MILP-Aided Bit-Based Division Property for Primitives with Non-Bit-Permutation Linear Layers

Ling Sun[1], Wei Wang[1], Meiqin Wang*[1,2]

[1] Key Laboratory of Cryptologic Technology and Information Security, Ministry of Education, Shandong University, Jinan, 250100, China
[2] State Key Laboratory of Cryptology, P.O. Box 5159, Beijing, 100878, China
lingsun@mail.sdu.edu.cn; weiwangsdu@sdu.edu.cn; mqwang@sdu.edu.cn

Abstract. Division property is a general integral property introduced by Todo at EUROCRYPT 2015. Recently, at ASIACRYPT 2016, Xiang *et al.* applied the Mixed Integer Linear Programming (MILP) method to search bit-based division property, and handled the complexity which restricted the application of bit-based division property proposed by Todo and Morii at FSE 2016. However, their MILP-aided search was only applied to some lightweight block ciphers whose linear layers were limited to bit-permutations, and the feasibility of MILP-aided bit-based division property for ciphers with non-bit-permutation linear layers was an open problem. This paper comes out with the affirmative answer.

First, we transform the complicated linear layers to their primitive representations, which only involves **Copy** and `XOR` operations. Then, the original **Copy** and `XOR` models are respectively generalized to deal with more output branches and input elements, and these generalized models are adopted to depict the primitive representations. Accordingly, the MILP-aided bit-based division property can be applied to much more primitives with complicated linear layers. As an illustration, we first evaluate the bit-based division propertyies of some word-oriented block ciphers including `Midori64`, LED, `Joltik-BC`, and AES. For `Midori64`, we obtain a 7-round integral distinguisher, which achieves one more round than the previous results. At the same time, the data requirements of some existing distinguishers are also reduced. We decrease the number of required chosen plaintexts of 4-round and 5-round integral distinguishers for LED and `Joltik-BC` by half. As to AES, our searching experiments show that integral distinguishers, which are based on the bit-based division property, covering more than four rounds probably do not exist. Then, the bit-based division properties of some bit-oriented block ciphers, such as Serpent and Noekeon, are considered. The data complexities of their distinguishers for short rounds are improved. Moreover, we evaluate the bit-based division properties of the internal permutations involved in some hash functions, *e.g.*, SPONGENT and PHOTON. An 18-round zero-sum distinguisher for SPONGENT-88 is proposed, which achieves four more rounds than the previous ones. We also provide 20-round and 21-round

* Corresponding Author

zero-sum distinguishers for SPONGENT-128 and SPONGENT-160, respectively. For most PHOTON permutations P_t with 4-bit cell, the data requirements for the 4-round distinguishers are reduced by half. Besides, the length of P_{256}'s distinguisher is extended by one round. Furthermore, for P_{288} using 8-bit S-boxes, we improve the data complexities of their integral distinguishers significantly.
Keywords: Integral distinguisher, bit-based division property, MILP, `Midori`, LED, `Joltik-BC`, AES, Serpent, Noekeon, SPONGENT, PHOTON

1 Introduction

The integral cryptanalysis was first introduced as a dedicate attack for the word-oriented block cipher SQUARE by Daemen *et al.* [8] at FSE 1997. Theoretically, integral attacks can be applied to bit-oriented block ciphers. However, till FSE 2008 [31], Z'aba *et al.* firstly gave an explicit tool to find integral distinguishers for bit-oriented block ciphers and the bit-pattern based integral attack was successfully demonstrated on reduced-round variants of the block ciphers Noekeon [9], PRESENT [5], and Serpent [3].

At EUROCRYPT 2015, Todo [26] generalized the integral property to division property, which can precisely depict the implicit properties between traditional ALL and BALANCE properties. By applying the division property, the integral distinguisher can be constructed even if block ciphers have non-bijective functions, bit-oriented structures, and low-degree functions. But he only made use of the algebraic degree of the S-box to trace its division property propagation so that longer distinguisher may be detected for a specific cipher. At CRYPTO 2015, Todo [25] showed that division property could be more useful if an S-box was supposed to be a public function. He detected a 6-round integral distinguisher for MISTY1 [18] by utilizing the vulnerable property of S_7, and achieved the first attack against full MISTY1.

At FSE 2016, Todo and Morii [27] proposed the bit-based division property and explored the 14-round integral distinguisher for SIMON32 [2]. They pointed that the time and memory complexities for the bit-based division property were roughly 2^n for an n-bit block cipher. On the one hand, the huge time and memory complexities restricted the application of bit-based division property. On the other hand, whether the bit-based division property could be adopted to analyse other bit-oriented block ciphers was unknown.

Many further researches focusing on these interesting issues have occurred in succession. At CRYPTO 2016, by introducing the notion of parity sets, Boura and Canteaut [6] presented a new approach to deal with division property. For PRESENT, they provided some low-data integral distinguishers. By replacing the **Substitution** rule, which managed the propagation of S-box, with a more subtle propagation table, Sun and Wang [22] worked out the table-aided bit-based division property, and successfully applied it to some bit-oriented primitives such as RECTANGLE [32] and SPONGENT-88 [4]. Thus, the bit-based

division property can be compatible with ciphers other than SIMON. At ASIACRYPT 2016, Xiang *et al.* [29] applied Mixed Integer Linear Programming (MILP) method to search integral distinguisher based on division property[3], and found some longer integral distinguishers for SIMON family, Simeck family [30], PRESENT, RECTANGLE, LBlock [28], and TWINE [24]. Their work handled the problem about the complexity and showed that bit-based division property could be efficiently applied to some ciphers whose block sizes are larger than 32. However, the linear layers for all these analytical ciphers are restricted to only simple bit-permutations. Thus, the feasibility of MILP method to analyze ciphers with linear layers besides bit-permutations was not settled [29].

Table 1: Comparison of Main Results for Some Block Ciphers with Previous Results.

Cipher	**$\log_2$(#texts)**				**Reference**
	$r=4$	$r=5$	$r=6$	$r=7$	
`Midori64`	**4**	**12**	**45**	**61**	**Section 4.1**
	28	52	60	-	[26]
`Midori128`	**8**	**32**	**92**	**124**	**Section 4.1**
	28	84	113	124	[26]
Serpent†	**23**	**83**	**113**	**124**	**Section 4.2**
	28	84	113	124	[26]
Noekeon	**27**	**83**	**113**	**124**	**Section 4.2**
	28	84	113	124	[26]

$\log_2$(#texts): The exponent of the number of required chosen plaintexts.

† Since Serpent uses different S-boxes, which have distinct properties, in different rounds, the starting round may influence the resulting distinguisher. Here, we refer to the case where the initial round is the first round.

Our Contributions. In this paper, we settle the open problem, and improve some integral distinguishers for various primitives by MILP-aided bit-based division property. The contributions of this paper are summarized as follows.

1. **Construct new searching models for complicated linear layers not limited to bit-permutations.** First, we transform the complicated linear layers to the primitive representations. Then, the original **Copy** and `XOR` models are respectively generalized to deal with more output branches and

[3] We name it *MILP-aided bit-based division property* in this paper.

input elements, and these generalized models are adopted to depict the primitive representations. In this way, we can model all kinds of linear layers only if we have their primitive representations, and we will find that getting the primitive representation of a linear layer is an easy task. Thus, the MILP-aided bit-based division property can be applied to much more primitives with relatively complicated linear layers.

2. **Apply MILP-aided bit-based division property to word-oriented block ciphers, including `Midori64` [1], LED [14], `Joltik-BC` [16], and AES [10].** For `Midori64`, obtain a 7-round integral distinguisher, which gains one more round than the previous analysis. Moreover, the data complexity is reduced significantly for r-round distinguisher where $r \leqslant 6$. As to LED and `Joltik-BC`, the data requirements for 4-round and 5-round distinguishers are decreased by half. As to AES, our searching experiments show that integral distinguishers, which are based on the bit-based division property, covering more than four rounds probably do not exist.
3. **Consider the bit-based division properties of some bit-oriented block ciphers, such as Serpent [3] and Noekeon [9].** Due to their relatively complicated linear layers and large block sizes, it is difficult to perform integral cryptanalysis. At FSE 2008, Z'aba *et al.* [31] proposed 3.5-round integral distinguishers for Noekeon and Serpent. Todo [26] improved it by traditional division property. Applying the new method, we also reduce the data complexities of some short-round distinguishers for these two ciphers.
4. **Evaluate the bit-based division properties of the internal permutations involved in some hash functions, *e.g.*, SPONGENT [4] and PHOTON [13].** The published results all focused on SPONGENT-88, that is, the 14-round zero-sum distinguishers proposed by Dong *et al.* [11], Fan *et al.* [12], and Sun and Wang [22], respectively. The best one we obtained is an 18-round zero-sum distinguisher with complexity 2^{87}, which gains four more rounds than the previous ones. Moreover, we provide 20-round and 21-round zero-sum distinguishers for SPONGENT-128 and SPONGENT-160, respectively. For PHOTON permutations with 4-bit cell, the data complexities for the 4-round distinguishers are reduced by half. Besides, we obtain a 9-round distinguisher for P_{256}, which gains one more round than the previous ones. Furthermore, for P_{288} using 8-bit S-boxes, the data complexities of the distinguishers are dramatically improved.

The comparisons of the main results with previous results for some block ciphers and internal permutations of hash functions are shown in **Table 1** and **Table 2**, respectively.

Outline of the Paper. The rest of this paper is organized as follows. In **Section 2**, we briefly review some notations and definitions such as division property, bit-based division property, table-aided bit-based division property, and MILP-aided bit-based division property. **Section 3** illustrates how to apply MILP-aided bit-based division property to ciphers with more complicated linear layers. **Section 4** gives some applications of MILP-aided bit-based division

Table 2: Comparison of Main Results for Some Internal Permutations of Hash Functions with Previous Results.

Cipher	Round	$\log_2$(#texts)	Reference
SPONGENT-88	**18**	**87**	**Section 4.3**
	17	**85**	
	16	**84**	
	15	**80**	
	14	84	[11]
	14	80	[12, 22]
SPONGENT-128	**20**	**126**	**Section 4.3**
SPONGENT-160	**21**	**159**	**Section 4.3**
P_{288} in PHOTON	**3**	**8**	**Section 4.3**
	3	253	[7]
	4	**48**	**Section 4.3**
	4	283	[7]

$\log_2$(#texts): The exponent of the number of required chosen plaintexts.

property. We conclude the paper in **Section 5**. Some auxiliary materials are supplied in **Supplementary Materials** following the paper.

2 Preliminary

2.1 Notations

In this subsection, we present the notations used throughout this paper. In order to simplify the representation, a bit-string will be written in hexadecimal format and is always written in italic `verbatim` font. We follow the notations defined in [26] and [25].

For an n-bit string $a \in \mathbb{F}_2^n$, the i-th element is expressed as $a[i]$, where the bit positions are labeled in big-endian, and the Hamming weight $wt(a)$ is calculated by $wt(a) = \sum_{i=0}^{n-1} a[i]$.

For any set $\mathbb{K}$, $|\mathbb{K}|$ denotes the number of elements in $\mathbb{K}$. Let $\emptyset$ be an empty set.

For any $\boldsymbol{a} = (a_0, a_1, \ldots, a_{m-1}) \in \mathbb{F}_2^{\ell_0} \times \mathbb{F}_2^{\ell_1} \times \cdots \times \mathbb{F}_2^{\ell_{m-1}}$, the vectorial Hamming weight of $\boldsymbol{a}$ is defined as $Wt(\boldsymbol{a}) = (wt(a_0), wt(a_1), \ldots, wt(a_{m-1})) \in \mathbb{Z}^m$. For any $\boldsymbol{k} \in \mathbb{Z}^m$ and $\boldsymbol{k'} \in \mathbb{Z}^m$, we define $\boldsymbol{k} \succeq \boldsymbol{k'}$ if $k_i \geq k'_i$ for all i. Otherwise, $\boldsymbol{k} \not\succeq \boldsymbol{k'}$. $\mathbb{K} \leftarrow \boldsymbol{k}$ means that $\mathbb{K}$ turns into $\mathbb{K} \bigcup \{\boldsymbol{k}\}$.

Definition 1 (Bit Product Function [26]). *Assume $u \in \mathbb{F}_2^n$ and $x \in \mathbb{F}_2^n$. The Bit Product Function π_u is defined as*

$$\pi_u(x) = \prod_{i=0}^{n-1} x[i]^{u[i]}.$$

For $\boldsymbol{u} = (u_0, u_1, \ldots, u_{m-1}) \in \mathbb{F}_2^{\ell_0} \times \mathbb{F}_2^{\ell_1} \times \cdots \times \mathbb{F}_2^{\ell_{m-1}}$, let $\boldsymbol{x} = (x_0, x_1, \ldots, x_{m-1}) \in \mathbb{F}_2^{\ell_0} \times \mathbb{F}_2^{\ell_1} \times \cdots \times \mathbb{F}_2^{\ell_{m-1}}$ be the input, the Bit Product Function $\pi_{\boldsymbol{u}}$ is defined as

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

The bit product function also appears in the Algebraic Normal Form (ANF) of a Boolean function. The ANF of a Boolean function $f : \mathbb{F}_2^n \to \mathbb{F}_2$ is represented as

$$f(x) = \bigoplus_{u \in \mathbb{F}_2^n} a_u^f \left(\prod_{i=1}^{n} x[i]^{u[i]} \right) = \bigoplus_{u \in \mathbb{F}_2^n} a_u^f \pi_u(x),$$

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

2.2 Division Property and Bit-Based Division Property

Traditional integral distinguisher is usually constructed by evaluating the propagation of integral property such as ALL and BALANCE properties. Division property, which was firstly proposed in [26], is a generalization of integral property. It can precisely depict the implicit properties between ALL and BALANCE properties, which makes division property an efficient tool to construct integral distinguisher. Bit-based division property [27] handles a special case of division property, where the space under consideration is restricted to the direct product of a series of binary fields. Comparing to traditional division property, bit-based division property traces the division property at the bit-level and showed its power by finding longer integral distinguisher for SIMON32. In this subsection, we will briefly review division property and bit-based division property, and list some propagation rules of bit-based division property.

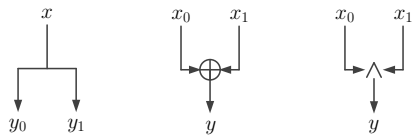
Definition 2 (Division Property [26]). *Let $\mathbb{X}$ be a multi-set whose elements take values from $\mathbb{F}_2^{\ell_0} \times \mathbb{F}_2^{\ell_1} \times \cdots \times \mathbb{F}_2^{\ell_{m-1}}$. When the multi-set $\mathbb{X}$ has the division property $\mathcal{D}_{\mathbb{K}}^{\ell_0,\ell_1,\ldots,\ell_{m-1}}$, where $\mathbb{K}$ denotes a set of m-dimensional vectors whose i-th element takes a value between 0 and ℓ_i, it fulfills the following conditions:*

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

Remark 1. If there are $\boldsymbol{k} \in \mathbb{K}$ and $\boldsymbol{k}' \in \mathbb{K}$ satisfying $\boldsymbol{k} \succeq \boldsymbol{k}'$ in the division property $\mathcal{D}_{\mathbb{K}}^{\ell_0,\ell_1,\ldots,\ell_{m-1}}$, $\boldsymbol{k}$ can be removed from $\mathbb{K}$ because it is redundant.

Remark 2. Note that ℓ_0, ℓ_1, …, ℓ_{m-1} are restricted to 1 when we consider **bit-based division property**.

Propagation Rules of Bit-Based Division Property Todo [26] proved some propagation rules for conventional division property and these rules were summarized into five rules in [25], which were **Substitution**, **Copy**, `XOR`, **Split**, and **Concatenation**, respectively. Among the five rules, only **Copy** and `XOR` are necessary for bit-based division property. The two necessary rules are restated in a bit-based look in the following.



(a) **Copy**. (b) `XOR`. (c) `AND`.

Fig. 1: Illustrations of Basic Operations.

Rule 1 (Copy) *Let F be a **Copy** function, where the input x takes a value of $\mathbb{F}_2$ and the output is calculated as $(y_0, y_1) = (x, x)$ (See **Figure 1(a)**). Let $\mathbb{X}$ and $\mathbb{Y}$ be the input multi-set and output multi-set, respectively. Assuming that the multi-set $\mathbb{X}$ has the division property $\mathcal{D}^1_{\{k\}}$, the division property of the multi-set $\mathbb{Y}$ is $\mathcal{D}^{1\times 1}_{\mathbb{K}'}$. There are only two possible cases for the propagation:*

$$\begin{cases} \mathbb{K}' = \{(0,0)\}, & \text{if } k = 0 \\ \mathbb{K}' = \{(0,1),(1,0)\}, & \text{if } k = 1 \end{cases}$$

Rule 2 (`XOR`) *Let F be a function composed of **XOR** operation, where the input (x_0, x_1) takes a value of $\mathbb{F}_2 \times \mathbb{F}_2$ and the output is calculated as $y = x_0 \oplus x_1$ (See **Figure 1(b)**). Let $\mathbb{X}$ and $\mathbb{Y}$ be the input multi-set and output multi-set, respectively. Assuming that the multi-set $\mathbb{X}$ has division property $\mathcal{D}^{1\times 1}_{\{\boldsymbol{k}\}}$, the division property of the multi-set $\mathbb{Y}$ is $\mathcal{D}^1_{\mathbb{K}'}$. There are only three possible cases for the propagation:*

$$\begin{cases} \mathbb{K}' = \{(0)\}, & \text{if } \boldsymbol{k} = (0,0) \\ \mathbb{K}' = \{(1)\}, & \text{if } \boldsymbol{k} = (0,1) \text{ or } (1,0) \\ \mathbb{K}' = \emptyset, & \text{if } \boldsymbol{k} = (1,1) \end{cases}$$

For some bit-oriented block ciphers such as SIMON, `AND` is another non-linear operation. The propagation for `AND` is given in [27] and we summarize it as follows.

Rule 3 (`AND`) *Let F be a function composed of **AND** operation, where the input (x_0, x_1) takes a value of $\mathbb{F}_2 \times \mathbb{F}_2$ and the output is calculated as $y = x_0 \wedge x_1$*

*(See **Figure 1(c)**). Let $\mathbb{X}$ and $\mathbb{Y}$ be the input multi-set and output multi-set, respectively. Assuming that the multi-set $\mathbb{X}$ has division property $\mathcal{D}^{1\times 1}_{\{\boldsymbol{k}\}}$, the division property of the multi-set $\mathbb{Y}$ is $\mathcal{D}^{1}_{\mathbb{K}'}$. There are only two possible cases for the propagation:*

$$\begin{cases} \mathbb{K}' = \{(0)\}, & \text{if } \boldsymbol{k} = (0,0) \\ \mathbb{K}' = \{(1)\}, & \text{otherwise} \end{cases}$$

Propagating the Bit-Based Division Property of S-box By integrating the ANF of the S-box, Sun and Wang [22] provided an idea to propagate the bit-based division property of S-box[4].

Let $\boldsymbol{x} = (x_0, x_1, \ldots, x_{b-1})$ and $\boldsymbol{y} = (y_0, y_1, \ldots, y_{b-1})$ be the input and output of a b-bit S-box. Suppose that the input multi-set $\mathbb{X}$ follows the division property $\mathcal{D}^{1^b}_{\{\boldsymbol{k}\}}$, which implies that $\bigoplus_{\boldsymbol{x}\in\mathbb{X}} \pi_{\boldsymbol{j}}(\boldsymbol{x})$ is unknown for any $\boldsymbol{j} \in \mathbb{F}_2^b$ with $\boldsymbol{j} \succeq \boldsymbol{k}$.

To determine the division property $\mathcal{D}^{1^b}_{\mathbb{K}}$ of the output multi-set $\mathbb{Y}$, the ANF of y_i $(0 \leqslant i \leqslant b-1)$ should be taken into consideration. For any b-bit string $\boldsymbol{k}'$, to judge whether the parity of $\pi_{\boldsymbol{k}'}(\boldsymbol{y})$ is always even or not, we should check the ANF of $\pi_{\boldsymbol{k}'}(\boldsymbol{y})$. Assume that

$$\pi_{\boldsymbol{k}'}(\boldsymbol{y}) = \prod_{i=0}^{b-1} \pi_{k'_i}(y_i) = \bigoplus_{\boldsymbol{u}\in\mathbb{F}_2^b} a_{\boldsymbol{u}} \pi_{\boldsymbol{u}}(\boldsymbol{x}),$$

where $a_{\boldsymbol{u}} \in \mathbb{F}_2$ is a constant value depending on $\pi_{\boldsymbol{k}'}(\boldsymbol{y})$ and $\boldsymbol{u}$. If there exists $\boldsymbol{j} \in \mathbb{F}_2^b$ satisfying $\boldsymbol{j} \succeq \boldsymbol{k}$ such that $a_{\boldsymbol{j}} = 1$, then the parity of $\pi_{\boldsymbol{k}'}(\boldsymbol{y})$ is unknown since the value of $\bigoplus_{x\in\mathbb{X}} \pi_{\boldsymbol{j}}(\boldsymbol{x})$ is unknown. Otherwise, the parity of $\pi_{\boldsymbol{k}'}(\boldsymbol{y})$ is always even. After inserting all the vectors $\boldsymbol{k}'$ such that $\pi_{\boldsymbol{k}'}(\boldsymbol{y})$ becomes unknown into $\mathbb{K}$, note that some of the vectors in $\mathbb{K}$ are redundant. Then we get the final $\mathbb{K}$ after removing the redundant vectors in $\mathbb{K}$.

For any $\boldsymbol{k} \in \mathbb{F}_2^b$, we can deduce its corresponding $\mathbb{K}$ according to the above procedure. In this way, a propagation table for S-box can be constructed. The propagation table has two columns, the first column is filled with $\boldsymbol{k}$ while the second column is filled with $\mathbb{K}$ corresponding to $\boldsymbol{k}$. Then, the propagation of bit-based division property of S-box becomes a simple table look-up. This method is named as table-aided bit-based division property in [22].

2.3 MILP-Aided Bit-Based Division Property

Although bit-based division property is proved to be a powerful tool to find integral distinguishers, the time and memory complexities of utilizing this method are roughly 2^n for an n-bit block cipher. Due to this restriction, searching integral distinguishers for some primitives whose sizes are larger than 32 bits is almost

[4] At CRYPTO 2016, Boura and Canteaut [6] proposed a method to propagate bit-based division property through S-box by computing the propagation of the parity set. And the underlying idea is same to the one provided in [22]. Please refer to [6] for more information.

impossible. At ASIACRYPT 2016, Xiang *et al.* [29] proposed the method of describing the bit-based division property with the MILP model. With the help of some openly available MILP optimizers such as Gurobi[5], the complexities of employing bit-based division property can dramatically decrease and the workload of designers and cryptanalysts is significantly reduced. In this subsection, we will give a brief review of MILP-aided bit-based division property.

The main idea of MILP-aided bit-based division property is modelling those propagation rules of bit-based division property with a series of linear inequalities[6].

Modelling Copy, `AND`, `XOR`, and S-box Corresponding to **Rule 1** to **Rule 3**, the following models are proposed to describe three basic bit-wise operations with linear inequalities.

Model 1 (Copy [29]) *Denote* $(a) \xrightarrow{Copy} (b_0, b_1)$ *a division trail of* ***Copy*** *function, the following inequalities are sufficient to describe the division propagation of* ***Copy****.*

$$\begin{cases} a - b_0 - b_1 = 0 \\ a, b_0, b_1 \text{ are binaries} \end{cases}$$

Model 2 (`AND` [29]) *Denote* $(a_0, a_1) \xrightarrow{AND} (b)$ *a division trail of* ***AND*** *function, the following linear inequalities are sufficient to describe the division propagation of* ***AND****.*

$$\begin{cases} b - a_0 \geqslant 0 \\ b - a_1 \geqslant 0 \\ b - a_0 - a_1 \leqslant 0 \\ a_0, a_1, b \text{ are binaries} \end{cases}$$

Model 3 (`XOR` [29]) *Denote* $(a_0, a_1) \xrightarrow{XOR} (b)$ *a division trail through* ***XOR*** *function, the following inequalities can describe the division trail through* ***XOR*** *function.*

$$\begin{cases} a_0 + a_1 - b = 0 \\ a_0, a_1, b \text{ are binaries} \end{cases}$$

Modelling S-box To deduce the linear inequality system of S-box, we firstly use table-aided bit-based division property to generate the propagation table of the S-box.[7] After that, by invoking the `inequality_generator()` function in the Sage[8] software, a set of linear inequalities will be returned. Sometimes, the

[5] http://www.gurobi.com/

[6] We do not distinguish linear equality and linear inequality in this paper, since MILP model can include linear inequality as well as linear equality.

[7] Another method to generate the propagation table of the S-box was introduced in [29]. Both of these two methods consider the ANF of the S-box. Although the starting points of them are different, the resulting propagations are exactly the same.

[8] http://www.sagemath.org/

number of linear inequalities in the set is very large such that adding all these inequalities into the MILP model will make the problem computational infeasible. Thus, Sun *et al.* [23] proposed an algorithm called **Greedy Algorithm** (**Algorithm 1** in [29]) to reduce this set. Since the **Greedy Algorithm** is not deterministic, the linear inequality systems of various S-boxes provided in this paper are not unique.

Up to now, for block ciphers based on the three operations and (or) S-box, we are able to construct a set of linear inequalities characterizing one round division property propagation. Iterating this process r times, we can get a linear inequality system $\mathcal{L}$ describing r rounds division property propagation. All feasible solutions of $\mathcal{L}$ correspond to all r-round division trails, which are defined below.

Definition 3 (Division Trail [29]). *Let f_r denote the round function of an iterated block cipher. Assume that the input multi-set of the block cipher has initial division property $\mathcal{D}^{1^n}_{\{\boldsymbol{k}\}}$, and denote the division property after i-round propagation through f_r by $\mathcal{D}^{1^n}_{\mathbb{K}_i}$. Thus we have the following chain of division property propagations:*

$$\{\boldsymbol{k}\} \triangleq \mathbb{K}_0 \xrightarrow{f_r} \mathbb{K}_1 \xrightarrow{f_r} \mathbb{K}_2 \xrightarrow{f_r} \cdots .$$

Moreover, for any vector $\boldsymbol{k}_i^* \in \mathbb{K}_i$ $(i \geqslant 1)$, 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 $(\boldsymbol{k}_0, \boldsymbol{k}_1, \ldots, \boldsymbol{k}_r) \in \mathbb{K}_0 \times \mathbb{K}_1 \times \cdots \times \mathbb{K}_r$, if $\boldsymbol{k}_{i-1}$ can propagate to $\boldsymbol{k}_i$ for all $i \in \{1, 2, \ldots, r\}$, we call $(\boldsymbol{k}_0, \boldsymbol{k}_1, \ldots, \boldsymbol{k}_r)$ an r-round division trail.

Initial Division Property and Stopping Rule Denote $(a_0^0, a_1^0, \ldots, a_{n-1}^0) \to \cdots \to (a_0^r, a_1^r, \ldots, a_{n-1}^r)$ an r-round division trail, $\mathcal{L}$ is a linear inequality system defined on variables a_i^j $(i = 0, 1, \cdots, n-1, j = 0, 1, \cdots, r)$ and some auxiliary variables. Let $\mathcal{D}^{1^n}_{\{\boldsymbol{k}\}}$ denote the initial input division property with $\boldsymbol{k} = (k_0, k_1, \ldots, k_{n-1})$, we need to add $a_i^0 = k_i$ $(i = 0, 1, \ldots, n-1)$ into $\mathcal{L}$, and all feasible solutions of $\mathcal{L}$ are division trails which start from vector $\boldsymbol{k}$.

By applying the definition of division property, the existence of any vector with Hamming weight larger than two indicates that all bits of the state satisfy zero-sum property. And the existence of a unit vector tells that the bit located at the position of the unique non-zero element does not follow zero-sum property. Thus, the objective function is set as

$$Obj : Min\{a_0^r + a_1^r + \cdots + a_{n-1}^r\}.$$

Let $\mathcal{D}^{1^n}_{\mathbb{K}_i}$ denote the output division property after i rounds of encryption and the input division property is denoted by $\mathcal{D}^{1^n}_{\mathbb{K}_0}$. If $\mathbb{K}_{r+1}$ contains all the n unit vectors for the first time, the division property propagation should stop and an r-round distinguisher can be derived from $\mathcal{D}^{1^n}_{\mathbb{K}_r}$.

Note that we only recall some key points here. For more details, please refer to [6, 22, 25–27, 29].

3 MILP-Aided Bit-Based Division Property for Primitives with Non-Bit-Permutation Linear Layers

Even though MILP-aided bit-based division property illustrated by Xiang *et al.* handles the huge complexities of bit-based division property, the primitives with non-bit-permutation linear layers are not considered. And the feasibility of MILP method applying to ciphers with more complicated linear layers was left as an open problem. To settle this problem, the key point is to transform the complex linear layer to an equivalent representation with only **Copy** and `XOR` operations, and to generalize the original **Copy** and `XOR` models to handle it. The invocations of the generalized models introduce some intermediate variables, which are reorganized according to the equivalent representation, and the linear inequality system for the division property propagation of linear layer is obtained. Finally, MILP-aided bit-based division property becomes more powerful and can be applied to more primitives with relatively complicated linear layers.

3.1 Generalizing Copy and `XOR` Models

Note that we have many different ways to define a linear transformation. However, we always can represent the linear transformation as a matrix over $\mathbb{F}_2$. We call this kind of representation the *primitive representation* as in [21], and always denote $M_*^{\mathcal{PR}}$ the primitive representation of a linear transformation. How to obtain the primitive representation of a linear transformation can be found in **Supplementary Material A**.

Claim. No matter how complicated the linear layer is, it can always be split into **Copy** and `XOR` operations according to the primitive representation.

Example 1 (An Intuitive Example). Suppose that the primitive representation of a toy linear layer is

$$M_{toy}^{\mathcal{PR}} = \begin{pmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 0 \end{pmatrix}.$$

Let $\boldsymbol{y}^T = M_{toy}^{\mathcal{PR}} \cdot \boldsymbol{x}^T$, where $\boldsymbol{x}^T$ represents the transpose of $\boldsymbol{x} = (x_0, x_1, x_2)$, *i.e.*,

$$\begin{cases} y_0 = x_0 \oplus x_1 \oplus x_2 \\ y_1 = x_1 \oplus x_2 \\ y_2 = x_0 \oplus x_1 \end{cases}.$$

From **Figure 2**, it is obvious that, for x_1, the number of output branches of **Copy** operation is 3 and the number of input elements of `XOR` operation for y_0 is 3, which are larger than the requirements of **Model 1** and **3** in **Section 2.3**. And these models need to be generalized in order to work for more complicated linear layers.

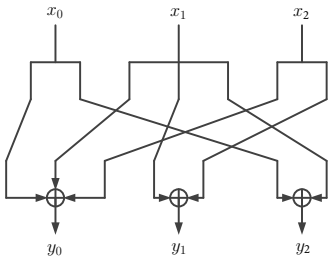


Fig. 2: An Illustration of the Toy Example.

Model 4 (Generalized Copy) *Denote* $(a) \xrightarrow{\textbf{\textit{Copy}}} (b_0, b_1, \ldots, b_m)$ *a division trail of* ***Copy*** *function, the following inequalities are sufficient to describe the division propagation of* ***Copy****.*

$$\begin{cases} a - b_0 - b_1 - \cdots - b_m = 0 \\ a, b_0, b_1, \ldots, b_m \text{ are binaries} \end{cases}$$

Model 5 (Generalized XOR) *Denote* $(a_0, a_1, \ldots, a_m) \xrightarrow{\texttt{XOR}} (b)$ *a division trail through* `XOR` *function, the following inequalities can describe the division trail through* `XOR` *function.*

$$\begin{cases} a_0 + a_1 + \cdots + a_m - b = 0 \\ a_0, a_1, \ldots, a_m, b \text{ are binaries} \end{cases}$$

With **Model 4** and **Model 5**, we can depict the division property propagation of any linear layer by introducing some intermediate variables according to the primitive representation of the linear layer.

3.2 Modelling the Primitive Representation of A Linear Layer

Let $M^{\mathcal{PR}}$ be an $n \times n$ matrix, which is a primitive representation of a (or a part of) linear layer, and denote

$$M^{\mathcal{PR}} = \begin{pmatrix} m_{0,0} & m_{0,1} & \cdots & m_{0,n-1} \\ m_{1,0} & m_{1,1} & \cdots & m_{1,n-1} \\ \vdots & \vdots & \ddots & \vdots \\ m_{n-1,0} & m_{n-1,1} & \cdots & m_{n-1,n-1} \end{pmatrix},$$

where $m_{i,j} \in \{0, 1\}$. We suppose that the Hamming weight of the i-th column of $M^{\mathcal{PR}}$ is c_i, and the Hamming weight of the j-th row is r_j. Let $c_M = \sum_{i=0}^{n-1} c_i = \sum_{j=0}^{n-1} r_j$ be the number of non-zero elements in $M^{\mathcal{PR}}$.

Inspired from the former toy example, we know that the i-th input bit of $M^{\mathcal{PR}}$ need to be copied c_i times. Thus, c_i intermediate variables need to be

introduced to represent the division properties of these copies. To propagate the division properties for all input bits, c_M intermediate variables $t_0 \sim t_{c_M-1}$ are required in total.

Suppose that the input multi-set of $M^{\mathcal{PR}}$ satisfies division property $\mathcal{D}^{1^n}_{\{\boldsymbol{x}\}}$, where $\boldsymbol{x} = (x_0, x_1, \ldots, x_{n-1})$, and the output multi-set follows $\mathcal{D}^{1^n}_{\{\boldsymbol{y}\}}$, where $\boldsymbol{y} = (y_0, y_1, \ldots, y_{n-1})$. We may allocate the first c_0 intermediate variables to x_0, and allocate the next c_1 variables to x_1, and so forth. Then, by utilizing **Model 4**, we can list the linear inequalities to describe the **Copy** operations for all input bits as follows.

$$\begin{cases} x_0 - t_0 - t_1 - \cdots - t_{c_0-1} = 0 \\ x_1 - t_{c_0} - t_{c_0+1} - \cdots - t_{c_0+c_1-1} = 0 \\ \cdots\cdots \\ x_{n-1} - t_{c_M-c_{n-1}} - t_{c_M-c_{n-1}+1} - \cdots - t_{c_M-1} = 0 \\ x_0, x_1, \ldots, x_{n-1}, t_0, t_1, \ldots, t_{c_M-1} \text{ are binaries} \end{cases} \tag{1}$$

To propagate the `XOR` operations, those intermediate variables should be allocated according to the arrangement of non-zero elements in $M^{\mathcal{PR}}$. For example, since $t_0 \sim t_{c_0-1}$ are assigned to depict the division properties of the output copies for the first input bit, they are put in those positions of the first column's non-zero elements in order. Let $I^{(i)} = \{I_0^{(i)}, I_1^{(i)}, \ldots, I_{r_i-1}^{(i)}\}$ be the index set of the i-th row, whose elements are the indexes of intermediate variables in the i-th row. According to **Model 5**, the linear inequalities to describe the `XOR` operations for all output bits are obtained, that is,

$$\begin{cases} t_{I_0^{(0)}} + t_{I_1^{(0)}} + \cdots + t_{I_{r_0-1}^{(0)}} - y_0 = 0 \\ t_{I_0^{(1)}} + t_{I_1^{(1)}} + \cdots + t_{I_{r_1-1}^{(1)}} - y_1 = 0 \\ \cdots\cdots \\ t_{I_0^{(n-1)}} + t_{I_1^{(n-1)}} + \cdots + t_{I_{r_{n-1}-1}^{(n-1)}} - y_{n-1} = 0 \\ y_0, y_1, \ldots, y_{n-1}, t_0, t_1, \ldots, t_{c_M-1} \text{ are binaries} \end{cases}. \tag{2}$$

Combining (1) and (2), we construct the linear inequality system used to trace the division property propagation of the linear operation $M^{\mathcal{PR}}$.

3.3 Application to the `MixColumns` of LED

In order to illustrate the above model, we take the `MixColumns` operation of LED [14] as an example. `MixColumns` is a part of linear operations for LED's round function, and it works like the `MixColumn` operation for AES [20]. It multiplies each column of the internal state by the same 4×4 MDS matrix M_{LED} over the field $\mathbb{F}_2^4$, where

$$M_{\text{LED}} = \begin{pmatrix} \texttt{4} & \texttt{2} & \texttt{1} & \texttt{1} \\ \texttt{8} & \texttt{6} & \texttt{5} & \texttt{6} \\ \texttt{b} & \texttt{e} & \texttt{a} & \texttt{9} \\ \texttt{2} & \texttt{2} & \texttt{f} & \texttt{b} \end{pmatrix}.$$

And the underlying polynomial for the field multiplication is $x^4 + x + 1$. We transform M_{LED} into its primitive representation as follows.

$$M_{\mathrm{LED}}^{\mathcal{PR}} = \left(\begin{array}{cccc|cccc|cccc|cccc} 0&0&1&0&0&1&0&0&1&0&0&0&1&0&0&0\\ 1&0&0&1&0&0&1&0&0&1&0&0&0&1&0&0\\ 1&1&0&0&1&0&0&1&0&0&1&0&0&0&1&0\\ 0&1&0&0&1&0&0&0&0&0&0&1&0&0&0&1\\ \hline 1&0&0&1&0&1&1&0&1&0&1&0&0&1&1&0\\ 1&1&0&0&1&0&1&1&1&1&0&1&1&0&1&1\\ 0&1&1&0&0&1&0&1&1&1&1&0&0&1&0&1\\ 0&0&1&0&1&1&0&0&0&1&0&1&1&1&0&0\\ \hline 0&1&0&1&1&1&1&1&1&1&0&1&0&0&0&1\\ 1&0&1&0&0&1&1&1&1&1&1&0&1&0&0&0\\ 1&1&0&1&0&0&1&1&1&1&1&1&0&1&0&0\\ 1&0&1&1&1&1&1&0&1&0&1&0&0&0&1&1\\ \hline 0&1&0&0&0&1&0&0&0&1&1&1&0&1&0&1\\ 0&0&1&0&0&0&1&0&0&0&1&1&1&0&1&0\\ 1&0&0&1&1&0&0&1&0&0&0&1&1&1&0&1\\ 1&0&0&0&1&0&0&0&1&1&1&1&1&0&1&1 \end{array}\right). \tag{3}$$

Note that there are four columns in the internal state of LED, and M_{LED} operates on each column independently. Thus, we just list the linear inequalities for one column in the following.

Suppose that the input multi-set of $M_{\mathrm{LED}}^{\mathcal{PR}}$ has division property $\mathcal{D}^{1^{16}}_{\{\boldsymbol{x}\}}$, where $\boldsymbol{x} = (x_0, x_1, \ldots, x_{15})$, and the output multi-set follows division property $\mathcal{D}^{1^{16}}_{\{\boldsymbol{y}\}}$, where $\boldsymbol{y} = (y_0, y_1, \ldots, y_{15})$. There are 123 non-zero elements in $M_{\mathrm{LED}}^{\mathcal{PR}}$. So that 123 intermediate variables ($t_0 \sim t_{122}$) are introduced and are arranged according to the positions of '1's in $M_{\mathrm{LED}}^{\mathcal{PR}}$ in the following way.

$$\left(\begin{array}{cccc|cccc|cccc|cccc} 0&0&t_{16}&0&0&t_{36}&0&0&t_{59}&0&0&0&t_{95}&0&0&0\\ t_0&0&0&t_{22}&0&0&t_{44}&0&0&t_{68}&0&0&0&t_{102}&0&0\\ t_1&t_9&0&0&t_{28}&0&0&t_{52}&0&0&t_{77}&0&0&0&t_{109}&0\\ 0&t_{10}&0&0&t_{29}&0&0&0&0&0&0&t_{86}&0&0&0&t_{115}\\ \hline t_2&0&0&t_{23}&0&t_{37}&t_{45}&0&t_{60}&0&t_{78}&0&0&t_{103}&t_{110}&0\\ t_3&t_{11}&0&0&t_{30}&0&t_{46}&t_{53}&t_{61}&t_{69}&0&t_{87}&t_{96}&0&t_{111}&t_{116}\\ 0&t_{12}&t_{17}&0&0&t_{38}&0&t_{54}&t_{62}&t_{70}&t_{79}&0&0&t_{104}&0&t_{117}\\ 0&0&t_{18}&0&t_{31}&t_{39}&0&0&0&t_{71}&0&t_{88}&t_{97}&t_{105}&0&0\\ \hline 0&t_{13}&0&t_{24}&t_{32}&t_{40}&t_{47}&t_{55}&t_{63}&t_{72}&0&t_{89}&0&0&0&t_{118}\\ t_4&0&t_{19}&0&0&t_{41}&t_{48}&t_{56}&t_{64}&t_{73}&t_{80}&0&t_{98}&0&0&0\\ t_5&t_{14}&0&t_{25}&0&0&t_{49}&t_{57}&t_{65}&t_{74}&t_{81}&t_{90}&0&t_{106}&0&0\\ t_6&0&t_{20}&t_{26}&t_{33}&t_{42}&t_{50}&0&t_{66}&0&t_{82}&0&0&0&t_{112}&t_{119}\\ \hline 0&t_{15}&0&0&0&t_{43}&0&0&0&t_{75}&t_{83}&t_{91}&0&t_{107}&0&t_{120}\\ 0&0&t_{21}&0&0&0&t_{51}&0&0&0&t_{84}&t_{92}&t_{99}&0&t_{113}&0\\ t_7&0&0&t_{27}&t_{34}&0&0&t_{58}&0&0&0&t_{93}&t_{100}&t_{108}&0&t_{121}\\ t_8&0&0&0&t_{35}&0&0&0&t_{67}&t_{76}&t_{85}&t_{94}&t_{101}&0&t_{114}&t_{122} \end{array}\right). \tag{4}$$

On the one hand, the variables located in the same column are exactly the variables used to describe the **Copy** operation for the corresponding input bit. Thus,

the linear inequality system (5) is sufficient to describe the **Copy** operations. On the other hand, the variables located in the same row are involved in the `XOR` operation of the corresponding output bit. So, the propagations of the `XOR` operations turn into linear inequality system (6). Thereby, we just need to combine linear inequality systems (5) and (6) as a whole linear inequality system, and can trace the propagation of division property for $M_{\mathrm{LED}}^{\mathcal{PR}}$.

$$\begin{cases} x_0 - t_0 - t_1 - t_2 - t_3 - t_4 - t_5 - t_6 - t_7 - t_8 = 0 \\ x_1 - t_9 - t_{10} - t_{11} - t_{12} - t_{13} - t_{14} - t_{15} = 0 \\ x_2 - t_{16} - t_{17} - t_{18} - t_{19} - t_{20} - t_{21} = 0 \\ x_3 - t_{22} - t_{23} - t_{24} - t_{25} - t_{26} - t_{27} = 0 \\ x_4 - t_{28} - t_{29} - t_{30} - t_{31} - t_{32} - t_{33} - t_{34} - t_{35} = 0 \\ x_5 - t_{36} - t_{37} - t_{38} - t_{39} - t_{40} - t_{41} - t_{42} - t_{43} = 0 \\ x_6 - t_{44} - t_{45} - t_{46} - t_{47} - t_{48} - t_{49} - t_{50} - t_{51} = 0 \\ x_7 - t_{52} - t_{53} - t_{54} - t_{55} - t_{56} - t_{57} - t_{58} = 0 \\ x_8 - t_{59} - t_{60} - t_{61} - t_{62} - t_{63} - t_{64} - t_{65} - t_{66} - t_{67} = 0 \\ x_9 - t_{68} - t_{69} - t_{70} - t_{71} - t_{72} - t_{73} - t_{74} - t_{75} - t_{76} = 0 \\ x_{10} - t_{77} - t_{78} - t_{79} - t_{80} - t_{81} - t_{82} - t_{83} - t_{84} - t_{85} = 0 \\ x_{11} - t_{86} - t_{87} - t_{88} - t_{89} - t_{90} - t_{91} - t_{92} - t_{93} - t_{94} = 0 \\ x_{12} - t_{95} - t_{96} - t_{97} - t_{98} - t_{99} - t_{100} - t_{101} = 0 \\ x_{13} - t_{102} - t_{103} - t_{104} - t_{105} - t_{106} - t_{107} - t_{108} = 0 \\ x_{14} - t_{109} - t_{110} - t_{111} - t_{112} - t_{113} - t_{114} = 0 \\ x_{15} - t_{115} - t_{116} - t_{117} - t_{118} - t_{119} - t_{120} - t_{121} - t_{122} = 0 \\ x_0, x_1, \ldots, x_{15}, t_0, t_1, \ldots, t_{122} \text{ are binaries} \end{cases} \tag{5}$$

3.4 Sketch of MILP-Aided Bit-Based Division Property for Primitives with Complicated Linear Layers

In the remaining of this section, we give an overview of applying the MILP-aided bit-based division property to primitives with complicated linear layers. All the analyses of primitives provided in **Section 4** follow the procedures given below.

1. **Generating Linear Inequality System for S-box**
   (a) We deduce the propagation table of the S-box.
   (b) All elements in the propagation table are put into `inequality_generator()` to generate the linear inequalities used to describe the S-box.
   (c) **Greedy Algorithm** is invoked to simplify the above linear inequality system.
2. **Generating Linear Inequality System for Linear Layer**
   (a) The linear layer is transformed into the primitive representation.
   (b) The intermediate variables are introduced and arranged according to the non-zero elements in the primitive representation of the linear layer, and the linear inequality system is obtained.
3. **Constructing Linear Inequality System for r Rounds Division Property Propagation**

(a) The linear inequality system used to propagate r rounds division property is constructed by combining the above two linear inequality systems following the structure of the specific cipher.

4. **Searching Integral Distinguishers with Different Initial Division Properties**
   (a) To obtain various distinguishers, we can change the initial division property of the MILP model.

$$
\begin{cases}
t_{16} + t_{36} + t_{59} + t_{95} - y_0 = 0 \\
t_0 + t_{22} + t_{44} + t_{68} + t_{102} - y_1 = 0 \\
t_1 + t_9 + t_{28} + t_{52} + t_{77} + t_{109} - y_2 = 0 \\
t_{10} + t_{29} + t_{86} + t_{115} - y_3 = 0 \\
t_2 + t_{23} + t_{37} + t_{45} + t_{60} + t_{78} + t_{103} + t_{110} - y_4 = 0 \\
t_3 + t_{11} + t_{30} + t_{46} + t_{53} + t_{61} + t_{69} + t_{87} + t_{96} + t_{111} + t_{116} - y_5 = 0 \\
t_{12} + t_{17} + t_{38} + t_{54} + t_{62} + t_{70} + t_{79} + t_{104} + t_{117} - y_6 = 0 \\
t_{18} + t_{31} + t_{39} + t_{71} + t_{88} + t_{97} + t_{105} - y_7 = 0 \\
t_{13} + t_{24} + t_{32} + t_{40} + t_{47} + t_{55} + t_{63} + t_{72} + t_{89} + t_{118} - y_8 = 0 \\
t_4 + t_{19} + t_{41} + t_{48} + t_{56} + t_{64} + t_{73} + t_{80} + t_{98} - y_9 = 0 \\
t_5 + t_{14} + t_{25} + t_{49} + t_{57} + t_{65} + t_{74} + t_{81} + t_{90} + t_{106} - y_{10} = 0 \\
t_6 + t_{20} + t_{26} + t_{33} + t_{42} + t_{50} + t_{66} + t_{82} + t_{112} + t_{119} - y_{11} = 0 \\
t_{15} + t_{43} + t_{75} + t_{83} + t_{91} + t_{107} + t_{120} - y_{12} = 0 \\
t_{21} + t_{51} + t_{84} + t_{92} + t_{99} + t_{113} - y_{13} = 0 \\
t_7 + t_{27} + t_{34} + t_{58} + t_{93} + t_{100} + t_{108} + t_{121} - y_{14} = 0 \\
t_8 + t_{35} + t_{67} + t_{76} + t_{85} + t_{94} + t_{101} + t_{114} + t_{122} - y_{15} = 0 \\
y_0, y_1, \ldots, y_{15}, t_0, t_1, \ldots, t_{122} \text{ are binaries}
\end{cases} \tag{6}
$$

4 Applications of MILP-Aided Bit-Based Division Property

In this section, we show some applications of MILP-aided bit-based division property. Firstly, we present the applications of MILP-aided bit-based division property to some word-oriented block ciphers, such as `Midori64`, LED, `Joltik-BC`, and AES. Then we evaluate some bit-oriented block ciphers, including Serpent and Noekeon. At last, the bit-based division properties of the internal permutations used in some hash functions are concerned.

4.1 Applications to Word-Oriented Block Ciphers

Application to `Midori64`

A Brief Introduction of `Midori64` [1] `Midori64` is a block cipher with 64-bit block and 128-bit key. The 64-bit state S is arranged in a 4×4 matrix of 4-bit

cells:

$$S = \begin{bmatrix} s_0 & s_4 & s_8 & s_{12} \\ s_1 & s_5 & s_9 & s_{13} \\ s_2 & s_6 & s_{10} & s_{14} \\ s_3 & s_7 & s_{11} & s_{15} \end{bmatrix}.$$

Our searching algorithm related to the data processing part $\texttt{MidoriCore}_{(16)}$, which is a 16-round SP-network, and each round takes the following four operations and the final round omits `ShuffleCell` and `MixColumn` operations. Before the first round, there is a key whitening operation. For more details, please refer to [1].

- `SubCell`: A 4-bit S-box, shown in **Table 3**, is applied to every cell of the state S.
- `ShuffleCell`: Each cell of the state is permuted as follows:

  $(s_0, s_1, \ldots, s_{15}) \leftarrow (s_0, s_{10}, s_5, s_{15}, s_{14}, s_4, s_{11}, s_1, s_9, s_3, s_{12}, s_6, s_7, s_{13}, s_2, s_8)$
- `MixColumn`: Multiplying each column by a 4×4 matrix $M_{\texttt{Midori64}}$ over $\mathbb{F}_2^4$, where

$$M_{\texttt{Midori64}} = \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix}.$$

- `KeyAdd`: The i-th 64-bit round key RK_i is `XOR`ed to the state S.

Table 3: `Midori64`'s S-Box $S_{\texttt{Midori64}}$ [1]

x	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
$S[x]$	c	a	d	3	e	b	f	7	8	9	1	5	0	2	4	6

Since `KeyAdd` operation does not affect the propagation of division property, we do not consider it in our analysis.

Applying MILP-Aided Bit-Based Division Property to `Midori64`

- **Generating Linear Inequality System for S-box:** The propagation table for $S_{\texttt{Midori64}}$ has 48 vectors, and 54 linear inequalities are returned by invoking `inequality_generator()`. After using Greedy Algorithm, only 5 linear inequalities are left, which are provided in **Supplementary Material B**.
- **Generating Linear Inequality System for `MixColumns` Operation:** There are 48 non-zero elements in the primitive representation of $M_{\texttt{Midori64}}$. Thus, $48 \times 4 = 192$ intermediate variables $t_0 \sim t_{191}$ are required for one round of encryption since there are four columns in the state.

Experimental Results of `Midori64` We put different initial division properties into the MILP model and a 7-round integral distinguisher is obtained, which gains one more round than the previous cryptanalysis. Besides, the data complexity is reduced significantly for r-round distinguisher where $r \leqslant 6$. Our results are in accordance with the expectation of the designers that the length of the integral distinguisher is bounded by 7. The concrete number of chosen plaintexts to construct r-round distinguisher are listed in **Table 1**, and the integral distinguishers for `Midori64` are presented in **Supplementary Material B**.

Applications to Some AES-like Block Ciphers We also test some AES-like block ciphers, *e.g.*, LED [14], `Joltik-BC` [16], and AES [20]. For LED and `Joltik-BC`, the data requirements for 4-round and 5-round distinguishers are reduced by half. As to AES, even the initial division property $\boldsymbol{k}$ = `[ffffffff,ffffffff,ffffffff,fffffffe]`, of which the Hamming weight is 127, is put into AES's MILP model, however, there is no bit satisfying zero-sum property after five-round encryption. Our experimental results indicate that integral distinguishers built upon bit-based division property, covering more than four rounds probably do not exist. The comparison and the numbers of chosen plaintexts to construct r-round integral distinguishers for these ciphers are given in **Table 4**.

Comparing to the dedicated attack to search integral distinguishers for AES-like ciphers in [26], we propagate the S-box at the bit-level and consider the concrete form of the linear layer. Thus, the bit-based division properties for specific ciphers should be better than or at least equal to the more general search in [26]. From **Table 4**, the data requirements of r-round distinguishers for LED and `Joltik-BC` are improved where $3 < r < 6$, which is in accordance with the former claim. But we do not provide the data requirements of 6-round distinguishers, since we observe that Gurobi cannot work efficiently when the Hamming weight of the initial division property is neither too large nor too small.

For example, since Todo [26] proposed that the numbers of chosen plaintexts to construct 6-round distinguishers for LED and `Joltik-BC`, which adopt $(4, 3, 4)$-AES structure, are 2^{52}, we tested an initial division property

$$\boldsymbol{k} = \texttt{[ffff,ff0f,fff0,0fff]}$$

with Hamming weight being 52 for the new MILP models of LED and `Joltik-BC`, and tried to trace their bit-based division properties after six rounds of encryption. But Gurobi gradually ran out of memory and no solution was exported.

We think the reason behind this circumstance is that the number of possibilities needed to be searched by Gurobi exceeds its tolerance. An intuitive example is the propagation tables of the S-boxes. We take the propagation table of LED's S-box as an illustration, which can be found in [29][9]. Namely, the output division property of $\boldsymbol{k}$ = `[0]` is $\mathbb{K} = \{\texttt{[0]}\}$, and the output division

[9] Note that LED takes PRESENT's S-box as its S-box.

property of $\boldsymbol{k}$ = [f] is $\mathbb{K} = \{[\mathtt{f}]\}$. The numbers of vectors in the output division property do not expand. But the number of vectors in the output division property increases sharply when $0 < wt(\boldsymbol{k}) < 4$. Although the propagation of bit-based division property is transformed into an MILP problem, Gurobi still needs to consider all possible division trails according to the propagation rules, which are just turned into a series of linear inequalities directly. Besides, **Algorithm 4** of [26] invoked a function `SizeReduce()` to eliminate the redundant vectors, while MILP model does not cover this procedure, which leads to the increasement in the number of possibilities managed by Gurobi. This is just our conjecture, and we should further study the solving mechanism of Gurobi to verify it. But we expect that we can get at least the same results as those provided by Todo if enough resources are available.

Table 4: Comparison of the Numbers of Chosen Plaintexts to Construct r-Round Integral Distinguishers for Some AES-Like Block Ciphers.

Cipher	**$\log_2$(#texts)**				**Reference**
	$r=3$	$r=4$	$r=5$	$r=6$	
LED & `Joltik-BC`	**4**	**11**	**31**	-	**Section 4.1**
	4	12	32	52	[26]
	12	28	52	60	[26]
	28	52	60	63	[7]
	4	16	-	-	[8, 17]
AES	**8**	**32**	-	-	**Section 4.1**
	56	120	-	-	[26]
	117	127	-	-	[7]
	8	32	-	-	[8, 17]

$\log_2$(#texts): The exponent of the number of required chosen plaintexts.

The integral distinguishers corresponding to the results in **Table 4** can be found in **Supplementary Material C**.

4.2 Applications to Bit-Oriented Block Ciphers

Application to Serpent

A Brief Introduction of Serpent [3] Serpent is a block cipher which was one of the five finalists for Advanced Encryption Standard [19]. It is a 32-round SPN structure operating on four 32-bit words (X_0, X_1, X_2, and X_3), thus giving a block size of 128 bits. Its round function consists of alternating layers of key

mixing, S-boxes, and linear transformation. Serpent has eight S-boxes ($S_0 \sim S_7$) and the set of eight S-boxes is used four times. Each round function uses a single S-box 32 times in parallel. The first round uses S_0 and the second round uses S_1. After using S_7 in the eighth round, S_0 is used again in the ninth round. The bit-wise linear transformation of Serpent is omitted for space limitation. Please refer to [3] for more information.

Applying MILP-Aided Bit-Based Division Property to Serpent

- **Generating Linear Inequality Systems for Eight S-boxes:** For space limitation, we do not give the propagation tables and the linear inequality systems for the eight S-boxes of Serpent.
- **Generating Linear Inequality System for Linear Transformation:** For the linear layer, we treat it as a large 128×128 matrix and there are 610 non-zero elements in the primitive representation of Serpent's linear layer. Thus 610 intermediate variables $t_0 \sim t_{609}$ are needed for one round of encryption.

Experimental Results for Serpent Since different rounds use different S-boxes, the starting round may influence the length of the resulting integral distinguisher. After analyzing all possible cases, we find that the data requirements are different for different initial rounds, and the experimental results are shown in **Table 5**. Comparing to the results given by Todo [26], we improve the data complexities of some distinguishers for shorter rounds ($r < 6$). For $r \geqslant 6$, the data requirements are same to the previous results. The explicit forms of these distinguishers can be found in **Supplementary Material D**.

Table 5: Comparison of Data Requirements for Serpent with Different Initial Rounds.

Initial Round	**$\log_2$(#texts)**				**Reference**
	$r=4$	$r=5$	$r=6$	$r=7$	
0	**23**	**83**	**113**	**124**	
1, 2, and 6	**24**	**83**	**113**	**124**	**Section 4.2**
3, 4, 5, and 7	**24**	**84**	**113**	**124**	
all	28	84	113	124	[26]

$\log_2$(#texts): The exponent of the number of required chosen plaintexts.

Application to Noekeon The bit-based division property for Noekeon [9] is also considered. Note that Noekeon follows PSP structure, actually. Since **Step 2** of **Algorithm 2** in Todo's work [26] deals with division property propagation

of non-linear layer firstly, we conjecture that Todo transformed Noekeon into the SPN structure, so that we also do the transformations in order to compare to the results of [26]. The experimental results can be found in **Table 1**, and the concrete forms of the integral distinguishers are given in **Supplementary Material E**.

4.3 Applications to Other Primitives

Application to SPONGENT

A Brief Introduction of SPONGENT [4] SPONGENT is a family of lightweight hash functions with different hash sizes and similar round functions. There are five variants of SPONGENT, and we only analyze SPONGENT-88, SPONGENT-128, and SPONGENT-160 with hash size 88, 128, and 160, respectively. SPONGENT uses SP-network and utilizes a PRESENT-type permutation which iterates 45, 70, and 90 times for the former mentioned three variants. The non-linear layer uses a 4-bit S-box (S_{SPONTENT}) in parallel. An illustration of SPONGENT-88's round function is depicted in **Figure 3**. For more details about SPONGENT, please refer to [4].

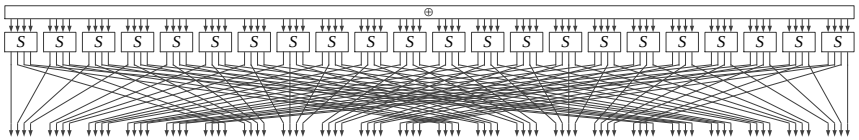


Fig. 3: Round Function of SPONGENT-88.

Applying MILP-Aided Bit-Based Division Property to SPONGENT

– **Generating Linear Inequality Systems for S-boxes:** Since we need to find zero-sum distinguishers in opposite directions, the propagation tables for S_{SPONGENT} and S^{-1}_{SPONGENT} are required. When we need to find the zero-sum distinguishers in the forward direction, we use S_{SPONGENT}'s propagation table. If we turn to search the zero-sum distinguishers in the backward direction, we apply the propagation table of S^{-1}_{SPONGENT}. There are 48 vectors in the propagation table of S_{SPONGENT}. 197 inequalities are returned by calling `inequality_generator()`. After utilizing **Greedy Algorithm**, we obtain 10 inequalities. For S^{-1}_{SPONGENT}, 48 elements return 180 inequalities. After applying **Greedy Algorithm**, 11 inequalities are left. The linear inequality systems for S_{SPONGENT} and S^{-1}_{SPONGENT} are provided in **Supplementary Material F**.

– **Generating Linear Inequality System for Linear Layer:** Since the linear layer of SPONGENT-88 (SPONGENT-128, SPONGENT-160) is simple bit-permutation, we do not need to introduce intermediate variables and $88 \times 2 = 176$ ($128 \times 2 = 256$, $160 \times 2 = 320$) variables (including the variables representing the division property for the input bits of the next round) are enough to constitute the linear inequality system for one round.

Experimental Results for SPONGENT For SPONGENT-88, we find four zero-sum distinguishers, and the general information is listed in **Table 2**. One of them is a 15-round zero-sum distinguisher with data complexity 2^{80}. Comparing to the former results, this newly obtained distinguisher achieves one more round than the one proposed by Fan and Duan [12] while keeps the same complexity. The best one is an 18-round zero-sum distinguisher with data complexity 2^{87}, which gains four more rounds than the previous ones. Besides, we also give some results for SPONGENT-128 and SPONGENT-160. The concrete information for these distinguishers can be obtained in **Supplementary Material F**.

Applications to PHOTON Permutations We also analyze the internal permutation P_t of PHOTON [13], which is a family of hash functions, where $t \in \{100, 144, 196, 256, 288\}$. All P_t's adopt AES-like structure, and the cell sizes of the first four variants are all 4-bit while the last one has 8-bit cell size. The experimental results for different variants can be found in **Table 6**. Just as what we have mentioned before, Gurobi cannot work efficiently when the Hamming weight of the initial division property is neither too large nor too small. Besides, the block sizes of these variants are relatively large. Thus, many programs gradually ran out of memory and no solution is exported, and the results we obtained are those listed in **Table 6**.

From the observation of **Table 6**, the advantages can be summarized into three points. Firstly, we improve the data complexities of 4-round integral distinguishers for variants with 4-bit cell size, and the data complexity of 7-round distinguisher for P_{144} is reduced by half. Secondly, we significantly reduce the data requirements of distinguishers for P_{288}, whose cell size is 8-bit. Besides, we obtain a 9-round distinguisher for P_{256} whose initial division property has a large Hamming weight, and extend the length of the integral distinguisher for P_{256} by one round. The explicit forms of these newly obtained distinguishers can be found in **Supplementary Material G**.

5 Conclusion

In this paper, we answer the open question proposed by Xiang *et al.* at ASIACRYPT 2016, and construct new models to illustrate that the MILP technique is applicable to primitives with non-bit-permutation linear layers.

The key point is to transform the complicated linear layers to the primitive representations, and generalize the original **Copy** and `XOR` models to depict the primitive representations. Accordingly, the MILP-aided bit-based division

Table 6: Comparison of the Numbers of Chosen Plaintexts to Construct r-Round Integral Distinguishers for Internal Permutations of PHOTON.

Cipher	**$\log_2$(#texts)**							**Reference**
	$r=$ **3**	$r=$ **4**	$r=$ **5**	$r=$ **6**	$r=$ **7**	$r=$ **8**	$r=$ **9**	
P_{100} in PHOTON	**4**	**11**	**20**	-	-	-	-	**Section 4.3**
	4	12	20	72	97	-	-	[26]
	12	28	76	92	-	-	-	[26]
	28	76	92	98	-	-	-	[7]
	4	20	-	-	-	-	-	[8, 17]
P_{144} in PHOTON	**4**	**11**	**24**	-	**131**	-	-	**Section 4.3**
	4	12	24	84	132	-	-	[26]
	12	28	84	124	140	-	-	[26]
	28	82	124	138	142	-	-	[7]
	4	24	-	-	-	-	-	[8, 17]
P_{196} in PHOTON	**4**	**11**	-	-	-	**192**	-	**Section 4.3**
	4	12	24	84	164	192	-	[26]
	12	28	84	160	184	192	-	[26]
	28	82	158	184	192	195	-	[7]
	4	28	-	-	-	-	-	[8, 17]
P_{256} in PHOTON	**4**	**11**	**28**	-	-	-	**252**	**Section 4.3**
	4	12	28	92	204	249	-	[26]
	12	28	84	200	237	252	-	[26]
	28	82	198	237	250	254	-	[7]
	4	32	-	-	-	-	-	[8, 17]
P_{288} in PHOTON	**8**	**48**	-	-	-	-	-	**Section 4.3**
	253	283	-	-	-	-	-	[7]

$\log_2$(#texts): The exponent of the number of required chosen plaintexts.

property can be performed. We adopt MILP-aided bit-based division property to detect integral distinguishers for some word-oriented block ciphers, such as `Midori64`, LED, `Joltik-BC`, and AES. For `Midori64`, we significantly improve the data requirements of the previous results and extend the length of integral distinguisher by one round. As to LED and `Joltik-BC`, we reduce the numbers of chosen plaintexts of 4-round and 5-round distinguishers by half. Although we do not discover any distinguisher covering more than four rounds for AES, we confirm that there is no integral distinguisher based on bit-based division property achieving five rounds. Then, some bit-oriented block ciphers, including

Serpent and Noekeon, are considered. For both of them, the data complexities of some short-round distinguishers are decreased. Furthermore, the bit-based division properties of the internal permutations employed in some hash functions are evaluated, too. An 18-round zero-sum distinguisher for SPONGENT-88 is obtained, which achieves four more rounds than the previous results. For all PHOTON permutation with 4-bit cell, the data requirements for 4-round distinguishers are reduced by half. Besides, the length of P_{256}'s distinguisher is extended by one round. Furthermore, we dramatically decrease the data complexities of distinguishers for P_{288}, which is a variant of PHOTON permutation with 8-bit S-box.

References

1. S. Banik, A. Bogdanov, T. Isobe, K. Shibutani, H. Hiwatari, T. Akishita, and F. 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.
2. R. Beaulieu, D. Shors, J. Smith, S. Treatman-Clark, B. Weeks, and L. Wingers. The SIMON and SPECK lightweight block ciphers. In *Proceedings of the 52nd Annual Design Automation Conference, San Francisco, CA, USA, June 7-11, 2015*, pages 175:1–175:6, 2015.
3. E. Biham, R. J. Anderson, and L. R. Knudsen. Serpent: A new block cipher proposal. In *Fast Software Encryption, 5th International Workshop, FSE '98, Paris, France, March 23-25, 1998, Proceedings*, pages 222–238, 1998.
4. A. Bogdanov, M. Knezevic, G. Leander, D. Toz, K. Varici, and I. Verbauwhede. SPONGENT: the design space of lightweight cryptographic hashing. *IEEE Trans. Computers*, 62(10):2041–2053, 2013.
5. A. Bogdanov, L. R. Knudsen, G. Leander, C. Paar, A. Poschmann, M. J. B. Robshaw, Y. 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.
6. C. Boura and A. Canteaut. Another view of the division property. In *Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part I*, pages 654–682, 2016.
7. C. Boura, A. Canteaut, and C. D. Cannière. Higher-order differential properties of Keccak and *Luffa*. In *Fast Software Encryption - 18th International Workshop, FSE 2011, Lyngby, Denmark, February 13-16, 2011, Revised Selected Papers*, pages 252–269, 2011.
8. J. Daemen, L. R. Knudsen, and V. Rijmen. The block cipher Square. In *Fast Software Encryption, 4th International Workshop, FSE '97, Haifa, Israel, January 20-22, 1997, Proceedings*, pages 149–165, 1997.
9. J. Daemen, M. Peeters, G. Van Assche, and V. Rijmen. Nessie proposal: Noekeon. In *First Open NESSIE Workshop*, pages 213–230, 2000.
10. J. Daemen and V. Rijmen. *The Design of Rijndael: AES - The Advanced Encryption Standard*. Information Security and Cryptography. Springer, 2002.

11. L. Dong, W.-L. Wu, S. Wu, and J. Zou. Another look at the integral attack by the higher-order differential attack. *Jisuanji Xuebao(Chinese Journal of Computers)*, 35(9):1906–1917, 2012.
12. S. Fan and M. Duan. Improved zero-sum distinguisher for SPONGENT-88. 2015.
13. J. Guo, T. Peyrin, and A. Poschmann. The PHOTON family of lightweight hash functions. In *Advances in Cryptology - CRYPTO 2011 - 31st Annual Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2011. Proceedings*, pages 222–239, 2011.
14. J. Guo, T. Peyrin, A. Poschmann, and M. 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.
15. J. Hefferon. *Linear Algebra*. Virginia Commonwealth University Mathematics, 2006.
16. J. Jean, I. Nikolić, and T. Peyrin. Joltik v1. 3. *CAESAR Round*, 2, 2015.
17. L. R. Knudsen and D. Wagner. Integral cryptanalysis. In *Fast Software Encryption, 9th International Workshop, FSE 2002, Leuven, Belgium, February 4-6, 2002, Revised Papers*, pages 112–127, 2002.
18. M. Matsui. New block encryption algorithm MISTY. In *Fast Software Encryption, 4th International Workshop, FSE '97, Haifa, Israel, January 20-22, 1997, Proceedings*, pages 54–68, 1997.
19. A. NIST. Request for candidate algorithm nominations for the AES. *Available on-line at http://www. nist. gov/aes*.
20. N. F. Pub. 197: Advanced encryption standard (AES). *Federal Information Processing Standards Publication*, 197:441–0311, 2001.
21. B. Sun, Z. Liu, V. Rijmen, R. Li, L. Cheng, Q. Wang, H. AlKhzaimi, and C. Li. Links among impossible differential, integral and zero correlation linear cryptanalysis. In *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part I*, pages 95–115, 2015.
22. L. Sun and M. Wang. Towards a further understanding of bit-based division property. *IACR Cryptology ePrint Archive*, 2016:392, 2016.
23. S. Sun, L. Hu, M. Wang, P. Wang, K. Qiao, X. Ma, D. Shi, L. Song, and K. Fu. Towards finding the best characteristics of some bit-oriented block ciphers and automatic enumeration of (related-key) differential and linear characteristics with predefined properties. Technical report, Cryptology ePrint Archive, Report 2014/747, 2014.
24. T. Suzaki, K. Minematsu, S. Morioka, and E. Kobayashi. TWINE : A lightweight block cipher for multiple platforms. In *Selected Areas in Cryptography, 19th International Conference, SAC 2012, Windsor, ON, Canada, August 15-16, 2012, Revised Selected Papers*, pages 339–354, 2012.
25. Y. Todo. Integral cryptanalysis on full MISTY1. In *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part I*, pages 413–432, 2015.
26. Y. Todo. Structural evaluation by generalized integral property. In *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I*, pages 287–314, 2015.
27. Y. Todo and M. Morii. Bit-based division property and application to SIMON family. In *Fast Software Encryption - 23rd International Conference, FSE 2016,*

Bochum, Germany, March 20-23, 2016, Revised Selected Papers, pages 357–377, 2016.

28. W. Wu and L. Zhang. LBlock: A lightweight block cipher. In *Applied Cryptography and Network Security - 9th International Conference, ACNS 2011, Nerja, Spain, June 7-10, 2011. Proceedings*, pages 327–344, 2011.
29. Z. Xiang, W. Zhang, Z. Bao, and D. Lin. Applying MILP method to searching integral distinguishers based on division property for 6 lightweight block ciphers. In *Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part I*, pages 648–678, 2016.
30. G. Yang, B. Zhu, V. Suder, M. D. Aagaard, and G. Gong. The Simeck family of lightweight block ciphers. In *Cryptographic Hardware and Embedded Systems - CHES 2015 - 17th International Workshop, Saint-Malo, France, September 13-16, 2015, Proceedings*, pages 307–329, 2015.
31. M. R. Z'aba, H. Raddum, M. Henricksen, and E. Dawson. Bit-pattern based integral attack. In *Fast Software Encryption, 15th International Workshop, FSE 2008, Lausanne, Switzerland, February 10-13, 2008, Revised Selected Papers*, pages 363–381, 2008.
32. W. Zhang, Z. Bao, D. Lin, V. Rijmen, B. Yang, and I. Verbauwhede. RECTANGLE: a bit-slice lightweight block cipher suitable for multiple platforms. *SCIENCE CHINA Information Sciences*, 58(12):1–15, 2015.

Supplementary Materials

Most of the **Supplementary Materials** are the distinguishers corresponding to the experimental results in the main body of the paper. When we present an integral distinguisher, the following symbols are adopted. '$\mathcal{A}^i$' represents an i-bit vector with every bit being active. '$\mathcal{C}^i$' denotes an i-bit vector with every bit being constant. '$\mathcal{B}^i$' indicates an i-bit vector with every bit satisfying zero-sum property. '$\mathcal{U}^i$' means an i-bit vector and the properties of the internal bits are all unknown.

A: Obtaining Primitive Representation of a Linear Transformation

Suppose that there is a linear transformation h operating on n-bit vectors, and we want to get its primitive representation. Namely, we aim to find an $n \times n$ matrix $M_h^{\mathcal{PR}}$ satisfying $h(\boldsymbol{x})^T = M_h^{\mathcal{PR}} \cdot \boldsymbol{x}^T$. Let

$$M_h^{\mathcal{PR}} = \begin{pmatrix} m_{0,0} & m_{0,1} & \cdots & m_{0,n-1} \\ m_{1,0} & m_{1,1} & \cdots & m_{1,n-1} \\ \vdots & \vdots & \ddots & \vdots \\ m_{n-1,0} & m_{n-1,1} & \cdots & m_{n-1,n-1} \end{pmatrix},$$

and denote $\boldsymbol{e}_i$ the i-th ($i = 0, 1, \ldots, n-1$) unit vector. Note that

$$M_h^{\mathcal{PR}} \cdot \boldsymbol{e}_i^T = \begin{pmatrix} m_{0,i} \\ m_{1,i} \\ \vdots \\ m_{n-1,i} \end{pmatrix}.$$

Thus, to determine the elements of $M_h^{\mathcal{PR}}$, we only need to let h operate on all n unit vectors, and put the resulted vectors corresponding to $\boldsymbol{e}_i$ into the i-th column of the matrix.

For more details, please refer to **Section III.1** of [15].

B: Auxiliary Materials for `Midori64`

B.1: Linear Inequalities of `Midori64`'s S-box

Suppose that

$$(x_0, x_1, x_2, x_3) \xrightarrow{S_{\texttt{Midori64}}} (y_0, y_1, y_2, y_3)$$

is a division trail of $S_{\texttt{Midori64}}$, then x_0, x_1, ..., x_3, y_0, y_1, ..., y_3 satisfy the linear inequality system (7).

$$\begin{cases} x_0 + x_1 + 4x_2 + x_3 - 2y_0 - 2y_1 - 2y_2 - 2y_3 \geqslant -1 \\ -3x_2 + y_0 + y_1 - 2y_2 + y_3 \geqslant -2 \\ -y_0 - y_1 + 2y_2 - y_3 \geqslant -1 \\ -x_0 - x_1 - x_3 + 2y_0 + 2y_1 + 2y_2 + 2y_3 \geqslant 0 \\ -x_1 - x_3 + y_1 + y_2 + y_3 \geqslant -1 \\ x_0, x_1, \ldots, x_3, y_0, y_1, \ldots, y_3 \text{ are binaries} \end{cases} \tag{7}$$

B.2: Integral Distinguishers for `Midori64`

The variables used to build MILP model for one round `Midori64` are illustrated in **Figure 4**. On the one hand, the allocation of the variables may influence the look of the resulting linear inequality system. On the other hand, since each variable represents the division property of the corresponding bit, the order of variables is exactly same to the order of elements in division property. For example, if we say that the initial division property of `Midori64` follows $\mathcal{D}^{1^{64}}_{\{\boldsymbol{k}\}}$, where $\boldsymbol{k} = (k_0, k_1, \ldots, k_{63})$, we mean that $x_0 = k_0$, $x_1 = k_1$, ..., and $x_{63} = k_{63}$.

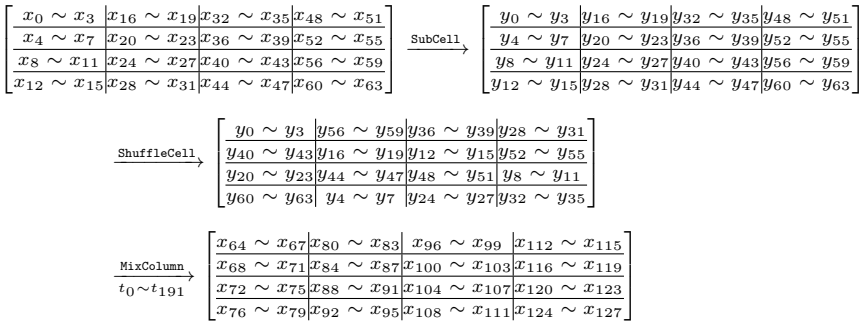


Fig. 4: Variables for One Round of `Midori64`.

4-Round Integral Distinguisher with Data Complexity 2^4

The input of the distinguisher satisfies

$$(\mathcal{A}^4\mathcal{C}^{12}, \mathcal{C}^{16}, \mathcal{C}^{16}, \mathcal{C}^{16}).$$

After four rounds of encryption, there are 4 bits satisfying zero-sum property, which are labeled as $0 \sim 3$.

5-Round Integral Distinguisher with Data Complexity 2^{12}

The input of the distinguisher follows

$$(\mathcal{A}^4\mathcal{C}^{12}, \mathcal{C}^4\mathcal{A}^4\mathcal{C}^8, \mathcal{C}^8\mathcal{A}^4\mathcal{C}^4, \mathcal{C}^{16}).$$

There still are 4 bits satisfying zero-sum property after five rounds of encryption, whose labels are 2, 6, 10 and 14.

6-Round Integral Distinguisher with Data Complexity 2^{45}

The input of the distinguisher has the following form

$$(\mathcal{A}^8\mathcal{C}^4\mathcal{A}^4, \mathcal{A}^{12}\mathcal{C}^4, \mathcal{C}^4\mathcal{A}^{12}, \mathcal{A}^1\mathcal{C}^7\mathcal{A}^8).$$

After six rounds of encryption, there are 16 bits satisfying zero-sum property. These bits are dyed in red in **Figure 5**.

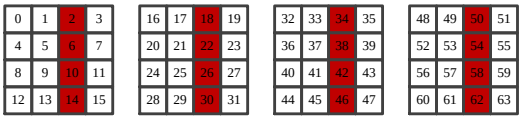


Fig. 5: The Output of the 6-Round and 7-Round Distinguishers for `Midori64`

7-Round Integral Distinguisher with Data Complexity 2^{61}

The input of the distinguisher follows

$$(\mathcal{A}^{16}, \mathcal{A}^{16}, \mathcal{A}^{16}, \mathcal{A}^{12}\mathcal{C}^{2}\mathcal{A}^{1}\mathcal{C}^{1}).$$

Then, after seven rounds of encryption, there are 16 zero-sum bits. They are also those bits dyed in red in **Figure 5**.

C: Integral Distinguishers for Some AES-Like Block Ciphers

C.1: Integral Distinguishers for LED

3-Round Integral Distinguisher with Data Complexity 2^{4}

$$(\mathcal{A}^{4}\mathcal{C}^{12}, \mathcal{C}^{16}, \mathcal{C}^{16}, \mathcal{C}^{16}) \xrightarrow{3\text{ Rounds}} (\mathcal{B}^{16}, \mathcal{B}^{16}, \mathcal{B}^{16}, \mathcal{B}^{16})$$

4-Round Integral Distinguisher with Data Complexity 2^{11}

$$(\mathcal{A}^{4}\mathcal{C}^{12}, \mathcal{C}^{4}\mathcal{A}^{4}\mathcal{C}^{8}, \mathcal{C}^{8}\mathcal{A}^{3}\mathcal{C}^{5}, \mathcal{C}^{16}) \xrightarrow{4\text{ Rounds}} (\mathcal{B}^{16}, \mathcal{B}^{16}, \mathcal{B}^{16}, \mathcal{B}^{16})$$

5-Round Integral Distinguisher with Data Complexity 2^{31}

$$(\mathcal{A}^{4}\mathcal{C}^{8}\mathcal{A}^{3}\mathcal{C}^{1}, \mathcal{A}^{8}\mathcal{C}^{8}, \mathcal{C}^{4}\mathcal{A}^{8}\mathcal{C}^{4}, \mathcal{C}^{8}\mathcal{A}^{8}) \xrightarrow{5\text{ Rounds}} (\mathcal{B}^{16}, \mathcal{B}^{16}, \mathcal{B}^{16}, \mathcal{B}^{16})$$

C.2: Integral Distinguishers for `Joltik-BC`

3-Round Integral Distinguisher with Data Complexity 2^{4}

$$(\mathcal{A}^{4}\mathcal{C}^{12}, \mathcal{C}^{16}, \mathcal{C}^{16}, \mathcal{C}^{16}) \xrightarrow{3\text{ Rounds}} (\mathcal{B}^{16}, \mathcal{B}^{16}, \mathcal{B}^{16}, \mathcal{B}^{16})$$

4-Round Integral Distinguishers with Data Complexity 2^{11}

$$(\mathcal{A}^{4}\mathcal{C}^{12}, \mathcal{C}^{4}\mathcal{A}^{4}\mathcal{C}^{8}, \mathcal{C}^{8}\mathcal{A}^{1}\mathcal{C}^{1}\mathcal{A}^{2}\mathcal{C}^{4}, \mathcal{C}^{16}) \xrightarrow{4\text{ Rounds}} (\mathcal{B}^{16}, \mathcal{B}^{16}, \mathcal{B}^{16}, \mathcal{B}^{16})$$
$$(\mathcal{A}^{4}\mathcal{C}^{12}, \mathcal{C}^{4}\mathcal{A}^{4}\mathcal{C}^{8}, \mathcal{C}^{8}\mathcal{A}^{2}\mathcal{C}^{1}\mathcal{A}^{1}\mathcal{C}^{4}, \mathcal{C}^{16}) \xrightarrow{4\text{ Rounds}} (\mathcal{B}^{16}, \mathcal{B}^{16}, \mathcal{B}^{16}, \mathcal{B}^{16})$$

5-Round Integral Distinguishers with Data Complexity 2^{31}

$$(\mathcal{A}^4\mathcal{C}^8\mathcal{A}^1\mathcal{C}^1\mathcal{A}^2, \mathcal{A}^8\mathcal{C}^8, \mathcal{C}^4\mathcal{A}^8\mathcal{C}^4, \mathcal{C}^8\mathcal{A}^8) \xrightarrow{5 \text{ Rounds}} (\mathcal{B}^{16}, \mathcal{B}^{16}, \mathcal{B}^{16}, \mathcal{B}^{16})$$
$$(\mathcal{A}^4\mathcal{C}^8\mathcal{A}^2\mathcal{C}^1\mathcal{A}^1, \mathcal{A}^8\mathcal{C}^8, \mathcal{C}^4\mathcal{A}^8\mathcal{C}^4, \mathcal{C}^8\mathcal{A}^8) \xrightarrow{5 \text{ Rounds}} (\mathcal{B}^{16}, \mathcal{B}^{16}, \mathcal{B}^{16}, \mathcal{B}^{16})$$

C.3: Integral Distinguishers for AES

3-Round Integral Distinguisher with Data Complexity 2^8

$$(\mathcal{A}^8\mathcal{C}^{24}, \mathcal{C}^{32}, \mathcal{C}^{32}, \mathcal{C}^{32}) \xrightarrow{3 \text{ Rounds}} (\mathcal{B}^{32}, \mathcal{B}^{32}, \mathcal{B}^{32}, \mathcal{B}^{32})$$

4-Round Integral Distinguisher with Data Complexity 2^{32}

$$(\mathcal{A}^8\mathcal{C}^{24}, \mathcal{C}^8\mathcal{A}^8\mathcal{C}^{16}, \mathcal{C}^{16}\mathcal{A}^8\mathcal{C}^8, \mathcal{C}^{24}\mathcal{A}^8) \xrightarrow{4 \text{ Rounds}} (\mathcal{B}^{32}, \mathcal{B}^{32}, \mathcal{B}^{32}, \mathcal{B}^{32})$$

4-Round Integral Distinguisher with Data Complexity 2^{127}

$$(\mathcal{A}^{32}, \mathcal{A}^{32}, \mathcal{A}^{32}, \mathcal{A}^{31}\mathcal{C}^1) \xrightarrow{4 \text{ Rounds}} (\mathcal{B}^{32}, \mathcal{B}^{32}, \mathcal{B}^{32}, \mathcal{B}^{32})$$

D: Integral Distinguishers for Serpent

The allocation of variables for one round Serpent is depicted in **Figure 6**. When across the linear layer, the first line of the internal state represents X_0 and the last line of the internal state represents X_3.

$$\begin{bmatrix} a_{124}|a_{120}|a_{116}|\cdots|a_4|a_0 \\ a_{125}|a_{121}|a_{117}|\cdots|a_5|a_1 \\ a_{126}|a_{122}|a_{118}|\cdots|a_6|a_2 \\ a_{127}|a_{123}|a_{119}|\cdots|a_7|a_3 \end{bmatrix} \xrightarrow{\text{S-box}} \begin{bmatrix} b_{124}|b_{120}|b_{116}|\cdots|b_4|b_0 \\ b_{125}|b_{121}|b_{117}|\cdots|b_5|b_1 \\ b_{126}|b_{122}|b_{118}|\cdots|b_6|b_2 \\ b_{127}|b_{123}|b_{119}|\cdots|b_7|b_3 \end{bmatrix} \xrightarrow[t_0 \sim t_{609}]{\text{Linear}} \begin{bmatrix} a_{252}|a_{248}|a_{244}|\cdots|a_{132}|a_{128} \\ a_{253}|a_{249}|a_{245}|\cdots|a_{133}|a_{129} \\ a_{254}|a_{250}|a_{246}|\cdots|a_{134}|a_{130} \\ a_{255}|a_{251}|a_{247}|\cdots|a_{135}|a_{131} \end{bmatrix}$$

Fig. 6: Variables for One Round of Serpent.

D.1: Integral Distinguishers for Serpent Starting from the First Round

4-Round Integral Distinguisher with Data Complexity 2^{23}

The input of the distinguisher has the following form

$$(\mathcal{A}^{22}\mathcal{C}^1\mathcal{A}^1\mathcal{C}^8, \mathcal{C}^{32}, \mathcal{C}^{32}, \mathcal{C}^{32}).$$

After four rounds of encryption, there still is one bit satisfying zero-sum property, whose label is 37.

5-Round Integral Distinguisher with Data Complexity 2^{83}

$$(\mathcal{A}^{32}, \mathcal{A}^{32}, \mathcal{A}^{18}\mathcal{C}^{1}\mathcal{A}^{1}\mathcal{C}^{12}, \mathcal{C}^{32}) \xrightarrow{5 \text{ Rounds}} (\mathcal{B}^{32}, \mathcal{B}^{32}, \mathcal{B}^{32}, \mathcal{B}^{32})$$

6-Round Integral Distinguisher with Data Complexity 2^{113}

$$(\mathcal{A}^{32}, \mathcal{A}^{32}, \mathcal{A}^{32}, \mathcal{A}^{17}\mathcal{C}^{15}) \xrightarrow{6 \text{ Rounds}} (\mathcal{B}^{32}, \mathcal{B}^{32}, \mathcal{B}^{32}, \mathcal{B}^{32})$$

7-Round Integral Distinguisher with Data Complexity 2^{124}

$$(\mathcal{A}^{32}, \mathcal{A}^{32}, \mathcal{A}^{32}, \mathcal{A}^{28}\mathcal{C}^{4}) \xrightarrow{7 \text{ Rounds}} (\mathcal{B}^{32}, \mathcal{B}^{32}, \mathcal{B}^{32}, \mathcal{B}^{32})$$

Since the 6-round and 7-round distinguishers for different initial rounds are exactly the same, we only list the 4-round and 5-round distinguishers in the remaining cases.

D.2: Integral Distinguishers for Serpent Starting from the Second Round

4-Round Integral Distinguisher with Data Complexity 2^{24}

Suppose that the input follows

$$(\mathcal{A}^{24}\mathcal{C}^{8}, \mathcal{C}^{32}, \mathcal{C}^{32}, \mathcal{C}^{32}).$$

Then there are 97 bits with zero-sum property after four rounds of encryption, which are dyed in red in **Figure 7**.

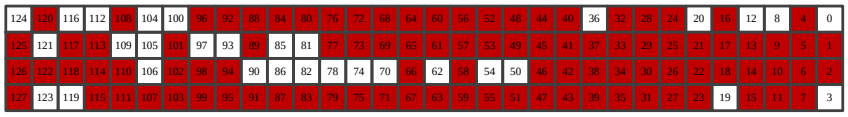


Fig. 7: The Output of the 4-Round Distinguisher for Serpent Starting from the Second Round.

5-Round Integral Distinguisher with Data Complexity 2^{83}

$$(\mathcal{A}^{32}, \mathcal{A}^{32}, \mathcal{A}^{19}\mathcal{C}^{13}, \mathcal{C}^{32}) \xrightarrow{5 \text{ Rounds}} (\mathcal{B}^{32}, \mathcal{B}^{32}, \mathcal{B}^{32}, \mathcal{B}^{32})$$

D.3: Integral Distinguishers for Serpent Starting from the Third Round

4-Round Integral Distinguisher with Data Complexity 2^{24}

Suppose that the input follows

$$(\mathcal{A}^{24}\mathcal{C}^{8}, \mathcal{C}^{32}, \mathcal{C}^{32}, \mathcal{C}^{32}).$$

After four rounds of encryption, there are 44 bits satisfying zero-sum property, which are dyed in red in **Figure 8**.

Fig. 8: The Output of the 4-Round Distinguisher for Serpent Starting from the Third Round.

5-Round Integral Distinguisher with Data Complexity 2^{83}

$$(\mathcal{A}^{32}, \mathcal{A}^{32}, \mathcal{A}^{16}\mathcal{C}^{1}\mathcal{A}^{3}\mathcal{C}^{12}, \mathcal{C}^{32}) \xrightarrow{5 \text{ Rounds}} (\mathcal{B}^{32}, \mathcal{B}^{32}, \mathcal{B}^{32}, \mathcal{B}^{32})$$

D.4: Integral Distinguishers for Serpent Starting from the Fourth Round

4-Round Integral Distinguisher with Data Complexity 2^{24}

Let the input of the distinguisher follow

$$(\mathcal{A}^{24}\mathcal{C}^{8}, \mathcal{C}^{32}, \mathcal{C}^{32}, \mathcal{C}^{32}).$$

Then, there are 115 bit satisfying zero-sum property after four rounds of encryption, which are dyed in red in **Figure 9**.

Fig. 9: The Output of the 4-Round Distinguisher for Serpent Starting from the Fourth Round.

5-Round Integral Distinguisher with Data Complexity 2^{84}

$$(\mathcal{A}^{32}, \mathcal{A}^{32}, \mathcal{A}^{20}\mathcal{C}^{12}, \mathcal{C}^{32}) \xrightarrow{5 \text{ Rounds}} (\mathcal{B}^{32}, \mathcal{B}^{32}, \mathcal{B}^{32}, \mathcal{B}^{32})$$

D.5: Integral Distinguishers for Serpent Starting from the Fifth Round

4-Round Integral Distinguisher with Data Complexity 2^{24}

Let the input of the distinguisher follow

$$(\mathcal{A}^{24}\mathcal{C}^{8}, \mathcal{C}^{32}, \mathcal{C}^{32}, \mathcal{C}^{32}).$$

There are 78 zero-sum bits after four rounds of encryption, which are dyed in red in **Figure 10**.

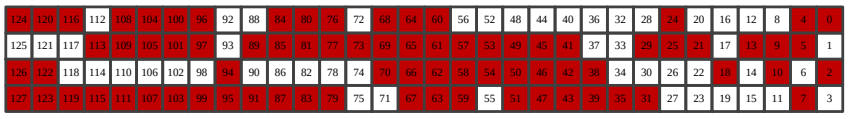


Fig. 10: The Output of the 4-Round Distinguisher for Serpent Starting from the Fifth Round.

5-Round Integral Distinguisher with Data Complexity 2^{84}

$$(\mathcal{A}^{32}, \mathcal{A}^{32}, \mathcal{A}^{20}\mathcal{C}^{12}, \mathcal{C}^{32}) \xrightarrow{5 \text{ Rounds}} (\mathcal{B}^{32}, \mathcal{B}^{32}, \mathcal{B}^{32}, \mathcal{B}^{32})$$

D.6: Integral Distinguishers for Serpent Starting from the Sixth Round

4-Round Integral Distinguisher with Data Complexity 2^{24}

$$(\mathcal{A}^{24}\mathcal{C}^{8}, \mathcal{C}^{32}, \mathcal{C}^{32}, \mathcal{C}^{32}) \xrightarrow{4 \text{ Rounds}} (\mathcal{B}^{32}, \mathcal{B}^{32}, \mathcal{B}^{32}, \mathcal{B}^{32})$$

5-Round Integral Distinguisher with Data Complexity 2^{84}

$$(\mathcal{A}^{32}, \mathcal{A}^{32}, \mathcal{A}^{20}\mathcal{C}^{12}, \mathcal{C}^{32}) \xrightarrow{5 \text{ Rounds}} (\mathcal{B}^{32}, \mathcal{B}^{32}, \mathcal{B}^{32}, \mathcal{B}^{32})$$

D.7: Integral Distinguishers for Serpent Starting from the Seventh Round

4-Round Integral Distinguisher with Data Complexity 2^{24}

Suppose that the input of the distinguisher follows

$$(\mathcal{A}^{24}\mathcal{C}^{8}, \mathcal{C}^{32}, \mathcal{C}^{32}, \mathcal{C}^{32}).$$

After four rounds of encryption, there still are 24 zero-sum bits, which are dyed in red in **Figure 11**.

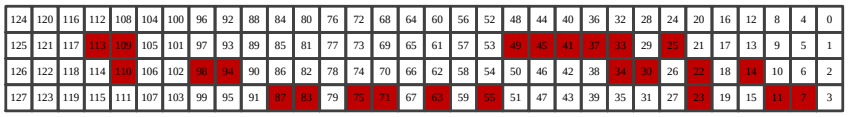


Fig. 11: The Output of the 4-Round Distinguisher for Serpent Starting from the Seventh Round.

5-Round Integral Distinguisher with Data Complexity 2^{83}

$$(\mathcal{A}^{32}, \mathcal{A}^{32}, \mathcal{A}^{17}\mathcal{C}^{1}\mathcal{A}^{2}\mathcal{C}^{12}, \mathcal{C}^{32}) \xrightarrow{5 \text{ Rounds}} (\mathcal{B}^{32}, \mathcal{B}^{32}, \mathcal{B}^{32}, \mathcal{B}^{32})$$

D.8: Integral Distinguishers for Serpent Starting from the Eighth Round

4-Round Integral Distinguisher with Data Complexity 2^{24}

Let the input of the distinguisher follows

$$(\mathcal{A}^{24}\mathcal{C}^{8}, \mathcal{C}^{32}, \mathcal{C}^{32}, \mathcal{C}^{32}).$$

There are 105 zero-sum bits after four rounds of encryption, which are dyed in red in **Figure 12**.

Fig. 12: The Output of the 4-Round Distinguisher for Serpent Starting from the Eighth Round.

5-Round Integral Distinguisher with Data Complexity 2^{84}

$$(\mathcal{A}^{32}, \mathcal{A}^{32}, \mathcal{A}^{20}\mathcal{C}^{12}, \mathcal{C}^{32}) \xrightarrow{5 \text{ Rounds}} (\mathcal{B}^{32}, \mathcal{B}^{32}, \mathcal{B}^{32}, \mathcal{B}^{32})$$

E: Integral Distinguishers for Noekeon

4-Round Integral Distinguishers with Data Complexity 2^{27}

Supposed that the input of the distinguisher follows

$$(\mathcal{A}^{6}\mathcal{C}^{26}, \mathcal{A}^{7}\mathcal{C}^{25}, \mathcal{A}^{7}\mathcal{C}^{25}, \mathcal{A}^{7}\mathcal{C}^{25}) \text{ or } (\mathcal{A}^{7}\mathcal{C}^{25}, \mathcal{A}^{6}\mathcal{C}^{26}, \mathcal{A}^{7}\mathcal{C}^{25}, \mathcal{A}^{7}\mathcal{C}^{25}).$$

Then, after four rounds of encryption, there are 25 zero-sum bits, which are dyed in red in **Figure 13**.

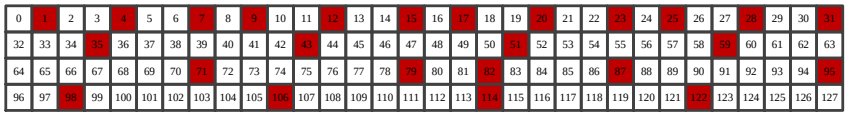


Fig. 13: The Output of the 4-Round Distinguisher for Noekeon.

5-Round Integral Distinguishers with Data Complexity 2^{83}

$$(\mathcal{A}^{20}\mathcal{C}^{12}, \mathcal{A}^{21}\mathcal{C}^{11}, \mathcal{A}^{21}\mathcal{C}^{11}, \mathcal{A}^{21}\mathcal{C}^{11}) \xrightarrow{5 \text{ Rounds}} (\mathcal{B}^{32}, \mathcal{B}^{32}, \mathcal{B}^{32}, \mathcal{B}^{32})$$
$$(\mathcal{A}^{21}\mathcal{C}^{11}, \mathcal{A}^{20}\mathcal{C}^{12}, \mathcal{A}^{21}\mathcal{C}^{11}, \mathcal{A}^{21}\mathcal{C}^{11}) \xrightarrow{5 \text{ Rounds}} (\mathcal{B}^{32}, \mathcal{B}^{32}, \mathcal{B}^{32}, \mathcal{B}^{32})$$

6-Round Integral Distinguishers with Data Complexity 2^{113}

$$(\mathcal{A}^{28}\mathcal{C}^{4}, \mathcal{A}^{28}\mathcal{C}^{4}, \mathcal{A}^{28}\mathcal{C}^{4}, \mathcal{A}^{29}\mathcal{C}^{3}) \xrightarrow{6 \text{ Rounds}} (\mathcal{B}^{32}, \mathcal{B}^{32}, \mathcal{B}^{32}, \mathcal{B}^{32})$$

7-Round Integral Distinguishers with Data Complexity 2^{124}

$$(\mathcal{A}^{31}\mathcal{C}^{1}, \mathcal{A}^{31}\mathcal{C}^{1}, \mathcal{A}^{31}\mathcal{C}^{1}, \mathcal{A}^{31}\mathcal{C}^{1}) \xrightarrow{7 \text{ Rounds}} (\mathcal{B}^{32}, \mathcal{B}^{32}, \mathcal{B}^{32}, \mathcal{B}^{32})$$

F: Auxiliary Materials for SPONGENT-88

F.1: Linear Inequality System of S_{SPONGENT}

Denote $(x_0, x_1, x_2, x_3) \xrightarrow{S_{\text{SPONGENT}}} (y_0, y_1, y_2, y_3)$ a division trail of S_{SPONGENT}. Then x_0, x_1, $\ldots$, x_3, y_0, y_1, $\ldots$, y_3 satisfy the linear inequality system (8).

$$
\begin{cases}
x_0 + x_1 + x_2 + x_3 - y_0 - y_1 - y_2 - y_3 \geqslant 0 \\
-x_1 - x_2 - 2x_3 + 2y_0 + y_1 + y_2 + y_3 \geqslant -1 \\
3x_3 - y_0 - y_1 - y_2 - y_3 \geqslant -1 \\
-x_0 + x_1 - 3y_0 + 3y_1 - 2y_2 - 2y_3 \geqslant -4 \\
-x_0 + x_2 - 3y_0 - 2y_1 + 3y_2 - 2y_3 \geqslant -4 \\
-2x_0 - x_1 - x_2 - 2x_3 + 5y_0 + 4y_1 + 4y_2 + 2y_3 \geqslant 0 \\
-x_0 - y_0 - y_1 - y_2 + 2y_3 \geqslant -2 \\
-x_0 + x_2 + y_0 - 2y_1 - y_2 - y_3 \geqslant -3 \\
-x_1 - x_2 + y_1 + y_2 + y_3 \geqslant -1 \\
3x_0 + x_1 + x_2 + x_3 - 3y_0 - 2y_1 - 2y_2 - y_3 \geqslant -2 \\
x_0, x_1, \ldots, x_3, y_0, y_1, \ldots, y_3 \text{ are binaries}
\end{cases} \tag{8}
$$

$$
\begin{cases}
x_0 + x_1 + x_2 + x_3 - y_0 - y_1 - y_2 - y_3 \geqslant 0 \\
-5x_0 - 3x_1 - 3x_2 - 4x_3 - y_0 + 2y_1 + 2y_2 + 4y_3 \geqslant -8 \\
3x_0 - y_0 - y_1 - y_2 - y_3 \geqslant -1 \\
-2x_0 - x_1 - x_3 + y_0 - 3y_1 + 2y_2 - y_3 \geqslant -5 \\
-2x_0 - x_2 - x_3 + y_0 + 2y_1 - 3y_2 - y_3 \geqslant -5 \\
-x_1 - x_2 + 2y_0 + 2y_1 + 2y_2 + y_3 \geqslant 0 \\
3x_2 + x_3 - y_0 - y_1 - 2y_2 - 2y_3 \geqslant -2 \\
3x_1 + x_3 - y_0 - 2y_1 - y_2 - 2y_3 \geqslant -2 \\
-2x_0 - x_1 - x_2 - x_3 + 2y_0 + 3y_1 + 3y_2 + 4y_3 \geqslant 0 \\
x_2 - y_0 - y_3 \geqslant -1 \\
-2x_0 - 2x_1 - x_2 + y_0 + y_2 + y_3 \geqslant -3 \\
x_0, x_1, \ldots, x_3, y_0, y_1, \ldots, y_3 \text{ are binaries}
\end{cases} \tag{9}
$$

F.2: Linear Inequality System of S^{-1}_{SPONGENT} Denote

$$
(x_0, x_1, x_2, x_3) \xrightarrow{S^{-1}_{\text{SPONGENT}}} (y_0, y_1, y_2, y_3)
$$

a division trail of S^{-1}_{SPONGENT}. Then x_0, x_1, …, x_3, y_0, y_1, …, y_3 satisfy the linear inequality system (9).

F.3: Zero-Sum Distinguishers of SPONGENT-88

The allocation of variables is simply $a_0 \sim a_{87}$, from left to right.

15-Round Zero-Sum Distinguisher with Data Complexity 2^{80}

Let $\mathcal{D}^{1^{88}}_{\{\texttt{[fffffffffff,fffffffff00]}\}}$ be the division property for the input multi-set of the seventh round, *i.e.*, we traverse the first 80 bits.

– In the forward direction, we find that the objective function is equal to 1 after eight rounds of encryption. But there are only 59 unit vectors. The absences of the other 29 unit vectors indicate that there are 29 bits satisfying zero-sum property. Those 29 bits satisfying zero-sum property are labeled as 10, $18 \sim 21$, 38, $41 \sim 43$, 60, $63 \sim 65$, 68, 70, $72 \sim 73$, $75 \sim 76$, and $78 \sim 87$.
– In the backward direction, we find that the objective function is equal to 1 after seven rounds of decryption. But there are only 68 unit vectors. The absences of the other 20 unit vectors indicate that there are 20 bits satisfying zero-sum property after seven rounds of decryption. Those 20 bits satisfying zero-sum property are labeled as 0, 8, 16, 20, 24, 28, 32, 36, 40, 44, 48, 56, 60, 64, 68, 72, $80 \sim 82$, and 84.

Combining these short integral distinguishers in different directions, we get a 15-round higher-order integral distinguisher for SPONGENT-88 with complexity 2^{80}. Comparing to the former results, our newly found distinguisher achieves one more round than the one proposed in [12] while keeps the same data complexity.

16-Round Zero-Sum Distinguisher with Data Complexity 2^{84}

Let $\mathcal{D}^{1^{88}}_{\{\texttt{[fffffffffff,ffffffffff0]}\}}$ be the division property for the input multi-set of the eighth round, *i.e.*, we traverse the first 84 bits.

– In the forward direction, we find that the objective function is equal to 1 after eight rounds of encryption. But there are only 55 unit vectors. The absences of the other 33 unit vectors tell that there are 33 zero-sum bits. Those 33 bits satisfying zero-sum property are labeled as 10, $18 \sim 21$, $38 \sim 39$, $41 \sim 43$, $60 \sim 61$, $63 \sim 65$, $68 \sim 73$, $75 \sim 76$, and $78 \sim 87$.
– In the backward direction, we observe that the objective function is equal to 1 after eight rounds of decryption. But there are only 82 unit vectors. The absences of the other 6 unit vectors indicate that there are 6 zero-sum bits, whose labels are 0, 16, 24, 40, 56, and 72.

Combining the above two short integral distinguishers in different directions, we get a 16-round higher-order integral distinguisher for SPONGENT-88 with complexity 2^{84}. Note that this newly obtained distinguisher achieves two more rounds than the previous ones.

17-Round Zero-Sum Distinguisher with Data Complexity 2^{85}

Let $\mathcal{D}^{1^{88}}_{\{\texttt{[fffffffffff,ffffffffff1]}\}}$ be the division property for the input multi-set of the eighth round.

– In the forward direction, we find that the objective function is equal to 1 after nine rounds of encryption. But there are only 86 unit vectors. The absences of the other 2 unit vectors tell that there still are 2 zero-sum bits, which are labeled as $86 \sim 87$.
– In the backward direction, we observe that the objective function is equal to 1 after eight rounds of decryption. But there are only 45 unit vectors. The

absences of the other 43 unit vectors indicate that there are 43 bits satisfying zero-sum property, whose labels are 0, 4, 8 $\sim$ 10, 12, 16 $\sim$ 18, 20, 24 $\sim$ 26, 28, 32 $\sim$ 36, 40 $\sim$ 42, 44, 48, 52, 56 $\sim$ 60, 64, 67 $\sim$ 68, 72 $\sim$ 76, and 80 $\sim$ 84.

Combining these two short distinguishers in opposite directions, we obtain a 17-round zero-sum distinguisher for SPONGENT-88 with data complexity 2^{85}.

18-Round Zero-Sum Distinguisher with Data Complexity 2^{87}

Let $\mathcal{D}^{1^{88}}_{\{[\texttt{fffffffffff,fffffffff7}]\}}$ be the division property for the input multi-set of the ninth round.

- In the forward direction, we find that the objective function is equal to 1 after nine rounds of encryption. But there are only 72 unit vectors, which indicates that there still are 16 zero-sum bits. The labels of those zero-sum bits are 21, 42 $\sim$ 43, 64 $\sim$ 65, 70, 75 $\sim$ 76, 79, and 81 $\sim$ 87.
- In the backward direction, we observe that the objective function is equal to 1 after nine rounds of decryption. However, there are only 50 unit vectors, which means that there are 38 zero-sum bits. The labels of these zero-sum bits are 0, 4, 8 $\sim$ 10, 16 $\sim$ 18, 20, 24 $\sim$ 26, 28, 32 $\sim$ 36, 40 $\sim$ 42, 44, 48, 52, 56 $\sim$ 59, 64, 67 $\sim$ 68, 72 $\sim$ 74, 80 $\sim$ 82, and 84.

Integrating these two distinguishers in the opposite directions, we get an 18-round zero-sum distinguisher, which gains four more rounds than the previous ones.

F.4: Zero-Sum Distinguishers of SPONGENT-128

The allocation of variables is simply $a_0 \sim a_{127}$, from left to right.

20-Round Zero-Sum Distinguisher with Data Complexity 2^{126}

$\mathcal{D}^{1^{128}}_{\{[\texttt{ffffffff, ffffffff, ffffffff, fffffff3}]\}}$ be the division property for the input multi-set of the tenth round.

- In the forward direction, we find that the objective function is equal to 1 after ten rounds of encryption. But there are only 127 unit vectors, which indicates that there still is one zero-sum bit. And its label is 127.
- In the backward direction, we observe that the objective function is equal to 1 after ten rounds of decryption. However, there are 127 unit vectors, which means that there still is one zero-sum bit. And its label is 0.

Combining the distinguishers in different directions, we obtain a 20-round zero-sum distinguisher.

F.5: Zero-Sum Distinguishers of SPONGENT-160

The allocation of variables is simply $a_0 \sim a_{159}$, from left to right.

21-Round Zero-Sum Distinguisher with Data Complexity 2^{159}

$\mathcal{D}^{1^{160}}_{\{[\texttt{ffffffff, ffffffff, ffffffff, ffffffff, fffffff7}]\}}$ be the division property for the input multi-set of the eleventh round.

- In the forward direction, we find that the objective function is equal to 1 after ten rounds of encryption. And there are 115 unit vectors, which indicates that there are 45 zero-sum bits. These zero-sum bits are labeled as 9, 19, 29, 34, $37 \sim 39$, 59, 69, 72, 74, $77 \sim 79$, 99, 109, 112, 114, $117 \sim 119$, $123 \sim 124$, $127 \sim 129$, $133 \sim 134$, $137 \sim 139$, $143 \sim 144$, $147 \sim 149$, and $151 \sim 159$.
- In the backward direction, we observe that the objective function is equal to 1 after eleven rounds of decryption. However, there are 159 unit vectors, which means that there still is one zero-sum bit. And its label is 96.

Integrating the distinguishers in opposite directions, we get a 21-round zero-sum distinguisher.

G: Auxiliary Materials for PHOTON Permutations

G.1: Integral Distinguishers for P_{100}

3-Round Integral Distinguisher with Data Complexity 2^4

$$(\mathcal{A}^4\mathcal{C}^{16}, \mathcal{C}^{20}, \mathcal{C}^{20}, \mathcal{C}^{20}, \mathcal{C}^{20}) \xrightarrow{3 \text{ Rounds}} (\mathcal{B}^{20}, \mathcal{B}^{20}, \mathcal{B}^{20}, \mathcal{B}^{20}, \mathcal{B}^{20})$$

4-Round Integral Distinguisher with Data Complexity 2^{11}

$$(\mathcal{A}^4\mathcal{C}^{16}, \mathcal{C}^4\mathcal{A}^4\mathcal{C}^{12}, \mathcal{C}^8\mathcal{A}^3\mathcal{C}^9, \mathcal{C}^{20}, \mathcal{C}^{20}) \xrightarrow{4 \text{ Rounds}} (\mathcal{B}^{20}, \mathcal{B}^{20}, \mathcal{B}^{20}, \mathcal{B}^{20}, \mathcal{B}^{20})$$

5-Round Integral Distinguisher with Data Complexity 2^{20}

$$(\mathcal{A}^4\mathcal{C}^{16}, \mathcal{C}^4\mathcal{A}^4\mathcal{C}^{12}, \mathcal{C}^8\mathcal{A}^4\mathcal{C}^8, \mathcal{C}^{12}\mathcal{A}^4\mathcal{C}^4, \mathcal{C}^{16}\mathcal{A}^4) \xrightarrow{5 \text{ Rounds}} (\mathcal{B}^{20}, \mathcal{B}^{20}, \mathcal{B}^{20}, \mathcal{B}^{20}, \mathcal{B}^{20})$$

G.2: Integral Distinguishers for P_{144}

3-Round Integral Distinguisher with Data Complexity 2^4

$$(\mathcal{A}^4\mathcal{C}^{20}, \mathcal{C}^{24}, \mathcal{C}^{24}, \mathcal{C}^{24}, \mathcal{C}^{24}, \mathcal{C}^{24}) \xrightarrow{3 \text{ Rounds}} (\mathcal{B}^{24}, \mathcal{B}^{24}, \mathcal{B}^{24}, \mathcal{B}^{24}, \mathcal{B}^{24}, \mathcal{B}^{24})$$

4-Round Integral Distinguisher with Data Complexity 2^{11}

$$(\mathcal{A}^4\mathcal{C}^{20}, \mathcal{C}^4\mathcal{A}^4\mathcal{C}^{16}, \mathcal{C}^8\mathcal{A}^3\mathcal{C}^{13}, \mathcal{C}^{24}, \mathcal{C}^{24}, \mathcal{C}^{24}) \xrightarrow{4 \text{ Rounds}} (\mathcal{B}^{24}, \mathcal{B}^{24}, \mathcal{B}^{24}, \mathcal{B}^{24}, \mathcal{B}^{24}, \mathcal{B}^{24})$$

5-Round Integral Distinguisher with Data Complexity 2^{24}

$$(\mathcal{A}^4\mathcal{C}^{20}, \mathcal{C}^4\mathcal{A}^4\mathcal{C}^{16}, \mathcal{C}^8\mathcal{A}^4\mathcal{C}^{12}, \mathcal{C}^{12}\mathcal{A}^4\mathcal{C}^8, \mathcal{C}^{16}\mathcal{A}^4\mathcal{C}^4, \mathcal{C}^{20}\mathcal{A}^4)$$
$$\xrightarrow{5 \text{ Rounds}} (\mathcal{B}^{24}, \mathcal{B}^{24}, \mathcal{B}^{24}, \mathcal{B}^{24}, \mathcal{B}^{24}, \mathcal{B}^{24})$$

7-Round Integral Distinguisher with Data Complexity 2^{131}

$$(\mathcal{A}^{24}, \mathcal{A}^{24}, \mathcal{A}^{11}\mathcal{C}^1\mathcal{A}^{12}, \mathcal{A}^{12}\mathcal{C}^4\mathcal{A}^8, \mathcal{A}^{16}\mathcal{C}^4\mathcal{A}^4, \mathcal{A}^{20}\mathcal{C}^4)$$
$$\xrightarrow{7 \text{ Rounds}} (\mathcal{B}^{24}, \mathcal{B}^{24}, \mathcal{B}^{24}, \mathcal{B}^{24}, \mathcal{B}^{24}, \mathcal{B}^{24})$$

G.3: Integral Distinguishers for P_{196}

3-Round Integral Distinguisher with Data Complexity 2^4

$$(\mathcal{A}^4\mathcal{C}^{24}, \mathcal{C}^{28}, \mathcal{C}^{28}, \mathcal{C}^{28}, \mathcal{C}^{28}, \mathcal{C}^{28}, \mathcal{C}^{28}) \xrightarrow{3 \text{ Rounds}} (\mathcal{B}^{28}, \mathcal{B}^{28}, \mathcal{B}^{28}, \mathcal{B}^{28}, \mathcal{B}^{28}, \mathcal{B}^{28}, \mathcal{B}^{28})$$

4-Round Integral Distinguisher with Data Complexity 2^{11}

$$(\mathcal{A}^4\mathcal{C}^{24}, \mathcal{C}^4\mathcal{A}^4\mathcal{C}^{20}, \mathcal{C}^8\mathcal{A}^3\mathcal{C}^{17}, \mathcal{C}^{28}, \mathcal{C}^{28}, \mathcal{C}^{28}, \mathcal{C}^{28})$$
$$\xrightarrow{4 \text{ Rounds}} (\mathcal{B}^{28}, \mathcal{B}^{28}, \mathcal{B}^{28}, \mathcal{B}^{28}, \mathcal{B}^{28}, \mathcal{B}^{28}, \mathcal{B}^{28})$$

8-Round Integral Distinguisher with Data Complexity 2^{192}

$$(\mathcal{C}^4\mathcal{A}^{24}, \mathcal{A}^{28}, \mathcal{A}^{28}, \mathcal{A}^{28}, \mathcal{A}^{28}, \mathcal{A}^{28}, \mathcal{A}^{28}) \xrightarrow{8 \text{ Rounds}} (\mathcal{B}^{28}, \mathcal{B}^{28}, \mathcal{B}^{28}, \mathcal{B}^{28}, \mathcal{B}^{28}, \mathcal{B}^{28}, \mathcal{B}^{28})$$

G.4: Integral Distinguishers for P_{256}

3-Round Integral Distinguisher with Data Complexity 2^4

$$(\mathcal{A}^4\mathcal{C}^{28}, \mathcal{C}^{32}, \mathcal{C}^{32}, \mathcal{C}^{32}, \mathcal{C}^{32}, \mathcal{C}^{32}, \mathcal{C}^{32}, \mathcal{C}^{32})$$
$$\xrightarrow{3 \text{ Rounds}} (\mathcal{B}^{32}, \mathcal{B}^{32}, \mathcal{B}^{32}, \mathcal{B}^{32}, \mathcal{B}^{32}, \mathcal{B}^{32}, \mathcal{B}^{32}, \mathcal{B}^{32})$$

4-Round Integral Distinguisher with Data Complexity 2^{11}

$$(\mathcal{A}^4\mathcal{C}^{28}, \mathcal{C}^4\mathcal{A}^4\mathcal{C}^{24}, \mathcal{C}^8\mathcal{A}^3\mathcal{C}^{21}, \mathcal{C}^{32}, \mathcal{C}^{32}, \mathcal{C}^{32}, \mathcal{C}^{32}, \mathcal{C}^{32})$$
$$\xrightarrow{4 \text{ Rounds}} (\mathcal{B}^{32}, \mathcal{B}^{32}, \mathcal{B}^{32}, \mathcal{B}^{32}, \mathcal{B}^{32}, \mathcal{B}^{32}, \mathcal{B}^{32}, \mathcal{B}^{32})$$

5-Round Integral Distinguisher with Data Complexity 2^{28}

$$(\mathcal{A}^4\mathcal{C}^{28}, \mathcal{C}^4\mathcal{A}^4\mathcal{C}^{24}, \mathcal{C}^8\mathcal{A}^4\mathcal{C}^{20}, \mathcal{C}^{12}\mathcal{A}^4\mathcal{C}^{16}, \mathcal{C}^{16}\mathcal{A}^4\mathcal{C}^{12}, \mathcal{C}^{20}\mathcal{A}^4\mathcal{C}^8, \mathcal{C}^{24}\mathcal{A}^4\mathcal{C}^4, \mathcal{C}^{32})$$
$$\xrightarrow{5 \text{ Rounds}} (\mathcal{B}^{32}, \mathcal{B}^{32}, \mathcal{B}^{32}, \mathcal{B}^{32}, \mathcal{B}^{32}, \mathcal{B}^{32}, \mathcal{B}^{32}, \mathcal{B}^{32})$$

9-Round Integral Distinguisher with Data Complexity 2^{252} The input of the distinguisher follows the following form.

$$(\mathcal{C}^4\mathcal{A}^{28}, \mathcal{A}^{32}, \mathcal{A}^{32}, \mathcal{A}^{32}, \mathcal{A}^{32}, \mathcal{A}^{32}, \mathcal{A}^{32}, \mathcal{A}^{32})$$

After nine rounds of encryption, there are 96 bits satisfying zero-sum property. These bits are dyed in red in **Figure 14**.

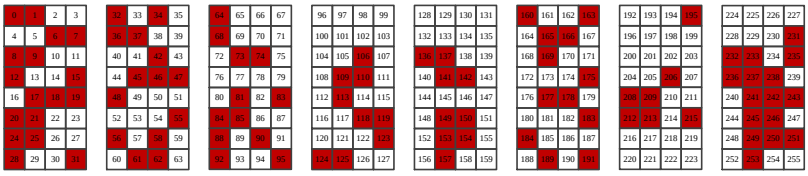


Fig. 14: The Output of the 9-Round Integral Distinguisher for P_{256}.

G.5: Integral Distinguishers for P_{288}

3-Round Integral Distinguisher with Data Complexity 2^8

$$(\mathcal{A}^8\mathcal{C}^{40}, \mathcal{C}^{48}, \mathcal{C}^{48}, \mathcal{C}^{48}, \mathcal{C}^{48}, \mathcal{C}^{48}) \xrightarrow{3 \text{ Rounds}} (\mathcal{B}^{48}, \mathcal{B}^{48}, \mathcal{B}^{48}, \mathcal{B}^{48}, \mathcal{B}^{48}, \mathcal{B}^{48})$$

4-Round Integral Distinguisher with Data Complexity 2^{48}

$$(\mathcal{A}^8\mathcal{C}^{40}, \mathcal{C}^8\mathcal{A}^8\mathcal{C}^{32}, \mathcal{C}^{16}\mathcal{A}^8\mathcal{C}^{24}, \mathcal{C}^{24}\mathcal{A}^8\mathcal{C}^{16}, \mathcal{C}^{32}\mathcal{A}^8\mathcal{C}^8, \mathcal{C}^{40}\mathcal{A}^8)$$
$$\xrightarrow{4 \text{ Rounds}} (\mathcal{B}^{48}, \mathcal{B}^{48}, \mathcal{B}^{48}, \mathcal{B}^{48}, \mathcal{B}^{48}, \mathcal{B}^{48})$$