Improved Division Property Based Cube Attacks Exploiting Algebraic Properties of Superpoly

Yonglin Hao, Takanori Isobe, Lin Jiao, Chaoyun Li, Willi Meier, Yosuke Todo and Qingju Wang.

Abstract—At CRYPTO 2017 and IEEE Transactions on Computers in 2018, Todo et al. proposed the division property based cube attack method making it possible to launch cube attacks with cubes of dimensions far beyond practical reach. However, assumptions are made to validate their attacks. In this paper, we further formulate the algebraic properties of the superpoly in one framework to facilitate cube attacks in more successful applications: we propose the "flag" technique to enhance the precision of MILP models, which enable us to identify proper non-cube IV assignments; a degree evaluation algorithm is presented to upper bound the degree of the superpoly s.t. the superpoly can be recovered without constructing its whole truth table and overall complexity of the attack can be largely reduced; we provide a divide-and-conquer strategy to TRIVIUM-like stream ciphers namely TRIVIUM, Kreyvium, TriviA-SC1/2 so that the large scale MILP models can be split into several small solvable ones enabling us to analyze TRIVIUM-like primitives with more than 1000 initialization rounds; finally, we provide a term enumeration algorithm for finding the monomials of the superpoly, so that the complexity of many attacks can be further reduced.
We apply our techniques to attack the initialization of several ciphers namely 839-round TRIVIUM, 891-round Kreyvium, 1009-round TriviA-SC1, 1004-round TriviA-SC2, 184-round Grain-128a and 750-round ACORN respectively.

Index Terms—Cube Attack, Division Property, MILP, TRIVIUM, Kreyvium, Grain-128a, ACORN, TriviA-SC1/2

1 Introduction

The cube attack, proposed by Dinur and Shamir [2] in 2009, is one of the general cryptanalytic techniques of analyzing symmetric-key cryptosystems. After its proposal, cube attack has been successfully applied to various cryptographic primitives, including stream ciphers [3], [4], [5], [6], [7], [8], [9], hash functions [10], [11], [12], and authenticated encryption [13], [14]. For a cipher with n secret variables $\vec{x} = (x_1, x_2, \ldots, x_n)$ and m public variables $\vec{v} = (v_1, v_2, \ldots, v_m)$, we can regard the algebraic normal form (ANF) of output bits as a polynomial of $\vec{x}$ and $\vec{v}$, denoted as $f(\vec{x}, \vec{v})$. For a randomly chosen set $I = \{i_1, i_2, ..., i_{|I|}\} \subset \{1, \ldots, m\}$, $f(\vec{x}, \vec{v})$ can be represented uniquely as

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

where $t_I = v_{i_1} \cdots v_{i_{|I|}}$, $p(\vec{x}, \vec{v})$, referred as the *superpoly* [2], only relates to v_s's ($s \notin I$) and the secret key bits $\vec{x}$, and $q(\vec{x}, \vec{v})$ misses at least one variable in t_I. When v_s's ($s \notin I$) and $\vec{x}$ are assigned statically, the value of $p(\vec{x}, \vec{v})$ can be computed by summing the output bit $f(\vec{x}, \vec{v})$ over a structure called *cube*, denoted as C_I, consisting of $2^{|I|}$ different $\vec{v}$ vectors with $v_i, i \in I$ being active (traversing all 0-1 combinations) and non-cube indices $v_s, s \notin I$ being static constants.

Traditional cube attacks are completely based on practical experiments. They can only utilize practically computable low-dimensional cubes with linear or quadratic superpolies. Breakthroughs have been made by Todo et al. in [15] where they introduce the bit-based division property, a tool for conducting integral attacks to the realm of cube attack. With the help of mixed integer linear programming (MILP), they can identify the variables excluded from the superpoly and explore cubes with larger size, e.g.,72 for 832-round TRIVIUM. This enables them to improve the traditional cube attack.

Division property, as a generalization of the integral property, was first proposed at EUROCRYPT 2015 [16]. With division property, the propagation of the integral characteristics can be deduced in a more accurate manner, and one prominent application is the first theoretic key recovery attack on full MISTY1 [17]. The original division property can only be applied to word-oriented primitives. At FSE 2016, the bit-based division property [18] was proposed to investigate integral characteristics for bit-based block ciphers. With the help of division property, the propagation of the integral characteristics can be represented by the operations on a set of 0-1 vectors identifying the bit positions with the zero-sum property. But, as has been pointed out by the authors of [18], the deduction of bit-based division property under their framework requires memory exponential to the block sizes of block ciphers, which largely limits its applications. Such a problem has been solved by Xiang et al. [19] at ASIACRYPT 2016 by utilizing the MILP model. The propagation of the division property can be translated to an MILP model, and the corresponding integral characteristics are acquired by

- *Yonglin Hao and Lin Jiao are with State Key Laboratory of Cryptology, Beijing, China. Lin Jiao is also with Gudangdong Provincial Key Laboratory of Data Security and Privacy Protection, Guangzhou 510632, China. E-mail: haoyonglin@yeah.net, jiaolin_jl@126.com.*
- *Takanori Isobe is with Applied informatics, University of Hyogo, Kobe, Hyogo, Japan. E-mail: takanori.isobe@ai.u-hyogo.ac.jp.*
- *Chaoyun Li is with imec-COSIC, Dept. Electrical Engineering (ESAT), KU Leuven, Leuven, Belgium. E-mail: chaoyun.li@esat.kuleuven.be.*
- *Willi Meier is with FHNW, Institute ISE, Windisch, Aargau Switzerland. E-mail: willi.meier@fhnw.ch.*
- *Yosuke Todo is with NTT Secure Platform Laboratories, Musashino, Tokyo Japan. E-mail: todo.yosuke@lab.ntt.co.jp.*
- *Qingju Wang is with SnT, University of Luxembourg, Esch-sur-Alzette, Luxembourg. E-mail: qingju.wang@uni.lu*

Manuscript received Month day, 2018; revised Month day, year.
This is the extended version of our CRYPTO 2018 paper [1].

solving the models with MILP solvers like Gurobi [20]. With this method, they are able to give integral characteristics for block ciphers with large block sizes. Xiang et al.'s method has now been applied to many other ciphers for improved integral attacks [21], [22], [23], [24].

In [15], Todo et al. adapt Xiang et al.'s method by taking key bits into the MILP model. With this technique, a set of key indices $J = \{j_1, j_2, \ldots, j_{|J|}\} \subset \{1, \ldots, n\}$ is deduced for the cube C_I s.t. $p(\vec{x}, \vec{v})$ can only be related to the key bits x_j's ($j \in J$). With the knowledge of I and J, Todo et al. can recover 1-bit of secret-key-related information by executing two phases. In the *offline phase*, a proper assignment to the non-cube IVs, denoted by $\vec{IV} \in \mathbb{F}_2^m$, is determined ensuring $p(\vec{x}, \vec{IV})$ non-constant. Also in this phase, the whole truth table of $p(\vec{x}, \vec{IV})$ is constructed through cube summations with a complexity $2^{|I|+|J|}$. In the *online phase*, the exact value of $p(\vec{x}, \vec{IV})$ is acquired through a cube summation and the candidate values of x_j's ($j \in J$) are identified by checking the precomputed truth table. A proportion of wrong key guesses are filtered as long as $p(\vec{x}, \vec{IV})$ is non-constant.

Due to division property and the power of MILP solver, cubes of larger dimension can now be used for key recoveries. By using a 72-dimensional cube, Todo et al. propose a theoretic cube attack on 832-round TRIVIUM. They also largely improve the previous best attacks on other primitives namely ACORN, Grain-128a and Kreyvium [15], [25] (also available online at [26]). It is not until recently that the result on TRIVIUM has been improved by Liu et al. [7] mounting to 835 rounds with a new method called the *correlation* cube attack. The correlation attack is based on the *numeric mapping* technique first appeared in [27] originally used for constructing zero-sum distinguishers.

1.1 Our Contributions.

This paper improves the existing cube attacks by exploiting the algebraic properties of the superpoly, which include the (non-)constantness, low degree and sparse monomial distribution properties. Inspired by the division property based cube attack work of Todo et al. in [15], we formulate all these properties in one framework by developing more precise MILP models, thus we can reduce the complexity of superpoly recovery. This also enables us to attack more rounds, or employ even larger cubes. Similar to [15], our methods regard the cryptosystem as a non-blackbox polynomial and can be used to evaluate cubes with large dimension compared with traditional cube attack and cube tester. In the following, our contributions are summarized into five aspects.

Flag technique for finding proper IV assignments. The superpoly can filter wrong key guesses only if a proper assignment $\vec{IV} \in \mathbb{F}_2^m$ is found for the non-cube IVs s.t. $p(\vec{x}, \vec{IV})$ is non-constant. In [15], such proper $\vec{IV}$'s can only be found by random trials in the offline phase. Since each trial requires a complexity of $2^{|I|+|J|}$, there is high risk that the complexity of a proper $\vec{IV}$ finding process may exceed the brute-force bound 2^n when large cubes are used ($|I|$ is big) or many key bits are involved ($|J|$ is large). Therefore, two following assumptions are made to validate their cube attacks:

Assumption 1 (Strong Assumption). For a cube C_I, there are many values in the constant part of IV whose corresponding superpoly is balanced.

Assumption 2 (Weak Assumption). For a cube C_I, there are many values in the constant part of IV whose corresponding superpoly is not a constant function.

It has been noticed in [25] that constant 0 bits can affect the propagation of division property. But we should pay more attention to constant 1 bits since constant 0 bits can be generated in the updating functions due to the XOR of an even number of constant 1's. Therefore, we propose a formal technique which we refer as the "flag" technique where the constant 0 and constant 1 as well as other non-constant MILP variables are treated properly. With this technique, proper IVs can now be found with MILP model rather than time-consuming trial and error summations in the offline phase as in [15], [25]. According to our experiments, the flag technique has a perfect 100% accuracy for finding proper non-cube IV assignments in most cases. Note that our flag technique has partially proved the validity of the two assumptions since we are able to find proper $\vec{IV}$'s in all our attacks.

Degree evaluation for going beyond the $|I| + |J| < n$ restriction. The superpoly recovery process dominates the overall complexity of the division property based cube attack in [15]. It requires to construct the whole truth table of the superpoly through cube summations so the complexity reaches $2^{|I|+|J|}$. Clearly, such an attack can only be meaningful if $|I|+|J| < n$, where n is the number of secret variables. To avoid constructing the whole truth table, we introduce a new technique that can upper bound the algebraic degree, denoted as d, of the superpoly using the MILP-aided bit-based division property. With the knowledge of its degree d (and key indices J), the superpoly can be represented with its $\binom{|J|}{\le d}$ coefficients rather than the whole truth table, where $\binom{|J|}{\le d}$ is defined as $\binom{|J|}{\le d} := \sum_{i=0}^{d} \binom{|J|}{i}$. When $d = |J|$, the complexity by our new method and that by [15] are equal. For $d < |J|$, we know that the coefficients of the monomials with degree higher than d are constantly 0. The complexity of superpoly recovery can be reduced from $2^{|I|+|J|}$ to $2^{|I|} \times \binom{|J|}{\le d}$. In fact, for some lightweight ciphers, the algebraic degrees of their round functions are quite low. Therefore, the degrees d are often much smaller than the number of involved key bits $|J|$, especially when high-dimensional cubes are used. Since $d \ll |J|$ for all previous attacks, we can improve the complexities of previous results and use larger cubes mounting to more rounds even if $|I| + |J| \ge n$.

Divide-and-Conquer strategy for modeling TRIVIUM-Like primitives. TRIVIUM-like stream ciphers, namely TRIVIUM, Kreyvium, TriviA-SC1/2, have simple updating and output functions but with sufficiently many initialization rounds. The sizes of MILP models expand significantly when we describe the division property propagation of too many initialization rounds. The sizes of MILP can be so large that the MILP solvers cannot handle. We cultivate the structural features of TRIVIUM-like primitives and introduce the divide-and-conquer strategy. A huge MILP can now be split into several small scaled models that can be solved efficiently. Combining the solutions of small models enable us to extract the information we need to launch key-recovery attacks.

Precise Term enumeration for further lowering complexities. Since the superpolies are generated through iterations, the number of higher-degree monomials in the superpoly is usually much smaller than its low-degree counterpart. For example, when the degree of the superpoly is $d < |J|$, the number of d-degree monomials are usually much smaller than the upper bound $\binom{|J|}{d}$. We propose a MILP model technique for enumerating all t-degree ($t = 1, \ldots, d$) monomials that may appear in the superpoly, so that the complexities of several attacks are further reduced.

Relaxed Term enumeration. For some primitives (such as 750-round ACORN), our MILP model can only enumerate the d-degree monomials since the number of lower-degree monomials are too large to be exhausted. For $t = 1, \ldots, d-1$, we can find a set of key indices $JR_t \subseteq J$ s.t. all t-degree monomials in the superpoly are composed of x_j, $j \in JR_t$. As long as $|JR_t| < |J|$ for some $t = 1, \ldots, d-1$, we can still reduce the complexities of superpoly recovery.

Combining the flag technique and the degree evaluation above, we are able to lower the complexities of the previous best cube attacks in [7], [15], [25]. Particularly, we can further provide key recovery results on 839-round TRIVIUM[2], 891-round Kreyvium, 184-round Grain-128a, and 750-round ACORN. Furthermore, the precise & relaxed term enumeration techniques allow us to lower the complexities of 833-round TRIVIUM, 849-round Kreyvium, 184-round Grain-128a and 750-round ACORN. Our concrete results are summarized in Table 1. In [31], Todo et al. revisit the fast correlation attack and analyze the key-stream generator (rather than the initialization) of the Grain family (Grain-128a, Grain-128, and Grain-v1). As a result, the key-stream generators of the Grain family are insecure. In other words, they can recover the internal state after initialization more efficiently than by exhaustive search. And the secret key is recovered from the internal state because the initialization is a public permutation. To the best of our knowledge, all our results of Kreyvium, TriviA-SC1/2, Grain-128a, and ACORN are the current best key recovery attacks on the initialization of the targeted ciphers. However, none of our results seems to threaten the security of the ciphers.

In comparison to the CRYPTO 2018 conference version [1], the following contents are new:

- Further improvements to our Flag Techniques are implemented.
- A Divide-and-Conquer strategy is applied to model the division property propagation for TRIVIUM-like primitives.
- We mounts to the best key-recovery attacks on TriviA-SC.

1.2 Organization.

Sect. 2 provides the background of cube attacks, division property, MILP model etc. Sect. 3 introduces our flag technique for identifying proper assignments to non-cube IVs. Sect. 4 details the degree evaluation technique upper bounding the algebraic degree of the superpoly. With the flag technique, the degree evaluation and the divide-and-conquer strategy, we give improved key recovery cube attacks on 4 TRIVIUM-like stream ciphers Sect. 5. We then apply the flag technique and the degree evaluation to Grain-128a and ACORN in Sect. 6. The precise & relaxed term enumeration as well as their applications are given in Sect. 7. Finally, we conclude in Sect. 8.

2. Fu et al. also propose a result on 855-round TRIVIUM in [28] but its practical examples are wrong and its theoretic basis has been severely challenged in [29]. Such mistakes are also admitted by Fu et al. in [30].

TABLE 1
Summary of our cube attack results

Cipher	#Full Rds	#Rds	Cube size	$\|J\|$	Complexity	Reference
TRIVIUM	1152	799	32 †	-	practical	[5]
		832	72	5	2^{77}	[15], [25]
		833	**73**	**7**	$\mathbf{2^{79}}$	**Sect. 5.1**
		833	**73**	**7**	$\mathbf{2^{76.91}}$	**Sect. 7.1**
		835	37/36$*$	-	2^{75}	[7]
		839	**78**	**1**	$\mathbf{2^{79}}$	**Sect. 5.1**
Kreyvium	1152	849	61	23	2^{84}	[25]
		849	**61**	**23**	$\mathbf{2^{81.7}}$	**Sect. 5.2**
		849	**61**	**23**	$\mathbf{2^{73.41}}$	**Sect. 7.1**
		872	85	39	2^{124}	[25]
		872	**85**	**39**	$\mathbf{2^{94.61}}$	**Sect. 5.2**
		891	**113**	**20**	$\mathbf{2^{120.73}}$	**Sect. 5.2**
TriviA-SC1	1152	**1009**	**110**	**1**	$\mathbf{2^{111}}$	**Sect. 5.3**
TriviA-SC2	1152	**1004**	**110**	**2**	$\mathbf{2^{111.6}}$	**Sect. 5.3**
Grain-128a	256	177	33	-	practical	[32]
		182	88	18	2^{106}	[15], [25]
		182	**88**	**14**	$\mathbf{2^{102}}$	**Sect. 6.1**
		183	92	16	2^{108}	[15], [25]
		183	**92**	**16**	$\mathbf{2^{108} - 2^{96.08}}$	**Sect. 6.1**
		184	**95**	**21**	$\mathbf{2^{115.95}}$	**Sect. 6.1**
		184	**95**	**21**	$\mathbf{2^{109.61}}$	**Sect. 7.1**
ACORN	1792	503	5 ‡	-	practical ‡	[6]
		704	64	58	2^{122}	[15], [25]
		704	**64**	**63**	$\mathbf{2^{93.23}}$	**Sect. 6.2**
		704	**64**	**63**	$\mathbf{2^{77.88}}$	**Sect. 7.1**
		750	**101**	**81**	$\mathbf{2^{125.71}}$	**Sect. 6.2**
		750	**101**	**81**	$\mathbf{2^{120.92}}$	**Sect. 7.2**

† 18 cubes whose size is from 32 to 37 are used, where the most efficient cube is shown to recover one bit of the secret key.
$*$ 28 cubes of sizes 36 and 37 are used, following the correlation cube attack scenario. It requires an additional 2^{51} complexity for preprocessing.
‡ The attack against 477 rounds is mainly described for the practical attack in [6]. However, when the goal is the superpoly recovery and to recover one bit of the secret key, 503 rounds are attacked.

2 PRELIMINARIES

2.1 Mixed Integer Linear Programming

MILP is an optimization or feasibility program some of whose variables are restricted to integers. A MILP model $\mathcal{M}$ consists of variables $\mathcal{M}.var$, constraints $\mathcal{M}.con$, and an objective function $\mathcal{M}.obj$. MILP models can be solved by solvers like Gurobi [20]. If there is no feasible solution at all, the solver simply returns *infeasible*. If no objective function is assigned, the MILP solver only evaluates the feasibility of the model. The application of MILP model to cryptanalysis dates back to the year 2011 [33], and has been widely used for searching characteristics corresponding to various methods such as differential [34], [35], linear [35], impossible differential [36], [37], zero-correlation linear [36], and integral characteristics with division property [19]. We will detail the MILP model of [19] later in this section.

2.2 Cube Attack

Considering a stream cipher with n secret key bits $\vec{x} = (x_1, x_2, \ldots, x_n)$ and m public initialization vector (IV) bits $\vec{v} = (v_1, v_2, \ldots, v_m)$. Then, the first output keystream bit can be regarded as a polynomial of $\vec{x}$ and $\vec{v}$ referred as $f(\vec{x}, \vec{v})$. For a set of indices $I = \{i_1, i_2, \ldots, i_{|I|}\} \subset \{1, 2, \ldots, n\}$, which is referred as cube indices and denote by t_I the monomial as $t_I = v_{i_1} \cdots v_{i_{|I|}}$, the algebraic normal form (ANF) of $f(\vec{x}, \vec{v})$ can be uniquely decomposed as

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

where the monomials of $q(\vec{x}, \vec{v})$ miss at least one variable from $\{v_{i_1}, v_{i_2}, \ldots, v_{i_{|I|}}\}$. Furthermore, $p(\vec{x}, \vec{v})$, referred as the superpoly in [2], is independent of $\{v_{i_1}, v_{i_2}, \ldots, v_{i_{|I|}}\}$. The value of $p(\vec{x}, \vec{v})$ can only be affected by the secret key bits $\vec{x}$ and the assignment to the non-cube IV bits v_s ($s \notin I$). For a secret key $\vec{x}$ and an assignment to the non-cube IVs $\vec{IV} \in \mathbb{F}_2^m$, we can define a structure called cube, denoted as $C_I(\vec{IV})$, consisting of $2^{|I|}$ 0-1 vectors as follows:

$$C_I(\vec{IV}) := \{\vec{v} \in \mathbb{F}_2^m : \vec{v}[i] = 0/1, i \in I \bigwedge \vec{v}[s] = \vec{IV}[s], s \notin I\}. \quad (1)$$

It has been proved by Dinur and Shamir [2] that the value of superpoly p corresponding to the key $\vec{x}$ and the non-cube IV assignment $\vec{IV}$ can be computed by summing over the cube $C_I(\vec{IV})$ as follows:

$$p(\vec{x}, \vec{IV}) = \bigoplus_{\vec{v} \in C_I(\vec{IV})} f(\vec{x}, \vec{v}). \quad (2)$$

In the remainder of this paper, we refer to the value of the superpoly corresponding to the assignment $\vec{IV}$ in Eq. (2) as $p_{\vec{IV}}(\vec{x})$ for short. We use C_I as the cube corresponding to arbitrary $\vec{IV}$ setting in Eq. (1). Since C_I is defined according to I, we may also refer I as the "cube" without causing ambiguities. The size of I, denoted as $|I|$, is also referred as the dimension of the cube.

Note: since the superpoly p is independent of cube IVs $v_i, i \in I$, the value of $\vec{IV}[i], i \in I$ cannot affect the result of the summation in Eq. (2) at all. Therefore in Sect. 6, our $\vec{IV}[i]$'s ($i \in I$) are just assigned randomly to 0-1 values.

2.3 Bit-Based Division Property and Its MILP Representation

At EUROCRYPT 2015, the division property, a generalization of the integral property, was proposed in [16] resulting in better integral characteristics for word-oriented primitives. Later, the bit-based division property was introduced in [18] so that the propagation of integral characteristics can be described in a more precise manner. The definition of the bit-based division property is as follows:

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

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

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

When the basic bitwise operations COPY, XOR, AND are applied to the elements in $\mathbb{X}$, transformations of the division property should also be made following the corresponding propagation rules `copy`, `xor`, `and` proved in [16], [18]. Since round functions of cryptographic primitives are combinations of bitwise operations, we only need to determine the division property of the chosen plaintexts, denoted by $\mathcal{D}_{\mathbb{K}_0}^{1^n}$. Then, after r-round encryption, the division property of the output ciphertexts, denoted by $\mathcal{D}_{\mathbb{K}_r}^{1^n}$, can be deduced according to the round function and the propagation rules. More specifically, when the plaintext bits at index positions $I = \{i_1, i_2, \ldots, i_{|I|}\} \subset \{1, 2, \ldots, n\}$ are active (the active bits traverse all $2^{|I|}$ possible combinations while other bits are assigned to static 0/1 values), the division property of such chosen plaintexts is $\mathcal{D}_{\vec{k}}^{1^n}$, where $k_i = 1$ if $i \in I$ and $k_i = 0$ otherwise. Then, the propagation of the division property from $\mathcal{D}_{\vec{k}}^{1^n}$ is evaluated as $\{\vec{k}\} := \mathbb{K}_0 \to \mathbb{K}_1 \to \mathbb{K}_2 \to \cdots \to \mathbb{K}_r$ where $\mathcal{D}_{\mathbb{K}_i}$ is the division property after i-round propagation. If the division property $\mathbb{K}_r$ does not have a unit vector $\vec{e}_i$ whose only ith element is 1, the ith bit of r-round ciphertexts is balanced.

However, when round r gets bigger, the size of $\mathbb{K}_r$ expands exponentially towards $O(2^n)$ requiring huge memory resources. So the bit-based division property has only been applied to block ciphers with tiny block sizes, such as SIMON32 and Simeck32 [18]. This memory-crisis has been solved by Xiang et al. using the MILP modeling method.

Propagation of Division Property with MILP. At ASIACRYPT 2016, Xiang et al. first introduced a new concept *division trail* defined as follows:

Definition 2 (Division Trail [19]). Let us consider the propagation of the division property $\{\vec{k}\} := \mathbb{K}_0 \to \mathbb{K}_1 \to \mathbb{K}_2 \to \cdots \to \mathbb{K}_r$. For any vector $\vec{k}_{i+1}^* \in \mathbb{K}_{i+1}$, there must exist a vector $\vec{k}_i^* \in \mathbb{K}_i$ such that $\vec{k}_i^*$ can propagate to $\vec{k}_{i+1}^*$ by the propagation rule of the division property. For $(\vec{k}_0, \vec{k}_1, \ldots, \vec{k}_r) \in (\mathbb{K}_0 \times \mathbb{K}_1 \times \cdots \times \mathbb{K}_r)$ if $\vec{k}_i$ can propagate to $\vec{k}_{i+1}$ for all $i \in \{0, 1, \ldots, r-1\}$, we call $(\vec{k}_0 \to \vec{k}_1 \to \cdots \to \vec{k}_r)$ an r-round division trail.

Let E_k be the target r-round iterated cipher. Then, if there is a division trail $\vec{k}_0 \xrightarrow{E_k} \vec{k}_r = \vec{e}_j$ ($j = 1, ..., n$), the summation of jth bit of the ciphertexts is unknown; otherwise, if there is no division trail s.t. $\vec{k}_0 \xrightarrow{E_k} \vec{k}_r = \vec{e}_j$, we know the ith bit of the ciphertext is balanced (the summation of the ith bit is constant 0). Therefore, we have to evaluate all possible division trails to verify whether each bit of ciphertexts is balanced or not. Xiang et al. proved that the basic propagation rules `copy`, `xor`, `and` of the division property can be translated as some variables and constraints of an MILP model. With this method, all possible division trails can be covered with an MILP model $\mathcal{M}$ and the division property of particular output bits can be acquired by analyzing the solutions of the $\mathcal{M}$. Afterwards, some simplifications have been made to the MILP descriptions of `copy`, `xor`, `and` in [15], [21]. We show such descriptions along with our new techniques in detail in Sect. 3

2.4 The Bit-Based Division Property for Cube Attack

When the number of initialization rounds is not large enough for a thorough diffusion, the superpoly $p(\vec{x},\vec{v})$ defined in Eq. (1) may not be related to all key bits $x_1,\ldots,x_n$ corresponding to some high-dimensional cube I. Instead, there is a set of key indices $J\subseteq\{1,\ldots,n\}$ s.t. for arbitrary $\vec{v}\in\mathbb{F}_2^m$, $p(\vec{x},\vec{v})$ can only be related to x_j's ($j\in J$). In CRYPTO 2017, Todo et al. [15] proposed a method for determining such a set J using the bit-based division property. They further showed that, with the knowledge of such J, cube attacks can be launched to recover some information related to the secret key bits. More specifically, they proved the following Lemma 1 and Proposition 1.

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

Proposition 1. Let $f(\vec{x},\vec{v})$ be a polynomial, where $\vec{x}$ and $\vec{v}$ denote the secret and public variables, respectively. For a set of indices $I=\{i_1,i_2,\ldots,i_{|I|}\}\subset\{1,2,\ldots,m\}$, let C_I be a set of $2^{|I|}$ values where the variables in $\{v_{i_1},v_{i_2},\ldots,v_{i_{|I|}}\}$ are taking all possible combinations of values. Let $\vec{k}_I$ be an m-dimensional bit vector such that $\vec{v}^{\vec{k}_I}=t_I=v_{i_1}v_{i_2}\cdots v_{i_{|I|}}$, i.e. $k_i=1$ if $i\in I$ and $k_i=0$ otherwise. Assuming there is no division trail such that $(\vec{e}_\lambda,\vec{k}_I)\xrightarrow{f}1$, x_λ is not involved in the superpoly of the cube C_I.

When f represents the first output bit after the initialization iterations, we can identify J by checking whether there is a division trail $(\vec{e}_\lambda,\vec{k}_I)\xrightarrow{f}1$ for $\lambda=1,\ldots,n$ using the MILP modeling method introduced in Sect. 2.3. If the division trail $(\vec{e}_\lambda,\vec{k}_I)\xrightarrow{f}1$ exists, we have $\lambda\in J$; otherwise, $\lambda\notin J$.

When J is determined, we know that for some assignment to the non-cube $\vec{IV}\in\mathbb{F}_2^m$, the corresponding superpoly $p_{\vec{IV}}(\vec{x})$ is not constant 0, and it is a polynomial of $x_j, j\in J$. With the knowledge of J, we recover offline the superpoly $p_{\vec{IV}}(\vec{x})$ by constructing its truth table using cube summations defined as Eq. (2). As long as $p_{\vec{IV}}(\vec{x})$ is not constant, we can go to the online phase where we sum over the cube $C_I(\vec{IV})$ to get the exact value of $p_{\vec{IV}}(\vec{x})$ and refer to the precomputed truth table to identify the $x_j, j\in J$ assignment candidates. We summarize the whole process as follows:

1) **Offline Phase: Superpoly Recovery.** Randomly pick an $\vec{IV}\in\mathbb{F}_2^m$ and prepare the cube $C_I(\vec{IV})$ defined as Eq. (1). For $\vec{x}\in\mathbb{F}_2^n$ whose $x_j, j\in J$ traverse all $2^{|J|}$ 0-1 combinations, we compute and store the value of the superpoly $p_{\vec{IV}}(\vec{x})$ as Eq. (2). The $2^{|J|}$ values compose the truth table of $p_{\vec{IV}}(\vec{x})$ and the ANF of the superpoly is determined accordingly. If $p_{\vec{IV}}(\vec{x})$ is constant, we pick another $\vec{IV}$ and repeat the steps above until we find an appropriate one s.t. $p_{\vec{v}}(\vec{x})$ is not constant.
2) **Online Phase: Partial Key Recovery.** Query the cube $C_I(\vec{IV})$ to encryption oracle and get the summation of the $2^{|I|}$ output bits. We denoted the summation by $\lambda\in\mathbb{F}_2$ and we know $p_{\vec{IV}}(x)=\lambda$ according to Eq. (2). So we look up the truth table of the superpoly and only reserve the $x_j, j\in J$ s.t. $p_{\vec{IV}}(x)=\lambda$.
3) **Brute-Force Search.** Guess the remaining secret variables to recover the entire value in secret variables.

Phase 1 dominates the time complexity since it takes $2^{|I|+|J|}$ encryptions to construct the truth table of size $2^{|J|}$. It is also possible that $p_{\vec{IV}}(\vec{x})$ is constant so we have to run several different $\vec{IV}$'s to find the one we need. The attack can only be meaningful when (1) $|I|+|J|<n$; (2) appropriate $\vec{IV}$'s are easy to be found. The former requires the adversary to use "good" cube I's with small J while the latter is the exact reason why Assumptions 1 and 2 are proposed [15], [25].

3 Modeling the Constant Bits to Improve the Precision of the MILP Model

In the initial state of stream ciphers, there are secret key bits, public modifiable IV bits and constant 0/1 bits. In the previous MILP model, the initial bit-based division properties of the cube IVs are set to 1, while those of the non-cube IVs, constant state bits or even secret key bits are all set to 0.

Obviously, when constant 0 bits are involved in multiplication operations, it always results in a constant 0 output. But, as is pointed out in [25], such a phenomenon cannot be reflected in previous MILP model method. In the previous MILP model, the widely used COPY+AND operation:

$$\text{COPY+AND}:(s_1,s_2)\to(s_1,s_2,s_1\wedge s_2). \qquad (3)$$

can result in division trails $(x_1,x_2)\xrightarrow{\text{COPY+AND}}(y_1,y_2,a)$ as follows:

$$(1,0)\xrightarrow{\text{COPY+AND}}(0,0,1),$$
$$(0,1)\xrightarrow{\text{COPY+AND}}(0,0,1).$$

Assuming that either s_1 or s_2 of Eq. (3) is a constant 0 bit, $(s_1\wedge s_2)$ is always 0. In this occasion, the division property of $(s_1\wedge s_2)$ must be 0 which is overlooked by the previous MILP model. To prohibit the propagation above, an additional constraint $\mathcal{M}.con\leftarrow a=0$ should be added when either s_1 or s_2 is constant 0.

In [25], the authors only consider the constant 0 bits. They thought the model can be precise enough when all the state bits initialized to constant 0 bits are handled. But in fact, although constant 1 bits do not affect the division property propagation, we should still be aware because 0 bits might be generated when even number of constant 1 bits are XORed during the updating process. This is later shown in a practical example for Kreyvium.

3.1 Flag Technique

In order to describe the division property propagation in a more precise manner, we introduce another parameter named "flag" to each intermediate bit. With the flag technique, the intermediate state bit s correspond to 2 parameters $s.val$ and $s.F$:

- $s.val$ is the bit-based division property value described as a binary variable of the MILP model, denoted as $s.val \in \mathcal{M}.var$.
- $s.F$ is the flag value $s.F \in \{0_c, 1_c, \delta\}$ where $0_c, 1_c, \delta$ specifies whether the state bit is constant 0, constant 1 or other variable (active IV, unknown key bits or non-cube IV bits with arbitrary value etc.).

Clearly, when $v.F = 0_c/1_c$, there is always a constraint $v.val = 0 \in \mathcal{M}.con$. We define $=$, $\oplus$ and $\times$ operations for the flag value $\{1_c, 0_c, \delta\}$. The $=$ operation tests whether two elements are equal(naturally $1_c = 1_c$, $0_c = 0_c$ and $\delta = \delta$). The $\oplus$ operation follows the rules:

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

The $\times$ operation follows the rules:

$$\begin{cases} 1_c \times x = x \times 1_c = x \\ 0_c \times x = x \times 0_c = 0_c \quad \text{for arbitrary } x \in \{1_c, 0_c, \delta\} \\ \delta \times \delta = \delta \end{cases} \tag{5}$$

Therefore, in the remainder of this paper, the MILP models for COPY, XOR and AND should also consider the effects of flags. So the previous `copy`, `xor`, and `and` should now add the assignment to flags. We denote the modified versions as `copyf`, `xorf`, and `andf` and define them as Proposition 2 3 and 4 as follows.

Proposition 2 (MILP Model for COPY with Flag). Let $a \xrightarrow{\text{COPY}} (b_1, b_2, \ldots, b_m)$ be a division trail of COPY. The following inequalities are sufficient to describe the propagation of the division property for `copyf`.

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

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

Proposition 3 (MILP Model for XOR with Flag). Let $(a_1, a_2, \ldots, a_m) \xrightarrow{\text{XOR}} b$ be a division trail of XOR. The following inequalities are sufficient to describe the propagation of the division property for `xorf`.

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

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

Proposition 4 (MILP Model for AND with Flag). Let $(a_1, a_2, \ldots, a_m) \xrightarrow{\text{AND}} b$ be a division trail of AND. The following inequalities are sufficient to describe the propagation of the division property for `andf`.

$$\begin{cases} \mathcal{M}.var \leftarrow a_1.val, \ldots, a_m.val, b.val \text{ as binary.} \\ \mathcal{M}.con \leftarrow b.val \geq a_i.val \text{ for all } i \in \{1, 2, \ldots, m\} \\ b.F = a_1.F \times a_2.F \times \cdots a_m.F \\ \mathcal{M}.con \leftarrow b.val = 0 \quad \text{if } b.F = 0_c \end{cases}$$

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

Algorithm 1 Evaluate secret variables by MILP with Flags

```
procedure attackFramework(Cube indices $I$, specific assignment to non-cube IVs $\vec{IV}$ or $\vec{IV}$ = NULL)
    Declare an empty MILP model $\mathcal{M}$
    Declare $\vec{x}$ as $n$ MILP variables of $\mathcal{M}$ corresponding to secret variables.
    Declare $\vec{v}$ as $m$ MILP variables of $\mathcal{M}$ corresponding to public variables.
    $\mathcal{M}.con \leftarrow v_i = 1$ and assign $v_i.F = \delta$ for all $i \in I$
    $\mathcal{M}.con \leftarrow v_i = 0$ for all $i \in (\{1, 2, \ldots, m\} - I)$
    $\mathcal{M}.con \leftarrow \sum_{i=1}^{n} x_i = 1$ and assign $x_i.F = \delta$ for all $i \in \{1, \ldots, n\}$
    if $\vec{IV}$ = NULL then
        $v_i.F = \delta$ for all $i \in (\{1, 2, \ldots, m\} - I)$
    else
        Assign the flags of $v_i$, $i \in (\{1, 2, \ldots, m\} - I)$ as:
            $v_i.F = \begin{cases} 1_c & \text{if } \vec{IV}[i] = 1 \\ 0_c & \text{if } \vec{IV}[i] = 0 \end{cases}$
    end if
    Update $\mathcal{M}$ according to round functions and output functions
    do
        solve MILP model $\mathcal{M}$
        if $\mathcal{M}$ is feasible then
            pick index $j \in \{1, 2, \ldots, n\}$ s.t. $x_j = 1$
            $J = J \cup \{j\}$
            $\mathcal{M}.con \leftarrow x_j = 0$
        end if
    while $\mathcal{M}$ is feasible
    return $J$
end procedure
```

With these modifications, we are able to improve the precision of the MILP model. The improved attack framework can be written as Algorithm 1. It enables us to identify the involved key bits when the non-cube IVs are set to specific constant $0/1$ values by imposing corresponding flags to the non-cube MILP binary variables. With this method, we can determine an $\vec{IV} \in \mathbb{F}_2^m$ s.t. the corresponding superpoly $p_{\vec{IV}}(\vec{x}) \neq 0$.

3.2 Faster Implementation of the Flag Technique

In order to improve the efficiency of MILP model solving, we need to reduce the numbers of variables and constraints in the MILP model. We show in this part that the division property value of the constant state bits can be safely ignored from the MILP model without affecting the propagation. Since the division property value of the constant state bits is statically 0, this is equivalent to proving that the static 0 property value cannot affect the MILP model. In fact, TRIVIUM-like ciphers, as well as other stream ciphers, are just calling the following three operations namely XOR, COPY+XOR and COPY+AND:

$$\text{XOR: } (x_1, \ldots, x_m) \to y, \text{ where } y = \bigoplus_{i=1}^{m} x_i \tag{6}$$

$$\text{COPY+XOR: } (x_1, \ldots, x_m) \to (x_1, \ldots, x_m, y) \text{ where } y = \bigoplus_{i=1}^{m} x_i \tag{7}$$

$$\text{COPY+AND: } (x_1, \ldots, x_m) \to (x_1, \ldots, x_m, y) \text{ where } y = \bigwedge_{i=1}^{m} x_i \tag{8}$$

It can be proved that such operations satisfy the following Proposition 5, 6, 7 and 8. These propositions indicate that we can safely eliminate the binary variables corresponding to the division property value of the constant state bits from the MILP model while keeping exactly the same accuracy.

In other words, the constant bits can be perfectly handled only with the flag values ($*.F$'s).

Proposition 5. (Constant 0/1 in XOR) Among the m bits in Eq. (6), x_1 is constant 0/1 and others are variables. Then, the corresponding division property $(a_1, a_2, \dots, a_m) \xrightarrow{\texttt{xor}} (b)$ is the same as $(a_2 \dots, a_m) \xrightarrow{\texttt{xor}} (b)$.

Proof: The division property propagation is $(a_1, a_2, \dots, a_m) \xrightarrow{\texttt{xor}} (b)$. The following constraints are added to the MILP model: $b.val = \sum_{i=1}^m a_i.val$. Since $a_1.val = 0$ and $b.val = \sum_{i=1}^m a_i.val$, we know that $b.val = \sum_{i=2}^m a_i.val$. Therefore, the division property of XOR is equivalent to $(a_2, \dots, a_m) \xrightarrow{\texttt{xor}} (b)$. □

Proposition 6. (Constant 0/1 in COPY+XOR) Among the m bits in Eq. (7), x_1 is constant 0/1 and others are variables. Then, the corresponding division property $(a_1, a_2, \dots, a_m) \xrightarrow{\texttt{copy+xor}} (a'_1, \dots, a'_m, b)$ is the same as $(a_1, a_2 \dots, a_m) \xrightarrow{\texttt{copy+xor}} (a_1, a'_2, \dots, a'_m, b)$.

Proof: The division property propagation is

$$(a_1, a_2, \dots, a_m) \xrightarrow{\texttt{copy}} (a'_1, \dots, a'_m, a^\star_1, \dots, a^\star_m) \xrightarrow{\texttt{xor}} (a'_1, \dots, a'_m, b)$$

The following constraints are added to the MILP model:

$$\begin{cases} a_i.val = a'_i.val + a^\star_i.val \\ b.val = \sum_{i=1}^m a^\star_i.val \end{cases} \quad \text{for } i = 1, \dots, m$$

Since $a_1.val = 0$ and $a_1.val = a'_1.val + a^\star_1.val$, we know $a'_1.val = 0$ and $a^\star_1.val = 0$. According to Proposition 6, we know $a^\star_1.val$ does not affect the newly generated $b.val$. Therefore, $a_1.val$ does not affect $b.val$ at all. □

Proposition 7. (Constant 1 in COPY+AND) Among the m bits in Eq. (8), x_1 is constant 1 and others are variables. Then, the corresponding division property $(a_1, a_2, \dots, a_m) \xrightarrow{\texttt{copy+and}} (a'_1, \dots, a'_m, b)$ is the same as $(a_1, a_2, \dots, a_m) \xrightarrow{\texttt{copy+and}} (a_1, a'_2 \dots, a'_m, b)$.

Proof: The division property propagation is

$$(a_1, a_2, \dots, a_m) \xrightarrow{\texttt{copy}} (a'_1, \dots, a'_m, a^\star_1, \dots, a^\star_m) \xrightarrow{\texttt{and}} (a'_1, \dots, a'_m, b)$$

The following constraints are added to the MILP model:

$$\begin{cases} a_i.val = a'_i.val + a^\star_i.val \\ b.val \geq a^\star_i.val \end{cases} \quad \text{for } i = 1, \dots, m$$

Since $a_1.val = 0$ and $a_1.val = a'_1.val + a^\star_1.val$, we know $a'_1.val = 0$ and $a^\star_1.val = 0$. Therefore, even if we ignore the constraints $a_1.val = a'_1.val + a^\star_1.val$ and $b.val \geq a^\star_1.val$, it does not affect the $b.val$ at all. □

Proposition 8. (Constant 0 in COPY+AND) Among the m bits in Eq. (8), x_1 is constant 0 and others are variables. Then, the corresponding division property $(a_1, a_2, \dots, a_m) \xrightarrow{\texttt{copy+and}} (a'_1, \dots, a'_m, b)$ is the same as $(a_1 \dots, a_m) \xrightarrow{\texttt{copy+and}} (a_1, \dots, a_m, 0)$.

Proof: The division property propagation is

$$(a_1, a_2, \dots, a_m) \xrightarrow{\texttt{copy}} (a'_1, \dots, a'_m, a^\star_1, \dots, a^\star_m) \xrightarrow{\texttt{and}} (a'_1, \dots, a'_m, b)$$

The following constraints are added to the MILP model:

$$\begin{cases} a_i.val = a'_i.val + a^\star_i.val & \text{for } i = 1, \dots, m \\ b.val \geq a^\star_i.val & \text{for } i = 1, \dots, m \\ b.val = 0 \end{cases}$$

Since $b.val = 0$ and $b.val \geq a^\star_i.val$ $(i = 1, \dots, m)$, we know that $a^\star_i.val = 0$. Since $a_i.val = a'_i.val + a^\star_i.val$ and $a^\star_i.val = 0$, we have $a_i.val = a'_i.val$. Therefore, the division property of COPY+AND is equivalent to $(a_1, a_2, \dots, a_m) \xrightarrow{\texttt{copy+and}} (a_1, \dots, a_m, 0)$. □

As can be seen in Proposition 5, the division value of constant 0/1 does not affect the division propagation of XOR operation. Therefore, for the newly generated variable b (representing the division property value of the newly generated state bit y), we only need to focus on its flag value $b.F$, which can be computed independently to the division property value.

According to Proposition 5 and 6, we can safely ignore the constant 0/1 bits involved in the XOR and COPY+XOR operation without affecting the MILP model. Proposition 7 further reveals that constant 1 bits involved in COPY+AND can also be ignored. In other words, the division property value ($a_1.val$) of these constant bits cannot affect the MILP model in these situations. It's safe for us to only consider their flags ($a_1.F$)

Specifically in Proposition 8, we know that when constant 0 bits are involved in COPY+AND, it is equivalent to declare a new variable with division property value 0, indicating that the bit generated by COPY+AND is constant 0. But $b.F$ computed from $a_i.F$'s can already determine the constant 0 property of the newly generated bit. So, in this situation, we can still ignore the division property value of the constant 0 bit ($a_1.val$) and only consider its flag ($a_1.F$).

For our modified flag technique, when the XOR, COPY+XOR, COPY+AND operations are triggered, the flags are first computed. The corresponding constraints are added to the models only if the newly generated bits are non-constant as demonstrated in Algorithm 2, 3 and 4. We also rewrite the `copyf` as Algorithm 5.

Algorithm 2 Improved flag technique for describing the division property propagation corresponding to the XOR operation in Eq. (6)

```
1: procedure xorf($\mathcal{M}, a_1, \dots, a_m$)
2:     Declare a new variable $b$ and construct a set of indices
               $\lambda := \{i \in \{1, \dots, m\} : a_i.F = \delta\}$
3:     Assign the flag of $b$ as $b.F \leftarrow \sum_{i=1}^m a_i.F$
4:     if $b.F = 1_c$ or $b.F = 0_c$ then
5:         Assign $b.val = NULL$
6:     else
7:         Declare a binary variable in the MILP model $\mathcal{M}.var \leftarrow b.val$
8:         Add a constraint to the model $\mathcal{M}.con \leftarrow b.val = \sum_{i \in \lambda} a_i.val$
9:     end if
10:    return $(\mathcal{M}, b)$
11: end procedure
```

4 Upper Bounding the Degree of the Superpoly

For an $\vec{IV} \in \mathbb{F}_2^m$ s.t. $p_{\vec{IV}}(\vec{x}) \neq 0$, the ANF of $p_{\vec{IV}}(\vec{x})$ can be represented as

$$p_{\vec{IV}}(\vec{x}) = \sum_{\vec{u} \in \mathbb{F}_2^n} a_{\vec{u}} \vec{x}^{\vec{u}} \tag{15}$$

where $a_{\vec{u}}$ is determined by the values of the non-cube IVs. If the degree of the superpoly is upper bounded by d, then for all $\vec{u}$'s with Hamming weight satisfying $hw(\vec{u}) > d$, we

Algorithm 3 Improved flag technique for describing the division property propagation corresponding to the COPY+XOR operation in Eq. (7)

```
1: procedure copy+xorf($\mathcal{M}, a_1, \dots, a_m$)
2:     Declare a new variable $b$ and construct an set of index
           $\lambda := \{i \in \{1, \dots, m\} : a_i.F = \delta\}$
3:     Assign the flag of $b$ as $b.F \leftarrow \sum_{i=1}^m a_i.F$
4:     if $b.F = 1_c$ or $b.F = 0_c$ then
5:         Assign $b.val = NULL$ and declare $a'_i = a_i$ for $i = 1, \dots, m$.
6:     else
7:         Declare a binary variable in the MILP model $\mathcal{M} \leftarrow b.val$
8:         Add variables and constraints to $\mathcal{M}$ as:
               $\mathcal{M}.var \leftarrow$ binary variable $a'_i.val, a^\star_i.val$ for $i \in \lambda$    (9)
               $\mathcal{M}.con \leftarrow a_i.val = a'_i.val + a^\star_i.val$ for $i \in \lambda$    (10)
               $\mathcal{M}.var \leftarrow$ binary variable $b.val$    (11)
               $\mathcal{M}.con \leftarrow b.val = \sum_{i \in \lambda} a^\star_i.val$
9:         For $i \in \lambda$, assign $a'_i.F \leftarrow a_i.F$ and $a^\star_i.F \leftarrow a_i.F$
10:        For $i \notin \lambda$, declare $a'_i$ and assign $a'_i \leftarrow a_i$ .
11:    end if
12:    return $(\mathcal{M}, a'_1, \dots, a'_m, b)$
13: end procedure
```

Algorithm 4 Improved flag technique for describing the division property propagation corresponding to the COPY+AND operation in Eq. (8)

```
1: procedure copy+andf($\mathcal{M}, a_1, \dots, a_m$)
2:     Declare a new variable $b$ and construct an set of index $\lambda := \{i \in \{1, \dots, m\} : a_i.F = \delta\}$
3:     Assign the flag of $b$ as $b.F \leftarrow \prod_{i=1}^m a_i.F$
4:     if $b.F = 1_c$ or $b.F = 0_c$ then
5:         Assign $b.val = NULL$ and declare $a'_i \leftarrow a_i$ for $i = 1, \dots, m$.
6:     else
7:         Declare a binary variable in the MILP model $\mathcal{M} \leftarrow b.val$
8:         Add variables and constraints to the model $\mathcal{M}$
               $\mathcal{M}.var \leftarrow$ binary variable $a'_i.val, a^\star_i.val$ for $i \in \lambda$    (12)
               $\mathcal{M}.con \leftarrow a_i.val = a'_i.val + a^\star_i.val$ for $i \in \lambda$    (13)
               $\mathcal{M}.var \leftarrow$ binary variable $b.val$    (14)
               $\mathcal{M}.con \leftarrow b.val \geq a^\star_i.val$ for $i \in \lambda$
9:         For $i \in \lambda$, assign $a'_i.F \leftarrow a_i.F$ and $a^\star_i.F \leftarrow a_i.F$
10:        For $i \notin \lambda$, declare $a'_i$ and assign $a'_i \leftarrow a_i$ .
11:    end if
12:    return $(\mathcal{M}, a'_1, \dots, a'_m, b)$
13: end procedure
```

constantly have $a_{\vec{u}} = 0$. In this case, we no longer have to build the whole truth table to recover the superpoly . Instead, we only need to determine the coefficients $a_{\vec{u}}$ for $hw(\vec{u}) \leq d$. Therefore, we select $\sum_{i=0}^{d} \binom{|J|}{i}$ different $\vec{x}$'s and construct a linear system with $\left(\sum_{i=0}^{d} \binom{|J|}{i}\right)$ variables and

Algorithm 5 Improved flag technique for describing the division property propagation corresponding to the COPY operation originated in Proposition 2

```
1: procedure copyf($\mathcal{M}$, division property $a$, copy number $m$ )
2:     Declare $m$ new variable $b_1, \dots, b_m$ and assign their flag values as $b_i.F = a.F$.
3:     if $a.F = 1_c$ or $a.F = 0_c$ then
4:         Assign $b_i.val = NULL$ for $i = 1, \dots, m$.
5:     else
6:         Declare a binary variable in the MILP model $\mathcal{M}.var \leftarrow b_i.val$ for $i = 1, \dots, m$.
7:         Add a constraint to the model $\mathcal{M}.con \leftarrow a.val = \sum_{i=1}^m b_i.val$
8:     end if
9:     return $(\mathcal{M}, b_1, \dots, b_m)$
10: end procedure
```

the coefficients as well as the whole ANF of $p_{\vec{IV}}(\vec{x})$ can be recovered by solving such a linear system. So the complexity of Phase 1 can be reduced from $2^{|I|+|J|}$ to $2^{|I|} \times \sum_{i=0}^{d} \binom{|J|}{i}$. For the simplicity of notations, we denote the summation $\sum_{i=0}^{d} \binom{|J|}{i}$ as $\binom{|J|}{\leq d}$ in the remainder of this paper. With the knowledge of the involved key indices $J = \{j_1, j_2, \dots, j_{|J|}\}$ and the degree of the superpoly $d = \deg p_{\vec{IV}}(\vec{x})$, the attack procedure can be adapted as follows:

1) **Offline Phase: Superpoly Recovery.** For all $\binom{|J|}{\leq d}$ $\vec{x}$'s satisfying $hw(\vec{x}) \leq d$ and $\bigoplus_{j \in J} \vec{e}_j \succeq \vec{x}$, compute the values of the superpolies as $p_{\vec{IV}}(\vec{x})$ by summing over the cube $C_I(\vec{IV})$ as Eq. (2) and generate a linear system of the $\binom{|J|}{\leq d}$ coefficients $a_{\vec{u}}$ ($hw(\vec{u}) \leq d$). Solve the linear system, determine the coefficient $a_{\vec{u}}$ of the $\binom{|J|}{\leq d}$ terms and store them in a lookup table T. The ANF of the $p_{\vec{IV}}(\vec{x})$ can be determined with the lookup table.
2) **Online Phase: Partial Key Recovery.** Query the encryption oracle and sum over the cube $C_I(\vec{IV})$ as Eq. (2) and acquire the exact value of $p_{\vec{IV}}(\vec{x})$. For each of the $2^{|J|}$ possible values of $\{x_{j_1}, \dots, x_{j_{|J|}}\}$, compute the values of the superpoly as Eq. (15) (the coefficient $a_{\vec{u}}$ are acquired by looking up the precomputed table T) and identify the correct key candidates.
3) **Brute-force search phase.** Attackers guess the remaining secret variables to recover the entire value in secret variables.

The complexity of Phase 1 becomes $2^{|I|} \times \binom{|J|}{\leq d}$. Phase 2 now requires $2^{|I|}$ encryptions and $2^{|J|} \times \binom{|J|}{\leq d}$ table lookups, so the complexity can be regarded as $2^{|I|} + 2^{|J|} \times \binom{|J|}{\leq d}$. The complexity of Phase 3 remains 2^{n-1}. Therefore, the number of encryptions a feasible attack requires is

$$\max\left\{2^{|I|} \times \binom{|J|}{\leq d}, 2^{|I|} + 2^{|J|} \times \binom{|J|}{\leq d}\right\} < 2^n. \tag{16}$$

The previous limitation of $|I| + |J| < n$ is removed.

The knowledge of the algebraic degree of superpolies can largely benefit the efficiency of the cube attack. Therefore, we show how to estimate the algebraic degree of superpolies using the division property. Before the introduction of the method, we generalize Proposition 1 as follows.

Proposition 9. Let $f(\vec{x}, \vec{v})$ be a polynomial, where $\vec{x}$ and $\vec{v}$ denote the secret and public variables, respectively. For a set of indices $I = \{i_1, i_2, \dots, i_{|I|}\} \subset \{1, 2, \dots, m\}$, let C_I be a set of $2^{|I|}$ values where the variables in $\{v_{i_1}, v_{i_2}, \dots, v_{i_{|I|}}\}$ are taking all possible combinations of values. Let $\vec{k}_I$ be an m-dimensional bit vector such that $\vec{v}^{\vec{k}_I} = t_I = v_{i_1} v_{i_2} \cdots v_{i_{|I|}}$. Let $\vec{k}_\Lambda$ be an n-dimensional bit vector. Assuming there is no division trail such that $(\vec{k}_\Lambda \| \vec{k}_I) \xrightarrow{f} 1$, the monomial $\vec{x}^{\vec{k}_\Lambda}$ is not involved in the superpoly of the cube C_I.

Proof: The ANF of $f(\vec{x}, \vec{v})$ is represented as follows

$$f(\vec{x}, \vec{v}) = \bigoplus_{\vec{u} \in \mathbb{F}_2^{n+m}} a^f_{\vec{u}} \cdot (\vec{x} \| \vec{v})^{\vec{u}},$$

where $a^f_{\vec{u}} \in \mathbb{F}_2$ denotes the ANF coefficients. The polynomial $f(\vec{x}, \vec{v})$ is decomposed into

$$\begin{aligned} f(\vec{x},\vec{v}) =& \bigoplus_{\vec{u}\in\mathbb{F}_2^{n+m}|\vec{u}\succeq(\vec{0}\|\vec{k}_I)} a^f_{\vec{u}}\cdot(\vec{x}\|\vec{v})^{\vec{u}} \oplus \\ & \bigoplus_{\vec{u}\in\mathbb{F}_2^{n+m}|\vec{u}\not\succeq(\vec{0}\|\vec{k}_I)} a^f_{\vec{u}}\cdot(\vec{x}\|\vec{v})^{\vec{u}} \\ =& \bigoplus_{\vec{u}\in\mathbb{F}_2^{n+m}|\vec{u}\succeq(\vec{0}\|\vec{k}_I)} t_I\cdot a^f_{\vec{u}}\cdot(\vec{x}\|\vec{v})^{\vec{u}\oplus(\vec{0}\|\vec{k}_I)} \oplus \\ & \bigoplus_{\vec{u}\in\mathbb{F}_2^{n+m}|\vec{u}\not\succeq(\vec{0}\|\vec{k}_I)} a^f_{\vec{u}}\cdot(\vec{x}\|\vec{v})^{(\vec{0}\|\vec{u})} \\ =& \, t_I\cdot p(\vec{x},\vec{v})\oplus q(\vec{x},\vec{v}). \end{aligned}$$

Therefore, the superpoly $p(\vec{x}, \vec{v})$ is represented as

$$p(\vec{x},\vec{v}) = \bigoplus_{\vec{u}\in\mathbb{F}_2^{n+m}|\vec{u}\succeq(\vec{0}\|\vec{k}_I)} a^f_{\vec{u}}\cdot(\vec{x}\|\vec{v})^{\vec{u}\oplus(\vec{0}\|\vec{k}_I)}.$$

Since there is no division trail $(\vec{k}_\Lambda\|\vec{k}_I) \xrightarrow{f} 1$, $a^f_{\vec{u}} = 0$ for $\vec{u} \succeq (\vec{k}_\Lambda\|\vec{k}_I)$ because of Lemma 1. Therefore,

$$p(\vec{x},\vec{v}) = \bigoplus_{\vec{u}\in\mathbb{F}_2^{n+m}|\vec{u}\succeq(\vec{0}\|\vec{k}_I),\vec{u}^{\vec{k}_\Lambda\|\vec{0}}=0} a^f_{\vec{u}}\cdot(\vec{x}\|\vec{v})^{\vec{u}\oplus(\vec{0}\|\vec{k}_I)}.$$

This superpoly is independent of the monomial $\vec{x}^{\vec{k}_\Lambda}$ since $\vec{u}^{\vec{k}_\Lambda\|\vec{0}}$ is always 0. □

Algorithm 6 Evaluate upper bound of algebraic degree on the superpoly

```
1: procedure DegEval(Cube indices $I$, specific assignment to non-cube IVs $\vec{IV}$ or $\vec{IV}$ = NULL)
2:     Declare an empty MILP model $\mathcal{M}$.
3:     Declare $\vec{x}$ be $n$ MILP variables of $\mathcal{M}$ corresponding to secret variables.
4:     Declare $\vec{v}$ be $m$ MILP variables of $\mathcal{M}$ corresponding to public variables.
5:     $\mathcal{M}.con \leftarrow v_i = 1$ and assign the flags $v_i.F = \delta$ for all $i \in I$
6:     $\mathcal{M}.con \leftarrow v_i = 0$ for $i \in (\{1,\ldots,n\} - I)$
7:     if $\vec{IV}$ = NULL then
8:         Assign the flags $v_i.F = \delta$ for $i \in (\{1,\ldots,n\} - I)$
9:     else
10:        Assign the flags of $v_i$, $i \in (\{1,2,\ldots,n\} - I)$ as:
               $v_i.F = \begin{cases} 1_c & \text{if } \vec{IV}[i] = 1 \\ 0_c & \text{if } \vec{IV}[i] = 0 \end{cases}$
11:    end if
12:    Set the objective function $\mathcal{M}.obj \leftarrow \max \sum_{i=1}^{n} x_i$
13:    Update $\mathcal{M}$ according to round functions and output functions
14:    Solve MILP model $\mathcal{M}$
15:    return The solution of $\mathcal{M}$.
16: end procedure
```

According to Proposition 9, the existence of the division trail $(\vec{k}_\Lambda||\vec{k}_I) \xrightarrow{f} 1$ is in accordance with the existence of the monomial $x^{\vec{k}_\Lambda}$ in the superpoly of the cube C_I.

If there is $d \geq 0$ s.t. for all $\vec{k}_\Lambda$ of hamming weight $hw(\vec{k}_\Lambda) > d$, the division trail $x^{\vec{k}_\Lambda}$ does not exist, then we know that the algebraic degree of the superpoly is bounded by d. Using MILP, this d can be naturally modeled as the maximum of the objective function $\sum_{j=1}^{n} x_j$. With the MILP model $\mathcal{M}$ and the cube indices I, we can bound the degree of the superpoly using Algorithm 6. Same as Algorithm 1, we can also consider the degree of the superpoly for specific assignment to the non-cube IVs. So we also add the input $\vec{IV}$ that can either be a specific assignment or a NULL referring to an arbitrary assignment. The solution $\mathcal{M}.obj = d$ is the upper bound of the superpoly's algebraic degree. Furthermore, corresponding to $\mathcal{M}.obj = d$ and according to the definition of $\mathcal{M}.obj$, there should also be a set of indices $\{l_1, \ldots, l_d\}$ s.t. the variables representing the initially declared $\vec{x}$ (representing the division property of the key bits) satisfy the constraints $x_{l_1} = \ldots = x_{l_d} = 1$. We can also enumerate all t-degree ($1 \leq t \leq d$) monomials involved in the superpoly using a similar technique which we will detail later in Sect. 7.

5 Application of Flag Technique and Degree Evaluation to Trivium-Like Stream Ciphers

In this part, we apply our method to 4 TRIVIUM-like stream ciphers namely TRIVIUM [38], Kreyvium [39], TriviA-SC1 and TriviA-SC2 [40]. Such primitives share many structural similarities. For example, the structure of such ciphers is a concatenation of 3 NFSRs; each NFSR use the same 2-degree core function as its updating function; the output functions are quite simple; the numbers of initialization rounds are all 1152. Too many initialization rounds will cause significant expansion in the scale of MILP models making them unsolvable. But according to the structural features, several shortcuts can be used to simplify the model. We detail the application of our method to TRIVIUM. For others, we simply specify the differences and list the theoretic results.

5.1 Application to Trivium

5.1.1 *Specification of* Trivium

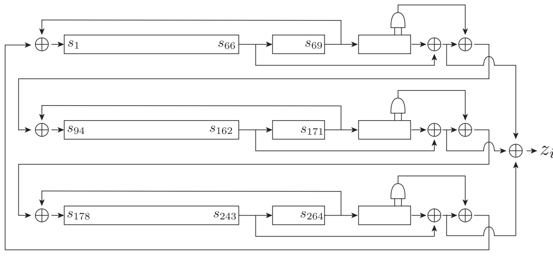

Fig. 1. Structure of TRIVIUM

TRIVIUM has a 288-bit state denoted as $\vec{s} = (s_1, s_2, \ldots, s_{288})$. Its key and IV lengths are both 80 bits. The initial state $\vec{s}^0$ is assigned as Eq. (17).

$$\begin{aligned} (s_1^0, s_2^0, \ldots, s_{93}^0) &= (K_1, K_2, \ldots, K_{80}, 0, \ldots, 0), \\ (s_{94}^0, s_{95}^0, \ldots, s_{177}^0) &= (IV_1, IV_2, \ldots, IV_{80}, 0, \ldots, 0), \\ (s_{178}^0, s_{279}^0, \ldots, s_{288}^0) &= (0, 0, \ldots, 0, 1, 1, 1). \end{aligned} \quad (17)$$

Fig. 1 shows the state update function of TRIVIUM which updates the internal state from $\vec{s}^{r-1}$ to $\vec{s}^r$ for R ($r = 1, 2, \ldots, R$, $R = 1152$ for full TRIVIUM) initialization rounds:

$$\begin{aligned} & t_1^{r-1} \leftarrow s_{91}^{r-1} \cdot s_{92}^{r-1} \oplus s_{66}^{r-1} \oplus s_{93}^{r-1} \oplus s_{171}^{r-1} \\ & t_2^{r-1} \leftarrow s_{175}^{r-1} \cdot s_{176}^{r-1} \oplus s_{162}^{r-1} \oplus s_{177}^{r-1} \oplus s_{264}^{r-1} \\ & t_3^{r-1} \leftarrow s_{286}^{r-1} \cdot s_{287}^{r-1} \oplus s_{243}^{r-1} \oplus s_{288}^{r-1} \oplus s_{69}^{r-1} \\ & (s_1^r, s_2^r, \ldots, s_{93}^r) \leftarrow (t_3^{r-1}, s_1^{r-1}, \ldots, s_{92}^{r-1}) \\ & (s_{94}^r, s_{95}^r, \ldots, s_{177}^r) \leftarrow (t_1^{r-1}, s_{94}^{r-1}, \ldots, s_{176}^{r-1}) \\ & (s_{178}^r, s_{279}^r, \ldots, s_{288}^r) \leftarrow (t_2^{r-1}, s_{178}^{r-1}, \ldots, s_{287}^{r-1}) \end{aligned} \quad (18)$$

After R rounds of initialization, TRIVIUM outputs the 1st key stream bit z^R by calling the output function:

$$z^R = s^R_{66} \oplus s^R_{93} \oplus s^R_{162} \oplus s^R_{177} \oplus s^R_{243} \oplus s^R_{288} \tag{19}$$

z^R can be regarded as a XOR combination of 6 independent terms denoted as

$$\begin{aligned} z^R_1 = s^R_{66}, z^R_2 = s^R_{93}, z^R_3 = s^R_{162}, \\ z^R_4 = s^R_{177}, z^R_5 = s^R_{243}, z^R_6 = s^R_{288} \end{aligned} \tag{20}$$

5.1.2 MILP Model of TRIVIUM

TRIVIUM-like stream ciphers share a 2-degree core function denoted as f_{core} that takes as input a state $\vec{s}$ and 5 indices $i_1, \ldots, i_5$, and outputs a new state $\vec{s'} \leftarrow f_{core}(\vec{s}, i_1, \ldots, i_5)$ where

$$s'_i = \begin{cases} s_{i_1}s_{i_2} + s_{i_3} + s_{i_4} + s_{i_5}, & i = i_5 \\ s_i, & \text{otherwise} \end{cases} \tag{21}$$

The division property propagation for the core function can be represented as Algorithm 7. The input of Algorithm 7 consists of $\mathcal{M}$ as the current MILP model, a vector $\vec{x}$ describing the current division property of the state bits, and 5 indices i_1, i_2, i_3, i_4, i_5 corresponding to positions the input bits. Then Algorithm 7 outputs the updated model $\mathcal{M}$, and vector $\vec{y}$ describes the division property after f_{core}.

Divide-and-Conquer Technique for Simple MILP Models. According to (20), the output bit z^R is a summation of 6 terms. Accordingly, we divide z_R into 6 parts Instead of constructing a MILP model for evaluating the division property of z^R, we construct 6 MILP models and evaluate z^R_i $(i = 1, \ldots, 6)$ independently. For each z^R_i, we can acquire a set of key indices J_i so that the superpoly of z^R_i is related to key bits x_j $(j \in J_i)$ so that the set containing all involved key bits can be computed as:

$$J = \bigcup_{i=1}^{6} J_i. \tag{22}$$

For degree evaluation, we also solve 6 solutions $d_1, \ldots, d_6$ corresponding to $z^R_1, \ldots, z^R_6$ and the degree of the targeted superpoly is

$$d = \max\{d_1, \ldots, d_6\} \tag{23}$$

As has been illustrated in [15], 45 constraints are added in the MILP model $\mathcal{M}$ in order to describe the division property propagation in 1-round updating. Therefore, for R-round TRIVIUM, $\mathcal{M}$ should contain $45R$ constraints. But for each of the 6 MILP models, the number of constraints can be largely diminished. For $z^R_1 = s^R_{66}$, we know that $s^R_{66} = s^{R-65}_1$. In other words, s^R_{66} is first generated at round $R-65$. Therefore, we have $z^R_1 = s^R_{66} = s^{R-65}_1$ and we can evaluate the division property of z^R_1 after $(R-65)$ rounds of initializations so the number of constraints in the MILP model shrinks to $45(R-65)$. Similarly, $z^R_2 = s^R_{93} = s^{R-92}_1$ so it only requires $45(R-92)$ constraints. The numbers of constraints for $z^R_3, \ldots, z^R_6$ are $45(R-68)$, $45(R-83)$, $45(R-65)$ and $45(R-110)$, respectively. With fewer constraints, the MILP models can be solved more easily. According to our experience, solving 6 small models separately can be much more time-saving than solving a large one. For z^R_i $(i = 1, \ldots, 6)$, the MILP can be constructed by calling Algorithm 8. This is a subroutine of Algorithm 1 and 6 to evaluate $J_1, \ldots, J_6$ and $d_1, \ldots, d_6$ respectively. The final involved key bits J and the degree d can be computed as Eq. (22) and Eq. (23). Note that constraints to the input division property are imposed by Algorithm 1 and 6.

Algorithm 7 MILP model of division property for the core function (21)

```
1: procedure Core($\mathcal{M}, \vec{x}, i_1, i_2, i_3, i_4, i_5$)
2:     Call Algorithm 4 as $(\mathcal{M}, y_{i_1}, y_{i_2}, z_1) \leftarrow \texttt{copy} + \texttt{andf}(\mathcal{M}, x_{i_1}, x_{i_2})$.
3:     Call Algorithm 3 as $(\mathcal{M}, y_{i_3}, y_{i_4}, z_2) \leftarrow \texttt{copy} + \texttt{xorf}(\mathcal{M}, x_{i_3}, x_{i_4})$.
4:     Call Algorithm 2 as $(\mathcal{M}, y_{i_5}) \leftarrow \texttt{xorf}(\mathcal{M}, z_1, z_2, x_{i_5})$
5:     for all $i \in \{1, 2, \ldots, 288\}$ w/o $i_1, i_2, i_3, i_4, i_5$ do
6:         $y_i = x_i$
7:     end for
8:     return $(\mathcal{M}, \vec{y})$
9: end procedure
```

Algorithm 8 MILP model of division property for TRIVIUM

```
1: procedure TriviumEval(round $R$, index $i \in [1, 6]$ of $z^R_i$, $\vec{v}$ and $\vec{x}$ reflect the division property and flag values of IV and key bits )
2:     Initialize 288-element vector $\vec{s}^0$.
3:     For $j = 1, \ldots, 80$, set $s^0_j = x_j$ and $s^0_{j+93} = v_j$.
4:     $\mathcal{M}.con \leftarrow s^0_i.val = NULL$ for $i = 81, \ldots, 93, 174, \ldots, 288$.
5:     $s^0_i.F = 0_c$ for $i = 81, \ldots, 285$ and $s^0_j.F = 1_c$ for $j = 286, 287, 288$.
6:     For $i = 1, \ldots, 6$, define $(i', R') = (1, R-65), (1, R-92), (94, R-68), (94, R-83), (178, R-65), (178, R-110)$ respectively.
7:     for $r = 1$ to $R'$ do
8:         $(\mathcal{M}, \vec{x})$ = Core($\mathcal{M}, \vec{s}^{r-1}$, 91, 92, 66, 171, 93)
9:         $(\mathcal{M}, \vec{y})$ = Core($\mathcal{M}, \vec{x}$, 175, 176, 162, 264, 177)
10:        $(\mathcal{M}, \vec{z})$ = Core($\mathcal{M}, \vec{y}$, 286, 287, 243, 69, 288)
11:        $\vec{s}^r = \vec{z} \ggg 1$
12:    end for
13:    for all $j \in \{1, 2, \ldots, 288\} \backslash \{i'\}$ do
14:        $\mathcal{M}.con \leftarrow s^{R'}_j.val = 0$
15:    end for
16:    $\mathcal{M}.con \leftarrow s^{R'}_{i'}.val = 1$
17:    return $\mathcal{M}$
18: end procedure
```

5.1.3 Experimental Verification

Identical to [15], we use the cube $I = \{1, 11, 21, 31, 41, 51, 61, 71\}$ to verify our attack and implementation. The experimental verification includes: the degree evaluation using Algorithm 6, specifying involved key bits using Algorithm 1 with $\vec{IV} = \texttt{NULL}$ or specific non-cube IV settings.

Example 1 (Verification of Our Attack against 591-round* TRIVIUM*). With $\vec{IV} = \texttt{NULL}$ using Algorithm 1, we are able to identify $J = \{23, 24, 25, 66, 67\}$. We know that with some assignment to the non-cube IV bits, the superpoly can be a polynomial of secret key bits $x_{23}, x_{24}, x_{25}, x_{66}, x_{67}$. These are the same as [15]. Then, we set $\vec{IV}$ to random values and acquire the degree through Algorithm 6, and verify the correctness of the degree by practically recovering the corresponding superpoly.

- When we set $\vec{IV} = \texttt{0xcc2e487b}, \texttt{0x78f99a93}, \texttt{0xbeae}$, and run Algorithm 6, we get the degree 3. The practically recovered superpoly is also of degree 3:

$$p_{\vec{v}}(\vec{x}) = x_{66}x_{23}x_{24} + x_{66}x_{25} + x_{66}x_{67} + x_{66},$$

which is in accordance with the deduction by Algorithm 6 through MILP model.

TABLE 2
Summary of theoretical cube attacks on TRIVIUM.

#Rounds	$\|I\|$	Degree	Involved Key bits J	#Time
832	72†	3	34, 58, 59, 60, 61 ($\|J\| = 5$)	$2^{76.7}$
833	73‡	3	49, 58, 60, 74, 75, 76 ($\|J\| = 7$)	2^{79}
839	78●	1	61 ($\|J\| = 1$)	2^{79}

†: $I =$ $\{1, 2, ..., 65, 67, 69, 71, 73, 75, 77, 79\}$
‡: $I =$ $\{1, 2, ..., 67, 69, 71, 73, 75, 77, 79\}$
●: $I =$ $\{1, ..., 33, 35, ..., 46, 48, ..., 80\}$ and $\vec{IV}[47] = 1$

- When we set $\vec{IV} = \texttt{0x61fbe5da}, \texttt{0x19f5972c}, \texttt{0x65c1}$, the degree evaluation of Algorithm 6 is 2. The practically recovered superpoly is also of degree 2:
$$p_{\vec{v}}(\vec{x}) = x_{23}x_{24} + x_{25} + x_{67} + 1.$$
- When we set $\vec{IV} = \texttt{0x5b942db1}, \texttt{0x83ce1016}, \texttt{0x6ce}$, the degree is 0 and the superpoly recovered is also constant 0.

On the accuracy of MILP model with flag technique. As a comparison, we use the cube above and conduct practical experiments on different rounds namely 576, 577, 587, 590, 591 (selected from Table 2 of [25]). We try 10000 randomly chosen $\vec{IV}$'s. For each of them, we use the MILP method to evaluate the degree d, in comparison with the practically recovered ANF of the superpoly $p_{\vec{IV}}(\vec{x})$. For 576, 577, 587 and 590 rounds, the accuracy is 100%.

For 591 rounds, the accuracies are distributed as:

1) When the MILP model gives degree evaluation $d = 0$, the accuracy is 100% that the superpoly is constant 0.
2) When the MILP model gives degree evaluation $d = 3$, there is an accuracy 49% that the superpoly is a 3-degree polynomial. For the rest, the superpoly is constant 0.
3) When the MILP model gives degree evaluation $d = 2$, there is accuracy 43% that the superpoly is a 2-degree polynomial. For the rest, the superpoly is constant 0.

The ratios of error can easily be understood: for example, in some case, one key bit may multiply with constant 1 in one step $x_i \cdot 1$ and be canceled by XORing with itself in the next round, this results in a newly generated constant 0 bit ($(x_i \cdot 1) \oplus x_i = 0$). However, by the flag technique, this newly generated bit has flag value $\delta = (\delta \times 1_c) + \delta$. In our attacks, the size of cubes tends to be large, which means that most of the IV bits become active, the above situation of $(x_i \cdot 1) \oplus x_i = 0$ will now become $(x_i \cdot v_j) \oplus x_i$. Therefore, when larger cubes are used, fewer constant 0/1 flags are employed, and the MILP models are becoming closer to those of $\vec{IV} = NULL$. It is predictable that the accuracy of the flag technique tends to increase when larger cubes are used. To verify this statement, we construct a 10-dimensional cube $I = \{5, 13, 18, 22, 30, 57, 60, 65, 72, 79\}$ for 591-round TRIVIUM. When $\vec{IV} = NULL$, we acquire the same upper bound of the degree $d = 3$. Then, we tried thousands of random IVs, and get an overall accuracy 80.9%. From above, we can conclude that the flag technique has high precision and can definitely improve the efficiency of the division property based cube attacks.

5.1.4 Theoretical Results

The best result in [15] mounts to 832-round TRIVIUM with cube dimension $|I| = 72$ and the superpoly involves $|J| = 5$ key bits. The complexity is 2^{77} in [15]. Using Algorithm 6, we further acquire that the degree of such a superpoly is 3. So the complexity for superpoly recovery is $2^{72} \times \binom{5}{\le 3} = 2^{76.7}$ and the complexity for recovering the partial key is $2^{72}+2^3\times\binom{5}{3}$. Therefore, according to Eq. (16), the complexity of this attack is $2^{76.7}$. A similar 73-dimensional cube is also constructed for 833-round TRIVIUM and the complexity for key recovery is 2^{79}.

TRIVIUM has many cubes whose superpolys only contain 1 key bit. These cubes are of great value for our key recovery attacks. Firstly, the truth table of such superpoly is balanced and the Partial Key Recovery phase can definitely recover 1 bit of secret information. Secondly, the Superpoly Recovery phase only requires $2^{|I|+1}$ and the online Partial Key Recovery only requires $2^{|I|}$ encryptions. Such an attack can be meaningful as long as $|I| + 1 < 80$, so we can try cubes having dimension as large as 78. Therefore, we investigate 78-dimensional cubes and find the best cube attack on TRIVIUM is 839 rounds. By running Algorithm 1 with $2^2 = 4$ different assignments to non-cube IVs, we know that the key bit x_{61} is involved in the superpoly for $\vec{IV} = \texttt{0x0}, \texttt{0x4000}, \texttt{0x0}$ or $\vec{IV} = \texttt{0x0}, \texttt{0x4002}, \texttt{0x0}$. In other words, the 47-th IV bit must be assigned to constant 1. The summary of our new results about TRIVIUM is in Table 2. Note that the time complexity in this table shows the time complexity of Superpoly Recovery (Phase 1) and Partial Key Recovery (Phase 2).

5.2 Application to Kreyvium

Kreyvium is designed for the use of fully Homomorphic encryption. It claims 128-bit security and accepts 128-bit IVs. Kreyvium consists of 5 registers. Two of them are LFSRs denoted as K^* and IV^* respectively and the remaining 3 NFSRs share the same structure with TRIVIUM. The updating function of Kreyvium also resembles that of TRIVIUM and make calls to the core function f_{core} in Eq. (21). It also shares the same output function Eq. (19). Due to such similarities, the MILP model for describing the division property propagation for Kreyvium is quite similar with that of TRIVIUM. We just list our main theoretic results here and leave the other details in Supplementary Material 1.

Firstly, by running Algorithm 1 with parameter $\vec{IV} = (0, 0, 0, 0)$, we are able to prove the validity of the zero-sum distinguishers given in [25], [27]. On the contrary, for higher-dimensional cube I's, the degrees acquired by running Algorithm 1 with $\vec{IV} = (1, 1, \ldots, 1)$ are usually equal to those acquired with $\vec{IV} = \texttt{NULL}$. This is true for all the theoretic key recovery results on Kreyvium in this section. Such a phenomenon indicates that the "all-one" setting is a good choice for making non-constant superpolies.

In [25], the 61-dimensional cube I (first appeared in [27]) was used for attacking 849-round Kreyvium. It shows that the corresponding J is of size 23. So the complexity of superpoly recovery is $2^{61+23} = 2^{84}$. With the same I and by running Algorithm 1 (both all-one setting and $\vec{IV} = \texttt{NULL}$), we identify the same J. By running Algorithm 6, we find that the degree of the superpoly at round 849 is 9. Therefore, using our new techniques, we are able to lower the complexity to $2^{94.61}$ according to Eq. (16). They also proposed a 85-dimensional cube and mounted on 872 rounds ($|J| = 39$). By running Algorithm 6, we know the degree of the superpoly is only 2 and the complexity can be lowered from the original 2^{124} to $2^{94.61}$.

Now that the $|I| + |J| < n$ limitation has disappeared, we can construct larger cubes for attacking more rounds. Our best attack has mounted to 891 rounds using a 113-dimensional cube. The superpoly at round 891 has degree 2 and $|J| = 20$. So the complexity for attacking round 891 is $2^{120.73}$ according to Eq. (16).

Details of all our attacks are listed in Table 3.

5.3 Application to TriviA-SC1/2

TriviA-SC1 and TriviA-SC2 share the same updating function with TRIVIUM but with a larger state of 384 bits. The two primitives only differ in the initial state $\vec{s}^0$. Their output function is a linear combinations of 7 terms including a 2-degree term $s^R_{101}s^R_{197}$. Therefore, the Divide-and-Conquer strategy requires to construct 7 MILP models to determine J and d. For theoretic results, we are able to mount to 1009 rounds for TriviA-SC1 and 1004 rounds for TriviA-SC2 with complexities 2^{111} and $2^{111.6}$ respectively. Detailed parameters of the two attacks are listed in Table 4. Descriptions to the specifications and MILP modelings are available in Supplementary Materials 2.

6 Applications to Grain-128a and Acorn

Besides TRIVIUM-like stream ciphers, we also apply our flag technique and degree evaluation to other bit-oriented stream ciphers namely Grain-128a [41] and ACORN [42]. In comparison with the TRIVIUM-like primitives, Grain-128a and ACORN have more complicated output functions so the divide-and-conquer strategy may not help much to simplify the MILP models. On the other hand, the number of initialization rounds becomes lower making the standard MILP easy enough to solve. We only list our theoretic results here and leave detailed analysis in Supplementary Materials 3 and 4.

6.1 Theoretic Results on Grain-128a

Grain-128a has 128 key bits and 96 IV bits. We first revisit the previous attacks on 182- and 183-round Grain-128a using cube dimensions 88 and 92 respectively. For the 88-dimensional cube, we use our improved flag technique using Algorithm 1, we find that there are only $|J| = 14$ rather than 18 involved key bits in the superpoly. Using Algorithm 6, we prove the degree of the superpoly is 14, equal to $|J|$. So the complexity of this attack is improved by 2^4 to $2^{88+14} = 2^{102}$.

For the 92-dimensional cube, our new methods give the same J of size 16. The degree of its superpoly is 14. So the complexity is improved slightly from $2^{92+16} = 2^{108}$ to $2^{92} \times \binom{16}{\leq 14} = 2^{108} - 2^{96.08}$.

In order to attack 184 rounds, we are supposed to use up all IVs. We select $I_i = \{1, \ldots, 96\} \setminus \{i\}$ for $i = 1, \ldots, 96$. Then, we set $\vec{IV}[i] = 1$ and $\vec{IV}[i] = 0$. We run Algorithm 6 with I and the two different $\vec{IV}$'s. We find that when $\vec{IV}[47] = 1$, the degree of the superpoly is 19 and the degree drops to only 14 when $\vec{IV}[47] = 0$. This may indicate that many key bits are no longer involved when $\vec{IV}[47] = 0$. So we run Algorithm 1 with I_{47} and $\vec{IV}[47] = 0$. Under such a setting, we have $|J| = 21$ and the attack is available with complexity $2^{115.95}$. We summarize our attacks as Table 5. We have lowered the complexities of previous cubes and improved the maximum attacked rounds by 1.

6.2 Theoretic Results on Acorn

ACORN has 128 key bits and 128 IV bits. We first revisit the result in [15]. Using Algorithm 1, we find $|J| = 63$, 5 additional key bits are detected compared to [15] due to the flag technique. With the application of Algorithm 6, we deduce the degree of the superpoly as 7. So the complexity of this attack is $2^{93.23}$ according to Eq. (16), much lower than the previous 2^{122}. In order to attack more rounds, we construct a 101-dimensional cube that can mount to 750 rounds. It has $|J| = 81$ and the degree of the superpoly is 5. The complexity for attacking 750-round ACORN is therefore $2^{125.71}$ according to Eq. (16). The detailed parameters of these attacks are listed in Table 6.

7 Lower Complexity with Term Enumeration

In this section, we show how to further lower the complexity of recovering the superpoly (Phase 1) in Sect. 4.

With cube indices I, key bits J and degree d, the complexity of the current superpoly recovery is $2^I \times \binom{|J|}{\leq d}$, where $\binom{|J|}{\leq d}$ corresponds to all $0-$, $1 - \ldots$, $d-$degree monomials. When $d \leq |J|/2$ (which is true in most of our applications), we constantly have $\binom{|J|}{0} \leq \ldots \leq \binom{|J|}{d}$. But in practice, high-degree terms are generated in later iterations and the high-degree monomials should be fewer than their low-degree counterparts. Therefore, for all $\binom{|J|}{i}$ monomials, only very few of them may appear in the superpoly. Similar to Algorithm 1 that decides all key bits that appear in the superpoly, we propose Algorithm 9 that enumerates all t-degree monomials that may appear in the superpoly. Clearly, when we use $t = 1$, we can get $J_1 = J$, the same output as Algorithm 1 containing all involved key bits. If we use $t = 2, 3, \ldots, d$, we get $J_2, \ldots, J_d$ that contains all possible monomials of degrees $2, 3, \ldots, d$. Therefore, we only need to determine $1 + |J_1| + |J_2| + \ldots + |J_d|$ coefficients in order to recover the superpoly and clearly, $|J_t| \leq \binom{|J|}{t}$ for $t = 1, \ldots d$. With the knowledge of $J_t, t = 1, \ldots, d$, the complexity for Superpoly Recovery (Phase 1) has now become

$$2^{|I|} \times (1 + \sum_{t=1}^{d} |J_t|) \leq 2^{|I|} \times \binom{|J|}{\leq d}. \tag{24}$$

And the size of the lookup table has also reduced to $(1 + \sum_{t=1}^{d} |J_t|)$. So the complexity of the attack is now

$$\max\left\{2^{|I|} \times (1 + \sum_{t=1}^{d} |J_t|), 2^{|I|} + 2^{|J|} \times (1 + \sum_{t=1}^{d} |J_t|)\right\}. \tag{25}$$

Furthermore, since high-degree monomials are harder to be generated through iterations than low-degree ones, we can often find $|J_i| < \binom{|J|}{i}$ when i approaches d. So the complexity for superpoly recovery has been reduced.

TABLE 3
Summary of theoretical cube attacks on Kreyvium.

#Rounds	$\lvert I\rvert$	Degree	Involved secret variables J	Time complexity
849	61	9	47, 49, 51, 53, 55, 64, 66, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 89, 90, 91, 92, 93 ($\lvert J\rvert = 23$)	$2^{81.7}$
872	85	2	5, 6, 20, 21, 22, 30, 31, 37, 39, 40, 41, 49, 53, 54, 56, 57, 58, 63, 64, 65, 66, 67, 74, 75, 76, 89, 91, 92, 93, 96, 98, 99, 100, 108, 122, 123, 124, 125, 126 ($\lvert J\rvert = 39$)	$2^{94.61}$
891	113	2	19, 37, 46, 47, 62, 63, 64, 66, 71, 72, 73, 78, 91, 92, 93, 96, 106, 121, 122, 123 ($\lvert J\rvert = 20$)	$2^{120.73}$

TABLE 4
Summary of theoretical cube attacks on TriviA-SC1/2.

Target	#Rounds	$\lvert I\rvert$	Degree	Involved secret variables J	Time complexity
TriviA-SC1	1009	110	1	18 ($\lvert J\rvert = 1$)	2^{111}
TriviA-SC2	1004	110	1	19, 20 ($\lvert J\rvert = 2$)	$2^{111.6}$

TABLE 5
Summary of theoretical cube attacks on Grain-128a.

#Rounds	$\lvert I\rvert$	Degree	Involved secret variables J	Time complexity
182	88	14	36, 40, 51, 52, 53, 56, 61, 62, 69, 79, 81, 82, 122, 127 ($\lvert J\rvert = 14$)	2^{102}
183	92	14	48, 49, 50, 51, 52, 54, 55, 61, 63, 83, 84, 90, 93, 95, 120, 128 ($\lvert J\rvert = 16$)	$2^{108} - 2^{96.08}$
184	95	14	23, 34, 39, 48, 49, 53, 58, 59, 62, 64, 81, 83, 84, 95, 98, 118, 120, 123, 125, 127, 128 ($\lvert J\rvert = 21$)	$2^{115.95}$

TABLE 6
Summary of theoretical cube attacks on ACORN. The time complexity in this table shows the time complexity of Phase 1 and Phase 2.

# Rounds	$\lvert I\rvert$	Degree	Involved secret variables J	Time complexity
704	64	7	1,...,12, 14,...21, 23,...,38, 40,...44, 48, 49, 50, 54, 58, 60, 63, 64, 65, 68, 69, 71, 74, 75, 97, 102, 108 ($\lvert J\rvert = 63$)	$2^{93.23}$
750	101	5	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 46, 47, 48, 49, 50, 51, 52, 53, 54, 56, 57, 58, 59, 61, 62, 63, 64, 65, 66, 67, 68, 69, 71, 76, 77, 81, 83, 86, 87, 90, 91, 96, 98, 100, 101, 102, 120 ($\lvert J\rvert = 81$)	$2^{125.71}$

Note: J_t's ($t = 1, \dots, d$) can be generated by `TermEnum` of Algorithm 9 and they satisfy the following Property 1. This property is equivalent to the "Embed Property" given in [22].

Property 1. For $t = 2, \dots, d$, if there is $T = (i_1, i_2, \dots, i_t) \in J_t$ and $T' = (i_{s_1}, \dots, i_{s_\ell})$ ($\ell < t$) is a subsequence of T ($1 \le s_1 < \dots < s_\ell \le t$). Then, we constantly have $T' \in J_\ell$.

Before proving Property 1, we first prove the following Lemma 2.

Lemma 2. If $\vec{k} \succeq \vec{k}'$ and there is division trail $\vec{k} \xrightarrow{f} \vec{\ell}$, then there is also division trail $\vec{k}' \xrightarrow{f} \vec{\ell}'$ s.t. $\vec{\ell} \succeq \vec{\ell}'$.

Proof: Since f is a combination of COPY, AND and XOR operations, and the proofs when f equals to each of them are similar, we only give a proof of the case when f equals to COPY. Let $f : (*, \dots, *, x) \xrightarrow{COPY} (*, \dots, *, x, x)$.

First assume the input division property be $\vec{k} = (\vec{k}_1, 0)$, since $\vec{k} \succeq \vec{k}'$, there must be $\vec{k}' = (\vec{k}_1', 0)$ and $\vec{k}_1 \succeq \vec{k}_1'$. We have $l = k$, $l' = k'$, thus the property holds.

When the input division property is $\vec{k} = (\vec{k}_1, 1)$, we know that the output division property can be $\vec{\ell} \in \{(\vec{k}_1, 0, 1), (\vec{k}_1, 1, 0)\}$. Since $\vec{k} \succeq \vec{k}'$, we know $\vec{k}' = (\vec{k}_1', 1)$ or $\vec{k}' = (\vec{k}_1', 0)$, and $\vec{k}_1 \succeq \vec{k}_1'$. When $\vec{k}' = (\vec{k}_1', 0)$, then $\ell' = k' = (k_1', 0)$, the relation holds. When $\vec{k}' = (\vec{k}_1', 1)$, we know $\vec{\ell}' \in \{(\vec{k}_1', 0, 1), (\vec{k}_1', 1, 0)\}$, the relation still holds. □

Now we are ready to prove Property 1.

Proof: Let $\vec{k}, \vec{k} \in \mathbb{F}_2^n$ satisfy $k_i = 1$ for $i \in T$ and $k_i = 0$ otherwise; $k_i' = 1$ for $i \in T'$ and $k_i' = 0$ otherwise. Since $T \in J_t$, we know that there is division trail $(\vec{k}, \vec{k}_I) \xrightarrow{R-Rounds} (\vec{0}, 1)$ Since $k \succeq k'$, we have $(\vec{k}, \vec{k}_I) \succeq (\vec{k}', \vec{k}_I)$ and according to Lemma 2, there is division trail s.t. $(\vec{k}', \vec{k}_I) \xrightarrow{R-Rounds} (\vec{0}^{m+n}, s)$ where $(\vec{0}^{m+n}, 1) \succeq (\vec{0}^{m+n}, s)$. The hamming weight of $(\vec{k}', \vec{k}_I)$ is larger than 0 and there is an operation making non-zero division property to all-zero one. So we have $s = 1$ and there exist division trail $(\vec{k}', \vec{k}_I) \xrightarrow{R-Rounds} (\vec{0}, 1)$. □

Property 1 reveals a limitation of Algorithm 9. Assume the superpoly is

$$p_{\vec{v}}(x_1, x_2, x_3, x_4) = x_1x_2x_3 + x_1x_4.$$

We can acquire $J_3 = \{(1, 2, 3)\}$ by running `TermEnum` of Algorithm 9. But, if we run `TermEnum` with $t = 2$, we will not acquire just $J_2 = \{(1, 4)\}$ but $J_2 = \{(1, 4), (1, 2), (1, 3), (2, 3)\}$ due to $(1, 2, 3) \in J_3$ and $(1, 2)$, $(1, 3)$, $(2, 3)$ are its subsequences. Although there are still redundant terms, the reduction from $\binom{|J|}{d}$ to $|J_d|$ is usually huge enough to improve the existing cube attack results.

Applying such a term enumeration technique, we are able to lower complexities of many existing attacks namely:

Algorithm 9 Enumerate all the terms of degree t

```
1: procedure TermEnum(Cube indices $I$, specific assignment to non-cube IVs $\vec{IV}$ or $\vec{IV}$ = NULL, targeted degree $t$)
2:     Declare an empty MILP model $\mathcal{M}$ and an empty set $J_t = \phi \subseteq \{1,\dots,n\}^n$
3:     Declare $\vec{x}$ as $n$ MILP variables of $\mathcal{M}$ corresponding to secret variables.
4:     Declare $\vec{v}$ as $m$ MILP variables of $\mathcal{M}$ corresponding to public variables.
5:     $\mathcal{M}.con \leftarrow v_i = 1$ and assign $v_i.F = \delta$ for all $i \in I$
6:     $\mathcal{M}.con \leftarrow v_i = 0$ for all $i \in (\{1,2,\dots,n\} - I)$
7:     $\mathcal{M}.con \leftarrow \sum_{i=1}^n x_i = t$ and assign $x_i.F = \delta$ for all $i \in \{1,\dots,n\}$
8:     if $\vec{IV}$ = NULL then
9:         $v_i.F = \delta$ for all $i \in (\{1,2,\dots,n\} - I)$
10:    else
11:        Assign the flags of $v_i$, $i \in (\{1,2,\dots,n\} - I)$ as:
           $v_i.F = \begin{cases} 1_c & \text{if } \vec{IV}[i] = 1 \\ 0_c & \text{if } \vec{IV}[i] = 0 \end{cases}$
12:    end if
13:    Update $\mathcal{M}$ according to round functions and output functions
14:    do
15:        solve MILP model $\mathcal{M}$
16:        if $\mathcal{M}$ is feasible then
17:            pick index sequence $(j_1,\dots,j_t) \subseteq \{1,\dots,n\}^t$ s.t. $x_{j_1} = \dots = x_{j_t} = 1$
18:            $J_t = J_t \cup \{(j_1,\dots,j_t)\}$
19:            $\mathcal{M}.con \leftarrow \sum_{i=1}^t x_{j_i} \le t-1$
20:        end if
21:    while $\mathcal{M}$ is feasible
22:    return $J_t$
23: end procedure
```

```
1: procedure RTermEnum(Cube indices $I$, specific assignment to non-cube IVs $\vec{IV}$ or $\vec{IV}$ = NULL, targeted degree $t$)
2:     Declare an empty MILP model $\mathcal{M}$ and an empty set $JR_t = \phi \subseteq \{1,\dots,n\}$
3:     Declare $\vec{x}$ as $n$ MILP variables of $\mathcal{M}$ corresponding to secret variables.
4:     Declare $\vec{v}$ as $m$ MILP variables of $\mathcal{M}$ corresponding to public variables.
5:     $\mathcal{M}.con \leftarrow v_i = 1$ and assign $v_i.F = \delta$ for all $i \in I$
6:     $\mathcal{M}.con \leftarrow v_i = 0$ for all $i \in (\{1,2,\dots,n\} - I)$
7:     $\mathcal{M}.con \leftarrow \sum_{i=1}^n x_i \ge t$ and assign $x_i.F = \delta$ for all $i \in \{1,\dots,n\}$
8:     if $\vec{IV}$ = NULL then
9:         $v_i.F = \delta$ for all $i \in (\{1,2,\dots,n\} - I)$
10:    else
11:        Assign the flags of $v_i$, $i \in (\{1,2,\dots,n\} - I)$ as:
           $v_i.F = \begin{cases} 1_c & \text{if } \vec{IV}[i] = 1 \\ 0_c & \text{if } \vec{IV}[i] = 0 \end{cases}$
12:    end if
13:    Update $\mathcal{M}$ according to round functions and output functions
14:    do
15:        solve MILP model $\mathcal{M}$
16:        if $\mathcal{M}$ is feasible then
17:            pick index set $\{j_1,\dots,j_{t'}\} \subseteq \{1,\dots,n\}$ s.t. $t' \ge t$ and $x_{j_1} = \dots = x_{j_{t'}} = 1$
18:            $JR_t = JR_t \cup \{j_1,\dots,j_{t'}\}$
19:            $\mathcal{M}.con \leftarrow \sum_{i \notin JR_t} x_i \ge 1$
20:        end if
21:    while $\mathcal{M}$ is feasible
22:    return $JR_t$
23: end procedure
```

832-, 833-round TRIVIUM, 849-round Kreyvium, 184-round Grain-128a and 704-round ACORN. The attack on 750-round ACORN can also be improved using a relaxed version of `TermEnum` which is presented as `RTermEnum` on the righthand side of Algorithm 9. In the relaxed algorithm, `RTermEnum` is acquired from `TermEnum` by replacing some states which are marked in red in Algorithm 9, and we state details later in Sect. 7.2.

7.1 Application to TRIVIUM, Kreyvium, Grain-128a and 704-Round ACORN

As can be seen in Table 2, the attack on 832-round TRIVIUM has $J = J_1 = 5$ and degree $d = 3$, so we have $\binom{5}{\le 3} = 26$ using the previous technique. But by running Algorithm 9, we find that $|J_2| = 5$, $|J_3| = 1$, so we have

$$1 + \sum_{t=1}^{3} |J_t| = 12 < \binom{5}{\le 3} = 26.$$

Therefore, the complexity has now been lowered from $2^{76.7}$ to $2^{75.8}$. A similar technique can also be applied to the 73 dimensional cube of Table 2. Details are shown in Table 7.

The same procedure can also be carried out on Kreyvium, Grain-128a and 704-round ACORN. The details are also shown in Table 7.

7.2 Applications to 750-Round ACORN

For the attack on 704-round ACORN, we can use the previous precise term enumeration to reduce the complexity as shown in Table 7. For the attack on 750-round ACORN (the superpoly is of degree $d = 5$), the left half of Algorithm 9 can only be carried out for the 5-degree terms $|J_5| = 46$. For $t = 2, 3, 4$, the sizes of J_t are too large to be enumerated. So we introduce a relaxed term enumeration technique.

Relaxed Algorithm 9. We settle for the index set JR_t containing the key indices that compose all the t-degree terms. For example, when $J_3 = \{(1,2,3),(1,2,4)\}$, we have $JR_3 = \{1,2,3,4\}$. The relationship between J_t and JR_t is $|J_t| \le \binom{|JR_t|}{t}$ and $J_1 = JR_1$ (We denote $|JR_0| = 1$). The search space for J_t in Algorithm 9 is $\binom{|J_1|}{t}$ while that of the relaxed algorithm is only $\binom{|JR_t|}{t}$. So it is much easier to enumerate JR_t, therefore the complexity can still be improved (in comparison with Eq. (16)) as long as $|JR_t| < |J_1|$. The complexity of this relaxed version can be written as

$$\max\left\{2^{|I|} \times (\binom{|JR_t|}{\le d-1} + J_d), 2^{|I|} + 2^{|J|} \times (\binom{|JR_t|}{\le d-1} + J_d)\right\}. \quad (26)$$

For 750-round ACORN, we enumerate J_5 and $JR_1,\dots,JR_4$ whose sizes are listed in Table 8. The improved complexity, according to Eq. (26), is $2^{120.92}$, lower than the original $2^{125.71}$ given in Sect. 6.2.

8 CONCLUSION

In this paper, we improve the division property based cube attack method by further cultivating the algebraic properties of superpolies. Firstly, we developed the flag

TABLE 7
Results of TRIVIUM, Kreyvium, Grain-128a and 704-Round ACORN with Precise Term Enumeration

Cipher	#Rounds	$\|I\|$	$\|J_1\|$	$\|J_t\|, t = 2, \ldots, d$	$1 + \sum_{t=1}^{d} \|J_t\|$	Previous	Improved
TRIVIUM	832	72	5	5,1	$12 \approx 2^{3.58}$	$2^{76.7}$	$2^{75.58}$
TRIVIUM	833	73	7	6,1	$15 \approx 2^{3.91}$	2^{79}	$2^{76.91}$
Kreyvium	849	61	23	158, 555, 1162, 1518, 1235, 618, 156, 26	$5452 \approx 2^{12.41}$	$2^{81.7}$	$2^{73.41}$
Grain-128a	184	95	21	157, 651, 1765, 3394, 4838, 5231, 4326, 2627, 1288, 442, 104, 15, 1	$2^{14.61}$	$2^{115.95}$	$2^{109.61}$
ACORN	704	64	63	1598, 4911, 5755, 2556, 179, 3	$2^{13.88}$	$2^{93.23}$	$2^{77.88}$

TABLE 8
Results of 750-Round ACORN with Relaxed Term Enumeration

#Rounds	$\|I\|$	$\|JR_1\|$	$\|JR_2\|$	$\|JR_3\|$	$\|JR_4\|$	$\|J_5\|$	$1 + \sum_{t=1}^{d-1} \binom{\|JR_t\|}{t} + \|J_d\|$	Previous	Improved
750	101	81	81	77	70	46	$2^{19.92}$	$2^{125.71}$	$2^{120.92}$

technique for identifying proper non-cube IV assignments: we can identify proper non-cube IV assignments leading to a non-constant superpoly, rather than randomizing trials and error summations in the offline phase. We also propose a fast implementation for the flag technique to simplify the MILP models corresponding to the division property propagations. Then, we derived a division property based MILP model for upper bounding the algebraic degree of the superpoly. Such an improvement can break the $|I| + |J| < n$ barrier enabling us to construct larger cubes for attacking more rounds. Thirdly, for TRIVIUM-like primitives, namely TRIVIUM, Kreyvium, TriviA-SC1/2, we give the divide-and-conquer modeling strategy that largely improves the efficiency of division property deductions. Finally, we propose the accurate & relaxed term enumeration techniques that can further reduce the complexities. The combination of these new techniques gives the current best key recovery attacks on several stream ciphers namely Kreyvium, TriviA-SC1/2, Grain-128a, and ACORN.

ACKNOWLEDGMENTS

We would like to thank Christian Rechberger, Elmar Tischhauser, Lorenzo Grassi and Liang Zhong for their fruitful discussions, and the anonymous reviewers for their valuable comments. This work is supported by University of Luxembourg project - FDISC, National Key Research and Development Program of China (No. 2018YFA0306404), National Natural Science Foundation of China (No. 61472250, No. 61672347, No. 61802021), Program of Shanghai Academic/Technology Research Leader (No. 16XD1401300), Opening Project of Guangdong Provincial Key Laboratory of Data Security and Privacy Protection (No. 2017B030301004-18), the Research Council KU Leuven: C16/15/058, OT/13/071, the Flemish Government through FWO projects and by European Union's Horizon 2020 research and innovation programme under grant agreement No H2020-MSCA-ITN-2014-643161 ECRYPT-NET.

REFERENCES

[1] Q. Wang, Y. Hao, Y. Todo, C. Li, T. Isobe, and W. Meier, "Improved division property based cube attacks exploiting algebraic properties of superpoly," in *Advances in Cryptology – CRYPTO 2018*, H. Shacham and A. Boldyreva, Eds. Cham: Springer International Publishing, 2018, pp. 275–305.

[2] I. Dinur and A. Shamir, "Cube attacks on tweakable black box polynomials," in *EUROCRYPT 2009*, ser. LNCS, A. Joux, Ed., vol. 5479. Springer, 2009, pp. 278–299.

[3] J. Aumasson, I. Dinur, W. Meier, and A. Shamir, "Cube testers and key recovery attacks on reduced-round MD6 and Trivium," in *FSE 2009*, ser. LNCS, O. Dunkelman, Ed., vol. 5665. Springer, 2009, pp. 1–22.

[4] I. Dinur and A. Shamir, "Breaking Grain-128 with dynamic cube attacks," in *FSE 2011*, ser. LNCS, A. Joux, Ed., vol. 6733. Springer, 2011, pp. 167–187.

[5] P. Fouque and T. Vannet, "Improving key recovery to 784 and 799 rounds of Trivium using optimized cube attacks," in *FSE 2013*, ser. LNCS, S. Moriai, Ed., vol. 8424. Springer, 2013, pp. 502–517.

[6] M. I. Salam, H. Bartlett, E. Dawson, J. Pieprzyk, L. Simpson, and K. K. Wong, "Investigating cube attacks on the authenticated encryption stream cipher ACORN," in *ATIS 2016*, ser. CCIS, L. Batten and G. Li, Eds., vol. 651. Springer, 2016, pp. 15–26.

[7] M. Liu, J. Yang, W. Wang, and D. Lin, "Correlation Cube Attacks: From Weak-Key Distinguisher to Key Recovery," in *EUROCRYPT 2018 Part II*, ser. Lecture Notes in Computer Science, J. B. Nielsen and V. Rijmen, Eds., vol. 10821. Springer, 2018, pp. 715–744.

[8] Y. Sun, "Cube attack on round-reduced Fruit," *Journal of Cryptologic Research*, vol. 4, no. 6, p. 528, 2017. [Online]. Available: http://www.jcr.cacrnet.org.cn:8080/mmxb/CN/abstract/abstract223.shtml

[9] Y. Ren, Y. Sun, and Y. Wang, "A space-time tradeoff cube attack on Grain-v1," *Journal of Cryptologic Research*, vol. 2, no. 3, p. 235, 2015. [Online]. Available: http://www.jcr.cacrnet.org.cn:8080/mmxb/CN/abstract/abstract88.shtml

[10] I. Dinur, P. Morawiecki, J. Pieprzyk, M. Srebrny, and M. Straus, "Cube attacks and cube-attack-like cryptanalysis on the round-reduced Keccak sponge function," in *EUROCRYPT 2015 Part I*, ser. LNCS, E. Oswald and M. Fischlin, Eds., vol. 9056. Springer, 2015, pp. 733–761.

[11] S. Huang, X. Wang, G. Xu, M. Wang, and J. Zhao, "Conditional Cube Attack on Reduced-Round Keccak Sponge Function," in *EUROCRYPT 2017 Part II*, ser. LNCS, J. Coron and J. B. Nielsen, Eds., vol. 10211. Springer, 2017, pp. 259–288.

[12] Z. Li, W. Bi, X. Dong, and X. Wang, "Improved conditional cube attacks on Keccak keyed modes with MILP method," in *ASIACRYPT 2017 Part I*, ser. LNCS, T. Takagi and T. Peyrin, Eds., vol. 10624. Springer, 2017, pp. 99–127.

[13] Z. Li, X. Dong, and X. Wang, "Conditional cube attack on round-reduced ASCON," *IACR Trans. Symmetric Cryptol.*, vol. 2017, no. 1, pp. 175–202, 2017.

[14] X. Dong, Z. Li, X. Wang, and L. Qin, "Cube-like attack on round-reduced initialization of Ketje Sr," *IACR Trans. Symmetric Cryptol.*, vol. 2017, no. 1, pp. 259–280, 2017.

[15] Y. Todo, T. Isobe, Y. Hao, and W. Meier, "Cube attacks on non-blackbox polynomials based on division property," in *CRYPTO 2017 Part III*, ser. LNCS, J. Katz and H. Shacham, Eds., vol. 10403. Springer, 2017, pp. 250–279.

[16] Y. Todo, "Structural evaluation by generalized integral property," in *EUROCRYPT 2015 Part I*, ser. LNCS, E. Oswald and M. Fischlin, Eds., vol. 9056. Springer, 2015, pp. 287–314.

[17] ——, "Integral cryptanalysis on full MISTY1," in *CRYPTO 2015*

Part I, ser. LNCS, R. Gennaro and M. Robshaw, Eds., vol. 9215. Springer, 2015, pp. 413–432.

[18] Y. Todo and M. Morii, "Bit-based division property and application to SIMON family," in *FSE 2016*, ser. LNCS, T. Peyrin, Ed., vol. 9783. Springer, 2016, pp. 357–377.

[19] Z. Xiang, W. Zhang, Z. Bao, and D. Lin, "Applying MILP method to searching integral distinguishers based on division property for 6 lightweight block ciphers," in *ASIACRYPT 2016 Part I*, ser. LNCS, J. H. Cheon and T. Takagi, Eds., vol. 10031. Springer, 2016, pp. 648–678.

[20] Z. Gu, E. Rothberg, and R. Bixby, "Gurobi optimizer," http://www.gurobi.com/.

[21] L. Sun, W. Wang, and M. Wang, "MILP-Aided Bit-Based Division Property for Primitives with Non-Bit-Permutation Linear Layers," Cryptology ePrint Archive, Report 2016/811, 2016, https://eprint.iacr.org/2016/811.

[22] ——, "Automatic search of bit-based division property for ARX ciphers and word-based division property," in *ASIACRYPT 2017 Part I*, ser. LNCS, T. Takagi and T. Peyrin, Eds., vol. 10624. Springer, 2017, pp. 128–157.

[23] Y. Funabiki, Y. Todo, T. Isobe, and M. Morii, "Improved integral attack on HIGHT," in *ACISP 2017 Part I*, ser. LNCS, J. Pieprzyk and S. Suriadi, Eds., vol. 10342. Springer, 2017, pp. 363–383.

[24] Q. Wang, L. Grassi, and C. Rechberger, "Zero-sum partitions of PHOTON permutations," in *CT-RSA 2018*, ser. LNCS, N. Smart, Ed., vol. 10808. Springer, 2018.

[25] Y. Todo, T. Isobe, Y. Hao, and W. Meier, "Cube attacks on non-blackbox polynomials based on division property," *IEEE Transactions on Computers*, p. 1. [Online]. Available: doi.ieeecomputersociety.org/10.1109/TC.2018.2835480

[26] ——, "Cube attacks on non-blackbox polynomials based on division property (full version)," Cryptology ePrint Archive, Report 2017/306, 2017, https://eprint.iacr.org/2017/306.

[27] M. Liu, "Degree evaluation of NFSR-based cryptosystems," in *CRYPTO 2017 Part III*, ser. LNCS, J. Katz and H. Shacham, Eds., vol. 10403. Springer, 2017, pp. 227–249.

[28] X. Fu, X. Wang, X. Dong, and W. Meier, "A key-recovery attack on 855-round Trivium," Cryptology ePrint Archive, Report 2018/198, 2018, https://eprint.iacr.org/2018/198.

[29] Y. Hao, L. Jiao, C. Li, W. Meier, Y. Todo, and Q. Wang, "Observations on the dynamic cube attack of 855-round TRIVIUM from Crypto'18," Cryptology ePrint Archive, Report 2018/972, 2018, https://eprint.iacr.org/2018/972.

[30] X. Fu, X. Wang, X. Dong, W. Meier, Y. Hao, and B. Zhao, "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.

[31] Y. Todo, T. Isobe, W. Meier, K. Aoki, and B. Zhang, "Fast correlation attack revisited–cryptanalysis on full Grain-128a, Grain-128, and Grain-v1," CRYPTO 2018, 2018, (accepted).

[32] M. Lehmann and W. Meier, "Conditional differential cryptanalysis of Grain-128a," in *CANS 2012*, ser. LNCS, J. Pieprzyk, A. Sadeghi, and M. Manulis, Eds., vol. 7712. Springer, 2012, pp. 1–11.

[33] N. Mouha, Q. Wang, D. Gu, and B. Preneel, "Differential and linear cryptanalysis using mixed-integer linear programming," in *Inscrypt 2011*, ser. LNCS, C. Wu, M. Yung, and D. Lin, Eds., vol. 7537. Springer, 2011, pp. 57–76.

[34] S. Sun, L. Hu, P. Wang, K. Qiao, X. Ma, and L. Song, "Automatic security evaluation and (related-key) differential characteristic search: Application to SIMON, PRESENT, LBlock, DES(L) and other bit-oriented block ciphers," in *ASIACRYPT 2014 Part I*, ser. LNCS, P. Sarkar and T. Iwata, Eds., vol. 8873. Springer, 2014, pp. 158–178.

[35] S. Sun, L. Hu, M. Wang, P. Wang, K. Qiao, X. Ma, D. Shi, L. Song, and K. Fu, "Towards finding the best characteristics of some bit-oriented block ciphers and automatic enumeration of (related-key) differential and linear characteristics with predefined properties," Cryptology ePrint Archive, Report 2014/747, 2014, https://eprint.iacr.org/2014/747.

[36] T. Cui, K. Jia, K. Fu, S. Chen, and M. Wang, "New automatic search tool for impossible differentials and zero-correlation linear approximations," Cryptology ePrint Archive, Report 2016/689, 2016, https://eprint.iacr.org/2016/689.

[37] Y. Sasaki and Y. Todo, "New impossible differential search tool from design and cryptanalysis aspects - revealing structural properties of several ciphers," in *EUROCRYPT 2017 Part III*, ser. LNCS, J. Coron and J. B. Nielsen, Eds., vol. 10212. Springer, 2017, pp. 185–215.

[38] C. D. Cannière and B. Preneel, "TRIVIUM specifications," 2006, eSTREAM portfolio, Profile 2 (HW). [Online]. Available: http://www.ecrypt.eu.org/stream/p3ciphers/trivium/trivium_p3.pdf

[39] A. Canteaut, S. Carpov, C. Fontaine, T. Lepoint, M. Naya-Plasencia, P. Paillier, and R. Sirdey, "Stream ciphers: A practical solution for efficient homomorphic-ciphertext compression," in *FSE 2016*, ser. LNCS, T. Peyrin, Ed., vol. 9783. Springer, 2016, pp. 313–333.

[40] A. Chakraborti, A. Chattopadhyay, M. Hassan, and M. Nandi, "TriviA: A fast and secure authenticated encryption scheme," in *International Workshop on Cryptographic Hardware and Embedded Systems*, 2015, pp. 330–353.

[41] M. Ågren, M. Hell, T. Johansson, and W. Meier, "Grain-128a: a new version of grain-128 with optional authentication," *IJWMC*, vol. 5, no. 1, pp. 48–59, 2011.

[42] H. Wu, "Acorn v3," 2016, submission to CAESAR competition.

Yonglin Hao received B.S. degree from Shandong University in 2012 and his PhD degree from Tsinghua University in 2017. He is now working in the State Key Laboratory of Cryptology, Beijing, China. His research interests include cryptanalysis of symmetric ciphers.

Takanori Isobe received the B.E., M.E., and Ph.D. degrees from Kobe University, Japan, in 2006, 2008, and 2013, respectively. He worked in the Sony Corporation from 2008 to 2017. He is currently an Associate Professor in University of Hyogo. His current research interests include information security and cryptography. He also received the Best Paper Award from IEICE in 2015. He also received the FSE 2011 Best Paper Award from IACR.

Lin Jiao received B.S. degree from Jilin University in 2010 and her PhD degree from Institute of Software, Chinese Academy of Sciences in 2016. She is now working in the State Key Laboratory of Cryptology, Beijing, China. Her research interests include cryptanalysis of symmetric ciphers.

Chaoyun Li is currently a PhD candidate at imec-COSIC in Department of Electrical Engineering(ESAT), KU Leuven. He received his B.S. and M.S. degree from Hubei University in 2012 and 2015 respectively. His research interests include the design and analysis of symmetric ciphers.

Willi Meier obtained his diploma in mathematics and the doctoral degree in mathematics from the Swiss Federal Institute of Technology Zurich (ETHZ), Zurich, Switzerland, in 1972 and 1975, respectively. He has been a Guest Researcher with the Oxford Universiy and Heidelberg University and a Research Assistant with University Siegen, Germany. Since 1985, he has been a Professor of mathematics and computer science with the University of Applied Sciences, Windisch, Northwestern Switzerland. His current interests include analysis and design of cryptographic primitives like stream ciphers and hash functions.

Yosuke Todo received B.E. and M.E. degrees from Kobe University in 2010 and 2012. Since 2012, he has been a researcher at NTT Secure Platform Laboratories. He received a Ph.D from Kobe University in 2017. He received the best papers from SCIS 2015, IEICE, and the best paper and best young researcher awards from CRYPTO 2015, IACR. His current research interest is cryptography.

Qingju Wang obtained M.S degree from Central South University China and has been an Assistant Lecturer in Shaoxing University. She received a joint PhD degree from the KU Leuven Belgium and the Shanghai Jiao Tong University China in 2016 and joined the Technical University of Denmark as a postdoc researcher. Since 2018, she has been working as a Research Associate at the University of Luxembourg. Her main research interests are design and cryptanalysis of cryptographic primitives.