IACR Transactions on Symmetric Cryptology
ISSN 2519-173X, Vol. 2019, No. 3, pp. 81–102. DOI:10.13154/tosc.v2019.i3.81-102

Revisit Division Property Based Cube Attacks: Key-Recovery or Distinguishing Attacks?

Chen-Dong Ye and Tian Tian

People's Liberation Army Strategic Support Force Information Engineering University, 62 Kexue Road, Zhengzhou, 450001, China. ye_chendong@126.com,tiantian_d@126.com

Abstract. Cube attacks are an important type of key recovery attacks against stream ciphers. In particular, they are shown to be powerful against Trivium-like ciphers. Traditional cube attacks are experimental attacks which could only exploit cubes of size less than 40. At CRYPTO 2017, division property based cube attacks were proposed by Todo et al., and an advantage of introducing the division property to cube attacks is that large cube sizes which are beyond the experimental range could be explored, and so powerful theoretical attacks were mounted on many lightweight stream ciphers.

In this paper, we revisit the division property based cube attacks. There is an important assumption, called Weak Assumption, proposed in division property based cube attacks to support the effectiveness of key recovery. Todo et al. in CRYPTO 2017 said that the Weak Assumption was expected to hold for theoretically recovered superpolies of Trivium according to some experimental results on small cubes. In this paper, it is shown that the Weak Assumption often fails in cube attacks against Trivium, and moreover a new method to recover the exact superpoly of a given cube is developed based on the bit-based division property. With our method, for the cube I proposed by Todo et al. at CRYPTO 2017 to attack the 832-round Trivium, we recover its superpoly $p_I(\boldsymbol{x}, \boldsymbol{v}) = v_{68}v_{78} \cdot (x_{58} \oplus v_{70}) \cdot (x_{59}x_{60} \oplus x_{34} \oplus x_{61})$. Furthermore, we prove that some best key recovery results given at CRYPTO 2018 on Trivium are actually distinguishing attacks. Hopefully this paper gives some new insights on accurately recovering the superpolies with the bit-based division property and also attract some attention on the validity of division property based cube attacks against stream ciphers.

Keywords: Division property, cube attacks, MILP, Trivium

1 Introduction

The cube attack is a powerful cryptanalytic technique against stream ciphers proposed by Dinur and Shamir at Eurocrypt 2009 in [DS09]. Let $f(\boldsymbol{x}, \boldsymbol{v})$ be a tweakable Boolean polynomial describing the first output bit of a stream cipher, where $\boldsymbol{x}$ are secret key variables and $\boldsymbol{v}$ are public IV variables. Let I be a subset of IV indices. Then $f(\boldsymbol{x}, \boldsymbol{v})$ could be written as

$$f(\boldsymbol{x}, \boldsymbol{v}) = t_I \cdot p_I(\boldsymbol{x}, \boldsymbol{v}) \oplus q_I(\boldsymbol{x}, \boldsymbol{v}),$$

where $t_I = \prod_{i \in I} v_i$ and each term of q_I is not divisible by t_I. The variables indexed by I are called cube variables. By assigning all possible combinations of 0/1 values to the cube variables, one could derive $2^{|I|}$ polynomials from f, where the set of $2^{|I|}$ possible assignments is called a cube. The symbolic sum $p_I(\boldsymbol{x}, \boldsymbol{v})$ of all these $2^{|I|}$ resultant polynomials is called the superpoly of I in f. The goal of cube attacks is to find a set I and a proper assignment $\boldsymbol{IV}$ to the noncube variables such that the superpoly $p_I(\boldsymbol{x}, \boldsymbol{IV})$ is a low-degree and nonconstant polynomial in $\boldsymbol{x}$. Once such a set I and a proper assignment

Received: 2018-11-23, Revised: 2019-06-01, Accepted: 2019-08-01, Published: 2019-09-20

$\boldsymbol{IV}$ are found, an equation on secret key variables $\boldsymbol{x}$ could be built by inquiring the value of $p_I(\boldsymbol{x}, \boldsymbol{IV})$ online.

In [DS09, FV13, MS11, YT18b], the original polynomial $f(\boldsymbol{x}, \boldsymbol{v})$ is treated as a black-box polynomial and linearity/quadraticity tests are used to detect low-degree superpolies. If a superpoly passes through linearity/quadraticity tests, the algebraic normal form (ANF) of it could also be experimentally recovered by interpolation. Cube attacks in [DS09, FV13, MS11, YT18b] are called experimental cube attacks, since superpolies are recovered by experimental tests. The advantage of an experimental cube attack is that it is easy to verify the correctness of a recovered superpoly since its ANF is clearly provided. However, testing cubes of size greater than 35 is time consuming. Hence, in experimental cube attacks, the sizes of cubes are typically confined to 40, which greatly restricts the number of attacking rounds.

In [YT18a], based on the numeric mapping technique proposed by Liu in [Liu17], the authors proposed a new variant of cube attacks, which is named deterministic cube attacks. In [YT18a], by expressing the output bit z as a polynomial on the internal state $s^{(t)}$, i.e. $z_t = g_t(s^{(t)})$, they first introduced a new type of cubes, called *useful cubes*. For a useful cube I, the numeric degree of each term of g_t is less than or equal to $|I|$. Due to this special property of useful cubes, their superpolies could be recovered more easily. Then, they developed an algebraic method to recover the exact superpoly of a useful cube. As a result, they recovered some superpolies of up to the 838-round Trivium. However, it seems that the deterministic cube attacks could not exploit large cubes and the recovered superolies are highly biased, where the number of 0's is much larger than the number of 1's.

In [TIHM17], by introducing the bit-based division property into cube attacks, Todo et al. could exploit large cube sizes and theoretically evaluate the security of a stream cipher against cube attacks. The division property was first proposed by Todo in [Tod15] as a generalization of integral property used in integral cryptanalysis against block ciphers. In [Tod15], Todo systematically studied propagation rules of division property against Feistel Networks and Substitute-Permutation Networks (SPN). Later, in [TM16], Todo and Morii further proposed the bit-based division property and applied it to the SIMON family yielding several new integral distinguishers. In [XZBL16], Xiang et al. introduced mixed integer linear programming (MILP) models to evaluate the division propagation which was shown to be more efficient.

In [TIHM17], for a cube C_I, with the help of MILP aided division property, the authors could determine a set J of key variables which includes all the key variables appearing in the superpoly $p_I(\boldsymbol{x}, \boldsymbol{v})$. Then, by constructing the truth table of p_I for some randomly chosen assignments to noncube variables, they attempted to find a proper assignment $\boldsymbol{IV}$ such that $p_I(\boldsymbol{x}, \boldsymbol{IV})$ was nonconstant. Once a nonconstant superpoly p_I was found, a part of the key information could be recovered. Due to the power of MILP solvers, large cubes could be explored. For example, in [TIHM17], it was shown that the superpoly of a given 72-dimensional cube depended on at most five key variables for the 832-round Trivium. Later in [WHT+18], the authors introduced several techniques to improve the division property based cube attacks proposed in [TIHM17]. Their techniques focused on finding proper assignments of noncube variables faster and reducing the complexity of recovering the superpoly. It was shown in [WHT+18] that the superpoly of a given 78-dimensional cube was dependent on at most one key variable for the 839-round Trivium. However, in the division property based cube attacks, it could not guarantee a nonconstant superpoly. Hence, the key recover attacks proposed in [TIHM17, WHT+18] may be just distinguishing attacks.

Besides recovering the superpolies to retrieve key variables directly, there are another two important variants of cube attacks, namely dynamic cube attacks [DS11] and correlation cube attacks [LYWL18]. Dynamic cube attacks recover key variables by exploiting

distinguishers on superpolies such as unbalance and constantness. To obtain such distinguishers, the main idea of dynamic cube attacks is to simplify the ANF representation of some intermediate state bits by assigning dynamic constraints to public variables. Dynamic cube attacks were successfully used to break Grain-128 [DS11, DGP+11]. Although in [FWDM18], the authors propose dynamic cube attacks against the 721- and 855-round Trivium, quickly the attack against 721-Trivium was experimentally verified to fail and some complexity analysis also indicated that the 855-round attack was questionable in [HJL+18]. Correlation cube attacks recover key variables by solving a system of probabilistic equations in key variables derived from conditional correlation properties between superpolies and a set of simple key expressions which is a basis of the superpoly. In [LYWL18], a correlation cube attack was applied to the 835-round Trivium which could recover about 5-bit key information with time complexity 2^{44}, using 2^{45} keystream bits and preprocessing time 2^{51}.

1.1 Motivations

In this part, we first revisit what is guaranteed by a bit-based division trail for an r-round iterative cipher. Second, we briefly discuss how division trails are used in cube attacks against stream ciphers, and we will see that a problem may occur when using division trails in cube attacks. Third, we discuss the countermeasure proposed by previous papers to the problem. Finally, we explain why we focus on Trivium.

Let E be a target r-round n-bit iterated cipher, whose input variables are denoted by $x_1, x_2, \ldots, x_n$ and the output variable is denoted by y, respectively, i.e., $y = E(x_1, x_2, \ldots, x_n)$. If E is a block cipher, then $x_1, x_2, \ldots, x_n$ will represent plaintext bits and y means a ciphertext bit[1]. If E is a stream cipher, then $x_1, x_2, \ldots, x_n$ will represent both IV and key variables and y means the first keystream bit. If there is a division trail $\boldsymbol{k}_0 \xrightarrow{E} \boldsymbol{k}_r = 1$, then attackers do *not* know whether the output bit y is balanced or not[2]. On the other hand, if there is no division trail such that $\boldsymbol{k}_0 \xrightarrow{E} 1$, then it is guaranteed that the output bit y is balanced. The prevail method to check the existence of a division trail like $\boldsymbol{k}_0 \xrightarrow{E} 1$ is using MILP solvers, that is, generating an MILP model to cover all division trails starting from $\boldsymbol{k}_0$, and the feasibility of the model will tell us whether there is a division trail from $\boldsymbol{k}_0$ to 1 or not.

Next let us revisit how the division property is used in a cube attack against a stream cipher $f(\boldsymbol{x}, \boldsymbol{v})$ where $\boldsymbol{x}$ and $\boldsymbol{v}$ denote the secret key and IV variables, respectively. Given a cube C_I and a secret key variable x_j where I is a subset of IV indices, generate an MILP model $\mathcal{M}$ that covers all division trails from $(\boldsymbol{e}_j, \boldsymbol{k}_I)$ and evaluate whether there exists a division trail such that $(\boldsymbol{e}_j, \boldsymbol{k}_I) \xrightarrow{f} 1$, where $\boldsymbol{k}_I$ is the division property of the cube C_I and $\boldsymbol{e}_j$ is the unit vector whose only j-th bit is 1. If there is no division trail such that $(\boldsymbol{e}_j, \boldsymbol{k}_I) \xrightarrow{f} 1$, then it is guaranteed that the secret key variable x_j is *not* involved in the superpoly p_I of the cube C_I. But on the other hand, if there is a division trail such that $(\boldsymbol{e}_j, \boldsymbol{k}_I) \xrightarrow{f} 1$, attackers do not know whether the secret key variable x_j is involved in $p_I(\boldsymbol{x}, \boldsymbol{v})$ or not. Hence, what is guaranteed by the division property in a cube attack is a set of secret key variables not appearing in a target superpoly. However, to mount a key recovery attack, attackers need to know all key variables that are involved in a superpoly.

The countermeasure used in the previous papers on division property based cube attacks is giving the following assumption to support the existence of key variables in a

[1] Generally, for a block cipher, there are n-bit ciphertexts. Because we focus on stream ciphers, here we simplify the representation of an iterated block cipher to make it look unified with that of a stream cipher.

[2] Since there is a division trail $\boldsymbol{k}_0 \xrightarrow{E} \boldsymbol{k}_r = 1$, the output bit has the division property $D_1^{1^1}$. According to the definition of division property, it indicates that attackers do not know whether the output bit y is balanced or not.

superpoly, namely Weak Assumption.

Assumption 1 (Weak Assumption [TIHM17, TIHM18, WHT+18]). *For a cube C_I, there are many values in the constant part of IV whose corresponding superpoly is not a constant function.*

Based on this assumption, a division property based cube attack goes roughly like this. Also take $f(\boldsymbol{x}, \boldsymbol{v})$ and the cube C_I for example. First, using the division property to rule out a set J_1, where x_j does not appear in the superpoly p_I for $x_j \in J_1$. On the other hand, with J_1, one could know the set J_2 which includes all the key variables which may appear in the superpoly p_I. However, for $x_j \in J_2$, it is not guaranteed that x_j appears in the superpoly p_I. Particularly, when p_I is a constant polynomial, there would not be key variables involved in p_I. Second, according to the Weak Assumption, one could easily find a proper assignment to the constant part of IV variables such that $p_I(\boldsymbol{x}, \boldsymbol{IV})$ is nonconstant. Third, after finding a nonconstant $p_I(\boldsymbol{x}, \boldsymbol{IV})$, attackers recover the ANF of p_I on the variables of J_2. Finally, attackers could build an equation on key variables by obtaining the value of p_I online.

Because cube sizes used in division property based cube attacks were very large, the best results are theoretical attacks that are impossible to verify experimentally. We argue that if Assumption 1 fails, some key recovery attacks claimed in [TIHM17, TIHM18, WHT+18] will be distinguishing attacks only. The validity of the Weak Assumption was ever briefly discussed in [TIHM18, Sect. 7]. Based on some experiments on small cubes, it was concluded in [TIHM18] that the Weak Assumption was expected to hold in theoretical recovered superpolies for Trivium, and if the assumption did not hold, the recovered superpoly is useful for distinguishing attacks. Although we agree that Assumption 1 should hold with a large probability, it still might fail for a few specific attacks, especially when few key variables are involved in a superpoly, say 1 or 2 key variables, which often happens on Trivium.

The validity of Assumption 1 as well as our experimental observation that Assumption 1 fails in some best key recovery attacks on Trivium is the motivation of our work in this paper.

Finally, there are two reasons for us focusing on Trivium in this paper. First, Trivium is an important and typical target for cube attacks. Second, compared with other NFSR-based ciphers, we feel that Weak Assumption is more likely to fail for Trivium since Trivium has quite simple state update function and the recovered superpolies often involve few key variables. So far we do not observe invalid key recovery results for other NFSR-based ciphers.

1.2 Our Contributions

In this paper, we propose a new method to recover the superpoly $p_I(\boldsymbol{x}, \boldsymbol{v})$ of a cube indexed by a set I in the output bit function $z(\boldsymbol{x}, \boldsymbol{v})$ of a cipher based on the MILP-aided division property. It is known that this is a quite difficult problem for a well-designed cipher since the ANF of $z(\boldsymbol{x}, \boldsymbol{v})$ is designed to be very complex which is generally thought to be a black-box polynomial. However, it is shown in this paper that our method is practically feasible for lots of large cubes based on the MILP-aided division property. Thus, our method has two implications. First, the division property could be used to algebraically recover the superpoly of a large cube while previous division property based cube attacks are only theoretical attacks where the exact ANFs of superpolies are not given. Second, with the aid of the division property, the superpoly of a large cube could be recovered which is impossible for previous experimental cube attacks.

Our basic idea is very simple and consists in gradually expressing z as a polynomial on the initial state $s^{(0)}$ iteratively and by discarding during each iteration all the terms on which the superpoly of I is proved to be zero using the MILP-aided division property. It

can be seen that we only use the division property to judge 0-sum property not identifying key bits. Therefore, our method does not rely on the Weak Assumption. Besides, we emphasize that in our MILP models, all IV bits are set to be variables (active or non-active). When we reach the initial state $s^{(0)}$, we could recover the superpoly $p_I(\boldsymbol{x}, \boldsymbol{v})$ according to how $s^{(0)}$ is initialized. Due to the efficiency of MILP solvers, this iterative progress of computing the local algebraical expansion of z for a given cube is practical on a normal PC.

With the knowledge of the exact superpoly of a given cube, we could easily verify the correctness of the Weak Assumption for previous results. Besides, there are two more benefits brought by our new method. Let $p_I(\boldsymbol{x}, \boldsymbol{v})$ be a recovered superpoly which includes key variables.

- It is very convenient for us to give different assignments of noncube IV variables yielding different superpolies according to the ANF of $p_I(\boldsymbol{x}, \boldsymbol{v})$.
- When the ANF of $p_I(\boldsymbol{x}, \boldsymbol{v})$ has many terms, it is very convenient for us to obtain several simple superpolies by enlarging the set of cube variables.

As an application, we apply our method to the round-reduced Trivium. Consequently, we get the following results.

- For $I_1 = \{1, 2, \ldots, 65, 67, 69, \ldots, 79\}$ proposed in [TIHM17], we recover the superpoly $p_{I_1}(\boldsymbol{x}, \boldsymbol{v})$ of I_1 in the output bit of the 832-round Trivium given by

$$p_{I_1}(\boldsymbol{x}, \boldsymbol{v}) = v_{68}v_{78} \cdot (x_{58} \oplus v_{70}) \cdot (x_{59}x_{60} \oplus x_{34} \oplus x_{61}).$$

  - If we set $\boldsymbol{IV}$ =0x20080000000000000000, then the corresponding superpoly is

$$p_{I_1}(\boldsymbol{x}, \boldsymbol{IV}) = x_{58} \cdot (x_{59}x_{60} \oplus x_{34} \oplus x_{61}).$$

  - If we set $\boldsymbol{IV}$ =0x20280000000000000000, then the corresponding superpoly is

$$p_{I_1}(\boldsymbol{x}, \boldsymbol{IV}) = (x_{58} \oplus 1) \cdot (x_{59}x_{60} \oplus x_{34} \oplus x_{61}).$$

- For the cubes proposed in [WHT+18], we prove that their superpolies in the output bit of the 833-, 835-, 836- and 839-round Trivium are 0-constant.

It can be seen that Weak Assumption proposed in [TIHM17, WHT+18] does not always hold. We summarize our results in the following table.

Table 1: Results on Trivium variants with up to 839 rounds

Rounds	Cube	"Involved" Key Variables	Exact Superpoly
832	I_1	$x_{34}, x_{58}, x_{59}, x_{60}, x_{61}$ [TIHM17]	$v_{68}v_{78} \cdot (x_{58} \oplus v_{70}) \cdot (x_{59}x_{60} \oplus x_{34} \oplus x_{61})$
833	I_2	$x_{49}, x_{58}, x_{60}, x_{64}, x_{74}, x_{75}, x_{76}$ [WHT+18]	0-constant
833	I_3	x_{60} [WHT+18]	0-constant
835	I_4	x_{57} [WHT+18]	0-constant
836	I_5	x_{57} [WHT+18]	0-constant
839	I_6	x_{61} [WHT+18]	0-constant

$I_1 = \{1, 2, \ldots, 65, 67, 69, \ldots, 79\}$
$I_2 = \{1, 2, \ldots, 67, 69, 71, \ldots, 79\}$
$I_3 = \{1, 2, \ldots, 69, 71, 73, \ldots, 79\}$
$I_4 = \{1, 2, 3, 4, 6, 7, \ldots, 50, 52, 53, \ldots, 64, 66, 67, \ldots, 80\}$
$I_5 = \{1, \ldots, 11, 13, \ldots, 42, 44, \ldots, 80\}$
$I_6 = \{1, \ldots, 33, 35, \ldots, 46, 48, \ldots, 80\}$

1.3 Organization

The rest of this paper is organized as follows. Sect. 2 briefly introduces the necessary backgrounds of this paper. In Sect. 3, we introduce two useful lemmas and propose a new method to recover the superpoly. In Sect. 4, we apply our attack framework to the round-reduced Trivium. Finally, Sect. 5 concludes this paper.

2 Preliminaries

2.1 Mixed Integer Linear Programming

Mixed Integer Linear Programming (MILP) is a kind of mathematical optimization whose objective function and constraints are linear, and all or some of the variables are constrained to be integers. Generally, there are variables $\mathcal{M}.var$, constraints $\mathcal{M}.con$, and the objective function $\mathcal{M}.obj$ in an MILP model $\mathcal{M}$. If there is no objective function in $\mathcal{M}$, then MILP solvers like Gurobi [GRB] will return whether $\mathcal{M}$ is feasible. The following is a small example.

Example 1.

$$\begin{aligned}
&\mathcal{M}.var \leftarrow a, b, c \text{ as binary.}\\
&\mathcal{M}.con \leftarrow a + b + c \geq 1\\
&\mathcal{M}.con \leftarrow a + 2b + c \leq 3\\
&\mathcal{M}.obj \leftarrow \text{minimize } a + b + 2c
\end{aligned}$$

The minimum value of $a + b + 2c$ is 1, where $(a, b, c) = (1, 0, 0)$ is one optimal solution.

MILP was first applied to differential and linear cryptanalysis by N. Mouha et al. in [MWGP11]. Since then, MILP has been applied to search characteristics in many cryptanalysis techniques such as differential cryptanalysis [SHW+14, SS14], impossible differential cryptanalysis [ST17] and integral cryptanalysis based on the division property [XZBL16].

2.2 Cube Attacks

The idea of cube attacks was first proposed by Dinur and Shamir in [DS09]. In a cube attack against stream ciphers, an output bit z is described as a tweakable Boolean function f in key variables $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)$ and public IV variables $\boldsymbol{v} = (v_1, v_2, \ldots, v_m)$, i.e., $z = f(\boldsymbol{x}, \boldsymbol{v})$. Let $I = \{i_1, i_2, \ldots, i_d\}$ be a subset of IV indices. Then f can be rewritten as

$$f(\boldsymbol{x}, \boldsymbol{v}) = t_I \cdot p_I(\boldsymbol{x}, \boldsymbol{v}) \oplus q(\boldsymbol{x}, \boldsymbol{v}), \tag{1}$$

where $t_I = \prod_{i \in I} v_i$, p_I does not contain any variable in $\{v_{i_1}, v_{i_2}, \ldots, v_{i_d}\}$, and each term in q is not divisible by t_I. It can be seen that the summation of 2^d functions derived from f by assigning all the possible values to d variables indexed by the set I equals to p_I, that is,

$$\bigoplus_{(v_{i_1}, v_{i_2}, \ldots, v_{i_d}) \in \mathbb{F}_2^d} f(\boldsymbol{x}, \boldsymbol{v}) = p_I(\boldsymbol{x}, \boldsymbol{v}). \tag{2}$$

The public variables in $\{v_{i_1}, v_{i_2}, \ldots, v_{i_d}\}$ are called *cube variables*, while the remaining public variables are called noncube variables. The set C_I of all 2^d possible assignments of the cube variables is called a *d-dimensional cube*, and the polynomial p_I is called the *superpoly* of C_I in f. For the sake of convenience, we also call p_I the superpoly of I in f.

When the noncube IV variables are set to a specific value $\boldsymbol{IV}$, we could get a superpoly that only depends on key variables, which is denoted by $p_I(\boldsymbol{x}, \boldsymbol{IV})$.

A cube attack consists of the preprocessing phase and the online phase. In the preprocessing phase, attackers try to find cubes with low-degree superpolies. In the online phase, the previously found superpolies are evaluated under the real key. By solving a system of low-degree equations, some key variables could be recovered.

2.3 The Bit-Based Division Property

The conventional bit-based division property was introduced in [TM16]. The authors of [TM16] also introduced the bit-based division property using three subsets. In this paper, we focus on the conventional bit-based division property. The definition of the conventional bit-based division property is as follows.

Definition 1 (**Bit-Based Division Property**)**.** Let $\mathbb{X}$ be a multiset whose elements take a value of $\mathbb{F}_2^n$. Let $\mathbb{K}$ be a set whose elements take an n-dimensional bit vector. When the multiset $\mathbb{X}$ has the division property $D_{\mathbb{K}}^{1^n}$, it fulfills the following conditions:

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

where $\boldsymbol{u} \succeq \boldsymbol{k}$ if and only if $u_i \geq k_i$ for all i and $\boldsymbol{x}^{\boldsymbol{u}} = \prod_{i=1}^{n} x_i^{u_i}$.

Let E_r be an r-round iterative cipher of size n, which is initialized with $x_1, x_2, \ldots, x_n$. Assume that $\mathbb{X}$ is the input set with the division property $D_{\mathbb{K}_0}^{1^n}$. Denote by $\mathbb{Y}$ the corresponding output set created from $\mathbb{X}$ by E_r. Generally, it is difficult to evaluate the division property of $\mathbb{Y}$ directly. Based on the propagation rules of basic operations proved in [TM16, XZBL16], the division property of $\mathbb{Y}$, denoted by $D_{\mathbb{K}_r}^{1^n}$, can be figured out by evaluating the propagation of the division property for every round function. More specifically, when the input set $\mathbb{X}$ is generated by a set of active variables indexed by $I = \{i_1, i_2, \ldots, i_d\}$, where the active variables traverse all $2^{|I|}$ possible combinations while the other variables are assigned to constants. Then, $\mathbb{X}$ has the division property $D_{\boldsymbol{k}}^{1^n}$, where $k_i = 1$ if i in I and $k_i = 0$ otherwise. In this case, the division property of $\mathbb{Y}$ can be evaluated as $\{\boldsymbol{k}\} = \mathbb{K}_0 \to \mathbb{K}_1 \to \mathbb{K}_2 \cdots \to \mathbb{K}_r$, where $D_{\mathbb{K}_i}^{1^n}$ is the division property of the internal state after i rounds. Moreover, if there does not exist a vector $\boldsymbol{e}_j$ (only the j-th element is 1) in $D_{\mathbb{K}_r}^{1^n}$, then the j-th output bit is balanced.

However, as r increases, $|\mathbb{K}_r|$ would expand rapidly and lead to a high memory complexity [WHT+18]. It confines the bit-based division property to be applied to small block ciphers such as SIMON32 and Simeck32 [TM16]. To avoid the high memory complexity, in [XZBL16], the authors applied the MILP methods to the bit-based division property. They first introduced the concept of division trails, which is defined as follows.

Definition 2 (**Division Trail** [XZBL16])**.** Let us consider the propagation of the division property $\{\boldsymbol{k}\} = \mathbb{K}_0 \to \mathbb{K}_1 \to \mathbb{K}_2 \cdots \to \mathbb{K}_r$. Moreover, for any vector $\boldsymbol{k}_{i+1}^* \in \mathbb{K}_{i+1}$, there must exist a vector $\boldsymbol{k}_i^* \in \mathbb{K}_i$ such that $\boldsymbol{k}_i^*$ can propagate to $\boldsymbol{k}_{i+1}^*$ by the propagation rules of division property. 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 $\boldsymbol{k}_i$ can propagate to $\boldsymbol{k}_{i+1}$ for $i \in \{0, 1, \ldots, r-1\}$, we call $\boldsymbol{k}_0 \to \boldsymbol{k}_1 \to \cdots \to \boldsymbol{k}_r$ an r-round division trail.

In [XZBL16], the authors described the propagation rules for AND, COPY and XOR with MILP models, see [XZBL16] for the detailed definition of AND, COPY and XOR. Therefore, they could build an MILP model to cover all the possible division trails generated during the propagation. Besides, in [TIHM17, SWW16], the authors made some simplifications to those MILP models in [XZBL16]. Later, to describe the propagation of

division property more precisely, the authors of [WHT+18] proposed the flag technique. In the following, we briefly recall the flag technique and MILP models describing the modified propagation rules of basic operations considering the effects of flags.

The Flag Technique [WHT+18]. To treat each variable more appropriately, for each variable in v the MILP model $\mathcal{M}$, the authors introduced a flag variable $v.F \in \{0_c, 1_c, \delta\}$, where 1_c means the bit is constant 1, 0_c means constant 0 and δ means variable. Then, the authors defined $=, \oplus$ and $\times$ operations for the elements of set $\{0_c, 1_c, \delta\}$. The $=$ operation tests whether two elements are equal (naturally $1_c = 1_c, 0_c = 0_c$ and $\delta = \delta$). The $\oplus$ operation follows the rules:

$$\begin{cases} 1_c \oplus 1_c = 0_c \\ 0_c \oplus x = x \oplus 0_c = x \text{ for arbitrary } x \in \{1_c, 0_c, \delta\} \\ \delta \oplus x = x \oplus \delta = \delta \end{cases} \tag{3}$$

The $\times$ operation follows the rules:

$$\begin{cases} 1_c \times x = x \times 1_c = x \\ 0_c \times x = x \times 0_c = 0_c \text{ for arbitrary } x \in \{1_c, 0_c, \delta\} \\ \delta \times \delta = \delta \end{cases} \tag{4}$$

As a result, the propagation rules of the basic operations XOR, AND and $COPY$ should be modified considering the effects of flags. The modified versions are denoted by copyf, xorf, and andf whose MILP models are described in Propositions 1, 2 and 3 as follows.

Proposition 1 (**MILP Model for AND with Flag** [WHT+18]). *Let* $(a_1, a_2, \ldots, a_m) \xrightarrow{\text{AND}} b$ *be a division trail of* AND. *The following inequalities are sufficient to describe the propagation of the division property for* AND.

$$\begin{cases} \mathcal{M}.var \leftarrow a_1, a_2, \ldots, a_m, b \text{ as binary}, \\ \mathcal{M}.con \leftarrow b = \max(a_1, a_2, \ldots, a_m), \\ b.F = a_1.F \times a_2.F \times \cdots \times a_m.F \\ \mathcal{M}.con \leftarrow b = 0 \text{ if } b.F = 0_c. \end{cases}$$

We denote this process as $(\mathcal{M}, b) \leftarrow andf(\mathcal{M}, a_1, a_2, \ldots, a_m)$.

Proposition 2 (**MILP Model for XOR with Flag** [WHT+18]). *Let* $(a_1, a_2, \ldots, a_m) \xrightarrow{\text{XOR}} b$ *be a division trail of* XOR. *The following inequalities are sufficient to describe the propagation of the division property for* XOR.

$$\begin{cases} \mathcal{M}.var \leftarrow a_1, a_2, \ldots, a_m, b \text{ as binary}, \\ \mathcal{M}.con \leftarrow b = a_1 + a_2 + \cdots + a_m, \\ b.F = a_1.F \oplus a_2.F \oplus \cdots \oplus a_m.F \end{cases}$$

We denote this process as $(\mathcal{M}, b) \leftarrow xorf(\mathcal{M}, a_1, a_2, \ldots, a_m)$.

Proposition 3 (**MILP Model for COPY with Flag** [WHT+18]). *Let* $a \xrightarrow{\text{COPY}} (b_1, b_2, \ldots, b_m)$ *be a division trail of* COPY. *The following inequalities are sufficient to describe the propagation of the division property for* COPY.

$$\begin{cases} \mathcal{M}.var \leftarrow a, b_1, b_2, \ldots, b_m \text{ as binary}, \\ \mathcal{M}.con \leftarrow a = b_1 + b_2 + \cdots + b_m, \\ a.F = b_1.F = b_2.F = \cdots = b_m.F. \end{cases}$$

We denote this process as $(\mathcal{M}, b_1, \cdots, b_m) \leftarrow copyf(\mathcal{M}, a)$.

2.4 Cube Attacks Combining with the Bit-based Division Property

In [TIHM17], the authors applied the bit-based division property to cube attacks. Instead of using the division property to find zero-sum integral distinguishers, in [TIHM17], they used the division property to analyze the ANF coefficients of a Boolean function f. Based on the following lemma and proposition, they proposed the division property based cube attacks.

Lemma 1. *Let $f(\boldsymbol{x})$ be a polynomial from $\mathbb{F}_2^n$ to $\mathbb{F}_2$ and $a_{\boldsymbol{u}}^f$ be the ANF coefficients. Let $\boldsymbol{k}$ be an n-dimensional bit vector. If there is no division trail such that $\boldsymbol{k} \xrightarrow{f} 1$, then $a_{\boldsymbol{u}}^f$ is always 0 for $\boldsymbol{u} \succeq \boldsymbol{k}$.*

Proposition 4. *Let $f(\boldsymbol{x}, \boldsymbol{v})$ be a polynomial, where $\boldsymbol{x}$ and $\boldsymbol{v}$ denote the secret and public variables, respectively. For a set of indices $I = \{i_1, i_2, \ldots, i_d\} \subset \{1, 2, \ldots, m\}$, let C_I be a set where $\{v_{i_1}, v_{i_2}, \ldots, v_{i_d}\}$ traverse all $2^{|I|}$ values and the other public variables are set to constants. Let $\boldsymbol{k}_I$ be an m-dimensional bit vector such that $\boldsymbol{v}^{\boldsymbol{k}_I} = t_I = v_{i_1} v_{i_2} \cdots v_{i_d}$, i.e., $k_i = 1$ if $i \in I$ and $k_i = 0$ otherwise. If there is no division trail such that $(\boldsymbol{e}_j, \boldsymbol{k}_I) \xrightarrow{f} 1$, then x_j is not involved in the superpoly of the cube C_I.*

When f represents the output bit of the target cipher, based on Proposition 4, for a cube C_I, the authors could identify a set J, where $\{x_j | j \in J\}$ is the set of all the key variables involved in the superpoly $p_I(\boldsymbol{x}, \boldsymbol{v})$. More concretely, if there exists a division trail such that $(\boldsymbol{e}_j, \boldsymbol{k}_I) \xrightarrow{f} 1$, then the secret variable x_j is regraded as one of the key variables involved in the superpoly, i.e., J is updated as $J \cup j$. After knowing the set J, in [TIHM17], division property based cube attacks are described in the following three steps.

1. (Offline phase.) Find a preferable superpoly. Set the noncube variables to constants randomly. For each possible value of the key variables indexed by J, query the oracle and obtain the summation of all the $2^{|I|}$ output values. Thus, the truth table of $p_I(\boldsymbol{x}, \boldsymbol{v})$ can be constructed with time complexity $2^{|I|+|J|}$. Search for nonconstant $p_I(\boldsymbol{x}, \boldsymbol{v})$ by changing the values of noncube variables.

2. (Online phase.) Set the noncube variables to previous found values which make the corresponding $p_I(\boldsymbol{x}, \boldsymbol{v})$ nonconstant. Query the encryption oracle and get one bit $p_I(\boldsymbol{x}, \boldsymbol{v})$, denoted by a. Then, we obtain an equation $p_I(\boldsymbol{x}, \boldsymbol{v}) = a$ about secret key variables. The values of key variables which do not satisfy $p_I(\boldsymbol{x}, \boldsymbol{v}) = a$ could be discarded.

3. (Brute-force search phase.) Guess the remaining secret key variables to recover the entire key.

After the division property based cube attacks were first presented in [TIHM17], some improvements were given in [TIHM18, WHT+18]. To name a few, in [TIHM18], which is the full version of [TIHM17], noncube IV variables could be filled up with 0, later in [WHT+18] noncube IV variables could be filled up with either 0 or 1, and also in [WHT+18] with degree bounding and term enumeration techniques, the time complexity to recover the superpoly could be reduced from $2^{|I|+|J|}$ to $2^{|I|} \times \binom{|J|}{\leq d}$ where d is the degree upper bound of the superpoly.

3 Towards Recovering the Superpoly

Let E be the target cipher, which is updated iteratively with a round function. Assume that $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)$ and $\boldsymbol{v} = (v_1, v_2, \ldots, v_m)$ are the secret and public variables in E, respectively. Denote the output bit by $z(\boldsymbol{x}, \boldsymbol{v})$. Let I be a set of cube indices. If we could

recover the exact ANF of superpoly $p_I(\boldsymbol{x}, \boldsymbol{v})$ of I in z, then we could check the validity of Weak Assumption.

However, the output bit $z(\boldsymbol{x}, \boldsymbol{v})$ is usually a very complex polynomial on $\boldsymbol{x}$ and $\boldsymbol{v}$. Namely, it seems difficult to recover $p_I(\boldsymbol{x}, \boldsymbol{v})$ by calculating the complete ANF of $z(\boldsymbol{x}, \boldsymbol{v})$. Note that the superpoly $p_I(\boldsymbol{x}, \boldsymbol{v})$ of I in z usually depends on a tiny part of $z(\boldsymbol{x}, \boldsymbol{v})$. Therefore, rather than calculating the complete ANF of $z(\boldsymbol{x}, \boldsymbol{v})$, our basic idea is to express z as a polynomial on the initial state iteratively and discard the terms where the superpolies of I are 0-constant in each iteration. When we reach the initial internal state $s^{(0)}$, we could recover the superploly $p_I(\boldsymbol{x}, \boldsymbol{v})$ according to how $s^{(0)}$ is initialized in the target cipher.

In the following two subsections, we will first introduce two useful lemmas in Subsection 3.1 and describe our new attack framework in Subsection 3.2.

3.1 Two Useful Lemmas

Let $s^{(t)} = (s_1^{(t)}, s_2^{(t)}, \ldots, s_N^{(t)})$ be the internal state of E after t rounds. Note that $s_i^{(t)}$ is a polynomial on $\boldsymbol{x}$ and $\boldsymbol{v}$ for $1 \leq i \leq N$. Namely,

$$s^{(t)}(\boldsymbol{x}, \boldsymbol{v}) = (s_1^{(t)}(\boldsymbol{x}, \boldsymbol{v}), s_2^{(t)}(\boldsymbol{x}, \boldsymbol{v}), \ldots, s_N^{(t)}(\boldsymbol{x}, \boldsymbol{v})).$$

For simplicity, we denote $s_i^{(t)}(\boldsymbol{x}, \boldsymbol{v})$ by $s_i^{(t)}(\boldsymbol{x}, \boldsymbol{v})$ without any ambiguity. Similarly, $s^{(t)}(\boldsymbol{x}, \boldsymbol{v})$ is denoted by $s^{(t)}$. We propose the following lemma first.

Lemma 2. *For a set of indices $I \subset \{1, 2, \ldots, m\}$, let $\boldsymbol{k}_I$ be an m-dimensional bit vector such that $\boldsymbol{v}^{\boldsymbol{k}_I} = \prod_{i \in I} v_i$. Assume that $u = \prod_{i=1}^{N} (s^{(t)})^{w_u^i}$, where $w_u^i \in \{0, 1\}$. Namely, u is a term which is the product of some internal state bits of $s^{(t)}$. If there is no division trail such that $(\boldsymbol{0}, \boldsymbol{k}_I) \xrightarrow{s^{(t)}} \boldsymbol{w} = (w_1, w_2, \ldots, w_N)$ for each $\boldsymbol{w} \preceq \boldsymbol{w}_u = (w_u^1, w_u^2, \ldots, w_u^N)$, then the superpoly of I in u is 0-constant, where $\boldsymbol{w} \preceq \boldsymbol{w}_u$ means that $w_i \leq w_u^i$ for $1 \leq i \leq N$.*

Proof. Let $\boldsymbol{w} = (w_1, w_2, \ldots, w_N)$ be an N-dimensional vector such that $\boldsymbol{w} \preceq \boldsymbol{w}_u$. Let $F^{\boldsymbol{w}}(\boldsymbol{x}, \boldsymbol{v}) = \prod_{i=1}^{N} (s_i^{(t)}(\boldsymbol{x}, \boldsymbol{v}))^{w_i}$. The ANF representation of $F^{\boldsymbol{w}}(\boldsymbol{x}, \boldsymbol{v})$ is as following.

$$F^{\boldsymbol{w}}(\boldsymbol{x}, \boldsymbol{v}) = \bigoplus_{\boldsymbol{\gamma} \in \mathbb{F}_2^{m+n}} a_{\boldsymbol{\gamma}}^{F^{\boldsymbol{w}}} (\boldsymbol{x} || \boldsymbol{v})^{\boldsymbol{\gamma}}.$$

Since there is no division trial such that $(\boldsymbol{0}, \boldsymbol{k}_I) \xrightarrow{s^{(t)}} (w_1, w_2, \ldots, w_N)$, according to Lemma 1, we have that the coefficient $a_{\boldsymbol{\gamma}}^{F^{\boldsymbol{w}}} = 0$ for $\boldsymbol{\gamma} \succeq (\boldsymbol{0}, \boldsymbol{k}_I)$. Namely, the superpoly of I in $F^{\boldsymbol{w}}$ is 0-constant.

Since there is no division trail $(\boldsymbol{0}, \boldsymbol{k}_I) \xrightarrow{F_t} \boldsymbol{w} = (w_1, w_2, \ldots, w_N)$ for each $\boldsymbol{w} \preceq \boldsymbol{w}_u$, we have that the superpoly of I in u is 0-constant. □

Remark 1. Let $\boldsymbol{w} = (w_1, w_2, \ldots, w_N)$ be an N-dimensional vector such that $\boldsymbol{w} \preceq \boldsymbol{w}_u = (w_u^1, w_u^2, \ldots, w_u^N)$. If there exists a division trail $(\boldsymbol{0}, \boldsymbol{k}_I) \xrightarrow{s^{(t)}} (w_1, w_2, \ldots, w_N)$, then the coefficient $a_{\boldsymbol{\gamma}}^{F^{\boldsymbol{w}}}$ for $\boldsymbol{\gamma} \succeq (\boldsymbol{0}, \boldsymbol{k}_I)$ is not guaranteed to be 0. Namely, the superpoly of I in $F^{\boldsymbol{w}}(\boldsymbol{x}, \boldsymbol{v})$ is not guaranteed to be 0-constant. Since $\boldsymbol{w}_u \succeq \boldsymbol{w}$, i.e, $F^u(\boldsymbol{x}, \boldsymbol{v})$ is divisible by $F^{\boldsymbol{w}}(\boldsymbol{x}, \boldsymbol{v})$, the superpoly of I in u is not guaranteed to be 0-constant according to the definition of the division property.

For simplicity, for a given set I, a term $u = \prod_{j=1}^{h} s_{i_j}^{(t)}$ satisfying the condition in Lemma 2 is called an invalid term for I in the rest of this paper. Besides, for a polynomial $g_t(s^{(t)})$, we denote all terms of g_t by $T(g_t)$, i.e.,

$$T(g_t) = \{a_c \prod_{i=1}^{N} (s_i^{(t)})^{c_i} | a_c = 1, c = (c_1, c_2, \ldots, c_N) \in \mathbb{F}_2^N\}.$$

Based on Lemma 2, we could obtain the following lemma.

Lemma 3. *Let I be a set of cube indices. Assume that the output bit z is presented as a polynomial in $s^{(t)}$, i.e., $z = g_t(s^{(t)})$. Then, according to Lemma 2, $g_t(s^{(t)})$ could be rewritten as $g_t(s^{(t)}) = g_t^1(s^{(t)}) \oplus g_t^2(s^{(t)})$, where each term $u \in T(g_t^2(s^{(t)}))$ is an invalid term for I. It can be seen that the superpoly of I in $z = g_t(s^{(t)})$ is exactly the superpoly of I in $g_t^1(s^{(t)})$.*

Since each term $u \in T(g_t^1(s^{(t)}))$ is an invalid term for I, we have that the superpoly of I in $g_t^2(s^{(t)})$ is 0-constant. Consequently, the superpoly of I in $z = g_t(s^{(t)})$ is exactly the superpoly of I in $g_t^1(s^{(t)})$. According to Lemma 3, when recovering the superpoly of I in z, we could discard the invalid terms for I by expressing the output bit z as a polynomial in the internal state $s^{(t)}$.

3.2 A New Method to Recover Superpolies

In fact, Lemma 3 offers us a new approach to calculate the superpoly $p_I(\boldsymbol{x}, \boldsymbol{v})$ corresponding to a cube indexed by the set I. The basic idea is to express the output z iteratively and utilize Lemma 3 to discard invalid terms for I during each iteration.

To outline our idea, we assume that the output bit z is represented as a polynomial on the internal state $s^{(t)}$, i.e., $z = g_t(s^{(t)})$. Then, for each term $u \in T(g_t)$, we check whether the superpoly of I in u is 0-constant with the MILP-aided division property. If the superpoly of I in u is 0-constant, then we discard u. As a result, we could obtain a simplified polynomial $g_t^1(s^{(t)})$, where $g_t^1(s^{(t)})$ includes all the remaining terms. According to Lemma 3, $p_I(\boldsymbol{x}, \boldsymbol{v})$ equals to the superpoly of I in $g_t^1(s^{(t)})$. Naturally, we could further express $g_t^1(s^{(t)})$ as a polynomial on the internal state $s^{(t-n_t)}$, i.e., $g_t^1(s^{(t)}) = g_{t-n_t}(s^{(t-n_t)})$, where n_t is a positive integer. Similarly, after discarding the invalid terms for I in $T(g_{t-n_t})$, we could obtain a simplified polynomial $g_{t-n_t}^1(s^{(t-n_t)})$. Note that the superpoly of I in $g_{t-n_t}^1(s^{(t-n_t)})$ equals to the superpoly of I in $g_t^1(s^{(t)})$, which equals to $p_I(\boldsymbol{x}, \boldsymbol{v})$. By performing the above procedure iteratively, we can finally obtain a polynomial $g_0^1(s^{(0)})$ such that the superpoly of I in $g_0^1(s^{(0)})$ is exactly the superpoly of I in z. Therefore, we can easily recover $p_I(\boldsymbol{x}, \boldsymbol{v})$ according to how $s^{(0)}$ is initialized.

We describe our idea in Algorithm 1 formally. In Algorithm 1, the procedure **RecursivelyExpressing** is used to express the $f(s^{(t)})$ as a polynomial on the internal state $s^{(t-n_t)}$, where n_t is a positive integer. The procedure **IsZeroConstant** judges whether the superpoly of I in u is 0-constant with the help of the MILP-aided division property. Based on the procedures **RecursivelyExpressing** and **IsZeroConstant**, the procedure **RecoveringSuperpoly** shows the detailed procedure to recover the superpoly of a given cube indexed by the set I against the target cipher.

Remark 2. In the procedure **IsZeroConstant**, we do not set the noncube IV variables to specific constant values, namely, we treat them as non-active variables whose flags are δ. Hence, during each iteration, we only discard the terms on which the superpolies of I are 0-constant for all possible assignments to the noncube IV variables. Consequently, we could recover $p_I(\boldsymbol{x}, \boldsymbol{v})$ completely and exactly rather than the ANF of $p_I(\boldsymbol{x}, \boldsymbol{IV})$, which is the superpoly under the condition that noncube IV variables are assigned to a specific value $\boldsymbol{IV}$.

4 Experimental Results

In this section, we apply our new attack framework to the round-reduced Trivium. We first present the concrete attacking algorithm for Trivium. Then, we perform various experiments on several variants of the round-reduced Trivium.

Algorithm 1 Recover the ANF of the superpoly of a given cube

procedure RecursivelyExpressing($f(s^{(t)})$, THE NUMBER OF RECURSIVE ROUNDS n_t)
    Express $f(s^{(t)})$ as a polynomial on the internal state $s^{(t-n_t)}$, i.e., determine the polynomial g_{t-n_t} such that

$$f(s^{(t)}) = g_{t-n_t}(s^{(t-n_t)}).$$

    return $g_{t-n_t}(s^{(t-n_t)})$
end procedure

procedure IsZeroConstant(THE CHOSEN TERM u, THE SET I OF CUBE INDICES, THE TARGET ROUND t)
    Declare an empty MILP model $\mathcal{M}$;
    Declare $x_1, \cdots, x_n$ be n MILP variables of $\mathcal{M}$ corresponding to secret variables;
    Declare $v_1, \cdots, v_m$ be m MILP variables of $\mathcal{M}$ corresponding to public variables;
    $\mathcal{M}.Con \leftarrow v_i = 1$ for $i \in I$;
    $\mathcal{M}.Con \leftarrow v_i = 0$ for $i \notin I$;
    $x_i.Flag = \delta$ for $i \in \{1, 2, \ldots, n\}$;
    $v_i.Flag = \delta$ for $i \in \{1, 2, \ldots, n\}$;
    for i from 1 to t **do**
        Update $\mathcal{M}$ according to the round function of the target cipher;
    end for
    Set $J = \{j | s_j^{(t)}$appears in the term $u\}$;
    $\mathcal{M}.Con \leftarrow \sum_{j \in J} a_j^{(t)} \leq |J|$, where $a_1^{(t)}, a_2^{(t)}, \cdots, a_N^{(t)}$ are the MILP variables describing the final division property after t rounds propagation.
    $\mathcal{M}.Con \leftarrow a_j^{(t)} = 0$ for $j \notin J$;
    Solve $\mathcal{M}$;
    if $\mathcal{M}$ is feasible **then**
        return FALSE
    else
        return TRUE
    end if
end procedure

procedure RecoveringSuperpoly(THE SET I OF CUBE INDICES, THE TARGET ROUND r)
    Set $f = h(s^{(r)})$ initially, where $h(s^{(r)})$ is the output function of the target cipher;
    Set $t = r$;
    Set $n_t = N_0$, where N_0 is the number of recursive rounds;
    while $t > 0$ **do**
        Set $n_t = t$ if $t \leq N_0$;
        Set $f =$**RecursivelyExpressing**(f, n_t);
        Set $t = t - n_t$;
        Set $tempf = 0$;
        for $u \in T(f)$ **do**
            if IsZeroConstant(u,I,t) returns FALSE **then**
                $tempf = tempf \oplus u$;
            end if
        end for
        Set $f = tempf$;
    end while
    Set $p_I = 0$;
    for $u \in T(f)$ **do**
        Recover $t_u = \prod_{j \in J_u} s_j^{(0)}$ according to how $s_j^{(0)}$ is initialized in the target cipher;
        $p_I = p_I \oplus t_u$;
    end for
    return p_I;
end procedure

Algorithm 2 Pseudo-code of Trivium

$(s_1, s_2, \ldots, s_{93}) \leftarrow (x_1, x_2, \ldots, x_{80}, 0, \ldots, 0)$;
$(s_{94}, s_{95}, \ldots, s_{177}) \leftarrow (v_1, v_2, \ldots, v_{80}, 0, \ldots, 0)$;
$(s_{178}, s_{179}, \ldots, s_{288}) \leftarrow (0, \ldots, 0, 1, 1, 1)$;
for i from 1 to N **do**
    $t_1 \leftarrow s_{66} \oplus s_{93} \oplus s_{91} \cdot s_{92} \oplus s_{171}$;
    $t_2 \leftarrow s_{162} \oplus s_{177} \oplus s_{175} \cdot s_{176} \oplus s_{264}$;
    $t_3 \leftarrow s_{243} \oplus s_{288} \oplus s_{286} \cdot s_{287} \oplus s_{69}$;
    if $i > 1152$ **then**
        $z_{i-1152} \leftarrow s_{66} \oplus s_{93} \oplus s_{162} \oplus s_{177} \oplus s_{243} \oplus s_{288}$;
    end if
    $(s_1, s_2, \ldots, s_{93}) \leftarrow (t_3, s_1, \ldots, s_{92})$;
    $(s_{94}, s_{95}, \ldots, s_{177}) \leftarrow (t_1, s_{94}, \ldots, s_{176})$;
    $(s_{178}, s_{179}, \ldots, s_{288}) \leftarrow (t_2, s_{178}, \ldots, s_{287})$;
end for

4.1 Specification of Trivium

Trivium is a bit oriented synchronous stream cipher designed by De Cannière and Preneel, which was selected as one of eSTREAM hardware-oriented portfolio ciphers. The main building block of Trivium is a 288-bit Galois nonlinear feedback shift register. For every clock cycle there are three bits of the internal state updated by quadratic feedback functions and all the other bits of the internal sate are updated by shifting. The internal state of Trivium, denoted by $(s_1, s_2, \ldots, s_{288})$, is initialized by loading an 80-bit secret key and an 80-bit IV into the registers, and setting all the remaining bits to 0 except for the last three bits of the third register. Then, the algorithm would not output any keystream bit until the internal state is updated $4 \times 288 = 1152$ rounds. This is described by the pseudo-code shown in Algorithm 2. For more details, please refer to [CP08].

4.2 Discarding Terms with Degree Evaluation Methods

Let I be a set of cube indices. Assume that $f(s^{(t)})$ is a polynomial generated in some iteration of Algorithm 1. In our new attack framework, for each term $u \in T(f)$, we need to check whether the superpoly of I in u is 0-constant by solving a corresponding MILP model $\mathcal{M}_u$. When the order of $T(f)$ is large, the time complexity of checking all the terms of f would be high. Our solution is using the degree evaluation method. For the chosen set I, if $\deg_I(u) < |I|$, then the superpoly of I in u is 0-constant, where $\deg_I(u)$ is the degree of u on variables indexed by I. Namely, by utilizing the degree evaluation method based on the numeric mapping, we could discard a large amount of terms of f, and so the number of MILP models needed to be solved could be reduced dramatically. Due to the high efficiency of degree evaluation method, the total solving time could be reduced dramatically. For example, in our application to the 832-round Trivium, we could discard about 4999 out of 5295 terms in the first iteration. Note that it would definitely cost a large amount of times to discard 4999 terms.

4.3 The Concrete Attacking Algorithm for Trivium

In this subsection, we show the concrete algorithm which is used to recover superpolies in cube attacks against Trivium.

First, in the procedure **TriviumCore** of Algorithm 3, we show the MILP model which describes the propagation of division property through the round function of Trivium. Based on **TriviumCore**, for a given term u, the procedure **GenerateMILPModelForTrivium** shows how to build an MILP model $\mathcal{M}_u$ which is used to check whether the superpoly of I in u is 0-constant.

Algorithm 3 Generate MILP models for Trivium

procedure TriviumCore($\mathcal{M}$,$\boldsymbol{x}$,i_1,i_2,i_3,i_4,i_5,i)
  for j from 1 to 4 **do**
    $(\mathcal{M}, y_{5\cdot i+j}, z_{4\cdot i+j}) \leftarrow$ **copyf**$(\mathcal{M}, x_{i_j})$;
  end for
  $(\mathcal{M}, a_i) \leftarrow$ **andf**$(\mathcal{M}, z_{4\cdot i+3}, z_{4\cdot i+4})$;
  $(\mathcal{M}, y_{5\cdot i+5}) \leftarrow$ **xorf**$(\mathcal{M}, a_i, z_{4\cdot i+1}, z_{4\cdot i+2}, x_{i_5})$;
  for $j \in \{1, 2, \ldots, 288\} \setminus \{i_1, i_2, i_3, i_4, i_5\}$ **do**
    Set $w_j = x_j$;
  end for
  Set $w_{i_j} = y_{5\cdot i+j}$ for $1 \le j \le 5$;
  return $(\mathcal{M}, \boldsymbol{w})$
end procedure

procedure GenerateMILPModelForTrivium(THE SET I OF CUBE INDICES, THE TARGET ROUND t, THE CHOSEN TERM $u = \prod_{i=1}^{h} s_{j_i}^{(t)}$)
  Declare an empty MILP model $\mathcal{M}$;
  $\mathcal{M}.var \leftarrow w_i^0$ for all $i \in \{1, 2, \ldots, 288\}$;
  Set $w_i^0.F = \delta$ for $i \in \{1, 2, \ldots, 80, 94, 95, \ldots, 173\}$;
  Set $w_i^0.F = 0$ for $i \in \{81, 82, \ldots, 93, 174, 175, \ldots, 285\}$;
  Set $w_i^0.F = 1$ for $i \in \{286, 287, 288\}$;
  for $i = 1$ to t **do**
    $(\mathcal{M}, \boldsymbol{x})$=**TriviumCore**$(\mathcal{M}$,$\boldsymbol{w}^{i-1}$, $66, 171, 91, 92, 93, 3 \cdot i + 1)$;
    $(\mathcal{M}, \boldsymbol{y})$=**TriviumCore**$(\mathcal{M}$,$\boldsymbol{x}$,$162, 264, 175, 176, 177, 3 \cdot i + 2)$;
    $(\mathcal{M}, \boldsymbol{z})$=**TriviumCore**$(\mathcal{M}$,$\boldsymbol{y}$,$243, 69, 286, 287, 288, 3 \cdot i + 3)$;
    Set $\boldsymbol{w}^i = \boldsymbol{z} >>> 1$;
  end for
  $\mathcal{M}.con \leftarrow w_i^t = 0$ for all $i \in \{1, 2, \ldots 288\}$ w/o $\{j_1, j_2, \ldots, j_h\}$;
  $\mathcal{M}.con \leftarrow \sum_{i=1}^{h} w_{j_i}^t \le h$; ▷ Add the constraint corresponding to u;
  return $\mathcal{M}$;
end procedure

Then, in Algorithm 4, we describe how to recover the superpolies in cube attacks against Trivium. The critical part of Algorithm 4 is the while-loop from line 5 to line 19, which consists of the following three phases.

- Call the procedure **ExpressRecursivelyForTrivium** to express the current polynomial $f(s^{(t)})$ as a polynomial on the internal state $s^{(t-n_t)}$, where n_t is positive integer.
- After updating f with the procedure **ExpressRecursivelyForTrivium**, we discard some terms of f with the degree evaluation method based on numeric mapping.
- For each term $u \in RT$, we build an MILP model $\mathcal{M}_u$ to check whether the superpoly of I in u is 0-constant, where RT is the set of terms which cannot be discarded by the degree evaluation method [Liu17].

In Algorithm 4, we perform this while-loop iteratively until we reach the initial internal state $s^{(0)}$. As a result, we could obtain a simplified polynomial $f(s^{(0)})$ such that the superpoly of I in $f(s^{(0)})$ equals to $p_I(\boldsymbol{x}, \boldsymbol{v})$. Finally, according to the way how the state $s^{(0)}$ is initialized, we could recover the superpoly $p_I(\boldsymbol{x}, \boldsymbol{v})$ exactly.

On Determining n_t. For a term $u = \prod_{i=1}^{h} s_{j_i}^{(t)}$, we need to build an MILP model $\mathcal{M}_u$ which describes the propagation of the division property through t rounds. When t is large, $\mathcal{M}_u$ includes a large amount of MILP variables and constraints. Generally, it is not easy to solve such a large and complex MILP model. Hence, we initially set $n_t = 300$, i.e., we express the output bit z after r rounds as a polynomial on the internal state $s^{(r-300)}$. As a result, in the first iteration of the while-loop, we only need to build and solve MILP models covering $r - 300$ rounds propagation of division property which are much easier to

Algorithm 4 Recover the superpoly of a given cube for Trivium

procedure RecoverSuperpolyForTrivium(THE SET I OF CUBE INDICES, THE TARGET ROUND r)
  Set $f = s_{66}^r \oplus s_{93}^r \oplus s_{162}^r \oplus s_{177}^r \oplus s_{243}^r \oplus s_{288}^r$;
  Set $t = r$;
  Set $n_t = 300$;
  while $t > 0$ **do**
    Set $f =$**ExpressRecursivelyForTrivium**(f, n_t); ▷ See Algorithm B in Appendix B.
    Set $t = t - n_t$;
    Set $tempf = 0$;
    Compute the set $RT = \{u | u \in T(f) \text{ and } \mathrm{DEG}_I(u) \geq |I|\}$ with the degree evaluation method proposed by Liu in [Liu17], where $\mathrm{DEG}_I(u)$ is the evaluated degree of u w.r.t. I;
    ▷ When calculating $\mathrm{DEG}_I(u)$, the noncube IV variables are treated as variables of degree 0.
    for each term u in RT **do**
      $\mathcal{M}_u \leftarrow$**GenerateMILPModelForTrivium**(I, u, t);
      Solve $\mathcal{M}_u$;
      if $\mathcal{M}_u$ is feasible **then**
        Set $tempf = tempf \oplus u$;
      end if
    end for
    Set $f = tempf$;
    Set n_t to a proper value; ▷ We would explain the rules in the following.
  end while
  Set $p_I = 0$;
  for $u \in T(f)$ **do**
    Recover $t_u = \prod_{j \in J_u} s_j^{(0)}$ according to how $s_j^{(0)}$ is initialized in the target cipher;
    Set $p_I = p_I \oplus t_u$;
  end for
  return $p_I(\boldsymbol{x}, \boldsymbol{v})$;
end procedure

solve. Furthermore, due to the degree evaluation method, the number of MILP models needed to be solved could be reduced dramatically in the first iteration.

In the following iterations of the while-loop in Algorithm 4, we first set n_t to 30 and decrease n_t if $f_{t-n_t}(s^{(t-n_t)})$ has too many terms, where f_{t-n_t} is generated by expressing $f(s^{(t)})$ as a polynomial on the internal state $s^{(t-n_t)}$. We show the detailed rules in Algorithm 5.

4.4 Experimental Verification

In this subsection, we provide two small examples to illustrate and verify our attacks. In our experiments, the MILP solver Gurobi is used to solve our MILP models. More specifically, we use the .Net API of Gurobi, and we use Visual Studio 2015 to compile our source code under a PC with a Windows operation system.

Example 2. Let $I = \{1, 11, 21, 31, 41, 51, 61, 71\}$ be the set of cube indices. We take the 591-round Trivium for an example. With Algorithm 4, the superpoly $p_I^{591}(\boldsymbol{x}, \boldsymbol{v})$ is recovered in about 4 minutes under a PC with an Intel(R) Core(TM) i5-6200U CPU and 8G RAM.

With the knowledge of $p_I^{591}(\boldsymbol{x}, \boldsymbol{v})$, we could easily determine some different assignments to the noncube IV variables corresponding to different superpolies. In the following, we list some assignments to the noncube IV variables and the corresponding superpolies.

– If we set $\boldsymbol{IV}$ [3] =0x00000000000080040010, then the corresponding superpoly is

$$p_I^{591}(\boldsymbol{x}, \boldsymbol{IV}) = x_{23}x_{24} \oplus x_{25} \oplus x_{67}.$$

[3] $\boldsymbol{IV} = v_{80}||v_{79}||\cdots||v_1$

Algorithm 5 Determine the number of backward rounds

procedure DETERMINEBACKWARDROUNDS(f, t)
  Set $n_t = 30$ if $t \geq 30$, set $n_t = t$ otherwise;
  for $u = \prod_{i=1}^{h} s_{j_i}^{(t)} \in T(f)$ **do**
    Set exp=0;
    for i from 1 to h **do**
      if (($j_i - n_t < 1$) or ($94 \leq j_i \leq 177$ and $j_i - n_t < 94$) or ($178 \leq j_i$ and $j_i - n_t < 178$)) **then**
        Set exp=exp+1;
      end if
    end for
    if exp>3 **then**
      Set $n_t = n_t - 1$;
    end if
  end for
  return return n_t;
end procedure

– If we set $\boldsymbol{IV}$=0x00200000000020000040, then the corresponding superpoly is

$$p_I^{591}(\boldsymbol{x}, \boldsymbol{IV}) = x_{66} \cdot (x_{23}x_{24} \oplus x_{25} \oplus x_{67} \oplus 1).$$

– If we set $\boldsymbol{IV}$=0x00000000000000000000, then the corresponding superpoly is 0-constant, i.e.,

$$p_I^{591}(\boldsymbol{x}, \boldsymbol{0}) = 0.$$

It worth noting that, when setting the noncube IV variables to 0's, we only obtain a zero-sum distinguisher. This implies that setting the noncube IV variables to 0's, which is widely adopted, may not be the optimal choice.

Beside setting noncube IV variables to different values, with the knowledge of the superpoly $p_I(\boldsymbol{x}, \boldsymbol{v})$, we could obtain various superpolies by appending some noncube IV variables to the set of cube variables. With this trick, although $p_I(\boldsymbol{x}, \boldsymbol{v})$ is complex, we can obtain some simple superpolies. We provide the following example.

Example 3. Let $I = \{1, 11, 21, 31, 41, 51, 61, 71\}$ be the set of cube indices. We take the 586-round Trivium for an example. With Algorithm 4, $p_I^{586}(\boldsymbol{x}, \boldsymbol{v})$ is recovered in about 3 minutes under a PC with an Intel(R) Core(TM) i5-6200U CPU and 8G RAM. After knowing $p_I^{586}(\boldsymbol{x}, \boldsymbol{v})$ which is not simple, we easily obtain 5 simple superpolies by appending some noncube variables to the set of cube variables. In Table 2, we show some new cubes with simple superpolies, where all the noncube IV variables are set to 0's.

Table 2: New cubes and the corresponding superpolies

new cubes indices set	superpoly
$I \cup \{32, 37, 42, 50, 73\}$	x_{58}
$I \cup \{32, 37, 42, 49, 50, 70\}$	$x_{60} \oplus 1$
$I \cup \{32, 37, 50, 70, 73\}$	$x_{30} \oplus x_{55}x_{56} \oplus x_{57}$
$I \cup \{23, 24, 32, 42\}$	$x_{65}x_{66} \oplus x_{40} \oplus x_{67}$
$I \cup \{23, 24, 42\}$	$(x_{45}x_{46} \oplus x_{20} \oplus x_{47}) \cdot (x_{65}x_{66} \oplus x_{40} \oplus x_{67})$

4.5 Key Recovery Attacks on Trivium

In this subsection, we target at the Trivium variants with not less than 832 initialization rounds. For the cubes proposed in [TIHM17] and [WHT+18], we recover their superpolies with our methods. All the experiments in this subsection are performed on a PC with an Inter(R) Core i1-7700K CPU and 32G RAM.

Results for the 832-round Trivium. For $I_1 = \{1, 2, \ldots, 65, 67, 69, \ldots, 79\}$ proposed in [TIHM17], we recover the exact ANF of its superpoly $p_{I_1}(\boldsymbol{x}, \boldsymbol{v})$ in the output of the 832-round Trivium given by

$$p_{I_1}(\boldsymbol{x}, \boldsymbol{v}) = v_{68}v_{78} \cdot (x_{58} \oplus v_{70}) \cdot (x_{59}x_{60} \oplus x_{34} \oplus x_{61}).$$

With the knowledge of the p_{I_1}, we could easily determine assignments of the noncube variables which could lead to different superpolies.

- If we set $\boldsymbol{IV}$ =0x20080000000000000000, then the corresponding superpoly is

$$p_I(\boldsymbol{x}, \boldsymbol{IV}) = x_{58} \cdot (x_{59}x_{60} \oplus x_{34} \oplus x_{61}).$$

- If we set $\boldsymbol{IV}$ =0x20280000000000000000, then the corresponding superpoly is

$$p_I(\boldsymbol{x}, \boldsymbol{IV}) = (x_{58} \oplus 1) \cdot (x_{59}x_{60} \oplus x_{34} \oplus x_{61}).$$

Thus, we can recover the value of $x_{59}x_{60} \oplus x_{34} \oplus x_{61}$ regardless of the value of x_{58}. Namely, we can recover at least one key variables for the 832-round Trivium. In particular, when $x_{59}x_{60} \oplus x_{34} \oplus x_{61} = 1$, we could recover the value of x_{58} as well. Consequently, we could attack the 832-round Trivium with a complexity less than 2^{79}.

Results for Trivium Variants with More than 832 Rounds. In [WHT+18], the authors proposed some cubes whose superpolies were declaimed to include key variables. Based on these cubes, the authors presented the key recovery attacks on up to the 839-round Trivium. With our method, we recover the superpolies of these cubes, and we find that all the superpolies are actually zero constant. It indicates that the key recovery attacks declared in [WHT+18] are distinguishing attacks in fact. We summarize our results in the following table.

Table 3: The exact superpolies of the cubes given in [WHT+18]

Rounds	Cube indices	"Involved" Key Variables	Superpoly
833	$\{1, 2, \ldots, 67, 69, 71, \ldots, 79\}$	$x_{49}, x_{58}, x_{60}, x_{74}, x_{75}, x_{76}$	0-constant
833	$\{1, 2, \ldots, 69, 71, 73, \ldots, 79\}$	x_{60}	0-constant
835	$\{1, 2, \ldots, 80\} \setminus \{5, 51, 65\}$	x_{57}	0-constant
836	$\{1, \ldots, 11, 13, \ldots, 42, 44, \ldots, 80\}$	x_{57}	0-constant
839	$\{1, \ldots, 33, 35, \ldots, 46, 48, \ldots, 80\}$	x_{61}	0-constant

5 Conclusion

In this paper, based on the bit-based division property, we propose a new method to recover the exact ANF of the superpoly $p_I(\boldsymbol{x}, \boldsymbol{v})$ corresponding to a given cube indexed by the set I. We practically recovered the superpolies of the cubes proposed in CRYPTO 2017 and CRYPTO 2018, where except the 832-round Trivium, all superpolies are 0. Hence these best key recovery attacks are actually distinguishing attacks. These results on the other hand imply that the validity of the Weak Assumption used in division property based cube attacks should be reconsidered at least for the case of Trivium.

References

[CP08] Christophe De Cannière and Bart Preneel. Trivium. In *New Stream Cipher Designs - The eSTREAM Finalists*, pages 244–266. 2008.

[DGP+11] Itai Dinur, Tim Güneysu, Christof Paar, Adi Shamir, and Ralf Zimmermann. An experimentally verified attack on full grain-128 using dedicated reconfigurable hardware. In *Advances in Cryptology - ASIACRYPT 2011 - 17th International Conference on the Theory and Application of Cryptology and Information Security, Seoul, South Korea, December 4-8, 2011. Proceedings*, pages 327–343, 2011.

[DS09] Itai Dinur and Adi Shamir. Cube attacks on tweakable black box polynomials. In *Advances in Cryptology - EUROCRYPT 2009, 28th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cologne, Germany, April 26-30, 2009. Proceedings*, pages 278–299, 2009.

[DS11] Itai Dinur and Adi Shamir. Breaking grain-128 with dynamic cube attacks. In *Fast Software Encryption - 18th International Workshop, FSE 2011, Lyngby, Denmark, February 13-16, 2011, Revised Selected Papers*, pages 167–187, 2011.

[FV13] Pierre-Alain Fouque and Thomas Vannet. Improving key recovery to 784 and 799 rounds of Trivium using optimized cube attacks. In *Fast Software Encryption - 20th International Workshop, FSE 2013, Singapore, March 11-13, 2013. Revised Selected Papers*, pages 502–517, 2013.

[FWDM18] Ximing Fu, Xiaoyun Wang, Xiaoyang Dong, and Willi Meier. A key-recovery attack on 855-round trivium. In *Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part II*, pages 160–184, 2018.

[GRB] Zonghao Gu, Edward Rothberg, and Robert Bixby. Gurobi optimizer. http://www.gurobi.com/.

[HJL+18] Yonglin Hao, Lin Jiao, Chaoyun Li, Willi Meier, Yosuke Todo, and Qingju Wang. Observations on the dynamic cube attack of 855-round trivium from crypto'18. Cryptology ePrint Archive, Report 2018/972, 2018. https://eprint.iacr.org/2018/972.

[Liu17] Meicheng Liu. Degree evaluation of NFSR-Based cryptosystems. In *Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part III*, pages 227–249, 2017.

[LYWL18] Meicheng Liu, Jingchun Yang, Wenhao Wang, and Dongdai Lin. Correlation cube attacks: From weak-key distinguisher to key recovery. In *Advances in Cryptology - EUROCRYPT 2018 - 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29 - May 3, 2018 Proceedings, Part II*, pages 715–744, 2018.

[MS11] Piotr Mroczkowski and Janusz Szmidt. Corrigendum to: The cube attack on stream cipher Trivium and quadraticity tests. *IACR Cryptology ePrint Archive*, 2011:32, 2011.

[MWGP11] Nicky Mouha, Qingju Wang, Dawu Gu, and Bart Preneel. Differential and linear cryptanalysis using mixed-integer linear programming. In *Information Security and Cryptology - 7th International Conference, Inscrypt 2011, Beijing, China, November 30 - December 3, 2011. Revised Selected Papers*, pages 57–76, 2011.

[SHW+14] Siwei Sun, Lei Hu, Peng Wang, Kexin Qiao, Xiaoshuang Ma, and Ling Song. Automatic security evaluation and (related-key) differential characteristic search: Application to simon, present, lblock, DES(L) and other bit-oriented block ciphers. In *Advances in Cryptology - ASIACRYPT 2014 - 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, R.O.C., December 7-11, 2014. Proceedings, Part I*, pages 158–178, 2014.

[SS14] Meiqin Wang Peng Wang Kexin Qiao Xiaoshuang Ma Danping Shi Ling Song Kai Fu Siwei Sun, Lei Hu. Towards finding the best characteristics of some bit-oriented block ciphers and automatic enumeration of (related-key) differential and linear characteristics with predefined properties. Cryptology ePrint Archive, Report 2014/747, 2014. https://eprint.iacr.org/2014/747.

[ST17] Yu Sasaki and Yosuke Todo. New impossible differential search tool from design and cryptanalysis aspects - revealing structural properties of several ciphers. In *Advances in Cryptology - EUROCRYPT 2017 - 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Paris, France, April 30 - May 4, 2017, Proceedings, Part III*, pages 185–215, 2017.

[SWW16] Ling Sun, Wei Wang, and Meiqin Wang. Milp-aided bit-based division property for primitives with non-bit-permutation linear layers. Cryptology ePrint Archive, Report 2016/811, 2016. https://eprint.iacr.org/2016/811.

[TIHM17] Yosuke Todo, Takanori Isobe, Yonglin Hao, and Willi Meier. Cube attacks on non-blackbox polynomials based on division property. In *Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part III*, pages 250–279, 2017.

[TIHM18] Yosuke Todo, Takanori Isobe, Yonglin Hao, and Willi Meier. Cube attacks on non-blackbox polynomials based on division property. *IEEE Trans. Computers*, 67(12):1720–1736, 2018.

[TM16] 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.

[Tod15] 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.

[WHT+18] Qingju Wang, Yonglin Hao, Yosuke Todo, Chaoyun Li, Takanori Isobe, and Willi Meier. Improved division property based cube attacks exploiting algebraic properties of superpoly. In *Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part I*, pages 275–305, 2018.

[XZBL16] 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. 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.

[YT18a] Chen-Dong Ye and Tian Tian. Deterministic cube attacks: A new method to recover superpolies in practice. Cryptology ePrint Archive, Report 2018/1082, 2018. https://eprint.iacr.org/2018/1082.

[YT18b] Chen-Dong Ye and Tian Tian. A new framework for finding nonlinear superpolies in cube attacks against trivium-like ciphers. In Willy Susilo and Guomin Yang, editors, *Information Security and Privacy - 23rd Australasian Conference, ACISP 2018, Wollongong, NSW, Australia, July 11-13, 2018, Proceedings*, volume 10946 of *Lecture Notes in Computer Science*, pages 172–187. Springer, 2018.

A The Technique of Removing Invalid Division Trails and Its Error

First, we illustrate the technique of removing invalid division by a small example.

Example 4. Let nf be a toy 8-bit NFSR-based stream cipher. Denote the state of nf at time clock t by $s^{(t)} = (s_1^{(t)}, s_2^{(t)}, \cdots, s_8^{(t)})$. Assume that nf is initialized with $x_1, x_2, \cdots, x_8$, i.e., $s^{(0)} = (x_1, x_2, \ldots, x_8)$. The state of nf is updated as

$$\begin{aligned} s_1^{(t+1)} &= s_5^{(t)} \oplus s_6^{(t)} s_7^{(t)} \oplus s_8^{(t)} \\ s_i^{(t+1)} &= s_{i-1}^{(t)} \text{ for } i \in \{2,3,4,5,6,7,8\} \end{aligned}$$

Furthermore, the output bit after r rounds is defined as $z_r = s_1^{(r)} s_3^{(r)}$.

We choose x_2, x_3, x_4, x_5 as active variables. Then, according to the propagation rules of the conventional division property, the following conventional division trail would exist.

round 0	$(0,1,1,1,1,0,0,0)$
round 1	$(1,0,1,1,1,0,0,0)$
round 2	$(1,1,0,1,1,0,0,0)$
round 3	$(0,1,1,0,1,1,0,0)$
round 4	$(0,0,1,1,0,1,1,0)$
round 5	$(1,0,0,1,1,0,0,0)$
round 6	$(0,1,0,0,1,1,0,0)$
round 7	$(0,0,1,0,0,1,1,0)$
round 8	$(1,0,0,1,0,0,0,0)$

In fact, for the above conventional division trail, we could know the distribution of active variables through this division trail. We show the corresponding more detailed division trail, called new division trail, in the following. This new division trial records more information than the conventional one. For example, $[1, x_2x_3]$, the second component of the vector at round 6, indicates that x_2x_3 appears in s_1^6.

round 0	$([0,1],[1,x_2],[1,x_3],[1,x_4],[1,x_5],[0,1],[0,1],[0,1])$
round 1	$([1,x_5],[0,1],[1,x_2],[1,x_3],[1,x_4],[0,1],[0,1],[0,1])$
round 2	$([1,x_4],[1,x_5],[0,1],[1,x_2],[1,x_3],[0,1],[0,1],[0,1])$
round 3	$([0,1],[1,x_4],[1,x_5],[0,1],[1,x_2],[1,x_3],[0,1],[0,1])$
round 4	$([0,1],[0,1],[1,x_4],[1,x_5],[0,1],[1,x_2],[1,x_3],[0,1])$
round 5	$([1,x_2x_3],[0,1],[0,1],[1,x_4],[1,x_5],[0,1],[0,1],[0,1])$
round 6	$([0,1],[1,x_2x_3],[0,1],[0,1],[1,x_4],[1,x_5],[0,1],[0,1])$
round 7	$([0,1],[0,1],[1,x_2x_3],[0,1],[0,1],[1,x_4],[1,x_5],[0,1])$
round 8	$([1,x_4x_5],[0,1],[0,1],[1,x_2x_3],[0,1],[0,1],[0,1],[0,1])$

The above division trail indicates that

$$\bigoplus_{(x_2,x_3,x_4,x_5)\in\mathbb{F}_2^4} z_8$$

is unknown, where $z_8 = s_1^{(8)}s_3^{(8)}$. However, by computing the real ANFs of $s^{(t)}$ for $0 \le t \le 8$, we know that x_4x_5 does not appear in the $s_1^{(8)}$ and $x_2x_3x_4x_5$ does not appear in z_8. This indicates that the above division trail is invalid and should be aborted.

To remove the invalid division trails, our main idea is recording the new division trails and adding constrains corresponding to the vanished terms. In this example, we could add constrains to require that the x_4x_5 does not appear in $s_1^{(8)}$.

However, there is a subtle error in the above technique, and we would like to explain it in the following. The propagation rules of the division property for active variables is not consist with the computation of ANFs for iterative ciphers. We could not remove a division trial by comparing its intermediate result with the ANF for some internal state bit. Here is a small example to illustrate it which we hope will be helpful. Let NF be a stream cipher of size 7. Denote the internal state of NF after t rounds by $s^{(t)} = (s_0^{(t)}, s_1^{(t)}, \cdots, s_6^{(t)})$. Assume that NF is initialized with $x_0, x_1, x_2, x_3, x_4, x_5, x_6$, i.e. $s^{(0)} = (x_0, x_1, x_2, x_3, x_4, x_5, x_6)$. The state of NF is updated as

$$s_0^{(t+1)} = s_3^{(t)} + s_4^{(t)}s_5^{(t)} + s_6^{(t)}$$

$$s_i^{(t+1)} = s_{i-1}^{(t)} \text{ for } 1 \le i \le 6$$

Furthermore, the output bit after r rounds is defined as $z_r = s_0^{(r)}s_1^{(r)}$. Now, we consider the output bit after 9 rounds and choose x_1, x_2, x_3, x_4, x_5 as active variables. According to the update function, $s_1^{(9)} = s_0^{(8)}$ can be computed as

$$s_1^{(9)} = s_0^{(8)} = s_3^{(7)} + s_4^{(7)}s_5^{(7)} + s_6^{(7)},$$

where $s_6^{(7)} = x_3 + x_4x_5 + x_6$, $s_5^{(7)} = x_2 + x_3x_4 + x_5$, $s_4^{(7)} = x_1 + x_2x_3 + x_4$, and $s_3^{(7)} = x_0 + x_1x_2 + x_3$. It can be seen that the term x_4x_5 does not appear in $s_1^{(9)}$. Then , based on our previous idea, a division trail indicating x_4x_5 in $s_1^{(9)}$ is defined as an invalid division trail. After removing all such invalid division trails and solving a corresponding MILP model, it will tell us that there is no division trail such that $(0,1,1,1,1,0) \to 1$, and so we conclude that the multiples of $x_1x_2x_3x_4x_5$ would not appear in the output z_9. However, it can be verified that $x_1x_2x_3x_4x_5$ actually appears in z_9. Therefore, even if x_4x_5 is not in the ANF of $s_1^{(9)}$, we could not remove a division trail indicating that x_4x_5 appears in $s_1^{(9)}$.

B Expressing the Polynomial on Earlier Internal state Variables

In the following algorithm, for a given polynomial $f(s^{(t)})$ on the internal state $s^{(t)}$, we show how to express f as a polynomial on the internal state $s^{(t-n_t)}$, where n_t is a positive integer. When n_t is larger than the length of a register, we can not express f as a polynomial on $s^{(t-n_t)}$ by calling Algorithm B directly. However, this could be achieve by call Algorithm B iteratively.

Algorithm 6 Express the polynomial with earlier internal state

procedure ExpressRecursivelyForTrivium(f, n_t)
  Set $g(s^{(t-n_t)}) = 0$;
  for $u = \prod_{j\in J_u} s_j^{(t)} \in T(f)$ **do**
    for $j \in J_u$ **do**
      Set $l = 1$;
      if $1 \le j \le 93$ **then**
        if $j > n_t$ **then**
          Set $h_l = s_{j-n_t}^{(t-n_t)}$;
        else
          Set $h_l = s_{286-\Delta}^{(t-n_t)} \cdot s_{287-\Delta}^{(t-n_t)} \oplus s_{288-\Delta}^{(t-n_t)} \oplus s_{243-\Delta}^{(t-n_t)} \oplus s_{69-\Delta}^{(t-n_t)}$, where $\Delta = n_t - j$;
        end if
        Set $l = l + 1$;
      end if
      if $94 \le j \le 177$ **then**
        if $j - n_t \ge 94$ **then**
          Set $h_l = s_{j-n_t}^{(t-n_t)}$;
        else
          Set $h_l = s_{91-\Delta}^{(t-n_t)} \cdot s_{92-\Delta}^{(t-n_t)} \oplus s_{93-\Delta}^{(t-n_t)} \oplus s_{66-\Delta}^{(t-n_t)} \oplus s_{171-\Delta}^{(t-n_t)}$, where $\Delta = 94 + n_t - j$;
        end if
        Set $l = l + 1$;
      end if
      if $178 \le j \le 288$ **then**
        if $j - n_t \ge 178$ **then**
          Set $h_l = s_{j-n_t}^{(t-n_t)}$;
        else
          Set $h_l = s_{175-\Delta}^{(t-n_t)} \cdot s_{176-\Delta}^{(t-n_t)} \oplus s_{177-\Delta}^{(t-n_t)} \oplus s_{243-\Delta}^{(t-n_t)} \oplus s_{264-\Delta}^{(t-n_t)}$, where $\Delta = 178 + n_t - j$;
        end if
        Set $l = l + 1$;
      end if
    end for
    Set $g(s^{(t-n_t)}) = g(s^{(t-n_t)}) \oplus \prod_{i=1}^{l} h_i(s^{(t-n_t)})$;
  end for
  return $g(s^{(t-n_t)})$;
end procedure