}$ method is applicable to various linear layers but not accurate enough. The $\mathcal{ZR}$ and $\mathcal{HW}$ methods are completely accurate, but both of them are limited by the size of the linear matrix and the former is also limited by the invertibility of the matrix. Therefore, it is significant to consider how to reach a balance between the feasibility and the accuracy. In order to find this trade-off, in this paper, we introduce a new method to achieve a more accurate as well as applicable MILP-aided description of the BDP propagation of complex linear layers. This method is inspired by optimizing implementations of matrices, and combines several optimizing tools with the existing COPY and XOR modeling rules. Concretely, given a matrix $M \in \mathbb{F}_2^{n\times n}$, we first respectively use three methods **Paar** [14], **BP** [15] and **XZ** [16] to optimize the implementation of M. Next, we apply both COPY and XOR modeling rules to constrain each optimized implementation. Finally, we simultaneously add all the constraints to an MILP model to characterize the BDP propagation of M. In order to intuitively evaluate the number of constraints, we convert the optimized implementations to the simplest form like $\mathrm{c} = \mathrm{a} \oplus \mathrm{b}$, i.e. one equation contains only one XOR operation (see Equation (2) in **Section** 3). Then, the upper bound on the number of constraints is 3N, where N denotes the total number of the linear equations or the XOR operations in the

optimized implementations. Based on this new method, we propose a framework as depicted in Algorithm 1 to automatically generate a system of inequalities. Compared with the $\mathcal{S}$ method, it is more accurate since the $\mathcal{S}$ method corresponds to the D-xor [25] metric of a linear matrix and this metric is far from optimal to represent the real cost of implementing a matrix. Thus, modeling these optimized implementations can effectively decrease the number of invalid division trails. We present a detailed discussion in **Section** 3 and prove that our technique is never worse than the $\mathcal{S}$ method. Moreover, it is more applicable than the $\mathcal{ZR}$ method thanks to its non-limitation on the type of matrices and more lightweight constraints even if it is not completely accurate. It is worth noting that this new method can also be implemented based on SMT/SAT, since all constraints are linear. Moreover, the number of the generated constraints is small compared with that of the $\mathcal{HW}$ method. As an illustration, we apply this method to search for integral distinguishers for three block ciphers Midori64 [17], Skinny64 [18] and LED [19], and compare our results with the previous works as summarized in Table 1. The related results[2] and source codes are available at `https://github.com/hcl21/More-Accurate-BDP-for-LinearLayer`.

1.2 Organization of the Paper

The rest of this paper is organized as follows. In **Section 2**, we briefly review the definition and propagation rules of division property and several heuristics for implementing linear layers. In **Section 3**, we introduce the new method proposed in this paper and give a discussion on the comparison between this method and the $\mathcal{S}$ method. In **Section 4**, we present applications of our new method to some block ciphers. We conclude this paper in **Section 5**.

2 Preliminaries

We first introduce some notations that appear frequently in this paper. Denote $\mathbb{F}_2$ the finite field that contains only two elements (0 and 1) and $\boldsymbol{a} \in \mathbb{F}_2^n$ an n-bit vector where $a_i \in \mathbb{F}_2$ denotes the i-th bit of $\boldsymbol{a}$. The Hamming weight of $\boldsymbol{a} \in \mathbb{F}_2^n$, denoted by $wt(\boldsymbol{a})$, is defined as $wt(\boldsymbol{a}) = \#\{i : a_i = 1, 0 \le i \le n-1\}$. Let $\boldsymbol{k}$ and $\boldsymbol{k}'$ be two vectors in $\mathbb{F}_2^n$, we define $\boldsymbol{k} \succeq \boldsymbol{k}'$ if $k_i \ge k'_i$ for all i, otherwise $\boldsymbol{k} \not\succeq \boldsymbol{k}'$.
Bit Product Functions. Let $\pi_{\boldsymbol{u}} : \mathbb{F}_2^n \to \mathbb{F}_2$ be a function for any $\boldsymbol{u} \in \mathbb{F}_2^n$. Let $\boldsymbol{x} \in \mathbb{F}_2^n$ be an input of $\pi_{\boldsymbol{u}}$, then $\pi_{\boldsymbol{u}}(\boldsymbol{x})$ is defined as

$$\pi_{\boldsymbol{u}}(\boldsymbol{x}) := \prod_{i=0}^{n-1} x_i^{u_i}.$$

[2] Note that these ciphers' MixColumns are composed of 16-bit matrix. In order to exhibit the universality of our method, we also experimented on AES, and we respectively took about 5 and 10 minutes to find 4- and 5-round integral distinguishers in the key-dependent scenario. Our distinguishers are consistent with that of the $\mathcal{HW}$ method, but we took less time. Unfortunately, we can not obtain these results when using the $\mathcal{S}$ method.

Table 1. Comparison of our results with the previous works.

Ciphers	#Rounds	$\log_2$(Data)	#Balanced Bits	Time	Ref.
Midori64	5	12	4	-	[9]
	5	12	**7**	12s	**Sect. 4.1**
	6	45	16	-	[9]
	6	45	**19**	140s	**Sect. 4.1**
	7	63	64	-	[11]
	7†	63	64	427s	**Sect. 4.1**
Skinny64	8	56	40	-	[9]
	8	56	**64**	< 2s	**Sect. 4.2**
	10	60	64	-	[11]
	10	60	64	5.5s	**Sect. 4.2**
LED	6	52	– ‡	-	[9]
	6	52	64	15min	[12]
	6	52	64	< 4h	**Sect. 4.3**
	7	63	64	14min	[12]
	7	63	64	< 5h	**Sect. 4.3**

† In [9], Sun *et al.* presented the 7-round distinguisher with 61 active input bits and 16 balanced output bits. We also found this distinguisher and failed to explore more balanced bits using 61 active bits. For the case of setting 63 active input bits, [9] does not give the relevant result.

‡ For the data of 2^{52}, their model did not return any results.

Algebraic Normal Form. Any Boolean funciton $f : \mathbb{F}_2^n \to \mathbb{F}_2$ can be represented in its Algebraic Normal Form (ANF) as:

$$f(\boldsymbol{x}) = \bigoplus_{\boldsymbol{u}\in\mathbb{F}_2^n} a_{\boldsymbol{u}}^f (\prod_{i=0}^{n-1} x_i^{u_i}) = \bigoplus_{\boldsymbol{u}\in\mathbb{F}_2^n} a_{\boldsymbol{u}}^f \pi_{\boldsymbol{u}}(\boldsymbol{x}),$$

where $a_{\boldsymbol{u}}^f \in \mathbb{F}_2$ is a constant depending on f and $\boldsymbol{u}$.

2.1 (Bit-Based) Division Property and Its MILP-aided Applications

At EUROCRYPT 2015, division property [5] was proposed by Todo as a generalization of the integral property, which was originally defined at word level. Later, the bit-based division property [7] was introduced by Todo and Morii to investigate the DP at bit level. Note that the BDP is composed of two members, two-subset BDP and three-subset BDP. In this paper, we only focus on the two-subset BDP, thus we straightforwardly use BDP to represent the two-subset BDP for short. The definition of BDP is presented as follows.

Definition 1 (Bit-Based Division Property [7]). *Let $\mathbb{X}$ be a multiset whose elements belong to $\mathbb{F}_2^n$ and $\mathbb{K}$ be a set of n-bit vectors whose elements take the value 0 or 1. Then we call the multiset $\mathbb{X}$ has the division property $\mathcal{D}_{\mathbb{K}}^{1^n}$ if it fulfills the following conditions for any $\boldsymbol{u} \in \mathbb{F}_2^n$:*

$$\bigoplus_{\boldsymbol{x}\in\mathbb{X}} \pi_{\boldsymbol{u}}(\boldsymbol{x}) = \begin{cases} unknown & if\ there\ exists\ \boldsymbol{k} \in \mathbb{K}\ s.t.\ \boldsymbol{u} \succeq \boldsymbol{k}, \\ 0 & otherwise. \end{cases}$$

As a more accurate DP, the BDP can be applied to search for better integral distinguishers. However, the application of BDP is greatly limited by its high time and memory complexities, which caused the fact that the BDP was only applicable to the ciphers with block sizes no more than 32. In order to overcome this drawback, Xiang *et al.* [8] first adopted MILP-aided method, which has shown its great power in cryptanalysis such as [20–23], to automatically search for integral distinguishers based on BDP. Moreover, the concept of *division trail* was introduced to descibe the BDP propagation and the MILP-aided modeling rules were proposed. In this paper, we devote our attention to modeling linear layers, thus we only revisit the modeling rules of XOR and COPY operations as well as the definition of division trail as follows. One can refer to [8, 9] for more details about the modeling rules of AND operation and Sboxes.

Definition 2 (Division Trail [8]). *Let f_r denote the round function of an iterated block cipher with size of n. Assume the input multiset to the block cipher has initial division property $\mathcal{D}_{\mathbb{K}_0}^{1^n}$, and denote the division property after i-round propagation through f_r by $\mathcal{D}_{\mathbb{K}_i}^{1^n}$. Thus, we have the following chain of division property propagations:*

$$\{\boldsymbol{k}\} \triangleq \mathbb{K}_0 \xrightarrow{f_r} \mathbb{K}_1 \xrightarrow{f_r} \mathbb{K}_i \xrightarrow{f_r} \cdots \mathbb{K}_r.$$

Moreover, for any vector $\boldsymbol{k}_i^*$ in $\mathbb{K}_i^*$, there must exist a vector $\boldsymbol{k}_{i-1}^*$ in $\mathbb{K}_{i-1}^*$ such that $\boldsymbol{k}_{i-1}^*$ can propagate to $\boldsymbol{k}_i^*$ by division property propagation rules. Furthermore, for $(\boldsymbol{k}_0, \boldsymbol{k}_1, \cdots, \boldsymbol{k}_r) \in \mathbb{K}_0 \times \mathbb{K}_1 \times \cdots \times \mathbb{K}_r$, if $\boldsymbol{k}_{i-1}$ can propagate to $\boldsymbol{k}_i$ for all $i \in \{1, 2, \cdots, r\}$ we call $(\boldsymbol{k}_0, \boldsymbol{k}_1, \cdots, \boldsymbol{k}_r)$ an r-round division trail.

Proposition 1 (MILP Modeling Rule for COPY [9]). *Denote a $\xrightarrow{COPY}$ $(b_0, b_1, \cdots, b_{m-1})$ a division trail of COPY function, the following inequalities are sufficient to describe the division propagation of COPY:*

$$\begin{cases} a - b_0 - b_1 - \cdots - b_{m-1} = 0, \\ a, b_0, b_1, \cdots, b_{m-1}\ are\ binaries. \end{cases}$$

Proposition 2 (MILP Modeling Rule for XOR [9]). *Denote $(a_0, a_1, \cdots, a_{m-1}) \xrightarrow{XOR} b$ a division trail of XOR function, the following inequalities can describe the division propagation of XOR:*

$$\begin{cases} a_0 + a_1 + \cdots + a_{m-1} - b = 0, \\ a_0, a_1, \cdots, a_{m-1}, b\ are\ binaries. \end{cases}$$

For the sake of convenience, we name the combined utilization of COPY and XOR rules as **CX** rules in this paper. Note that the $\mathcal{S}$ method is based on the **CX** rules to model the BDP propagation of linear layers.

2.2 Heuristics for Optimizing the Implementations of Linear Layers

The linear layer of a symmetric cipher can be represented as a matrix over $\mathbb{F}_2$, whose implementation is a sequence of XOR operations. Thus, the implementation cost of a linear layer can be estimated by the number of XOR gates required to implement the corresponding matrix. In order to find an optimized implementation of a given matrix with fewer XOR gates, several heuristics have been proposed, the widely used three of which are introduced in [14–16] respectively.

The Paar algorithm [14]. Taking the matrix over $\mathbb{F}_2$ as the input, in each step, the **Paar** algorithm chooses a pair of columns from the matrix exhaustively and calculates the bitwise AND of these two columns, the pair of columns whose AND reaches the largest Hamming weight will be kept and their bitwise AND will be added to the matrix as a new column. Before choosing the next two columns from the new matrix, we should update the matrix by XORing the selected two columns with the newly added column. The above steps will be repeated until each row of the matrix has exactly one "1".

Note that the **Paar** heuristic is *cancellation-free*, it means that the operands of any operation given by this method share no common variables. Each time the matrix is updated, both the AND of the last column and the updated two columns will lead to zero vectors. Therefore, these columns will never be selected as the operands of any operation in the subsequent implementation, i.e., if $a = b \oplus c$ is one of the operations, the operations such as $a \oplus b, a \oplus c$ and $b \oplus c$ will never appear afterwards.

The BP algorithm [15]. Given a matrix $M_{m\times n}$ over $\mathbb{F}_2$, let $wt(M_i)$ be the Hamming weight of the i-th row of M, where $i \in [0, m-1]$. Firstly, the **BP** algorithm defines a base S and a vector dist[]. The base S is initialized as the set of all input bits of M, i.e., $S = \{x_0, x_1, \cdots, x_{n-1}\}$. The distance vector is initialized as $dist = \{wt(M_0) - 1, wt(M_1) - 1, \cdots, wt(M_{m-1}) - 1\}$. Then, pick two elements $S[i]$ and $S[j](i \neq j)$ from S in each step and treat the XOR of $S[i]$ and $S[j]$ as a possible element which might be added to S, update the distance vector as the minimum number of XOR gates required for calculating the output bits according to the elements from S. Keep the XOR of two elements selected from base that minimizes the sum of the distance vector and add it to the base S as a new element. If there are multiple candidates, choose the one that maximizes the Euclidean Norm of the updated distance vector. Repeat the above steps until all the elements in $dist[]$ are zero.

The XZ algorithm [16]. Inspired by Gauss-Jordan elimination, Xiang *et al.* proposed several strategies to decompose an invertible matrix over $\mathbb{F}_2$ into a product of a sequence of type-1 and type-3 elementary matrices. A type-1 elementary matrix costs no XOR gate since it is produced by exchanging two rows/columns of an identity matrix, while a type-3 elementary matrix is produced by adding a row/column of an identity matrix to another row/column and

thus costs one XOR gate. Therefore, the matrix decomposition theory builds a relationship between the cost for implementing an invertible matrix and the number of type-3 elementary matrices in its decomposition. In order to further reduce the number of type-3 elementary matrices, Xiang *et al.* summarized seven reduction rules according to the properties of matrix multiplication, which contribute to saving XOR gates when implementing a matrix. Based on those observations and combined with the rules of exchanging the order of two adjacent elementary matrices presented in [16], Xiang *et al.* designed a heuristic to search efficient implementations of a given matrix, and an improved implementation with fewer XOR gates can be expected by running the algorithm multiple times.

In this paper, we utilize the source codes to implement those algorithms given in [16] and [24].

3 BDP Propagations Based on Linear Layer Optimization

In this section, we propose a new method to characterize BDP propagations of linear layers which is based on the **CX** rules and the optimized implementations of matrices. Firstly, we start by introducing an example to intuitively show the details and effects of the new method. Then we discuss and analyze the results obtained by using different optimized implementations to model linear transformation. Finally, we give a theoretical argument to prove that our new method will never be worse than the $\mathcal{S}$ method. This paper mainly considers three heuristics for optimizing the implementations of matrices, i.e., the **Paar**, the **BP** and the **XZ** algorithms. For convenience, we use **Paar** + **CX** to represent the combination of the **Paar** algorithm with the **CX** rules. Other heuristics combined with the **CX** rules are also denoted similarly.

3.1 Construct BDP Propagation Models of Linear Layers

We will begin with a small example listed in the following to illustrate our idea.

Example 1. Let $L : (x_0, x_1, x_2, x_3) \mapsto (x_1 \oplus x_2 \oplus x_3, x_0 \oplus x_2 \oplus x_3, x_0 \oplus x_1 \oplus x_3, x_0 \oplus x_1 \oplus x_2)$ be a linear transformation on $\mathbb{F}_2^4$, the corresponding matrix M is as follows:

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

Assuming that $(u_0, u_1, u_2, u_3) \to (v_0, v_1, v_2, v_3)$ is a division trail through L. We compute and list all possible division trails of M according to the $\mathcal{S}$ method, $\mathcal{ZR}$ method and our technique.

(1) **The $\mathcal{S}$ method**: Based on the theory of the $\mathcal{S}$ method to characterize linear layers, we introduce s_k's as auxiliary binary variables, and the inequalities

within the model are as follows:

$$\begin{cases} u_0 - s_0 - s_1 - s_2 = 0 \\ u_1 - s_3 - s_4 - s_5 = 0 \\ u_2 - s_6 - s_7 - s_8 = 0 \\ u_3 - s_9 - s_{10} - s_{11} = 0 \\ v_0 - s_3 - s_6 - s_9 = 0 \\ v_1 - s_0 - s_7 - s_{10} = 0 \\ v_2 - s_1 - s_4 - s_{11} = 0 \\ v_3 - s_2 - s_5 - s_8 = 0 \end{cases}. \tag{1}$$

Among them, $u_i, v_j, s_k (0 \leq i, j \leq 3,\ 0 \leq k \leq 11)$ are binary variables. By solving the above inequalities, the division trails of M can be obtained. The results are listed in Table 2.

Table 2. Division trails of different method. The trails highlighted in red are invalid.

Input	$\mathcal{S}$ **method**	**Our method**	$\mathcal{ZR}$ **method**
$0x0$	$0x0$	$0x0$	$0x0$
$0x1$	$0x2$, $0x4$, $0x8$	$0x2$, $0x4$, $0x8$	$0x2$, $0x4$, $0x8$
$0x2$	$0x1$, $0x4$, $0x8$	$0x1$, $0x4$, $0x8$	$0x1$, $0x4$, $0x8$
$0x3$	$0x3$, $0x5$, $0x6$, $0x9$, $0xA$, $0xC$	$0x3$, $0x5$, $0x6$, $0x9$, $0xA$	$0x3$, $0x5$, $0x6$, $0x9$, $0xA$
$0x4$	$0x1$, $0x2$, $0x8$	$0x1$, $0x2$, $0x8$	$0x1$, $0x2$, $0x8$
$0x5$	$0x3$, $0x5$, $0x6$, $0x9$, $0xA$, $0xC$	$0x3$, $0x5$, $0x6$, $0x9$, $0xC$	$0x3$, $0x5$, $0x6$, $0x9$, $0xC$
$0x6$	$0x3$, $0x5$, $0x6$, $0x9$, $0xA$, $0xC$	$0x3$, $0x5$, $0x6$, $0xA$, $0xC$	$0x3$, $0x5$, $0x6$, $0xA$, $0xC$
$0x7$	$0x7$, $0xB$, $0xD$, $0xE$	$0xB$, $0xD$, $0xE$	$0xB$, $0xD$, $0xE$
$0x8$	$0x1$, $0x2$, $0x4$	$0x1$, $0x2$, $0x4$	$0x1$, $0x2$, $0x4$
$0x9$	$0x3$, $0x5$, $0x6$, $0x9$, $0xA$, $0xC$	$0x3$, $0x5$, $0x9$, $0xA$, $0xC$	$0x3$, $0x5$, $0x9$, $0xA$, $0xC$
$0xA$	$0x3$, $0x5$, $0x6$, $0x9$, $0xA$, $0xC$	$0x3$, $0x6$, $0x9$, $0xA$, $0xC$	$0x3$, $0x6$, $0x9$, $0xA$, $0xC$
$0xB$	$0x7$, $0xB$, $0xD$, $0xE$	$0x7$, $0xD$, $0xE$	$0x7$, $0xD$, $0xE$
$0xC$	$0x3$, $0x5$, $0x6$, $0x9$, $0xA$, $0xC$	$0x5$, $0x6$, $0x9$, $0xA$, $0xC$	$0x5$, $0x6$, $0x9$, $0xA$, $0xC$
$0xD$	$0x7$, $0xB$, $0xD$, $0xE$	$0x7$, $0xB$, $0xE$	$0x7$, $0xB$, $0xE$
$0xE$	$0x7$, $0xB$, $0xD$, $0xE$	$0x7$, $0xB$, $0xD$	$0x7$, $0xB$, $0xD$
$0xF$	$0xF$	$0xF$	$0xF$

(2) **Our method**: Our new technique to model the BDP propagation of M needs to combine optimized implementations of M with the **CX** rules. Thus, we first need to obtain several optimized implementations of M, then use the **CX** rules to model its BDP propagations. We detail this process in the following two

phases.

Implementating Phase: In this phase, we use the algorithms of **Paar**, **BP** and **XZ** to get the optimized implementations of M. The implementations of M are listed in Equation (2), where the first set of inequalities denotes the direct implementation and others are implemented by **Paar**, **BP** and **XZ**, respectively.

$$\begin{cases} y_0 = x_1 \oplus x_2 \oplus x_3 \\ y_1 = x_0 \oplus x_2 \oplus x_3 \\ y_2 = x_0 \oplus x_1 \oplus x_3 \\ y_3 = x_0 \oplus x_1 \oplus x_2 \end{cases} \begin{cases} t_0 = x_0 \oplus x_1 \\ t_1 = x_2 \oplus x_3 \\ y_0 = x_1 \oplus t_1 \\ y_1 = x_0 \oplus t_1 \\ y_2 = x_3 \oplus t_0 \\ y_3 = x_2 \oplus t_0 \end{cases} \begin{cases} t_0 = x_0 \oplus x_2 \\ y_3 = x_1 \oplus t_0 \\ y_1 = x_3 \oplus t_0 \\ t_1 = y_3 \oplus y_1 \\ y_2 = x_0 \oplus t_1 \\ y_0 = t_0 \oplus y_2 \end{cases} \begin{cases} t_0 = x_1 \oplus x_2 \\ t_1 = x_0 \oplus x_3 \\ y_1 = x_2 \oplus t_1 \\ y_0 = x_3 \oplus t_0 \\ y_2 = t_0 \oplus y_1 \\ y_3 = t_1 \oplus y_0 \end{cases}. \quad (2)$$

Modeling Phase: We combine the **CX** rules with the three optimized implementations of M listed in Equation (2) to model BDP propagations of M. This process needs to introduce binary auxiliary variables s_k's. These three set of inequalities contained in Equation (3) are the constraints, which are constructed according to the **Paar**, the **BP** and the **XZ** implementations respectively. In order to illustrate the modeling details, we take the **Paar** as an example. From the implementation, we know that x_0 and x_1 both appear twice, thus we need to copy u_0 and u_1 to two pieces: s_0, s_1 and s_2, s_3, where the variable u_i denotes the DP of x_i. This corresponds to the inequalities $u_0 - s_0 - s_1 = 0$ and $u_1 - s_2 - s_3 = 0$. The variable s_4 denotes the division property of the intermediate variable t_0 and s_4 is generated by the XOR operation of the copied DP from u_0 and u_1 i.e., s_0 and s_2, thus $s_4 - s_0 - s_2 = 0$. Note that t_0 is reused in the 5-th and 6-th equations. Thus, we need to copy s_4 to two pieces: s_{12} and s_{13}, which corresponds to the inequality $s_4 - s_{12} - s_{13} = 0$. Other equations can be modeled in a similar way, so we omit the details.

$$\begin{cases} u_0 - s_0 - s_1 = 0 \\ u_1 - s_2 - s_3 = 0 \\ s_2 + s_0 - s_4 = 0 \\ u_2 - s_5 - s_6 = 0 \\ u_3 - s_7 - s_8 = 0 \\ s_7 + s_5 - s_9 = 0 \\ s_9 - s_{10} - s_{11} = 0 \\ s_{10} + s_3 - v_0 = 0 \\ s_{11} + s_1 - v_1 = 0 \\ s_4 - s_{12} - s_{13} = 0 \\ s_{12} + s_8 - v_2 = 0 \\ s_{13} + s_6 - v_3 = 0 \end{cases} \begin{cases} u_0 - s_{14} - s_{15} = 0 \\ u_2 + s_{14} - s_{16} = 0 \\ s_{16} - s_{17} - s_{18} = 0 \\ s_{19} - v_3 - s_{20} = 0 \\ s_{17} + u_1 - s_{19} = 0 \\ s_{18} - s_{21} - s_{22} = 0 \\ s_{23} - v_1 - s_{24} = 0 \\ s_{21} + u_3 - s_{23} = 0 \\ s_{24} + s_{20} - s_{25} = 0 \\ s_{26} - v_2 - s_{27} = 0 \\ s_{25} + s_{15} - s_{26} = 0 \\ s_{27} + s_{22} - v_0 = 0 \end{cases} \begin{cases} u_2 - s_{28} - s_{29} = 0 \\ s_{28} + u_1 - s_{30} = 0 \\ u_3 - s_{31} - s_{32} = 0 \\ s_{31} + u_0 - s_{33} = 0 \\ s_{33} - s_{34} - s_{35} = 0 \\ s_{36} - v_1 - s_{37} = 0 \\ s_{34} + s_{29} - s_{36} = 0 \\ s_{30} - s_{38} - s_{39} = 0 \\ s_{40} - v_0 - s_{41} = 0 \\ s_{38} + s_{32} - s_{40} = 0 \\ s_{37} + s_{39} - v_2 = 0 \\ s_{41} + s_{35} - v_3 = 0 \end{cases}. \quad (3)$$

Among them, $u_i, v_j, s_k (0 \leq i, j \leq 3,\ 0 \leq k \leq 41)$ are binary variables. The procedure for constructing the BDP propagation model of a matrix by our technique is shown in Algorithm 1.

Algorithm 1 traverses each row of a given matrix implementation and models it one by one. We explain Algorithm 1 in the following three steps. **Step 1**: Traverse the right side of an XOR operation within the matrix implementation from Line 11 to 31. If there is a variable reused in the subsequent matrix implementation, its DP needs to be copied and we should model this using the COPY rule (Proposition 1 in Subsection 2.1). During this process, two new variables are generated, and the first new variable replaces the first occurrence of the considered variable's DP which is copied. Note that if a variable is reused in the subsequent matrix implementation, its DP may be reused for multiple times. We do not count the exact occurrences of this variable. We instead add a new term $temp[L[i][j]] = t_1$ into the dictionary. With this new term, we can make sure that $L[i][j]$ will be reused in the subsequent matrix implementation and its DP has been copied earlier. Thus, each time we have to check if $L[i][j]$ is an index of the dictionary, if this is the case, we should use the copied piece stored in the dictionary instead. **Step 2**: Traverse the left side variable of an XOR operation within the implementation from Line 32 to 40. If this variable is equal to an output bit of the matrix and reused in the subsequent implementation of a matrix, the variable's DP is copied. During the COPY operation, this will generate two new variables, the first new variable replaces the first occurrence's DP and the second new variable will be used to model the following occurrence's DP as in Step 1. **Step 3**: Line 41 performs an XOR operation on the updated variables in each line and adds them into the model. Finally, this algorithm will return an entire model $\mathcal{M}$ from Line 43.

Note that Line 22 and 36 are quite different, this is because if an output bit is reused, we should use one piece of the COPY operation to represent the output DP, as the other piece will be used to compute the following output bits.

3.2 Division Trails of Different Models

With the help of automatic solvers, such as Gurobi, Equation (1) and (3) can be solved and the obtained solutions are division trails. Table 2 and 3 list the division trails of various methods, in which the binary representation of division trails is equivalent to the hexadecimal representation, denoted as: $(u_0, u_1, u_2, u_3) = (1, 0, 0, 0) \stackrel{\triangle}{=} 0x8$.

Comparing the division trails obtained by various methods in Table 2 and 3, it can be found that the division trails obtained using $\mathbf{Paar} + \mathbf{CX}$ are less than those obtained by the $\mathcal{S}$ method. For $\mathbf{BP} + \mathbf{CX}$ and $\mathbf{XZ} + \mathbf{CX}$, both methods can eliminate some (invalid) division trails. However, they will also introduce some new (invalid) division trails at the same time.

Note that Zhang and Rijmen [11] presented a theoretical technique to determine if a division trail is valid, which computes the determinant of the submatrix defined by the input and the output division property, or equivalently

Algorithm 1 Construct the MILP model of linear layer BDP propagation

Input: A matrix implementation
Output: The MILP model of BDP propagation $\mathcal{M}$

```
count = 0;
L ← Read the matrix implementation by row, add the variables of each row to the
  corresponding row of the two-dimensional list L from left to right;
temp ← dict(); //Initialized as an empty dictionary
function Get_new_var( )
    new_var = s_count;
    count = count + 1;
    return new_var;
end function
M.var ← u_i, v_i, new_var; //u_i and v_i represent the input and output division
  property, new_var denotes a newly generated binary variable
for i = 0; i < len(L) do
    for j = 1; j < len(L[i]) do
        if L[i][j] appears in the k-th(k > i) row of L then
            if L[i][j] is not an index of temp then
                t_0 = Get_new_var( );
                t_1 = Get_new_var( );
                M.con ← L[i][j]' = t_0 + t_1; //L[i][j]' represent the DP of L[i][j]
                temp[L[i][j]] = t_1;
                L[i][j]' = t_0 ;
            else
                t'_0 = Get_new_var( );
                t'_1 = Get_new_var( );
                M.con ← temp[L[i][j]] = t'_0 + t'_1;
                temp[L[i][j]] = t'_1;
                L[i][j]' = t'_0 ;
            end if
        else
            if L[i][j] in an index of temp then
                L[i][j] = temp[L[i][j]];
            end if
        end if
    end for
    if L[i][0] = Output then
        if L[i][0] appears in the k-th(k > i) row of L then
            t_0 = Get_new_var( );
            t_1 = Get_new_var( );
            M.con ← t_0 = L[i][0]' + t_1; //L[i][0]' represent the DP of L[i][0]
            temp[L[i][j]] = t_1;
            temp ← Store [L[i][0], t_1] in temp;
            L[i][0]' = t_0;
        end if
    end if
    M.con ← L[i][0]' = L[i][1]' + L[i][2]';
end for
return M;
```

Table 3. Division trails of different implementations. The trails highlighted in red are invalid.

Input	**Paar + CX**	**BP + CX**	**XZ + CX**
$0x0$	$0x0$	$0x0$	$0x0$
$0x1$	$0x2$, $0x4$, $0x8$	$0x2$, $0x4$, $0x8$	$0x1$, $0x2$, $0x4$, $0x8$
$0x2$	$0x1$, $0x4$, $0x8$	$0x1$, $0x2$, $0x4$, $0x8$	$0x1$, $0x2$, $0x4$, $0x8$
$0x3$	$0x3$, $0x5$, $0x6$, $0x9$, $0xA$	$0x3$, $0x5$, $0x6$, $0x9$, $0xA$, $0xC$	$0x3$, $0x5$, $0x6$, $0x9$, $0xA$, $0xC$
$0x4$	$0x1$, $0x2$, $0x8$	$0x1$, $0x2$, $0x8$	$0x1$, $0x2$, $0x8$
$0x5$	$0x3$, $0x5$, $0x6$, $0x9$, $0xA$, $0xC$	$0x3$, $0x5$, $0x6$, $0x9$, $0xC$	$0x3$, $0x5$, $0x6$, $0x9$, $0xA$, $0xC$
$0x6$	$0x3$, $0x5$, $0x6$, $0x9$, $0xA$, $0xC$	$0x3$, $0x5$, $0x6$, $0x9$, $0xA$, $0xC$	$0x3$, $0x5$, $0x6$, $0xA$, $0xC$
$0x7$	$0x7$, $0xB$, $0xD$, $0xE$	$0xB$, $0xD$, $0xE$	$0x7$, $0xB$, $0xD$, $0xE$
$0x8$	$0x1$, $0x2$, $0x4$	$0x1$, $0x2$, $0x4$, $0x8$	$0x1$, $0x2$, $0x4$
$0x9$	$0x3$, $0x5$, $0x6$, $0x9$, $0xA$, $0xC$	$0x3$, $0x5$, $0x6$, $0x9$, $0xA$, $0xC$	$0x3$, $0x5$, $0x9$, $0xA$, $0xC$
$0xA$	$0x3$, $0x5$, $0x6$, $0x9$, $0xA$, $0xC$	$0x3$, $0x6$, $0x9$, $0xA$, $0xC$	$0x3$, $0x5$, $0x6$, $0x9$, $0xA$, $0xC$
$0xB$	$0x7$, $0xB$, $0xD$, $0xE$	$0x7$, $0xD$, $0xE$	$0x7$, $0xB$, $0xD$, $0xE$
$0xC$	$0x5$, $0x6$, $0x9$, $0xA$, $0xC$	$0x3$, $0x5$, $0x6$, $0x9$, $0xA$, $0xC$	$0x3$, $0x5$, $0x6$, $0x9$, $0xA$, $0xC$
$0xD$	$0x7$, $0xB$, $0xD$, $0xE$	$0x7$, $0xB$, $0xD$, $0xE$	$0x7$, $0xB$, $0xE$
$0xE$	$0x7$, $0xB$, $0xD$, $0xE$	$0x7$, $0xB$, $0xD$	$0x7$, $0xB$, $0xD$
$0xF$	$0xF$	$0xF$	$0xF$

checks if the ANF of the output (defined by the output division property) contains the input monomial (defined by the input division property). Since the $\mathcal{S}$ method corresponds to direct implementation of a matrix, and our technique uses an optimized implementation, division trails obtained by these methods are reasonable characterizations of BDP propagations (in the sense that this may introduce invalid division trails). Thus, we can conclude that all increased division trails using our method are invalid trails compared with the $\mathcal{S}$ method. Let's consider the division trail $0x2 \xrightarrow{M} 0x2$ obtained by $\mathbf{BP} + \mathbf{CX}$. This is a newly increased trail compared with the $\mathcal{S}$ method. The output bit defined by the output division property is y_2, which equals to $x_0 \oplus x_1 \oplus x_3$ according to the matrix. Obviously, the ANF of this output bit does not contain x_2, which indicates that $0x2 \xrightarrow{M} 0x2$ is an invalid trail. However, we can get a deeper look of this trail. According to the implementation returned by the **BP** algorithm, y_2 is computed as $t_0 = x_0 \oplus x_2, y_3 = x_1 \oplus t_0, y_1 = x_3 \oplus t_0, t_1 = y_3 \oplus y_1, y_2 = x_0 \oplus t_1$. The input division property of the matrix is $0x2$, thus the division property of x_2 is 1 which can propagate to t_0, and this can further propagate to y_3 and y_1, which finally propagate to y_2. This leads to the increased invalid division trail.

On the other hand, since each set of inequalities as listed in Equation (1) and (3) can describe the BDP propagation of M in a non-accurate way, all de-

creased division trails of our method are all invalid trails. Let's take the division trail $0xC \xrightarrow{M} 0x3 \triangleq (1,1,0,0) \xrightarrow{M} (0,0,1,1)$ as an example. According to the matrix implementation of Equation (2), we can compute the ANF of y_2y_3, i.e., $y_2y_3 = (x_0 \oplus x_1 \oplus x_3)(x_0 \oplus x_1 \oplus x_2) = x_0 \oplus x_0x_1 \oplus x_0x_2 \oplus x_0x_1 \oplus x_1 \oplus x_1x_2 \oplus x_0x_3 \oplus x_1x_3 \oplus x_2x_3$. It is easy to find that x_0x_1 appears twice, and the monomial will be cancelled after the XOR operation, then y_2y_3 does not include x_0x_1, so the input division property $(1,1,0,0)$ can not propagate to the output division property $(0,0,1,1)$. Therefore, this trail is not a valid propagation according to Zhang and Rijmen's theory. Similarly, it can be concluded that all decreased trails are invalid.

Since $\mathbf{Paar} + \mathbf{CX}$ method reduces two invalid trails, and both $\mathbf{BP} + \mathbf{CX}$ and $\mathbf{XZ} + \mathbf{CX}$ methods introduce new invalid trails at the same time when reducing invalid trails. It seems that one should prefer $\mathbf{Paar} + \mathbf{CX}$ to $\mathbf{BP} + \mathbf{CX}$ and $\mathbf{XZ} + \mathbf{CX}$. However, $\mathbf{Paar} + \mathbf{CX}$ method only reduces two trails which is not satisfactory. In the following, we combine all of the three methods. Note that each method $(\mathbf{Paar}, \mathbf{BP}, \mathbf{XZ} + \mathbf{CX})$ can describe the BDP propagation of M (in a non-accurate way), which means all valid trails of M should be contained in the trail set obtained by each of the three methods. Thus, the intersection of the three trail sets can characterize the BDP propagation. This has the advantage that each invalid trail eliminated by one of the methods will not be included in the intersection, thus eliminated. In practice, the trail set of each method is obtained by solving the corresponding system of inequalities. Therefore,we can gather the latter three sets of the inequalities listed in Equation (2) as a whole, and the solutions of this model are the common solutions of the three methods, that is, the intersection of the solutions as desired. The column marked with "$\mathbf{Paar} + \mathbf{BP} + \mathbf{XZ} + \mathbf{CX}$" in Table 2 lists the results of this combination. The results show that it can reduce a large number of invalid trails, which are fully identical to results by the $\mathcal{ZR}$ method as shown in the 4-th column of Table 2. In Table 3, we can clearly see that the optimized implementations of different algorithms can eliminate some different invalid trails. Due to the randomness of these algorithms and the linearly growth of modeling implementations using the **CX** rules, it is possible to use these algorithms multiple times to obtain multiple matrix implementations, and add the corresponding inequalities into the model to get a more accurate propagation, which enables us to search for more available integral properties.

3.3 On the Effectiveness of Our Method

The example discussed in Subsection 3.2 shows that $\mathbf{BP} + \mathbf{CX}$ or $\mathbf{XZ} + \mathbf{CX}$ method would possibly introduce some new invalid trails, even though they could eliminate several invalid trails at the same time. Thus, we can not conclude that $\mathbf{BP} + \mathbf{CX}$ or $\mathbf{XZ} + \mathbf{CX}$ method is better than the $\mathcal{S}$ method. As we discussed in Subsection 3.2, the division trail $0x2 \xrightarrow{M} 0x2$ is a newly introduced invalid trail, and this happens because two x_2's are involved in the computation of y_2 and they will be cancelled by XOR. According to the computation process of the

Paar algorithm described in Subsection 2.2, this algorithm outputs *cancellation-free* implementations. That is, x_2 will never appear in the computation of y_2. Moreover, we can find in the example listed in Subsection 3.2 that **Paar + CX** method only eliminates invalid trails and no new invalid trails are introduced compared with the $\mathcal{S}$ method. We will discuss in this subsection and conclude that **Paar + CX** method is always superior to the $\mathcal{S}$ method, i.e., the trail set computed by **Paar + CX** method is a subset of the trail set obtained by the $\mathcal{S}$ method.

Let's first revisit Zhang and Rijmen's theory and the $\mathcal{S}$ method.

Theorem 1 ([11]). *Let $M = (a_{i,j})$ be the $n \times n$ matrix of an invertible linear transform. Let $(\boldsymbol{u}, \boldsymbol{v}) = (u_1, \cdots, u_n, v_1, \cdots, v_n) \in \mathbb{F}_2^n \times \mathbb{F}_2^n$, $I_{\boldsymbol{u}} = \{i, u_i = 1\} = \{i_1, \cdots, i_{wt(\boldsymbol{u})}\}$, $I_{\boldsymbol{v}} = \{j, v_j = 1\} = \{j_1, \cdots, j_{wt(\boldsymbol{v})}\}$. Then $(\boldsymbol{u}, \boldsymbol{v})$ is a valid division trail if and only if the order $wt(\boldsymbol{u})$ sub-matrix whose rows indices are taken from $I_{\boldsymbol{u}}$ and columns indices are taken from $I_{\boldsymbol{v}}$ is invertible.*

Let $P = (p_{i,j})_{n\times n}$ and $Q = (q_{i,j})_{n\times n}$ be two $n \times n$ matrices, we denote $P\&Q = (p_{i,j} \times q_{i,j})_{n\times n}$ an $n \times n$ matrix which is the element-wise AND of P and Q. In the following, we say that P contains Q or Q is contained in P if $P\&Q = Q$. Given the modeling process of the $\mathcal{S}$ method, we can easily deduce the following proposition.

Proposition 3. *Let $M = (a_{i,j})$ be the $n \times n$ matrix of an invertible linear transform. Let $(\boldsymbol{u}, \boldsymbol{v}) = (u_1, \cdots, u_n, v_1, \cdots, v_n) \in \mathbb{F}_2^n \times \mathbb{F}_2^n$, $I_{\boldsymbol{u}} = \{i, u_i = 1\} = \{i_1, \cdots, i_{wt(\boldsymbol{u})}\}$, $I_{\boldsymbol{v}} = \{j, v_j = 1\} = \{j_1, \cdots, j_{wt(\boldsymbol{v})}\}$. Then $(\boldsymbol{u}, \boldsymbol{v})$ is a division trail of $\mathcal{S}$ method if and only if the order $wt(\boldsymbol{u})$ sub-matrix whose rows indices are taken from $I_{\boldsymbol{u}}$ and columns indices are taken from $I_{\boldsymbol{v}}$ contains a permutation matrix.*

From Theorem 1 and Proposition 3, we can conclude that each valid division trail (indicated by Theorem 1) is a division trail of the $\mathcal{S}$ method, since each invertible matrix must contain a permutation matrix. Conversely, each division trail of the $\mathcal{S}$ method is not necessarily a valid division trail. Consider a matrix whose all elements being 1. This matrix is not an invertible matrix, however this matrix contains the identity matrix as a permutation matrix.

Since the **Paar** algorithm is *cancellation-free*, if we compute an output bit y_i from the reverse order of a given implementation, input variables that are not contained in the ANF of y_i will never appear. For example, if we compute y_2 from the **Paar** implementation in Equation (2), $y_2 = x_3 \oplus t_0 = x_0 \oplus x_1 \oplus x_3$. Thus, x_2 is not involved. However, if we consider y_2 from the **BP** implementation, $y_2 = x_0 \oplus t_1 = x_0 \oplus y_3 \oplus y_1 = x_0 \oplus x_1 \oplus t_0 \oplus x_3 \oplus t_0 = x_0 \oplus x_1 \oplus x_3 \oplus x_0 \oplus x_2 \oplus x_0 \oplus x_2$, and x_2 is involved. Given the *cancellation-free* property of the **Paar** algorithm, it can be easily concluded.

Property 1. *If an output bit has the division property of 1, one of the inputs involved in its ANF has the division property of 1.*

Moreover, assuming that two output bits y_i and y_j share several common input bits, and both y_i and y_j have input division property of 1. According to Property 1, the division property of y_i and y_j being 1 will indicate two input bits taking the division property of 1. Moreover, these two input bits cannot be the same. This is due to the fact that if any input bit x_k is involved in the ANFs of y_i and y_j, we should use the COPY operation to split the division property of x_k , and both y_i and y_j will take a piece of x_k. Thus, if y_i indicates that x_k has the division property of 1, which means the piece of x_k fed to y_i has division property of 1. Thus, the other piece fed to y_j will never take the division property of 1.

Property 2. *Different output bits taking the division property of 1 will indicate different input bits taking the division property of 1.*

Given a division trail $\boldsymbol{u} \rightarrow \boldsymbol{v}$ deduced from $\mathbf{Paar} + \mathbf{CX}$ method, let w denote the Hamming weight of $\boldsymbol{u}$ (and $\boldsymbol{v}$), and $i_1, \cdots, i_w (i_1 < i_2 < \cdots < i_w), j_1, \cdots, j_w (j_1 < j_2 < \cdots < j_w)$ denote the indices of $\boldsymbol{u}$ and $\boldsymbol{v}$ whose corresponding coordinates take the division property of 1. According to Property 2, we can pair $x_{i_1}, \cdots, x_{i_w}$ and $y_{j_1}, \cdots, y_{j_w}$, where x_{i_w} and y_{i_w} denote the input and out variables. For the sake of simplicity, assume that x_{i_s} and y_{j_s} form a pair, which means x_{i_s} is involved in the ANF of y_{j_s}. Thus, $M[j_s][i_s] = 1$ where $M[j_s][i_s]$ denotes the elment in the j_s-th row and i_s-th column of M. Let $M_{\boldsymbol{v},\boldsymbol{u}}$ denote the $w \times w$ sub-matrix of M as explained in Proposition 3, then this matrix contains the identity matrix as a permutation matrix, which means each division trail deduced from $\mathbf{Paar} + \mathbf{CX}$ method is a division trail of the $\mathcal{S}$ method.

Furthermore, we consider the case where the **Paar** algorithm will generate a new column with Hamming weight greater than 1, which indicates that there is at least one $x_i \oplus x_j$ for $i < j$ appears at least twice in the ANFs of $y_0, \cdots, y_{n-1}$. Without loss of generality, assume that $x_0 \oplus x_1$ appears both in y_0 and y_1, then the sub-matrix of M with the first, second row and first, second column must be a 2×2 matrix with all elements being 1. This is not an invertible sub-matrix, however it contains the 2×2 identity matrix as a permutation matrix. Then the division trail $(1, 1, \cdots, 0, 0) \longrightarrow (1, 1, \cdots, 0, 0)$ can be deduced from the $\mathcal{S}$ method by Proposition 3. But it must not be deduced from **Paar** + **CX** method, since the division property 1 of the intermediate variable $t = x_0 \oplus x_1$ can not be copied to two pieces both with division property 1, i.e., the division property of y_0 and y_1 cannot both be 1. This means that **Paar** + **CX** method must eliminate some invalid division trails which can be deduced from the $\mathcal{S}$ method.

Proposition 4. *For any given matrix M, division trails deduced from* **Paar** + **CX** *method are all division trails of the $\mathcal{S}$ method. Moreover, when the* **Paar** *algorithm generates some new columns with Hamming weight greater than 1, division trails deduced from* **Paar** + **CX** *method must be less than division trails deduced from the $\mathcal{S}$ method.*

The above proposition guarantees that $\mathbf{Paar} + \mathbf{CX}$ method is never worse than the $\mathcal{S}$ method. In other words, $\mathbf{Paar} + \mathbf{CX}$ method is always likely to eliminate

the invalid trails and will never introduce newly invalid trails compared with the $\mathcal{S}$ method. Therefore, in a practical application, the more optimal implementation by **Paar** algorithm is, the more invalid trails will be eliminated. In addition, the **BP** + **CX** and **XZ** + **CX** methods both can eliminate some invalid trails meanwhile introduce some invalid trails compared with the $\mathcal{S}$ method. Thus, it remains unknown about the times or degrees of the optimization when running **BP** or **XZ** algorithm. In fact, we have experimented on optimizing different degrees by **BP** algorithm on LED cipher. It is true that optimizing twice can obtain the implementation with less XOR gates than only optimizing one time. Interestingly, the number of trails (valid and invalid) of the better implementation is large than the worse implementation. However, we do not know which one can eliminate more invalid trails as well as introduce less invalid trails. Thus, we only simply consider the combination of the three algorithms, which is absolutely more accurate than using only one of these algorithms.

4 Applications of Our New Technique

In this section, we show the applications of our technique to Midori64, Skinny64 and LED. Meanwhile, we also have reproduced the results of 4- and 5-round dependent-key integral distinguishers AES as reported in [12]. Our new technique mainly focuses on efficiently modeling BDP propagations of the linear layer. By combining different implementations of the linear layer, we exclude as many invalid division trails as possible, which may lead to longer distinguishers or more balanced bits compared with the $\mathcal{S}$ method. Table 4 partially lists division trails of linear layers of these block ciphers by the $\mathcal{S}$ method, the $\mathcal{ZR}$ method, $\mathcal{HW}$ method and our method. The main results for integral characteristics of these block ciphers can be found in Table 1.

Table 4. Comparison of the number of division trails by different methods.

Ciphers	Hamming weight†	$\mathcal{S}$ method	$\mathcal{ZR}$/ $\mathcal{HW}$ method	Our method
Midori64	3	12160	11280	11458
Skinny64	all‡	1500624	1185921	1185921
LED	3	144053	101938	123195

† The Hamming weight of the input division property. In our experiment, it can hardly to calculate the number of division trails when we traverse all the cases of the Hamming weight of the input division property for Midori64 and LED. Therefore, we only list the division trails that the Hamming weight of the input division property is 3 for Midori64 and LED.

‡ Traverse all the cases of the Hamming weight of the input division property.

Some special notations will be used in this section. $\mathcal{A}$, $\mathcal{B}$ and $\mathcal{U}$ indicate that a certain nibble is active, balanced and unknown, respectively. The little letters a, b and u indicate the active bit, the balanced bit and the unknown bit respectively. And we use nR to generally denote an n-round encryption where R denotes a one-round encryption.

4.1 Application to Midori64

In [9], Sun *et al.* obtained the longest 7-round integral distinguisher of Midori64. Although our method deduces distinguishers as long as the current longest one, but more balanced output bits can be obtained by our method. The 5-, 6- and 7-round integral distinguishers obtained by using the $\mathcal{S}$ method are expressed as follows.

$$\begin{bmatrix} \mathcal{A} & \mathcal{C} & \mathcal{C} & \mathcal{C} \\ \mathcal{C} & \mathcal{A} & \mathcal{C} & \mathcal{C} \\ \mathcal{C} & \mathcal{C} & \mathcal{A} & \mathcal{C} \\ \mathcal{C} & \mathcal{C} & \mathcal{C} & \mathcal{C} \end{bmatrix} \xrightarrow{5R} \begin{bmatrix} uubu & \mathcal{U} & \mathcal{U} & \mathcal{U} \\ uubu & \mathcal{U} & \mathcal{U} & \mathcal{U} \\ uubu & \mathcal{U} & \mathcal{U} & \mathcal{U} \\ uubu & \mathcal{U} & \mathcal{U} & \mathcal{U} \end{bmatrix} \begin{bmatrix} \mathcal{A} & \mathcal{A} & \mathcal{C} & accc \\ \mathcal{A} & \mathcal{A} & \mathcal{A} & \mathcal{C} \\ \mathcal{C} & \mathcal{A} & \mathcal{A} & \mathcal{A} \\ \mathcal{A} & \mathcal{C} & \mathcal{A} & \mathcal{A} \end{bmatrix} \xrightarrow{6R} \begin{bmatrix} uubu & uubu & uubu & uubu \\ uubu & uubu & uubu & uubu \\ uubu & uubu & uubu & uubu \\ uubu & uubu & uubu & uubu \end{bmatrix}$$

$$\begin{bmatrix} \mathcal{A} & \mathcal{A} & \mathcal{A} & \mathcal{A} \\ \mathcal{A} & \mathcal{A} & \mathcal{A} & \mathcal{A} \\ \mathcal{A} & \mathcal{A} & \mathcal{A} & \mathcal{A} \\ \mathcal{A} & \mathcal{A} & \mathcal{A} & ccac \end{bmatrix} \xrightarrow{7R} \begin{bmatrix} uubu & uubu & uubu & uubu \\ uubu & uubu & uubu & uubu \\ uubu & uubu & uubu & uubu \\ uubu & uubu & uubu & uubu \end{bmatrix}$$

Our 5-, 6- and 7-round integral distinguishers of Midori64 are shown as follows.

$$\begin{bmatrix} \mathcal{A} & \mathcal{C} & \mathcal{C} & \mathcal{C} \\ \mathcal{C} & \mathcal{A} & \mathcal{C} & \mathcal{C} \\ \mathcal{C} & \mathcal{C} & \mathcal{A} & \mathcal{C} \\ \mathcal{C} & \mathcal{C} & \mathcal{C} & \mathcal{C} \end{bmatrix} \xrightarrow{5R} \begin{bmatrix} uubu & \mathcal{U} & \mathcal{U} & \mathcal{U} \\ uubu & \mathcal{U} & \mathcal{U} & \mathcal{U} \\ uubu & \mathcal{U} & \mathcal{U} & \mathcal{U} \\ \mathcal{B} & \mathcal{U} & \mathcal{U} & \mathcal{U} \end{bmatrix} \begin{bmatrix} \mathcal{A} & \mathcal{A} & \mathcal{C} & accc \\ \mathcal{A} & \mathcal{A} & \mathcal{A} & \mathcal{C} \\ \mathcal{C} & \mathcal{A} & \mathcal{A} & \mathcal{A} \\ \mathcal{A} & \mathcal{C} & \mathcal{A} & \mathcal{A} \end{bmatrix} \xrightarrow{6R} \begin{bmatrix} uubu & uubu & uubu & uubu \\ uubu & uubu & uubu & uubu \\ uubu & uubu & uubu & uubu \\ \mathcal{B} & uubu & uubu & uubu \end{bmatrix}$$

$$\begin{bmatrix} aaca & \mathcal{A} & \mathcal{A} & \mathcal{A} \\ \mathcal{A} & \mathcal{A} & \mathcal{A} & \mathcal{A} \\ \mathcal{A} & \mathcal{A} & \mathcal{A} & \mathcal{A} \\ \mathcal{A} & \mathcal{A} & \mathcal{A} & \mathcal{A} \end{bmatrix} \xrightarrow{7R} \begin{bmatrix} \mathcal{B} & \mathcal{B} & \mathcal{B} & \mathcal{B} \\ \mathcal{B} & \mathcal{B} & \mathcal{B} & \mathcal{B} \\ \mathcal{B} & \mathcal{B} & \mathcal{B} & \mathcal{B} \\ \mathcal{B} & \mathcal{B} & \mathcal{B} & \mathcal{B} \end{bmatrix}$$

The experimental results show that for 5-round integral distinguisher of Midori64, we can get 3 more balanced bits than the $\mathcal{S}$ method by setting 12 active bits. For 6-round integral distinguisher, we can also get 3 more balanced bits than the $\mathcal{S}$ method by setting 45 active bits. In addition, our 7-round distinguisher is consistent with that of the $\mathcal{ZR}$ method as in [11].

4.2 Application to Skinny64

By applying our new technique to Skinny64, we get a 10-round integral distinguisher by setting 60 active bits. However, as shown in [11], when using the $\mathcal{S}$

method, only 9-round integral distinguisher can be obtained by setting 63 active bits. Our 9- and 10-round integral distinguishers are shown as follows.

$$\begin{bmatrix} \mathcal{C} & \mathcal{A} & \mathcal{A} & \mathcal{A} \\ \mathcal{A} & \mathcal{A} & \mathcal{A} & \mathcal{A} \\ \mathcal{A} & \mathcal{A} & \mathcal{A} & \mathcal{A} \\ \mathcal{A} & \mathcal{A} & \mathcal{A} & \mathcal{A} \end{bmatrix} \xrightarrow{9R} \begin{bmatrix} \mathcal{B} & \mathcal{B} & \mathcal{B} & \mathcal{B} \\ \mathcal{B} & \mathcal{B} & \mathcal{B} & \mathcal{B} \\ \mathcal{B} & \mathcal{B} & \mathcal{B} & \mathcal{B} \\ \mathcal{B} & \mathcal{B} & \mathcal{B} & \mathcal{B} \end{bmatrix} \qquad \begin{bmatrix} \mathcal{C} & \mathcal{A} & \mathcal{A} & \mathcal{A} \\ \mathcal{A} & \mathcal{A} & \mathcal{A} & \mathcal{A} \\ \mathcal{A} & \mathcal{A} & \mathcal{A} & \mathcal{A} \\ \mathcal{A} & \mathcal{A} & \mathcal{A} & \mathcal{A} \end{bmatrix} \xrightarrow{10R} \begin{bmatrix} \mathcal{B} & \mathcal{B} & \mathcal{B} & \mathcal{B} \\ \mathcal{B} & \mathcal{B} & \mathcal{B} & \mathcal{B} \\ \mathcal{B} & \mathcal{B} & \mathcal{B} & \mathcal{B} \\ \mathcal{B} & \mathcal{B} & \mathcal{B} & \mathcal{B} \end{bmatrix}$$

The experimental results show that our method inputs 56 active bits and obtains 64 balanced bits for the 9-round integral distinguisher, where fewer active bits are needed compared with the distinguishers obtained by $\mathcal{S}$ method. For 10-round integral distinguisher, there are 60 active input bits and 64 balanced output bits, which is consistent with the result by $\mathcal{ZR}$ method in [11].

4.3 Application to LED

In [9], Sun *et al.* can only search for 6-round integral distinguisher of LED. But using our method, 7-round integral distinguisher can be got which are consistent with the result by $\mathcal{HW}$ method in [12] and are expressed as follows.

$$\begin{bmatrix} \mathcal{A} & aaac & \mathcal{A} & \mathcal{A} \\ \mathcal{A} & \mathcal{A} & \mathcal{A} & \mathcal{A} \\ \mathcal{A} & \mathcal{A} & \mathcal{A} & \mathcal{A} \\ \mathcal{A} & \mathcal{A} & \mathcal{A} & \mathcal{A} \end{bmatrix} \xrightarrow{7R} \begin{bmatrix} \mathcal{B} & \mathcal{B} & \mathcal{B} & \mathcal{B} \\ \mathcal{B} & \mathcal{B} & \mathcal{B} & \mathcal{B} \\ \mathcal{B} & \mathcal{B} & \mathcal{B} & \mathcal{B} \\ \mathcal{B} & \mathcal{B} & \mathcal{B} & \mathcal{B} \end{bmatrix}$$

The experimental results show that our method can find one more round integral distinguisher than the $\mathcal{S}$ method, and when setting 63 active input bits, the distinguisher is same as the result obtained by the $\mathcal{HW}$ method.

5 Conclusion

In this paper, we propose a new technique to improve the accuracy of modeling BDP propagations of complex linear layers, whose core idea is to combine the optimized implementations of matrices and the modeling rules for COPY and XOR (**CX** rules). In particular, we use three heuristics (**Paar**, **BP** and **XZ**) to obtain optimized implementations of a matrix and then model these implementations based on the **CX** rules simultaneously to generate a system of inequalities. Moreover, we theoretically prove that this new method is always superior to the $\mathcal{S}$ method, which straightforwardly ultilizes the **CX** rules to model the direct implementation of a matrix. In order to exhibit the effect of this new method, we apply it to several block ciphers. As a result, we can obtain longer distinguishers or more balanced bits for Midori64, Skinny64 and LED than that of the $\mathcal{S}$ method. Additionally, what we need emphasize is that our method is not completely accurate compared with the $\mathcal{ZR}$ and $\mathcal{HW}$ methods. However, our results are consistent with that of the $\mathcal{ZR}$ and $\mathcal{HW}$ methods. Furthermore, our method

can be implemented based on MILP as well as SMT/SAT, and the constraints are fully linear, thus simple. In consequence, for modeling the BDP propagation of linear layers based on very large and complex matrices, our method may be more practical among these exsiting methods.

Acknowledgements

We would like to thank the anonymous reviewers for their helpful comments and suggestions. This work was supported by the Application Foundation Frontier Project of Wuhan Science and Technology Bureau (NO. 2020010601012189), the National Natural Science Foundation of China (NO. 61802119) and the Research Foundation of Department of Education of Hubei Province, China (No. D2020104).

References

1. Biham E., Shamir A.: Differential cryptanalysis of DES-like cryptosystems. In: Menezes A., Vanstone S. (eds.) CRYPTO 1990. LNCS, vol. 537, pp. 2–21. Springer, Heidelberg (1990). https://doi.org/10.1007/3-540-38424-3_1
2. Matsui M.: Linear cryptanalysis method for DES cipher. In: Helleseth T. (eds.) EUROCRYPT 1993. LNCS, vol. 765, pp. 386–397. Springer, Heidelberg (1993). https://doi.org/10.1007/3-540-48285-7_33
3. Knudsen L., David A.: Integral cryptanalysis. In: Daemen J., Rijmen V. (eds.) FSE 2002. LNCS, vol. 2365, pp. 112–127. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-45661-9_9
4. Daemen J., Knudsen L., Rijmen V.: The block cipher Square. In: Daemen J., Rijmen V. (eds.) FSE 1997. LNCS, vol. 1267, pp. 149–165. Springer, Heidelberg (1997). https://doi.org/10.1007/BFb0052343
5. Todo Y.: Structural evaluation by generalized integral property. In: Oswald E., Fischlin M. (eds.) EUROCRYPT 2015. LNCS, vol. 9056, pp. 287–314. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46800-5_12
6. Todo Y.: Integral cryptanalysis on full MISTY1. In: Gennaro R., Robshaw M. (eds.) CRYPTO 2015. LNCS, vol. 9215, pp. 413–432. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-47989-6_20
7. Todo Y., Morii M.: Bit-based division property and application to SIMON family. In: Peyrin T. (eds.) FSE 2016. LNCS, vol. 9783, pp. 357–377. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-52993-5_18
8. Xiang Z J., Zhang W T., Bao Z Z., Lin D D.: Applying MILP method to searching integral distinguishers based on division property for 6 lightweight block ciphers. In: Cheon J., Takagi T. (eds.) ASIACRYPT 2016. LNCS, vol. 10031, pp. 648–678. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53887-6_24
9. Sun L., Wang W., Wang M Q.: MILP-aided bit-based division property for primitives with non-bit-permutation linear layers. In: IET Inf. Secur. 2020. vol. 14, pp. 12–20 (2020). https://doi.org/10.1049/iet-ifs.2018.5283
10. Sun L., Wang W., Liu R., Wang M Q.: Milp-aided bit-based division property for ARX ciphers. In: Sci. China Inf. Sci 2018, vol. 61, pp. 118102:1–118102:3 (2018). https://doi.org/10.1007/s11432-017-9321-7

11. Zhang W Y., Rijmen V.: Division cryptanalysis of block ciphers with a binary diffusion layer. In: IET Inf. Secur. 2019. vol. 13, pp. 87–95 (2019). https://doi.org/10.1049/iet-ifs.2018.5151
12. Hu K., Wang Q J., Wang M Q.: Finding bit-based division property for ciphers with complex linear layers. In: IACR Trans. Symmetric Cryptol. 2020. vol. 2020, pp. 396–424 (2020). https://doi.org/10.13154/tosc.v2020.i1.396-424
13. Elsheikh M., Youssef A.: On MILP-based Automatic Search for Bit-Based Division Property for Ciphers with (large) Linear Layers. (Submitted to ACISP 2021). https://eprint.iacr.org/2021/643
14. Paar C.: Optimized arithmetic for reed-solomon encoders. In: Proceedings of IEEE International Symposium on Information Theory 1997. pp. 250 (1997). https://doi.org/10.1109/ISIT.1997.613165
15. Boyar J., Matthews P., Peralta R.: Logic minimization techniques with applications to cryptology. J. Cryptology 2013, vol. 26, pp. 280–312 (2013). https://doi.org/10.1007/s00145-012-9124-7
16. Xiang Z J., Zeng X Y., Lin D., Bao Z Z., Zhang S S.: Optimizing implementations of linear layers. In: IACR Trans. Symmetric Cryptol. 2020, vol. 2020, pp.120–145 (2020). https://doi.org/10.13154/tosc.v2020.i2.120-145
17. Banik S., Bogdanov A., Isobe T., Shibutani K., Toru A., Francesco R.: Midori: A block cipher for low energy. In: ASIACRYPT. 2015. LNCS, vol. 9453, pp. 411–436. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-48800-3_17
18. Beierle C., Jean J., Stefan K., Leander G., Sim S.: The SKINNY family of block ciphers and its low-latency variant MANTIS. In: CRYPTO. 2016. LNCS, vol. 9815, pp. 123–153. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53008-5_5
19. Guo J., Peyrin T., Poschmann A., Matthew J.: The LED block cipher. In: Preneel B., Takagi T. (eds.) CHES 2011. LNCS, vol. 6917, pp. 326–341. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-23951-9_22
20. Mouha N., Wang Q J., Gu D W., Preneel B.: Differential and linear cryptanalysis using mixed-integer linear programming. In: Wu C., Yung M., Lin D. (eds.) Inscrypt 2011. LNCS, vol. 7537, pp. 57–76. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-34704-7_5
21. Sasaki Y., Todo Y.: New impossible differential search tool from design and cryptanalysis aspects. In: Jean b., Nielsen J. (eds.) EUROCRYPT 2017. LNCS, vol. 10212, pp. 185–215. Springer, Heidelberg (2017). https://doi.org/10.1007/978-3-319-56617-7_7
22. Cui T., Chen S., Fu K., Wang M., Jia K.: New automatic tool for finding impossible differentials and zero-correlation linear approximations. Sci. China Inf. Sci 2021. vol. 64, pp. 2 (2021). https://doi.org/10.1007/s11432-018-1506-4
23. Sun S., Hu L., Wang P., Qiao K., Ma X., Song L.: Automatic security evaluation of block ciphers with s-bp structures against related-key differential attacks. In: Palash S., Iwata T. (eds.) ASIACRYPT 2014. LNCS, vol. 8873, pp. 158–178. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-45611-8_9
24. Kranz T., Leander G., Stoffelen K., Wiemer F.: Shorter linear straight-line programs for MDS matrices. In: IACR Trans. Symmetric Cryptol. vol. 2017, pp. 188–211 (2017). https://doi.org/10.13154/tosc.v2017.i4.188-211
25. Khoo K., Peyrin T., Poschmann A., Yap H.: FOAM: searching for hardware-optimal SPN structures and components with a fair comparison. In: Batina L. , Robshaw M. (eds.) CHES 2014. LNCS, vol. 8731, pp. 433–450. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-44709-3_24