MILP-Aided Bit-Based Division Property for ARX-Based Block Cipher

Ling Sun[1], Wei Wang[1], Ru Liu[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. The huge time and memory complexities of utilizing bit-based division property, which was first presented by Todo and Morri at FSE 2016, bothered cryptographers for quite some time and it had been solved by Xiang *et al.* at ASIACRYPT 2016. They applied MILP method to search integral distinguisher based on division property, and used it to analyze six lightweight block ciphers. Later on, Sun *et al.* handled the feasibility of MILP-aided bit-based division property for primitives with non-bit-permutation linear layers. Although MILP-aided bit-based division property has gave many perfect results since its appearance, there still are many left problems when we want to develop its further applications. In this paper, we focus on the feasibility of MILP-aided bit-based division property for ARX-based primitive. More specifically, we consider the construction of MILP models for some components of ARX-based structure. Firstly, the `Modulo` model is proposed by using its iterated expression and introducing some auxiliary variables. Then, to propagate the operations of `AND` and `OR` with a constant (or a subkey), we prove that the known-region deduced by the input division property is always included in the known-region derived from the output division property, which allows us to ignore these operations. Furthermore, with its help, we also handle the `Modulo` operation with a constant (or a subkey). As a result, these new models are exploited to search integral distinguishers for some ARX-based block ciphers. For HIGHT and LEA, the lengths of the distinguishers both are improved by one round. Some 15-round integral distinguishers for TEA/XTEA are presented. Comparing with the existing one transformed by utilizing the equivalence between zero-correlation and integral cryptanalysis, our newly obtained distinguishers either reduces the data requirement or increases the number of zero-sum bits. Moreover, the bit-based division properties for KATAN and KTANTAN families of block ciphers are also provided.

Keywords: MILP-aided bit-based division property, HIGHT, LEA, TEA, XTEA, KATAN, KTANTAN

* Corresponding Author

1 Introduction

Division property, which was proposed by Todo [12] at EUROCRYPT 2015, is a generalization of integral property. It could explicitly depict the obscure property between the traditional ALL and BALANCE properties, which made it an effective method to search integral distinguishers even if the internal components of the objective primitives had non-bijective, bit-oriented, and low-degree structures. However, the utilization of the algebraic degree alone did not maximize the advantage of the division property. At CRYPTO 2015, after analyzing the Algebraic Normal Form (ANF) of S_7 for MISTY1 [7], Todo [11] observed a vulnerable property consisted in S_7, and constructed a 6-round integral distinguisher. With this newly proposed distinguisher, he provided the first theoretical attack for full MISTY1.

By restricting the underlying space to be the direct product of a series of binary fields, Todo and Morri [13] proposed bit-based division property, which could be treated as a special case of division property. The bit-based division property broke the internal states into bits, which allowed it to use not only the algebraic degree but also the details of the round function's structure, and directly traced the division property at the bit-level. It was applied to construct a 14-round integral distinguisher for SIMON32 [1], which achieved four more rounds than the one given in [12]. However, the complexity of utilizing bit-based division property was roughly equal to 2^n for an n-bit primitives. On the one hand, the huge complexity restricted the wide applications of bit-based division property. On the other hand, whether it could be used to find integral distinguishers for other block ciphers with S-boxes was not known.

To solve the restriction of the huge complexity, Xiang *et al.* [17] applied MILP-method to search integral distinguisher based on bit-based division property[3], which allowed us to analyze primitives whose block sizes were larger than 32. With the help of MILP-method, bit-based division property was successfully applied to search integral distinguishers for six lightweight block ciphers. Later on, the feasibility of MILP-aided bit-based division property for primitives with non-bit-permutation linear layers, which was an open problem left by Xiang *et al.*, was settled by Sun *et al.* [10]. After generalizing the former **Copy** and `XOR` models, they considered the primitive representations of the linear layers and proposed a new method to construct the model used to propagate bit-based division property of linear layers by introducing some intermediate variables according to the primitive representations.

Although MILP-aided bit-based division property has gave many wonderful results since its appearance, there still are many left problems when we want to develop its further applications. In this paper, we focus on the feasibility of MILP-aided bit-based division property for ARX-based primitive. More specifically, we consider the construction of MILP models for some components of ARX-based structure. ARX, which refers to as Addition/Rotation/XOR, is a

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

class of block ciphers that only use the following simple operations[4]: `Modulo`, bitwise rotation and `XOR`. In contrast to those block ciphers with S-boxes, their nonlinearities rely on the `Modulo` operation. ARX designs are simple, efficient and easy to implement. Given their flexibility and efficiency, it is expected that ARX designs will continue to be popular. However, the security analysis of this kind of ciphers has not been extensively studied and any general method for analyzing them will influence future cipher designs.

1.1 Our Contributions

In this paper, we aim at solving the feasibility of MILP-aided bit-based division property for ARX-based primitive and expending its application range. The contributions are summarized as follows.

1. Note that the ANF of the `Modulo` operation becomes more and more complicated with the increasing of modulus and directly using the ANF to construct MILP model is very hard, we consider an iterated expression of the `Modulo` operation. After introducing some auxiliary variables and allocating these variables according to the iterated expression, the `Modulo` model is constructed by successively invoking **Copy**, `AND`, and `XOR` models. This linear inequality system can be absorbed into the original MILP model of bit-based division property to find integral distinguishers for some ARX-based block ciphers.
2. To propagate the bit-based division property for the operations of `AND` and `OR` with a constant (or a subkey), we prove that the known-region deduced by input division property is always included in the known-region derived from the output division property, with which these operations can be ignored during our analysis. Although doing this way may lose some useful information, sometimes we can not know such constants in advance[5]. As a result, dealing with it in this way is reasonable. With its help, we also handle the `Modulo` operation with a constant. Up to now, we solve the feasibility of MILP-aided bit-based division property for ARX-based block cipher.
3. These newly constructed models are applied to search integral distinguishers for some ARX-based block ciphers. HIGHT [6], which was proposed at CHES 2006, was designed for usage in lightweight applications such as sensor network and RFID tags, and has been adopted as ISO standard. We construct two 18-round integral distinguishers for HIGHT, and our newly obtained distinguishers gain one more round comparing with the former two 17-round distinguishers provided by Zhang *et al.* at CANS 2009. For

[4] Broadly, some ciphers that also involve the `AND` operation are also included in this class, such as SIMON and KATAN/KTANTAN. The nonlinearities of these ciphers are also provided by the `AND` operation.

[5] These constants can be subkeys, like the operations in MISTY1's FL function. Even though we know the IR's of KATAN/KTANTAN in advance, these IR's are round-dependent. If we want to find integral distinguishers workable for different starting rounds, these constants are supposed to be unknown values to some degree.

LEA [5], we find a 7-round integral distinguisher with data complexity 2^{96}, which also attains one more round than the one mentioned by the designers. TEA [16] and XTEA [8] being rather popular ciphers, both are implemented in the Linux kernal. Some 15-round integral distinguishers for TEA and XTEA are presented. Comparing with the 15-round one transformed from the 15-round zero-correlation linear approximation [3] by utilizing the equivalence between zero-correlation linear cryptanalysis and integral cryptanalysis for ARX-based block ciphers proposed by Wen and Wang [15], our newly obtained distinguishers either reduces the data requirement or increases the number of zero-sum bits. Besides, the bit-based division properties for KATAN and KTANTAN [4] families of block ciphers are also provided.

Some comparison between our newly obtained integral distinguishers and some former results can be found in **Table 1**.

Outline of the Paper. The remainder of this paper is organized as follows. In **Section 2**, we briefly review some notations, division property, and MILP-aided bit-based division property. **Section 3** focuses on the construction of MILP models for some components of ARX-based structure and illustrates how to apply MILP-aided bit-based division property to ARX-based block ciphers. **Section 4** gives the applications of some ARX-based block ciphers. We conclude the paper in **Section 5**. Some auxiliary materials are supplied in **Appendix A** and **Appendix B**.

2 Priliminary

2.1 Notations

Some notations, which are used throughout this paper, are introduced in this section.

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 [12] and [11].

For any $a \in \mathbb{F}_2^n$, the i-th element is expressed as $a[i]$, where the bit positions are labeled in big-ending, 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, $\boldsymbol{k} \not\succeq \boldsymbol{k'}$.

Table 1: Summarization of Results.

Cipher	Block Size	Length	Data Requirement	#{Zero-Sum Bits}	Ref.
HIGHT	64	11	2^8	1	[18]
		12	2^{16}	1	
		17	2^{56}	1	
		18	$\mathbf{2^{63}}$	**1**	**Section 4.1**
LEA	128	6	2^{32}	1	[5]
		6	$\mathbf{2^{32}}$	**2**	**Section 4.2**
		7	$\mathbf{2^{96}}$	**1**	
TEA/XTEA	64	**15**	2^{63}	1	[3]†
			$\mathbf{2^{62}}$	**1**	**Section 4.3**
			$\mathbf{2^{63}}$	**2**	
KATAN (KTANTAN)	32	**90**	$\mathbf{2^{30}}$	**1**	**Section 4.4**
		99	$\mathbf{2^{31}}$	**1**	
	48	**77**	$\mathbf{2^{46}}$	**1**	
		83.5	$\mathbf{2^{47}}$	**1**	
	64	**67.3**	$\mathbf{2^{62}}$	**1**	
		72.3	$\mathbf{2^{63}}$	**1**	

Length: The length of the resulting distinguisher.
#{**Zero-Sum Bits**}: The number of zero-sum bits of the distinguisher.
† Note that the 15-round zero-correlation linear approximation in [3] can be transformed into a 15-round integral distinguisher by applying **Proposition 2** in [15].

Definition 1 (Bit Product Function [12]). *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

The division property, which was proposed by Todo [12], was a generalization of the integral property. It could precisely describe the implicit properties between the traditional ALL and BALANCE properties, which made division property an efficient tool to search integral distinguishers for some ciphers with non-bijective, bit-oriented, or low-degree components. Bit-based division property [13] handled a special case of division property, where the underlying space was restricted to be the direct product of a series of binary fields. Comparing with the traditional division property, bit-based division property could trace 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 summarize some propagation rules for bit-based division property.

Definition 2 (Division Property [12]). *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 $\ell_i - 1$, 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 the vector $\boldsymbol{k}$ is redundant.*

Remark 2 (Known-Region & Unknown-Region) *In this paper, we call the region where the parity is equal to 0 the known-region and the region where the parity becomes unknown is called the unknown-region.*

Remark 3 *Note that ℓ_0, ℓ_1, ..., ℓ_{m-1} are restricted to 1 when we consider **bit-based division property**.*

Bit-Based Division Property Propagation Rules for Some Basic Operations Todo [12] proved some propagation rules for division property and these rules were summarized into five rules in [11], which were respectively **Substitution**, **Copy**, `XOR`, **Split**, and **Concatenation**. 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.

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 $(y_0, y_1) = (x, x)$. 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 $\mathcal{D}_{\{k\}}^1$, the division property of the multi-set $\mathbb{Y}$ is $\mathcal{D}_{\mathbb{K}'}^{1\times 1}$. Then the propagation only have two possible cases:*

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

Rule 2 (`XOR`) *Let F be a function compressed by an* `XOR`*, where the input (x_0, x_1) takes a value of $\mathbb{F}_2 \times \mathbb{F}_2$ and the output is calculated as $y = x_0 \oplus x_1$. 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 $\mathcal{D}^{1}_{\mathbb{K}'}$. Then the propagation only have four possible cases:*

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

For some bit-oriented block ciphers such as SIMON, `AND` is a necessary non-linear operation. The propagation for `AND` is given in [13], and we summarize it as follows.

Rule 3 (`AND`) *Let F be a function compressed by an* `AND`*, where the input (x_0, x_1) takes a value of $\mathbb{F}_2 \times \mathbb{F}_2$ and the output is calculated as $y = x_0 \wedge x_1$. 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 $\mathcal{D}^{1}_{\mathbb{K}'}$. Then the propagation only have four possible cases:*

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

We do not introduce the bit-based division property propagation of S-box, since the objectives of this paper do not involve primitives with S-box.

2.3 MILP-Aided Bit-Based Division Property

Although bit-based division property showed its power by finding longer integral distinguisher for SIMON32 [13], the time and memory complexities of utilizing it were roughly 2^n for an n-bit block cipher. To handle the huge complexities of bit-based division property, Xiang *et al.* [17] integrated MILP method with bit-based division property. After transforming division property propagation into a series of linear inequalities[6], the integral distinguisher searching problem by using bit-based division property became solving MILP problem. With the help of some openly available MILP optimizers such as Gurobi[7], the complexities of calling bit-based division property dramatically decreased. In this subsection, we will briefly review MILP-aided bit-based division property.

The main technique of MILP-aided bit-based division property is to model those bit-based division property propagation rules introduced in **Section 2.2** with a series of linear inequalities. Again, the construction of MILP model for S-box is not covered in this paper.

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

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

Modeling Copy, AND, and XOR Corresponding to three basic operations (**Copy**, AND, and XOR), there are three models which are used to depict them. Note that Sun *et al.* [10] have generalized the **Copy** and XOR models proposed by Xiang *et al.*. For simplicity, we only list the generalized **Copy** and XOR models in the following.

Model 1 (Generalized Copy, [10]) *Denote* $(a) \xrightarrow{\boldsymbol{Copy}} (b_0, b_1, \ldots, b_m)$ *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 2 (Generalized XOR, [10]) *Denote* $(a_0, a_1, \ldots, a_m) \xrightarrow{XOR} (b)$ *a division trail through* ***XOR*** *function, the following inequalities can describe the division trail through* ***XOR*** *function.*

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

Model 3 (AND, [17]) *Denote* $(a_0, a_1) \xrightarrow{AND} (b)$ *a division trail of* ***AND*** *function, the following linear inequalities are sufficient to describe this propagation.*

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

Until now, for block ciphers based on the three basic operations, like SIMON [1], 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 [17]). *Let* f_r *denote the round function of an iterated block cipher. Assume that the input multi-set to 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 an 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 $(a_0^0, a_1^0, \ldots, a_{n-1}^0) \rightarrow \cdots \rightarrow (a_0^r, a_1^r, \ldots, a_{n-1}^r)$ 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 $\mathcal{D}_{\{\boldsymbol{k}\}}^{1^n}$ 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 $\boldsymbol{k}$. And the objective function is set as

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

Let $\mathcal{D}_{\mathbb{K}_i}^{1^n}$ denote the output division property after i rounds of encryption and the input division property is denoted by $\mathcal{D}_{\mathbb{K}_0}^{1^n}$. 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 $\mathcal{D}_{\mathbb{K}_r}^{1^n}$.

Note that we only review some key points here. For more details, please refer to [9–13, 17].

3 MILP-Aided Bit-Based Division Property for ARX-Based Primitive

Even though MILP-aided bit-based division property illustrated by Xiang *et al.* [17] handled the huge complexities of bit-based division property, the feasibility of MILP-aided bit-based division property for primitives with non-bit-permutation linear layers was not considered. By generalizing the former **Copy** and `XOR` models and introducing some intermediate variables among the linear layer according to the primitive representation of the linear layer, Sun *et al.* [10] settled this open problem, and successfully applied MILP-aided bit-based division property to search integral distinguishers for various primitives with non-bit-permutation linear layers.

However, there still are many left problems when we want to develop its further applications. The feasibility of MILP-aided bit-based division property for ARX-based primitive is not known. In this section, we focus on this problem, and consider the construction of MILP model for some components of ARX-based structure.

3.1 Modelling the `Modulo` Operation

For some ARX-based block ciphers, `Modulo` operation is an important constitution which is used to provide nonlinearity. In this section, we will illustrate how to model it in MILP-aided bit-based division property.

To model the `Modulo` operation, an intuitive idea is to deal with its Boolean functions. But the ANF of the `Modulo` operation becomes more and more complicated as the modulus increasing. Thus directly handling the ANF of the `Modulo` operation is not a wise choice. Note that there is an iterated way to express the `Modulo` operation. Suppose that $\boldsymbol{x} = (x_0, x_1, \ldots, x_{n-1})$, $\boldsymbol{y} = (y_0, y_1, \ldots, y_{n-1})$,

and $\boldsymbol{z} = (z_0, z_1, \ldots, z_{n-1})$ are n-bit vectors, and $\boldsymbol{z} = \boldsymbol{x} \boxplus \boldsymbol{y}$. Then the Boolean functions of z_i can be expressed with (1).

$$\begin{cases} z_i = x_i \oplus y_i \oplus c_i \\ c_i = x_{i+1}y_{i+1} \oplus (x_{i+1} \oplus y_{i+1})c_{i+1}, i = n-2, n-3, \ldots, 0 \\ c_{n-1} = 0 \end{cases} \tag{1}$$

With this iterated expression, we can model the `Modulo` operation step by step. In the following, we take a 4-bit `Modulo` operation as an illustration. Firstly, we represent the 4-bit `Modulo` operation with the iterated expression, which is given in (2).

$$\begin{cases} z_3 = x_3 \oplus y_3 \\ z_2 = x_2 \oplus y_2 \oplus c_2, c_2 = x_3y_3 \\ z_1 = x_1 \oplus y_1 \oplus c_1, c_1 = x_2y_2 \oplus (x_2 \oplus y_2)c_2 \\ z_0 = x_0 \oplus y_0 \oplus c_0, c_0 = x_1y_1 \oplus (x_1 \oplus y_1)c_1 \end{cases} \tag{2}$$

After observation, we find that the input bits x_i's and y_i's are firstly copied a certain number of times. Then, these copies will combine with each other by using `XOR` or `AND` operation to constitute the final output bits. Since we already have the models of describing three basic operations (**Model 1**, **Model 2**, and **Model 3**), we can generate linear inequality system used to depict division property propagation for the `Modulo` operation step by step by introducing some auxiliary variables.

More specifically, intermediate variables $t_0 \sim t_3$, $u_0 \sim u_{12}$, $v_0 \sim v_2$, $m_0 \sim m_1$, $r_0 \sim r_1$, $g_0 \sim g_1$, $q_0 \sim q_1$, and $w_0 \sim w_1$ are introduced. The allocation of these intermediate variables is illustrated in **Table 2**. The linear inequality system used to propagate the bit-based division property of the 4-bit `Modulo` operation can be found in **Table 3**, and we do not distinguish between the input bits x_i's, y_i's (output bits z_i's) and their corresponding input (output) division properties here. We will give a brief explanation of the linear inequalities in **Table 3**.

- The linear inequalities used to describe the **Copy** operations are given in the first line to the sixth line of **Table 3**, where $t_i (0 \leqslant i \leqslant 3)$ and $u_j (0 \leqslant j \leqslant 12)$ are introduced to represent different copies of $x_k (1 \leqslant i \leqslant 3)$ and $y_l (1 \leqslant l \leqslant 3)$. Note that the frequencies of occurrences of x_0 and y_0 in (2) are equal to 1, so x_0 and y_0 do not needed to be copied.
- Then, according to the iterated expression of the `Modulo` operation, the remaining intermediated variables are called. For example, to depict x_3y_3 in the expression of c_2, we introduce v_0 as an auxiliary variable, and the eighth line to the tenth line of **Table 3** are explained by calling **Model 3**. It is noteworthy that the intermediate values c_2 and c_1 also need to be copied, and g_i's and r_i's bear this responsibility.
- Successively invoking **Model 1**, **Model 2**, and **Model 3**, we get 31 linear inequalities to depict the bit-based division property propagation of the 4-bit `Modulo` operation, which are listed in **Table 3**.

Table 2: Allocation of Intermediate Variables for 4-bit `Modulo`.

Bit	Variable Allocation	Bit	Variable Allocation
z_3	$\underbrace{x_3}_{t_0} \oplus \underbrace{y_3}_{t_2}$		
z_2	$\underbrace{x_2}_{u_0} \oplus \underbrace{y_2}_{u_6} \oplus \underbrace{c_2}_{g_0}$	c_2	$\overbrace{\underbrace{x_3}_{t_1}\ \underbrace{y_3}_{t_3}}^{v_0}$
z_1	$\underbrace{x_1}_{u_3} \oplus \underbrace{y_1}_{u_9} \oplus \underbrace{c_1}_{g_1}$	$\underbrace{c_1}_{w_0}$	$\overbrace{\underbrace{x_2}_{u_1}\ \underbrace{y_2}_{u_7}}^{v_1} \oplus \overbrace{\overbrace{\underbrace{x_2}_{u_2} \oplus \underbrace{y_2}_{u_8}}^{m_0}\ \overbrace{c_2}^{r_0}}^{q_0}$
z_0	$x_0 \oplus y_0 \oplus c_0$	$\underbrace{c_0}_{w_1}$	$\overbrace{\underbrace{x_1}_{u_4}\ \underbrace{y_1}_{u_{10}}}^{v_2} \oplus \overbrace{\overbrace{\underbrace{x_1}_{u_5} \oplus \underbrace{y_1}_{u_{11}}}^{m_1}\ \overbrace{c_1}^{r_1}}^{q_1}$

Table 3: Linear Inequality System of 4-Bit `Modulo`.

Label	Linear Inequalities Used to Describe 4-Bit `Modulo`
1	$x_3 - t_0 - t_1 = 0$
2	$y_3 - t_2 - t_3 = 0$
3	$x_2 - u_0 - u_1 - u_2 = 0$
4	$x_1 - u_3 - u_4 - u_5 = 0$
5	$y_2 - u_6 - u_7 - u_8 = 0$
6	$y_1 - u_9 - u_{10} - u_{11} = 0$
7	$t_0 + t_2 - z_3 = 0$
8	$v_0 - t_1 \geqslant 0$
9	$v_0 - t_3 \geqslant 0$
10	$v_0 - t_1 - t_3 \leqslant 0$
11	$v_1 - u_1 \geqslant 0$
12	$v_1 - u_7 \geqslant 0$
13	$v_1 - u_1 - u_7 \leqslant 0$
14	$v_2 - u_4 \geqslant 0$
15	$v_2 - u_{10} \geqslant 0$
16	$v_2 - u_4 - u_{10} \leqslant 0$
17	$u_2 + u_8 - m_0 = 0$
18	$u_5 + u_{11} - m_1 = 0$
19	$g_0 + r_0 - v_0 = 0$
20	$u_0 + u_6 + g_0 - z_2 = 0$
21	$q_0 - m_0 \geqslant 0$
22	$q_0 - r_0 \geqslant 0$
23	$q_0 - m_0 - r_0 \leqslant 0$

Continued on next page

Table 3 – continued from previous page

Label	Linear Inequalities Used to Describe 4-Bit `Modulo`
24	$v_1 + q_0 - w_0 = 0$
25	$g_1 + r_1 - w_0 = 0$
26	$u_3 + u_9 + g_1 - z_1 = 0$
27	$q_1 - m_1 \geqslant 0$
28	$q_1 - r_1 \geqslant 0$
29	$q_1 - m_1 - r_1 \leqslant 0$
30	$v_2 + q_1 - w_1 = 0$
31	$x_0 + y_0 + w_1 - z_0 = 0$

To ensure the validity of the model, we do some experiments on a small variant of SPECK [1], and we call it *Small-SPECK*. The description of Small-SPECK and the detailed experimental procedure can be found in **Appendix A**. We use MILP-aided bit-based division property to find integral distinguishers for Small-SPECK, and these distinguishers constitute a set which is called $\mathcal{S}_{MILP}$. Then we traverse particular parts of the plaintexts, which are determined by the concrete forms of distinguishers in $\mathcal{S}_{MILP}$, to verify the validity of these already obtained distinguishers. From the experimental results, we ensure that all zero-sum bits found by MILP-aided bit-based division property are indeed zero-sum bits. Thus, the `Modulo` model proposed in this paper is a valid method, and can be applied to construct MILP models for ARX-based structures.

3.2 Modelling the Operation of `AND`/`OR` with a Constant

After observing various primitives, we find that there are cases where a constant (or a key) is involved into the cipher by using logical operation `AND` or `XOR`. For example, the FL function of MISTY1 [7] involves the operations of `AND` and `OR` with some subkeys. Besides, the f_a functions for all KATAN/KTANTAN [4] variants import an irregular constant IR by `AND`ing it with certain internal bit. And the problem of modelling these operations with MILP models has not been concerned. In this section, we will show how to deal with these cases.

In the remaining of this subsection, we suppose that $\boldsymbol{x}$, $\boldsymbol{y}$, and $\boldsymbol{c}$ are n-bit vectors and the division properties of the input and output multi-sets are respectively denoted by $\mathbb{X}$ and $\mathbb{Y}$. Then, for the `AND` and `OR` operations with a constant, we have the following two proposition.

Proposition 1. *Let $\boldsymbol{y} = \boldsymbol{x} \wedge \boldsymbol{c}$. Let $\mathbb{X}$ and $\mathbb{Y}$ be the input multi-set and output multi-set, respectively. Assuming that the multi-set $\mathbb{X}$ has the division property $\mathcal{D}^{1^n}_{\mathbb{K}_\mathbb{X}}$, the division property of the multi-set $\mathbb{Y}$ is $\mathcal{D}^{1^n}_{\mathbb{K}_\mathbb{Y}}$. Then the known-region deduced by $\mathbb{K}_\mathbb{Y}$ covers the known-region deduced by $\mathbb{K}_\mathbb{X}$.*

Proof: We only need to prove that the vectors located in the known-region of $\mathbb{K}_\mathbb{X}$ are also involved in the known-region of $\mathbb{K}_\mathbb{Y}$. Suppose that $\boldsymbol{v} = (v_0, v_1, \ldots, v_{n-1})$ is a vector located in the known-region of $\mathbb{K}_\mathbb{X}$. Thus we have $\bigoplus_{\boldsymbol{x}\in\mathbb{X}} \pi_{\boldsymbol{v}}(\boldsymbol{x}) = 0$. To decide the value of $\bigoplus_{\boldsymbol{y}\in\mathbb{Y}} \pi_{\boldsymbol{v}}(\boldsymbol{y}) = \bigoplus_{\boldsymbol{x}\in\mathbb{X}} \pi_{\boldsymbol{v}}(\boldsymbol{x} \wedge \boldsymbol{c})$, two cases need to be considered:

1. If $\boldsymbol{v} \wedge \boldsymbol{c} = \boldsymbol{v}$, we have

$$\bigoplus_{\boldsymbol{y} \in \mathbb{Y}} \pi_{\boldsymbol{v}}(\boldsymbol{y}) = \bigoplus_{\boldsymbol{x} \in \mathbb{X}} \pi_{\boldsymbol{v}}(\boldsymbol{x} \wedge \boldsymbol{c}) = \bigoplus_{\boldsymbol{x} \in \mathbb{X}} \pi_{\boldsymbol{v}}(\boldsymbol{x}) = 0.$$

2. If $\boldsymbol{v} \succeq \boldsymbol{v} \wedge \boldsymbol{c}$, we know that there is at least one index i such that $v_i = 1$ and $c_i = 0$. Then we know that $\pi_{\boldsymbol{v}}(\boldsymbol{y})$ is always equal to 0 for all $\boldsymbol{y}$, since $y_i = x_i \wedge c_i = x_i \wedge 0 = 0$. Thus

$$\bigoplus_{\boldsymbol{y} \in \mathbb{Y}} \pi_{\boldsymbol{v}}(\boldsymbol{y}) = \bigoplus_{\boldsymbol{x} \in \mathbb{X}} \pi_{\boldsymbol{v}}(\boldsymbol{x} \wedge \boldsymbol{c}) = 0.$$

Now we have proved that the known-region of $\mathbb{K}_{\mathbb{X}}$ is included in the known-region of $\mathbb{K}_{\mathbb{Y}}$. □

Proposition 2. *Let $\boldsymbol{y} = \boldsymbol{x} \vee \boldsymbol{c}$. Let $\mathbb{X}$ and $\mathbb{Y}$ be the input multi-set and output multi-set, respectively. Assuming that the multi-set $\mathbb{X}$ has the division property $\mathcal{D}^{1^n}_{\mathbb{K}_{\mathbb{X}}}$, the division property of the multi-set $\mathbb{Y}$ is $\mathcal{D}^{1^n}_{\mathbb{K}_{\mathbb{Y}}}$. Then the known-region deduced by $\mathbb{K}_{\mathbb{Y}}$ contains the known-region deduced by $\mathbb{K}_{\mathbb{X}}$.*

Proof: The only thing we need to prove is that the vectors which are included in the known-region of $\mathbb{K}_{\mathbb{X}}$ are also included in the known-region of $\mathbb{K}_{\mathbb{Y}}$. Suppose that $\boldsymbol{v}$ is a vector in the known region of $\mathbb{K}_{\mathbb{X}}$. Then by using the definition of division property, we know that $\bigoplus_{\boldsymbol{x} \in \mathbb{X}} \pi_{\boldsymbol{v}}(\boldsymbol{x}) = 0$, and $\bigoplus_{\boldsymbol{x} \in \mathbb{X}} \pi_{\boldsymbol{u}}(\boldsymbol{x}) = 0$ for all $\boldsymbol{u}$ with $\boldsymbol{v} \succeq \boldsymbol{u}$. Two cases should be considered.

1. If $\boldsymbol{v} \wedge \boldsymbol{c} = \boldsymbol{0}$, we have

$$\bigoplus_{\boldsymbol{y} \in \mathbb{Y}} \pi_{\boldsymbol{v}}(\boldsymbol{y}) = \bigoplus_{\boldsymbol{x} \in \mathbb{X}} \pi_{\boldsymbol{v}}(\boldsymbol{x} \vee \boldsymbol{c}) = \bigoplus_{\boldsymbol{x} \in \mathbb{X}} \pi_{\boldsymbol{v}}(\boldsymbol{x}) = 0.$$

2. If $\boldsymbol{v} \wedge \boldsymbol{c} \neq \boldsymbol{0}$, then there is at least one index i such that $v_i = 1$ and $c_i = 1$, and

$$\pi_{\boldsymbol{v}}(\boldsymbol{y}) = \pi_{\boldsymbol{v}}(\boldsymbol{x} \vee \boldsymbol{c}) = \pi_{\boldsymbol{v} \oplus (\boldsymbol{v} \wedge \boldsymbol{c})}(\boldsymbol{x}).$$

Since $\boldsymbol{v} \succeq \boldsymbol{v} \oplus (\boldsymbol{v} \wedge \boldsymbol{c})$, we have

$$\bigoplus_{\boldsymbol{y} \in \mathbb{Y}} \pi_{\boldsymbol{v}}(\boldsymbol{y}) = \bigoplus_{\boldsymbol{x} \in \mathbb{X}} \pi_{\boldsymbol{v}}(\boldsymbol{x} \vee \boldsymbol{c}) = \bigoplus_{\boldsymbol{x} \in \mathbb{X}} \pi_{\boldsymbol{v} \oplus (\boldsymbol{v} \wedge \boldsymbol{c})}(\boldsymbol{x}) = 0.$$

Thus, we prove that $\boldsymbol{v}$ is also included in the known-region of $\mathbb{K}_{\mathbb{Y}}$. □

From **Proposition 1** and **Proposition 2**, we know that the known-region deduced by $\mathbb{K}_{\mathbb{Y}}$ always contains the known-region deduced by $\mathbb{K}_{\mathbb{X}}$ for the operation `AND` with a constant as well as the operation `OR` with a constant. And this conclusion has nothing to do with the concrete value of the constant $\boldsymbol{c}$. Thus, we can just ignore the operation of `AND`/`OR` with a constant when we encounter it. Although doing this may lost some useful information when we propagate the division property, sometimes we can not know such constants in advance which may be some subkeys. As a result, dealing with it in this way is reasonable.

3.3 Modelling the Operation of `Modulo` with a Constant

In **Section 3.1**, we consider the case where the two input branches of the `Modulo` operation both are variable. But if we look more closely, we will find that there is a case where only one input branch of the `Modulo` operation is variable, while the other one is constant (or subkey). For example, the round functions of HIGHT [6] and TEA [16] contain this kind of operation. In this subsection, we will consider how to build MILP model for this operation.

Again, we will consider the iterated expression of the `Modulo` operation, and take a 4-bit `Modulo` operation as an illustration. Suppose $\boldsymbol{x} = (x_0, x_1, x_2, x_3)$ and $\boldsymbol{k} = (k_0, k_1, k_2, k_3)$ are two input branches, where $\boldsymbol{x}$ represent the variable and $\boldsymbol{k}$ is the constant branch. Let $\boldsymbol{z} = (z_0, z_1, z_2, z_3)$ be the output branch. The iterated expression of this operation is provided in (3).

$$\begin{cases} z_3 = x_3 \oplus k_3 \\ z_2 = x_2 \oplus k_2 \oplus c_2,\ c_2 = x_3 k_3 \\ z_1 = x_1 \oplus k_1 \oplus c_1,\ c_1 = x_2 k_2 \oplus (x_2 \oplus k_2) c_2 \\ z_0 = x_0 \oplus k_0 \oplus c_0,\ c_0 = x_1 k_1 \oplus (x_1 \oplus k_1) c_1 \end{cases} \tag{3}$$

Note that all the operations involving k_i's are `XOR` and `AND`. It is already known to us that `XOR`ing with a constant does not influence the division property, and **Proposition 1** indicates that the operation of `AND` with a constant can be ignored when we trace the bit-based division property. Thus (3) can be transformed into a much simpler look, which can be found in **Table 4**. In **Table 4**, we wipe all those operations comprising k_i's, and it is sufficient to construct MILP model for these reduced expressions. Again, some auxiliary variables need to be involved to describe the propagation of bit-based division property, and $a_0 \sim a_7$, $b_0 \sim b_2$, $f_0 \sim f_1$, $d_0 \sim d_1$, and $e_0 \sim e_1$ are introduced for this reason. The allocation of variables for 4-bit `Modulo` operation with a constant also given in **Table 4**. A similar analysis like the one in **Section 3.1** gives the linear inequality system used to describe the bit-based division property propagation for `Modulo` operation with a constant, which can be found in **Table 5**.

Table 4: Allocation of Intermediate Variables for 4-bit `Modulo` Operation with a Constant.

Bit	Variable Allocation	Bit	Variable Allocation
z_3	$\underbrace{x_3}_{a_0}$		
z_2	$\underbrace{x_2}_{a_2} \oplus \underbrace{c_2}_{f_0}$	$\underbrace{c_2}_{b_0}$	$\underbrace{x_3}_{a_1}$
z_1	$\underbrace{x_1}_{a_5} \oplus \underbrace{c_1}_{f_1}$	$\underbrace{c_1}_{b_1}$	$\underbrace{x_2}_{a_3} \oplus \overbrace{\underbrace{x_2}_{a_4}\ \underbrace{c_2}_{d_0}}^{e_0}$
Continued on next page			

Table 4 – continued from previous page

Bit	Variable Allocation	Bit	Variable Allocation
z_0	$x_0 \oplus c_0$	$\underbrace{c_0}_{b_2}$	$\underbrace{x_1}_{a_6} \oplus \overbrace{\underbrace{x_1}_{a_7}\ \underbrace{c_1}_{d_1}}^{e_1}$

Table 5: Linear Inequality System for 4-bit `Modulo` Operation with a Constant.

Label	Linear Inequalities Used to Describe 4-Bit `Modulo`
1	$x_3 - a_0 - a_1 = 0$
2	$x_2 - a_2 - a_3 - a_4 = 0$
3	$x_1 - a_5 - a_6 - a_7 = 0$
4	$z_3 - a_0 = 0$
5	$b_0 - a_1 = 0$
6	$b_0 - f_0 - d_0 = 0$
7	$z_2 - a_2 - f_0 = 0$
8	$e_0 - a_4 \geqslant 0$
9	$e_0 - d_0 \geqslant 0$
10	$e_0 - a_4 - d_0 \leqslant 0$
11	$b_1 - a_3 - e_0 = 0$
12	$b_1 - f_1 - d_1 = 0$
13	$z_1 - a_5 - f_1 = 0$
14	$e_1 - a_7 \geqslant 0$
15	$e_1 - d_1 \geqslant 0$
16	$e_1 - a_7 - d_1 \leqslant 0$
17	$b_2 - a_6 - e_1 = 0$
18	$z_0 - x_0 - b_2 = 0$

Thus, with the methods explained in **Section 3.1**, **Section 3.2** and **Section 3.3**, we are sufficient to construct MILP models for some ARX-based primitives, and these models can help us search integral distinguishers with MILP optimizer. The applications of MILP-aided bit-based division property are provided in **Section 4**.

4 Applications of MILP-Aided Bit-Based Division Property for Some ARX-Based Block Ciphers

With the methods proposed in **Section 3**, we are able to construct MILP models for almost all ARX-based primitives. As an illustration, we apply MILP-aided bit-based division property to search integral distinguishers for HIGHT, LEA, TEA/XTEA, and KATAN/KTANTAN in this section.

4.1 MILP-Aided Bit-Based Division Property for HIGHT

HIGHT [6] HIGHT is an ARX-based block cipher with 64-bit block length and 128-bit key length. The 64-bit plaintext and ciphertext are considered as concatenations of 8 bytes and denote by $P = P_7\|P_6\|\cdots\|P_0$ and $C = C_7\|C_6\|\cdots\|C_0$, respectively. The 64-bit intermediate values are analogously represented as $X_i = X_{i,7}\|X_{i,6}\|\cdots\|X_{i,0}$ for $i = 0, 1, \ldots, 32$. The 128-bit master key is considered as a concatenation of 16 bytes and is denoted by $MK = MK_{15}\|MK_{14}\|\cdots\|MK_0$. The key schedule of HIGHT consists of two algorithms, `WhiteningKeyGeneration` which generates 8 whitening key bytes $WK_0, WK_1, \ldots, WK_7$, and the function called `SubkeyGeneration` generates 128 subkey bytes $SK_0, SK_1, \ldots, SK_{127}$. Each round function uses 4 subkey bytes $SK_{4i}, SK_{4i+1}, SK_{4i+2}, SK_{4i+3}$. It is important to note that the distinguishers we found in this paper for HIGHT does not include the initial transformation and the final transformation which load the whitening-key into the cipher. The basic structure of one round HIGHT encryption is depicted in **Fig. 1**.

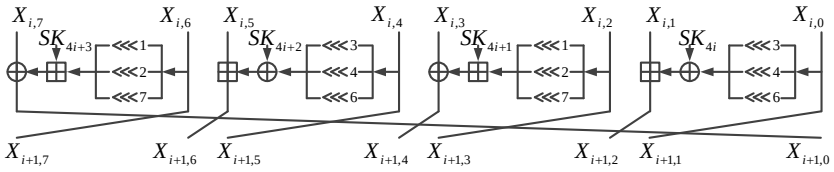


Fig. 1: Round Function of HIGHT.

Former Results The designers [6] showed that there exist 12-round integral distinguishers with data complexity 2^8. But Zhang *et al.* [18] pointed out and corrected an error in the 12-round distinguishers, and showed that the former 12-round distinguisher was only a 11-round one. They also proposed some 12-round and 17-round distinguishers.

Applying MILP-Aided Bit-Based Division Property to HIGHT By applying the `Modulo` model introduced in **Section 3.1** and the model for the operation of `Modulo` with a constant proposed in **Section 3.3**, we construct the MILP model used to depict the bit-based division property propagation of HIGHT. We put different initial division properties into the MILP model and the results can be found in **Table 6**. The specific looks of the distinguishers are provided in **Appendix B.1**. The 11-round, 12-round, and 17-round distinguishers given in **Table 6** are respectively same to those provided by Zhang *et al.* [18]. Besides, we obtain two 18-round distinguishers with data complexity 2^{63}. These 18-round distinguishers achieve one more round than the best ones proposed by Zhang *et al.* [18].

Table 6: Summarization of Experimental Results for HIGHT.

Division Property	**Round**	***Obj***	#{**Unit Vector**}
$\mathcal{D}^{1^{64}}_{\{\texttt{[ff000000,00000000]}\}}$	11	1	63
	12	1	64
$\mathcal{D}^{1^{64}}_{\{\texttt{[00000000,ff000000]}\}}$	11	1	63
	12	1	64
$\mathcal{D}^{1^{64}}_{\{\texttt{[ffff0000,00000000]}\}}$	12	1	63
	13	1	64
$\mathcal{D}^{1^{64}}_{\{\texttt{[00000000,ffff0000]}\}}$	12	1	63
	13	1	64
$\mathcal{D}^{1^{64}}_{\{\texttt{[ffffffff,ffffff00]}\}}$	17	1	63
	18	1	64
$\mathcal{D}^{1^{64}}_{\{\texttt{[ffffff00,ffffffff]}\}}$	17	1	63
	18	1	64
$\mathcal{D}^{1^{64}}_{\{\texttt{[ffffffff,fffffffe]}\}}$	18	1	63
	19	1	64
$\mathcal{D}^{1^{64}}_{\{\texttt{[fffffffe,ffffffff]}\}}$	18	1	63
	19	1	64

Division Property: The initial division property.
#{**Unit Vector**}: The number of unit vectors in the resulting division property.

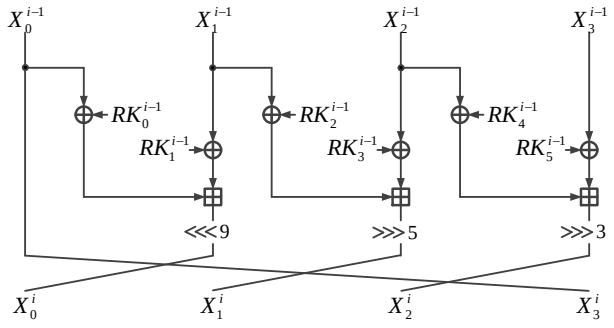


Fig. 2: Round Function of LEA.

4.2 MILP-Aided Bit-Based Division Property for LEA

LEA [5] LEA is an ARX-based block cipher designed by Hong *et al.* and provides a high speed software encryption on general-purpose processors. The block size is 128 bits and the key size can take 128, 192, or 256 bits. The number of rounds r is 24 for 128-bit keys, 28 for 192-bit keys, and 32 for 256-bit keys. The encryption of LEA maps a plaintext of four 32-bit words $X^0 = (X_0^0, X_1^0, X_2^0, X_3^0)$ to a ciphertext $X^r = (X_0^r, X_1^r, X_2^r, X_3^r)$. The round function for LEA is illustrated in **Fig. 2**, where RK_0^{i-1}, RK_1^{i-1}, RK_2^{i-1}, RK_3^{i-1}, RK_4^{i-1}, and RK_5^{i-1} are subkeys used in the i-th round.

Since `XOR`ing with a constant does not affect the propagation of division property, we do not consider the operation of `XOR` with subkeys. The key schedule is left behind. For more information about LEA, please refer to [5].

Former Results A 6-round integral distinguisher is proposed in [2], whose data complexity is 2^{32}.

Applying MILP-Aided Bit-Based Division Property to LEA We integrate the `Modulo` model introduced in **Section 3.1** into the construction of MILP model used to depict bit-based division property propagation for LEA. Different initial division properties are loaded into the MILP model and the experimental results can be found in **Table 7**. **Table 7** indicates that there is a 7-round distinguisher with data complexity 2^{96}, which attains one more round than the one provided in [2]. The specific expressions of these newly found distinguishers are given in **Appendix B.2**.

Table 7: Summarization of Experimental Results for LEA.

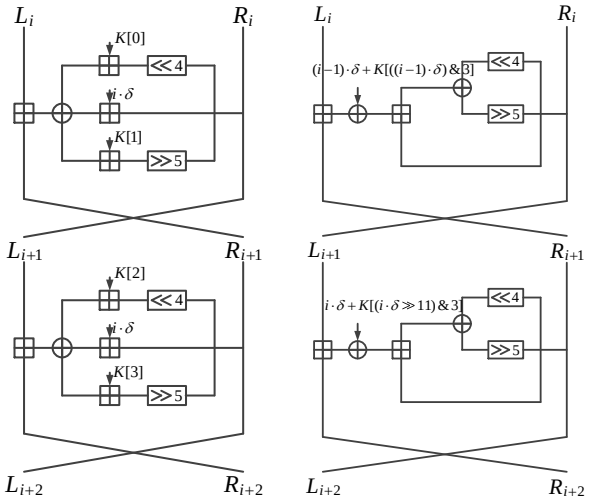
Division Property	**Round**	***Obj***	**#{Unit Vector}**
$\mathcal{D}^{1^{128}}_{\{\texttt{[ffffffff,00000000,00000000,00000000]}\}}$	6	1	127
	7	1	128
$\mathcal{D}^{1^{128}}_{\{\texttt{[00000000,00000000,00000000,ffffffff]}\}}$	6	1	126
	7	1	128
$\mathcal{D}^{1^{128}}_{\{\texttt{[00000000,ffffffff,ffffffff,ffffffff]}\}}$	7	1	127
	8	1	128

Division Property: The initial division property.
#{**Unit Vector**}: The number of unit vectors in the resulting division property.

4.3 MILP-Aided Bit-Based Division Property for TEA and XTEA

TEA [16] and XTEA [8] TEA and XTEA both are 64-round Feistel iterated block ciphers with 64-bit block and 128-bit key which consists of four 32-bit

words $K[0]$, $K[1]$, $K[2]$, and $K[3]$. TEA does not have any key schedule and the key words are used directly in round functions. For XTEA, the derivation of the subkey word number is slightly more complex. The round constant is derived from the constant $\delta = \texttt{9e3779b9}$ and the round number. Denote the plaintext by $P = (P_L, P_R)$, the ciphertext by $C = (C_L, C_R)$, and the input of the i-th round by (L_i, R_i). The round functions of two contiguous rounds for TEA and XTEA are illustrated in **Fig. 3(a)** and **Fig. 3(b)**, respectively.



(a) TEA. (b) XTEA.

Fig. 3: Illustrations of TEA and XTEA.

Former Results At FSE 2012, Bogdanov and Wang [3] proposed a 15-round zero-correlation linear approximation for TEA/XTEA. By utilizing the equivalence between zero-correlation linear cryptanalysis and integral cryptanalysis for ARX-based block ciphers proposed by Wen and Wang [15], this approximation can be equivalently transformed into a 15-round integral distinguisher for TEA/XTEA with data complexity 2^{63}.

Applying MILP-Aided Bit-Based Division Property to TEA and XTEA For XTEA, directly applying the `Modulo` model introduced in **Section 3.1**, we obtain the MILP model used to depict its bit-based division property propagation. While for TEA, we note that there are `Modulo` operations with constants. In the meanwhile, since some `Modulo` operations in TEA's round function will interact with left shift and right shift operations, the MILP models of these combinatory operations involving `Modulo` need to be slightly adjusted.

Considering the Combination of Left Shift Operation and `Modulo` Operation Let the input and the output of **Fig. 4(a)** be $\boldsymbol{x} = (x_0, x_1, \ldots, x_{31})$ and $\boldsymbol{z} = (z_0, z_1, \ldots, z_{31})$, respectively. Then $\boldsymbol{z} = (\boldsymbol{x} \ll 4) \boxplus \boldsymbol{K}$. More specifically, we have

$$(z_0, z_1, \ldots, z_{31}) = (x_0, x_1, \ldots, x_{27}, \underbrace{0, \ldots, 0}_{4}) \boxplus (K_0, K_1, \ldots, K_{31}).$$

Combining the fact with the iterated expression of 32-bit `Modulo` operation, we know that the output bits z_i's and the input bits x_i's satisfy the relations listed in (4). From (4), we find that the output $\boldsymbol{z}$ can be treat as the concatenation of two parts, *i.e.*, $\boldsymbol{z} = (\boldsymbol{x}' \boxplus \boldsymbol{K}_1) \| \boldsymbol{K}_2$, where $\boldsymbol{x}' = (x_0, x_1, \ldots, x_{27})$, $\boldsymbol{K}_1 = (K_0, K_1, \ldots, K_{27})$, $\boldsymbol{K}_2 = (K_{28}, K_{29}, K_{30}, K_{31})$, and $\boldsymbol{K} = \boldsymbol{K}_1 \| \boldsymbol{K}_2$. Thus, the original structure in TEA's round function can be equivalently transformed into the structure illustrated in **Fig. 4(b)**. The functions used in **Fig. 4(b)** are explained as follows.

- *Truncated* is a function and its output is equal to $(x_0, x_1, \ldots, x_{27})$ if the input is $(x_0, x_1, \ldots, x_{31})$.
- *Extended* is a function taking two branches of inputs, which are respectively denoted as $(x_0, x_1, \ldots, x_{27})$ and (k_0, k_1, k_2, k_3), and output is equal to

$$(x_0, x_1, \ldots, x_{27}, k_0, k_1, k_2, k_3).$$

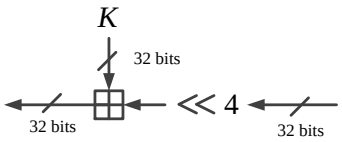
After transformation, the MILP model can be constructed in an general way.

$$\begin{cases} z_{31} = K_{31} \\ z_{30} = K_{30} \\ z_{29} = K_{29} \\ z_{28} = K_{28} \\ z_{27} = x_{27} \oplus K_{27} \\ z_{26} = x_{26} \oplus K_{26} \oplus c_5, \; c_5 = x_{27} K_{27} \\ z_{25} = x_{25} \oplus K_{25} \oplus c_6, \; c_6 = x_{26} K_{26} \oplus (x_{26} \oplus K_{26}) c_5 \\ z_{24} = x_{24} \oplus K_{24} \oplus c_7, \; c_7 = x_{25} K_{25} \oplus (x_{25} \oplus K_{25}) c_6 \\ \qquad\qquad\qquad\qquad \cdots \\ z_1 = x_1 \oplus K_1 \oplus c_{30}, \; c_{30} = x_2 K_2 \oplus (x_2 \oplus K_2) c_{29} \\ z_0 = x_0 \oplus K_0 \oplus c_{31}, \; c_{31} = x_1 K_1 \oplus (x_1 \oplus K_1) c_{30} \end{cases} \tag{4}$$

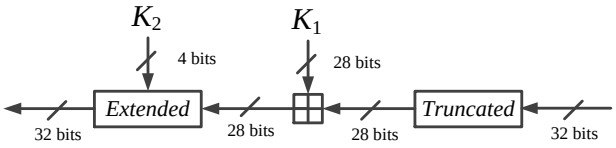
Considering the Combination of Right Shift Operation and `Modulo` Operation Let the input and output of **Fig. 5(a)** be $\boldsymbol{x} = (x_0, x_1, \ldots, x_{31})$ and $\boldsymbol{z} = (z_0, z_1, \ldots, z_{31})$, respectively. Then $\boldsymbol{z} = (\boldsymbol{x} \gg 5) \boxplus \boldsymbol{K}$, *i.e.*,

$$(z_0, z_1, \ldots, z_{31}) = (\underbrace{0, \ldots, 0}_{5}, x_0, x_1, \ldots, x_{26}) \boxplus (K_0, K_1, \ldots, K_{31}).$$

Iterating these variables into the iterated expression of 32-bit `Modulo` operation, we obtain the relations that the input bits x_i's and the output bits z_i's should



(a) The Original Structure.



(b) The Equivalently Transformed Structure.

Fig. 4: Illustrations of Equivalent Transformation for Left Shift.

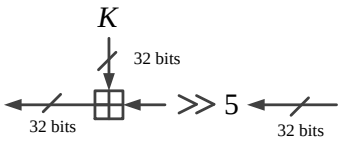
follow, which are provided in (5).

$$\begin{cases} z_{31} = x_{26} \oplus K_{31} & \\ z_{30} = x_{25} \oplus K_{30} \oplus c_1, \ c_1 & = x_{26}K_{31} \\ z_{29} = x_{24} \oplus K_{29} \oplus c_2, \ c_2 & = x_{25}K_{30} \oplus (x_{25} \oplus K_{30})c_1 \\ z_{28} = x_{23} \oplus K_{28} \oplus c_3, \ c_3 & = x_{24}K_{29} \oplus (x_{24} \oplus K_{29})c_2 \\ & \cdots \\ z_5 = x_0 \oplus K_5 \oplus c_{26}, & c_{26} = x_1K_6 \oplus (x_1 \oplus K_6)c_{25} \\ z_4 = K_4 \oplus c_{27}, & c_{27} = K_5c_{26} \\ z_3 = K_3 \oplus c_{28}, & c_{28} = K_4c_{27} \\ z_2 = K_2 \oplus c_{29}, & c_{29} = K_3c_{28} \\ z_1 = K_1 \oplus c_{30}, & c_{30} = K_2c_{29} \\ z_0 = K_0 \oplus c_{31}, & c_{31} = K_1c_{30} \end{cases} \tag{5}$$

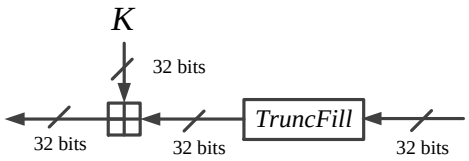
With (5), we can construct MILP model in the usual fashion. Also note that the original structure shown in **Fig. 5(a)** can be changed into an equivalent form, which can be found in **Fig. 5(b)**. The function named $TrunFill$ operates as follows.

$$(\underbrace{0, \ldots, 0}_{5}, x_0, x_1, \ldots, x_{26}) = TrunFill(x_0, x_1, \ldots, x_{31}).$$

Up to now, we also can build MILP model for TEA and search its integral distinguishers. We put different initial division properties into the MILP models of TEA as well as XTEA, and the results are shown in **Table 8**. The longest distinguishers for TEA and XTEA all achieve 15 rounds. Comparing with the former result, the distinguisher with data complexity 2^{62} obtains exactly the same



(a) The Original Structure.



(b) The Equivalently Transformed Structure.

Fig. 5: Illustrations of Equivalent Transformation for Right Shift.

zero-sum bit while reducing the data complexity by half. At the same time, the newly obtained distinguisher with data complexity 2^{63} gains one more zero-sum bit than the previously proposed one. The concrete expressions of these newly obtained distinguishers can be found in **Appendix B.3** and **Appendix B.4**.

Table 8: Summarization of Experimental Results for TEA and XTEA.

Cipher	**Division Property**	**Round**	***Obj***	**#{Unit Vector}**
TEA/XTEA	$\mathcal{D}^{1^{64}}_{\{\texttt{[ffffffff,fffffffc]}\}}$	15	1	63
		16	1	64
	$\mathcal{D}^{1^{64}}_{\{\texttt{[ffffffff,fffffffe]}\}}$	15	1	62
		16	1	64

Division Property: The initial division property.
#{Unit Vector}: The number of unit vectors in the resulting division property.

4.4 MILP-Aided Bit-Based Division Property for KATAN and KTANTAN

KATAN and KTANTAN [4] KATAN and KTANTAN are two families of hardware oriented block ciphers and both have three variants each, of 32-bit, 48-bit, 64-bit block. For all versions of KATAN and KTANTAN ciphers, key size is 80-bit. The only difference between KATAN and KTANTAN is the key schedule.

The integral structure of KATAN/KTANTAN consists of two LFSR's, called L_1 and L_2, loaded with the plaintext and then transformed by two nonlinear Boolean functions, f_a and f_b as follows.

$$f_a(L_1) = L_1[x_1] \oplus L_1[x_2] \oplus (L_1[x_3] \wedge L_1[x_4]) \oplus (L_1[x_5] \wedge IR) \oplus k_a$$
$$f_b(L_2) = L_2[y_1] \oplus L_2[y_2] \oplus (L_2[y_3] \wedge L_2[y_4]) \oplus (L_2[y_5] \wedge L_2[y_6]) \oplus k_b$$

For KATAN/KTANTAN48, in one round of the cipher the functions f_a and f_b are applied twice. The first pair of f_a and f_b is applied, and then after the update of the registers, they are applied again, using the same subkeys. In KATAN/KTANTAN64, each round applies f_a and f_b three times (again, with the same key bits). IR used in f_a represents irregular update rule, *i.e.*, $L_1[x_5]$ is `XOR`ed in the rounds where the irregular update is used. IR's are the output of an LFSR.

Since the subkey is involved in the ciphers by the `XOR` operation and `XOR`ing with constants does not affect the propagation of division property, we do not introduce the key schedules of KATAN and KTANTAN here. Note that this fact also indicates that the integral distinguishers found by using bit-based division property hold not only for KATAN but also for KTANTAN. Also by the **Proposition 1**, we can ignore the operation of `AND` with a constant. So the definition of IR is also left behind. For more details of KATAN/KTANTAN, please refer to [4].

Applying MILP-Aided Bit-Based Division Property to KATAN and KTANTAN We construct the MILP models and analyze the integral properties for different versions of KATAN/KTANTAN. Different initial division properties are inserted into the MILP model and the experimental results are listed in **Table 9**, which shows that the longest distinguishers for KATAN/KTANTAN32, 48, and 64 are 99-round, 83.5-round, and 72.3-round, respectively. The concrete forms of these distinguishers are provided in **Appendix B.5**.

5 Conclusion

In this paper, we focus on the feasibility of MILP-aided bit-based division property for ARX-based block cipher. More specifically, we concentrate on the construction of MILP models for some components of ARX-based structure. To propagate the bit-based division property of the `Modulo` operation, we use its iterated expression. After introducing some auxiliary variables and allocating these variables according to the iterated expression, the `Modulo` model is constructed by successively invoking **Copy**, `AND`, and `XOR` models. Then, the operations of `AND` and `OR` with a constant are considered. For both of these two operations, we prove that the known-region deduced by the input division property is always included in the known-region derived from the output division property, with which these operations can be ignored during our analysis. Furthermore, we also handle the `Modulo` operation with a constant. With these newly constructed

Table 9: Summarization of Experimental Results for KATAN/KTANTAN.

Cipher	Division Property	Round	*Obj*	#{Unit Vector}
KATAN/KTANTAN32	$\mathcal{D}^{1^{32}}_{\{\texttt{[3fffffff]}\}}$	88	1	29
		89		30
		90		31
		91		32
	$\mathcal{D}^{1^{32}}_{\{\texttt{[7fffffff]}\}}$	97	1	29
		98		30
		99		31
		100		32
KATAN/KTANTAN48†	$\mathcal{D}^{1^{48}}_{\{\texttt{[3fffff,ffffff]}\}}$	76	1	45
		76.5		46
		77		47
		77.5		48
	$\mathcal{D}^{1^{48}}_{\{\texttt{[7fffff,ffffff]}\}}$	82.5	1	45
		83		46
		83.5		47
		84		48
KATAN/KTANTAN64†	$\mathcal{D}^{1^{64}}_{\{\texttt{[3fffffff,ffffffff]}\}}$	66.6	1	61
		67		62
		67.3		63
		67.6		64
	$\mathcal{D}^{1^{64}}_{\{\texttt{[7fffffff,ffffffff]}\}}$	71.6	1	61
		72		62
		72.3		63
		72.6		64

Division Property: The initial division property.
#{**Unit Vector**}: The number of unit vectors in the resulting division property.

† Note that one round of KATAN/KTANTAN32 and KATAN/KTANTAN64 apply the functions f_a and f_b twice and thrice, respectively. 0.5-round for KATAN/KTANTAN48 indicates that f_a and f_b are used once. 0.3-round and 0.6-round for KATAN/KTANTAN64 can be explained similarly.

models, we solve the feasibility of MILP-aided bit-based division property for ARX-based block cipher. As a result, these methods are applied to search integral distinguishers for various ARX-based block ciphers. Two 18-round integral distinguishers for HIGHT are obtained, which achieve one more round than the previous best result. As to LEA, we find a 7-round integral distinguisher, which also gains one more round than the one proposed by its designer. Some 15-round integral distinguishers for TEA and XTEA are presented. Comparing with the one transformed from the 15-round zero-correlation linear approximation, our newly obtained distinguishers either reduces the data requirement or increases the number of zero-sum bits. Besides, the bit-based division properties for KATAN and KTANTAN families of block ciphers are also provided.

References

1. Ray Beaulieu, Douglas Shors, Jason Smith, Stefan Treatman-Clark, Bryan Weeks, and Louis 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.
2. A. Bogdanov, K. Varici, N. Mouha, V. Velichkov, E. Tischhauser, M. Wang, D. Toz, Q. Wang, and V. Rijmen. Security evaluation of the block cipher lea. Technical report, Technical report, July, 2011.
3. Andrey Bogdanov and Meiqin Wang. Zero correlation linear cryptanalysis with reduced data complexity. In *Fast Software Encryption - 19th International Workshop, FSE 2012, Washington, DC, USA, March 19-21, 2012. Revised Selected Papers*, pages 29–48, 2012.
4. Christophe De Cannière, Orr Dunkelman, and Miroslav Knezevic. KATAN and KTANTAN - A family of small and efficient hardware-oriented block ciphers. In *Cryptographic Hardware and Embedded Systems - CHES 2009, 11th International Workshop, Lausanne, Switzerland, September 6-9, 2009, Proceedings*, pages 272–288, 2009.
5. Deukjo Hong, Jung-Keun Lee, Dong-Chan Kim, Daesung Kwon, Kwon Ho Ryu, and Donggeon Lee. LEA: A 128-bit block cipher for fast encryption on common processors. In *Information Security Applications - 14th International Workshop, WISA 2013, Jeju Island, Korea, August 19-21, 2013, Revised Selected Papers*, pages 3–27, 2013.
6. Deukjo Hong, Jaechul Sung, Seokhie Hong, Jongin Lim, Sangjin Lee, Bonseok Koo, Changhoon Lee, Donghoon Chang, Jaesang Lee, Kitae Jeong, Hyun Kim, Jongsung Kim, and Seongtaek Chee. HIGHT: A new block cipher suitable for low-resource device. In *Cryptographic Hardware and Embedded Systems - CHES 2006, 8th International Workshop, Yokohama, Japan, October 10-13, 2006, Proceedings*, pages 46–59, 2006.
7. Mitsuru 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.
8. Roger M Needham and David J Wheeler. Tea extensions. *Report, Cambridge University, Cambridge, UK (October 1997)*, 1997.
9. Ling Sun and Meiqin Wang. Towards a further understanding of bit-based division property. *IACR Cryptology ePrint Archive*, 2016:392, 2016.
10. Ling Sun, Wei Wang, and Meiqin Wang. Milp-aided bit-based division property for primitives with non-bit-permutation linear layers. *IACR Cryptology ePrint Archive*, 2016:811, 2016.
11. Yosuke 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.
12. Yosuke 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.
13. Yosuke Todo and Masakatu 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.

14. Qingju Wang, Zhiqiang Liu, Kerem Varici, Yu Sasaki, Vincent Rijmen, and Yosuke Todo. Cryptanalysis of reduced-round SIMON32 and SIMON48. In *Progress in Cryptology - INDOCRYPT 2014 - 15th International Conference on Cryptology in India, New Delhi, India, December 14-17, 2014, Proceedings*, pages 143–160, 2014.
15. Long Wen and Meiqin Wang. Integral zero-correlation distinguisher for ARX block cipher, with application to SHACAL-2. In *Information Security and Privacy - 19th Australasian Conference, ACISP 2014, Wollongong, NSW, Australia, July 7-9, 2014. Proceedings*, pages 454–461, 2014.
16. David J. Wheeler and Roger M. Needham. Tea, a tiny encryption algorithm. In *Fast Software Encryption: Second International Workshop. Leuven, Belgium, 14-16 December 1994, Proceedings*, pages 363–366, 1994.
17. Zejun Xiang, Wentao Zhang, Zhenzhen Bao, and Dongdai Lin. Applying MILP method to searching integral distinguishers based on division property for 6 lightweight block ciphers. *IACR Cryptology ePrint Archive*, 2016:857, 2016.
18. Peng Zhang, Bing Sun, and Chao Li. Saturation attack on the block cipher HIGHT. In *Cryptology and Network Security, 8th International Conference, CANS 2009, Kanazawa, Japan, December 12-14, 2009. Proceedings*, pages 76–86, 2009.

A Experiments on Small-SPECK

To verify the validity of our newly proposed model for the `Modulo` operation. We do some experiments on a toy example called *Small-SPECK*.

A.1 A Brief Introduction of Small-SPECK

Small-SPECK is an ARX-based block cipher. The block size of Small-SPECK is 8-bit. The subkeys used in Small-SPECK are randomly generated. The round function of Small-SPECK is illustrated in **Fig. 6**, where (X_L^i, X_R^i) is the input of the i-th round, k_i is the subkey used in the i-th round.

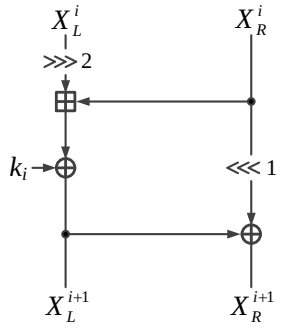


Fig. 6: The Round Function of Small-SPECK.

A.2 Experimental Procedures and Experimental Results

The experimental procedures are listed below.

1. For different initial division properties, we first use MILP-aided bit-based division property to find zero-sum bits of each round.
2. We randomly generate 2^{10} subkeys and exhaustively search zero-sum bits just like what have done for SIMON in [14].
3. By comparison, we check whether the zero-sum bits found by using MILP-aided bit-based division property are indeed zero-sum bits or not.

The comparison of zero-sum bits found by MILP-aided bit-based division property and those found by exhaustively searching under different input multi-sets is shown in **Table 10**. From **Table 10**, we know that the zero-sum bits found by MILP-aided bit-based division property indeed satisfy zero-sum property. And the `Modulo` model proposed in this paper is a valid method to help us to construct integral distinguishers for ARX-based block ciphers.

B Summarization of Integral Distinguishers

In this section, we will adopt the following symbols to present the newly obtained integral distinguishers. '$\mathcal{A}^i$' represents an i-bit vector with every bit active. '$\mathcal{B}^i$' denotes an i-bit vector with every bit satisfying zero-sum property. '$\mathcal{C}^i$' indicates an i-bit vector with every bit being constant. '$\mathcal{U}^i$' means an i-bit vector and the properties of its internal bits are unknown.

B.1 Integral Distinguishers for HIGHT

11-*Round Integral Distinguisher with Data Complexity* 2^8

$$(\mathcal{A}^8,\mathcal{C}^8,\mathcal{C}^8,\mathcal{C}^8,\mathcal{C}^8,\mathcal{C}^8,\mathcal{C}^8,\mathcal{C}^8) \xrightarrow{11\text{ Rounds}} (\mathcal{U}^8,\mathcal{U}^8,\mathcal{U}^8,\mathcal{U}^8,\mathcal{U}^7\mathcal{B}^1,\mathcal{U}^8,\mathcal{U}^8,\mathcal{U}^8)$$

11-*Round Integral Distinguisher with Data Complexity* 2^8

$$(\mathcal{C}^8,\mathcal{C}^8,\mathcal{C}^8,\mathcal{C}^8,\mathcal{A}^8,\mathcal{C}^8,\mathcal{C}^8,\mathcal{C}^8) \xrightarrow{11\text{ Rounds}} (\mathcal{U}^7\mathcal{B}^1,\mathcal{U}^8,\mathcal{U}^8,\mathcal{U}^8,\mathcal{U}^8,\mathcal{U}^8,\mathcal{U}^8,\mathcal{U}^8)$$

12-*Round Integral Distinguisher with Data Complexity* 2^{16}

$$(\mathcal{A}^8,\mathcal{A}^8,\mathcal{C}^8,\mathcal{C}^8,\mathcal{C}^8,\mathcal{C}^8,\mathcal{C}^8,\mathcal{C}^8) \xrightarrow{12\text{ Rounds}} (\mathcal{U}^8,\mathcal{U}^8,\mathcal{U}^8,\mathcal{U}^8,\mathcal{U}^7\mathcal{B}^1,\mathcal{U}^8,\mathcal{U}^8,\mathcal{U}^8)$$

12-*Round Integral Distinguisher with Data Complexity* 2^{16}

$$(\mathcal{C}^8,\mathcal{C}^8,\mathcal{C}^8,\mathcal{C}^8,\mathcal{A}^8,\mathcal{A}^8,\mathcal{C}^8,\mathcal{C}^8) \xrightarrow{12\text{ Rounds}} (\mathcal{U}^7\mathcal{B}^1,\mathcal{U}^8,\mathcal{U}^8,\mathcal{U}^8,\mathcal{U}^8,\mathcal{U}^8,\mathcal{U}^8,\mathcal{U}^8)$$

17-*Round Integral Distinguisher with Data Complexity* 2^{56}

$$(\mathcal{A}^8,\mathcal{A}^8,\mathcal{A}^8,\mathcal{A}^8,\mathcal{A}^8,\mathcal{A}^8,\mathcal{A}^8,\mathcal{C}^8) \xrightarrow{17\text{ Rounds}} (\mathcal{U}^8,\mathcal{U}^8,\mathcal{U}^8,\mathcal{U}^8,\mathcal{U}^7\mathcal{B}^1,\mathcal{U}^8,\mathcal{U}^8,\mathcal{U}^8)$$

Table 10: Comparison of Zero-Sum Bits Found by Using Two Methods for Small-SPECK

Input	Round	MILP-Aided Bit-Based Theory			Exhaustively Searching	
		Obj	#{Bits}	Bits	#{Bits}	Bits
$\mathcal{D}^{1^8}_{\{[01111111]\}}$	1	6	8	$0 \sim 7$	8	$0 \sim 7$
	2	3	8	$0 \sim 7$	8	$0 \sim 7$
	3	1	6	$1 \sim 3,\ 5 \sim 7$	6	$1 \sim 3,\ 5 \sim 7$
	4	1	1	3	1	3
	5	1	0	–	0	–
$\mathcal{D}^{1^8}_{\{[11111110]\}}$	1	7	8	$0 \sim 7$	8	$0 \sim 7$
	2	4	8	$0 \sim 7$	8	$0 \sim 7$
	3	1	6	$1 \sim 3,\ 5 \sim 7$	6	$1 \sim 3,\ 5 \sim 7$
	4	1	1	3	3	$2 \sim 3, 6$
	5	1	0	–	0	–
$\mathcal{D}^{1^8}_{\{[00001111]\}}$	1	2	8	$0 \sim 7$	8	$0 \sim 7$
	2	1	4	$2 \sim 3, 6 \sim 7$	4	$2 \sim 3, 6 \sim 7$
	3	1	0	–	0	–
$\mathcal{D}^{1^8}_{\{[11110000]\}}$	1	4	8	$0 \sim 7$	8	$0 \sim 7$
	2	2	8	$0 \sim 7$	8	$0 \sim 7$
	3	1	2	$3, 7$	2	$3, 7$
	4	1	0	–	0	–
$\mathcal{D}^{1^8}_{\{[11111000]\}}$	1	5	8	$0 \sim 7$	8	$0 \sim 7$
	2	2	8	$0 \sim 7$	8	$0 \sim 7$
	3	1	4	$2 \sim 3, 6 \sim 7$	4	$2 \sim 3, 6 \sim 7$
	4	1	0	–	0	–
$\mathcal{D}^{1^8}_{\{[11111000]\}}$	1	6	8	$0 \sim 7$	8	$0 \sim 7$
	2	3	8	$0 \sim 7$	8	$0 \sim 7$
	3	1	6	$1 \sim 3, 5 \sim 7$	6	$1 \sim 3, 5 \sim 7$
	4	1	1	3	1	3
	5	1	0	–	0	–

Input: The division property of the input multi-set.
Obj: The value of the objective function.
#{Bits}: The number of zero-sum bits under corresponding setting.
Bits: The bit indices of zero-sum bits.

17-*Round Integral Distinguisher with Data Complexity* 2^{56}

$$(\mathcal{A}^8,\mathcal{A}^8,\mathcal{A}^8,\mathcal{C}^8,\mathcal{A}^8,\mathcal{A}^8,\mathcal{A}^8,\mathcal{A}^8) \xrightarrow{17\text{ Rounds}} (\mathcal{U}^7\mathcal{B}^1,\mathcal{U}^8,\mathcal{U}^8,\mathcal{U}^8,\mathcal{U}^8,\mathcal{U}^8,\mathcal{U}^8,\mathcal{U}^8)$$

18-*Round Integral Distinguisher with Data Complexity* 2^{63}

$$(\mathcal{A}^8,\mathcal{A}^8,\mathcal{A}^8,\mathcal{A}^8,\mathcal{A}^8,\mathcal{A}^8,\mathcal{A}^8,\mathcal{A}^7\mathcal{C}^1) \xrightarrow{18\text{ Rounds}} (\mathcal{U}^8,\mathcal{U}^8,\mathcal{U}^8,\mathcal{U}^8,\mathcal{U}^7\mathcal{B}^1,\mathcal{U}^8,\mathcal{U}^8,\mathcal{U}^8)$$

18-*Round Integral Distinguisher with Data Complexity* 2^{63}

$$(\mathcal{A}^8,\mathcal{A}^8,\mathcal{A}^8,\mathcal{A}^7\mathcal{C}^1,\mathcal{A}^8,\mathcal{A}^8,\mathcal{A}^8,\mathcal{A}^8) \xrightarrow{18\text{ Rounds}} (\mathcal{U}^7\mathcal{B}^1,\mathcal{U}^8,\mathcal{U}^8,\mathcal{U}^8,\mathcal{U}^8,\mathcal{U}^8,\mathcal{U}^8,\mathcal{U}^8)$$

B.2 Integral Distinguishers for LEA

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

$$(\mathcal{A}^{32},\mathcal{C}^{32},\mathcal{C}^{32},\mathcal{C}^{32}) \xrightarrow{6\text{ Rounds}} (\mathcal{U}^{32},\mathcal{U}^4\mathcal{B}^1\mathcal{U}^{27},\mathcal{U}^{32},\mathcal{U}^{32})$$

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

$$(\mathcal{C}^{32},\mathcal{C}^{32},\mathcal{C}^{32},\mathcal{A}^{32}) \xrightarrow{6\text{ Rounds}} (\mathcal{U}^{32},\mathcal{U}^3\mathcal{B}^2\mathcal{U}^{27},\mathcal{U}^{32},\mathcal{U}^{32})$$

7-*Round Integral Distinguisher with Data Complexity* 2^{96}

$$(\mathcal{C}^{32},\mathcal{A}^{32},\mathcal{A}^{32},\mathcal{A}^{32}) \xrightarrow{7\text{ Rounds}} (\mathcal{U}^{32},\mathcal{U}^4\mathcal{B}^1\mathcal{U}^{27},\mathcal{U}^{32},\mathcal{U}^{32})$$

B.3 Integral Distinguishers for TEA

15-*Round Integral Distinguisher with Data Complexity* 2^{62}

$$(\mathcal{A}^{32},\mathcal{A}^{30}\mathcal{C}^2) \xrightarrow{15\text{ Rounds}} (\mathcal{U}^{31}\mathcal{B}^1,\mathcal{U}^{32})$$

15-*Round Integral Distinguisher with Data Complexity* 2^{63}

$$(\mathcal{A}^{32},\mathcal{A}^{31}\mathcal{C}^1) \xrightarrow{15\text{ Rounds}} (\mathcal{U}^{30}\mathcal{B}^2,\mathcal{U}^{32})$$

B.4 Integral Distinguishers for XTEA

15-*Round Integral Distinguisher with Data Complexity* 2^{62}

$$(\mathcal{A}^{32},\mathcal{A}^{30}\mathcal{C}^2) \xrightarrow{15\text{ Rounds}} (\mathcal{U}^{31}\mathcal{B}^1,\mathcal{U}^{32})$$

15-*Round Integral Distinguisher with Data Complexity* 2^{63}

$$(\mathcal{A}^{32},\mathcal{A}^{31}\mathcal{C}^1) \xrightarrow{15\text{ Rounds}} (\mathcal{U}^{30}\mathcal{B}^2,\mathcal{U}^{32})$$

B.5 Integral Distinguishers for KATAN/KTANTAN Family of Block Ciphers

Integral Distinguisher for KATAN/KTANTAN32

88-*Round Integral Distinguisher with Data Complexity* 2^{30}

$$(\mathcal{C}^2\mathcal{A}^6, \mathcal{A}^8, \mathcal{A}^8, \mathcal{A}^8) \xrightarrow{88 \text{ Rounds}} (\mathcal{U}^8, \mathcal{U}^8, \mathcal{B}^3\mathcal{U}^5, \mathcal{U}^8)$$

89-*Round Integral Distinguisher with Data Complexity* 2^{30}

$$(\mathcal{C}^2\mathcal{A}^6, \mathcal{A}^8, \mathcal{A}^8, \mathcal{A}^8) \xrightarrow{89 \text{ Rounds}} (\mathcal{U}^8, \mathcal{U}^8, \mathcal{U}^1\mathcal{B}^2\mathcal{U}^5, \mathcal{U}^8)$$

90-*Round Integral Distinguisher with Data Complexity* 2^{30}

$$(\mathcal{C}^2\mathcal{A}^6, \mathcal{A}^8, \mathcal{A}^8, \mathcal{A}^8) \xrightarrow{90 \text{ Rounds}} (\mathcal{U}^8, \mathcal{U}^8, \mathcal{U}^2\mathcal{B}^1\mathcal{U}^5, \mathcal{U}^8)$$

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

$$(\mathcal{C}^1\mathcal{A}^7, \mathcal{A}^8, \mathcal{A}^8, \mathcal{A}^8) \xrightarrow{97 \text{ Rounds}} (\mathcal{U}^8, \mathcal{U}^8, \mathcal{B}^3\mathcal{U}^5, \mathcal{U}^8)$$

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

$$(\mathcal{C}^1\mathcal{A}^7, \mathcal{A}^8, \mathcal{A}^8, \mathcal{A}^8) \xrightarrow{98 \text{ Rounds}} (\mathcal{U}^8, \mathcal{U}^8, \mathcal{U}^1\mathcal{B}^2\mathcal{U}^5, \mathcal{U}^8)$$

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

$$(\mathcal{C}^1\mathcal{A}^7, \mathcal{A}^8, \mathcal{A}^8, \mathcal{A}^8) \xrightarrow{99 \text{ Rounds}} (\mathcal{U}^8, \mathcal{U}^8, \mathcal{U}^2\mathcal{B}^1\mathcal{U}^5, \mathcal{U}^8)$$

Integral Distinguisher for KATAN/KTANTAN48

76-*Round Integral Distinguisher with Data Complexity* 2^{46}

$$(\mathcal{C}^2\mathcal{A}^{10}, \mathcal{A}^{12}, \mathcal{A}^{12}, \mathcal{A}^{12}) \xrightarrow{76 \text{ Rounds}} (\mathcal{U}^{12}, \mathcal{U}^{12}, \mathcal{U}^2\mathcal{B}^3\mathcal{U}^7, \mathcal{U}^{12})$$

76.5-*Round Integral Distinguisher with Data Complexity* 2^{46}

$$(\mathcal{C}^2\mathcal{A}^{10}, \mathcal{A}^{12}, \mathcal{A}^{12}, \mathcal{A}^{12}) \xrightarrow{76.5 \text{ Rounds}} (\mathcal{U}^{12}, \mathcal{U}^{12}, \mathcal{U}^3\mathcal{B}^2\mathcal{U}^7, \mathcal{U}^{12})$$

77-*Round Integral Distinguisher with Data Complexity* 2^{46}

$$(\mathcal{C}^2\mathcal{A}^{10}, \mathcal{A}^{12}, \mathcal{A}^{12}, \mathcal{A}^{12}) \xrightarrow{77 \text{ Rounds}} (\mathcal{U}^{12}, \mathcal{U}^{12}, \mathcal{U}^4\mathcal{B}^1\mathcal{U}^7, \mathcal{U}^{12})$$

77.5-*Round Integral Distinguisher with Data Complexity* 2^{47}

$$(\mathcal{C}^1\mathcal{A}^{11}, \mathcal{A}^{12}, \mathcal{A}^{12}, \mathcal{A}^{12}) \xrightarrow{77.5 \text{ Rounds}} (\mathcal{U}^{12}, \mathcal{U}^{12}, \mathcal{U}^2\mathcal{B}^3\mathcal{U}^7, \mathcal{U}^{12})$$

78-*Round Integral Distinguisher with Data Complexity* 2^{47}

$$(\mathcal{C}^1\mathcal{A}^{11}, \mathcal{A}^{12}, \mathcal{A}^{12}, \mathcal{A}^{12}) \xrightarrow{78 \text{ Rounds}} (\mathcal{U}^{12}, \mathcal{U}^{12}, \mathcal{U}^3\mathcal{B}^2\mathcal{U}^7, \mathcal{U}^{12})$$

78.5-*Round Integral Distinguisher with Data Complexity* 2^{47}

$$(\mathcal{C}^1\mathcal{A}^{11}, \mathcal{A}^{12}, \mathcal{A}^{12}, \mathcal{A}^{12}) \xrightarrow{78.5 \text{ Rounds}} (\mathcal{U}^{12}, \mathcal{U}^{12}, \mathcal{U}^4\mathcal{B}^1\mathcal{U}^7, \mathcal{U}^{12})$$

Integral Distinguisher for KATAN/KTANTAN64

66.6-*Round Integral Distinguisher with Data Complexity* 2^{62}

$$(\mathcal{C}^2\mathcal{A}^{14}, \mathcal{A}^{16}, \mathcal{A}^{16}, \mathcal{A}^{16}) \xrightarrow{66.6 \text{ Rounds}} (\mathcal{U}^{16}, \mathcal{U}^{16}, \mathcal{U}^4\mathcal{B}^3\mathcal{U}^9, \mathcal{U}^{16})$$

67-*Round Integral Distinguisher with Data Complexity* 2^{62}

$$(\mathcal{C}^2\mathcal{A}^{14}, \mathcal{A}^{16}, \mathcal{A}^{16}, \mathcal{A}^{16}) \xrightarrow{67 \text{ Rounds}} (\mathcal{U}^{16}, \mathcal{U}^{16}, \mathcal{U}^5\mathcal{B}^2\mathcal{U}^9, \mathcal{U}^{16})$$

67.3-*Round Integral Distinguisher with Data Complexity* 2^{62}

$$(\mathcal{C}^2\mathcal{A}^{14}, \mathcal{A}^{16}, \mathcal{A}^{16}, \mathcal{A}^{16}) \xrightarrow{67.3 \text{ Rounds}} (\mathcal{U}^{16}, \mathcal{U}^{16}, \mathcal{U}^6\mathcal{B}^1\mathcal{U}^9, \mathcal{U}^{16})$$

71.6-*Round Integral Distinguisher with Data Complexity* 2^{63}

$$(\mathcal{C}^1\mathcal{A}^{15}, \mathcal{A}^{16}, \mathcal{A}^{16}, \mathcal{A}^{16}) \xrightarrow{71.6 \text{ Rounds}} (\mathcal{U}^{16}, \mathcal{U}^{16}, \mathcal{U}^4\mathcal{B}^3\mathcal{U}^9, \mathcal{U}^{16})$$

72-*Round Integral Distinguisher with Data Complexity* 2^{63}

$$(\mathcal{C}^1\mathcal{A}^{15}, \mathcal{A}^{16}, \mathcal{A}^{16}, \mathcal{A}^{16}) \xrightarrow{72 \text{ Rounds}} (\mathcal{U}^{16}, \mathcal{U}^{16}, \mathcal{U}^5\mathcal{B}^2\mathcal{U}^9, \mathcal{U}^{16})$$

72.3-*Round Integral Distinguisher with Data Complexity* 2^{63}

$$(\mathcal{C}^1\mathcal{A}^{15}, \mathcal{A}^{16}, \mathcal{A}^{16}, \mathcal{A}^{16}) \xrightarrow{72.3 \text{ Rounds}} (\mathcal{U}^{16}, \mathcal{U}^{16}, \mathcal{U}^6\mathcal{B}^1\mathcal{U}^9, \mathcal{U}^{16})$$