An Algebraic Formulation of the Division Property: Revisiting Degree Evaluations, Cube Attacks, and Key-Independent Sums

Kai Hu[1,4], Siwei Sun[2,5], Meiqin Wang[1,4](✉), and Qingju Wang[3]

[1] School of Cyber Science and Technology, Shandong University,
Qingdao, Shandong, China
hukai@mail.sdu.edu.cn, mqwang@sdu.edu.cn

[2] State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China
siweisun.isaac@gmail.com

[3] SnT, University of Luxembourg, Esch-sur-Alzette, Luxembourg
qingju.wang@uni.lu

[4] Key Laboratory of Cryptologic Technology and Information Security, Ministry of Education, Shandong University, Qingdao, Shandong, China

[5] School of Cyber Security, University of Chinese Academy of Sciences,
Beijing, China

Abstract. Since it was proposed in 2015 as a generalization of integral properties, the division property has evolved into a powerful tool for probing the structures of Boolean functions whose algebraic normal forms are not available. We capture the most essential elements for the detection of division properties from a pure algebraic perspective, proposing a technique named as *monomial prediction*, which can be employed to determine the presence or absence of a monomial in any product of the coordinate functions of a vectorial Boolean function $\boldsymbol{f}$ by counting the number of the so-called *monomial trails* across a sequence of simpler functions whose composition is $\boldsymbol{f}$. Under the framework of the monomial prediction, we formally prove that most algorithms for detecting division properties in literature raise no false alarms but may miss. We also establish the equivalence between the monomial prediction and the three-subset bit-based division property without unknown subset presented at EUROCRYPT 2020, and show that these two techniques are perfectly accurate.

The monomial prediction technique can be regarded as a purification of the definitions of the division properties without resorting to external multisets. This algebraic formulation gives more insights into division properties and inspires new search strategies. With the monomial prediction, we obtain the *exact* algebraic degrees of Trivium up to 834 rounds for the first time. In the context of cube attacks, we are able to explore a larger search space in limited time and recover the exact algebraic normal forms of complex superpolies with the help of a divide-and-conquer

Due to page limits, all appendixes and some tables of this paper are provided in our full version [11].

S. Moriai and H. Wang (Eds.): ASIACRYPT 2020, LNCS 12491, pp. 446–476, 2020.
https://doi.org/10.1007/978-3-030-64837-4_15

strategy. As a result, we identify more cubes with smaller dimensions, leading to improvements of some near-optimal attacks against 840-, 841- and 842-round Trivium.

Keywords: Division property · Monomial prediction · Detection algorithm · Algebraic degree · Cube attack · Trivium

1 Introduction

The division property [25] was first proposed by Todo at EUROCRYPT 2015 to uncover and exploit the spectrum of properties hidden between the two extremes—the ALL and BLANCE properties in the traditional integral cryptanalysis [6,16] targeting word-oriented primitives. Compared with the traditional integral cryptanalysis, the division property presents a more refined way for cryptanalysts to identify balanced output bits, where the algebraic degree information of the local components of the target is fully utilized. Its powerfulness and potential were undoubtedly demonstrated by the break of the full Misty1 [24]. Subsequently, by considering the division property at the bit level, Todo and Morii [27] introduced the bit-based division property to find balanced bits of the round-reduced Simon. Moreover, to capture also constant output bits and some cancellation characteristics ignored by the conventional bit-based division property, the so-called three-subset bit-based division property was proposed in the same work [27].

This seemingly natural and obvious migration from words to bits (1-bit word) not only makes division properties applicable to bit-oriented designs, but also reveals the intimate relationship between division properties and the algebraic normal forms (ANF) of the target [26], well-beyond merely the algebraic degree. This relationship hints at how the division property can be employed to probe the ANF of a complex Boolean function whose explicit formula is typically not available. As expected, the division property was shown to be useful in (partially) determining the algebraic structures of the superpolies arising in cube attacks [9,26,29,30]. Essentially, every cryptanalysis attempt based on the division property employs some procedures which we call *detection algorithms*.

Detection Algorithms. Given a Boolean function f, a detection algorithm for a certain property $\mathcal{P}$ is a procedure used to determine whether $\mathcal{P}$ holds for f. The property $\mathcal{P}$ can be as simple as "f is a constant" or as complicated as "the sum of f over all possible values of certain variables is zero regardless of the values of some other variables". Given a Boolean function f and a detection algorithm for $\mathcal{P}$, four possibilities are in order:

- *Hit*: $\mathcal{P}$ holds and the output of the algorithm is positive;
- *Miss*: $\mathcal{P}$ holds but the output of the algorithm is negative;
- *False alarm*: $\mathcal{P}$ does not hold but the output of the algorithm is positive;
- *Correct reject*: $\mathcal{P}$ does not hold and the output of the algorithm is negative.

At this point, we remind the readers that a lot of research that has been done on division property so far is about the construction of detection algorithms, loosely speaking, for the balance (or more generally the key-independent constant) property, or more essentially, the absence of certain monomials. A no-false-alarm algorithm can be employed by an attacker (e.g., to find balanced output bits), while a no-miss algorithm can be employed by a designer in security proofs. Our ultimate goal is to devise a perfect and efficient detection algorithm that never misses and never raises false alarms.

Our Contributions. Capturing the algebraic essentials of many attempts to make the detection of division properties more accurate, we propose a new technique called *monomial prediction*. This is a perfect detection algorithm for detecting the presence and absence of any monomial $\boldsymbol{x}^{\boldsymbol{u}}$ in the product $\boldsymbol{y}^{\boldsymbol{v}}$ of any output bits of a vectorial Boolean function $\boldsymbol{y} = \boldsymbol{f}(\boldsymbol{x})$ by counting the number of the so-called *monomial trails* connecting $\boldsymbol{x}^{\boldsymbol{u}}$ and $\boldsymbol{y}^{\boldsymbol{v}}$ across a sequence of simpler vectorial Boolean functions whose composition is $\boldsymbol{f}$. We then establish an equivalence between the monomial prediction approach and the recently proposed three-subset bit-based division property without unknown subset at EUROCRYPT 2020 [9]. We also show that all the predecessors of [9] (except the *lazy propagation* method [27]) can be categorized as no-false-alarm detection algorithms.

The monomial prediction technique can be regarded as a new language for describing the division properties. The original language for the division properties is somehow indirect and vague since a property (the division property) of an object (a vectorial Boolean function) is defined via its effects on external objects (multisets) rather than via its own intrinsic natures. The monomial prediction delivers a definition of division properties fully getting rid of the external multisets. This new treatment not only gives us a unified view on the two-subset bit-based division property, three-subset bit-based division property, and three-subset division property without unknown subset, but also naturally leads to new search strategies. We revisit several well-known applications of the division property with the monomial prediction approach, and identify some improvements over the state-of-the-art.

By showing the presence of monomials with a certain degree and the absence of monomials with larger degrees, we obtain the *exact* algebraic degree of the output bits of Trivium up to 834 rounds for the first time. Our results show that the algebraic degree of 834-round Trivium is only 78, which is much lower than the previous estimations by Liu at CRYPTO 2017 [18], where the upper bound of 793-round Trivium has already reached 79. Along the way, we observe and report on an interesting and somewhat counter-intuitive phenomenon: The algebraic degree of Trivium can drop as the number of rounds grows. For example, the degree of 807-round Trivium has been proven to achieve 71, but the degree of the next round drops to 70.

For a Boolean function f, we can check the presence and absence of all monomials that are divisible by the cube term to recover the superpoly in the cube attack. With the help of a divide-and-conquer strategy, our algorithm achieves high efficiency and scales well, making it possible to test many cubes in a limited time. As a result, we are able to identify some cubes with smaller dimensions for Trivium than the previous best works, for instance, in [8,9] all the cubes chosen for 840-, 841- and 842-round Trivium are of dimension 78, which take 2^{78} encryptions of Trivium to recover one bit information of the key, and take 2^{79} Trivium encryption to recover the remaining key bits by exhaustive search. Thus the total complexity of the key-recovery attack is estimated as $2^{78} + 2^{79} \approx 2^{79.6}$. Using our technique, for 840-round Trivium, we can recover superpolies with three different cubes that have dimension of only 75, which reduces the complexity for recovering the key to $2^{77.8}$ encryption. For 841-round Trivium, we recover two superpolies with two different cubes of dimension 76, which reduces the complexity for recovering the full key to $2^{78.6}$ encryption. For 842-round Trivium, with two different cubes of dimension 76 together with their superpolies, we can recover the full key with time complexity $2^{78.6}$. We summarize our cube attacks on Trivium in Table 1.

Table 1. The complexity of cube attacks on 840-, 841- and 842-round Trivium measured by the encryption of Trivium. #Cube means the number of cubes used in the offline phase of the cube attack.

#Round	Offline phase			Online phase	Total time	Reference
	#Cube	Dimension	#Key			
840	1	78	1	2^{79}	$2^{79.6}$	[9]
	3	75, 75, 75	3	2^{77}	$2^{77.8}$	Sect. 5.2
841	1	78	1	2^{79}	$2^{79.6}$	[9]
	2	76, 76	2	2^{78}	$2^{78.6}$	Sect. 5.2
842	1	78	1	2^{79}	$2^{79.6}$	[8]
	2	76, 76	2	2^{78}	$2^{78.6}$	Sect. 5.2

Remark. Before going any further, we would like to briefly discuss the relationship between the monomial prediction and division properties. When used as detection algorithms for the key-independent sum property, both monomial prediction and the three-subset bit-based division property without unknown subsets are perfect. Originally, the division properties are defined over the multisets that the target cipher acts on, while the monomial prediction technique is fully formulated via the algebraic structure of the cipher itself. Our philosophy is that the effect of a cipher on multisets should be regarded as the manifestations of the cipher's intrinsic property, which should not be mixed with the definition of this property. A unified view naturally emerges with the monomial prediction technique for all previous division properties, since all of them are the manifestations of the properties of the ANFs of the target cipher. Finally, we would like to mention that Hebborn et al. [10] show that the three-subset bit-based division property without unknown subsets allows to decide whether or not a specific monomial appears in the ANF with the help of the *parity set* proposed in [2]. So we say that the monomial prediction and the division properties achieve the same goal through different routes.

Organization. In Sect. 2, we introduce necessary notations and preliminaries. The principle of the monomial prediction approach is established in Sect. 3. This leads to the applications to the degree evaluation in Sect. 4 and to cube attacks in Sect. 5. In Sect. 6, we establish the equivalence between the three-subset bit-based division property without unknown subsets and the monomial prediction technique, and theoretically prove that they are perfect in detecting the key-independent sum property. Also, we theoretically show that other algorithms for division properties raise no false alarms. Section 7 concludes and discusses potential future work.

2 Preliminaries

We use bold italic lowercase letters to represent bit vectors, and $\mathbf{0}$ represents a bit vector with all elements being 0. For an n-bit vector $\boldsymbol{u} \in \mathbb{F}_2^n$, its i-th coordinate is denoted by u_i, and thus $\boldsymbol{u} = (u_0, \cdots, u_{n-1})$. The complementary vector of $\boldsymbol{u}$ is denoted by $\bar{\boldsymbol{u}}$ where $u_i \oplus \bar{u}_i = 1$ for $0 \le i < n$. The Hamming weight of $\boldsymbol{u}$ is $wt(\boldsymbol{u}) = \sum_{i=0}^{n-1} u_i$. For any n-bit vectors $\boldsymbol{u}$ and $\boldsymbol{u}'$, we define $\boldsymbol{u} \succeq \boldsymbol{u}'$ if $u_i \ge u_i'$ for all i, otherwise, $\boldsymbol{u} \not\succeq \boldsymbol{u}'$. Similarly, we define $\boldsymbol{u} \preceq \boldsymbol{u}'$ if $u_i \le u_i'$ for all i, $\boldsymbol{u} \prec \boldsymbol{u}'$ if $u_i < u_i'$ for all i and $\boldsymbol{u} \succ \boldsymbol{u}'$ if $u_i > u_i'$ for all i.

Let $f : \mathbb{F}_2^n \to \mathbb{F}_2$ be a Boolean function in $\mathbb{F}_2[x_0, x_1, \ldots, x_{n-1}]/(x_0^2 - x_0, x_1^2 - x_1, \ldots, x_{n-1}^2 - x_{n-1})$ whose *algebraic normal form* (ANF) is

$$f(\boldsymbol{x}) = f(x_0, x_1, \ldots, x_{n-1}) = \bigoplus_{\boldsymbol{u} \in \mathbb{F}_2^n} a_{\boldsymbol{u}} \prod_{i=0}^{n-1} x_i^{u_i},$$

where $a_{\boldsymbol{u}} \in \mathbb{F}_2$, and

$$\boldsymbol{x}^{\boldsymbol{u}} = \pi_{\boldsymbol{u}}(\boldsymbol{x}) = \prod_{i=0}^{n-1} x_i^{u_i} \text{ with } x_i^{u_i} = \begin{cases} x_i, \text{ if } u_i = 1, \\ 1, \text{ if } u_i = 0, \end{cases}$$

is called a monomial. If the coefficient of $\boldsymbol{x}^{\boldsymbol{u}}$ in f is 1, we say $\boldsymbol{x}^{\boldsymbol{u}}$ is *contained* by f, denoted by $\boldsymbol{x}^{\boldsymbol{u}} \rightarrow f$. Otherwise, $\boldsymbol{x}^{\boldsymbol{u}}$ is not contained by f, we denote it by $\boldsymbol{x}^{\boldsymbol{u}} \nrightarrow f$. In the remaining paper, we will use $\boldsymbol{x}^{\boldsymbol{u}}$ and $\pi_{\boldsymbol{u}}(\boldsymbol{x})$ interchangeably to avoid using the awkward notation ${}_{x^{(i)}}{}^{u^{(j)}}$ when both $\boldsymbol{x}$ and $\boldsymbol{u}$ have superscripts.

Example 1. Let $f(x_0, x_1) = x_0x_1 \oplus x_0 \oplus 1$, then we have $x_0x_1 \rightarrow f$, $x_0 \rightarrow f$, $1 \rightarrow f$, and $x_1 \nrightarrow f$.

Let $\boldsymbol{y} = (y_0, \cdots, y_{m-1}) = \boldsymbol{f}(\boldsymbol{x}) = (f_0(\boldsymbol{x}), \cdots, f_{m-1}(\boldsymbol{x}))$ be a vectorial Boolean function from $\mathbb{F}_2^n$ to $\mathbb{F}_2^m$. For $\boldsymbol{v} = (v_0, v_1, \ldots, v_{m-1}) \in \mathbb{F}_2^m$, a monomial $\boldsymbol{y}^{\boldsymbol{v}}$ of $\boldsymbol{y}$ can be symbolically expressed as a polynomial of the variable $\boldsymbol{x}$:

$$\boldsymbol{y}^{\boldsymbol{v}} = \prod_{i=0}^{m-1} (f_i(\boldsymbol{x}))^{v_i} = \bigoplus_{\boldsymbol{u} \in \mathbb{F}_2^n} a_{\boldsymbol{u}} \boldsymbol{x}^{\boldsymbol{u}}, a_{\boldsymbol{u}} \in \mathbb{F}_2.$$

In the following, we show how to determine whether $\boldsymbol{x}^{\boldsymbol{u}} \rightarrow \boldsymbol{y}^{\boldsymbol{v}}$ for a given monomial $\boldsymbol{x}^{\boldsymbol{u}}$.

3 Monomial Prediction

Let $\boldsymbol{f} : \mathbb{F}_2^n \rightarrow \mathbb{F}_2^m$ be a vectorial Boolean function sending $\boldsymbol{x} = (x_0, \cdots, x_{n-1})$ to $\boldsymbol{y} = (y_0, \cdots, y_{m-1})$ with $y_i = f_i(\boldsymbol{x})$. By the *monomial prediction* we mean the problem of determining the presence or absence of a particular monomial $\boldsymbol{x}^{\boldsymbol{u}}$ in $\boldsymbol{y}^{\boldsymbol{v}}$, that is, whether $\boldsymbol{x}^{\boldsymbol{u}} \rightarrow \boldsymbol{y}^{\boldsymbol{v}}$. This is a trivial problem if the ANF of $\boldsymbol{f}$ is available. However, in the context of the symmetric-key cryptography, in most cases, the ANF of the targeted $\boldsymbol{f}$ is too complicated to be computed (or even to be stored) in practice. Typically, the only fact we know is that $\boldsymbol{f}$ is built by composition from a sequence of vectorial Boolean functions whose ANFs are known, i.e.,

$$\boldsymbol{y} = \boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{f}^{(r-1)} \circ \boldsymbol{f}^{(r-2)} \circ \cdots \circ \boldsymbol{f}^{(0)}(\boldsymbol{x}).$$

Now, how do we determine whether $\boldsymbol{x}^{\boldsymbol{u}} \rightarrow \boldsymbol{y}^{\boldsymbol{v}}$?

Let $\boldsymbol{x}^{(i)}$ and $\boldsymbol{x}^{(i+1)}$ be the input and output variables of $\boldsymbol{f}^{(i)} : \mathbb{F}_2^{n_i} \rightarrow \mathbb{F}_2^{n_{i+1}}$, respectively. Then $\boldsymbol{x}^{(i+1)} = \boldsymbol{f}^{(i)}(\boldsymbol{x}^{(i)})$ for $0 \leq i < r$, and thus $\boldsymbol{x}^{(i)}$ can be represented as a vectorial Boolean function of $\boldsymbol{x}^{(j)}$ with $j < i$:

$$\boldsymbol{x}^{(i)} = \boldsymbol{f}^{(i-1)} \circ \cdots \circ \boldsymbol{f}^{(j+1)} \circ \boldsymbol{f}^{(j)}(\boldsymbol{x}^{(j)}), \text{ for } 1 \leq i \leq r.$$

Since the ANF of $\boldsymbol{x}^{(i+1)} = \boldsymbol{f}^{(i)}(\boldsymbol{x}^{(i)})$ is available, one can determine whether $\pi_{\boldsymbol{u}^{(i)}}(\boldsymbol{x}^{(i)}) \rightarrow \pi_{\boldsymbol{u}^{(i+1)}}(\boldsymbol{x}^{(i+1)})$ for any $\boldsymbol{u}^{(i)}$ and $\boldsymbol{u}^{(i+1)}$, which gives rise to the concept of the *monomial trail.*

Definition 1 (Monomial Trail). *Let $\boldsymbol{x}^{(i+1)} = \boldsymbol{f}^{(i)}(\boldsymbol{x}^{(i)})$ for $0 \leq i < r$. We call a sequence of monomials $(\pi_{\boldsymbol{u}^{(0)}}(\boldsymbol{x}^{(0)}), \pi_{\boldsymbol{u}^{(1)}}(\boldsymbol{x}^{(1)}), \ldots, \pi_{\boldsymbol{u}^{(r)}}(\boldsymbol{x}^{(r)}))$ an r-round monomial trail connecting $\pi_{\boldsymbol{u}^{(0)}}(\boldsymbol{x}^{(0)})$ and $\pi_{\boldsymbol{u}^{(r)}}(\boldsymbol{x}^{(r)})$ with respect to the composite function $\boldsymbol{f} = \boldsymbol{f}^{(r-1)} \circ \boldsymbol{f}^{(r-2)} \circ \cdots \circ \boldsymbol{f}^{(0)}$ if*

$$\pi_{\boldsymbol{u}^{(0)}}(\boldsymbol{x}^{(0)}) \rightarrow \cdots \rightarrow \pi_{\boldsymbol{u}^{(i)}}(\boldsymbol{x}^{(i)}) \rightarrow \cdots \rightarrow \pi_{\boldsymbol{u}^{(r)}}(\boldsymbol{x}^{(r)}).$$

If there is at least one monomial trail connecting $\pi_{\boldsymbol{u}^{(0)}}(\boldsymbol{x}^{(0)})$ and $\pi_{\boldsymbol{u}^{(r)}}(\boldsymbol{x}^{(r)})$, we write $\pi_{\boldsymbol{u}^{(0)}}(\boldsymbol{x}^{(0)}) \rightsquigarrow \pi_{\boldsymbol{u}^{(r)}}(\boldsymbol{x}^{(r)})$. Otherwise, $\pi_{\boldsymbol{u}^{(0)}}(\boldsymbol{x}^{(0)}) \not\rightsquigarrow \pi_{\boldsymbol{u}^{(r)}}(\boldsymbol{x}^{(r)})$.

Note that a monomial trail is always specified with respect to a given composition sequence $\boldsymbol{f}^{(r-1)} \circ \boldsymbol{f}^{(r-2)} \circ \cdots \circ \boldsymbol{f}^{(0)}$. When this sequence is obvious from the context, we will omit it to keep the presentation concise. Also, we always assume in default that

$$\boldsymbol{x}^{(r)} = \boldsymbol{f}^{(r-1)}(\boldsymbol{x}^{(r-1)}) = \boldsymbol{f}^{(r-1)} \circ \boldsymbol{f}^{(r-2)}(\boldsymbol{x}^{(r-2)}) = \cdots = \boldsymbol{f}^{(r-1)} \circ \cdots \circ \boldsymbol{f}^{(0)}(\boldsymbol{x}^{(0)}).$$

Example 2. Let $\boldsymbol{z} = (z_0, z_1) = \boldsymbol{f}^{(1)}(y_0, y_1) = (y_0y_1, y_0 \oplus y_1)$, $\boldsymbol{y} = (y_0, y_1) = \boldsymbol{f}^{(0)}(x_0, x_1, x_2) = (x_0 \oplus x_1 \oplus x_2, x_0x_1 \oplus x_0 \oplus x_2)$ and $\boldsymbol{f} = \boldsymbol{f}^{(1)} \circ \boldsymbol{f}^{(0)}$.

Consider the monomial $(x_0, x_1, x_2)^{(1,0,0)} = x_0$. Since the ANF of $\boldsymbol{f}^{(0)}$ is available, we can compute all monomials of $\boldsymbol{y}$, i.e.,

$$(y_0, y_1)^{(0,0)} = 1, (y_0, y_1)^{(1,0)} = y_0 = \underline{x_0} \oplus x_1 \oplus x_2, (y_0, y_1)^{(0,1)} = y_1 = x_0x_1 \oplus \underline{x_0} \oplus x_2,$$
$$(y_0, y_1)^{(1,1)} = y_0y_1 = x_0x_1x_2 \oplus x_0x_1 \oplus x_1x_2 \oplus \underline{x_0} \oplus x_2.$$

Then

$$x_0 \rightarrow y_0,\ x_0 \rightarrow y_1,\ x_0 \rightarrow y_0y_1$$

are all the three monomial trails of $\boldsymbol{f}^{(0)}$ connecting x_0 and monomials of $\boldsymbol{y}$.

Similarly, we can compute all the monomials of $\boldsymbol{z}$ as follows,

$$(z_0, z_1)^{(0,0)} = 1, (z_0, z_1)^{(1,0)} = z_0 = \underline{y_0y_1}, (z_0, z_1)^{(0,1)} = z_1 = \underline{y_0} \oplus \underline{y_1},$$
$$(z_0, z_1)^{(1,1)} = z_0z_1 = 0.$$

There are three monomial trails of $\boldsymbol{f}$ connecting x_0 and monomials of $\boldsymbol{z}$:

$$x_0 \rightarrow y_0 \rightarrow z_1, \quad x_0 \rightarrow y_1 \rightarrow z_1, \quad x_0 \rightarrow y_0y_1 \rightarrow z_0.$$

Lemma 1. *$\pi_{\boldsymbol{u}^{(0)}}(\boldsymbol{x}^{(0)}) \rightsquigarrow \pi_{\boldsymbol{u}^{(r)}}(\boldsymbol{x}^{(r)})$ if $\pi_{\boldsymbol{u}^{(0)}}(\boldsymbol{x}^{(0)}) \rightarrow \pi_{\boldsymbol{u}^{(r)}}(\boldsymbol{x}^{(r)})$, and thus $\pi_{\boldsymbol{u}^{(0)}}(\boldsymbol{x}^{(0)}) \not\rightsquigarrow \pi_{\boldsymbol{u}^{(r)}}(\boldsymbol{x}^{(r)})$ implies $\pi_{\boldsymbol{u}^{(0)}}(\boldsymbol{x}^{(0)}) \nrightarrow \pi_{\boldsymbol{u}^{(r)}}(\boldsymbol{x}^{(r)})$.*

Proof. We prove it by induction on r. Assuming this lemma holds for $r < s$, we are going to show that it also holds for $r = s$. First, we expand $\pi_{\boldsymbol{u}^{(s)}}(\boldsymbol{x}^{(s)})$ on $\boldsymbol{x}^{(s-1)}$ as

$$\pi_{\boldsymbol{u}^{(s)}}(\boldsymbol{x}^{(s)}) = \bigoplus_{\pi_{\boldsymbol{u}^{(s-1)}}(\boldsymbol{x}^{(s-1)}) \rightarrow \pi_{\boldsymbol{u}^{(s)}}(\boldsymbol{x}^{(s)})} \pi_{\boldsymbol{u}^{(s-1)}}(\boldsymbol{x}^{(s-1)}).$$

Since $\pi_{\boldsymbol{u}^{(0)}}(\boldsymbol{x}^{(0)}) \rightarrow \pi_{\boldsymbol{u}^{(s)}}(\boldsymbol{x}^{(s)})$, there is at least one $\pi_{\boldsymbol{u}^{(s-1)}}(\boldsymbol{x}^{(s-1)})$ contained by $\pi_{\boldsymbol{u}^{(s)}}(\boldsymbol{x}^{(s)})$ satisfying $\pi_{\boldsymbol{u}^{(0)}}(\boldsymbol{x}^{(0)}) \rightarrow \pi_{\boldsymbol{u}^{(s-1)}}(\boldsymbol{x}^{(s-1)})$. According to our assumption, $\pi_{\boldsymbol{u}^{(0)}}(\boldsymbol{x}^{(0)}) \rightsquigarrow \pi_{\boldsymbol{u}^{(s-1)}}(\boldsymbol{x}^{(s-1)})$, then $\pi_{\boldsymbol{u}^{(0)}}(\boldsymbol{x}^{(0)}) \rightsquigarrow \pi_{\boldsymbol{u}^{(s)}}(\boldsymbol{x}^{(s)})$. □

According to Lemma 1, $\pi_{\boldsymbol{u}^{(0)}}(\boldsymbol{x}^{(0)}) \rightarrow \pi_{\boldsymbol{u}^{(r)}}(\boldsymbol{x}^{(r)})$ is sufficient for $\pi_{\boldsymbol{u}^{(0)}}(\boldsymbol{x}^{(0)}) \rightsquigarrow \pi_{\boldsymbol{u}^{(r)}}(\boldsymbol{x}^{(r)})$. However, the conversion is not true in general. Considering Example 2, although $x_0 \rightsquigarrow z_1$, we have $x_0 \nrightarrow z_1$ since

$$z_1 = y_0 \oplus y_1 = \underline{x_0} \oplus x_1 \oplus x_2 \oplus x_0x_1 \oplus \underline{x_0} \oplus x_2 = x_0x_1 \oplus x_1.$$

The reason is that two x_0's (underlined in the above equation) cancel each other. In the following, we will demonstrate that whether $\pi_{\boldsymbol{u}^{(0)}}(\boldsymbol{x}^{(0)}) \rightarrow \pi_{\boldsymbol{u}^{(r)}}(\boldsymbol{x}^{(r)})$ is determined by the number of monomial trails connecting them rather than the existence of the monomial trail, which raises the definition below.

Definition 2 (Monomial Hull). *For $\boldsymbol{f}$ with a specific composition sequence, the monomial hull of $\pi_{\boldsymbol{u}^{(0)}}(\boldsymbol{x}^{(0)})$ and $\pi_{\boldsymbol{u}^{(r)}}(\boldsymbol{x}^{(r)})$, denoted by $\pi_{\boldsymbol{u}^{(0)}}(\boldsymbol{x}^{(0)}) \bowtie \pi_{\boldsymbol{u}^{(r)}}(\boldsymbol{x}^{(r)})$, is the set of all monomial trails connecting them. The number of trails in the monomial hull is called the* ***size*** *of the hull and is denoted by $|\pi_{\boldsymbol{u}^{(0)}}(\boldsymbol{x}^{(0)}) \bowtie \pi_{\boldsymbol{u}^{(r)}}(\boldsymbol{x}^{(r)})|$.*

Example 3. Consider Example 2, the monomial hull of x_0 and z_1 is the set

$$x_0 \bowtie z_1 = \{x_0 \rightarrow y_0 \rightarrow z_1, x_0 \rightarrow y_1 \rightarrow z_1\}.$$

Thus the size of $x_0 \bowtie z_1$ is 2. Furthermore, since $x_0 \not\rightsquigarrow z_0z_1$, $x_0 \bowtie z_0z_1 = \emptyset$ and $|x_0 \bowtie z_0z_1| = 0$.

For $i \geq 1$, if $\pi_{\boldsymbol{u}^{(0)}}(\boldsymbol{x}^{(0)}) \rightsquigarrow \pi_{\boldsymbol{u}^{(i)}}(\boldsymbol{x}^{(i)})$, $|\pi_{\boldsymbol{u}^{(0)}}(\boldsymbol{x}^{(0)}) \bowtie \pi_{\boldsymbol{u}^{(i)}}(\boldsymbol{x}^{(i)})|$ can be calculated recursively as follows,

Lemma 2. *For $i \geq 1$, if $\pi_{\boldsymbol{u}^{(0)}}(\boldsymbol{x}^{(0)}) \rightsquigarrow \pi_{\boldsymbol{u}^{(i)}}(\boldsymbol{x}^{(i)})$,*

$$|\pi_{\boldsymbol{u}^{(0)}}(\boldsymbol{x}^{(0)}) \bowtie \pi_{\boldsymbol{u}^{(i)}}(\boldsymbol{x}^{(i)})| = \begin{cases} 1, & i = 1, \\ \displaystyle\sum_{\substack{\pi_{\boldsymbol{u}^{(i-1)}}(\boldsymbol{x}^{(i-1)}) \\ \rightarrow \pi_{\boldsymbol{u}^{(i)}}(\boldsymbol{x}^{(i)})}} |\pi_{\boldsymbol{u}^{(0)}}(\boldsymbol{x}^{(0)}) \bowtie \pi_{\boldsymbol{u}^{(i-1)}}(\boldsymbol{x}^{(i-1)})|, & i \geq 2. \end{cases}$$

The time has come to address the monomial prediction problem we mentioned at the beginning of this section.

Proposition 1. *$\pi_{\boldsymbol{u}^{(0)}}(\boldsymbol{x}^{(0)}) \rightarrow \pi_{\boldsymbol{u}^{(r)}}(\boldsymbol{x}^{(r)})$ if and only if $|\pi_{\boldsymbol{u}^{(0)}}(\boldsymbol{x}^{(0)}) \bowtie \pi_{\boldsymbol{u}^{(r)}}(\boldsymbol{x}^{(r)})|$ is odd.*

Proof. We prove it by induction on r. Assuming this proposition holds for $r < s$, we are going to show that it also holds for $r = s$. First, we expand $\pi_{\boldsymbol{u}^{(s)}}(\boldsymbol{x}^{(s)})$ on $\boldsymbol{x}^{(s-1)}$ as

$$\pi_{\boldsymbol{u}^{(s)}}(\boldsymbol{x}^{(s)}) = \bigoplus_{\pi_{\boldsymbol{u}^{(s-1)}}(\boldsymbol{x}^{(s-1)}) \rightarrow \pi_{\boldsymbol{u}^{(s)}}(\boldsymbol{x}^{(s)})} \pi_{\boldsymbol{u}^{(s-1)}}(\boldsymbol{x}^{(s-1)}).$$

Consequently, we have

$$|\pi_{\boldsymbol{u}^{(0)}}(\boldsymbol{x}^{(0)}) \bowtie \pi_{\boldsymbol{u}^{(s)}}(\boldsymbol{x}^{(s)})| = \sum_{\substack{\pi_{\boldsymbol{u}^{(s-1)}}(\boldsymbol{x}^{(s-1)}) \\ \rightarrow \pi_{\boldsymbol{u}^{(s)}}(\boldsymbol{x}^{(s)})}} |\pi_{\boldsymbol{u}^{(0)}}(\boldsymbol{x}^{(0)}) \bowtie \pi_{\boldsymbol{u}^{(s-1)}}(\boldsymbol{x}^{(s-1)})|.$$

Moreover, $\pi_{\boldsymbol{u}^{(0)}}(\boldsymbol{x}^{(0)}) \rightarrow \pi_{\boldsymbol{u}^{(s)}}(\boldsymbol{x}^{(s)})$ if and only if there are odd number of $\pi_{\boldsymbol{u}^{(s-1)}}(\boldsymbol{x}^{(s-1)})$ contained by $\pi_{\boldsymbol{u}^{(s)}}(\boldsymbol{x}^{(s)})$ such that $\pi_{\boldsymbol{u}^{(0)}}(\boldsymbol{x}^{(0)}) \rightarrow \pi_{\boldsymbol{u}^{(s-1)}}(\boldsymbol{x}^{(s-1)})$, or equivalently, according to the induction hypothesis we made at the beginning, there are odd number of $\pi_{\boldsymbol{u}^{(s-1)}}(\boldsymbol{x}^{(s-1)})$ contained by $\pi_{\boldsymbol{u}^{(s)}}(\boldsymbol{x}^{(s)})$ such that $|\pi_{\boldsymbol{u}^{(0)}}(\boldsymbol{x}^{(0)}) \bowtie \pi_{\boldsymbol{u}^{(s-1)}}(\boldsymbol{x}^{(s-1)})|$ is odd. Finally, Proposition 1 is true for $r = s$ since $|\pi_{\boldsymbol{u}^{(0)}}(\boldsymbol{x}^{(0)}) \bowtie \pi_{\boldsymbol{u}^{(s)}}(\boldsymbol{x}^{(s)})|$ is odd if and only if

$$\sum_{\pi_{\boldsymbol{u}^{(s-1)}}(\boldsymbol{x}^{(s-1)}) \rightarrow \pi_{\boldsymbol{u}^{(s)}}(\boldsymbol{x}^{(s)})} |\pi_{\boldsymbol{u}^{(0)}}(\boldsymbol{x}^{(0)}) \bowtie \pi_{\boldsymbol{u}^{(s-1)}}(\boldsymbol{x}^{(s-1)})| \text{ is odd.}$$

□

3.1 Derived Function

When applying the monomial prediction technique to cryptanalysis, we may consider functions that are derived from a vectorial Boolean function $\boldsymbol{f}$ by fixing some variables of $\boldsymbol{f}$ to known constants. In this case, the derived function has fewer variables than the original function $\boldsymbol{f}$. Also, the remaining variables are not treated equally. Some of them are public (IV bits, plaintext bits, tweak bits, etc.), while some of them are secret (key bits). To highlight the semantic difference of the variables and distinguish between the variables fixed to 0 and those fixed to 1, we introduce the notion of *variable masks*. Together with the original function $\boldsymbol{f}$, these masks completely determine the derived function, and tells us which variables of the derived function are public and which are secret.

Remark. The only purpose of introducing the concept of the derived function is to have a unified approach to specify the functions to which our techniques are applied. It has no theoretical significance and the readers who do not care about the details of the attacks on concrete targets can safely skip this part to avoid being overloaded by unnecessary notations. Actually, skipping this part is encouraged and the readers can look back when necessary.

Variable Masks and Derived Function. Let $\boldsymbol{\Gamma^0}$, $\boldsymbol{\Gamma^1}$, $\boldsymbol{\Gamma^p}$, and $\boldsymbol{\Gamma^s} \in \mathbb{F}_2^n$ be constant vectors such that $\{0 \leq i < n : \Gamma_i^0 = 1\}$, $\{0 \leq i < n : \Gamma_i^1 = 1\}$, $\{0 \leq i < n : \Gamma_i^p = 1\}$, and $\{0 \leq i < n : \Gamma_i^s = 1\}$ form a partition of $\{0, \cdots, n-1\}$, which are called variable masks. For a vectorial Boolean function $\boldsymbol{f}(\boldsymbol{x})$ from $\mathbb{F}_2^n$ to $\mathbb{F}_2^m$, we can derive a new function $\boldsymbol{f}_d$ from $\boldsymbol{f}$ with the variable masks by setting certain variables of $\boldsymbol{f}$ to constants according to the following rule for $i \in \{0, 1, \cdots, n-1\}$:

$$\begin{cases} x_i \leftarrow 0, \text{ if } \Gamma_i^0 = 1, \\ x_i \leftarrow 1, \text{ if } \Gamma_i^1 = 1. \end{cases}$$

The remaining x_i's are still treated as variables but with different access permissions: x_i's with $\Gamma_i^p = 1$ are public variables and can be manipulated by the attackers, while x_i's with $\Gamma_i^s = 1$ are secret variables. Although in practice secret variables typically represent secret key bits and are actually fixed to unknown constants, in our framework we still regard them as symbolic objects rather than constants. The concept of the derived function should be best understood by a concrete example.

Example 4. For $\boldsymbol{y} = \boldsymbol{f}(x_0, x_1, x_2, x_3, k_0, k_1, k_2, k_3)$ where x_0, x_1, x_2, x_3 are four public input bits and k_0, k_1, k_2, k_3 are four secret input bits. If we fix x_0 to 0 and x_1 to 1, the resulting function mapping $(0, 1, x_2, x_3, k_0, k_1, k_2, k_3)$ to

$$\boldsymbol{f}(0, 1, x_2, x_3, k_0, k_1, k_2, k_3)$$

is a derived function from $\boldsymbol{f}$ with the following variable masks

$$\boldsymbol{\Gamma}^0 = (1, 0, 0, 0, 0, 0, 0, 0), \quad \boldsymbol{\Gamma}^1 = (0, 1, 0, 0, 0, 0, 0, 0),$$
$$\boldsymbol{\Gamma}^p = (0, 0, 1, 1, 0, 0, 0, 0), \quad \boldsymbol{\Gamma}^s = (0, 0, 0, 0, 1, 1, 1, 1).$$

In the following sections, we typically first give a function $\boldsymbol{f}$ which can be directly obtained from the description of the targeted cipher, and then we specify the associated variable masks. Finally, the techniques presented in this work are applied to the corresponding derived function.

In the case of $\boldsymbol{f}_d$, we should note $\boldsymbol{x}^{\boldsymbol{v}} \equiv 1$ for any $\boldsymbol{v} \preceq \boldsymbol{\Gamma}^1$, then $\boldsymbol{x}^{\boldsymbol{u} \oplus \boldsymbol{v}} = \boldsymbol{x}^{\boldsymbol{u}} \cdot \boldsymbol{x}^{\boldsymbol{v}} = \boldsymbol{x}^{\boldsymbol{u}}$ for any $\boldsymbol{v} \preceq \boldsymbol{\Gamma}^1$ and the Proposition 1 can be converted to the following proposition.

Proposition 2. *Let $\boldsymbol{f}_d$ be the derived function of $\boldsymbol{f}$ with $\boldsymbol{\Gamma}^0, \boldsymbol{\Gamma}^1, \boldsymbol{\Gamma}^p, \boldsymbol{\Gamma}^s$. For $\boldsymbol{x}^{(r)} = \boldsymbol{f}_d(\boldsymbol{x}^{(0)})$ and $\boldsymbol{u}^{(0)} \preceq \boldsymbol{\Gamma}^p \oplus \boldsymbol{\Gamma}^s$, $\pi_{\boldsymbol{u}^{(0)}}(\boldsymbol{x}^{(0)}) \rightarrow \pi_{\boldsymbol{u}^{(r)}}(\boldsymbol{x}^{(r)})$ if and only if*

$$\sum_{\boldsymbol{v} \preceq \boldsymbol{\Gamma}^1} |\pi_{\boldsymbol{u}^{(0)} \oplus \boldsymbol{v}}(\boldsymbol{x}^{(0)}) \bowtie \pi_{\boldsymbol{u}^{(r)}}(\boldsymbol{x}^{(r)})| \bmod 2 = 1.$$

4 Application I: Degree Evaluation

Since the algebraic degree of a symmetric-key primitive significantly affects its security against cryptanalytic techniques such as algebraic attacks [20], higher-order differential attacks [15,17], interpolation attacks [14], and integral attacks [6,16], methods and tools for degree evaluation have been an important topic in the community all along. To put our approach into perspective, we highlight several important works in this line of research. At EUROCRYPT 2002, Canteaut and Videau developed a method for upper bounding the algebraic degree of composite functions [5], which was improved by Boura et al. [3] at FSE 2011. In [1], the authors identified a simple closed formula bounding the number of rounds necessary to achieve full degree for the block ciphers with secret components. At CRYPTO 2017, Liu presented a general framework known as *numeric mapping*, which is exclusively used for estimating the algebraic degrees of the cryptosystems based on the nonlinear feedback shift register (NFSR) [18].

Another approach for the degree evaluation is based on the division property. The accuracy of this approach is determined by the accuracy of the "propagation rules" of the underlying detection algorithms for division properties. When the detection algorithm is *perfect* (The meaning of perfect will be more concrete in Sect. 6), its estimation is exact. In the following, we show that the monomial prediction technique achieves this exactness.

4.1 Compute Exact Algebraic Degree of a Boolean Function

The algebraic degree of a Boolean function f is defined as follows,

$$\deg(f) = \max_{\pi_{\boldsymbol{u}^{(0)}}(\boldsymbol{x}^{(0)}) \rightarrow f} wt(\boldsymbol{u}^{(0)}). \tag{1}$$

To determine the algebraic degree of f, we only need to prove the existence of a monomial $\pi_{\boldsymbol{u}^{(0)}}(\boldsymbol{x}^{(0)})$ such that $\pi_{\boldsymbol{u}'}(\boldsymbol{x}^{(0)}) \nrightarrow f$ for any $\boldsymbol{u}'$ with $wt(\boldsymbol{u}') > d$, which can be done in two steps:

1. Find a monomial $\pi_{\boldsymbol{u}^{(0)}}(\boldsymbol{x}^{(0)}) \rightsquigarrow f$ with $wt(\boldsymbol{u}) = d$ and prove $\pi_{\boldsymbol{u}'}(\boldsymbol{x}^{(0)}) \not\rightsquigarrow f$ for any $wt(\boldsymbol{u}') > d$.
2. Compute $|\pi_{\boldsymbol{u}^{(0)}}(\boldsymbol{x}^{(0)}) \bowtie f|$ to confirm the presence of $\pi_{\boldsymbol{u}^{(0)}}(\boldsymbol{x}^{(0)})$, if the value is odd, then $\deg(f) = d$, else, we need to repeat the process until we find a desired monomial of f.

The Mixed Integer Linear Programming (MILP) approach has been extensively used to probe the structure of Boolean functions in previous works such as [9,22,26,28–31]. In this work, we also employ the MILP-based approach to search for the monomials of f. In this MILP model, the objective function of the model is to maximize $wt(\boldsymbol{u}^{(0)})$ according to Eq. (1). One solution of the MILP model is a sequence of $(\boldsymbol{u}^{(0)}, \boldsymbol{u}^{(1)}, \ldots, \boldsymbol{u}^{(r)})$[1], such that

$$\pi_{\boldsymbol{u}^{(0)}}(\boldsymbol{x}^{(0)}) \rightarrow \pi_{\boldsymbol{u}^{(1)}}(\boldsymbol{x}^{(1)}) \rightarrow \cdots \rightarrow \pi_{\boldsymbol{u}^{(r)}}(\boldsymbol{x}^{(r)}).$$

To confirm the presence of $\pi_{\boldsymbol{u}^{(0)}}(\boldsymbol{x}^{(0)})$ as in the above Step 2, we use the `PoolSearchMode` of Gurobi to compute $|\pi_{\boldsymbol{u}^{(0)}}(\boldsymbol{x}^{(0)}) \bowtie f|$.

`PoolSearchMode` of Gurobi. To judge whether the size of a monomial hull is an odd number, we frequently need to find all solutions of a MILP model. Following Hao et al.'s work at EUROCRYPT 2020 [9], we also employ the `PoolSearchMode` of Gurobi[2] to perform solution enumerations. The `PoolSearchMode` is a mode implemented by Gurobi to systematically search for multiple solutions. Let $\mathcal{M}$ be a MILP model, we use

$$\mathcal{M}.\texttt{PoolSearchMode} \leftarrow 1$$

to signal that the `PoolSearchMode` is turned on. All the source codes are available at https://github.com/hukaisdu/MonomialPrediction.

4.2 Application to Trivium

Specification of Trivium. Trivium [4] is an NFSR-based stream cipher with a 288-bit internal state $\boldsymbol{x} = (x_0, x_1, \ldots, x_{287})$ divided into three registers (denoted as Reg 0, Reg 1 and Reg 2 in Fig. 1). The 80-bit secret key K is loaded to the first register (Reg 0), and the 80-bit initialization vector IV is loaded to the second register. The other bits of the three registers are set to 0 except the last three bits of the third register. Namely, we have

$$\begin{aligned}
(x_0, x_1, \ldots, x_{92}) &\leftarrow (K[0], K[1], \ldots, K[79], 0, \ldots, 0),\\
(x_{93}, x_{94}, \ldots, x_{176}) &\leftarrow (IV[0], IV[2], \ldots, IV[79], 0, \ldots, 0),\\
(x_{177}, x_{178}, \ldots, x_{287}) &\leftarrow (0, 0, \ldots, 0, 1, 1, 1).
\end{aligned}$$

[1] In this section, we focus on the Boolean function, so $\boldsymbol{u}^{(r)}$ is always a unit vector.
[2] https://www.gurobi.com.

Let $h : \mathbb{F}_2^5 \to \mathbb{F}_2$ be a Boolean function such that $h(\alpha_0, \alpha_1, \alpha_2, \alpha_3, \alpha_4) = \alpha_0 \oplus \alpha_1\alpha_2 \oplus \alpha_3 \oplus \alpha_4$. The pseudo code of the update function is given by

$$\begin{aligned}
t_1 &\leftarrow h(x_{65}, x_{90}, x_{91}, x_{92}, x_{170}) = x_{65} \oplus x_{90}x_{91} \oplus x_{92} \oplus x_{170},\\
t_2 &\leftarrow h(x_{161}, x_{174}, x_{175}, x_{176}, x_{263}) = x_{161} \oplus_{174} x_{175} \oplus x_{176} \oplus x_{263},\\
t_3 &\leftarrow h(x_{242}, x_{285}, x_{286}, x_{287}, x_{68}) = x_{242} \oplus x_{285}x_{286} \oplus x_{287} \oplus x_{68}.
\end{aligned}$$

The state of the next clock is computed as

$$\begin{aligned}
(x_0, x_1, \dots, x_{92}) &\leftarrow (t_3, x_0, \dots, x_{91}),\\
(x_{93}, x_{94}, \dots, x_{176}) &\leftarrow (t_1, x_{93}, \dots, x_{175}),\\
(x_{177}, x_{178}, \dots, x_{287}) &\leftarrow (t_2, x_{177}, \dots, x_{286}).
\end{aligned}$$

During the initialization, the state is updated 1152 times without producing any output. After the initialization, one bit key is produced per application of the update function by the key stream generation function $g : \mathbb{F}_2^{288} \to \mathbb{F}_2$ as

$$z \ \leftarrow g(x_0, x_1, \dots, x_{287}) = x_{65} \oplus x_{92} \oplus x_{161} \oplus x_{176} \oplus x_{242} \oplus x_{287}.$$

MILP Model for a Monomial Trail of Trivium. Let $\boldsymbol{x}^{(0)}$ denote the initial state of Trivium and $\boldsymbol{x}^{(i+1)}$ denote the state after the i-th update function $\boldsymbol{f}^{(i)}$. The output bit after r-round Trivium[3] z_r is a Boolean function of $\boldsymbol{x}^{(0)}$ which is denoted by $z_r = f(\boldsymbol{x}^{(0)})$. Naturally, f is the composition of the update functions and the key stream generation function as

$$\begin{aligned}
z_r = f(\boldsymbol{x}^{(0)}) &= g \circ \boldsymbol{f}^{(r-1)} \circ \boldsymbol{f}^{(r-2)} \circ \cdots \circ \boldsymbol{f}^{(0)}(\boldsymbol{x}^{(0)})\\
&= g(\boldsymbol{x}^{(r)}) = x_{65}^{(r)} \oplus x_{92}^{(r)} \oplus x_{161}^{(r)} \oplus x_{176}^{(r)} \oplus x_{242}^{(r)} \oplus x_{287}^{(r)}. \qquad (2)
\end{aligned}$$

To construct the MILP model for the monomial trail of Trivium, we should study the ANFs of $\boldsymbol{f}^{(i)}$ and g and model the monomial trail locally for them.

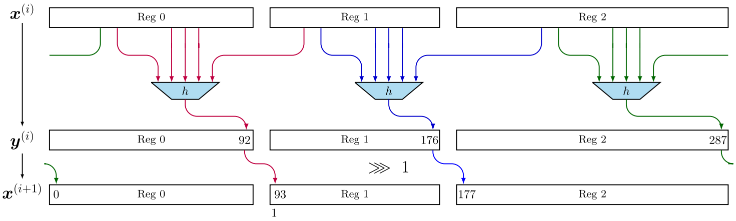

Fig. 1. The illustration of $\boldsymbol{f}^{(i)}$. In the first phase, if $j \notin \{92, 176, 287\}$, $y_j^{(i)} = x_j^{(i)}$. In the second phase, $x_{(j+1) \bmod 288}^{(i+1)} = y_j^{(i)}$.

[3] When saying (reduced) r-round of Trivium, we mean the update function $\boldsymbol{f}$ is called r times and then the key stream generation function g is finally performed.

According to Fig. 1, $\boldsymbol{f}^{(i)}$ can be represented by parallel bit-permutations and three H functions such as

$$x_{j+1 \bmod 288}^{(i+1)} = x_j^{(i)}, \text{ if } j \notin \{65{,}90{,}91{,}92{,}170{,}161{,}174{,}175{,}176{,}263{,}242{,}285{,}286{,}287{,}68\}, \tag{3}$$

$$(x_{66}^{(i+1)}, x_{91}^{(i+1)}, x_{92}^{(i+1)}, x_{93}^{(i+1)}, x_{171}^{(i+1)}) = H(x_{65}^{(i)}, x_{90}^{(i)}, x_{91}^{(i)}, x_{92}^{(i)}, x_{170}^{(i)}) \tag{4}$$

$$(x_{162}^{(i+1)}, x_{175}^{(i+1)}, x_{176}^{(i+1)}, x_{177}^{(i+1)}, x_{264}^{(i+1)} = H(x_{161}^{(i)}, x_{174}^{(i)}, x_{175}^{(i)}, x_{176}^{(i)}, x_{263}^{(i)}) \tag{5}$$

$$(x_{243}^{(i+1)}, x_{286}^{(i+1)}, x_{287}^{(i+1)}, x_{0}^{(i+1)}, x_{69}^{(i+1)}) = H(x_{242}^{(i)}, x_{285}^{(i)}, x_{286}^{(i)}, x_{287}^{(i)}, x_{68}^{(i)}) \tag{6}$$

where $H : \mathbb{F}_2^5 \to \mathbb{F}_2^5$ defined as follows,

$$(\beta_0, \beta_1, \beta_2, \beta_3, \beta_4) = H(\alpha_0, \alpha_1, \alpha_2, \alpha_3, \alpha_4) = (\alpha_0, \alpha_1, \alpha_2, \alpha_0 \oplus \alpha_1\alpha_2 \oplus \alpha_3 \oplus \alpha_4, \alpha_4).$$

H can be decomposed into a sequence of smaller functions such as COPY, AND and XOR, which is shown in Fig. 2.

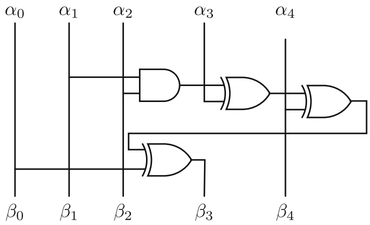

Fig. 2. The decomposition of H function by COPY, AND and XOR.

MILP Model for the Monomial Trail of $\boldsymbol{f}^{(i)}$. The operations in Eq. (3) are simple bit-permutations which can be handled by directly changing the positions of the variables, thus no inequalities are required for this condition. To model H function, we generate inequalities to model the monomial trials of `COPY`, `AND` and `XOR`. For `COPY`, consider $x \xrightarrow{\texttt{COPY}} (x, x)$ where x is a bit variable, we have

$$\begin{cases} x^0(=1) \to x^0 \cdot x^0(=1), & x^0(=1) \nrightarrow x^0 \cdot x^1(=x) \\ x^0(=1) \nrightarrow x^1 \cdot x^0(=x), & x^0(=1) \nrightarrow x^1 \cdot x^1(=x) \\ x^1(=x) \nrightarrow x^0 \cdot x^0(=1), & x^1(=x) \to x^0 \cdot x^1(=x) \\ x^1(=x) \to x^1 \cdot x^0(=x), & x^1(=x) \to x^1 \cdot x^1(=x) \end{cases}.$$

Then there are four valid monomial trails of `COPY`, i.e., (0, 0, 0), (1, 0, 1), (1, 1, 0) and (1, 1, 1). Similarly, `AND` has two monomial trials (0, 0, 0) and (1, 1, 1), while `XOR` has three monomial trials (0, 0, 0), (1, 0, 1) and (0, 1, 1).

To generate inequalities for monomial trails of each function, we follow Sun et al.'s approach in [23] to derive linear inequalities by Sage[4] and then use the

[4] https://www.sagemath.org.

greedy algorithm to simplify them. At last, a set of 15 inequalities $\mathcal{L}$ with 5 auxiliary variables (given in Appendix A of [11]) is sufficient to describe the H function. Thus we need 45 linear inequalities and 15 auxiliary variables to model $\boldsymbol{f}^{(i)}$. In Appendix B (Ref. [11]), we provide an alternative method to describe the monomial trails of H with less inequalities, where H is treated as a whole. Note that Proposition 1 implies that the decomposition with different granularity levels of the target Boolean function will not affect the parity of the number of the monomial trails of the Boolean function.

MILP Model for the Monomial trail of g. Since g is a simple Boolean function that contains 6 monomials (Eq. (2)), a set of simple constraints as

$$\begin{cases} u_{65}^{(r)} + u_{92}^{(r)} + u_{161}^{(r)} + u_{176}^{(r)} + u_{242}^{(r)} + u_{287}^{(r)} = 1, \\ u_j^{(r)} = 0, \text{ if } j \notin \{65, 92, 161, 176, 242, 287\}. \end{cases} \tag{7}$$

will complete our modeling.

In Algorithm 1, we demonstrate how to generate the MILP model for Trivium, where $\mathcal{L}$ represents the inequalities for the model of H. Note in some cases we may want to manipulate the first (e.g., line 16 of Algorithm 2) and last terms (e.g., line 11 of Algorithm 3) of the monomial trail. Then the MILP model in Algorithm 1 excludes the model of g, instead the variables representing the

```
Algorithm 1: $(\mathcal{M}, \boldsymbol{u}^{(0)}, \boldsymbol{u}^{(r)})$ = GenerateTriviumModel($r$)
Input: $r$, the targeted number of rounds of Trivium
Output: The MILP model $\mathcal{M}$ for $r$-round Trivium and the MILP variables
        representing the initial state $\boldsymbol{u}^{(0)}$
Declare an empty MILP model $\mathcal{M}$;
$\mathcal{M}.var \leftarrow u_0^{(0)}, u_1^{(0)}, \ldots, u_{287}^{(0)}$;
$\mathcal{M}.var \leftarrow u_0, u_1, \ldots, u_{287}$;
$\boldsymbol{u} \leftarrow \boldsymbol{u}^{(0)}$;
for $i = 0; i < r; i \leftarrow i + 1$ do
    $\mathcal{M}.var \leftarrow v_{65}, v_{90}, v_{91}, v_{92}, v_{170}, w_0, w_1, w_2, w_4, t$;
    $\mathcal{M}.con \leftarrow \mathcal{L}(u_{65}, u_{90}, u_{91}, u_{92}, u_{170}, v_{65}, v_{90}, v_{91}, v_{92}, v_{170}, w_0, w_1, w_2, w_4, t)$;
    $u_i \leftarrow v_i, i \in \{65, 90, 91, 92, 170\}$;
    $\mathcal{M}.var \leftarrow v_{161}, v_{174}, v_{175}, v_{176}, v_{263}, w_0, w_1, w_2, w_4, t$;
    $\mathcal{M}.con \leftarrow$
      $\mathcal{L}(u_{161}, u_{174}, u_{175}, u_{176}, u_{263}, v_{161}, v_{174}, v_{175}, v_{176}, v_{263}, w_0, w_1, w_2, w_4, t)$;
    $u_i \leftarrow v_i, i \in \{161, 174, 175, 176, 263\}$;
    $\mathcal{M}.var \leftarrow v_{242}, v_{285}, v_{286}, v_{287}, v_{68}, w_0, w_1, w_2, w_4, t$;
    $\mathcal{M}.con \leftarrow$
      $\mathcal{L}(u_{242}, u_{285}, u_{286}, u_{287}, u_{68}, v_{242}, v_{285}, v_{286}, v_{287}, v_{68}, w_0, w_1, w_2, w_4, t)$;
    $u_i \leftarrow v_i, i \in \{242, 285, 286, 287, 68\}$;
    $u_{i+1 \bmod 288} \leftarrow u_i$;
$\boldsymbol{u}^{(r)} \leftarrow \boldsymbol{u}$;
return $\mathcal{M}, \boldsymbol{u}^{(0)}, \boldsymbol{u}^{(r)}$;
```

first monomial $\pi_{\boldsymbol{u}^{(0)}}(\boldsymbol{x}^{(0)})$ and the last monomial $\pi_{\boldsymbol{u}^{(r)}}(\boldsymbol{x}^{(r)})$ are also returned in order for later usage.

Degree of Trivium. The output bit $z_r = f(\boldsymbol{x}^{(0)})$ after r-round Trivium is a Boolean function of the initial state $\boldsymbol{x}^{(0)}$. If we regard the IV bits as public variables and the key bits as secret variables, the initial setup of the state implies the following derived function with four variable masks $\boldsymbol{\Gamma}^0, \boldsymbol{\Gamma}^1, \boldsymbol{\Gamma}^p, \boldsymbol{\Gamma}^s$:

$$\Gamma_i^0 = \begin{cases} 1, & \text{if } 80 \le i \le 92 \text{ or } 173 \le i \le 284, \\ 0, & \text{otherwise.} \end{cases} \qquad \Gamma_i^1 = \begin{cases} 1, & \text{if } 285 \le i \le 287, \\ 0, & \text{otherwise.} \end{cases}$$

$$\Gamma_i^p = \begin{cases} 1, & \text{if } 93 \le i \le 172, \\ 0, & \text{otherwise.} \end{cases} \qquad \Gamma_i^s = \begin{cases} 1, & \text{if } 0 \le i \le 79, \\ 0, & \text{otherwise.} \end{cases}$$

In accordance, the derived function and its variable masks can be used to modify the algebraic degree expression given in Eq. (1), therefore the algebraic degree of z_r can be computed as

$$\deg(z_r) = \max_{\substack{\boldsymbol{u}^{(0)} \preceq \boldsymbol{\Gamma}^p \oplus \boldsymbol{\Gamma}^s \\ \pi_{\boldsymbol{u}^{(0)}}(\boldsymbol{x}^{(0)}) \to z_r}} \Big\{ \sum_{\boldsymbol{\Gamma}_i^p = 1} u_i^{(0)} \Big\} = \max_{\substack{\boldsymbol{u}^{(0)} \preceq \boldsymbol{\Gamma}^p \oplus \boldsymbol{\Gamma}^s \\ \pi_{\boldsymbol{u}^{(0)}}(\boldsymbol{x}^{(0)}) \to z_r}} \Big\{ \sum_{93 \le i \le 172} u_i^{(0)} \Big\}.$$

By calling Algorithm 1, Algorithm 2 finds the monomial with the potential maximum degree satisfying $\pi_{\boldsymbol{u}^{(0)}}(\boldsymbol{x}^{(0)}) \rightsquigarrow z_r$. Thereafter, $|\pi_{\boldsymbol{u}^{(0)}}(\boldsymbol{x}^{(0)}) \bowtie z_r|$ is computed under the `PoolSearchMode` to determine if $\pi_{\boldsymbol{u}^{(0)}}(\boldsymbol{x}^{(0)}) \to z_r$ holds. Once $\pi_{\boldsymbol{u}^{(0)}}(\boldsymbol{x}^{(0)}) \to z_r$ is confirmed, we derive the exact algebraic degree of r-round Trivium.

Our Results. With the help of the monomial prediction we are able to evaluate the exact algebraic degree of Trivium up to 834 rounds and the results are listed in Table 5 in Appendix E (Ref. [11]). Interestingly, for the first time, we notice a counter-intuitive phenomenon that the algebraic degree of Trivium is not monotonously increasing with rounds. For example, the degrees of 806-, 807- and 808-round Trivium are 69, 71, 70, respectively. It implies that some monomials with the maximum degree are canceled in the subsequent round. Such degree drops are highlighted in Table 5.

A comparison of monomial prediction and the numeric mapping technique for upper bounding the degree of NFSR ciphers [18] is illustrated in Fig. 3. As the number of iterated rounds gets larger, the gap between the upper bound and the exact degree becomes more significant. For the degree of the 793-round Trivium, the numeric mapping technique gives an upper bound of 79, while the monomial prediction method tells us that the exact degree is only 67.

Algorithm 2: $\deg = \mathsf{SearchDegree}(r)$

```
Input: $r$, the targeted number of rounds of Trivium
Output: The degree of $r$-round Trivium
/* Search For $\pi_{\boldsymbol{u}^{(0)}}(\boldsymbol{x}^{(0)}) \rightsquigarrow f$ */
$(\mathcal{M}_0, \boldsymbol{u}^{(0)}, \boldsymbol{u}^{(r)}) \leftarrow \mathsf{GenerateTriviumModel}(r)$
for $i = 0; i < 288; i \leftarrow i + 1$ do
    if $\Gamma_i^0$ is 1 then
        $u_i^{(0)} \leftarrow 0$
for $i = 0; i < 288; i \leftarrow i + 1$ do
    if $i \notin \{65, 92, 161, 176, 242, 287\}$ then
        $\mathcal{M}_0.con \leftarrow u_i^{(r)} = 0;$
$\mathcal{M}_0.con \leftarrow u_{65}^{(r)} + u_{92}^{(r)} + u_{161}^{(r)} + u_{176}^{(r)} + u_{242}^{(r)} + u_{287}^{(r)} = 1;$
$\mathcal{M}_0.obj \leftarrow \max(u_{93}^{(0)} + u_{94}^{(0)} + \cdots + u_{172}^{(0)});$
while true do
    $\mathcal{M}_0.optimize();$
    if $\mathcal{M}_0.status$ is OPTIMAL then
        /* Compute $|\pi_{\boldsymbol{u}^{(0)}}(\boldsymbol{x}^{(0)}) \bowtie f|$ */
        $(\mathcal{M}_1, \boldsymbol{u'}^{(0)}, \boldsymbol{u'}^{(r)}) \leftarrow \mathsf{GenerateTriviumModel}(r)$
        $\mathcal{M}_1$.SolutionPoolMode $\leftarrow 1;$
        for $i = 0; i < 288; i \leftarrow i + 1$ do
            $u_i'^{(0)} \leftarrow u_i^{(0)}.val;$
        for $i = 0; i < 288; i \leftarrow i + 1$ do
            if $i \notin \{65, 92, 161, 176, 242, 287\}$ then
                $\mathcal{M}_1.con \leftarrow u_i'^{(r)} = 0;$
        $\mathcal{M}_1.con \leftarrow u_{65}'^{(r)} + u_{92}'^{(r)} + u_{161}'^{(r)} + u_{176}'^{(r)} + u_{242}'^{(r)} + u_{287}'^{(r)} = 1;$
        $\mathcal{M}_1.optimize();$
        if $\mathcal{M}_1.status$ is OPTIMAL then
            if $\mathcal{M}_1.solnum$ is odd then
                return $\mathcal{M}_0.objval;$
            else
                /* Note the values of the last 3 bits are all 1 */
                $\mathcal{M}_0.con \leftarrow remove(u_0'^{(0)}, u_1'^{(0)}, \ldots, u_{284}'^{(0)})$
                $\mathcal{M}_0.update();$
```

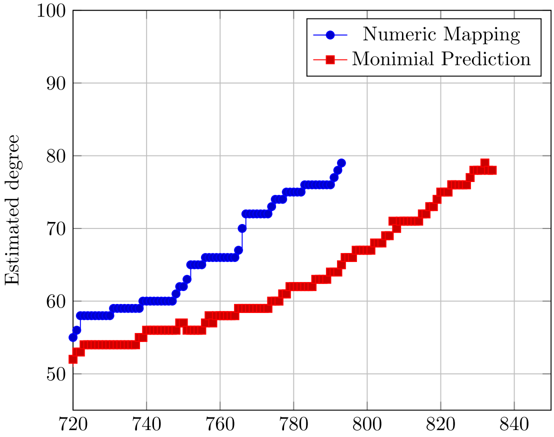

Fig. 3. The exact degree derived by monomial prediction and the upper bound derived by numeric mapping [18].

We also perform the degree evaluations with the two-subset bit-based division property [27] to estimate the upper bound of the degree of r-round Trivium. The results show that the division property is quite precise. From 1- to 834-round Trivium, there are only 14 cases where the division property fails to hit the exact degrees, which are listed in Table 2.

Table 2. The gaps among the exact degree, the upper bound obtained by the two-subset bit-based division property and the numeric mapping for several special cases of Trivium up to 834-round. For the other cases, the result obtained by the two-subset bit-based division property equals to the exact degree.

#Round	508	509	514	515	719	770	773	783	789	806	810	816	831	833
Exact degree	13	13	15	15	51	59	59	62	63	69	71	72	78	78
Division property	14	14	16	16	52	60	60	63	64	70	72	73	79	79
Numeric mapping	16	16	16	17	55	72	72	76	76	>80	>80	>80	>80	>80

5 Application II: Cube Attacks

The cube attack was proposed by Dinur and Shamir [7] at EUROCRYPT 2009. Let $f(\boldsymbol{x})$ be a Boolean function from $\mathbb{F}_2^n$ to $\mathbb{F}_2$, and $\boldsymbol{u} \in \mathbb{F}_2^n$ be a constant vector. Then $f(\boldsymbol{x})$ can be represented uniquely as

$$f(\boldsymbol{x}) = \boldsymbol{x}^{\boldsymbol{u}} p(\boldsymbol{x}) + q(\boldsymbol{x}),$$

where each term of $q(\boldsymbol{x})$ is not divisible by $\boldsymbol{x}^{\boldsymbol{u}}$. Note that in our notations, the set $I_{\boldsymbol{u}} = \{0 \le i \le n-1 : u_i = 1\} \subseteq \{0, \cdots, n-1\}$ and the monomial $\boldsymbol{x}^{\boldsymbol{u}}$ correspond to the *cube indices* and *cube term* that are commonly used in the literature of cube attacks[5]. If we compute the sum of f over the cube $\mathbb{C}_{\boldsymbol{u}} = \{\boldsymbol{x} \in \mathbb{F}_2^n : \boldsymbol{x} \preceq \boldsymbol{u}\}$, we have

$$\bigoplus\nolimits_{\boldsymbol{x} \in \mathbb{C}_{\boldsymbol{u}}} f(\boldsymbol{x}) = \bigoplus\nolimits_{\boldsymbol{x} \in \mathbb{C}_{\boldsymbol{u}}} (\boldsymbol{x}^{\boldsymbol{u}} p(\boldsymbol{x}) + q(\boldsymbol{x})) = p(\boldsymbol{x}),$$

where $p(\boldsymbol{x})$ is called the *superpoly* of the cube $\mathbb{C}_{\boldsymbol{u}}$, and $p(\boldsymbol{x})$ only involves variables x_j with $j \in I_{\bar{\boldsymbol{u}}} = \{0 \le i \le n-1 : u_i = 0\}$.

The superpoly recovery plays a critical role in the cube attack. The attacker recovers the superpoly in the offline phase, and then in the online phase, he/she queries the encryption oracle with the cube, and finally gets the value of the superpoly. If the superpoly is a balanced Boolean function, a bit information of the secret key can be obtained. The remaining key bits can be recovered by the exhaustive search.

At the early stage in the applications of cube attacks, the superpoly recovery is achieved experimentally by summing the outputs over certain "good" cubes, and therefore the sizes of cubes are largely confined in a practical range. Moreover, superpolies derived from small cubes have to be extremely simple (typically linear or quadratic functions [7,19]) in order to be recovered in a probabilistic way.

In [26], the division property was first introduced to enhance cube attacks, which allows us to identify the key bits that do not present in the superpoly. This approach is deterministic and can be used to analyze cubes whose sizes are beyond practical reach. By setting the key bits that are not involved in the superpoly to arbitrary constants and varying the remaining l key bits, one can obtain the truth table of the superpoly for a subsequent key-recovery attack with complexity $2^{|I|+l}$. At CRYPTO 2018, Wang et al. proposed the flag technique and term enumeration technique to recover directly all the monomials of the superpoly based on the two-subset bit-based division property, which further lowers the complexity of the superpoly recovery and thus attacks of more rounds on several targets are mounted [29].

However, in [26,29], it was assumed that every identified secret key variable or the monomial must be involved in the superpoly. If such an assumption does not hold, the superpoly can be much simpler than estimated, or even falls into the extreme case: $p(x) \equiv 0$. In fact it has been reported in [8,9,30,32] that some of previous key-recovery attacks are actually distinguishers. To get rid of this assumption, Wang et al. for the first time proposed a systematic method based on the three-subset bit-based division property to recover the exact superpoly [30]. In [9], the method was refined as the three-subset bit-based division property without unknown subsets and was modeled under the `PoolSearchMode` of Gurobi. As a result, they recovered the exact superpolies for 840-, 841- and 842-round Trivium.

5.1 Apply Monomial Prediction to Superpoly Recovery

It is natural to apply the monomial prediction to the recovery of the superpoly. For $f : \mathbb{F}_2^n \to \mathbb{F}_2$, we define a constant vector $\boldsymbol{u} \in \mathbb{F}_2^n$ and let the corresponding

[5] When there is no ambiguity, we denote the cube indices as I and its size as $|I|$.

cube term be $\boldsymbol{x}^{\boldsymbol{u}}$. To recover the superpoly which is a polynomial of x_i's with $\bar{u}_i = 1$, we find all the possible monomials like $\boldsymbol{x}^{\boldsymbol{u}\oplus\boldsymbol{w}} = \boldsymbol{x}^{\boldsymbol{u}} \cdot \boldsymbol{x}^{\boldsymbol{w}}$ where $\boldsymbol{w} \preceq \bar{\boldsymbol{u}}$ satisfying $\boldsymbol{x}^{\boldsymbol{u}\oplus\boldsymbol{w}} \rightarrow f$. Then the superpoly of $\boldsymbol{x}^{\boldsymbol{u}}$ is

$$p(\boldsymbol{x}) = \bigoplus_{\substack{\boldsymbol{w}\preceq\bar{\boldsymbol{u}} \\ \boldsymbol{x}^{\boldsymbol{u}\oplus\boldsymbol{w}}\rightarrow f}} \boldsymbol{x}^{\boldsymbol{w}} = \Big(\bigoplus_{\substack{\boldsymbol{w}\preceq\bar{\boldsymbol{u}} \\ \boldsymbol{x}^{\boldsymbol{u}\oplus\boldsymbol{w}}\rightarrow f}} \boldsymbol{x}^{\boldsymbol{u}\oplus\boldsymbol{w}} \Big) / \boldsymbol{x}^{\boldsymbol{u}}.$$

To find all $\boldsymbol{x}^{\boldsymbol{u}\oplus\boldsymbol{w}} \rightarrow f$ for $\boldsymbol{w} \preceq \bar{\boldsymbol{u}}$, we could take the `PoolSearchMode` of Gurobi solver to find all solutions satisfying $\boldsymbol{x}^{\boldsymbol{u}\oplus\boldsymbol{w}} \rightsquigarrow f$. Next, we store all the $\boldsymbol{x}^{\boldsymbol{u}\oplus\boldsymbol{w}}$ into a hash table which are indexed by $(\boldsymbol{u}, \boldsymbol{w})$, the size of each possible $\boldsymbol{x}^{\boldsymbol{u}\oplus\boldsymbol{w}} \bowtie f$ for $\boldsymbol{w} \preceq \bar{\boldsymbol{u}}$ can be counted naturally.

Speedup and Memory Reduction: A Divide-and-Conquer Strategy. In this paper, we only study the composite function f, where

$$f = \boldsymbol{f}^{(r-1)} \circ \boldsymbol{f}^{(r-2)} \circ \cdots \circ \boldsymbol{f}^{(0)}.$$

According to Lemma 2, if $\pi_{\boldsymbol{u}^{(0)}}(\boldsymbol{x}^{(0)}) \rightsquigarrow f$, then for $0 < i < r$,

$$|\pi_{\boldsymbol{u}^{(0)}}(\boldsymbol{x}^{(0)}) \bowtie f| \equiv \sum_{\pi_{\boldsymbol{u}^{(r-i)}}(\boldsymbol{x}^{(r-i)})\rightarrow f} |\pi_{\boldsymbol{u}^{(0)}}(\boldsymbol{x}^{(0)}) \bowtie \pi_{\boldsymbol{u}^{(r-i)}}(\boldsymbol{x}^{(r-i)})| \pmod 2. \tag{8}$$

Generally speaking, computing $|\pi_{\boldsymbol{u}^{(0)}}(\boldsymbol{x}^{(0)}) \bowtie \pi_{\boldsymbol{u}^{(r-i)}}(\boldsymbol{x}^{(r-i)})|$ one by one is much easier than computing $|\pi_{\boldsymbol{u}^{(0)}}(\boldsymbol{x}^{(0)}) \bowtie f|$ when i is significantly smaller than r. In this paper, we always expand f firstly and then obtain the speedups and memory reductions by the divide-and-conquer strategy.

5.2 Application to Trivium

Let $z_r = f(\boldsymbol{x}^{(0)})$ be the output of the r-round Trivium with $\boldsymbol{x}^{(0)} \in \mathbb{F}_2^{288}$. When the cube attack is applied to Trivium, only the cube variables indexed by the cube indices I and the secret key bits are regarded as symbolic variables in our analysis, and all other input variables are fixed to constants. Therefore, we are actually analyzing the derived function of f with the variable masks $\boldsymbol{\Gamma}^0$, $\boldsymbol{\Gamma}^1$, $\boldsymbol{\Gamma}^p$, and $\boldsymbol{\Gamma}^s$ given as follows:

$$\Gamma_i^0 = \begin{cases} 1, & \text{if } x_i \equiv 0, \\ 0, & \text{otherwise.} \end{cases} \qquad \Gamma_i^1 = \begin{cases} 1, & \text{if } x_i \equiv 1, \\ 0, & \text{otherwise.} \end{cases}$$
$$\Gamma_i^p = \begin{cases} 1, & \text{if } i \in I, \\ 0, & \text{otherwise.} \end{cases} \qquad \Gamma_i^s = \begin{cases} 1, & \text{if } 0 \le i \le 79, \\ 0, & \text{otherwise.} \end{cases} \tag{9}$$

To recover the superpoly corresponding to the cube indices $I = \{0 \le i \le 287 : \Gamma_i^p = 1\}$, we need to find all $\pi_{\boldsymbol{\Gamma}^p\oplus\boldsymbol{w}}(\boldsymbol{x}^{(0)}) \rightarrow f$ for all $\boldsymbol{w} \preceq \boldsymbol{\Gamma}^s$.

In practice, we take the divide-and-conquer strategy based on Eq. (8) to keep the consumption of computational resources under control. Let the internal state of the i-th round Trivium be $\boldsymbol{x}^{(i)}$. We first express z_r as a polynomial of $\boldsymbol{x}^{(r-r_0)}$ for some r_0. According to Proposition 3, when r_0 is not very large, the expression of z_r in $\boldsymbol{x}^{(r-r_0)}$ can be got by the monomial prediction technique [6].

[6] According to our experiments, a reasonable range of r_0 is from 200 to 300.

Proposition 3. *Let $z_r = f(\boldsymbol{x}^{(0)})$, and*

$$\mathbb{U}_{r-r_0} = \{\boldsymbol{u}^{(r-r_0)} : \ |\pi_{\boldsymbol{u}^{(r-r_0)}}(\boldsymbol{x}^{(r-r_0)}) \bowtie f| \bmod 2 = 1 \ \}, \quad \text{then}$$

$$f = \bigoplus_{\boldsymbol{u}^{(r-r_0)} \in \mathbb{U}_{r-r_0}} \pi_{\boldsymbol{u}^{(r-r_0)}}(\boldsymbol{x}^{(r-r_0)}).$$

Based on Proposition 3, an algorithm to express r-round Trivium in $\boldsymbol{x}^{(r-r_0)}$ is presented in Algorithm 4 in Appendix D (Ref. [11]).

Remark. We can also get the expression by symbolic computation. We choose the monomial prediction technique because most variables and constraints needed to complete this step are already presented in our model, which significantly reduces the burden of extra coding efforts.

Algorithm 3 shows how we recover the superpoly of a certain cube based on the divide-and-conquer strategy. The divide-and-conquer strategy leads to remarkable speedups and memory reductions in practice, which makes it possible

Algorithm 3: $\mathbb{U}_k = \mathsf{ComputeSuperpoly}(r, \boldsymbol{\Gamma}^0, \boldsymbol{\Gamma}^1, \boldsymbol{\Gamma}^p, \boldsymbol{\Gamma}^s)$

```
Input: The targeted number of rounds $r$ and the four variables masks for $f_d$
Output: A set $\mathbb{U}_k$ for the monomials in superpoly like $\pi_{\boldsymbol{\Gamma}^p \oplus \boldsymbol{w}}(\boldsymbol{x}^{(0)})$ for $\boldsymbol{w} \preceq \boldsymbol{\Gamma}^s$
Allocate a hash table $T$;
$\mathbb{U}_{r-r_0} \leftarrow \mathsf{ExpandTrivium}(r, r_0)$; // Practically, we set $r_0 = 200$
for each $\boldsymbol{u'}^{(r-r_0)} \in \mathbb{U}_{r-r_0}$ do
    $(\mathcal{M}, \boldsymbol{u}^{(0)}, \boldsymbol{u}^{(r-r_0)}) \leftarrow \mathsf{GenerateModel}(r - r_0)$;
    $\mathcal{M}$.PoolSearchMode $\leftarrow 1$;
    for $i = 0; i < 288; i \leftarrow i + 1$ do
        if $\Gamma_i^0$ is 1 then
            $u_i^{(0)} \leftarrow 0$;
        if $\Gamma_i^p$ is 1 then
            $u_i^{(0)} \leftarrow 1$;
    $\boldsymbol{u}^{(r-r_0)} \leftarrow \boldsymbol{u'}^{(r-r_0)}$;
    $\mathcal{M}.optimize()$;
    if $\mathcal{M}.status$ is OPTIMAL then
        /* Store all the solutions in hash table and count */
        for $i = 0; i < \mathcal{M}.solnum; i \leftarrow i + 1$ do
            $\mathcal{M}.SolutionNumber \leftarrow i$;
            $T[(u_0^{(0)}, u_1^{(0)}, \ldots, u_{79}^{(0)})] \leftarrow T[(u_0^{(0)}, u_1^{(0)}, \ldots, u_{79}^{(0)})] + 1$;
for $i = 0; i < H.linenumber; i \leftarrow i + 1$ do
    if $T[i] \bmod 2$ is 1 then
        $\mathbb{U}_k \leftarrow \mathbb{U}_k \cup \{i\}$;
return $\mathbb{U}_k$
```

to test more cubes with limited resources. As a result, we identify some cubes with smaller dimensions for Trivium, and thus improve upon several currently known best attacks on Trivium. We list our experimental results with different smaller-dimension cubes in Table 3 (Ref. [11]). To verify our program, we reconduct the experiments in [9] using the same cube indices for 840- and 841-round Trivium and obtain the same superpolies.

Cube Attack on 840-Round Trivium. We find the superpolies p_{I_1}, p_{I_2} and p_{I_3} for three different cube indices I_1, I_2 and I_3[7], whose dimensions are 75, 76, and 76, respectively.

Taking the cube of dimension 75 as $I_1 = \{0, 1, \ldots, 69, 71, 73, 75, 77, 79\}$ with

$$IV[70] = IV[72] = IV[74] = IV[76] = IV[78] = 0,$$

we recover a balanced superpoly for 840-round Trivium that has 41 terms and of the algebraic degree 4. The independent monomial of the superpoly is labeled by the red text.

$$\begin{aligned} p_{I_1} =& k_{79} \oplus k_{77} \oplus k_{78}k_{77} \oplus k_{76}k_{75} \oplus k_{76}k_{63} \oplus k_{75}k_{74}k_{63} \oplus k_{73}k_{63} \oplus k_{72}k_{63} \oplus k_{71}k_{63} \oplus \\ & k_{72}k_{71}k_{63} \oplus k_{71}k_{70}k_{63} \oplus k_{70}k_{69}k_{63} \oplus k_{63}k_{61} \oplus k_{63}k_{60} \oplus k_{61}k_{60} \oplus k_{63}k_{59} \oplus \\ & k_{63}k_{59}k_{58} \oplus k_{61}k_{59}k_{58} \oplus k_{63}k_{57} \oplus k_{63}k_{57}k_{56} \oplus k_{52} \oplus k_{50} \oplus k_{63}k_{50} \oplus k_{63}k_{49} \oplus \\ & k_{63}k_{46} \oplus k_{63}k_{45} \oplus k_{63}k_{44} \oplus k_{63}k_{33} \oplus k_{61}k_{33} \oplus k_{63}k_{32} \oplus k_{63}k_{31} \oplus k_{63}k_{26} \oplus \\ & k_{71}k_{63}k_{12} \oplus k_{70}k_{69}k_{63}k_{12} \oplus k_{63}k_{59}k_{12} \oplus k_{63}k_{58}k_{12} \oplus k_{63}k_{57}k_{12} \oplus k_{63}k_{58}k_{57}k_{12} \oplus \\ & k_{63}k_{50}k_{12} \oplus k_{63}k_{44}k_{12} \oplus k_{63}k_{26}k_{12}. \end{aligned}$$

Taking the cube of dimension 76 as $I_2 = \{0, 1, \ldots, 71, 73, 75, 77, 79\}$ with

$$IV[72] = IV[74] = IV[76] = IV[78] = 0,$$

we recover a balanced superpoly for 840-round Trivium that has 4 terms and algebraic degree of 2, and give it as follows

$$p_{I_2} = 1 \oplus k_{64} \oplus k_{63}k_{62} \oplus k_{37}.$$

Taking the cube of dimension 76 as $I_3 = \{0, 1, \ldots, 69, 71, 72, 73, 75, 77, 79\}$ with

$$IV[70] = IV[74] = IV[76] = IV[78] = 0,$$

we recover a balanced superpoly for 840-round Trivium that has 6 terms and algebraic degree of 3 as below,

$$p_{I_3} = 1 \oplus k_{63} \oplus k_{59} \oplus k_{59}k_{50} \oplus k_{59}k_{49}k_{48} \oplus k_{59}k_{23}.$$

Let $\mathbb{C}_I = \{\boldsymbol{x} \in \mathbb{F}_2^{288} : \boldsymbol{x} \preceq \boldsymbol{\Gamma}^p\}$, where $\boldsymbol{\Gamma}^p$ is set as Eq. (9). since $I_2 = I_1 \cup \{70\}$,

$$p_{I_2} = \bigoplus_{\boldsymbol{x} \in \mathbb{C}_{I_2}} f(\boldsymbol{x}) = \bigoplus_{\boldsymbol{x} \in \mathbb{C}_{I_1}, IV[70]=1} f(\boldsymbol{x}) \oplus \bigoplus_{\boldsymbol{x} \in \mathbb{C}_{I_1}, IV[70]=0} f(\boldsymbol{x}),$$

[7] For convenience, every element in the cube indices $I_i, 0 \leq i \leq 11$ in this subsection is the index of IV, i.e. from 0 to 79.

and

$$p_{I_1} = \bigoplus_{\boldsymbol{x} \in \mathbb{C}_{I_1}} f(\boldsymbol{x}) = \bigoplus_{\boldsymbol{x} \in \mathbb{C}_{I_1}, IV[70]=0} f(\boldsymbol{x}),$$

then we can deduce that $p_{I_4} = p_{I_1} \oplus p_{I_2}$ is the superpoly for the cube indices $I_4 = \{0, 1, \dots, 69, 71, 73, 75, 77, 79\}$ with

$$IV[72] = IV[74] = IV[76] = IV[78] = 0, IV[70] = 1.$$

Similarly, we can deduce that $p_{I_5} = p_{I_1} \oplus p_{I_3}$ is the superpoly for the cube indices $I_5 = \{0, 1, \dots, 69, 71, 73, 75, 77, 79\}$ with

$$IV[70] = IV[74] = IV[76] = IV[78] = 0, IV[72] = 1.$$

p_{I_1}, p_{I_4} and p_{I_5} are balanced Boolean functions because there are monomials that are independent of other monomials, respectively. Therefore, we can recover 3 bits of key information by using $3 \times 2^{75} \approx 2^{76.6}$ time complexity. The dominant part of the whole key recovery attack is the exhaustive search after the recovery of the 3-bit key information, which is 2^{77} time complexity. So in total, the time complexity for this 840-round Trivium is $2^{76.6} + 2^{77} \approx 2^{77.8}$.

Cube Attack on 841-Round Trivium. We find the superpolies p_{I_6} and p_{I_7} for the set of cube indices I_6 and I_7, whose dimensions are 76 and 77, respectively. Taking the cube of dimension 76 as $I_6 = \{0, 1, \dots, 69, 71, 73, 74, 75, 77, 79\}$ with

$$IV[70] = IV[72] = IV[76] = IV[78] = 0,$$

we recover a balanced superpoly p_6 for 841-round Trivium that has 3632 terms and algebraic degree of 9. Since the number of terms in p_{I_6} (and other superpolies, e.g., $p_{I_7}, p_{I_9}$and $p_{I_{10}}$ are too many, we provide them at https://github.com/hukaisdu/MonomialPrediction/blob/master/superpoly.pdf.

Taking the cube of dimension 77 as $I_7 = \{0, 1, \dots, 71, 73, 74, 75, 77, 79\}$ with

$$IV[72] = IV[76] = IV[78] = 0,$$

we recover a balanced superpoly p_{I_7} for 841-round Trivium that has 1400 terms and algebraic degree of 8.

Similar with p_{I_4}, $p_{I_8} = p_{I_6} \oplus p_{I_7}$ is the superpoly for the cube indices $I_8 = \{0, 1, \dots, 69, 71, 73, 7475, 77, 79\}$ with

$$IV[72] = IV[76] = IV[78] = 0, IV[70] = 1.$$

Hence, we can recover 2 bits of the key information with time complexity $2^{77} = 2 \times 2^{76}$. The dominant part of the whole key recovery attack is the exhaustive search after 2-bit key recovery, which is 2^{78} time complexity. Therefore, totally the time complexity of the attack on the 841-round Trivium is $2^{78} + 2^{77} \approx 2^{78.6}$.

Cube Attack on 842-Round Trivium. We find the superpolies p_{I_9} and $p_{I_{10}}$ for the set of cube indices I_9 and I_{10}, whose dimensions are 76 and 77, respectively.

Taking the cube of dimension 76 as $I_9 = \{0, 1, \dots, 71, 73, 75, 77, 79\}$ with

$$IV[72] = IV[74] = IV[76] = IV[78] = 0,$$

we recover a balanced superpoly for 842-round Trivium that has 5147 terms and algebraic degree of 8.

Taking the cube of dimension 77 as $I_{10} = \{0, 1, \dots, 73, 75, 77, 79\}$ with

$$IV[74] = IV[76] = IV[78] = 0,$$

we recover a balanced superpoly p_{10} for 842-round Trivium that has 4174 terms and algebraic degree of 8.

Similar with p_{I_4}, $p_{I_{11}} = p_{I_9} \oplus p_{I_{10}}$ is the superpoly of the cube indices $I_{11} = \{0, 1, \dots, 71, 73, 75, 77, 79\}$ with $IV[74] = IV[76] = IV[78] = 0, IV[72] = 1$. Therefore, we can recover 2 bits of key information by using $2^{77} = 2 \times 2^{76}$ time complexities. The dominant part of the whole key recovery attack is the exhaustive search after 2-bit key recovery, which is 2^{78} time complexity. Totally, the time complexity is $2^{78} + 2^{77} \approx 2^{78.6}$.

6 Division Property from an Algebraic Viewpoint

Since 2015, various division properties together with their "propagation rules" are proposed in the literature, including the word-based division property [21,25], the two-subset bit-based division property [27] (a.k.a. the conventional bit-based division property), the three-subset bit-based division property [27], and the recent three-subset bit-based division property without unknown subset [9,30]. Based on these properties with their associated propagation rules, detection algorithms or tools can be built. In a narrow sense, these detection algorithms are used to detect whether the sum of an output bit of a symmetric-key primitive over a carefully constructed input data set is *key-independent*, that is, the sum is a constant (0 or 1) for any key.

We now look at the detection algorithms for the *key-independent* property from an algebraic viewpoint. Before we go any further, we would like to mention that the first attempt to formulate the division property in an algebraic way was made by Boura and Canteaut at CRYPTO 2016 [2]. However, they only focused themselves on local components rather than on the global (keyed) Boolean functions. Furthermore, Biryukov, Khovratovich, and Perrin proposed the multiset-algebraic cryptanalysis which can also be seen as an algebraic treatment of the division property [1]. But they focused more on the algebraic degree only. Now, let us proceed to show the following conclusions:

- A *perfect* detection algorithm for the *key-independent* property can be constructed based on the monomial prediction (i.e., this algorithm never raises false alarms and never misses).
- The word-based division property [25], two-subset bit-based division property [27] and three-subset bit-based division property [27] together with their propagation rules lead to *no-false-alarm* detection algorithms for the *key-independent* property (however, these algorithms can miss).

- The three-subset bit-based division property without unknown subset with its propagation rules [9] forms a *perfect* detection algorithm for the *key-independent* property, and an equivalence between it and the monomial prediction technique can be established.

6.1 A Perfect Detection Algorithm Based on Monomial Prediction

For a composite function $\boldsymbol{f} : \mathbb{F}_2^n \to \mathbb{F}_2^m, \boldsymbol{x}^{(r)} = \boldsymbol{f}(\boldsymbol{x}^{(0)})$, we define a constant vector $\boldsymbol{u} \in \mathbb{F}_2^n$ then we derive a structure of the input values $\mathbb{X} = \{\boldsymbol{x} \in \mathbb{F}_2^n : \boldsymbol{x} \preceq \boldsymbol{u}\}$. We want to detect whether

$$\lambda = \bigoplus_{\boldsymbol{x} \in \mathbb{X}} \pi_{\boldsymbol{u}^{(r)}}(\boldsymbol{f}(\boldsymbol{x}))$$

is independent of the variables x_i's with $\bar{u}_i = 1$ denoted by $\bar{\boldsymbol{u}}$-(in)dependent. From the viewpoint of presence and absence of monomials, we have

$$\lambda = \begin{cases} \bar{\boldsymbol{u}}\text{-dependent}, & \text{if } \pi_{\boldsymbol{u}\oplus\boldsymbol{w}}(\boldsymbol{x}^{(0)}) \to \pi_{\boldsymbol{u}^{(r)}}(\boldsymbol{x}^{(r)}) \text{ for some } \boldsymbol{0} \prec \boldsymbol{w} \preceq \bar{\boldsymbol{u}} \\ \bar{\boldsymbol{u}}\text{-independent}, & \text{if } \pi_{\boldsymbol{u}\oplus\boldsymbol{w}}(\boldsymbol{x}^{(0)}) \nrightarrow \pi_{\boldsymbol{u}^{(r)}}(\boldsymbol{x}^{(r)}) \text{ for all } \boldsymbol{0} \prec \boldsymbol{w} \preceq \bar{\boldsymbol{u}} \end{cases}$$

Hence, for $\boldsymbol{f}$, the monomial prediction can detect whether λ is independent of x_i with $\bar{u}_i = 1$ precisely in theory by computing $|\pi_{\boldsymbol{u}\oplus\boldsymbol{w}}(\boldsymbol{x}^{(0)}) \bowtie \pi_{\boldsymbol{u}^{(r)}}(\boldsymbol{x}^{(r)})|$ for every possible $\boldsymbol{0} \prec \boldsymbol{w} \preceq \bar{\boldsymbol{u}}$.

Application to Derived Function. When applying the monomial prediction to a practical cipher, some part of the public variables will be fixed as a constant value. Let $\boldsymbol{\Gamma}^0, \boldsymbol{\Gamma}^1, \boldsymbol{\Gamma}^p$ and $\boldsymbol{\Gamma}^s$ be four constant vectors indicating the 0-constant public variables, 1-constant public variables, the non-constant public variables and the secret variables, respectively. Then we study the derived function $\boldsymbol{f}_d$ of $\boldsymbol{f}$ with $\boldsymbol{\Gamma}^0, \boldsymbol{\Gamma}^1, \boldsymbol{\Gamma}^p, \boldsymbol{\Gamma}^s$. In the integral attack, the chosen plaintext set is

$$\mathbb{X}_0 = \{\boldsymbol{x} \oplus \boldsymbol{\Gamma}^1 \in \mathbb{F}_2^n : \boldsymbol{x} \preceq \boldsymbol{\Gamma}^p\}. \tag{10}$$

And we are interested in whether

$$\Lambda = \bigoplus_{\boldsymbol{x} \in \mathbb{X}_0} \pi_{\boldsymbol{u}^{(r)}}(\boldsymbol{f}_d(\boldsymbol{x})).$$

is independent of the secret variables x_i with $\Gamma_i^s = 1$, denoted by key-(in)dependent. Similarly,

$$\Lambda = \begin{cases} \text{key-dependent}, & \text{if } \pi_{\boldsymbol{\Gamma}^p\oplus\boldsymbol{w}}(\boldsymbol{x}^{(0)}) \to \pi_{\boldsymbol{u}^{(r)}}(\boldsymbol{x}^{(r)}) \text{ for some } \boldsymbol{0} \prec \boldsymbol{w} \preceq \boldsymbol{\Gamma}^s \\ \text{key-independent}, & \text{if } \pi_{\boldsymbol{\Gamma}^p\oplus\boldsymbol{w}}(\boldsymbol{x}^{(0)}) \nrightarrow \pi_{\boldsymbol{u}^{(r)}}(\boldsymbol{x}^{(r)}) \text{ for all } \boldsymbol{0} \prec \boldsymbol{w} \preceq \boldsymbol{\Gamma}^s \end{cases}$$

Hence, by computing $|\pi_{\boldsymbol{\Gamma}^p\oplus\boldsymbol{w}}(\boldsymbol{x}^{(0)}) \bowtie \pi_{\boldsymbol{u}^{(r)}}(\boldsymbol{x}^{(r)})|$ for every possible $\boldsymbol{0} \prec \boldsymbol{w} \preceq \boldsymbol{\Gamma}^s$, we can predict whether Λ is or not key-independent.

6.2 No-False-Alarm Detection Algorithms

Although the monomial prediction can predict the key-independent property precisely, computing the size of a monomial hull is commonly difficult, especially for a block cipher because the size of the monomial hull is usually huge. Furthermore, for attackers, integral property of any bits (it is not necessary to find all) is useful in distinguishing attacks. Therefore, some trade-off between the efficiency and precision is necessary and reasonable.

Following this idea of trade-off, we show a simple observation. Recall Lemma 1, if $\pi_{\boldsymbol{u}}(\boldsymbol{x}^{(0)}) \not\rightsquigarrow \pi_{\boldsymbol{u}^{(r)}}(\boldsymbol{x}^{(r)})$, we have $\pi_{\boldsymbol{u}}(\boldsymbol{x}^{(0)}) \not\twoheadrightarrow \pi_{\boldsymbol{u}^{(r)}}(\boldsymbol{x}^{(r)})$. Then if we are able to make the claim that $\mathit{\Lambda}$ is key-independent according to $\pi_{\boldsymbol{\Gamma}^p \oplus \boldsymbol{w}}(\boldsymbol{x}^{(0)}) \not\rightsquigarrow \pi_{\boldsymbol{u}^{(r)}}(\boldsymbol{x}^{(r)})$ for any $\boldsymbol{w} \preceq \boldsymbol{\Gamma}^s$, the detection algorithm we employ will never raise false alarms.

Definition 3 (No-False-Alarm Approximations). *For two detection algorithms $\mathcal{A}_1$ and $\mathcal{A}_2$, if $\mathcal{A}_1$ claims a certain property $\mathcal{P}$ holds, $\mathcal{A}_2$ must also claim $\mathcal{P}$ holds, then we say $\mathcal{A}_1$ is a no-false-alarm approximation of $\mathcal{A}_2$.*

Next we prove that the two-subset bit-based division property is a no-false-alarm approximation of the monomial prediction.

Definition 4 (Two-Subset Bit-Based Division Property [27]). *Let $\mathbb{X}$ be a multiset whose elements are n-bit vectors and $\mathbb{K}$ be a set whose elements are n-bit vectors. When the multiset $\mathbb{X}$ has the division property $\mathcal{D}_{\mathbb{K}}^{1^n}$, it fulfills the following conditions:*

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

Let $\boldsymbol{f}_d$ be the derived function of $\boldsymbol{f}$ with $\boldsymbol{\Gamma}^0, \boldsymbol{\Gamma}^1, \boldsymbol{\Gamma}^p, \boldsymbol{\Gamma}^s$. Suppose the initially chosen set (multiset) of the plaintext is $\mathbb{X}_0$ as defined in Eq. (10) and the multiset of the ciphertext is $\mathbb{X}_r = \{\boldsymbol{y} : \boldsymbol{y} = \boldsymbol{f}_d(\boldsymbol{x}), \boldsymbol{x} \in \mathbb{X}_0\}$. Then we first compute the division property of $\mathbb{X}_0$ as $\mathcal{D}_{\mathbb{K}_0}^{1^n}$, where

$$\mathbb{K}_0 = \{\boldsymbol{k} \in \mathbb{F}_2^n : \boldsymbol{k} \succeq \boldsymbol{\Gamma}^p\}. \tag{11}$$

To compute the division property of $\mathbb{X}_r$, i.e., $\mathcal{D}_{\mathbb{K}_r}^{1^n}$, we will trace all the propagation from the vectors in $\mathbb{K}_0$. The propagation rules for the two-subset bit-based division property are listed in [13,27,31].

Proposition 4. *The two-subset bit-based division property is a no-false-alarm approximation of the monomial prediction in detecting the balance property, therefore the two-subset bit-based division property claims $\bigoplus_{\boldsymbol{x}^{(r)} \in \mathbb{X}_r} \pi_{\boldsymbol{k}^{(r)}}(\boldsymbol{x}^{(r)}) \equiv 0$ without false alarms.*

Proof. Firstly, for any $\boldsymbol{k}^{(0)} \in \mathbb{K}_0$, $\pi_{\boldsymbol{k}^{(0)}}(\boldsymbol{x}^{(0)}) = \pi_{\boldsymbol{\Gamma}^p \oplus \boldsymbol{w}}(\boldsymbol{x}^{(0)})$ where $\boldsymbol{w} = \boldsymbol{\Gamma}^p \oplus \boldsymbol{k}^{(0)} \preceq \boldsymbol{\Gamma}^1 \oplus \boldsymbol{\Gamma}^s$. Next, we consider the propagation from these vectors in $\mathbb{K}_0$. Note all kinds of components of a cipher can be seen as an S-box: $\boldsymbol{y} = \boldsymbol{S}(\boldsymbol{x})$, and the propagation of the S-box for the two-subset bit-based division property has been concluded as a rule: Let $\mathcal{D}^{1^n}_{\mathbb{K}_{in}}$ and $\mathcal{D}^{1^n}_{\mathbb{K}_{out}}$ be the input and output two-subset bit-based division property of $\boldsymbol{S}$, respectively. If $\boldsymbol{u} \in \mathbb{K}_{in}$ can propagates to $\boldsymbol{v} \in \mathbb{K}_{out}$, there must be $\boldsymbol{u'} \succeq \boldsymbol{u}$ satisfying $\pi_{\boldsymbol{u'}}(\boldsymbol{x}) \rightarrow \boldsymbol{y}^{\boldsymbol{v}}$. Since the monomial trail requires $\boldsymbol{x}^{\boldsymbol{u}} \rightarrow \boldsymbol{y}^{\boldsymbol{v}}$, then from the same $\boldsymbol{u}$, the two-subset bit-based division property can propagate to a larger range of vectors $\boldsymbol{v}$.

Hence, if $\boldsymbol{k}^{(r)} \notin \mathbb{K}_r$, we have $\pi_{\boldsymbol{k}}(\boldsymbol{x}^{(0)}) \not\rightsquigarrow \pi_{\boldsymbol{k}^{(r)}}(\boldsymbol{x}^{(r)})$ for all $\boldsymbol{k} \in \mathbb{K}_0$. Therefore, $\pi_{\boldsymbol{k}^{(r)}}(\boldsymbol{x}^{(r)})$ does not contain any terms like $\pi_{\boldsymbol{\Gamma}^p \oplus \boldsymbol{w}}(\boldsymbol{x}^{(0)}) = \pi_{\boldsymbol{w}}(\boldsymbol{x}^{(0)})\pi_{\boldsymbol{\Gamma}^p}(\boldsymbol{x}^{(0)})$ for $\boldsymbol{w} \preceq \boldsymbol{\Gamma}^1 \oplus \boldsymbol{\Gamma}^s$, naturally,

$$\bigoplus_{\boldsymbol{x}^{(r)} \in \mathbb{X}_r} \pi_{\boldsymbol{k}^{(r)}}(\boldsymbol{x}^{(r)}) = \bigoplus_{\boldsymbol{x}^{(0)} \in \mathbb{X}_0} \pi_{\boldsymbol{k}^{(r)}}(\boldsymbol{f}_d(\boldsymbol{x}^{(0)})) \equiv 0.$$

□

According to the proof, it can be checked even if $\boldsymbol{k}^{(r)} \in \mathbb{K}_r$, we cannot determine whether $\pi_{\boldsymbol{k}^{(0)}}(\boldsymbol{x}^{(0)}) \rightsquigarrow \pi_{\boldsymbol{k}^{(r)}}(\boldsymbol{x}^{(r)})$ (let alone $\pi_{\boldsymbol{k}^{(0)}}(\boldsymbol{x}^{(0)}) \rightarrow \pi_{\boldsymbol{k}^{(r)}}(\boldsymbol{x}^{(r)})$), while the two-subset division property claims that the parity is an unknown value, i.e., the two-subset bit-based division property may miss some balance properties.

Similarly, we can prove that the three-subset bit-based division property and the word-based division property are also no-false-alarm approximation of the monomial prediction. The proofs are provided in Appendix C (Ref. [11]).

6.3 The Three-Subset Bit-Based Division Property Without Unknown Subset is Perfect

In [30], Wang et al. found that we can only focus on a part of the propagation of the three-subset bit-based division property when processing a public-update cipher. Later in [9], Hao et al. formulated this method to the three-subset bit-based division property without unknown subset. In this subsection, we show it is perfect in detecting the key-independent property.

Definition 5 (Three-Subset Bit-Based Division Property w/o Unknown Subset [9,30]**).** *Let $\mathbb{X}$ and $\mathbb{L}$ be two multisets whose elements are n-bit vectors. When the multiset $\mathbb{X}$ has the three-subset bit-based division property without unknown subset $\mathcal{T}^{1^n}_{\mathbb{L}}$, it fulfills the following conditions:*

$$\bigoplus_{\boldsymbol{x} \in \mathbb{X}} \pi_{\boldsymbol{\ell}}(\boldsymbol{x}) = \begin{cases} 1, & \text{if there are } \textit{odd-number}\ \boldsymbol{\ell} \text{ in } \mathbb{L}, \\ 0, & \text{if there are } \textit{even-number}\ \boldsymbol{\ell} \text{ in } \mathbb{L}. \end{cases}$$

Let $\boldsymbol{f}_d$ be the derived function of $\boldsymbol{f}$ with $\boldsymbol{\Gamma}^0, \boldsymbol{\Gamma}^1, \boldsymbol{\Gamma}^p, \boldsymbol{\Gamma}^s$[8]. Suppose the initial chosen set (multiset) of the plaintext is $\mathbb{X}_0$ in Eq. (10), and the multiset of the

[8] In [9], the definition of the three-subset division property without unknown subset made no distinction between the public and secret variables, equivalently, $\boldsymbol{\Gamma}^s = \boldsymbol{0}$ and $\boldsymbol{\Gamma}^p$ indicates all variables.

ciphertext is $\mathbb{X}_r = \{\boldsymbol{y} : \boldsymbol{y} = \boldsymbol{f}_d(\boldsymbol{x}), \boldsymbol{x} \in \mathbb{X}_0\}$. Then we first compute the division property of $\mathbb{X}_0$ as $\mathcal{T}_{\mathbb{L}_0}^{1^n}$ [30], where

$$\mathbb{L}_0 = \{\boldsymbol{\ell} \in \mathbb{F}_2^n : \boldsymbol{\Gamma}^p \preceq \boldsymbol{\ell} \preceq \boldsymbol{\Gamma}^p \oplus \boldsymbol{\Gamma}^1\}. \tag{12}$$

To compute the division property of $\mathbb{X}_r$, i.e., $\mathcal{T}_{\mathbb{L}_r}^{1^n}$, we will trace all the propagation from the vectors in $\mathbb{L}_0$. The propagation rules for three-subset bit-based division property without unknown subset are listed in [9,30].

Proposition 5. *The three-subset bit-based division property without unknown subset predicts $\bigoplus_{\boldsymbol{x}^{(r)} \in \mathbb{X}_r} \pi_{\boldsymbol{\ell}^{(r)}}(\boldsymbol{x}^{(r)})$ for any $\boldsymbol{\ell}^{(r)}$ perfectly.*

Proof. Firstly, for any $\boldsymbol{\ell}^{(0)} \in \mathbb{L}_0$, $\pi_{\boldsymbol{\ell}^{(0)}}(\boldsymbol{x}^{(0)}) = \pi_{\boldsymbol{\Gamma}^p \oplus \boldsymbol{w}}(\boldsymbol{x}^{(0)})$ where $\boldsymbol{w} = \boldsymbol{\Gamma}^p \oplus \boldsymbol{\ell}^{(0)} \preceq \boldsymbol{\Gamma}^1$. Then $\pi_{\boldsymbol{\Gamma}^p \oplus \boldsymbol{w}}(\boldsymbol{x}^{(0)}) = \pi_{\boldsymbol{\Gamma}^p}(\boldsymbol{x}^{(0)})$. Next, we consider the propagation from these vectors in $\mathbb{L}_0$. Since all kinds of components of a cipher can be seen as an S-box: $\boldsymbol{y} = \boldsymbol{S}(\boldsymbol{x})$ and the propagation of the S-box for the three-subset bit-based division property without unknown subset has been concluded as a rule that guarantees $\boldsymbol{x}^{\boldsymbol{u}} \to \boldsymbol{y}^{\boldsymbol{v}}$ [30], we can trace the propagation and compute out $\mathbb{L}_r$. Therefore, for every vector $\boldsymbol{\ell}^{(r)} \in \mathbb{L}_r$, there is a monomial trail connecting $\pi_{\boldsymbol{\ell}^{(0)}}(\boldsymbol{x}^{(0)})$ and $\pi_{\boldsymbol{\ell}^{(r)}}(\boldsymbol{x}^{(r)})$ since $\boldsymbol{x}^{\boldsymbol{u}} \to \boldsymbol{y}^{\boldsymbol{v}}$ is also required by Definition 1. Let $\boldsymbol{\ell}^{(r)}$ appears N times in $\mathbb{L}_r$, then

$$N = \sum_{\boldsymbol{\ell} \in \mathbb{L}_0} |\pi_{\boldsymbol{\ell}}(\boldsymbol{x}^{(0)}) \bowtie \pi_{\boldsymbol{\ell}^{(r)}}(\boldsymbol{x}^{(r)})| = \sum_{\boldsymbol{w} \preceq \boldsymbol{\Gamma}^1} |\pi_{\boldsymbol{\Gamma}^p \oplus \boldsymbol{w}}(\boldsymbol{x}^{(0)}) \bowtie \pi_{\boldsymbol{\ell}^{(r)}}(\boldsymbol{x}^{(r)})|.$$

According to Proposition 2, $\pi_{\boldsymbol{\Gamma}^p}(\boldsymbol{x}^{(0)}) \to \pi_{\boldsymbol{\ell}^{(r)}}(\boldsymbol{x}^{(r)})$, if and only if $N \bmod 2 = 1$. □

6.4 An Alternative Detection Algorithm for Division Property

The algebraic insights into the division property bring us much more flexibility in designing new detection algorithms for balance properties. Although the three-subset bit-based division property is more accurate than the two-subset bit-based division property [30], the latter is more MILP-friendly and needs simpler programming, therefore the two-subset version is more efficient. According to the existing literature, the three-subset bit-based division property can find several more balanced bits, but hardly surpass the two-subset version by rounds. Hence, the two-subset bit-based division property is still the dominant method in searching for the integral property.

From an algebraic viewpoint, we show how to design a new detection algorithm of division property which surpasses the capability but achieves the similar efficiency with the two-subset bit-based division property. For the derived function $\boldsymbol{f}_d$ with $\boldsymbol{\Gamma}^0, \boldsymbol{\Gamma}^1, \boldsymbol{\Gamma}^p, \boldsymbol{\Gamma}^s$, if we want to determine whether $\bigoplus_{\boldsymbol{x} \in \mathbb{X}_0} \pi_{\boldsymbol{u}^{(r)}}(\boldsymbol{f}_d(\boldsymbol{x}))$ is key-independent or not, we only need to check whether $\pi_{\boldsymbol{u}^{(r)}}(\boldsymbol{x}^{(r)})$ contains any term in

$$\mathbb{S}_0 = \{\pi_{\boldsymbol{\Gamma}^p \oplus \boldsymbol{w}}(\boldsymbol{x}^{(0)}) : \boldsymbol{0} \prec \boldsymbol{w} \preceq \boldsymbol{\Gamma}^s\}.$$

Table 3. Some experimental results of our new detection algorithm compared with the previous ones. All results are re-produced on the same platform.

Cipher	#Data	#Round	#Constant	Time	Method
Simon32	2^{31}	15	–†	–	[31]
			3	27 s	[12]
			3	120 s	[30]
			3	3 s	Ours
Simon32 (102)‡	2^{31}	20	1	3 s	[31]
			3	25 s	[12]
			3	3 s	Ours
Simon48 (102)	2^{47}	28	3	8 s	[31]
			3	9 s	[12]
			3	8 s	Ours
Simon64 (102)	2^{63}	36	1	23 s	[31]
			3	1.1 h	[12]
			3	30 s	Ours

† The two-subset bit-based division property cannot find the 15-round integral distinguisher for Simon32.

‡ Simon32 (102) means the rotation constants are (1,0,2) rather than (8,1,2), see [31].

Consider $\mathbb{S}_r = \{\pi_{\boldsymbol{u}^{(r)}}(\boldsymbol{x}^{(r)}) : \pi_{\boldsymbol{u}^{(0)}}(\boldsymbol{x}^{(0)}) \rightsquigarrow \pi_{\boldsymbol{u}^{(r)}}(\boldsymbol{x}^{(r)})\}$, if $\pi_{\boldsymbol{u}^{(r)}}(\boldsymbol{x}^{(r)}) \notin \mathbb{S}_r$, then we know $\boldsymbol{f}_d$ does not contain any monomials in $\mathbb{S}_0$ since there is no monomial trail. Therefore $\bigoplus_{\boldsymbol{x}\in\mathbb{X}_0} \pi_{\boldsymbol{u}^{(r)}}(\boldsymbol{f}_d(\boldsymbol{x}))$ is a key-independent value.

To detect it, firstly, we construct the model of $\pi_{\boldsymbol{u}^{(0)}}(\boldsymbol{x}^{(0)}) \rightsquigarrow \pi_{\boldsymbol{u}^{(r)}}(\boldsymbol{x}^{(r)})$ by decomposing the target cipher like we do for Trivium. Secondly, we impose another constraint on all the round key bits k_i on the MILP model $\mathcal{M}$ as

$$\mathcal{M} \leftarrow \sum_i k_i \geq 1.$$

Finally, we check the validity of this model. If the model is infeasible, then $\pi_{\boldsymbol{u}^{(r)}}(\boldsymbol{x}^{(r)})$ contains no monomial in $\mathbb{S}_0$ and $\bigoplus_{\boldsymbol{x}\in\mathbb{X}_0} \pi_{\boldsymbol{u}^{(r)}}(\boldsymbol{f}_d(\boldsymbol{x}))$ is key-independent. Since we do not need to compute the size of the monomial hull, the model is easy to solve. Some experiments are conducted to show the capibility of this alternative detection algorithm, we list the results in Table 3.

7 Conclusion and Discussion

In this work, a pure algebraic treatment of the division property is presented, and we propose the monomial prediction technique which determines the presence or absence of a monomial by counting the number of monomial trails in the corresponding monomial hull. Based on this technique, we manage to obtain the exact algebraic degrees of Trivium up to 834 rounds and improved key-recovery attacks on 840-, 841- and 842-round Trivium.

Moreover, we categorize existing detection algorithms for division properties into perfect, no-false-alarm, and no-missing classes. In particular, we prove that the three-subset bit-based division property without unknown subset and monomial prediction are perfect. At this point, a natural question arises. Can we design an efficient no-missing detection algorithm for the division property that does not raise too many false alarms, which would be very useful for designers to theoretically determine the security bounds against attacks based on division properties.

Acknowledgements. We thank the anonymous reviewers for their valuable comments. This work is supported by the National Key Research and Development Program of China (No. 2018YFA0704702, 2018YFA0704704), the Major Scientific and Technological Innovation Project of Shandong Province, China (No. 2019JZZY010133), the Chinese Major Program of National Cryptography Development Foundation (MMJJ20180102), and the National Natural Science Foundation of China (61772519). The work of Qingju Wang is funded by the University of Luxembourg Internal Research Project (IRP) FDISC.

References

1. Biryukov, A., Khovratovich, D., Perrin, L.: Multiset-algebraic cryptanalysis of reduced Kuznyechik, Khazad, and secret SPNs. IACR Trans. Symmetric Cryptol. **2016**(2), 226–247 (2016)
2. Boura, C., Canteaut, A.: Another view of the division property. In: Robshaw, M., Katz, J. (eds.) CRYPTO 2016. LNCS, vol. 9814, pp. 654–682. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53018-4_24
3. Boura, C., Canteaut, A., De Cannière, C.: Higher-order differential properties of Keccak and *Luffa*. In: Joux, A. (ed.) FSE 2011. LNCS, vol. 6733, pp. 252–269. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-21702-9_15
4. De Cannière, Christophe, Preneel, Bart: Trivium. In: Robshaw, Matthew, Billet, Olivier (eds.) New Stream Cipher Designs. LNCS, vol. 4986, pp. 244–266. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-68351-3_18
5. Canteaut, A., Videau, M.: Degree of composition of highly nonlinear functions and applications to higher order differential cryptanalysis. In: Knudsen, L.R. (ed.) EUROCRYPT 2002. LNCS, vol. 2332, pp. 518–533. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-46035-7_34
6. Daemen, J., Knudsen, L., Rijmen, V.: The block cipher square. In: Biham, E. (ed.) FSE 1997. LNCS, vol. 1267, pp. 149–165. Springer, Heidelberg (1997). https://doi.org/10.1007/BFb0052343
7. Dinur, I., Shamir, A.: Cube attacks on tweakable black box polynomials. In: Joux, A. (ed.) EUROCRYPT 2009. LNCS, vol. 5479, pp. 278–299. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-01001-9_16
8. Hao, Y., Leander, G., Meier, W., Todo, Y., Wang, Q.: Modeling for three-subset division property without unknown subset. IACR Cryptology ePrint Archive, 2020:441 (2020)

9. Hao, Y., Leander, G., Meier, W., Todo, Y., Wang, Q.: Modeling for three-subset division property without unknown subset. In: Canteaut, A., Ishai, Y. (eds.) EUROCRYPT 2020. LNCS, vol. 12105, pp. 466–495. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-45721-1_17
10. Hebborn, P., Lambin, B., Leander, G., Todo, Y.: Lower bounds on the degree of block ciphers. In: Moriai, S., Wang, H. (eds.) ASIACRYPT 2020 (to appear)
11. Hu, K., Sun, S., Wang, M., Wang, Q.: An algebraic formulation of the division property: revisiting degree evaluations, cube attacks, and key-independent sums (2020). https://eprint.iacr.org/2020/1048
12. Hu, K., Wang, M.: Automatic search for a variant of division property using three subsets. In: Matsui, M. (ed.) CT-RSA 2019. LNCS, vol. 11405, pp. 412–432. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-12612-4_21
13. Kai, H., Wang, Q., Wang, M.: Finding bit-based division property for ciphers with complex linear layers. IACR Trans. Symmetric Cryptol. **2020**(1), 236–263 (2020)
14. Jakobsen, T., Knudsen, L.R.: The interpolation attack on block ciphers. In: Biham, E. (ed.) FSE 1997. LNCS, vol. 1267, pp. 28–40. Springer, Heidelberg (1997). https://doi.org/10.1007/BFb0052332
15. Knudsen, L.R.: Truncated and higher order differentials. In: Preneel, B. (ed.) FSE 1994. LNCS, vol. 1008, pp. 196–211. Springer, Heidelberg (1995). https://doi.org/10.1007/3-540-60590-8_16
16. Knudsen, L., Wagner, D.: Integral cryptanalysis. In: Daemen, J., Rijmen, V. (eds.) FSE 2002. LNCS, vol. 2365, pp. 112–127. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-45661-9_9
17. Lai, X.: Higher order derivatives and differential cryptanalysis. In: Blahut, R.E., Costello, D.J., Maurer, U., Mittelholzer, T. (eds.) Communications and Cryptography, pp. 227–233. Springer, Boston (1994). https://doi.org/10.1007/978-1-4615-2694-0_23
18. Liu, M.: Degree evaluation of NFSR-based cryptosystems. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017. LNCS, vol. 10403, pp. 227–249. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63697-9_8
19. Mroczkowski, P., Szmidt, J.: The cube attack on stream cipher Trivium and quadraticity tests. Fundam. Inform. **114**(3–4), 309–318 (2012)
20. Murphy, S., Robshaw, M.J.B.: Essential algebraic structure within the AES. In: Yung, M. (ed.) CRYPTO 2002. LNCS, vol. 2442, pp. 1–16. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-45708-9_1
21. Sun, B., Hai, X., Zhang, W., Cheng, L., Yang, Z.: New observation on division property. Sci. China Inf. Sci. **60**(9), 1–3 (2016). https://doi.org/10.1007/s11432-015-0376-x
22. Sun, L., Wang, W., Wang, M.: Automatic search of bit-based division property for ARX ciphers and word-based division property. In: Takagi, T., Peyrin, T. (eds.) ASIACRYPT 2017. LNCS, vol. 10624, pp. 128–157. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-70694-8_5
23. Sun, S., Hu, L., Wang, P., Qiao, K., Ma, X., Song, L.: Automatic security evaluation and (related-key) differential characteristic search: application to SIMON, PRESENT, LBlock, DES(L) and other bit-oriented block ciphers. In: Sarkar, P., Iwata, T. (eds.) ASIACRYPT 2014. LNCS, vol. 8873, pp. 158–178. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-45611-8_9
24. Todo, Y.: Integral cryptanalysis on full MISTY1. In: Gennaro, R., Robshaw, M. (eds.) CRYPTO 2015. LNCS, vol. 9215, pp. 413–432. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-47989-6_20

25. Todo, Y.: Structural evaluation by generalized integral property. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015. LNCS, vol. 9056, pp. 287–314. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46800-5_12
26. Todo, Y., Isobe, T., Hao, Y., Meier, W.: Cube attacks on non-blackbox polynomials based on division property. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017. LNCS, vol. 10403, pp. 250–279. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63697-9_9
27. Todo, Y., Morii, M.: Bit-based division property and application to Simon family. In: Peyrin, T. (ed.) FSE 2016. LNCS, vol. 9783, pp. 357–377. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-52993-5_18
28. Wang, Q., Grassi, L., Rechberger, C.: Zero-sum partitions of PHOTON permutations. In: Smart, N.P. (ed.) CT-RSA 2018. LNCS, vol. 10808, pp. 279–299. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-76953-0_15
29. Wang, Q., Hao, Y., Todo, Y., Li, C., Isobe, T., Meier, W.: Improved division property based cube attacks exploiting algebraic properties of superpoly. In: Shacham, H., Boldyreva, A. (eds.) CRYPTO 2018. LNCS, vol. 10991, pp. 275–305. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-96884-1_10
30. Wang, S., Hu, B., Guan, J., Zhang, K., Shi, T.: MILP-aided method of searching division property using three subsets and applications. In: Galbraith, S.D., Moriai, S. (eds.) ASIACRYPT 2019. LNCS, vol. 11923, pp. 398–427. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-34618-8_14
31. Xiang, Z., Zhang, W., Bao, Z., Lin, D.: Applying MILP method to searching integral distinguishers based on division property for 6 lightweight block ciphers. In: Cheon, J.H., Takagi, T. (eds.) ASIACRYPT 2016. LNCS, vol. 10031, pp. 648–678. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53887-6_24
32. Ye, C., Tian, T.: Revisit division property based cube attacks: Key-recovery or distinguishing attacks? IACR Trans. Symmetric Cryptol. **2019**(3), 81–102 (2019)