Modeling for Three-Subset Division Property Without Unknown Subset

Improved Cube Attacks Against Trivium and Grain-128AEAD

Yonglin Hao[1](✉), Gregor Leander[2], Willi Meier[3], Yosuke Todo[4](✉), and Qingju Wang[5]

[1] State Key Laboratory of Cryptology, P.O. Box 5159, Beijing 100878, China
`haoyonglin@yeah.net`

[2] Horst Görtz Institute for IT Security, Ruhr University Bochum, Bochum, Germany
`gregor.leander@rub.de`

[3] FHNW, Windisch, Switzerland
`willimeier48@gmail.com`

[4] NTT Secure Platform Laboratories, Tokyo 180-8585, Japan
`yosuke.todo.xt@hco.ntt.co.jp`

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

Abstract. A division property is a generic tool to search for integral distinguishers, and automatic tools such as MILP or SAT/SMT allow us to evaluate the propagation efficiently. In the application to stream ciphers, it enables us to estimate the security of cube attacks theoretically, and it leads to the best key-recovery attacks against well-known stream ciphers. However, it was reported that some of the key-recovery attacks based on the division property degenerate to distinguishing attacks due to the inaccuracy of the division property. Three-subset division property (without unknown subset) is a promising method to solve this inaccuracy problem, and a new algorithm using automatic tools for the three-subset division property was recently proposed at Asiacrypt2019. In this paper, we first show that this state-of-the-art algorithm is not always efficient and we cannot improve the existing key-recovery attacks. Then, we focus on the feature of the three-subset division property without unknown subset and propose another new efficient algorithm using automatic tools. Our algorithm is more efficient than existing algorithms, and it can improve existing key-recovery attacks. In the application to Trivium, we show a 841-round key-recovery attack. We also show that a 855-round key-recovery attack, which was proposed at CRYPTO2018, has a critical flaw and does not work. As a result, our 841-round attack becomes the best key-recovery attack. In the application to Grain-128AEAD, we show that the known 184-round key-recovery attack degenerates to distinguishing attacks. Then, the distinguishing attacks are improved up to 189 rounds, and we also show the best key-recovery attack against 190 rounds.

Keywords: Stream ciphers · Cube attack · Division property · Three-subset division property · MILP · Trivium · Grain-128AEAD

A. Canteaut and Y. Ishai (Eds.): EUROCRYPT 2020, LNCS 12105, pp. 466–495, 2020.
https://doi.org/10.1007/978-3-030-45721-1_17

1 Introduction

Division Property. Integral cryptanalysis [1], a.k.a. Square attacks [2] or higher-order differential attacks [3], are one of the most powerful cryptanalysis techniques. Let C_I be the set of chosen plaintexts. The integral distinguisher for a cipher E_k is defined as the property $\bigoplus_{p \in C_I} E_k(p) = 0$ for any secret key k. Since the probability that such a zero-sum property holds is low for ideal ciphers, we can distinguish E_k from an ideal one.

The division property, as originated in [4], is the most accurate and generic tool to search for integral distinguishers. Ever since its proposal, it has been widely applied to many block ciphers ([5–8] etc.). For a set of texts $\mathbb{X} \subseteq \mathbb{F}_2^n$, its division property is defined by dividing a set of $\boldsymbol{u}$'s into two subsets: vectors $\boldsymbol{u} \in \mathbb{F}_2^n$ of the 1st subset satisfy $\bigoplus_{\boldsymbol{x} \in \mathbb{X}} \boldsymbol{x}^{\boldsymbol{u}} = 0$ (referred as *0-subset* hereafter), and those of the 2nd subset make $\bigoplus_{\boldsymbol{x} \in \mathbb{X}} \boldsymbol{x}^{\boldsymbol{u}}$ undetermined (referred as *unknown subset* hereafter). The initial division property is defined according to a set of chosen plaintexts, and those of the intermediate states are deduced round by round according to propagation rules. Finally, the division property for the set of corresponding ciphertexts is evaluated, and the integral distinguisher can be derived accordingly. The propagation of the division property was evaluated with the breadth-first search algorithm in [4,5,7], but it is computationally impractical for ciphers with large block size. Then, Xiang et al. introduced the useful concept called *division trail* and propose an MILP-based algorithm [9], enabling us to apply the division property to various ciphers ([10–12] etc.). Nowadays, the division property is often used not only for third party cryptanalysis but also for the design of new ciphers ([13,14] etc.).

Although the division property can find more accurate integral distinguishers than other methods, the accuracy is never perfect. As is pointed out by Todo and Morii [7], the practically verified 15-round integral distinguisher for Simon32 [15] cannot be proved with the conventional division property. To find more accurate distinguishers, the three-subset division property was proposed in [7]. A set of $\boldsymbol{u}$'s is divided into three subsets rather than two ones: the first one is the *0-subset*, another one is the *unknown subset*, and the third one is the subset satisfying $\bigoplus_{\boldsymbol{x} \in \mathbb{X}} \boldsymbol{x}^{\boldsymbol{u}} = 1$ (referred as *1-subset* hereafter). The three-subset division property enables us to prove the 15-round integral distinguisher of Simon32 [7].

Despite of its successful combination of the MILP and the conventional division property, the MILP modeling technique does not work quite well with the three-subset version. Very recently, two methods were proposed to tackle this problem. The first method is a variant of the three-subset division property [16]. Although it sacrifices quite some accuracy of the three-subset division property, this method has MILP-model-friendly propagation rules and improves some integral distinguishers. The latter, proposed by Wang et al. [17], models the propagation for the three-subset division property accurately. Wang et al.'s idea is to combine the MILP with the original breadth-first search algorithm [7]. In their algorithm, each node on the breadth-first search algorithm is regarded as the starting point of division trails, and the MILP evaluates whether there is a feasible solution from every node. When there is no feasible solution, we can prune these nodes from the breadth-first search algorithm as redundant ones.

Cube Attack. The cube attack was proposed by Dinur and Shamir in [18]. For a cipher with public variables $\boldsymbol{v} \in \mathbb{F}_2^m$ and secret variables $\boldsymbol{x} \in \mathbb{F}_2^n$, the cipher can be regarded as a polynomial of $\boldsymbol{v}, \boldsymbol{x}$ denoted as $f(\boldsymbol{x}, \boldsymbol{v})$. A set of indices, referred as the *cube indices*, is selected as $I = \{i_1, i_2, \ldots, i_{|I|}\} \subset \{1, 2, \ldots, m\}$. Such an I determines a specific structure called *cube*, denoted as C_I, containing $2^{|I|}$ values where variables in $\{v_{i_1}, v_{i_2}, \ldots, v_{i_{|I|}}\}$ take all possible combinations of values and all remaining (key and non-cube IV) variables are static. Then the sum of f over all values of the cube C_I is

$$\bigoplus_{C_I} f(\boldsymbol{x}, \boldsymbol{v}) = \bigoplus_{C_I} (t_I \cdot p(\boldsymbol{x}, \boldsymbol{v}) + q(\boldsymbol{x}, \boldsymbol{v})) = p(\boldsymbol{x}, \boldsymbol{v}),$$

where t_I denotes a monomial as $t_I = v_{i_1} \cdot v_{i_2} \cdots v_{i_{|I|}}$, and each term of $q(\boldsymbol{x}, \boldsymbol{v})$ misses at least one variable from $\{v_{i_1}, v_{i_2}, \ldots, v_{i_{|I|}}\}$. Then, $p(\boldsymbol{x}, \boldsymbol{v})$ is called the *superpoly* of the cube C_I. The cube attack consists of two steps. First, attackers recover the superpoly in the offline phase. Then, attackers query the cube to the encryption oracle, compute the summation, and get the value of the superpoly. The secret key can be recovered when the polynomial $p(\boldsymbol{x}, \boldsymbol{v})$ is simple. Therefore, the superpoly recovery plays the critical role in the cube attack.

Previously, superpolies could only be recovered experimentally. Therefore, the size of cube indices $|I|$ had to be limited within practical reach. In [11], the division property was first introduced to cube attacks, and it enables us to identify the secret variables **NOT** involved in the superpoly efficiently. After removing such secret variables, the remaining variables are stored into the set J as the secret variables that might be involved. It enables the attackers to recover the truth table of the superpoly with a time complexity $2^{|I|+|J|}$. Then, Wang et al. improved it by introducing flag and term enumeration technique that can lower the complexities for the superpoly recoveries [12]. It is noticeable that neither [11] nor [12] recovers the superpoly directly, and it only guarantees the time complexity to recover the superpoly $p(\boldsymbol{x}, \boldsymbol{v})$. They only identify the key variables (or monomials [12]) and make the assumption that such variables (monomials) might be involved in the superpoly. If such an assumption does not hold, the superpoly can be much simpler than estimated, or even in the extreme case: $p \equiv 0$ degenerates key-recovery attacks to distinguishing attacks. Such degeneration issues are reported in [19] and [17], where Wang et al.'s attack on 839-round Trivium in [12] cannot recover secret keys because $p \equiv 0$.

Motivation. Our work is motivated by the latest three-subset division property model with pruning technique [17]. In its application to the cube attack, they claim that the three-subset division property without unknown subset can recover the actual superpoly because it deterministically divides the set of $\boldsymbol{u} \in \mathbb{F}_2^n$ into two subsets whose summations are either 0 or 1. We do not need to assume

Table 1. Summary of flaws or issues in some of the previous best key-recovery attacks

Cipher	# Rounds	Ref.	Note	Where discovered
Trivium	839	[12]	Degeneration to distinguisher	[17,19]
Trivium	855	[20]	Attack does not work because of a flaw in the degree estimation	This paper
Grain-128a	184	[12]	Degeneration to distinguisher	This paper

the accuracy of the division property, and the recovered superpolies are always accurate. In spite of such a powerful tool, it was used to degenerate the key-recovery attack against 839-round Trivium in [12]. Such a degeneration from key-recovery to distinguisher implies unexpectedly simpler superpolies. Therefore, we can expect that the superpolies for 840-round Trivium are also simpler than previous estimations, and the key-recovery attacks can be carried out to 840 or more rounds. Thus, we implemented and executed the algorithm based on the pruning technique, and we find that the algorithm is not always efficient: we cannot recover the superpoly of 840-round Trivium in reasonable time. To recover the more complicated superpoly, a more efficient algorithm for the three-subset division property is required.

Our Contribution. We propose a new modeling method for the three-subset division property without unknown subset. Here, we first introduce a modified three-subset division property that is completely equivalent with the three-subset division property without unknown subset. While the original three-subset division property without unknown subset is defined by using the set $\mathbb{L}$, the modified one is defined by using the multiset $\tilde{\mathbb{L}}$ instead of the set $\mathbb{L}$, and it is suited to modeling with MILP or SAT/SMT solvers. The previous algorithm focuses on the feasibility of the model, but our algorithm focuses on all feasible solutions that are enumerated by using the solver.

To demonstrate the efficiency of our new algorithm, we apply it to cube and cube-like attacks against Trivium and Grain-128AEAD. We have two types of contributions. The first one is to show flaws or issues in some of the best previous key-recovery attacks, and these results are summarized in Table 1. The second one is the best key-recovery attacks against Trivium and Grain-128AEAD, and these results are summarized in Table 2.

We first apply our algorithm to the superpoly recovery for 840-round Trivium, which was impossible in the previous algorithm. As a result, we can recover the exact superpoly for not only 840-round Trivium but also for 841-round Trivium. Moreover, the recovered superpolies are simple balanced Boolean functions. In other words, we can recover 1-bit of information on the secret key against 840- and 841-round Trivium, and exhaustive search with the recovered superpoly allows us to recover the entire secret key with the time complexity 2^{79}. Note that the recovered superpoly is accurate and there is no assumption like in the theoretical superpoly recoveries [11,12]. We next use our algorithm to

Table 2. Summary of our results

Cipher	# Rounds	Type of attacks	Time complexity
Trivium	840	Key recovery	2^{79}
Trivium	841	Key recovery	2^{79}
Grain-128AEAD	184, 185, 186, 187, 188, 189	Distinguisher	2^{96}
Grain-128AEAD	190	Key recovery	2^{123}

verify a new-type of cube attack [20] shown by Fu et al. In the new-type of cube attack, the part of secret key bits is first guessed, one bit of the intermediate state (denoted by P_1) is computed, and the sum of $(1 + P_1) \cdot z$ over the cube is evaluated, where z denotes the key stream bit. The authors claimed that the sum of $(1 + P_1) \cdot z$ can be simpler than the sum of z by choosing P_1 appropriately. As a result, they claimed that the algebraic degree of $(1 + P_1) \cdot z$ is at most 70. Unfortunately, this claim was based on their algorithm including some man-made work that is not written in the paper, and a cluster of 600–2400 cores is necessary to run their code. Thus, no one can verify their algorithm. Our algorithm is very simple, can run on a normal PC, and recovers the exact superpoly. As we recover the superpoly of $(1+P_1)\cdot z$ over the cube, we find that the algebraic degree of $(1+P_1)\cdot z$ is not bounded by 70, and there is a monomial whose degree is $75 + 26 = 101$. In other words, even if we guess the correct P_1, the sum of $(1 + P_1) \cdot z$ over the cube is not 0. It implies that we cannot attack 855-round Trivium by using their method.

Another application is Grain-128AEAD, which was previously referred to as Grain-128a. Grain-128AEAD is one of the 2nd round candidates of the NIST LWC standardization process. And the specification is slightly revised from Grain-128a according to [21,22]. Assuming that the first pre-output key stream can be observed, there is no difference between Grain-128AEAD and Grain-128a in the context of the cube attack. As a result, we show that the key-recovery attack against 184-round Grain-128AEAD shown in [12] is a distinguisher rather than a key recovery. Moreover, we show that the distinguishing attack can be improved up to 189 rounds. From 190 rounds onwards, the superpoly involves some secret key bits, and it can be used in a key-recovery attack. However, since the recovered superpoly is highly biased toward 0, using one superpoly is not sufficient to recover any secret key bit. Therefore, we recover 15 different superpolies for 190-round Grain-128AEAD, and show an attack procedure to recover the secret key by using their superpolies. As a result, we can recover the secret key of 190-round Grain-128AEAD with 2^{123} time complexity.

2 Brief Introduction of Division Property

We first introduce some notations for bitvectors. For any bitvector $\boldsymbol{x} \in \mathbb{F}_2^m$, $x[i]$ denotes the ith bit of $\boldsymbol{x}$. Given two bitvectors $\boldsymbol{x} \in \mathbb{F}_2^m$ and $\boldsymbol{u} \in \mathbb{F}_2^m$, $\boldsymbol{x}^{\boldsymbol{u}} = \prod_{i=1}^{m} x[i]^{u[i]}$. Moreover, $\boldsymbol{x} \succeq \boldsymbol{u}$ denotes $x[i] \geq u[i]$ for all $i \in \{1, 2, \ldots, m\}$.

2.1 Conventional Division Property

The (conventional) division property was proposed at Eurocrypt 2015, and it is regarded as the generalization of the integral property.

Definition 1 ((Bit-based) division property). *Let $\mathbb{X}$ be a multiset whose elements take a value of $\mathbb{F}_2^m$, and $\boldsymbol{k} \in \mathbb{F}_2^m$. When the multiset $\mathbb{X}$ has the division property $\mathcal{D}_{\mathbb{K}}^{1^m}$, it fulfills the following conditions:*

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

For example, when a multiset $\mathbb{X} \subset \mathbb{F}_2^4$ has the division property $\mathcal{D}^{1^4}_{\{1100,1010,0011\}}$, it guarantees that $\bigoplus_{x\in\mathbb{X}} \boldsymbol{x}^{\boldsymbol{u}} = 0$ for any $\boldsymbol{u} \in \{0000, 1000, 0100, 0010, 0001, 1001, 0110, 0101\}$.

2.2 Three-Subset Division Property

The set of u is divided into two subsets in the conventional division property, where one is the subset such that $\bigoplus_{\boldsymbol{x}\in\mathbb{X}} \boldsymbol{x}^{\boldsymbol{u}}$ is unknown and the other is the subset such that the sum is 0. Three-subset division property was proposed in [7], where the number of divided subsets is extended from two to three.

Definition 2 (Three-subset division property). *Let $\mathbb{X}$ be a multiset whose elements take a value of $\mathbb{F}_2^m$, and $\boldsymbol{k} \in \mathbb{F}_2^m$. When the multiset $\mathbb{X}$ has the three-subset division property $\mathcal{D}^{1^m}_{\mathbb{K},\mathbb{L}}$, it fulfills the following conditions:*

$$\bigoplus_{\boldsymbol{x} \in \mathbb{X}} \boldsymbol{x}^{\boldsymbol{u}} = \begin{cases} \text{unknown} & \textit{if there are } \boldsymbol{k} \in \mathbb{K} \textit{ s.t. } \boldsymbol{u} \succeq \boldsymbol{k}, \\ 1 & \textit{else if there is } \boldsymbol{\ell} \in \mathbb{L} \textit{ s.t. } \boldsymbol{u} = \boldsymbol{\ell}, \\ 0 & \textit{otherwise.} \end{cases}$$

For example, when a multiset $\mathbb{X} \subset \mathbb{F}_2^4$ has the three-subset division property $\mathcal{D}^{1^4}_{\mathbb{K},\mathbb{L}}$, where $\mathbb{K} = \{1100, 1010, 0011\}$ and $\mathbb{L} = \{1000, 0010, 0110\}$, it guarantees that $\bigoplus_{\boldsymbol{x}\in\mathbb{X}} \boldsymbol{x}^{\boldsymbol{u}}$ is 0 for any $\boldsymbol{u} \in \{0000, 0100, 0001, 1001, 0101\}$ and 1 for any $\boldsymbol{u} \in \{1000, 0010, 0110\}$.

2.3 Propagation Rules for Division Property

The propagation rule of the division property is shown for three basic operations: "`copy`," "`and`," and "`xor`" in [7].

Rule 1 (`copy`). Let F be a copy function, where the input $\boldsymbol{x} \in \mathbb{F}_2^m$ and the output is calculated as $(x[1], x[1], x[2], x[3], \ldots, x[m])$. Let $\mathbb{X}$ and $\mathbb{Y}$ be the input and output multisets, respectively. Assuming that $\mathbb{X}$ has $\mathcal{D}^{1^m}_{\mathbb{K},\mathbb{L}}$, $\mathbb{Y}$ has $\mathcal{D}^{1^{m+1}}_{\mathbb{K}',\mathbb{L}'}$, where $\mathbb{K}'$ and $\mathbb{L}'$ are computed as

$$\mathbb{K}' \leftarrow \begin{cases} (0,0,k[2],\ldots,k[m]), & \text{if } k[1]=0 \\ (1,0,k[2],\ldots,k[m]),(0,1,k[2],\ldots,k[m]), & \text{if } k[1]=1 \end{cases},$$

$$\mathbb{L}' \leftarrow \begin{cases} (0,0,\ell[2],\ldots,\ell[m]), & \text{if } \ell[1]=0 \\ (1,0,\ell[2],\ldots,\ell[m]),(0,1,\ell[2],\ldots,\ell[m]),(1,1,\ell[2],\ldots,\ell[m]) & \text{if } \ell[1]=1 \end{cases}.$$

from all $\boldsymbol{k} \in \mathbb{K}$ and all $\boldsymbol{\ell} \in \mathbb{L}$, respectively. Here, $\mathbb{K}' \leftarrow \boldsymbol{k}$ (resp. $\mathbb{L}' \leftarrow \boldsymbol{\ell}$) denotes that $\boldsymbol{k}$ (resp. $\boldsymbol{\ell}$) is inserted into $\mathbb{K}'$ (resp. $\mathbb{L}'$).

Rule 2 (`and`). Let F be a function compressed by an AND, where the input $\boldsymbol{x} \in \mathbb{F}_2^m$ and the output is calculated as $(x[1] \wedge x[2], x[3], \ldots, x[m])$. Let $\mathbb{X}$ and $\mathbb{Y}$ be the input and output multisets, respectively. Assuming that $\mathbb{X}$ has $\mathcal{D}^{1^m}_{\mathbb{K},\mathbb{L}}$, $\mathbb{Y}$ has $\mathcal{D}^{1^{m-1}}_{\mathbb{K}',\mathbb{L}'}$, where $\mathbb{K}'$ is computed from all $\boldsymbol{k} \in \mathbb{K}$ as

$$\mathbb{K}' \leftarrow \left(\left\lceil \frac{k[1]+k[2]}{2} \right\rceil, k[3], k[4], \ldots, k[m] \right).$$

Moreover, $\mathbb{L}'$ is computed from all $\boldsymbol{\ell} \in \mathbb{L}$ s.t. $(\ell_1, \ell_2) = (0,0)$ or $(1,1)$ as

$$\mathbb{L}' \leftarrow \left(\left\lceil \frac{\ell[1]+\ell[2]}{2} \right\rceil, \ell[3], \ell[4], \ldots, \ell[m] \right).$$

Rule 3 (`xor`). Let F be a function compressed by an XOR, where the input $\boldsymbol{x} \in \mathbb{F}_2^m$, and the output is calculated as $(x[1] \oplus x[2], x[3], \ldots, x[m])$. Let $\mathbb{X}$ and $\mathbb{Y}$ be the input and output multisets, respectively. Assuming that $\mathbb{X}$ has $\mathcal{D}^{1^m}_{\mathbb{K},\mathbb{L}}$, $\mathbb{Y}$ has $\mathcal{D}^{1^{m-1}}_{\mathbb{K}',\mathbb{L}'}$, where $\mathbb{K}'$ is computed from all $\boldsymbol{k} \in \mathbb{K}$ s.t. $(k[1], k[2]) = (0,0)$, $(1,0)$, or $(0,1)$ as

$$\mathbb{K}' \leftarrow (k[1]+k[2], k[3], k[4], \ldots, k[m]).$$

Moreover, $\mathbb{L}'$ is computed from all $\boldsymbol{\ell} \in \mathbb{L}$ s.t. $(\ell[1], \ell[2]) = (0,0)$, $(1,0)$, or $(0,1)$ as

$$\mathbb{L}' \overset{\mathtt{x}}{\leftarrow} (\ell[1]+\ell[2], \ell[3], \ell[4], \ldots, \ell[m]).$$

Here, $\mathbb{L}' \overset{\mathtt{x}}{\leftarrow} \boldsymbol{\ell}$ denotes that $\boldsymbol{\ell}$ is inserted if it is not included in $\mathbb{L}'$. If it is already included in $\mathbb{L}'$, $\boldsymbol{\ell}$ is removed from $\mathbb{L}'$. Hereinafter, we call this property the *cancellation property*.

Another important rule is that bitvectors in $\mathbb{L}$ influence $\mathbb{K}$. Assuming that a state has $\mathcal{D}^{1^m}_{\mathbb{K},\mathbb{L}}$, the secret key is XORed with the first bit in the state. Then, for all $\boldsymbol{\ell} \in \mathbb{L}$ satisfying $\ell[1] = 0$, a new bitvector $(1, \ell[2], \ldots, \ell[m])$ is generated and stored into $\mathbb{K}$. Hereinafter, we call this property the *unknown-producing property*.

2.4 Various Algorithms to Evaluate Propagation of Division Property and Three-Subset Division Property

Breadth-First Search Algorithm. Evaluating the propagation of the division property is not easy. The first few papers [4,5,7] use the so-called breadth-first search algorithm, where $\mathbb{K}_{i+1}$ (resp. $\mathbb{L}_{i+1}$) is computed from $\mathbb{K}_i$ (resp. $\mathbb{L}_i$) from $i = 0$ to $i = R - 1$ step by step to evaluate R-round ciphers. Each node in the depth level i corresponds to each bitvector in $\mathbb{K}_i$ and $\mathbb{L}_i$. When the block length is large, the sizes of $\mathbb{K}_i$ and $\mathbb{L}_i$ increase explosively. Therefore, we cannot manage all nodes, and the in breadth-first search algorithm becomes impractical.

MILP Modeling for Conventional Division Property. Xiang et al. showed that a mixed integer linear programming (MILP) can efficiently evaluate the propagation of the conventional division property [9]. First, they introduced the *division trail* as follows.

Definition 3 (Division Trail). *Let $\mathcal{D}_{\mathbb{K}_i}$ be the division property of the input for the ith round function. Let us consider the propagation of the division property $\{\boldsymbol{k}\} \overset{\text{def}}{=} \mathbb{K}_0 \rightarrow \mathbb{K}_1 \rightarrow \mathbb{K}_2 \rightarrow \cdots \rightarrow \mathbb{K}_r$. Moreover, for any bitvector $\boldsymbol{k}^*_{i+1} \in \mathbb{K}_{i+1}$, there must exist a bitvector $\boldsymbol{k}^*_i \in \mathbb{K}_i$ such that $\boldsymbol{k}^*_i$ can propagate to $\boldsymbol{k}^*_{i+1}$ by the propagation rule of the division property. Furthermore, for $(\boldsymbol{k}_0, \boldsymbol{k}_1, \ldots, \boldsymbol{k}_r) \in (\mathbb{K}_0 \times \mathbb{K}_1 \times \cdots \times \mathbb{K}_r)$ if $\boldsymbol{k}_i$ can propagate to $\boldsymbol{k}_{i+1}$ for all $i \in \{0, 1, \ldots, r-1\}$, we call $(\boldsymbol{k}_0 \rightarrow \boldsymbol{k}_1 \rightarrow \cdots \rightarrow \boldsymbol{k}_r)$ an r-round division trail.*

Let E_k be the target r-round iterated cipher. If we can prove that there is no division trail $\boldsymbol{k}_0 \xrightarrow{E_k} \boldsymbol{e}_i$, which is an unit vector whose ith element is 1, the ith bit of r-round ciphertexts is always balanced.

Using MILP we can efficiently solve this problem. Three fundamental operations, i.e., `copy`, `xor`, and `and`, can be modeled by using MILP. We generate an MILP model that covers all division trails, and the MILP solver evaluates the feasibility whether there are division trails from the input division property to the output one or not. If the solver guarantees that there is no division trail, we can prove that the target bit is balanced.

MILP Modeling for Variant Three-Subset Division Property. Unlike the conventional division property, evaluating the propagation of the three-subset division property is difficult. The main difficulty comes from the cancellation property in Rule 3 (`xor`) and the unknown-producing property. The cancellation property implies that just focusing on the single trail is not enough, and the unknown-producing property implies that we need to know $\mathbb{L}_i$ when the secret key is XORed.

Hu and Wang tackled this problem [16], and they built the so-called variant three-subset division property, where only the cancellation property is neglected from the original one. The accuracy of the variant three-subset division property is worse than the original three-subset division property because of this neglect. However, they showed that such a variant is still useful and it is at least more accurate than the conventional division property.

Pruning Technique for Three-Subset Division Property. The technique for the accurate modeling for three-subset division property was proposed by Wang et al. [17]. The new idea is the combination between the breadth-first search algorithm and an intelligent MILP-based pruning technique. The first step of their algorithm is the same as the breadth-first search algorithm. The pruning technique is applied to $\mathbb{K}_i$ and $\mathbb{L}_i$ for every i. For all $\boldsymbol{\ell} \in \mathbb{L}_i$, we create an MILP model of the conventional division property for the $(R-i)$-round cipher, and evaluate the feasibility of the division trail from $\boldsymbol{\ell}$ to the observed bit. Then, the bitvector $\boldsymbol{\ell}$ can be removed from $\mathbb{L}_i$ if it is infeasible. We also apply the similar pruning technique to $\mathbb{K}_i$. As a result, this pruning technique allows the sizes of $\mathbb{K}_i$ and $\mathbb{L}_i$ to decrease dramatically, and the evaluation of the three-subset division property becomes possible.

They applied this new modeling technique to Simon, Simeck, PRESENT, RECTANGLE, LBlock, and TWINE. Moreover, they also applied this algorithm to the cube attack against Trivium. As a result, they showed that the 839-round key recovery attack proposed in [12] degenerates into a zero-sum distinguisher.

3 Cube Attack and Division Property

3.1 Cube Attack

The cube attack was proposed by Dinur and Shamir in [18]. A cipher is regarded as a public Boolean function whose input is divided into two parts: secret variables $\boldsymbol{x}$ and public ones $\boldsymbol{v}$. Then, the algebraic normal form of the Boolean function is represented as

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

For a set of indices $I = i_1, i_2, \ldots, i_{|I|} \subset \{1,2,\ldots,m\}$, which is referred as cube indices, t_I denotes a monomial as $t_I = v_{i_1} \cdot v_{i_2} \cdots v_{i_{|I|}}$. The Boolean function $f(\boldsymbol{x},\boldsymbol{v})$ can also be decomposed as

$$f(\boldsymbol{x},\boldsymbol{v}) = t_I \cdot p(\boldsymbol{x},\boldsymbol{v}) + q(\boldsymbol{x},\boldsymbol{v}).$$

Let C_I, which is referred as a cube (defined by I), be a set of $2^{|I|}$ values where variables in $\{v_{i_1}, v_{i_2}, \ldots, v_{i_{|I|}}\}$ are taking all possible combinations of values, and all remaining variables are fixed to any value. The sum of f over all values of the cube C_I is

$$\bigoplus_{C_I} f(\boldsymbol{x},\boldsymbol{v}) = \bigoplus_{C_I} t_I \cdot p(\boldsymbol{x},\boldsymbol{v}) + \bigoplus_{C_I} q(\boldsymbol{x},\boldsymbol{v}) = p(\boldsymbol{x},\boldsymbol{v})$$

because $t_I = 1$ for only one case in C_I and each term in $q(\boldsymbol{x},\boldsymbol{v})$ misses at least one variable from $\{v_{i_1}, v_{i_2}, \ldots, v_{i_{|I|}}\}$. Then, $p(\boldsymbol{x},\boldsymbol{v})$ is called the superpoly of the cube C_I, and the goal of the cube attack is to recover the superpoly.

3.2 Division Property and Cube Attack

The division property is formally developed as the generalization of the integral property, and it has been initially used to evaluate the integral distinguisher. When the division property is applied to the cube attack [11], the authors showed the relationship between the division property and the algebraic normal form of public functions.

Lemma 1 **([11]).** *Let $f(\boldsymbol{x})$ be a polynomial from $\mathbb{F}_2^n$ to $\mathbb{F}_2$ and $a_{\boldsymbol{u}}^f \in \mathbb{F}_2$ $(u \in \mathbb{F}_2^n)$ be the ANF coefficients. Let k be an n-dimensional bitvector. Then, assuming that the initial division property $\mathcal{D}_{\{\boldsymbol{k}\}}^{1^n}$ cannot propagate to $\mathcal{D}_1^1$ after evaluating the function f, $a_{\boldsymbol{u}}^f$ is always 0 for $\boldsymbol{u} \succeq \boldsymbol{k}$.*

Even if the function f is complicated and practically impossible to describe the algebraic normal form, the partial information can be recovered by using the division property. The division property based cube attack first evaluates secret variables that are not involved in the superpoly. Let $\bar{J}$ be the set of such secret variables, and the set $J := \{1, 2, \ldots, n\} \setminus \bar{J}$ denotes secret variables that could be involved in the superpoly. Then, we can recover the superpoly with the time complexity of $2^{|I|+|J|}$.

In the ANF of the superpoly recovered by the division property, if certain coefficients are 0, it is guaranteed that these coefficients are 0. However, if certain coefficients are 1, they cannot be guaranteed to be 1. Therefore, only using the division property does not allow us to recover the exact algebraic normal form. This limitation of the division property causes the so-called strong and weak assumptions in [11], i.e., they assume $a_{\boldsymbol{u}}^f = 1$ when the division property $\mathcal{D}_{\boldsymbol{u}}^{1^n}$ can propagate to $\mathcal{D}_1^1$. When these assumptions do not hold, the superpoly can be much simpler than estimated, and in the extreme case, the superpoly becomes a constant function. Then, the key-recovery attack degenerates into the distinguishing attack. Such degeneration is reported in [19] and [17], where the key-recovery attack against 839-round Trivium in [12] degenerates into the distinguishing attack.

3.3 Three-Subset Division Property and Cube Attack

The authors in [17] showed that these assumptions can be removed by using three-subset division property. Proposition 4 in [17] addresses this problem, but a more simple formula is enough for our application.

Lemma 2 **(Simple case of [17]).** *Let $f(\boldsymbol{x})$ be a polynomial from $\mathbb{F}_2^n$ to $\mathbb{F}_2$ and $a_{\boldsymbol{u}}^f \in \mathbb{F}_2$ $(\boldsymbol{u} \in \mathbb{F}_2^n)$ be the ANF coefficients. Let $\boldsymbol{\ell}$ be an n-dimensional bitvector. Then, assuming that the initial division property $\mathcal{D}_{\phi,\{\boldsymbol{\ell}\}}^{1^n}$ propagates to $\mathcal{D}_{\phi,1}^1$ after evaluating the function f, $a_{\boldsymbol{\ell}}^f = 1$.*

Note that we only consider the case that the function f is a public function. Then, since the function f is not key-dependent, the propagation for $\mathbb{K}$ and that for $\mathbb{L}$ are perfectly independent. In other words, we no longer consider the propagation for $\mathbb{K}$ because the initial division property is empty ϕ.

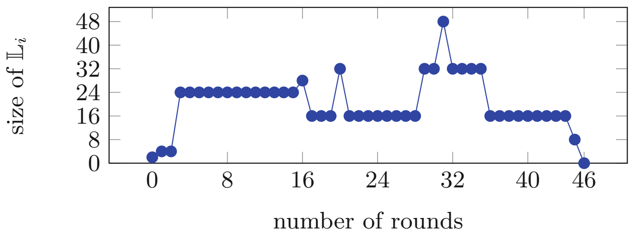


Fig. 1. Size of $\mathbb{L}_i$ after applying the pruning technique. Check if the superpoly involves $K[61]$ in the cube shown in [12].

4 Three-Subset Division Property w/o Unknown Subset

4.1 Motivation and Limitation of Pruning Technique

Our initial motivation is to verify the potential of the state-of-the-art modeling technique with the pruning technique [17]. They claimed that the exact superpoly can be recovered, but the application for the largest number of rounds was the degeneration from the key-recovery attack to a zero-sum distinguisher.[1] The natural question is why they did not show improved key-recovery attacks. Since such a degeneration implies unexpectedly simpler superpoly, we can expect that the cube described in [12] leads to a key-recovery attack for 840-round Trivium. If we can recover the superpoly of such a cube, we can directly improve the key-recovery attack against Trivium. Therefore, we implemented their algorithm by ourselves and verified whether or not we can recover the actual superpoly of 840-round Trivium. As a result, in order to make the breadth-first search algorithm with pruning technique feasible, it requires an assumption that almost all elements in $\mathbb{L}_i$ must be pruned.

We first verify that the breadth-first search algorithm with pruning technique is feasible to prove that the 839-round cube attack shown in [12] cannot recover any secret key bit. In this attack, the number of cube bits is 78, where all IV bits except for $IV[34]$ and $IV[47]$ are active and these constant bits are fixed as $(IV[34], IV[47]) = (0, 1)$. Then, the conventional division property shows that a secret key bit $K[61]$ could be involved in the superpoly [12]. We now evaluate the same cube by using the three-subset division property. According to [17], the corresponding initial property $\mathbb{L}_0$ consists of sixteen 288-bit bitvectors, where 1 is assigned for cube bits and involved-key bit, any value is assigned for four constant-1 bits $(s_{93+47}, s_{286}, s_{287}, s_{288})$, and 0 is assigned for other bits. We applied the pruning technique to sixteen bitvectors, and only two bitvectors are remaining and the other fourteen bitvecotrs can be removed. We applied the

[1] They showed that the superpoly of 842-round Trivium can be recovered with the complexity 2^{32}, but the unit of the complexity is the breadth-first search algorithm with pruning technique. Even one unit requires to solve many MILPs, and the complexity of the algorithm is not bounded. Therefore, unlike the previous theoretical cube attack [11,12], we cannot guarantee that it is faster than the exhaustive search.

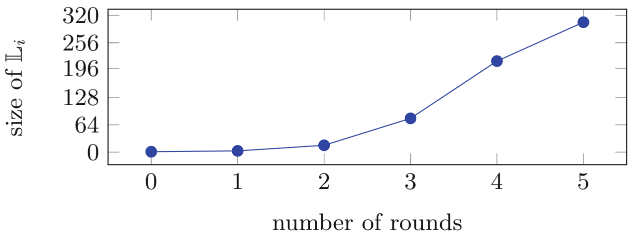


Fig. 2. Size of $\mathbb{L}_i$ after applying the pruning technique. Check if the superpoly for 840-round Trivium has constant-1 term.

pruning technique in every round, and Fig. 1 summarizes the size of $\mathbb{L}_i$ for the ith round. The size of $\mathbb{L}_i$ is bounded by a reasonable range and all bitvectors are removed in 46 rounds. It implies that the actual superpoly does not involve $K[61]$.

We next try whether or not the breadth-first search algorithm with pruning technique is available to attack 840-round Trivium. We use a cube similar to the one above, but non-cube bits $(IV[34], IV[47])$ are fixed to 0 in order for the superpoly to be more simplified. Before we recover all monomials in the superpoly, as the first step, we aim to identify if the superpoly has the constant-1 term. In other words, we evaluate whether or not 840-round Trivium has a monomial $\prod_{i\in\{1,2,\ldots,80\}\setminus\{34,47\}} s_{93+i}$. Figure 2 shows the increase of $\mathbb{L}_i$. The more the size of $\mathbb{L}_i$ increases, the more MILP instances we need to solve. We used Gurobi Optimizer on a server (Intel Xeon CPU E5-2699 v3, 18 cores, 128 GB RAM), and we spent almost two weeks to even draw Fig. 2, where only five rounds are evaluated. To recover the superpoly for 841-round Trivium, we need to finish this algorithm and apply the same algorithm to all other monomials that could be involved. Therefore, we conclude that the breadth-first search algorithm with pruning technique cannot recover the superpoly for 841-round Trivium in reasonable time. It is inefficient unless the size of $\mathbb{L}_i$ is bounded by reasonable size, e.g., 100, for all i.

4.2 Three-Subset Division Property Without Unknown Subset

The pruning technique is not always efficient to evaluate the cube attack, and we cannot improve the key-recovery attack against Trivium due to the explosive increase of $|\mathbb{L}_i|$. To address this problem, we need to develop a new modeling technique. Two properties, i.e., the unknown-producing property and the cancellation property, make it difficult to model the three-subset division property directly. Thus, we first explain how to overcome these properties.

Unknown-Producing Property. Due to the unknown-producing property, we need to evaluate the accurate $\mathbb{L}$ when the secret key is XORed. Otherwise, we cannot generate accurate bitvectors that are newly inserted to $\mathbb{K}$. Unfortunately, no efficient model is known to handle the accurate intermediate $\mathbb{L}$ by using automatic tools.

The simplest solution to address this property is the use of three-subset division property without unknown subset. Recall the definition of the division property. The unknown subset is defined as the set of $\boldsymbol{u}$ in which a parity $\bigoplus_{\boldsymbol{x}\in\mathbb{X}} \boldsymbol{x}^{\boldsymbol{u}}$ is unknown, where “unknown” means that the parity depends on the secret key. The unknown subset is used to evaluate the key-dependent function such as in block ciphers. On the other hand, when we evaluate the ANF coefficients of the public function, we do not need to use the unknown subset. At first glance, it looks like the application is restricted to public functions, but it does not matter in the application to the cube attack. Besides, if the key-schedule function is also included into the evaluated function, we can regard the block cipher as the public function.

Cancellation Property. Another property that we need to address is the cancellation property. Our idea to overcome this property is to count the number of solutions by using an MILP instead of evaluating the feasibility[2]. To understand our modeling, we introduce the following slightly modified definition. Note that this definition is equivalent to the definition of the three-subset division property without unknown subset. It is introduced only for ease of understanding of our modeling, and by itself does not yield new insight.

Definition 4 (Modified three-subset division property). *Let $\mathbb{X}$ be a multiset whose elements take a value of $\mathbb{F}_2^m$. Let $\tilde{\mathbb{L}}$ be also a* ***multiset*** *whose elements take a value of $\mathbb{F}_2^m$. When the multiset $\mathbb{X}$ has the modified three-subset division property (shortly $\mathcal{T}_{\tilde{\mathbb{L}}}^{1^m}$), it fulfils the following conditions:*

$$\bigoplus_{\boldsymbol{x}\in\mathbb{X}} \boldsymbol{x}^{\boldsymbol{u}} = \begin{cases} 1 & \textit{if there are odd-number of } \boldsymbol{u}\textit{'s in } \tilde{\mathbb{L}}, \\ 0 & \textit{otherwise.} \end{cases}$$

Note that $\boldsymbol{x}^{\boldsymbol{u}} = \prod_{i=1}^{m} x[i]^{u[i]}$.

Instead of considering the cancellation property, we count the number of appearances in each bitvector in the multiset $\tilde{\mathbb{L}}$ and check its parity. Since we do not need to consider the cancellation property, the modeling for `xor` is simplified as follows:

Rule 3' (`xor`). Let F be a function compressed by an XOR, where the input $\boldsymbol{x} \in \mathbb{F}_2^m$, and the output is calculated as $(x[1] \oplus x[2], x[3], \ldots, x[m])$. Let $\mathbb{X}$ and $\mathbb{Y}$ be the input and output multisets, respectively. Assuming that $\mathbb{X}$ has $\mathcal{T}_{\tilde{\mathbb{L}}}^{1^m}$, $\mathbb{Y}$ has $\mathcal{T}_{\tilde{\mathbb{L}}'}^{1^{m-1}}$, where $\tilde{\mathbb{L}}'$ is computed from all $\boldsymbol{\ell} \in \mathbb{L}$ s.t. $(\ell[1], \ell[2]) = (0,0)$, $(1,0)$, or $(0,1)$ as

$$\tilde{\mathbb{L}}' \leftarrow (\ell[1] + \ell[2], \ell[3], \ell[4], \ldots, \ell[m])\,.$$

[2] The same idea was already described in [17] although the authors did not use the idea in their model.

Here, $\tilde{\mathbb{L}}$ and $\tilde{\mathbb{L}}'$ are multisets, and $\tilde{\mathbb{L}}' \leftarrow \boldsymbol{\ell}$ allows the same $\boldsymbol{\ell}$ is stored into $\tilde{\mathbb{L}}'$ several times.

We no longer use insertions with the cancellation property, and the produced bitvector is always inserted to a multiset. We introduce a *three-subset division trail*, which is similar to the division trail.

Definition 5 (Three-Subset Division Trail). *Let $\mathcal{T}_{\tilde{\mathbb{L}}_i}$ be the three-subset division property of the input for the ith round function. Let us consider the propagation of the three-subset division property $\{\boldsymbol{\ell}\} \overset{\text{def}}{=} \tilde{\mathbb{L}}_0 \to \tilde{\mathbb{L}}_1 \to \tilde{\mathbb{L}}_2 \to \cdots \to \tilde{\mathbb{L}}_r$. Moreover, for any bitvector $\boldsymbol{\ell}^*_{i+1} \in \tilde{\mathbb{L}}_{i+1}$, there must exist a bitvector $\boldsymbol{\ell}^*_i \in \tilde{\mathbb{L}}_i$ such that $\boldsymbol{\ell}^*_i$ can propagate to $\boldsymbol{\ell}^*_{i+1}$ by the propagation rule of the modified three-subset division property. Furthermore, for $(\boldsymbol{\ell}_0, \boldsymbol{\ell}_1, \ldots, \boldsymbol{\ell}_r) \in (\tilde{\mathbb{L}}_0 \times \tilde{\mathbb{L}}_1 \times \cdots \times \tilde{\mathbb{L}}_r)$ if $\boldsymbol{\ell}_i$ can propagate to $\boldsymbol{\ell}_{i+1}$ for all $i \in \{0, 1, \ldots, r-1\}$, we call $(\boldsymbol{\ell}_0 \to \boldsymbol{\ell}_1 \to \cdots \to \boldsymbol{\ell}_r)$ an r-round three-subset division trail.*

The modified three-subset division property implies that we do not need to consider the cancellation property in every round. We just enumerate the number of three-subset division trails $\boldsymbol{\ell} \xrightarrow{f} \boldsymbol{e}_i$. When the number of trails is odd, the algebraic normal form of f contains $\boldsymbol{x}^{\boldsymbol{\ell}}$. Otherwise, it does not contain $\boldsymbol{x}^{\boldsymbol{\ell}}$.

In summary, removing the unknown subset allows us to skip recovering the accurate $\mathbb{L}$ when the secret key is XORed. Using multisets instead of sets allows us to handle the cancellation property by automatic tools such as MILP easily.

4.3 New Modeling Method

Unlike the pruning technique in [17], our method no longer uses the breadth-first search algorithm and it just uses an MILP model. The previous algorithm uses the MILP model for the conventional division property. On the other hand, we use the MILP model for the modified three-subset division property, and all feasible solutions are enumerated by using an off-the-shelf MILP solver[3].

Proposition 1 (MILP Model for copy). *Let $\mathtt{a} \xrightarrow{\mathtt{copy}} (\mathtt{b}_1, \mathtt{b}_2)$ be a three-subset division trail of* `copy`. *The following inequalities are sufficient to describe the propagation of the modified three-subset division property for* `copy`.

$$\begin{cases} \mathcal{M}.var \leftarrow \mathtt{a}, \mathtt{b}_1, \mathtt{b}_2 \text{ as binary.} \\ \mathcal{M}.con \leftarrow \mathtt{b}_1 + \mathtt{b}_2 \geq \mathtt{a} \\ \mathcal{M}.con \leftarrow \mathtt{a} \geq \mathtt{b}_1 \\ \mathcal{M}.con \leftarrow \mathtt{a} \geq \mathtt{b}_2 \end{cases}$$

[3] Our model is very similar to the model for variant three-subset division property proposed in [16], but there are two differences. First, we do not treat the unknown subset. Second, the goal of our model is to enumerate all feasible solutions, but the goal in [16] is to evaluate the feasibility of the model.

When the or operation is supported in the MILP solver, e.g., Gurobi optimizer supports the or operation, we can simply write $\mathcal{M}.con \leftarrow \mathtt{a} = \mathtt{b}_1 \vee \mathtt{b}_2$. Unlike the conventional division property, we need to allow the following propagation $\mathtt{1} \xrightarrow{\mathtt{copy}} (\mathtt{1}, \mathtt{1})$. Otherwise, we miss any feasible solutions.

Proposition 2 (MILP Model for `and`**).** *Let* $(\mathtt{a}_1, \mathtt{a}_2, \ldots, \mathtt{a}_\mathtt{m}) \xrightarrow{\mathtt{and}} \mathtt{b}$ *be a three-subset division trail of* `and`*. The following inequalities are sufficient to describe the propagation of the modified three-subset division property for* `and`*.*

$$\begin{cases} \mathcal{M}.var \leftarrow \mathtt{a}_1, \mathtt{a}_2, \ldots, \mathtt{a}_\mathtt{m}, \mathtt{b} \textit{ as binary.} \\ \mathcal{M}.con \leftarrow \mathtt{b} = \mathtt{a}_\mathtt{i} \textit{ for all } \mathtt{i} \in \{\mathtt{1}, \mathtt{2}, \ldots, \mathtt{m}\} \end{cases}$$

Some feasible propagation on the conventional division property becomes infeasible. For example, $(\mathtt{1}, \mathtt{1}, \mathtt{0}) \xrightarrow{\mathtt{and}} \mathtt{1}$ is feasible for the conventional division property, but it is not so in the modified three-subset division property.

Proposition 3 (MILP Model for `xor`**).** *Let* $(\mathtt{a}_1, \mathtt{a}_2, \ldots, \mathtt{a}_\mathtt{m}) \xrightarrow{\mathtt{xor}} \mathtt{b}$ *be a three-subset division trail of* `xor`*. The following inequalities are sufficient to describe the propagation of the modified three-subset division property for* `xor`*.*

$$\begin{cases} \mathcal{M}.var \leftarrow \mathtt{a}_1, \mathtt{a}_2, \ldots, \mathtt{a}_\mathtt{m}, \mathtt{b} \textit{ as binary.} \\ \mathcal{M}.con \leftarrow \mathtt{a}_1 + \mathtt{a}_2 + \cdots + \mathtt{a}_\mathtt{m} = \mathtt{b} \end{cases}$$

Note that this is the same as the one for the conventional division property.

While the goal of the previous method is to find one feasible solution or to prove its infeasibility, the goal of our method is to enumerate all feasible solutions. Three Propositions are enough to represent any cipher, but such a straightforward model sometimes increases the number of feasible solutions explosively. A more clever model is sometimes required to avoid the explosive increase of feasible (but redundant) solutions, and we discuss this in Sect. 6 in detail.

4.4 Algorithm to Recover ANF Coefficients of Public Function

Let f be a public Boolean function whose input denotes an n-bit string $\boldsymbol{x} = (x[1], x[2], \ldots, x[n])$, and let it consist of the iteration of simple public functions. Then, the algebraic normal form of f is represented as

$$f(\boldsymbol{x}) = \bigoplus_{\boldsymbol{u} \in \mathbb{F}_2^n} a_{\boldsymbol{u}}^f \boldsymbol{x}^{\boldsymbol{u}}.$$

Our goal is to recover the value of $a_{\boldsymbol{u}}^f$ for some $\boldsymbol{u}$. We first prepare an MILP model $\mathcal{M}$ that represents the modified three-subset division property of the function f. Algorithm 1 shows the algorithm to recover an ANF coefficient $a_{\boldsymbol{u}}^f$. The initial modified three-subset division property is defined by $\boldsymbol{u}$, and the number of feasible solutions is enumerated by using the MILP solver. Note that

Algorithm 1. Algorithm to recover an ANF coefficient $a_{\boldsymbol{u}}^f$

```
 1: procedure attackFramework($\mathcal{M}$, $u$)
 2:     Let $\mathtt{x_i}$ be an MILP variable of $\mathcal{M}$ corresponding to the $i$th input of $f$.
 3:     $\mathcal{M}.con \leftarrow \mathtt{x_i} = 1$ for all $i$ s.t. $u[i] = 1$.
 4:     $\mathcal{M}.con \leftarrow \mathtt{x_i} = 0$ for all $i$ s.t. $u[i] = 0$.
 5:     solve MILP model $\mathcal{M}$ and enumerate all feasible solutions
 6:     if the number of solutions is odd then
 7:         $a_{\boldsymbol{u}}^f = 1$
 8:     else
 9:         $a_{\boldsymbol{u}}^f = 0$
10:     end if
11: end procedure
```

Algorithm 2. Algorithm to recover the superpoly

```
 1: procedure attackFramework($\mathcal{M}$, $I$, $(C_0)$)
 2:     Let $\mathtt{x_i}$ be an MILP variable of $\mathcal{M}$ corresponding to the $i$th secret variable.
 3:     Let $\mathtt{v_i}$ be an MILP variable of $\mathcal{M}$ corresponding to the $i$th public variable.
 4:     $\mathcal{M}.con \leftarrow \mathtt{v_i} = 1$ for all $i \in I$
 5:     $\mathcal{M}.con \leftarrow \mathtt{v_i} = 0$ for all $i \in C_0$
 6:     prepare a hash table $J$ whose key is $(n+m)$-bit string and value is counter.
 7:     solve MILP model $\mathcal{M}$ and enumerate all feasible solutions
 8:     for all feasible solutions do
 9:         get $\boldsymbol{u} = (\mathtt{x_1}, \mathtt{x_2}, \ldots, \mathtt{x_n}, \mathtt{v_1}, \mathtt{v_2}, \ldots, \mathtt{v_m})$ in every found solution
10:         increase $J[\boldsymbol{u}]$ by 1
11:     end for
12:     prepare a polynomial $p = 0$
13:     for all $\boldsymbol{u}$ whose $J[\boldsymbol{u}]$ is an odd number do
14:         $p = p + (\boldsymbol{x} \| \boldsymbol{v})^{\boldsymbol{u}}$.
15:     end for
16:     return $p/t_I$
17: end procedure
```

the efficiency of Algorithm 1 depends on the number of feasible solutions. When there are too many solutions, it is practically impossible to enumerate all feasible solutions. In other words, the necessary condition that Algorithm 1 stops by reasonable time is that the number of feasible solutions is bounded by reasonable size, e.g., at most 2^{16}.

While Algorithm 1 is very simple, it is less efficient for the application to the cube attack because we need to recover all monomials in the superpoly. The number of monomials that Algorithm 1 can evaluate is only one. Therefore, we need to repeat Algorithm 1 many times while changing the input $\boldsymbol{u}$ until all monomials are recovered exactly. One of the advantages of our modeling method is that we can simply extend the algorithm to recover the superpoly, and the extended algorithm uses only one MILP model. Algorithm 2 shows the dedicated algorithm to recover the superpoly. Unlike Algorithm 1, the initial division property is not determined and only the part corresponding to the cube bits is fixed to 1. When we enumerate all feasible solutions under such constraints, all monomials that could be involved in the superpoly can be found as the feasible solutions. The third input C_0 is an option to declare that some public variables are fixed to 0. Specific attention should be paid to the situation that $C_0 = \phi$. In this case, Algorithm 2 gives the ANF of $p(\boldsymbol{x}, \boldsymbol{v})$ consisting of all secret and non-cube public variables. In other words, we do not need to specify the assignment of non-cube public variables in advance. This is an obvious advantage of our method over the existing breadth-first search algorithm with pruning technique. On the other hand, when the assignment of non-cube public variables is determined in advance, C_0 should be set because it decreases the number of three-subset division trails and increases the efficiency of the algorithm.

As far as we applied these algorithms to the cube attack against Trivium or Grain-128AEAD, Algorithm 2 is not only simpler but also more efficient than the iteration of Algorithm 1. Unfortunately, we cannot say the explicit reason because it depends on the inside of MILP solvers. As one observation, many three-subset division trails with different initial division property share the same trail in the last several rounds. Therefore, we expect that their trails are efficiently enumerated in Algorithm 2. On the other hand, the iteration of Algorithm 1 needs to find the shared part of trails every time.

5 Improved Cube Attacks Against Trivium

5.1 Specification of Trivium and Its MILP Model

Trivium [23] is an NLFSR-based stream cipher, and the internal state is represented by a 288-bit state $(s_1, s_2, \dots, s_{288})$. The 80-bit secret key K is loaded to the first register, and the 80-bit initialization vector IV is loaded to the second register. The other state bits are set to 0 except the last three bits in the third register. Namely, the initial state bits are represented as

$$\begin{aligned}
(s_1, s_2, \dots, s_{93}) &= (K[1], K[2], \dots, K[80], 0, \dots, 0),\\
(s_{94}, s_{95}, \dots, s_{177}) &= (IV[1], IV[2], \dots, IV[80], 0, \dots, 0),\\
(s_{178}, s_{279}, \dots, s_{288}) &= (0, 0, \dots, 0, 1, 1, 1).
\end{aligned}$$

The pseudo code of the update function is given as follows.

```
$t_1 \leftarrow s_{66} \oplus s_{93}$,                    $t_2 \leftarrow s_{162} \oplus s_{177}$,                    $t_3 \leftarrow s_{243} \oplus s_{288}$,
$z \leftarrow t_1 \oplus t_2 \oplus t_3$,
$t_1 \leftarrow t_1 \oplus s_{91}s_{92} \oplus s_{171}$,   $t_2 \leftarrow t_2 \oplus s_{175}s_{176} \oplus s_{264}$,   $t_3 \leftarrow t_3 \oplus s_{286}s_{287} \oplus s_{69}$,
```

where z denotes the key stream. The state of the next round is computed as

$$\begin{aligned}
(s_1, s_2, \dots, s_{93}) &\leftarrow (t_3, s_1, \dots, s_{92}),\\
(s_{94}, s_{95}, \dots, s_{177}) &\leftarrow (t_1, s_{94}, \dots, s_{176}),\\
(s_{178}, s_{279}, \dots, s_{288}) &\leftarrow (t_2, s_{178}, \dots, s_{287}).
\end{aligned}$$

Algorithm 3. Model for modified three-subset division property for Trivium

```
1: procedure TriviumCore($\mathcal{M}, \mathtt{x}_1, \dots, \mathtt{x}_{288}, i_1, i_2, i_3, i_4, i_5$)
2:     $\mathcal{M}.var \leftarrow \mathtt{y}_{\mathtt{i}_1}, \mathtt{y}_{\mathtt{i}_2}, \mathtt{y}_{\mathtt{i}_3}, \mathtt{y}_{\mathtt{i}_4}, \mathtt{y}_{\mathtt{i}_5}, \mathtt{z}_1, \mathtt{z}_2, \mathtt{z}_3, \mathtt{z}_4, \mathtt{a}$ as binary
3:     $\mathcal{M}.con \leftarrow \mathtt{x}_{\mathtt{i}_\mathtt{j}} = \mathtt{y}_{\mathtt{i}_\mathtt{j}} \vee \mathtt{z}_\mathtt{j}$ for all $\mathtt{j} \in \{1, 2, 3, 4\}$
4:     $\mathcal{M}.con \leftarrow \mathtt{a} = \mathtt{z}_3$
5:     $\mathcal{M}.con \leftarrow \mathtt{a} = \mathtt{z}_4$
6:     $\mathcal{M}.con \leftarrow \mathtt{y}_{\mathtt{i}_5} = \mathtt{x}_{\mathtt{i}_5} + \mathtt{a} + \mathtt{z}_1 + \mathtt{z}_2$
7:     for all $\mathtt{i} \in \{1, 2, \dots, 288\}$ w/o $\mathtt{i}_1, \mathtt{i}_2, \mathtt{i}_3, \mathtt{i}_4, \mathtt{i}_5$ do
8:         $\mathtt{y}_\mathtt{i} = \mathtt{x}_\mathtt{i}$
9:     end for
10:    return $(\mathcal{M}, \mathtt{y}_1, \dots, \mathtt{y}_{288})$
11: end procedure
1: procedure TriviumEval(round $R$)
2:     Prepare empty MILP Model $\mathcal{M}$
3:     $\mathcal{M}.var \leftarrow \mathtt{s}^0_\mathtt{i}$ for $\mathtt{i} \in \{1, 2, \dots, 288\}$
4:     for $\mathtt{i} = 81$ to $93$ and $\mathtt{i} = 93 + 80$ to $285$ do
5:         $\mathcal{M}.con \leftarrow \mathtt{s}^0_\mathtt{i} = 0$
6:     end for
7:     for $r = 1$ to $R$ do
8:         $(\mathcal{M}, \mathtt{x}_1, \dots, \mathtt{x}_{288}) = \mathtt{TriviumCore}(\mathcal{M}, \mathtt{s}^{r-1}_1, \dots, \mathtt{s}^{r-1}_{288}, 66, 171, 91, 92, 93)$
9:         $(\mathcal{M}, \mathtt{y}_1, \dots, \mathtt{y}_{288}) = \mathtt{TriviumCore}(\mathcal{M}, \mathtt{x}_1, \dots, \mathtt{x}_{288}, 162, 264, 175, 176, 177)$
10:        $(\mathcal{M}, \mathtt{z}_1, \dots, \mathtt{z}_{288}) = \mathtt{TriviumCore}(\mathcal{M}, \mathtt{y}_1, \dots, \mathtt{y}_{288}, 243, 69, 286, 287, 288)$
11:        $(\mathtt{s}^r_1, \dots, \mathtt{s}^r_{288}) = (\mathtt{z}_{288}, \mathtt{z}_1, \dots, \mathtt{z}_{287})$
12:    end for
13:    for all $\mathtt{i} \in \{1, 2, \dots, 288\}$ w/o $66, 93, 162, 177, 243, 288$ do
14:        $\mathcal{M}.\mathtt{con} \leftarrow \mathtt{s}^R_\mathtt{i} = 0$
15:    end for
16:    $\mathcal{M}.con \leftarrow (\mathtt{s}^R_{66} + \mathtt{s}^R_{93} + \mathtt{s}^R_{162} + \mathtt{s}^R_{177} + \mathtt{s}^R_{243} + \mathtt{s}^R_{288}) = 1$
17:    return $\mathcal{M}$
18: end procedure
```

In the initialization, the state is updated 1152 times without producing an output. After the initialization, one bit key stream is produced by every update function.

MILP Model. `TriviumEval` in Algorithm 3 generates a model $\mathcal{M}$ as the input of Algorithm 1 or 2, and all three-subset division trails are included as feasible solutions of this model $\mathcal{M}$. `TriviumCore` in Algorithm 3 generates MILP variables and constraints of the update function for each register.

5.2 Practical Verification

To verify our new algorithm, we select the same parameters as the one in the previous works [11,12]. Example 1 takes parameters from [11] and set the empty set ϕ for C_0. Then, Algorithm 2 recovers the algebraic normal form of $p(\boldsymbol{x}, \boldsymbol{v})$ involving all key and non-cube IV bits.

Table 3. The monomial $(\boldsymbol{x}\|\boldsymbol{v})^{\boldsymbol{u}}/t_I$'s and their $J[\boldsymbol{u}]$'s corresponding to Example 1

Parity	$J[\boldsymbol{u}]$	$(\boldsymbol{x}\|\boldsymbol{v})^{\boldsymbol{u}}/t_I$	Parity	$J[\boldsymbol{u}]$	$(\boldsymbol{x}\|\boldsymbol{v})^{\boldsymbol{u}}/t_I$
0	2	$x_{60}v_{22}$	1	1	v_9v_{20}
1	1	$x_{60}v_{19}v_{20}$	1	1	$v_6v_7v_8v_{20}$
1	1	$x_{60}v_{20}$	0	2	$v_{22}v_{72}$
1	1	$x_{60}v_6v_{20}$	1	1	v_7v_8
1	1	$x_{60}v_7$	1	1	$v_6v_9v_{20}$
1	1	$v_7v_8v_{19}v_{20}$	1	1	$v_{19}v_{20}v_{72}$
0	2	$v_7v_8v_{22}$	1	1	v_7v_9
1	1	$v_9v_{19}v_{20}$	1	1	$v_{20}v_{72}$
0	2	v_9v_{22}	1	1	$v_6v_{20}v_{72}$
1	1	$v_7v_8v_{20}$	1	1	v_7v_{72}

Example 1 **(Parameters from [11]).** We let $I = \{1, 11, 21, 31, 41, 51, 61, 71\}$ and evaluate z_{590}. We first run Algorithm 3 as $\mathcal{M} \leftarrow$ `TriviumEval`(590) and get the MILP model based three-subset division property. Then, we set $C_0 = \phi$ and acquire $p(\boldsymbol{x}, \boldsymbol{v})$ by running Algorithm 2 as $p(\boldsymbol{x}, \boldsymbol{v}) \leftarrow$ `attackFramework`$(I, \mathcal{M}, \phi)$. The monomial $(\boldsymbol{x}\|\boldsymbol{v})^{\boldsymbol{u}}/t_I$'s along with their $J[\boldsymbol{u}]$'s are listed in Table 3. The ANF of $p(\boldsymbol{x}, \boldsymbol{v})$ can therefore be determined as

$$\begin{aligned} p(x) = {} & x_{60}(v_{19}v_{20} + v_{20} + v_6v_{20} + v_7) \\ & + (v_7v_8v_{19}v_{20} + v_9v_{19}v_{20} + v_7v_8v_{20} + v_9v_{20} + v_6v_7v_8v_{20} + v_7v_8 \\ & + v_6v_9v_{20} + v_{19}v_{20}v_{72} + v_7v_9v_{20}v_{72} + v_6v_{20}v_{72} + v_7v_{72}) \end{aligned}$$

5.3 Cube Attacks Against 840-Round and 841-Round Trivium

To demonstrate that our modeling method is more efficient than the previous method, we applied it to Trivium. For R-round Trivium, the model $\mathcal{M}$ is generated as $\mathcal{M} \leftarrow$ `TriviumEval`(R) by calling Algorithm 3. Then, we set all non-cube IV bits to constant 0, i.e., for arbitrary cube I, the corresponding parameter C_0 is defined as the complement of I: $C_0 \leftarrow \{0, \ldots, 80\}\backslash I$. With such $\mathcal{M}$, I and C_0, the superpoly is defined as $p(x) \leftarrow$ `attackFramework`$(\mathcal{M}, I, C_0)$ by calling Algorithm 2. As a result, we can successfully recover the superpoly of 840-round and 841-round Trivium. In other words, we show key-recover attacks against 840- and 841-round Trivium without any assumption. The detailed parameters of the two attacks are as follows:

Superpoly of 840-Round Trivium. We used the same cube as the one shown in Sect. 4.1, i.e., the cube indices are

$$I = \{1, 2, \ldots, 33, 35, 36, \ldots, 46, 48, 49, \ldots, 80\},$$

and $IV[34] = IV[47] = 0$. Note that the previous algorithm cannot recover the corresponding superpoly as we already showed in Sect. 4.1. As a result, 12, 909 feasible three-subset division trails are enumerated, and $J[\boldsymbol{u}]$ in Algorithm 2 is non zero for 228 different $\boldsymbol{u}$'s. Out of 228 $\boldsymbol{u}$'s, there are 67 $\boldsymbol{u}$'s whose $J[\boldsymbol{u}]$ is an odd number. In other words, the superpoly is represented as the sum of 67 monomials, and the following

$$\begin{aligned}p(\boldsymbol{x}) =& 1 + x_{80} + x_{79} + x_{79}x_{80} + x_{78}x_{79} + x_{76}x_{77} + x_{75}x_{76}x_{78} + x_{75}x_{76}x_{77}\\ &+ x_{70} + x_{68} + x_{68}x_{80} + x_{68}x_{79}x_{80} + x_{68}x_{78}x_{79} + x_{68}x_{69} + x_{66}x_{67}\\ &+ x_{66}x_{67}x_{80} + x_{66}x_{67}x_{79}x_{80} + x_{66}x_{67}x_{78}x_{79} + x_{65} + x_{64}x_{66} + x_{64}x_{65}\\ &+ x_{63}x_{64} + x_{59}x_{63} + x_{54}x_{68} + x_{54}x_{66}x_{67} + x_{53}x_{68} + x_{53}x_{66}x_{67} + x_{52}\\ &+ x_{52}x_{53} + x_{51}x_{77} + x_{51}x_{75}x_{76} + x_{51}x_{52} + x_{50}x_{78} + x_{50}x_{76}x_{77} + x_{50}x_{51}\\ &+ x_{43} + x_{41} + x_{41}x_{80} + x_{41}x_{79}x_{80} + x_{41}x_{78}x_{79} + x_{41}x_{54} + x_{41}x_{53} + x_{39}\\ &+ x_{39}x_{64} + x_{38} + x_{37}x_{38} + x_{35}x_{55} + x_{33}x_{34}x_{55} + x_{27} + x_{26} + x_{22}x_{66}\\ &+ x_{22}x_{64}x_{65} + x_{22}x_{39} + x_{20}x_{21}x_{66} + x_{20}x_{21}x_{64}x_{65} + x_{20}x_{21}x_{39} + x_{12}\\ &+ x_{8}x_{78} + x_{8}x_{77} + x_{8}x_{76}x_{77} + x_{8}x_{75}x_{76} + x_{8}x_{55} + x_{8}x_{51} + x_{8}x_{50}\\ &+ x_{1}x_{35} + x_{1}x_{33}x_{34} + x_{1}x_{8}\end{aligned}$$

is the recovered superpoly, where $\boldsymbol{x} = (x_1, x_2, \ldots, x_{80})$ denotes the secret key, i.e., $x_i = K[i]$. This superpoly is a balanced Boolean function because there is a monomial x_{12} that is independent of other monomials. Therefore, we can recover 1 bit of information by using 2^{78} data and time complexities. The dominant part of the whole key recovery attack is the exhaustive search after 1-bit key recovery, which is 2^{79} time complexity.

Superpoly of 841-Round Trivium. We next aim to recover the superpoly of 841-round TRIVIUM, but it has too many trails to enumerate all of them. Therefore, we heuristically change cube indices such that the number of trails is not large. As a result, the following cube is considered:

$$I = \{1, 2, \ldots, 8, 10, 11, \ldots, 78, 80\},$$

and $IV[9] = IV[79] = 0$. As a result, 30,177 feasible three-subset division trails are enumerated, and $J[\boldsymbol{u}]$ in Algorithm 2 is non zero for 216 different $\boldsymbol{u}$'s. Out of 216 $\boldsymbol{u}$'s, there are 53 $\boldsymbol{u}$'s whose $J[\boldsymbol{u}]$ is an odd number. In other words, the superpoly $p(\boldsymbol{x})$ is represented as the sum of 53 monomials, and the following

$$\begin{aligned}p(\boldsymbol{x}) =& x_{78} + x_{76} + x_{75}x_{76} + x_{74} + x_{74}x_{75} + x_{74}x_{75}x_{77} + x_{74}x_{75}x_{76} + x_{72}x_{73}\\ &+ x_{68} + x_{67} + x_{63} + x_{61}x_{62} + x_{59} + x_{59}x_{72} + x_{59}x_{70}x_{71} + x_{59}x_{61} + x_{58}\\ &+ x_{58}x_{80} + x_{58}x_{78}x_{79} + x_{58}x_{66} + x_{58}x_{59} + x_{53}x_{58} + x_{51}x_{74} + x_{51}x_{73}\\ &+ x_{51}x_{72}x_{73} + x_{51}x_{71}x_{72} + x_{50}x_{76} + x_{50}x_{74}x_{75} + x_{49} + x_{49}x_{77}\\ &+ x_{49}x_{75}x_{76} + x_{49}x_{50}x_{74} + x_{49}x_{50}x_{73} + x_{49}x_{50}x_{72}x_{73} + x_{49}x_{50}x_{71}x_{72}\\ &+ x_{47} + x_{47}x_{51} + x_{47}x_{49}x_{50} + x_{46}x_{51} + x_{46}x_{49}x_{50} + x_{45}x_{59} + x_{36} + x_{32}\\ &+ x_{30}x_{31} + x_{24} + x_{24}x_{74} + x_{24}x_{73} + x_{24}x_{72}x_{73} + x_{24}x_{71}x_{72} + x_{24}x_{47}\\ &+ x_{24}x_{46} + x_{9} + x_{5}\end{aligned}$$

is the recovered superpoly. This superpoly is also a balanced Boolean function because there is a monomial x_5 that is independent of other monomials. Therefore, we can recover 1 bit of information by using 2^{78} data and time complexities. The dominant part of the whole key recovery attack is the exhaustive search after 1-bit key recovery, which is 2^{79} time complexity.

5.4 Verification of 855-Round Attack from CRYPTO2018 [20]

In CRYPTO2018, a new type of cube attacks was proposed, where a key recovery attack against 855-round Trivium was shown. The authors claimed the following statement.

Statement 1 **([20]).** *When* $IV[31] = IV[49] = IV[61] = IV[75] = IV[76] = 0$*, the degree of* $(1 + s_{94}^{210})z_{855}$ *is bounded by 70.*

Attackers first guess the part of a secret key involved in s_{94}^{210} and compute the sum of $(1 + s_{94}^{210})z_{855}$ over cubes whose dimension is larger than 70. When the correct key is guessed, the sum must be 0. In other words, if the sum is 1, we can discard the guessed key.

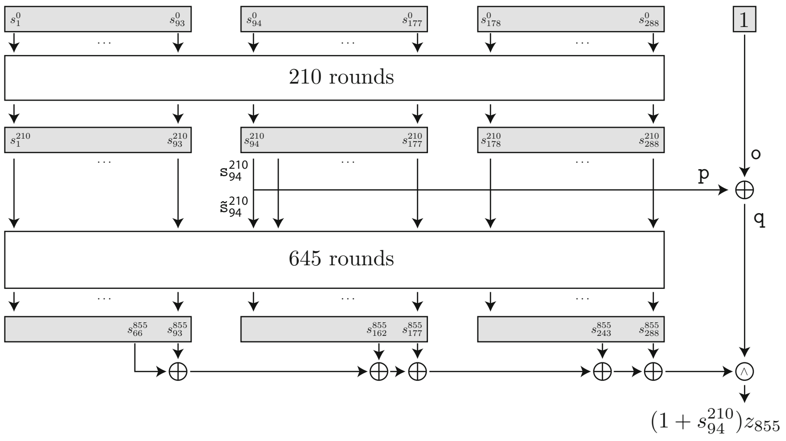
To prove Statement 1, the authors developed a new algorithm to evaluate the upper bound of the degree. However, their algorithm includes some man-made work that is not written in their paper, and a cluster of 600–2400 cores is necessary to run their code. As a result, no one can verify their algorithm and the correctness of Statement 1. The only supportive material is the practical example by using 721-round Trivium[4]. Later, Hao et al. reviewed Statement 1 by using the conventional bit-based division property [24]. They showed that the sum of $(1 + s_{94}^{210})z_{855}$ over 75-dimensional cube could involve all 80 key bits with degree bound 27. According to this result, Hao et al. pointed out that Statement 1 unlikely holds. However, as we already pointed out, the conventional bit-based division property is not always accurate. Therefore, the correctness of Statement 1 becomes an open question.

In comparison with Fu et al.'s algorithm, our algorithm using three-subset division property has three advantages:

- Cheap implementation cost. Our task is to generate an MILP model, and the complicated part is solved by using off-the-shelf MILP solvers. Our verification code using Gurobi C++ API contains about 300 lines.
- Run on the normal PC. We do not need to prepare many clusters.
- Tight bound is proven. Our algorithm can recover the ANF coefficient $a_{\boldsymbol{u}}^f$ for some $\boldsymbol{u}$ accurately.

With such a method, we inspect Statement 1.

[4] In [20], the authors showed that the degree of $(1+s_{94}^{290})z_{721}$ is bounded by 32 when the correct s_{94}^{290} is guessed. However, Hao et al. pointed out that the degree is bounded by 32 even if we guess s_{94}^{290} with incorrect secret key, as a consequence we cannot distinguish the correct key from the wrong keys [24]. Response to this error, Fu et al. reproduced the practical example for 721-round Trivium [25].



s_1^0 s_{93}^0 s_{94}^0 s_{177}^0 s_{178}^0 s_{288}^0
1
210 rounds
s_1^{210} s_{93}^{210} s_{94}^{210} s_{177}^{210} s_{178}^{210} s_{288}^{210}
s_{94}^{210}
$\tilde{s}_{94}^{210}$
o
p
q
645 rounds
s_{66}^{855} s_{93}^{855} s_{162}^{855} s_{177}^{855} s_{243}^{855} s_{288}^{855}
$(1 + s_{94}^{210})z_{855}$

Fig. 3. Overview of new type of cube attack for 855-round TRIVIUM

MILP Model to Verify 855-Round Attack. To verify Statement 1, we consider a circuit shown in Fig. 3 and generate the corresponding MILP model by calling Algorithm 4 as $\mathcal{M} \leftarrow$ `TriviumSecEval`$(855, 210)$. Corresponding to the setting of [20], we set I as the largest possible cube, i.e., $I = \{1, \ldots, 80\} \setminus \{31, 49, 61, 75, 76\}$, and all non-cube IVs are set to 0, i.e., $C_0 = \{31, 49, 61, 75, 76\}$. Then, with such $\mathcal{M}, I, C_0$, we run Algorithm 2 as $p(\boldsymbol{x}) \leftarrow$ `attackFramework`$(\mathcal{M}, I, C_0)$ to check whether $p(\boldsymbol{x})$ is constant 0. According to the result by Hao et al. by using the conventional bit-based division property, we first evaluated whether or not $p(\boldsymbol{x})$ has monomials whose degree is 27. Then, the number of appearance $J[\boldsymbol{u}]$ is non-zero for the following two 27-degree monomials

$$\prod_{i \in \{29,30,41,42,44,45,46,47,49,54,55,56,57,59,60,63,66,67,68,69,70,71,72,73,74,75,76\}} x_i,$$

$$\prod_{i \in \{29,30,41,42,43,44,45,46,47,49,54,55,56,57,59,60,63,66,67,69,70,71,72,73,74,75,76\}} x_i,$$

but $J[\boldsymbol{u}] = 2$ for the two monomials above. Therefore, these monomials do not appear in $p(\boldsymbol{x})$. We next evaluated whether or not $p(\boldsymbol{x})$ has monomials whose degree is 26. Since there are quite many candidates of $\boldsymbol{u}$ whose $J[\boldsymbol{u}]$ is non zero, we randomly picked one from these candidates and evaluated the number of trails. As a result, $J[\boldsymbol{u}] = 1$ in the following monomial

$$\prod_{i \in \{40,41,42,53,54,55,56,57,58,61,62,63,65,66,67,68,69,70,71,72,73,74,75,76,78,79\}} x_i.$$

Note that finding one $\boldsymbol{u}$ such that $J[\boldsymbol{u}]$ is an odd number is enough to disprove Statement 1.

Algorithm 4. Model for modified three-subset division property of Trivium corresponding to the Fu et al.'s method in [20]

```
 1: procedure TriviumSecEval(round $R$, sector round $R'$)
 2:     Prepare empty MILP Model $\mathcal{M}$
 3:     $\mathcal{M}.var \leftarrow \mathtt{s}_\mathtt{i}^0$ for $\mathtt{i} \in \{1,2,\ldots,288\}$ and $\mathcal{M}.var \leftarrow \mathtt{o}$
 4:     for $\mathtt{i} = 81$ to $93$ and $\mathtt{i} = 93+80$ to $285$ do
 5:         $\mathcal{M}.con \leftarrow \mathtt{s}_\mathtt{i}^0 = 0$
 6:     end for
 7:     $\mathcal{M}.var \leftarrow o$
 8:     for $i = 81$ to $93$ and $i = 93+80$ to $285$ do
 9:         $\mathcal{M}.con \leftarrow s_i^0 = 0$
10:     end for
11:     for $r = 1$ to $R$ do
12:         $(\mathcal{M}, \mathtt{x}_1,\ldots,\mathtt{x}_{288}) = \mathtt{TriviumCore}(\mathcal{M}, \mathtt{s}_1^{\mathtt{r}-1},\ldots,\mathtt{s}_{288}^{\mathtt{r}-1}, 66, 171, 91, 92, 93)$
13:         $(\mathcal{M}, \mathtt{y}_1,\ldots,\mathtt{y}_{288}) = \mathtt{TriviumCore}(\mathcal{M}, \mathtt{x}_1,\ldots,\mathtt{x}_{288}, 162, 264, 175, 176, 177)$
14:         $(\mathcal{M}, \mathtt{z}_1,\ldots,\mathtt{z}_{288}) = \mathtt{TriviumCore}(\mathcal{M}, \mathtt{y}_1,\ldots,\mathtt{y}_{288}, 243, 69, 286, 287, 288)$
15:         $(\mathtt{s}_1^\mathtt{r},\ldots,\mathtt{s}_{288}^\mathtt{r}) = (\mathtt{z}_{288}, \mathtt{z}_1,\ldots,\mathtt{z}_{287})$
16:         if $\mathtt{r} = R'$ then
17:             $\mathcal{M}.var \leftarrow \tilde{\mathtt{s}}_{94}^{\mathtt{R}'}, \mathtt{p}, \mathtt{q}$
18:             $\mathcal{M}.con \leftarrow \mathtt{s}_{94}^{\mathtt{R}'} = \tilde{\mathtt{s}}_{94}^{\mathtt{R}'} \bigvee \mathtt{p}$
19:             $\mathcal{M}.con \leftarrow \mathtt{q} = \mathtt{o} + \mathtt{p}$
20:             $\mathtt{s}_{94}^{\mathtt{R}'} = \tilde{\mathtt{s}}_{94}^{\mathtt{R}'}$
21:         end if
22:     end for
23:     for all $\mathtt{i} \in \{1,2,\ldots,288\}$ w/o $66, 93, 162, 177, 243, 288$ do
24:         $\mathcal{M}.\mathtt{con} \leftarrow \mathtt{s}_\mathtt{i}^\mathtt{R} = 0$
25:     end for
26:     $\mathcal{M}.con \leftarrow (\mathtt{s}_{66}^\mathtt{R} + \mathtt{s}_{93}^\mathtt{R} + \mathtt{s}_{162}^\mathtt{R} + \mathtt{s}_{177}^\mathtt{R} + \mathtt{s}_{243}^\mathtt{R} + \mathtt{s}_{288}^\mathtt{R}) = \mathtt{q}$
27:     $\mathcal{M}.con \leftarrow \mathtt{q} = 1$
28:     return $\mathcal{M}$
29: end procedure
```

6 Improved Cube Attacks Against Grain-128AEAD

6.1 Specification of Grain-128AEAD and Its MILP Model

Grain-128AEAD [26] is a member of Grain family and also one of the 2nd-round candidates of the NIST LWC standardization process. Grain-128AEAD inherits many specifications from Grain-128a, which was proposed in 2011 [27]. There are four differences between Grain-128AEAD and Grain-128a: (1) larger MACs, (2) no encryption-only mode, (3) initialization hardening, and (4) keystream limitation. These differences do not come only from the requirement for the NIST LWC standardization process but also from recent cryptanalysis result against Grain-128a [21,22].

The internal state is represented by two 128-bit states, $(b_0, b_1, \ldots, b_{127})$ and $(s_0, s_1, \ldots, s_{127})$. The 128-bit key is loaded to the first register $\boldsymbol{b}$, and the 96-bit initialization vector is loaded to the second register $\boldsymbol{s}$. The other state bits are set to 1 except the least one bit in the second register. Namely, the initial state bits are represented as

$$(b_0, b_1, \dots, b_{127}) = (K_1, K_2, \dots, K_{128}),$$
$$(s_0, s_1, \dots, s_{127}) = (IV_1, IV_2, \dots, IV_{96}, 1, \dots, 1, 0).$$

The pseudo code of the update function in the initialization is given as follows.

$$\begin{aligned} g &\leftarrow b_0 + b_{26} + b_{56} + b_{91} + b_{96} + b_3 b_{67} + b_{11} b_{13} + b_{17} b_{18} + b_{27} b_{59} \\ &\quad + b_{40} b_{48} + b_{61} b_{65} + b_{68} b_{84} + b_{88} b_{92} b_{93} b_{95} + b_{22} b_{24} b_{25} + b_{70} b_{78} b_{82}, \qquad (1) \\ f &\leftarrow s_0 + s_7 + s_{38} + s_{70} + s_{81} + s_{96}, \qquad (2) \\ h &\leftarrow b_{12} s_8 + s_{13} s_{20} + b_{95} s_{42} + s_{60} s_{79} + b_{12} b_{95} s_{94}, \qquad (3) \\ z &\leftarrow h + s_{93} + b_2 + b_{15} + b_{36} + b_{45} + b_{64} + b_{73} + b_{89}, \qquad (4) \\ (b_0, b_1, \dots, b_{127}) &\leftarrow (b_1, \dots, b_{127}, g + s_0 + z), \\ (s_0, s_1, \dots, s_{127}) &\leftarrow (s_1, \dots, s_{127}, f + z). \end{aligned}$$

In the initialization, the state is updated 256 times without producing an output. After the initialization, the update function is tweaked such that z is not fed to the state, and z is used as a pre-output key stream. Hereinafter, we assume that the first bit of the pre-output key stream can be observed. Note that there is no difference between Grain-128a and Grain-128AEAD under this assumption.

MILP Model. `Grain128aEval` in Algorithm 5 generates MILP model $\mathcal{M}$ as the input of Algorithm 1 and 2, and the model $\mathcal{M}$ can evaluate all three-subset division trails for Grain-128AEAD whose initialization rounds are reduced to R. `funcZ` generates MILP variables and constraints for Eq. (3) and Eq. (4), `funcG` generates MILP variables and constraints for Eq. (1), and `funcF` generates MILP variables and constraints for Eq. (2).

6.2 Verification of 184-Round Attack from [12]

In [12], the cube attack against 184-round Grain-128AEAD (Grain-128a) was shown. Here, the following cube indices

$$I = \{1, 2, \dots, 46, 48, 49, \dots, 96\},$$

where $IV[47] = 0$ are used.[5] The conventional bit-based division property with flag technique reveals that the algebraic degree of the corresponding superpoly is at most 14 and the number of monomials is at most $2^{14.61}$. It implies that the corresponding superpoly can be recovered with $2^{95+14.61}$ time complexity.

We run Algorithm 2 with the model generated by Algorithm 5. Surprisingly, the superpoly does not involve the secret key. There are 16,384 three-subset

[5] The first bit of IV is included in the cube index. When the target is Grain-128a, this attack requires queries to both authentication and encryption-only modes. Note that the first bit of IV can also be active in Grain-128AEAD.

Algorithm 5. Model for Grain-128AEAD

```
1: procedure Grain128aEval(round $R$)
2:     Prepare empty MILP Model $\mathcal{M}$
3:     $\mathcal{M}.var \leftarrow \mathtt{b}_\mathtt{i}^0$ for $\mathtt{i} \in \{0, 1, \dots, 127\}$ as binary
4:     $\mathcal{M}.var \leftarrow \mathtt{s}_\mathtt{i}^0$ for $\mathtt{i} \in \{0, 1, \dots, 127\}$ as binary
5:     $\mathcal{M}.con \leftarrow \mathtt{s}_{127}^0 = 0$
6:     for $\mathtt{r} = 1$ to $\mathtt{R}$ do
7:         $(\mathcal{M}, \mathtt{b}_0', \dots, \mathtt{b}_{127}', \mathtt{s}_0', \dots, \mathtt{s}_{127}', \mathtt{z}^\mathtt{r}) = \mathtt{funcZ}(\mathcal{M}, \mathtt{b}_0^{\mathtt{r}-1}, \dots, \mathtt{b}_{127}^{\mathtt{r}-1}, \mathtt{s}_0^{\mathtt{r}-1}, \dots, \mathtt{s}_{127}^{\mathtt{r}-1})$
8:         $\mathcal{M}.var \leftarrow \mathtt{zg}, \mathtt{zf}$ as binary
9:         $\mathcal{M}.con \leftarrow \mathtt{z}^\mathtt{r} = \mathtt{zg} \vee \mathtt{zf}$
10:        $(\mathcal{M}, \mathtt{b}_0'', \dots, \mathtt{b}_{127}'', \mathtt{g}) = \mathtt{funcG}(\mathcal{M}, \mathtt{b}_0', \dots, \mathtt{b}_{127}')$
11:        $(\mathcal{M}, \mathtt{s}_0'', \dots, \mathtt{s}_{127}'', \mathtt{f}) = \mathtt{funcF}(\mathcal{M}, \mathtt{s}_0', \dots, \mathtt{s}_{127}')$
12:        for $i = 0$ to $126$ do
13:            $\mathtt{b}_\mathtt{i}^\mathtt{r} = \mathtt{b}_{\mathtt{i}+1}''$
14:            $\mathtt{s}_\mathtt{i}^\mathtt{r} = \mathtt{s}_{\mathtt{i}+1}''$
15:        end for
16:        $\mathcal{M}.var \leftarrow \mathtt{b}_{127}^\mathtt{r}, \mathtt{s}_{127}^\mathtt{r}$ as binary
17:        $\mathcal{M}.con \leftarrow \mathtt{b}_0'' = 0$
18:        $\mathcal{M}.con \leftarrow \mathtt{b}_{127}^\mathtt{r} = \mathtt{g} + \mathtt{s}_0'' + \mathtt{zg}$
19:        $\mathcal{M}.con \leftarrow \mathtt{s}_{127}^\mathtt{r} = \mathtt{f} + \mathtt{zf}$
20:    end for
21:    $(\mathcal{M}, \mathtt{b}_0', \dots, \mathtt{b}_{127}', \mathtt{s}_0', \dots, \mathtt{s}_{127}', \mathtt{z}) = \mathtt{funcZ}(\mathcal{M}, \mathtt{b}_0^\mathtt{R}, \dots, \mathtt{b}_{127}^\mathtt{R}, \mathtt{s}_0^\mathtt{R}, \dots, \mathtt{s}_{127}^\mathtt{R})$
22:    for all $\mathtt{i} \in \{0, 1, \dots, 127\}$ do
23:        $\mathcal{M}.con \leftarrow \mathtt{b}_\mathtt{i}' = 0$
24:        $\mathcal{M}.con \leftarrow \mathtt{s}_\mathtt{i}' = 0$
25:    end for
26:    $\mathcal{M}.con \leftarrow \mathtt{z} = 1$
27:    return $\mathcal{M}$
28: end procedure
```

division trails, but only three initial properties can be feasible (see Table 4, where $\boldsymbol{x} = (x_1, x_2, \dots, x_{128})$ denotes the secret key). Moreover, all of them have even-number of trails, i.e., the superpoly shown in [12] is constant-0. Therefore, the cube attack against 184-round Grain-128AEAD is a zero-sum distinguisher.

6.3 Additional Constraints and Superpoly for 190 Rounds

Algorithm 5 evaluates `funcZ`, `funcG`, and `funcF` independently, and combines them. While this algorithm can enumerate all three-subset division trails, it includes many redundant trails. For example, let us consider that there are two propagations for one round from the fixed bitvector to fixed one. Then, considering such propagations is redundant because the number of three-subset division trails including such propagations in its inside is always even number. Therefore, we should remove such propagations from the model in advance to reduce the number of feasible three-subset division trails. We carefully checked three-subset division trails found in the attack against 184-round Grain-128AEAD. As a result, we find a frequently used (but redundant) propagation.

Table 4. Detailed results for superpoly against 184-round Grain-128AEAD.

Parity	# Trails	Monomial
0	4096	$x_{34}x_{39}x_{53}x_{62}x_{64}x_{81}x_{83}x_{84}x_{95}x_{125}$
0	4096	$x_{34}x_{39}x_{49}x_{53}x_{62}x_{64}x_{81}x_{83}x_{84}x_{95}x_{123}x_{127}x_{128}$
0	8192	$x_{23}x_{39}x_{48}x_{49}x_{53}x_{58}x_{59}x_{62}x_{64}x_{83}x_{84}x_{98}x_{118}x_{120}$

Property 1. In any round r, either $\mathtt{s}_0^\mathtt{r}$ or $\mathtt{z}^\mathtt{r}$ must be 0.

Proof. In round r, we assume that $\mathtt{s}_0^\mathtt{r} = 1$ and $\mathtt{z}^\mathtt{r} = 1$. The keystream bit ($\mathtt{z}^\mathtt{r} = 1$) can propagate to the rightmost bit of NFSR ($\mathtt{b}_{127}^{\mathtt{r+1}}$) and the rightmost bit of LFSR ($\mathtt{s}_{127}^{\mathtt{r+1}}$). The leftmost bit of the LFSR ($\mathtt{s}_0^\mathtt{r}$) can also propagate to the same two bits. Therefore, unless either of s_{127}^{r+1}, b_{127}^{r+1}, or $s_{127}^{r+1} \cdot b_{127}^{r+1}$ has monomial $s_0^r \cdot z^r$, such a propagation is infeasible. Clearly, s_{127}^{r+1} and b_{127}^{r+1} do not have such a monomial. Moreover, the monomial $s_0^r \cdot z^r$ is always canceled out in

$$\begin{aligned} s_{127}^{r+1} \cdot b_{127}^{r+1} &= (f^r + z^r) \cdot (g^r + z^r + s_0^r) \\ &= f^r \cdot g^r + f^r \cdot s_0^r + (f^r + g^r + 1 + s_0^r) \cdot z^r \\ &= f^r \cdot g^r + f^r \cdot s_0^r + (s_7^r + s_{38}^r + s_{70}^r + s_{81}^r + s_{96}^r + g^r + 1) \cdot z^r. \end{aligned}$$

$\square$

Property 1 is very simple and powerful. We just add the following constraint

$$\mathcal{M}.con \leftarrow \mathtt{s}_0^\mathtt{r} + \mathtt{z}^\mathtt{r} \leq 1$$

between the line 6 and 7 in Algorithm 5. We re-run Algorithm 2 by using the model generated by Algorithm 5 with the modification above. Then, 16, 384 trails become impossible, and there is no feasible solution.

Superpoly from 185 to 189 rounds. We showed that the 184-round attack is a zero-sum distinguisher and cannot recover any secret key bit. Similarly to the case of Trivium, we expect that the number of rounds that we can attack can be improved. To attack more rounds, we use cube indices $I = \{1, 2, \ldots, 96\}$, where all IV bits are active. As a result, there is no feasible solution up to 189 rounds. In other words, we find zero-sum distinguishers from 185 to 189 rounds.

Superpoly for 190 rounds. From 190 rounds onwards, secret key bits can be involved. As a result, 7, 621 feasible three-subset division trails are enumerated, and $J[\boldsymbol{u}]$ in Algorithm 2 is non zero for 3, 006 different $\boldsymbol{u}$'s. Out of 3, 006 $\boldsymbol{u}$'s, there are 1, 097 $\boldsymbol{u}$'s whose $J[\boldsymbol{u}]$ is an odd number. In other words, the superpoly is represented as the sum of 1, 097 monomials. Interestingly, the recovered superpoly has completely different features of the one of Trivium. While the superpoly of Trivium is a very low-degree and simple Boolean function, the recovered superpoly for Grain128-AEAD has algebraic degree 21 and is a complicated Boolean function with no monomials of degree lower than 6. Since the Boolean function is too complicated to evaluate its weight theoretically, we experimentally evaluated the balancedness. We picked 2^{15} secret keys randomly and compute the output of the Boolean function. As a result, it is highly biased, and the fraction of keys that output 1 is about 0.032. Therefore, the information recovered from this superpoly is very small. Indeed, if the superpoly in the online phase evaluates to one, we gain almost 5 bit (i.e. $-\log_2(0.032)$) in an attack when filtering wrong keys. However, in the case where the superpoly evaluates to zero, we gain less than 0.04 bits (i.e. $-\log_2(1-0.032)$) in an attack. The average gain, given by the entropy, is only

$$-0.032\log_2(0.032) - (1-0.032)\log_2(1-0.032) \approx 0.2$$

which limits the interest in this approach.

6.4 Towards Efficient Key-Recovery Attacks

To recover more bits of information, we use multiple cubes whose size decreases from 96 to 95. However, if the cube index misses one IV bit, the number of three-subset division trails increases. We need to pick appropriate non-cube indices, where the number of three-subset division trails does not expand to much. We were able to compute the representation of 15 superpolys p_j where the cube index set was $\{1..96\} \setminus j$ with

$$j \in J = \{27, 30, 31, 32, 34, 41, 44, 45, 46, 48, 58, 59, 64, 70, 72\}.$$

Those polynomials vary significantly in size (between 176 and 19,925 monomials) but also share interesting properties. Again, due to their size, some of the properties can only be estimated experimentally.

Interestingly, all polynomials are highly biased toward zero and none of the polynomials involves all key bits. In particular none of the polynomials depends on the key bits

$$K_1, K_2, K_3, K_6 \text{ and } K_9.$$

Moreover, all polynomials can be evaluated rather efficiently on average. The details are given in Table 5. Note that the average total cost of evaluating the polynomials is an upper bound on the number of XORs and ANDs needed. This bound was derived using a time-memory tradeoff for the evaluation process, by fixing 14 key bits that appear frequently in all 15 polynomials. Fixing to all 2^{14} possible values resulted in $15 \cdot 2^{14}$ polynomials. Those polynomials are significantly simpler and simply counting the number of required AND and XOR operations in a trivial evaluation process resulted in the numbers in Table 5 that are sufficient for our attack. In particular, the average cost of evaluating all 15 polynomials together is smaller than 2^{12}, which is smaller than producing a single key stream bit with Grain128-AEAD reduced to 190 rounds.

Besides being highly unbalanced, the polynomials are also not independent when evaluated on random keys. In order to estimate how many wrong keys are

Table 5. Properties of the superpolys for Grain128-AEAD.

Poly	p_{27}	p_{30}	p_{31}	p_{32}	p_{34}	p_{41}	p_{44}	p_{45}	p_{46}	p_{48}	p_{58}	p_{59}	p_{64}	p_{70}	p_{72}
Nb. of ind. K_i	7	6	12	8	6	13	14	47	6	16	6	10	12	11	8
$\Pr(p_j = 0)$	0.077	0.116	0.055	0.089	0.090	0.099	0.019	0.012	0.081	0.055	0.123	0.196	0.097	0.156	0.083
Av. cost	544	408	107	196	452	148	19	10	199	213	406	497	432	336	205

filtered on average, we estimated the entropy of $(p_{27}, \ldots, p_{72})$ when evaluated at uniformly random chosen keys. That is, for $v_j \in \{0, 1\}$ we estimated

$$\Pr((P_{27}, \ldots, P_{72}) = (v_{27}, \ldots, v_{72}))$$

for all 2^{15} possible outcomes. The distribution is still highly biased, in particular $\Pr(0, \ldots, 0) \approx 0.57$. However, the entropy, which was estimated using 2^{25} samples, increased to 5.03 which now makes the following attack possible.

1. The attacker evaluates in the online phase the values of the 15 superpolys for the given secret key.
2. The attacker guesses all key-bits except the bits K_1, K_2, K_3, K_6, K_9 and for each guess filters with the correct values of the superpolys given from the online phase.
3. For each guess that passes the filtering, the attacker runs through all possible values of K_1, K_2, K_3, K_6, K_9 and verifies the key against given key-stream.

The cost of the online phase is 15×2^{95} time and 2^{96} data, i.e. using all possible IV values for the given secret key.

In the second step, the number of guesses is 2^{128-5} and, due to the entropy, the average amount of not filtered guesses is $2^{128-5-5.03}$. As evaluating the polynomials is cheaper than evaluating Grain128-AEAD, the cost for this step is less than 2^{123} evaluations of Grain128-AEAD.

In the third step, the average cost is $2^5 \cdot 2^{128-5-5.03}$, i.e. less than 2^{123} evaluations of Grain128-AEAD as well. To conclude, the attack has an average time complexity of less than 2^{123} evaluations of Grain128-AEAD and a data complexity of 2^{96}. Note that this complexity is averaged over the given secret key. In particular, after the first step of the attack, the attacker already knows how efficient filtering will be in her particular case. For some keys filtering is significantly stronger. This observation might be further elaborated into a stronger attack for a smaller fraction of keys, i.e. a weak-key attack.

7 Conclusion

In this paper, we proposed a new modeling technique for the three-subset division property without unknown subset. Our technique is significant for the application to the cube attack. Unlike the previous experimental or theoretical cube attacks, our method does not need any assumption and can recover the actual superpoly in practical time. Our method leads to the best key-recovery attack on two of the most important stream ciphers.

Acknowledgement. The authors thank the anonymous Eurocrypt 2020 reviewers for careful reading and many helpful comments. Yonglin Hao is supported by National Key Research and Development Program of China (No. 2018YFA0306404). Gregor Leander is supported by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy - EXC 2092 CASA - 390781972. Qingju Wang is funded by the University of Luxembourg Internal Research Project (IRP) FDISC.

References

1. 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
2. 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
3. Lai, X.: Higher order derivatives and differential cryptanalysis. In: Blahut, R.E., Costello, D.J., Maurer, U., Mittelholzer, T. (eds.) Communications and Cryptography. SECS, vol. 276, pp. 227–233. Springer, Boston (1994). https://doi.org/10.1007/978-1-4615-2694-0_23
4. Todo, Y.: Structural evaluation by generalized integral property. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015, Part I. LNCS, vol. 9056, pp. 287–314. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46800-5_12
5. Todo, Y.: Integral cryptanalysis on full MISTY1. In: Gennaro, R., Robshaw, M. (eds.) CRYPTO 2015, Part I. LNCS, vol. 9215, pp. 413–432. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-47989-6_20
6. Sasaki, Y., Todo, Y.: New differential bounds and division property of Lilliput: block cipher with extended generalized Feistel network. In: Avanzi, R., Heys, H. (eds.) SAC 2016. LNCS, vol. 10532, pp. 264–283. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-69453-5_15
7. 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
8. Sugio, N., Igarashi, Y., Kaneko, T., Higuchi, K.: New integral characteristics of KASUMI derived by division property. In: Choi, D., Guilley, S. (eds.) WISA 2016. LNCS, vol. 10144, pp. 267–279. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-56549-1_23
9. 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, Part I. LNCS, vol. 10031, pp. 648–678. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53887-6_24
10. 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, Part I. LNCS, vol. 10624, pp. 128–157. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-70694-8_5
11. 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, Part III. LNCS, vol. 10403, pp. 250–279. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63697-9_9

12. 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, Part I. LNCS, vol. 10991, pp. 275–305. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-96884-1_10
13. Bernstein, D.J., et al.: Gimli: a cross-platform permutation. In: Fischer, W., Homma, N. (eds.) CHES 2017. LNCS, vol. 10529, pp. 299–320. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-66787-4_15
14. Banik, S., Pandey, S.K., Peyrin, T., Sasaki, Y., Sim, S.M., Todo, Y.: GIFT: a small present - towards reaching the limit of lightweight encryption. In: Fischer, W., Homma, N. (eds.) CHES 2017. LNCS, vol. 10529, pp. 321–345. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-66787-4_16
15. Wang, Q., Liu, Z., Varıcı, K., Sasaki, Y., Rijmen, V., Todo, Y.: Cryptanalysis of reduced-round SIMON32 and SIMON48. In: Meier, W., Mukhopadhyay, D. (eds.) INDOCRYPT 2014. LNCS, vol. 8885, pp. 143–160. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-13039-2_9
16. 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
17. 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, Part III. LNCS, vol. 11923, pp. 398–427. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-34618-8_14
18. 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
19. Ye, C.D., Tian, T.: Revisit division property based cube attacks: key-recovery or distinguishing attacks? IACR Trans. Symm. Cryptol. **2019**(3), 81–102 (2019)
20. Fu, X., Wang, X., Dong, X., Meier, W.: A key-recovery attack on 855-round Trivium. In: Shacham, H., Boldyreva, A. (eds.) CRYPTO 2018, Part II. LNCS, vol. 10992, pp. 160–184. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-96881-0_6
21. Hamann, M., Krause, M.: On stream ciphers with provable beyond-the-birthday-bound security against time-memory-data tradeoff attacks. Cryptogr. Commun. **10**(5), 959–1012 (2018). https://doi.org/10.1007/s12095-018-0294-5
22. Todo, Y., Isobe, T., Meier, W., Aoki, K., Zhang, B.: Fast correlation attack revisited - cryptanalysis on full Grain-128a, Grain-128, and Grain-v1. In: Shacham, H., Boldyreva, A. (eds.) CRYPTO 2018, Part II. LNCS, vol. 10992, pp. 129–159. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-96881-0_5
23. Cannière, C.D., Preneel, B.: Trivium specifications. eSTREAM portfolio, Profile 2 (HW) (2006)
24. Hao, Y., Jiao, L., Li, C., Meier, W., Todo, Y., Wang, Q.: Observations on the dynamic cube attack of 855-round TRIVIUM from Crypto '18. Cryptology ePrint Archive, Report 2018/972 (2018). https://eprint.iacr.org/2018/972
25. Fu, X., Wang, X., Dong, X., Meier, W., Hao, Y., Zhao, B.: A refinement of "a key-recovery attack on 855-round Trivium" from crypto 2018. Cryptology ePrint Archive, Report 2018/999 (2018). https://eprint.iacr.org/2018/999
26. Hell, M., Johansson, T., Meier, W., Sönnerup, J., Yoshida, H.: Grain-128AEAD: a lightweight AEAD stream cipher. Lightweight Cryptography (LWC) Standardization (2019)
27. Ågren, M., Hell, M., Johansson, T., Meier, W.: Grain-128a: a new version of Grain-128 with optional authentication. IJWMC **5**(1), 48–59 (2011)