Improved Division Property Based Cube Attacks Exploiting Algebraic Properties of Superpoly (Full Version)

Qingju Wang[1,2,3], Yonglin Hao[4⋆], Yosuke Todo[5⋆], Chaoyun Li[6⋆], Takanori Isobe[7], and Willi Meier[8]

[1] Shanghai Jiao Tong Uninversity, China
[2] Technical University of Denmark
[3] SnT, University of Luxembourg
[4] State Key Laboratory of Cryptology, Beijing, China
[5] NTT Secure Platform Laboratories, Japan
[6] imec-COSIC, Dept. Electrical Engineering (ESAT), KU Leuven, Belgium
[7] University of Hyogo, Japan
[8] FHNW, Switzerland

qingju.wang@uni.lu,haoyonglin@yeah.net,todo.yosuke@lab.ntt.co.jp
chaoyun.li@esat.kuleuven.be,takanori.isobe@ai.u-hyogo.ac.jp,willi.meier@fhnw.ch

Abstract. The cube attack is an important technique for the cryptanalysis of symmetric key primitives, especially for stream ciphers. Aiming at recovering some secret key bits, the adversary reconstructs a superpoly with the secret key bits involved, by summing over a set of the plaintexts/IV which is called a cube. Traditional cube attack only exploits linear/quadratic superpolies. Moreover, for a long time after its proposal, the size of the cubes has been largely confined to an experimental range, e.g., typically 40. These limits were first overcome by the division property based cube attacks proposed by Todo et al. at CRYPTO 2017. Based on MILP modelled division property, for a cube (index set) I, they identify the small (index) subset J of the secret key bits involved in the resultant superpoly. During the precomputation phase which dominates the complexity of the cube attacks, $2^{|I|+|J|}$ encryptions are required to recover the superpoly. Therefore, their attacks can only be available when the restriction $|I|+|J|<n$ is met.

In this paper, we introduced several techniques to improve the division property based cube attacks by exploiting various algebraic properties of the superpoly.

1. We propose the "flag" technique to enhance the preciseness of MILP models so that the proper non-cube IV assignments can be identified to obtain a non-constant superpoly.
2. A degree evaluation algorithm is presented to upper bound the degree of the superpoly. With the knowledge of its degree, the superpoly can be recovered without constructing its whole truth table. This enables us to explore larger cubes I's even if $|I|+|J|\geq n$.

⋆ Corresponding authors.

3. We provide a term enumeration algorithm for finding the monomials of the superpoly, so that the complexity of many attacks can be further reduced.

As an illustration, we apply our techniques to attack the initialization of several ciphers. To be specific, our key recovery attacks have mounted to 839-round Trivium, 891-round Kreyvium, 184-round Grain-128a and 750-round Acorn respectively.

Keywords: Cube attack, Division Property, MILP, Trivium, Kreyvium, Grain-128a, Acorn, Clique

1 Introduction

Cube attack, proposed by Dinur and Shamir [1] 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 ciphers, including stream ciphers [2,3,4,5,6,7,8], hash functions [9,10,11], and authenticated encryptions [12,13]. For a cipher with n secret variables $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)$ and m public variables $\boldsymbol{v} = (v_1, v_2, \ldots, v_m)$, we can regard the algebraic normal form (ANF) of output bits as a polynomial of $\boldsymbol{x}$ and $\boldsymbol{v}$, denoted as $f(\boldsymbol{x}, \boldsymbol{v})$. For a randomly chosen set $I = \{i_1, i_2, ..., i_{|I|}\} \subset \{1, \ldots, m\}$, $f(\boldsymbol{x}, \boldsymbol{v})$ can be represented uniquely as

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

where $t_I = v_{i_1} \cdots v_{i_{|I|}}$, $p(\boldsymbol{x}, \boldsymbol{v})$ only relates to v_s's ($s \notin I$) and the secret key bits $\boldsymbol{x}$, and $q(\boldsymbol{x}, \boldsymbol{v})$ misses at least one variable in t_I. When v_s's ($s \notin I$) and $\boldsymbol{x}$ are assigned statically, the value of $p(\boldsymbol{x}, \boldsymbol{v})$ can be computed by summing the output bit $f(\boldsymbol{x}, \boldsymbol{v})$ over a structure called *cube*, denoted as C_I, consisting of $2^{|I|}$ different $\boldsymbol{v}$ 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 mainly concerned about linear or quadratic superpolies. By collecting linear or quadratic equations from the superpoly, the attacker can recover some secret key bits information during the online phase. Aiming to mount distinguishing attack by property testing, cube testers are obtained by evaluating superpolies of carefully selected cubes. In [2], probabilistic tests are applied to detect some algebraic properties such as constantness, low degree and sparse monomial distribution. Moreover, cube attacks and cube testers are acquired experimentally by summing over randomly chosen cubes. So the sizes of the cubes are largely confined. Breakthroughs have been made by Todo et al. in [14] where they introduce the bit-based division property, a tool for conducting integral attacks[1], to the realm of cube attack. With the help of mixed integer linear programming (MILP) aided division property, they can identify the variables excluded from the superpoly and explore cubes with larger size,

[1] Integral attacks also require to traverse some active plaintext bits and check whether the summation of the corresponding ciphertext bits have zero-sum property, which equals to check whether the superpoly has $p(\boldsymbol{x}, \boldsymbol{v}) \equiv 0$.

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 [15]. 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 [16].

The original division property can only be applied to word-oriented primitives. At FSE 2016, bit-based division property [17] 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. Therefore, for the first time, integral characteristics for bit-based block ciphers SIMON32 and Simeck32 have been proved. However, the sizes of the 0-1 vector sets are exponential to the block size of the ciphers. Therefore, as has been pointed out by the authors themselves, the deduction of bit-based division property under their framework requires high memory for block ciphers with larger block sizes, which largely limits its applications. Such a problem has been solved by Xiang et al. [18] at ASIACRYPT 2016 by utilizing the MILP model. The operations on 0-1 vector sets are transformed to imposing division property values (0 or 1) to MILP variables, and the corresponding integral characteristics are acquired by solving the models with MILP solvers like Gurobi [19]. With this method, they are able to give integral characteristics for block ciphers with block sizes much larger than 32 bits. Xiang et al.'s method has now been applied to many other ciphers for improved integral attacks [20,21,22,23].

In [14], 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. $p(\boldsymbol{x}, \boldsymbol{v})$ 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 $\boldsymbol{IV} \in \mathbb{F}_2^m$, is determined ensuring $p(\boldsymbol{x}, \boldsymbol{IV})$ non-constant. Also in this phase, the whole truth table of $p(\boldsymbol{x}, \boldsymbol{IV})$ is constructed through cube summations. In the *online phase*, the exact value of $p(\boldsymbol{x}, \boldsymbol{IV})$ 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 keys are filtered as long as $p(\boldsymbol{x}, \boldsymbol{IV})$ 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 [14,24]. It is not until recently that the result on TRIVIUM has been improved by Liu et al. [6] 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 [25] originally used for constructing zero-sum distinguishers.

1.1 Motivations.

Due to [14,24], the power of cube attacks has been enhanced significantly, however, there are still problems remaining unhandled that we will reveal explicitly.

Finding proper $\boldsymbol{IV}$'s may require multiple trials. As is mentioned above, the superpoly can filter wrong keys only if a *proper IV assignment* $\boldsymbol{IV} \in \mathbb{F}_2^m$ in the constant part of IVs is found such that the corresponding superpoly $p(\boldsymbol{x}, \boldsymbol{IV})$ is non-constant. The MILP model in [14,24] only proves the existence of the proper $\boldsymbol{IV}$'s but finding them may not be easy. According to practical experiments, there are quite some $\boldsymbol{IV}$'s making $p(\boldsymbol{x}, \boldsymbol{IV}) \equiv 0$. Therefore, $t \geq 1$ different $\boldsymbol{IV}$'s might be trailed in the precomputation phase before finding a proper one. Since each $\boldsymbol{IV}$ requires to construct a truth table with complexity $2^{|I|+|J|}$, the overall complexity of the offline phase can be $t \times 2^{|I|+|J|}$. When large cubes are used ($|I|$ is big) or many key bits are involved ($|J|$ is large), such a complexity might be at the risk of exceeding the brute-force bound 2^n. Therefore, two assumptions are made to validate their cube attacks as follows.

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.*

These assumptions are proposed to guarantee the validity of the attacks as long as $|I| + |J| < n$, but the rationality of such assumptions is hard to be proved, especially when $|I| + |J|$ are so close to n in many cases. The best solution is to evaluate different IVs in the MILP model so that the proper $\boldsymbol{IV}$ of the constant part of IVs and the set J are determined simultaneously before implementing the attack.

Restriction of $|I| + |J| < n$. The superpoly recovery has always been dominating the complexity of the cube attack, especially in [14], the attacker knows no more information except which secret key bits are involved in the superpoly. Then she/he has to first construct the whole truth table for the superpoly in the offline phase. In general, the truth-table construction requires repeating the cube summation $2^{|J|}$ times, and makes the complexity of the offline phase about $2^{|I|+|J|}$. Apparently, such an attack can only be meaningful if $|I|+|J| < n$, where n is the number of secret variables. The restriction of $|I|+|J| < n$ barricades the adversary from exploiting cubes of larger dimension or mounting more rounds (where $|J|$ may expand). This restriction can be removed if we can avoid the truth table construction in the offline phase.

1.2 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 [14], 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 [14], 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 previous MILP model in [14] has not taken the effect of constant 0/1 bits of the constant part of IVs into account. In their model, the active bits are initialized with division property value 1 and other non-active bits are all initialized to division property value 0. The non-active bits include constant part of IVs, together with some secret key bits and state bits that are assigned statically to 0/1 according to the specification of ciphers. It has been noticed in [24] 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 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, we are able to find proper assignments to constant IVs ($\boldsymbol{IV}$) that makes the corresponding superpoly ($p(\boldsymbol{x}, \boldsymbol{IV})$) non-constant. With this technique, proper IVs can now be found with MILP model rather than time-consuming trial & summations in the offline phase as in [14,24]. 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 availability of the two assumptions since we are able to find proper $\boldsymbol{IV}$'s in all our attacks.

Degree evaluation for going beyond the $|I| + |J| < n$ restriction. To avoid constructing the whole truth table using cube summations, 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 $\binom{|J|}{\leq d}$ coefficients rather than the whole truth table, where $\binom{|J|}{\leq d}$ is defined as

$$\binom{|J|}{\leq d} := \sum_{i=0}^{d} \binom{|J|}{i}. \tag{1}$$

When $d = |J|$, the complexity by our new method and that by [14] 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 $2^{|I|} \times \binom{|J|}{\leq d}$. 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 $|I| + |J| \geq n$.

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. Alternately, 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 [6,14,24]. 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 [27], 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, 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.

Clique view of the superpoly recovery. In order to lower the complexity of the superpoly recovery, the term enumeration technique has to execute many MILP instances, which is difficult for some applications. We represent the resultant superpoly as a graph, so that we can utilize the clique concept from the graph theory to upper bound the complexity of the superpoly recovery phase, without requiring MILP solver as highly as the term enumeration technique.

Organizations. 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. Combining the flag technique and degree evaluation, we give improved key recovery cube attacks on 4 targeted ciphers in Sect. 5. The precise & relaxed term enumeration as well as their ap-

[2] While this paper was under submission, Fu et al. released a paper on ePrint [26] and claimed that 855 rounds initialization of TRIVIUM can be attacked.

plications are given in Sect. 6. We revisit the term enumeration technique from the clique overview in Sect. 7. Finally, we conclude in Sect. 8.

Table 1. Summary of our cube attack results

Applications	#Full Rounds	#Rounds	Cube size	$\|J\|$	Complexity	Reference
Trivium	1152	799	32 †	-	practical	[4]
		832	72	5	2^{77}	[14,24]
		833	**73**	**7**	$\mathbf{2^{76.91}}$	**Sect. 6.1**
		835	37/36∗	-	2^{75}	[6]
		836	**78**	**1**	$\mathbf{2^{79}}$	**Sect. 5.1**
		839	**78**	**1**	$\mathbf{2^{79}}$	**Sect. 5.1**
Kreyvium	1152	849	61	23	2^{84}	[24]
		849	**61**	**23**	$\mathbf{2^{81.7}}$	**App. A**
		849	**61**	**23**	$\mathbf{2^{73.41}}$	**Sect. 6.2**
		872	85	39	2^{124}	[24]
		872	**85**	**39**	$\mathbf{2^{94.61}}$	**App. A**
		891	**113**	**20**	$\mathbf{2^{120.73}}$	**App. A**
Grain-128a	256	177	33	-	practical	[28]
		182	88	18	2^{106}	[14,24]
		182	**88**	**14**	$\mathbf{2^{102}}$	**App. B**
		183	92	16	2^{108}	[14,24]
		183	**92**	**16**	$\mathbf{2^{108} - 2^{96.08}}$	**App. B**
		184	**95**	**21**	$\mathbf{2^{109.61}}$	**Sect. 6.3**
Acorn	1792	503	5 ‡	-	practical ‡	[5]
		704	64	58	2^{122}	[14,24]
		704	**64**	**63**	$\mathbf{2^{77.88}}$	**Sect. 6.4**
		750	**101**	**81**	$\mathbf{2^{125.71}}$	**App. C**
		750	**101**	**81**	$\mathbf{2^{120.92}}$	**Sect. 6.4**

† 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 [5]. 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 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 [19]. 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 [29], and has been widely used for searching characteristics corresponding to various methods such as differential [30,31], linear [31], impossible differential [32,33], zero-correlation linear [32], and integral characteristics with division property [18]. We will detail the MILP model of [18] later in this section.

2.2 Cube Attack

Considering a stream cipher with n secret key bits $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)$ and m public initialization vector (IV) bits $\boldsymbol{v} = (v_1, v_2, \ldots, v_m)$. Then, the first output keystream bit can be regarded as a polynomial of $\boldsymbol{x}$ and $\boldsymbol{v}$ referred as $f(\boldsymbol{x}, \boldsymbol{v})$. 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 $f(\boldsymbol{x}, \boldsymbol{v})$ can be uniquely decomposed as

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

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

$$C_I(\boldsymbol{IV}) := \{\boldsymbol{v} \in \mathbb{F}_2^m : \boldsymbol{v}[i] = 0/1, i \in I \bigwedge \boldsymbol{v}[s] = \boldsymbol{IV}[s], s \notin I\}. \tag{2}$$

It has been proved by Dinur and Shamir [1] that the value of superpoly p corresponding to the key $\boldsymbol{x}$ and the non-cube IV assignment $\boldsymbol{IV}$ can be computed by summing over the cube $C_I(\boldsymbol{IV})$ as follows:

$$p(\boldsymbol{x}, \boldsymbol{IV}) = \bigoplus_{\boldsymbol{v} \in C_I(\boldsymbol{IV})} f(\boldsymbol{x}, \boldsymbol{v}). \tag{3}$$

In the remainder of this paper, we refer to the value of the superpoly corresponding to the assignment $\boldsymbol{IV}$ in Eq. (3) as $p_{\boldsymbol{IV}}(\boldsymbol{x})$ for short. We use C_I as the cube corresponding to arbitrary $\boldsymbol{IV}$ setting in Eq. (2). 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 irrelevant to cube IVs $v_i, i \in I$, the value of $\boldsymbol{IV}[i], i \in I$ cannot affect the result of the summation in Eq. (3) at all. Therefore in Sect. 5, our $\boldsymbol{IV}[i]$'s ($i \in I$) are just assigned randomly to 0-1 values.

2.3 Bit-Based Division Property and its MILP Representation

At 2015, the division property, a generalization of the integral property, was proposed in [15] with which better integral characteristics for word-oriented cryptographic primitives have been detected. Later, the bit-based division property

was introduced in [17] 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:*

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

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

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 propagation corresponding rules `copy`, `xor`, `and` proved in [15,17]. 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 $\mathcal{D}_{\boldsymbol{k}}^{1^n}$, where $k_i = 1$ if $i \in I$ and $k_i = 0$ otherwise. Then, the propagation of the division property from $\mathcal{D}_{\boldsymbol{k}}^{1^n}$ is evaluated as

$$\{\boldsymbol{k}\} := \mathbb{K}_0 \to \mathbb{K}_1 \to \mathbb{K}_2 \to \cdots \to \mathbb{K}_r,$$

where $\mathcal{D}_{\mathbb{K}_i}$ is the division property after i-round propagation. If the division property $\mathbb{K}_r$ does not have an unit vector $\boldsymbol{e}_i$ 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 [17]. 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 [18]). *Let us consider the propagation of the division property $\{\boldsymbol{k}\} \stackrel{\text{def}}{=} \mathbb{K}_0 \to \mathbb{K}_1 \to \mathbb{K}_2 \to \cdots \to \mathbb{K}_r$. Moreover, for any vector $\boldsymbol{k}_{i+1}^* \in \mathbb{K}_{i+1}$, there must exist a vector $\boldsymbol{k}_i^* \in \mathbb{K}_i$ such that $\boldsymbol{k}_i^*$ can propagate to $\boldsymbol{k}_{i+1}^*$ by the propagation rule of the division property. Furthermore, for $(\boldsymbol{k}_0, \boldsymbol{k}_1, \ldots, \boldsymbol{k}_r) \in (\mathbb{K}_0 \times \mathbb{K}_1 \times \cdots \times \mathbb{K}_r)$ if $\boldsymbol{k}_i$ can propagate to $\boldsymbol{k}_{i+1}$ for all $i \in \{0, 1, \ldots, r-1\}$, we call $(\boldsymbol{k}_0 \to \boldsymbol{k}_1 \to \cdots \to \boldsymbol{k}_r)$ an r-round division trail.*

Let E_k be the target r-round iterated cipher. Then, if there is a division trail $\boldsymbol{k}_0 \xrightarrow{E_k} \boldsymbol{k}_r = \boldsymbol{e}_j$ $(j = 1, ..., n)$, the summation of jth bit of the ciphertexts is unknown; otherwise, if there is no division trial s.t. $\boldsymbol{k}_0 \xrightarrow{E_k} \boldsymbol{k}_r = \boldsymbol{e}_j$, 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 trials 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 $\mathcal{M}$. After Xiang et al.'s work, some simplifications have been made to the MILP descriptions of `copy`, `xor`, `and` in [20,14]. We present the current simplest MILP-based `copy`, `xor`, `and` as follows:

Proposition 1 (MILP Model for COPY [20]). *Let $a \xrightarrow{COPY} (b_1, b_2, \ldots, b_m)$ be a division trail of COPY. The following inequalities are sufficient to describe the propagation of the division property for* `copy`*.*

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

Proposition 2 (MILP Model for XOR [20]). *Let $(a_1, a_2, \ldots, a_m) \xrightarrow{XOR} b$ be a division trail of XOR. The following inequalities are sufficient to describe the propagation of the division property for* `xor`*.*

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

Proposition 3 (MILP Model for AND [14]). *Let $(a_1, a_2, \ldots, a_m) \xrightarrow{AND} b$ be a division trail of AND. The following inequalities are sufficient to describe the propagation of the division property for* `and`*.*

$$\begin{cases} \mathcal{M}.var \leftarrow a_1, a_2, \ldots, a_m, b \text{ as binary.} \\ \mathcal{M}.con \leftarrow b \geq a_i \text{ for all } i \in \{1, 2, \ldots, m\} \end{cases}$$

Note: Proposition 3 includes redundant propagations of the division property, but they do not affect preciseness of the obtained characteristics [14].

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 $p(\boldsymbol{x}, \boldsymbol{v})$ defined in Eq. (2) 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 $\boldsymbol{v} \in \mathbb{F}_2^m$, $p(\boldsymbol{x}, \boldsymbol{v})$ can only be related to x_j's $(j \in J)$. In CRYPTO 2017, Todo et al. proposed a method for

determining such a set J using the bit-based division property [14]. 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 4.

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

Proposition 4. *Let $f(\boldsymbol{x}, \boldsymbol{v})$ be a polynomial, where $\boldsymbol{x}$ and $\boldsymbol{v}$ denote the secret and public variables, respectively. For a set of indices $I = \{i_1, i_2, \ldots, i_{|I|}\} \subset \{1, 2, \ldots, m\}$, 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 $\boldsymbol{k}_I$ be an m-dimensional bit vector such that $\boldsymbol{v}^{\boldsymbol{k}_I} = t_I = v_{i_1} v_{i_2} \cdots v_{i_{|I|}}$, i.e. $k_i = 1$ if $i \in I$ and $k_i = 0$ otherwise. Assuming there is no division trail such that $(\boldsymbol{e}_\lambda, \boldsymbol{k}_I) \xrightarrow{f} 1$, 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 trial $(\boldsymbol{e}_\lambda, \boldsymbol{k}_I) \xrightarrow{f} 1$ for $\lambda = 1, \ldots, n$ using the MILP modeling method introduced in Sect. 2.3. If the division trial $(\boldsymbol{e}_\lambda, \boldsymbol{k}_I) \xrightarrow{f} 1$ exists, we have $\lambda \in J$; otherwise, $\lambda \notin J$.

When J is determined, we know that for some assignment to the non-cube $\boldsymbol{IV} \in \mathbb{F}_2^m$, the corresponding superpoly $p_{\boldsymbol{IV}}(\boldsymbol{x})$ 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 $p_{\boldsymbol{IV}}(\boldsymbol{x})$ by constructing its truth table using cube summations defined as Eq. (3). As long as $p_{\boldsymbol{IV}}(\boldsymbol{x})$ is not constant, we can go to the online phase where we sum over the cube $C_I(\boldsymbol{IV})$ to get the exact value of $p_{\boldsymbol{IV}}(\boldsymbol{x})$ 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 $\boldsymbol{IV} \in \mathbb{F}_2^m$ and prepare the cube $C_I(\boldsymbol{IV})$ defined as Eq. (2). For $\boldsymbol{x} \in \mathbb{F}_2^n$ whose $x_j, j \in J$ traverse all $2^{|J|}$ 0-1 combinations, we compute and store the value of the superpoly $p_{\boldsymbol{IV}}(\boldsymbol{x})$ as Eq. (3). The $2^{|J|}$ values compose the truth table of $p_{\boldsymbol{IV}}(\boldsymbol{x})$ and the ANF of the superpoly is determined accordingly. If $p_{\boldsymbol{IV}}(\boldsymbol{x})$ is constant, we pick another $\boldsymbol{IV}$ and repeat the steps above until we find an appropriate one s.t. $p_{\boldsymbol{v}}(\boldsymbol{x})$ is not constant.
2. **Online Phase: Partial Key Recovery.** Query the cube $C_I(\boldsymbol{IV})$ 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 $p_{\boldsymbol{IV}}(x) = \lambda$ according to Eq. (3). So we look up the truth table of the superpoly and only reserve the $x_j, j \in J$ s.t. $p_{\boldsymbol{IV}}(x) = \lambda$.
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 $p_{\boldsymbol{IV}}(\boldsymbol{x})$ is constant

so we have to run several different $\boldsymbol{IV}$'s to find the one we need. The attack can only be meaningful when (1) $|I|+|J|<n$; (2) appropriate $\boldsymbol{IV}$'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 [14,24].

3 Modeling the Constant Bits to Improve the Preciseness 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 an constant 0 output. But, as is pointed out in [24], such a phenomenon cannot be reflected in previous MILP model method. In the previous MILP model, the widely used COPY+AND operation:

$$\text{COPY+AND} : (s_1, s_2) \rightarrow (s_1, s_2, s_1 \wedge s_2). \tag{4}$$

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

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

Assuming that either s_1 or s_2 of Eq. (4) 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 [24], 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 Example 2 for Kreyvium in App. A.

Therefore, for all variables in the MILP $v \in \mathcal{M}.var$, we give them an additional flag $v.F \in \{1_c, 0_c, \delta\}$ where 1_c means the bit is constant 1, 0_c means constant 0 and δ means variable. Apparently, when $v.F = 0_c/1_c$, there is always a constraint $v = 0 \in \mathcal{M}.con$. We define $=$, $\oplus$ and $\times$ operations for the elements of set $\{1_c, 0_c, \delta\}$. 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:

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

The $\times$ operation follows the rules:

$$\begin{cases} 1_c \times x = x \times 1_c = x \\ 0_c \times x = x \times 0_c = 0_c \ \text{ for arbitrary } x \in \{1_c, 0_c, \delta\} \\ \delta \times \delta = \delta \end{cases} \tag{6}$$

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 5 6 and 7 as follows.

Proposition 5 (MILP Model for COPY with Flag). *Let a* $\xrightarrow{COPY}$ $(b_1, b_2, \ldots, b_m)$ *be a division trail of COPY. The following inequalities are sufficient to describe the propagation of the division property for* `copyf`*.*

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

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

Proposition 6 (MILP Model for XOR with Flag). *Let* $(a_1, a_2, \ldots, a_m) \xrightarrow{XOR} b$ *be a division trail of XOR. The following inequalities are sufficient to describe the propagation of the division property for* `xorf`*.*

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

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

Proposition 7 (MILP Model for AND with Flag). *Let* $(a_1, a_2, \ldots, a_m) \xrightarrow{AND} b$ *be a division trail of AND. The following inequalities are sufficient to describe the propagation of the division property for* `andf`*.*

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

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

With these modifications, we are able to improve the preciseness of the MILP model. The improved attack framework can be written as Algorithm 1. It enables us to identify the involved keys 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 $\boldsymbol{IV} \in \mathbb{F}_2^m$ s.t. the corresponding superpoly $p_{\boldsymbol{IV}}(\boldsymbol{x}) \neq 0$.

Algorithm 1 Evaluate secret variables by MILP with Flags

procedure `attackFramework`(Cube indices I, specific assignment to non-cube IVs $\boldsymbol{IV}$ or $\boldsymbol{IV} = \texttt{NULL}$)
  Declare an empty MILP model $\mathcal{M}$
  Declare $\boldsymbol{x}$ as n MILP variables of $\mathcal{M}$ corresponding to secret variables.
  Declare $\boldsymbol{v}$ 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 $i \in (\{1, 2, \ldots, n\} - I)$
  $\mathcal{M}.con \leftarrow \sum_{i=1}^{n} x_i = 1$ and assign $x_i.F = \delta$ for all $i \in \{1, \ldots, n\}$
  if $\boldsymbol{IV} = \texttt{NULL}$ **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:

$$v_i.F = \begin{cases} 1_c & \text{if } \boldsymbol{IV}[i] = 1 \\ 0_c & \text{if } \boldsymbol{IV}[i] = 0 \end{cases}$$

  end if
  Update $\mathcal{M}$ according to round functions and output functions

    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

4 Upper Bounding the Degree of the Superpoly

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

$$p_{\boldsymbol{IV}}(\boldsymbol{x}) = \sum_{\boldsymbol{u} \in \mathbb{F}_2^n} a_{\boldsymbol{u}} \boldsymbol{x}^{\boldsymbol{u}} \tag{7}$$

where $a_{\boldsymbol{u}}$ is determined by the values of the non-cube IVs. If the degree of the superpoly is upper bounded by d, then for all $\boldsymbol{u}$'s with Hamming weight satisfying $hw(\boldsymbol{u}) > d$, we constantly have $a_{\boldsymbol{u}} = 0$. 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 $a_{\boldsymbol{u}}$ for $hw(\boldsymbol{u}) \leq d$. Therefore, we select $\sum_{i=0}^{d} \binom{|J|}{i}$ different $\boldsymbol{x}$'s and construct a linear system with $\left(\sum_{i=0}^{d} \binom{|J|}{i}\right)$ variables and the coefficients as well as the whole ANF of $p_{\boldsymbol{IV}}(\boldsymbol{x})$ 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 $J = \{j_1, j_2, \ldots, j_{|J|}\}$ and the degree of the superpoly $d = \deg p_{\boldsymbol{IV}}(\boldsymbol{x})$, the attack procedure can be adapted as follows:

1. **Offline Phase: Superpoly Recovery.** For all $\binom{|J|}{\leq d}$ $\boldsymbol{x}$'s satisfying $hw(\boldsymbol{x}) \leq d$ and $\bigoplus_{j\in J} \boldsymbol{e}_j \succeq \boldsymbol{x}$, compute the values of the superpolys as $p_{\boldsymbol{IV}}(\boldsymbol{x})$ by summing over the cube $C_I(\boldsymbol{IV})$ as Eq. (3) and generate a linear system of the $\binom{|J|}{\leq d}$ coefficients $a_{\boldsymbol{u}}$ ($hw(\boldsymbol{u}) \leq d$). Solve the linear system, determine the coefficient $a_{\boldsymbol{u}}$ of the $\binom{|J|}{\leq d}$ terms and store them in a lookup table T. The ANF of the $p_{\boldsymbol{IV}}(\boldsymbol{x})$ can be determined with the lookup table.
2. **Online Phase: Partial Key Recovery.** Query the encryption oracle and sum over the cube $C_I(\boldsymbol{IV})$ as Eq. (3) and acquire the exact value of $p_{\boldsymbol{IV}}(\boldsymbol{x})$. For each of the $2^{|J|}$ possible values of $\{x_{j_1}, \ldots, x_{j_{|J|}}\}$, compute the values of the superpoly as Eq. (7) (the coefficient $a_{\boldsymbol{u}}$ 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

$$\max\left\{2^{|I|} \times \binom{|J|}{\leq d}, 2^{|I|} + 2^{|J|} \times \binom{|J|}{\leq d}\right\} < 2^n. \tag{8}$$

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

The knowledge of the algebraic degree of superpolys can largely benefit the efficiency of the cube attack. Therefore, we show how to estimate the algebraic degree of superpolys using the division property. Before the introduction of the method, we generalize Proposition 4 as follows.

Proposition 8. *Let $f(\boldsymbol{x}, \boldsymbol{v})$ be a polynomial, where $\boldsymbol{x}$ and $\boldsymbol{v}$ denote the secret and public variables, respectively. For a set of indices $I = \{i_1, i_2, \ldots, i_{|I|}\} \subset \{1, 2, \ldots, m\}$, 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 $\boldsymbol{k}_I$ be an m-dimensional bit vector such that $\boldsymbol{v}^{\boldsymbol{k}_I} = t_I = v_{i_1} v_{i_2} \cdots v_{i_{|I|}}$. Let $\boldsymbol{k}_\Lambda$ be an n-dimensional bit vector. Assuming there is no division trail such that $(\boldsymbol{k}_\Lambda || \boldsymbol{k}_I) \xrightarrow{f} 1$, the monomial $\boldsymbol{x}^{\boldsymbol{k}_\Lambda}$ is not involved in the superpoly of the cube C_I.*

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

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

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

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

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

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

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

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

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

Algorithm 2 Evaluate upper bound of algebraic degree on the superpoly

procedure `DegEval`(Cube indices I, specific assignment to non-cube IVs $\boldsymbol{IV}$ or $\boldsymbol{IV} =$ `NULL`)
  Declare an empty MILP model $\mathcal{M}$.
  Declare $\boldsymbol{x}$ be n MILP variables of $\mathcal{M}$ corresponding to secret variables.
  Declare $\boldsymbol{v}$ be m MILP variables of $\mathcal{M}$ corresponding to public variables.
  $\mathcal{M}.con \leftarrow v_i = 1$ and assign the flags $v_i.F = \delta$ for all $i \in I$
  $\mathcal{M}.con \leftarrow v_i = 0$ for $i \in (\{1, \ldots, n\} - I)$
  if $\boldsymbol{IV} =$ `NULL` **then**
    Assign the flags $v_i.F = \delta$ for $i \in (\{1, \ldots, n\} - I)$
  else
    Assign the flags of v_i, $i \in (\{1, 2, \ldots, n\} - I)$ as:

$$v_i.F = \begin{cases} 1_c & \text{if } \boldsymbol{IV}[i] = 1 \\ 0_c & \text{if } \boldsymbol{IV}[i] = 0 \end{cases}$$

  end if
  Set the objective function $\mathcal{M}.obj \leftarrow \max \sum_{i=1}^{n} x_i$
  Update $\mathcal{M}$ according to round functions and output functions
  Solve MILP model $\mathcal{M}$
  return The solution of $\mathcal{M}$.
end procedure

According to Proposition 8, the existence of the division trial $(\boldsymbol{k}_\Lambda||\boldsymbol{k}_I) \xrightarrow{f} 1$ is in accordance with the existence of the monomial $x^{\boldsymbol{k}_\Lambda}$ in the superpoly of the cube C_I.

If there is $d \geq 0$ s.t. for all $\boldsymbol{k}_\Lambda$ of hamming weight $hw(\boldsymbol{k}_\Lambda) > d$, the division trail $x^{\boldsymbol{k}_\Lambda}$ 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 2. Same with 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 $\boldsymbol{IV}$ that can either be a specific assignment or a `NULL` referring to 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 $\boldsymbol{x}$ (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. 6.

5 Applications of Flag Technique and Degree Evaluation

We apply our method to 4 NLFSR-based ciphers namely Trivium, Kreyvium, Grain-128a and Acorn. Among them, Trivium, Grain-128a and Acorn are also targets of [14]. Using our new techniques, we can both lower the complexities of previous attacks and give new cubes that mount to more rounds. We give details of the application to Trivium in this section, and the applications to Kreyvium, Grain-128a and Acorn in App. A,B and C respectively.

5.1 Specification of Trivium

Trivium is an NLFSR-based stream cipher, and the internal state is represented by 288-bit state $(s_1, s_2, \ldots, s_{288})$. Fig. 1 shows the state update function of Trivium. The 80-bit key is loaded to the first register, and the 80-bit IV is loaded to the second register. The other state bits are set to 0 except the least three bits in the third register. Namely, the initial state bits are represented as

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

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

$$\begin{aligned}t_1 &\leftarrow s_{66} \oplus s_{93}\\ t_2 &\leftarrow s_{162} \oplus s_{177}\\ t_3 &\leftarrow s_{243} \oplus s_{288}\\ z &\leftarrow t_1 \oplus t_2 \oplus t_3\end{aligned}$$

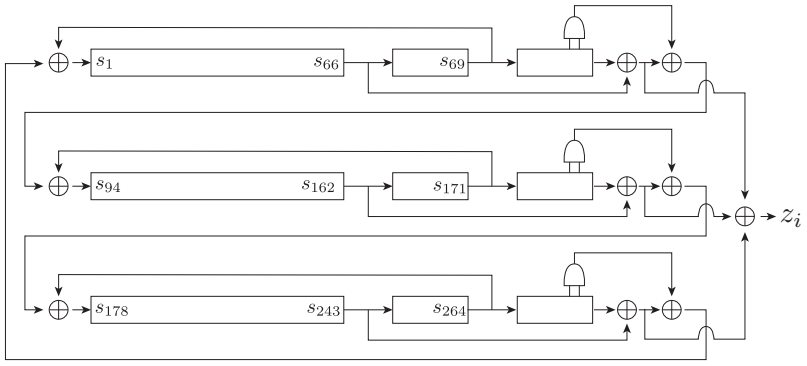

Fig. 1. Structure of TRIVIUM

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

Here z denotes the 1-bit key stream. First, in the key initialization, the state is updated $4 \times 288 = 1152$ times without producing an output. After the key initialization, one bit key stream is produced by every update function.

5.2 MILP Model of TRIVIUM

The only non-linear component of TRIVIUM is a 2-degree core function denoted as f_{core} that takes as input a 288-bit state $\boldsymbol{s}$ and 5 indices $i_1, \ldots, i_5$, and outputs a new 288-bit state $\boldsymbol{s}' \leftarrow f_{core}(\boldsymbol{s}, i_1, \ldots, i_5)$ where

$$
s'_i = \begin{cases} s_{i_1} s_{i_2} + s_{i_3} + s_{i_4} + s_{i_5}, & i = i_5 \\ s_i, & \text{otherwise} \end{cases} \tag{9}
$$

The division property propagation for the core function can be represented as Algorithm 3. The input of Algorithm 3 consists of $\mathcal{M}$ as the current MILP model, a vector of 288 binary variables $\boldsymbol{x}$ describing the current division property of the 288-bit NFSR state, and 5 indices i_1, i_2, i_3, i_4, i_5 corresponding to the input bits. Then Algorithm 3 outputs the updated model $\mathcal{M}$, and a 288-entry vector $\boldsymbol{y}$ describing the division property after f_{core}.

With the definition of `Core`, the MILP model of R-round TRIVIUM can be described as Algorithm 4. This algorithm is a subroutine of Algorithm 1 for generating the MILP model $\mathcal{M}$, and the model $\mathcal{M}$ can evaluate all division trails

Algorithm 3 MILP model of division property for the core function (9)

procedure $\texttt{Core}(\mathcal{M}, \boldsymbol{x}, i_1, i_2, i_3, i_4, i_5)$
  $(\mathcal{M}, y_{i_1}, z_1) \leftarrow \texttt{copyf}(\mathcal{M}, x_{i_1})$
  $(\mathcal{M}, y_{i_2}, z_2) \leftarrow \texttt{copyf}(\mathcal{M}, x_{i_2})$
  $(\mathcal{M}, y_{i_3}, z_3) \leftarrow \texttt{copyf}(\mathcal{M}, x_{i_3})$
  $(\mathcal{M}, y_{i_4}, z_4) \leftarrow \texttt{copyf}(\mathcal{M}, x_{i_4})$
  $(\mathcal{M}, a) \leftarrow \texttt{andf}(\mathcal{M}, z_1, z_2)$
  $(\mathcal{M}, y_{i_5}) \leftarrow \texttt{xorf}(\mathcal{M}, a, z_2, z_3, z_4, x_{i_5})$
  for all $i \in \{1, 2, \ldots, 288\}$ w/o i_1, i_2, i_3, i_4, i_5 **do**
    $y_i = x_i$
  end for
  return $(\mathcal{M}, \boldsymbol{y})$
end procedure

for Trivium whose initialization rounds are reduced to R. Note that constraints to the input division property are imposed by Algorithm 1.

5.3 Experimental Verification

Identical to [14], 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 2, specifying involved key bits using Algorithm 1 with $\boldsymbol{IV} = \texttt{NULL}$ or specific non-cube IV settings.

Example 1 (Verification of Our Attack against 591-round Trivium*).* With $\boldsymbol{IV} = \texttt{NULL}$ 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 with [14]. Then, we set $\boldsymbol{IV}$ to random values and acquire the degree through Algorithm 2, and verify the correctness of the degree by practically recovering the corresponding superpoly.

- When we set $\boldsymbol{IV} = \texttt{0xcc2e487b}, \texttt{0x78f99a93}, \texttt{0xbeae}$, and run Algorithm 2, we get the degree 3. The practically recovered superpoly is also of degree 3:

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

  which is in accordance with the deduction by Algorithm 2 through MILP model.
- When we set $\boldsymbol{IV} = \texttt{0x61fbe5da}, \texttt{0x19f5972c}, \texttt{0x65c1}$, the degree evaluation of Algorithm 2 is 2. The practically recovered superpoly is also of degree 2:

$$p_{\boldsymbol{v}}(\boldsymbol{x}) = x_{23}x_{24} + x_{25} + x_{67} + 1.$$

- When we set $\boldsymbol{IV} = \texttt{0x5b942db1}, \texttt{0x83ce1016}, \texttt{0x6ce}$, the degree is 0 and the superpoly recovered is also constant 0.

Algorithm 4 MILP model of division property for TRIVIUM

procedure `TriviumEval`(round R)
  Prepare empty MILP Model $\mathcal{M}$
  $\mathcal{M}.var \leftarrow v_i$ for $i \in \{1, 2, \ldots, 128\}$. ▷ Declare Public Modifiable IVs
  $\mathcal{M}.var \leftarrow x_i$ for $i \in \{1, 2, \ldots, 128\}$. ▷ Declare Secret Keys
  $\mathcal{M}.var \leftarrow s_i^0$ for $i \in \{1, 2, \ldots, 288\}$
  $s_i^0 = x_i$, $s_{i+93}^0 = v_i$ for $i = 1, \ldots, 80$.
  $\mathcal{M}.con \leftarrow s_i^0 = 0$ for $i = 81, \ldots, 93, 174, \ldots, 288$.
  $s_i^0.F = 0_c$ for $i = 81, \ldots, 285$ and $s_j^0.F = 1_c$ for $j = 286, 287, 288$. ▷ Assign the flags for constant state bits
  for $r = 1$ to R **do**
    $(\mathcal{M}, \boldsymbol{x}) = \texttt{Core}(\mathcal{M}, \boldsymbol{s}^{r-1}, 66, 171, 91, 92, 93)$
    $(\mathcal{M}, \boldsymbol{y}) = \texttt{Core}(\mathcal{M}, \boldsymbol{x}, 162, 264, 175, 176, 177)$
    $(\mathcal{M}, \boldsymbol{z}) = \texttt{Core}(\mathcal{M}, \boldsymbol{y}, 243, 69, 286, 287, 288)$
    $\boldsymbol{s}^r = \boldsymbol{z} \ggg 1$
  end for
  for all $i \in \{1, 2, \ldots, 288\}$ w/o $66, 93, 162, 177, 243, 288$ **do**
    $\mathcal{M}.con \leftarrow s_i^R = 0$
  end for
  $\mathcal{M}.con \leftarrow (s_{66}^R + s_{93}^R + s_{162}^R + s_{177}^R + s_{243}^R + s_{288}^R) = 1$
  return $\mathcal{M}$
end procedure

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 [24]). We try 10000 randomly chosen $\boldsymbol{IV}$'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 $p_{\boldsymbol{IV}}(\boldsymbol{x})$. For 576, 577, 587 and 590 rounds, the accuracy is 100%. In fact, such 100% accuracy is testified in most of our applied ciphers, as shown in App. A, B and C. For 591-round, 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 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 $\boldsymbol{IV} = NULL$. 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 $\boldsymbol{IV} = NULL$, 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 preciseness and can definitely improve the efficiency of the division property based cube attacks.

5.4 Theoretical Results

The best result in [14] mounts to 832-round Trivium with cube dimension $|I| = 72$ and the superpoly involves $|J| = 5$ key bits. The complexity is 2^{77} in [14]. Using Algorithm 2, we further acquire that the degree of such a superpoly is 3. So the complexity for superpoly recovery is $2^{72} \times \binom{5}{\leq 3} = 2^{76.7}$ and the complexity for recovering the partial key is $2^{72} + 2^3 \times \binom{5}{3}$. Therefore, according to Eq. (8), the complexity of this attack is $2^{76.7}$.

We further construct a 77-dimensional cube, $I = \{1, \ldots, 80\} \setminus \{5, 51, 65\}$. Its superpoly after 835 rounds of initialization only involves 1 key bit $J = \{57\}$. So the complexity of the attack is 2^{78}. Since there are only 3 non-cube IVs, we let $\boldsymbol{IV}$ be all 2^3 possible non-cube IV assignments and run Algorithm 1. We find that x_{57} is involved in all of the 2^3 superpolys. So the attack is available for any of the 2^3 non-cube IV assignments. This can also be regarded as a support to the rationality of Assumption 1.

According previous results, 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 $\boldsymbol{IV} = \texttt{0x0}, \texttt{0x4000}, \texttt{0x0}$ or $\boldsymbol{IV} = \texttt{0x0}, \texttt{0x4002}, \texttt{0x0}$. 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.

6 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 $0-$, $1-\ldots$,

Table 2. Summary of theoretical cube attacks on TRIVIUM. The time complexity in this table shows the time complexity of Superpoly Recovery (Phase 1) and Partial Key Recovery (Phase 2).

#Rounds	$\|I\|$	Degree	Involved keys J	Time complexity
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}
833	74∗	1	60 ($\|J\| = 1$)	2^{75}
835	77⋆	1	57 ($\|J\| = 1$)	2^{78}
836	78∘	1	57 ($\|J\| = 1$)	2^{79}
839	78•	1	61 ($\|J\| = 1$)	2^{79}

†: I = {1, 2, ..., 65, 67,69, ..., 79}
‡: I = { 1,2, ..., 67, 69,71, ..., 79}
∗: I = {1,2, ..., 69, 71, 73, ..., 79}
⋆: I = {1, 2, 3, 4, 6, 7, . . . , 50, 52, 53,. . . ,64, 66, 67, . . . , 80 }
∘: I = {1, ..., 11, 13, ..., 42, 44, ..., 80 }
•: I = {1, ..., 33, 35, ..., 46, 48, ..., 80 } and $\boldsymbol{IV}[47] = 1$

$d-$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 appear in the superpoly, we propose Algorithm 5 that enumerates all t-degree monomials that may appear in the superpoly. Apparently, when we use $t = 1$, we can get $J_1 = J$, the same output as Algorithm 1 containing all involved keys. 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 apparently, $|J_t| \leq \binom{|J|}{t}$ 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

$$2^{|I|} \times (1 + \sum_{t=1}^{d} |J_t|) \leq 2^{|I|} \times \binom{|J|}{\leq d}. \tag{10}$$

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

$$\max\{2^{|I|} \times (1 + \sum_{t=1}^{d} |J_t|), 2^{|I|} + 2^{|J|} \times (1 + \sum_{t=1}^{d} |J_t|)\}. \tag{11}$$

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.
Note: J_t's $(t = 1, \ldots, d)$ can be generated by `TermEnum` of Algorithm 5 and they satisfy the following Property 1. This property is equivalent to the "Embed Property" given in [21].

Property 1. For $t = 2, \ldots, d$, if there is $T = (i_1, i_2, \ldots, i_t) \in J_t$ and $T' = (i_{s_1}, \ldots, i_{s_l})$ $(l < t)$ is a subsequence of T $(1 \leq s_1 < \ldots < s_l \leq t)$. Then, we constantly have $T' \in J_l$.

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

Lemma 2. *If $\boldsymbol{k} \succeq \boldsymbol{k}'$ and there is division trial $\boldsymbol{k} \xrightarrow{f} \boldsymbol{l}$, then there is also division trial $\boldsymbol{k}' \xrightarrow{f} \boldsymbol{l}'$ s.t. $\boldsymbol{l} \succeq \boldsymbol{l}'$.*

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 $f : (*, \ldots, *, x) \xrightarrow{COPY} (*, \ldots, *, x, x)$.

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

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

Now we are ready to prove Property 1.

Proof. Let $\boldsymbol{k}, \boldsymbol{k} \in \mathbb{F}_2^n$ 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 trial $(\boldsymbol{k}, \boldsymbol{k}_I) \xrightarrow{R-Rounds} (\boldsymbol{0}, 1)$ Since $k \succeq k'$, we have $(\boldsymbol{k}, \boldsymbol{k}_I) \succeq (\boldsymbol{k}', \boldsymbol{k}_I)$ and according to Lemma 2, there is division trial s.t. $(\boldsymbol{k}', \boldsymbol{k}_I) \xrightarrow{R-Rounds} (\boldsymbol{0}^{m+n}, s)$ where $(\boldsymbol{0}^{m+n}, 1) \succeq (\boldsymbol{0}^{m+n}, s)$. Since the hamming weight of $(\boldsymbol{k}', \boldsymbol{k}_I)$ is larger than 0 and there is no combination of COPY, AND and XOR that makes non-zero division property to all-zero division property. So we have $s = 1$ and there exist division trial $(\boldsymbol{k}', \boldsymbol{k}_I) \xrightarrow{R-Rounds} (\boldsymbol{0}, 1)$. □

Property 1 reveals a limitation of Algorithm 5. Assume the superpoly is

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

We can acquire $J_3 = \{(1, 2, 3)\}$ by running `TermEnum` of Algorithm 5. 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.

Algorithm 5 Enumerate all the terms of degree t

procedure `TermEnum`(Cube indices I, specific assignment to non-cube IVs $\boldsymbol{IV}$ or $\boldsymbol{IV}$ = `NULL`, targeted degree t)
  Declare an empty MILP model $\mathcal{M}$ and an empty set $J_t = \phi \subseteq \{1, \ldots, n\}^n$
  Declare $\boldsymbol{x}$ as n MILP variables of $\mathcal{M}$ corresponding to secret variables.
  Declare $\boldsymbol{v}$ 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 $i \in (\{1, 2, \ldots, n\} - I)$
  $\mathcal{M}.con \leftarrow \sum_{i=1}^{n} x_i = t$ and assign $x_i.F = \delta$ for all $i \in \{1, \ldots, n\}$
  if $\boldsymbol{IV}$ = `NULL` **then**
    $v_i.F = \delta$ for all $i \in (\{1, 2, \ldots, n\} - I)$
  else
    Assign the flags of v_i, $i \in (\{1, 2, \ldots, n\} - I)$ as:

$$v_i.F = \begin{cases} 1_c & \text{if } \boldsymbol{IV}[i] = 1 \\ 0_c & \text{if } \boldsymbol{IV}[i] = 0 \end{cases}$$

  end if
  Update $\mathcal{M}$ according to round functions and output functions

    solve MILP model $\mathcal{M}$
    if $\mathcal{M}$ is feasible **then**
      pick index sequence $(j_1, \ldots, j_t) \subseteq \{1, \ldots, n\}^t$ s.t. $x_{j_1} = \ldots = x_{j_t} = 1$
      $J_t = J_t \cup \{(j_1, \ldots, j_t)\}$
      $\mathcal{M}.con \leftarrow \sum_{i=1}^{t} x_{j_i} \leq t - 1$
    end if
  while $\mathcal{M}$ is feasible
  return J_t
end procedure

procedure `RTermEnum`(Cube indices I, specific assignment to non-cube IVs $\boldsymbol{IV}$ or $\boldsymbol{IV}$ = `NULL`, targeted degree t)
  Declare an empty MILP model $\mathcal{M}$ and an empty set $JR_t = \phi \subseteq \{1, \ldots, n\}$
  Declare $\boldsymbol{x}$ as n MILP variables of $\mathcal{M}$ corresponding to secret variables.
  Declare $\boldsymbol{v}$ 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 $i \in (\{1, 2, \ldots, n\} - I)$
  $\mathcal{M}.con \leftarrow \sum_{i=1}^{n} x_i \geq t$ and assign $x_i.F = \delta$ for all $i \in \{1, \ldots, n\}$
  if $\boldsymbol{IV}$ = `NULL` **then**
    $v_i.F = \delta$ for all $i \in (\{1, 2, \ldots, n\} - I)$
  else
    Assign the flags of v_i, $i \in (\{1, 2, \ldots, n\} - I)$ as:

$$v_i.F = \begin{cases} 1_c & \text{if } \boldsymbol{IV}[i] = 1 \\ 0_c & \text{if } \boldsymbol{IV}[i] = 0 \end{cases}$$

  end if
  Update $\mathcal{M}$ according to round functions and output functions

    solve MILP model $\mathcal{M}$
    if $\mathcal{M}$ is feasible **then**
      pick index set $\{j_1, \ldots, j_{t'}\} \subseteq \{1, \ldots, n\}$ s.t. $t' \geq t$ and $x_{j_1} = \ldots = x_{j_{t'}} = 1$
      $JR_t = JR_t \cup \{j_1, \ldots, j_{t'}\}$
      $\mathcal{M}.con \leftarrow \sum_{i \notin JR_t} x_i \geq 1$
    end if
  while $\mathcal{M}$ is feasible
  return JR_t
end procedure

Applying such term enumeration technique, we are able to lower complexities of many existing attacks namely: 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 5. In the relaxed algorithm, `RTermEnum` is acquired from `TermEnum` by replacing some states which are marked in red in Algorithm 5, and we state details later in Sect. 6.4.

6.1 Application to TRIVIUM

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 $\binom{5}{\leq 3} = 26$ using previous technique. But by running Algorithm 5, we find that $|J_2| = 5$, $|J_3| = 1$, so we have

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

Therefore, the complexity has now been lowered from $2^{76.7}$ to $2^{75.8}$. Similar technique can also be applied to the 73 dimensional cube of Table 2. Details are shown in Table 3.

Table 3. Results of TRIVIUM with Precise Term Enumeration

#Rounds	$\|I\|$	$\|J_1\|$	$\|J_2\|$	$\|J_3\|$	$\|J_4\|$	$\|J_5\|$	$\|J_t\|$, $t \geq 6$	$1 + \sum_{t=1}^{d} \|J_t\|$	Previous	Improved
832	72	5	5	1	0	0	0	$12 \approx 2^{3.58}$	$2^{76.7}$	$2^{75.58}$
833	73	7	6	1	0	0	0	$15 \approx 2^{3.91}$	2^{79}	$2^{76.91}$

6.2 Applications to Kreyvium

We revisit the 61-dimensional cube first given in [25] and transformed to a key recovery attack on 849-round Kreyvium in [24]. The degree of the superpoly is 9 so the complexity is given as $2^{81.7}$ in Appex. A. Since $J = J_1$ is of size 23, we enumerate all the terms of degree 2-9 and acquire the sets $J_2, \ldots, J_9$. $1 + \sum_{t=1}^{d} |J_t| = 5452 \approx 2^{12.41}$. So the complexity is now lowered to $2^{73.41}$. The details are listed in Table 4.

Table 4. Results of Kreyvium with Precise Term Enumeration

#Rounds	$\|I\|$	$\|J_1\|$	$\|J_2\|$	$\|J_3\|$	$\|J_4\|$	$\|J_5\|$	$\|J_6\|$	$\|J_7\|$	$\|J_8\|$	$\|J_9\|$	$1 + \sum_{t=1}^{d} \|J_t\|$	Previous	Improved
849	61	23	158	555	1162	1518	1235	618	156	26	$5452 \approx 2^{12.41}$	$2^{81.7}$	$2^{73.41}$

6.3 Applications to Grain-128a

For the attack on 184-round Grain-128a, the superpoly has degree $d = 14$, the number of involved key bits is $|J| = |J_1| = 21$ and we are able to enumerate all terms of degree 1-14 as Table 5.

Table 5. Results of Grain-128a with Term Enumeration

#Rounds	$\|I\|$	$\|J_1\|$	$\|J_i\|$ $(2 \le i \le 14)$	$1 + \sum_{t=1}^{d} \|J_t\|$	Previous	Improved
184	95	21	157, 651, 1765, 3394, 4838, 5231, 4326, 2627, 1288, 442, 104, 15, 1	$2^{14.61}$	$2^{115.95}$	$2^{109.61}$

6.4 Applications to Acorn

For the attack on 704-round Acorn, with the cube dimension 64, the number of involved key bits in the superpoly is 72, and the degree is 7. We enumerate all the terms of degree from 2 to 7 as in Table 6, therefore we manage to improve the complexity of our cube attack in the previous section.

Table 6. Results of Acorn with Precise Term Enumeration

#Rounds	$\|I\|$	$\|J_1\|$	$\|J_2\|$	$\|J_3\|$	$\|J_4\|$	$\|J_5\|$	$\|J_6\|$	$\|J_7\|$	$1 + \sum_{t=1}^{d} \|J_t\|$	Previous	Improved
704	64	72	1598	4911	5755	2556	179	3	$2^{13.88}$	$2^{93.23}$	$2^{77.88}$

Relaxed Algorithm 5. For the attack on 750-round Acorn (the superpoly is of degree $d = 5$), The left part of Algorithm 5 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. We settle for the index set JR_t containing the key indices that composing 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 $J_1 = JR_1$. The searching space for J_t in Algorithm 5 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. (8)) as long as $|JR_t| < |J_1|$. The complexity of this relaxed version can be written as

$$\max\{2^{|I|} \times (1 + \sum_{t=1}^{d-1} \binom{|JR_t|}{t} + J_d), 2^{|I|} + 2^{|J|} \times (1 + \sum_{t=1}^{d-1} \binom{|JR_t|}{t} + J_d)\} \quad (12)$$

For 750-round Acorn, we enumerate J_5 and $JR_1, \ldots, JR_4$ whose sizes are listed in Table 7. The improved complexity, according to Eq. (12), is $2^{120.92}$, lower than the original $2^{125.71}$ given in App. A.

Table 7. Results of 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	$2^{19.92}$	$2^{125.71}$	$2^{120.92}$

7 A Clique View of the Superpoly Recovery

The precise & relaxed term enumeration technique introduced in Sect. 6 have to execute many MILP instances, which is difficult for some applications. In this section, we represent the resultant superpoly as a graph, which is called *superpoly graph*, so that we can utilize the clique concept from the graph theory to upper bound the complexity of the superpoly recovery phase in our attacks, without requiring MILP solver as highly as the term enumeration technique.

Definition 3 (Clique[34]). *In a graph $G = (V, E)$, where V is the set of vertices and E is the set of edges, a subset $C \subseteq V$, s.t. each pair of vertices in C is connected by an edge is called a clique.*

A *i-clique* is defined as a clique consists of i vertices, and i is called the *clique number*. A 1-clique is a vertex, a 2-clique is just an edge, and a 3-clique is called a triangle.

Given a cube C_I, by running Algorithm 5 for degree i, we determine J_i, which is the set of all the degree-i terms that might appear in the superpoly $p(\boldsymbol{x}, \boldsymbol{v})$ (see Sect. 6). Then we represent $p(\boldsymbol{x}, \boldsymbol{v})$ as a graph $G = (J_1, J_2)$, where the vertices in J_1 correspond to the involved secret key bits in $p(\boldsymbol{x}, \boldsymbol{v})$, the edges between any pairs of the vertices reveal the quadratic terms involved in $p(\boldsymbol{x}, \boldsymbol{v})$, We call the graph $G = (J_1, J_2)$ the *superpoly graph* of the cube C_I. The set of i-cliques in the superpoly graph is denoted as $\mathcal{K}_i$. Note that there is a natural one-to-one correspondence between the sets J_i and $\mathcal{K}_i$ for $i = 1, 2$.

It follows from the definition of a clique that any i-clique in $\mathcal{K}_i$ ($i \geq 2$) represents a monomial of degree i whose all divisors of degree 2 belong to J_2. On the other hand, due to the "embed" Property 1 in Sect. 6, we have that all its quadratic divisors must be in J_2. Then any monomial in J_i can be represented by an i-clique in $\mathcal{K}_i$. Hence for all $i \geq 2$, J_i corresponds to a subset of $\mathcal{K}_i$. Denote the number of i-cliques as $|\mathcal{K}_i|$, then $|J_i| \leq |\mathcal{K}_i|$. Apparently, $|\mathcal{K}_i| \leq \binom{|J|}{i}$ for all $1 \leq i \leq d$.

Now we show a simple algorithm for constructing $\mathcal{K}_i$ from J_1 and J_2 for $i \geq 3$. For instance, when constructing $\mathcal{K}_3$, we take the union operation of all possible combinations of three elements from J_2, and only keep the elements of degree 3. Similarly, we construct $\mathcal{K}_i$ for $3 < i \leq d$, where d is the degree of the superpoly. Therefore, all the i-cliques ($3 \leq i \leq d$) are found by the simple algorithm, i.e. the number of i-cliques $|\mathcal{K}_i|$ in $G(J_1, J_2)$ is determined. We therefore can upper bound the complexity of the offline phase as

$$2^{|I|} \times (1 + \sum_{i=1}^{d} |\mathcal{K}_i|). \tag{13}$$

Note that we have

$$|J_i| \leq \ |\mathcal{K}_i| \leq \binom{|J_1|}{i}.$$

It indicates that the upper bound of the superpoly recovery given by clique theory in Eq. (13) is better than the one provided by our degree evaluation in Eq. (8), while it is weaker than the one presented by our term enumeration techniques in Eq. (10). However, it is unclear if there exists a specific relation between $|\mathcal{K}_i|$ and $\binom{|JR_i|}{i}$ in the relaxed terms enumeration technique.

Advantage over the terms enumeration techniques. In Sect. 6 when calculating J_i ($i \geq 3$) by Algorithm 5, we set the target degree as i and solve the newly generated MILP to obtain J_i, regardless of the knowledge of J_{i-1} we already hold. On the other hand, as is known in some cases, the MILP solver might take long time before providing J_i as desired. However, by using clique theory, we first acquire J_1 and J_2, which are essential for the term enumeration method as well. According to the "embed" property, we then make full use of the knowledge of J_1 and J_2, to construct $\mathcal{K}_i$ for $i \geq 3$ by an algorithm which is actually just performing simple operations (like union operations among elements, or removal of repeated elements, etc) in sets. So hardly any cost is required to find all the $\mathcal{K}_i$ ($3 \leq i \leq d$) we want. This significantly saves the computation costs since solving MILP is usually very time-consuming.

8 Conclusion

Algebraic properties of the resultant superpoly of the cube attacks were further studied. We developed a division property based framework of cube attacks enhanced by the flag technique for identifying proper non-cube IV assignments. The relevance of our framework is three-fold: For the first time, it can identify proper non-cube IV assignments of a cube leading to a non-constant superpoly, rather than randomizing trails & summations in the offline phase. Moreover, our model derived the upper bound of the superpoly degree, which can break the $|I| + |J| < n$ barrier and enable us to explore even larger cubes or mount to attacks on more rounds. Furthermore, our accurate term enumeration techniques further reduced the complexities of the superpoly recovery, which brought us the current best key recovery attacks on ciphers namely Trivium, Kreyvium, Grain-128a and Acorn.

Besides, when term enumeration cannot be carried out, we represent the resultant superpoly as a graph. By constructing all the cliques of our superpoly graph, an upper bound of the complexity of superpoly recovery can be obtained.

Acknowledgements. 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 (Grant No. 2018YFA0306404), National Natural Science

Foundation of China (No. 61472250, No. 61672347), Program of Shanghai Academic/Technology Research Leader (No. 16XD1401300), 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. Dinur, I., Shamir, A.: Cube attacks on tweakable black box polynomials. In Joux, A., ed.: EUROCRYPT 2009. Volume 5479 of LNCS., Springer (2009) 278–299
2. Aumasson, J., Dinur, I., Meier, W., Shamir, A.: Cube testers and key recovery attacks on reduced-round MD6 and Trivium. In Dunkelman, O., ed.: FSE 2009. Volume 5665 of LNCS., Springer (2009) 1–22
3. Dinur, I., Shamir, A.: Breaking Grain-128 with dynamic cube attacks. In Joux, A., ed.: FSE 2011. Volume 6733 of LNCS., Springer (2011) 167–187
4. Fouque, P., Vannet, T.: Improving key recovery to 784 and 799 rounds of Trivium using optimized cube attacks. In Moriai, S., ed.: FSE 2013. Volume 8424 of LNCS., Springer (2013) 502–517
5. Salam, M.I., Bartlett, H., Dawson, E., Pieprzyk, J., Simpson, L., Wong, K.K.: Investigating cube attacks on the authenticated encryption stream cipher ACORN. In Batten, L., Li, G., eds.: ATIS 2016. Volume 651 of CCIS., Springer (2016) 15–26
6. Liu, M., Yang, J., Wang, W., Lin, D.: Correlation Cube Attacks: From Weak-Key Distinguisher to Key Recovery. In Nielsen, J.B., Rijmen, V., eds.: EUROCRYPT 2018 Part II. Volume 10821 of Lecture Notes in Computer Science., Springer (2018) 715–744
7. Sun, Y.S.: Cube attack on round-reduced Fruit. Journal of Cryptologic Research **4**(6) (2017) 528
8. Ren, Y.Q., Sun, Y., Wang, Y.J.: A space-time tradeoff cube attack on Grain-v1. Journal of Cryptologic Research **2**(3) (2015) 235
9. Dinur, I., Morawiecki, P., Pieprzyk, J., Srebrny, M., Straus, M.: Cube attacks and cube-attack-like cryptanalysis on the round-reduced Keccak sponge function. In Oswald, E., Fischlin, M., eds.: EUROCRYPT 2015 Part I. Volume 9056 of LNCS., Springer (2015) 733–761
10. Huang, S., Wang, X., Xu, G., Wang, M., Zhao, J.: Conditional Cube Attack on Reduced-Round Keccak Sponge Function. In Coron, J., Nielsen, J.B., eds.: EUROCRYPT 2017 Part II. Volume 10211 of LNCS., Springer (2017) 259–288
11. Li, Z., Bi, W., Dong, X., Wang, X.: Improved conditional cube attacks on Keccak keyed modes with MILP method. In Takagi, T., Peyrin, T., eds.: ASIACRYPT 2017 Part I. Volume 10624 of LNCS., Springer (2017) 99–127
12. Li, Z., Dong, X., Wang, X.: Conditional cube attack on round-reduced ASCON. IACR Trans. Symmetric Cryptol. **2017**(1) (2017) 175–202
13. Dong, X., Li, Z., Wang, X., Qin, L.: Cube-like attack on round-reduced initialization of Ketje Sr. IACR Trans. Symmetric Cryptol. **2017**(1) (2017) 259–280
14. Todo, Y., Isobe, T., Hao, Y., Meier, W.: Cube attacks on non-blackbox polynomials based on division property. In Katz, J., Shacham, H., eds.: CRYPTO 2017 Part III. Volume 10403 of LNCS., Springer (2017) 250–279
15. Todo, Y.: Structural evaluation by generalized integral property. In Oswald, E., Fischlin, M., eds.: EUROCRYPT 2015 Part I. Volume 9056 of LNCS., Springer (2015) 287–314

16. Todo, Y.: Integral cryptanalysis on full MISTY1. In Gennaro, R., Robshaw, M., eds.: CRYPTO 2015 Part I. Volume 9215 of LNCS., Springer (2015) 413–432
17. Todo, Y., Morii, M.: Bit-based division property and application to SIMON family. In Peyrin, T., ed.: FSE 2016. Volume 9783 of LNCS., Springer (2016) 357–377
18. Xiang, Z., Zhang, W., Bao, Z., Lin, D.: Applying MILP method to searching integral distinguishers based on division property for 6 lightweight block ciphers. In Cheon, J.H., Takagi, T., eds.: ASIACRYPT 2016 Part I. Volume 10031 of LNCS., Springer (2016) 648–678
19. Gu, Z., Rothberg, E., Bixby, R.: Gurobi optimizer. `http://www.gurobi.com/`
20. Sun, L., Wang, W., Wang, M.: 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`.
21. Sun, L., Wang, W., Wang, M.: Automatic search of bit-based division property for ARX ciphers and word-based division property. In Takagi, T., Peyrin, T., eds.: ASIACRYPT 2017 Part I. Volume 10624 of LNCS., Springer (2017) 128–157
22. Funabiki, Y., Todo, Y., Isobe, T., Morii, M.: Improved integral attack on HIGHT. In Pieprzyk, J., Suriadi, S., eds.: ACISP 2017 Part I. Volume 10342 of LNCS., Springer (2017) 363–383
23. Wang, Q., Grassi, L., Rechberger, C.: Zero-sum partitions of PHOTON permutations. In Smart, N., ed.: CT-RSA 2018. Volume 10808 of LNCS., Springer (2018)
24. Todo, Y., Isobe, T., Hao, Y., Meier, W.: 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`.
25. Liu, M.: Degree evaluation of NFSR-based cryptosystems. In Katz, J., Shacham, H., eds.: CRYPTO 2017 Part III. Volume 10403 of LNCS., Springer (2017) 227–249
26. Fu, X., Wang, X., Dong, X., Meier, W.: A key-recovery attack on 855-round Trivium. Cryptology ePrint Archive, Report 2018/198 (2018) `https://eprint.iacr.org/2018/198`.
27. Todo, Y., Isobe, T., Meier, W., Aoki, K., Zhang, B.: Fast correlation attack revisited–cryptanalysis on full Grain-128a, Grain-128, and Grain-v1. CRYPTO 2018 (2018) (accepted).
28. Lehmann, M., Meier, W.: Conditional differential cryptanalysis of Grain-128a. In Pieprzyk, J., Sadeghi, A., Manulis, M., eds.: CANS 2012. Volume 7712 of LNCS., Springer (2012) 1–11
29. Mouha, N., Wang, Q., Gu, D., Preneel, B.: Differential and linear cryptanalysis using mixed-integer linear programming. In Wu, C., Yung, M., Lin, D., eds.: Inscrypt 2011. Volume 7537 of LNCS., Springer (2011) 57–76
30. Sun, S., Hu, L., Wang, P., Qiao, K., Ma, X., Song, L.: Automatic security evaluation and (related-key) differential characteristic search: Application to SIMON, PRESENT, LBlock, DES(L) and other bit-oriented block ciphers. In Sarkar, P., Iwata, T., eds.: ASIACRYPT 2014 Part I. Volume 8873 of LNCS., Springer (2014) 158–178
31. Sun, S., Hu, L., Wang, M., Wang, P., Qiao, K., Ma, X., Shi, D., Song, L., Fu, K.: 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`.
32. Cui, T., Jia, K., Fu, K., Chen, S., Wang, M.: 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`.

33. Sasaki, Y., Todo, Y.: New impossible differential search tool from design and cryptanalysis aspects - revealing structural properties of several ciphers. In Coron, J., Nielsen, J.B., eds.: EUROCRYPT 2017 Part III. Volume 10212 of LNCS., Springer (2017) 185–215
34. Bondy, J.A., Murty, U.S.R.: Graph theory with applications. Volume 290. Macmillan London (1976)
35. Wu, H.: Acorn v3 (2016) Submission to CAESAR competition.

A Application to Kreyvium

A.1 Specification of Kreyvium

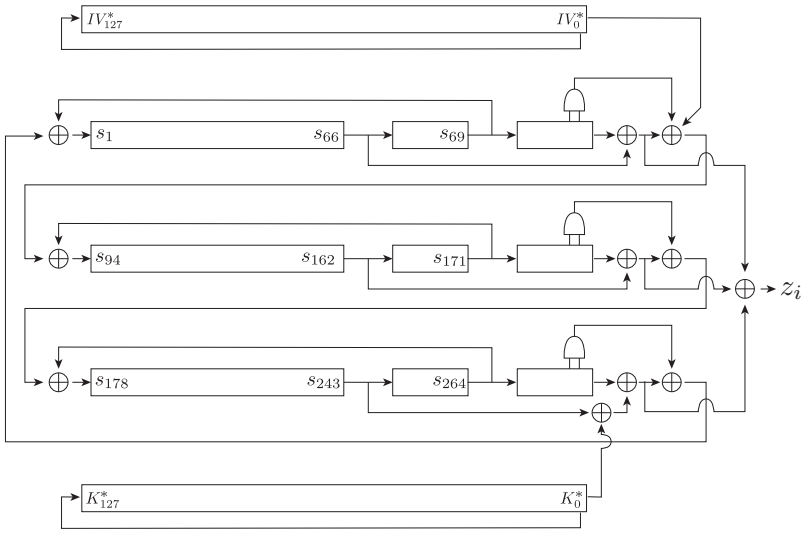

Fig. 2. Structure of Kreyvium

Kreyvium is designed for the use of fully Homomorphic encryption. It claims 128-bit security and accepts 128-bit IV. Kreyvium consists of 5 registers. Two of them are LFSRs denoted as K^* and IV^* respectively. The remaining three are NFSRs that are concatenated to make up a 288-bit state denoted as

$$S = ((s_1, \ldots, s_{93}), (s_{94}, \ldots, s_{177}), (s_{178}, \ldots, s_{288}))$$

The registers are initialized with the 128 key bits, $K_1, \ldots, K_{128}$, and 128 IV bits, $IV_1, \ldots, IV_{128}$ as follows:

$$\begin{aligned}
(IV^*_{127}, IV^*_{126}, \ldots, IV^*_0) &= (IV_1, IV_2, \ldots, IV_{128}),\\
(s_1, s_2, \ldots, s_{93}) &= (K_1, K_2, \ldots, K_{93}),\\
(s_{94}, s_{95}, \ldots, s_{177}) &= (IV_1, IV_2, \ldots, IV_{84}),\\
(s_{178}, s_{279}, \ldots, s_{288}) &= (IV_{85}, IV_{86}, \ldots, IV_{128}, 1, 1, \ldots, 1, 0),\\
(K^*_{127}, K^*_{126}, \ldots, K^*_0) &= (K_1, K_2, \ldots, K_{128}),
\end{aligned}$$

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

$$\begin{aligned}
&t_1 \leftarrow s_{66} \oplus s_{93}\\
&t_2 \leftarrow s_{162} \oplus s_{177}\\
&t_3 \leftarrow s_{243} \oplus s_{288} \oplus K^*_0\\
&z \leftarrow t_1 \oplus t_2 \oplus t_3\\
&t_1 \leftarrow t_1 \oplus s_{91} \cdot s_{92} \oplus s_{171} \oplus IV^*_0\\
&t_2 \leftarrow t_2 \oplus s_{175} \cdot s_{176} \oplus s_{264}\\
&t_3 \leftarrow t_3 \oplus s_{286} \cdot s_{287} \oplus s_{69}\\
&(s_1, s_2, \ldots, s_{93}) \leftarrow (t_3, s_1, \ldots, s_{92})\\
&(s_{94}, s_{95}, \ldots, s_{177}) \leftarrow (t_1, s_{94}, \ldots, s_{176})\\
&(s_{178}, s_{279}, \ldots, s_{288}) \leftarrow (t_2, s_{178}, \ldots, s_{287})\\
&(K^*_{127}, K^*_{126}, \ldots, K^*_0) \leftarrow (K^*_0, K^*_{127}, K^*_{126}, \ldots, K^*_1)\\
&(IV^*_{127}, IV^*_{126}, \ldots, IV^*_0) \leftarrow (IV^*_0, IV^*_{127}, IV^*_{126}, \ldots, IV^*_1)
\end{aligned}$$

Here z denotes the 1-bit key stream. First, in the key initialization, the state is updated $4 \times 288 = 1152$ times without producing an output. After the key initialization, one bit key stream is produced by every update function. Fig. 2 depicts the state update function of Kreyvium.

A.2 MILP Model of Kreyvium

Kreyvium shares the same core function with TRIVIUM as is defined in Eq. (9), and the division property propagation for the core function is denoted as `Core` and defined in Algorithm 3. Besides `Core`, the LFSRs $\boldsymbol{K}^*$ and $\boldsymbol{IV}^*$ are modeled as `LFSR` in Algorithm 6. With the current MILP model $\mathcal{M}$ and a vector $\boldsymbol{x}$ of 128 binary MILP variables as input, Algorithm 6 outputs the updated model $\mathcal{M}$, a new 128-variable vector $\boldsymbol{y}$ describing the division property of the LFSR after one round of update, and an additional variable o describing the output of LFSR.

With the definitions of `Core` and `LFSR`, the MILP model for R-round Kreyvium can be described as Algorithm 7. This algorithm generates MILP model $\mathcal{M}$ as the input of Algorithm 1, and the model $\mathcal{M}$ can evaluate all division trails for Kreyvium whose initialization rounds are reduced to R.

Algorithm 6 MILP model of division property for the $\boldsymbol{K}^*$ and $\boldsymbol{IV}^*$

procedure LFSR($\mathcal{M}, \boldsymbol{x}$)
  $(\mathcal{M}, a, o) \leftarrow$ copyf($\mathcal{M}, x_0$)
  for all $i \in \{0, 1, \ldots, 126\}$ **do**
    $y_i = x_{i+1}$
  end for
  $y_{127} = a$
  return $(\mathcal{M}, \boldsymbol{y}, o)$
end procedure

A.3 Experimental Verification

Identical to [24], we still use 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 2, specifying involved key bits using Algorithm 1 with $\boldsymbol{IV} =$ NULL or specific non-cube IV settings.

Example 2 (Verification of Our Attack against 576-round Kreyvium). With $\boldsymbol{IV} =$ NULL using Algorithm 1, we are able to identify $J = \{48, 73, 74, 75\}$. We know that with some assignment to the non-cube IV bits, the superpoly can be a polynomial of secret key bits $x_{48}, x_{73}, x_{74}, x_{75}$. Using Algorithm 2, we know that degree of the superpoly is upper bounded by 2, so the superpoly is at most a 2-degree polynomial of $x_{48}, x_{73}, x_{74}, x_{75}$.

Then, we run Algorithm 1 with specific randomly chosen $\boldsymbol{IV}$ and verify the MILP-deduced results by exactly running the cube attacks with over 2^{10} randomly chosen keys. We find that the MILP evaluation is in high accordance with the experiment:

- When $\boldsymbol{IV}$ is assigned to specific value such as $\boldsymbol{IV} =$ (0x613fa9ca, 0x5068e953, 0xe0f73db6, 0xc8c3491f), the MILP model reports that all 4 key bits x_{48}, x_{73}, x_{74}, x_{75} are involved in the superpoly. Furthermore, if we run Algorithm 2 with this $\boldsymbol{IV}$, it can output degree 2 and identify a 2-degree monomial $x_{73}x_{74}$. It has been experimentally verified that the superpoly corresponding to this non-cube IV setting is

$$p_{\boldsymbol{IV}}(\boldsymbol{x}) = x_{48} + x_{73}x_{74} + x_{75}.$$

- Contrarily, if we use another IV assignment $\boldsymbol{IV} =$ (0x3ce780c, 0x6e0445b3, 0xefe379b5, 0x58e15b6c), and run Algorithm 1, we will find that the set J is empty—the superpoly is irrelevant with any secret key bits. This is experimentally verified since the summation is constantly 0 for arbitrary many randomly chosen keys.
- When we assign all non-cube IV bits to 1, or equivalently $\boldsymbol{IV} =$ (0xffffffff, 0xffffffff, 0xffffffff, 0xffffffff), both the MILP model and the practical cube summation shows that $p_{\boldsymbol{IV}}(\boldsymbol{x}) = 0$. While using the method of identifying the constant 0 bits given in [24] cannot detect such a phenomenon. This example verifies the advantage of our flag technique.

Algorithm 7 MILP model of division property for Kreyvium

procedure `KreyviumEval`(round R)
  Prepare empty MILP Model $\mathcal{M}$
  $\mathcal{M}.var \leftarrow v_i$ for $i \in \{1, 2, \ldots, 128\}$. ▷ Declare Public Modifiable IVs
  $\mathcal{M}.var \leftarrow x_i$ for $i \in \{1, 2, \ldots, 128\}$. ▷ Declare Secret Keys
  for $i = 1$ to 128 **do** ▷ Initialize $\boldsymbol{K}^*$
    if $i \leq 93$ **then**
      $(\mathcal{M}, s_i^0, K_{128-i}^0) \leftarrow \texttt{copyf}(\mathcal{M}, x_i, 2)$.
    else
      $K_{128-i}^0 = x_i$.
    end if
  end for
  for $i = 1$ to 128 **do** ▷ Initialize $\boldsymbol{IV}^*$
    $(\mathcal{M}, s_{93+i}^0, IV_{128-i}^0) \leftarrow \texttt{copyf}(\mathcal{M}, v_i, 2)$.
  end for
  for $i = 222$ to 287 **do** ▷ Constant 1 bits
    $\mathcal{M}.con \leftarrow s_i^0 = 0,\ s_i^0.F = 1_c$.
  end for
  $\mathcal{M}.con \leftarrow s_{288}^0 = 0,\ s_{288}^0.F = 0_c$. ▷ Constant 0 bit
  for $r = 1$ to R **do**
    $(\mathcal{M}, \boldsymbol{x}) \leftarrow \texttt{Core}(\mathcal{M}, \boldsymbol{s}^{r-1}, 66, 171, 91, 92, 93)$
    $(\mathcal{M}, \boldsymbol{IV}^r, v^*) \leftarrow \texttt{LFSR}(\mathcal{M}, \boldsymbol{IV}^{r-1})$
    $(\mathcal{M}, t_1) \leftarrow \texttt{xorf}(\mathcal{M}, v^*, x_{93})$
    $x_{93} = t_1$ ▷ Update the 93rd entry of $\boldsymbol{x}$
    $(\mathcal{M}, \boldsymbol{y}) \leftarrow \texttt{Core}(\mathcal{M}, \boldsymbol{x}, 162, 264, 175, 176, 177)$
    $(\mathcal{M}, \boldsymbol{z}) \leftarrow \texttt{Core}(\mathcal{M}, \boldsymbol{y}, 243, 69, 286, 287, 288)$
    $(\mathcal{M}, \boldsymbol{K}^r, k^*) \leftarrow \texttt{LFSR}(\mathcal{M}, \boldsymbol{K}^{r-1})$
    $(\mathcal{M}, t_3) \leftarrow \texttt{xorf}(\mathcal{M}, k^*, y_{288})$
    $z_{288} = t_3$ ▷ Update the 288th entry of $\boldsymbol{z}$
    $\boldsymbol{s}^r = \boldsymbol{z} \ggg 1$
  end for
  for all $i \in \{1, 2, \ldots, 288\}$ w/o $66, 93, 162, 177, 243, 288$ **do**
    $\mathcal{M}.con \leftarrow s_i^R = 0$
  end for
  for all $i \in \{1, 2, \ldots, 128\}$ **do**
    $\mathcal{M}.con \leftarrow K_i^R = 0$
    $\mathcal{M}.con \leftarrow IV_i^R = 0$
  end for
  $\mathcal{M}.con \leftarrow (s_{66}^r + s_{93}^r + s_{162}^r + s_{177}^r + s_{243}^r + s_{288}^r) = 1$
  return $\mathcal{M}$
end procedure

On the accuracy of MILP model with flag technique. We've tried 10000 random $\boldsymbol{IV}$ and the accuracy is 100% for Kreyvium: when the MILP degree evaluation gives $d = 2$, the superpoly is constantly a 2-degree polynomial; if $d = 0$, the superpoly is constant 0. These experiments have verified that the method of modeling constant bits using the method in Sect. 3 can largely improve the preciseness of the cube attack. Under the effect of flags technique, the MILP model can now specify the key bits and identify most significant terms for specific non-cube IV assignments.

A.4 Theoretical Results

Firstly, by running Algorithm 1 with parameter $\boldsymbol{IV} = (0, 0, 0, 0)$, we are able to prove the availability of the zero-sum distinguishers given in [25,24]. On the contrary, for higher-dimensional cube I's, the degrees acquired by running Algorithm 1 with $\boldsymbol{IV} = (\texttt{0xffffffff}, \texttt{0xffffffff}, \texttt{0xffffffff}, \texttt{0xffffffff})$ are usually equal to those acquired with $\boldsymbol{IV} = \texttt{NULL}$. 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 [24], the 61-dimensional cube I (first appeared in [25]) 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 $\boldsymbol{IV} = \texttt{NULL}$), we identify the same J. By running Algorithm 2, 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. (8). They also proposed a 85-dimensional cube and mounted to 872 rounds ($|J| = 39$). By running Algorithm 2, 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. We construct a 102-dimensional cube finding whose superpoly has degree 2 at round 887 and 3 at round 888. The sizes of J's are 44 and 36 respectively. So this cube can be used for attacking 887-round Kreyvium with complexity $2^{115.80}$ and 888-round with complexity $2^{111.38}$. 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. (1).

Details of all our attacks are listed in Table 8.

B Application to Grain-128a

B.1 Specification of Grain-128a

Grain-128a is one of Grain family of NLFSR-based stream ciphers, and the internal state is represented by two 128-bit states, $(b_0, b_1, \ldots, b_{127})$ and $(s_0, s_1, \ldots, s_{127})$. The 128-bit key is loaded to the first register $\boldsymbol{b}$, and the 96-bit IV is loaded to

Table 8. Summary of theoretical cube attacks on Kreyvium. The time complexity in this table shows the time complexity to recover the superpoly.

#Rounds	$\|I\|$	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 ($\|J\| = 23$)	$2^{81.7}$
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 ($\|J\| = 39$)	$2^{94.61}$
887	102⋆	3	9, 10, 11, 23, 24, 25, 28, 33, 34, 35, 39, 42, 44, 46, 47, 48, 49, 50, 53, 56, 57, 59, 65, 68, 69, 70, 78, 79, 80, 82, 83, 84, 87, 90, 91, 92, 93, 94, 101, 103, 108, 115, 116, 118 ($\|J\| = 44$)	$2^{115.80}$
888	102⋆	2	7, 8, 9, 17, 18, 22, 32, 33, 41, 45, 46, 47, 48, 51, 52, 53, 55, 56, 57, 58, 68, 76, 77, 78, 79, 80, 81, 83, 84, 85, 91, 100, 114, 115, 116, 117 ($\|J\| = 36$)	$2^{111.38}$
891	113♠	2	19, 37, 46, 47, 62, 63, 64, 66, 71, 72, 73, 78, 91, 92, 93, 96, 106, 121, 122, 123 ($\|J\| = 20$)	$2^{120.73}$

†: $I =$ {1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 30, 32, 34, 37, 39, 41, 43, 45, 47, 49, 51, 53, 55, 58, 60, 62, 64, 66, 68, 70, 73, 75, 77, 79, 81, 83, 85, 87, 89, 91, 93, 95, 97, 99, 101, 103, 105, 108, 110, 112, 114, 116, 118, 120, 123, 125, 127}

‡: $I =$ {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36, 38, 40, 42, 44, 46, 48, 50, 52, 54, 56, 58, 60, 62, 64, 66, 68, 70, 72, 74, 76, 78, 80, 82, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 111, 113, 115, 117, 119, 121, 123, 125, 127}

⋆: $I =$ {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, 34, 36, 38, 40, 42, 44, 46, 48, 50, 52, 54, 56, 58, 60, 62, 64, 66, 68, 70, 72, 74, 76, 78, 80, 82, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128}

♠: $I =$ {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 23, 24, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 45, 46, 47, 48, 49, 50, 51, 53, 54, 55, 57, 58, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 101, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 115, 116, 117, 119, 121, 122, 124, 125, 126, 127, 128}

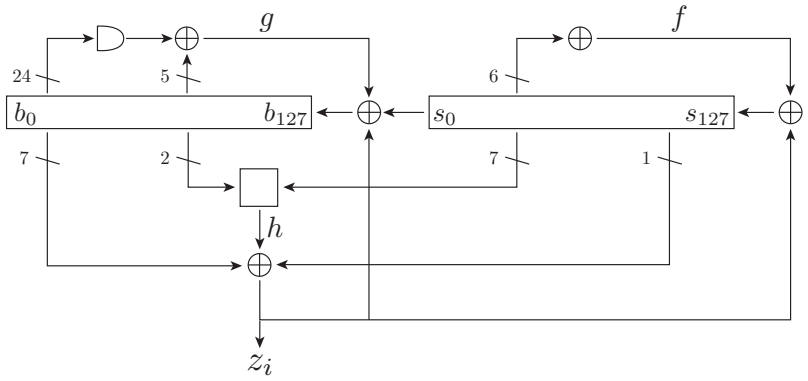

Fig. 3. Structure of Grain-128a

the second register $\boldsymbol{s}$. The other state bits are set to 1 except the least one bit in the second register. Namely, the initial state bits are represented as

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

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

$$\begin{aligned} g \leftarrow\ & b_0 + b_{26} + b_{56} + b_{91} + b_{96} \\ & + b_3 b_{67} + b_{11} b_{13} + b_{17} b_{18} + b_{27} b_{59} + b_{40} b_{48} + b_{61} b_{65} + b_{68} b_{84} \\ & + b_{88} b_{92} b_{93} b_{95} + b_{22} b_{24} b_{25} + b_{70} b_{78} b_{82}. \end{aligned} \tag{14}$$

$$f \leftarrow s_0 + s_7 + s_{38} + s_{70} + s_{81} + s_{96} \tag{15}$$

$$h \leftarrow b_{12} s_8 + s_{13} s_{20} + b_{95} s_{42} + s_{60} s_{79} + b_{12} b_{95} s_{94} \tag{16}$$

$$z \leftarrow h + s_{93} + \sum_{j \in A} b_j \tag{17}$$

$$(b_0, b_1, \ldots, b_{127}) \leftarrow (b_1, \ldots, b_{127}, g + s_0 + z)$$
$$(s_0, s_1, \ldots, s_{127}) \leftarrow (s_1, \ldots, s_{127}, f + z)$$

Here, $A = \{2, 15, 36, 45, 64, 73, 89\}$. First, in the key initialization, the state is updated 256 times without producing an output. After the key initialization, the update function is tweaked such that z is not fed to the state, and z is used as a key stream. Fig. 3 shows the state update function of Grain-128a.

B.2 MILP Model of Grain-128a

`Grain128aEval` in Algorithm 8 generates MILP model $\mathcal{M}$ of Algorithm 1, and the model $\mathcal{M}$ can evaluate all division trails for Grain-128a whose initialization rounds are reduced to R. `funcZ` generates MILP variables and constraints for Eq. (16) and Eq. (17). `funcG` generates MILP variables and constraints for Eq. (14). `funcF` generates MILP variables and constraints for Eq. (15). Compared with the description in [14], the number of constraints are not static due to the involvement of flags. The details of `funcZ`, `funcG`, and `funcF` are described in Algorithm 10. Note that the subroutine functions `CAND` and `CXOR` of `funcZ`, `funcG`, and `funcF` are just modeling the COPY+AND, COPY+XOR operations on many bits. We detail them in Algorithm 9.

B.3 Experimental Verification

We implemented the MILP model $\mathcal{M}$ for the propagation of the division property on Grain-128a and evaluated involved secret variables by using Algorithm 1. In order to show the advantage of our new model, we execute the same small cube as that of [14]: $I = \{1, 2, \ldots, 9\}$. We show that our new techniques can provide more information than the traditional method.

Algorithm 8 MILP model for the initialization of Grain-128a

procedure `Grain128aEval`(round R)
  Prepare empty MILP Model $\mathcal{M}$
  $\mathcal{M}.var \leftarrow v_i$ for $i \in \{1, 2, \ldots, 128\}$. ▷ Declare Public Modifiable IVs
  $\mathcal{M}.var \leftarrow x_i$ for $i \in \{1, 2, \ldots, 128\}$. ▷ Declare Secret Keys
  $\mathcal{M}.var \leftarrow b_i^0$ for $i \in \{0, 1, \ldots, 127\}$. ▷ Initial Division Property of NFSR
  $\mathcal{M}.var \leftarrow s_i^0$ for $i \in \{0, 1, \ldots, 127\}$. ▷ Initial Division Property of LFSR
  $\mathcal{M}.con \leftarrow b_i^0 = x_{i+1}$ for $i \in \{0, 1, \ldots, 127\}$.
  $\mathcal{M}.con \leftarrow s_i^0 = v_{i+1}$ for $i \in \{0, 1, \ldots, 95\}$.
  $\mathcal{M}.con \leftarrow s_i^0 = 0$ for $i \in \{96, \ldots, 127\}$.
  Initialize the flags of constant state bits: $s_i^0.F = 1_c$ for $i \in \{96, \ldots, 126\}$ and $s_{127}^0.F = 0_c$. ▷ Constant Bits are Initialized with the corresponding flags
  for $r = 1$ to R **do**
    $(\mathcal{M}, \boldsymbol{b}', \boldsymbol{s}', z) = \texttt{funcZ}(\mathcal{M}, \boldsymbol{b}^{r-1}, \boldsymbol{s}^{r-1})$
    $(\mathcal{M}, z_g, z_f) \leftarrow \texttt{copyf}(\mathcal{M}, z)$
    $(\mathcal{M}, \boldsymbol{b}'', g) = \texttt{funcG}(\mathcal{M}, \boldsymbol{b}')$
    $(\mathcal{M}, \boldsymbol{s}'', f) = \texttt{funcF}(\mathcal{M}, \boldsymbol{s}')$
    for $i = 0$ to 126 **do**
      $b_i^r = b_{i+1}''$
      $s_i^r = s_{i+1}''$
    end for
    $(\mathcal{M}, b_{127}^r) \leftarrow \texttt{xorf}(\mathcal{M}, g, s_0'', z_g)$
    $(\mathcal{M}, s_{127}^r) \leftarrow \texttt{xorf}(\mathcal{M}, f, z_f)$
  end for
  $(\mathcal{M}, \boldsymbol{b}', \boldsymbol{s}', z) = \texttt{funcZ}(\mathcal{M}, \boldsymbol{b}^R, \boldsymbol{s}^R)$
  for all $i \in \{0, 1, \ldots, 127\}$ **do**
    $\mathcal{M}.con \leftarrow b_i' = 0$
    $\mathcal{M}.con \leftarrow s_i' = 0$
  end for
  $\mathcal{M}.con \leftarrow z = 1$
  return $\mathcal{M}$
end procedure

Example 3 (Verification of Our Attack against 106-round Grain-128a). With $\boldsymbol{IV} = \texttt{NULL}$, we run Algorithm 1 and identify the involved key bits as $J = \{53, 85, 119, 122, 126, 127\}$. Compared with [14], K_{46} is excluded. We have tried thousands of non-cube IVs and none of them has superpoly related to K_{46}. Comparing our $\boldsymbol{IV} = \texttt{NULL}$ model with the previous method, we find that the only difference lies in the effect of flags imposed to the initial constant state bits $s_{96}^0, \ldots, s_{127}^0$. This experiment has proved the preciseness improvement of our new flag technique. By running Algorithm 2, we know the degree of the superpoly is 5 and simultaneously, we are able to acquire all possible 5-degree monomials: $x_{53}x_{119}x_{122}x_{126}x_{127}$ and $x_{85}x_{119}x_{122}x_{126}x_{127}$. These findings are in accordance with the experimentally recovered ANF of the superpoly

$$p_{\boldsymbol{v}}(\boldsymbol{x}) = (x_{53} + x_{85}) \cdot x_{119} \cdot (x_{122} + v_{76}) \cdot x_{126} \cdot x_{127},$$

which is given in [14].

Algorithm 9 MILP model for COPY+XOR and COPY+AND in Grain-128a

procedure $\texttt{CAND}(\mathcal{M}, \boldsymbol{b}, \boldsymbol{s}, I, J)$
  $(\mathcal{M}, b'_i, x_i) \leftarrow \texttt{copyf}(\mathcal{M}, b_i)$ for all $i \in I$
  $(\mathcal{M}, s'_j, y_j) \leftarrow \texttt{copyf}(\mathcal{M}, s_j)$ for all $j \in J$
  for all $i \in \{0, 1, \ldots, 127\} - I$ **do**
    $b'_i = b_i$
  end for
  for all $j \in \{0, 1, \ldots, 127\} - J$ **do**
    $s'_i = s_i$
  end for
  $(\mathcal{M}, z) \leftarrow \texttt{andf}(\mathcal{M}, {b'}_{i,i\in I}, {s'}_{j,j\in J})$
  return $(\mathcal{M}, \boldsymbol{b}', \boldsymbol{s}', z)$
end procedure

procedure $\texttt{CXOR}(\mathcal{M}, \boldsymbol{b}, \boldsymbol{s}, I, J)$
  $(\mathcal{M}, b'_i, x_i) \leftarrow \texttt{copyf}(\mathcal{M}, b_i)$ for all $i \in I$
  $(\mathcal{M}, s'_j, y_j) \leftarrow \texttt{copyf}(\mathcal{M}, s_j)$ for all $j \in J$
  for all $i \in \{0, 1, \ldots, 127\} - I$ **do**
    $b'_i = b_i$
  end for
  for all $j \in \{0, 1, \ldots, 127\} - J$ **do**
    $s'_i = s_i$
  end for
  $(\mathcal{M}, z) \leftarrow \texttt{xorf}(\mathcal{M}, {b'}_{i,i\in I}, {s'}_{j,j\in J})$
  return $(\mathcal{M}, \boldsymbol{b}', \boldsymbol{s}', z)$
end procedure

Algorithm 10 MILP model for NLFSR and LFSR in Grain-128a

procedure $\texttt{funcZ}(\mathcal{M}, \boldsymbol{b}, \boldsymbol{s})$
  $(\mathcal{M}, \boldsymbol{b}_1, \boldsymbol{s}_1, a_1) = \texttt{CAND}(\mathcal{M}, \boldsymbol{b}, \boldsymbol{s}, \{12\}, \{8\})$
  $(\mathcal{M}, \boldsymbol{b}_2, \boldsymbol{s}_2, a_2) = \texttt{CAND}(\mathcal{M}, \boldsymbol{b}_1, \boldsymbol{s}_1, \phi, \{13, 20\})$
  $(\mathcal{M}, \boldsymbol{b}_3, \boldsymbol{s}_3, a_3) = \texttt{CAND}(\mathcal{M}, \boldsymbol{b}_2, \boldsymbol{s}_2, \{95\}, \{42\})$
  $(\mathcal{M}, \boldsymbol{b}_4, \boldsymbol{s}_4, a_4) = \texttt{CAND}(\mathcal{M}, \boldsymbol{b}_3, \boldsymbol{s}_3, \phi, \{60, 79\})$
  $(\mathcal{M}, \boldsymbol{b}_5, \boldsymbol{s}_5, a_5) = \texttt{CAND}(\mathcal{M}, \boldsymbol{b}_4, \boldsymbol{s}_4, \{12, 95\}, \{94\})$
  $(\mathcal{M}, \boldsymbol{b}_6, \boldsymbol{s}_6, x) = \texttt{CXOR}(\mathcal{M}, \boldsymbol{b}_5, \boldsymbol{s}_5, \{2, 15, 36, 45, 64, 73, 89\}, \{93\})$
  $(\mathcal{M}, z) \leftarrow \texttt{xorf}(\mathcal{M}, x, a_1, \ldots, a_5)$
  return $(\mathcal{M}, \boldsymbol{b}_6, \boldsymbol{s}_6, z)$
end procedure

procedure $\texttt{funcF}(\mathcal{M}, \boldsymbol{s})$
  $(\mathcal{M}, \phi, \boldsymbol{s}_1, f) = \texttt{CXOR}(\mathcal{M}, \phi, \boldsymbol{s}, \phi, \{0, 7, 38, 70, 81, 96\})$
  return $(\mathcal{M}, \boldsymbol{s}_1, f)$
end procedure

procedure $\texttt{funcG}(\mathcal{M}, \boldsymbol{b})$
  $(\mathcal{M}, \boldsymbol{b}_1, \phi, a_1) = \texttt{CAND}(\mathcal{M}, \boldsymbol{b}, \phi, \{3, 67\}, \phi)$
  $(\mathcal{M}, \boldsymbol{b}_2, \phi, a_2) = \texttt{CAND}(\mathcal{M}, \boldsymbol{b}_1, \phi, \{11, 13\}, \phi)$
  $(\mathcal{M}, \boldsymbol{b}_3, \phi, a_3) = \texttt{CAND}(\mathcal{M}, \boldsymbol{b}_2, \phi, \{17, 18\}, \phi)$
  $(\mathcal{M}, \boldsymbol{b}_4, \phi, a_4) = \texttt{CAND}(\mathcal{M}, \boldsymbol{b}_3, \phi, \{27, 59\}, \phi)$
  $(\mathcal{M}, \boldsymbol{b}_5, \phi, a_5) = \texttt{CAND}(\mathcal{M}, \boldsymbol{b}_4, \phi, \{40, 48\}, \phi)$
  $(\mathcal{M}, \boldsymbol{b}_6, \phi, a_6) = \texttt{CAND}(\mathcal{M}, \boldsymbol{b}_5, \phi, \{61, 65\}, \phi)$
  $(\mathcal{M}, \boldsymbol{b}_7, \phi, a_7) = \texttt{CAND}(\mathcal{M}, \boldsymbol{b}_6, \phi, \{68, 84\}, \phi)$
  $(\mathcal{M}, \boldsymbol{b}_9, \phi, a_9) = \texttt{CAND}(\mathcal{M}, \boldsymbol{b}_8, \phi, \{22, 24, 25\}, \phi)$
  $(\mathcal{M}, \boldsymbol{b}_{10}, \phi, a_{10}) = \texttt{CAND}(\mathcal{M}, \boldsymbol{b}_9, \phi, \{70, 78, 82\}, \phi)$
  $(\mathcal{M}, \boldsymbol{b}_{11}, \phi, x) = \texttt{CXOR}(\mathcal{M}, \boldsymbol{b}_{10}, \phi, \{0, 26, 56, 91, 96\}, \phi)$
  $(\mathcal{M}, g) \leftarrow \texttt{xorf}(\mathcal{M}, x, a_1, \ldots, a_{10})$
  return $(\mathcal{M}, \boldsymbol{b}_{11}, g)$
end procedure

On the accuracy of MILP model with flag technique. We've tried 10000 random $\boldsymbol{IV}$ and the accuracy is 100% for Grain-128a: when the MILP degree evaluation gives $d = 5$, the superpoly is constantly a 5-degree polynomial; if $d = 0$, the superpoly is constant 0.

Table 9. Summary of theoretical cube attacks on Grain-128a.

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

† $I = \{1, ..., 40, 42, 44, \ldots, 51, 53, ..., 87, 89, 91, 93, 95\}$
‡ $I = \{1, ..., 51, 53, ..., 91, 93, 95\}$
∗ $I = \{1, ..., 46, 48, ..., 96\}$, and the non-cube IV bit $\boldsymbol{IV}[47] = 0$

B.4 Theoretical Results

We first revisit the previous attacks on 182- and 183-round attacks on Grain-128a using cube dimensions 88 and 92 respectively. For the 88-dimensional cube, we use our improved flag technique using Algorithm 1 and 8, we find that there are only $|J| = 14$ rather than 18 involved keys in the superpoly. Using Algorithm 2, 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 method 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 round 184, 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 $\boldsymbol{IV}[i] = 1$ and $\boldsymbol{IV}[i] = 0$. We run Algorithm 2 with I and the two different $\boldsymbol{IV}$'s. We find that when $\boldsymbol{IV}[47] = 1$, the degree of the superpoly is 19 and the degree drops to only 14 when $\boldsymbol{IV}[47] = 0$. This may indicate that many keys are no longer involved when $\boldsymbol{IV}[47] = 0$. So we run Algorithm 1 with I_{47} and $\boldsymbol{IV}[47] = 0$. Under such a setting, we have $|J| = 21$ and the attack is available with complexity $2^{115.95}$. We summarize our attacks as Table 9. We have lowered the complexities of previous cubes and improved the maximum attacked rounds by 1.

C Application to Acorn

C.1 Specification of Acorn

Acorn is an authenticated encryption and one of the 3rd round candidates in CAESAR competition. The structure is based on NLFSR, and the internal

state is represented by 293-bit state $(S_0, S_1, \ldots, S_{292})$. There are two component functions, $ks = KSG128(S)$ and $f = FBK128(S)$, in the update function, and each is defined as

$$ks = S_{12} \oplus S_{154} \oplus maj(S_{235}, S_{61}, S_{193}) \oplus ch(S_{230}, S_{111}, S_{66}),$$
$$f = S_0 \oplus \tilde{S}_{107} \oplus maj(S_{244}, S_{23}, S_{160}) \oplus (ca \wedge S_{196}) \oplus (cb \wedge ks),$$

where ks is used as the key stream, and maj and ch are defined as

$$maj(x, y, z) = (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z),$$
$$ch(x, y, z) = (x \wedge y) \oplus ((x \oplus 1) \wedge z).$$

Then, the update function is given as follows.

$$S_{289} \leftarrow S_{289} \oplus S_{235} \oplus S_{230}$$
$$S_{230} \leftarrow S_{230} \oplus S_{196} \oplus S_{193}$$
$$S_{193} \leftarrow S_{193} \oplus S_{160} \oplus S_{154}$$
$$S_{154} \leftarrow S_{154} \oplus S_{111} \oplus S_{107}$$
$$S_{107} \leftarrow S_{107} \oplus S_{66} \oplus S_{61}$$
$$S_{61} \leftarrow S_{61} \oplus S_{23} \oplus S_0$$
$$ks = KSG128(S)$$
$$f = FBK128(S, ca, cb)$$
$$(S_0, S_1, \ldots, S_{291}, S_{292}) \leftarrow (S_1, S_2, \ldots, S_{292}, f \oplus m)$$

The 293-bit state is first initialized to $\mathbf{0}$. Second, 128-bit secret key is sequentially loaded to the NLFSR via m. Third, 128-bit initialization vector is sequentially loaded to the NLFSR via m. Fourth, 128-bit secret key is sequentially loaded to the NLFSR via m twelve times. The constant bits ca and cb are always 1 in the initial 1792 rounds. The associated data is always loaded before the output of the key stream, but we do not care about this process in this paper because the number of rounds that we can attack is smaller than 1792 rounds. Fig. 4 shows the structure of Acorn. Please refer to [35] in detail.

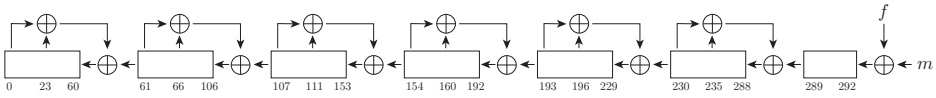

Fig. 4. Structure of Acorn

C.2 MILP Model of Acorn

In [14,24], the authors have already noticed the effect of constant 0's on the division propagation. As an example, they pointed out the situation that when two

Algorithm 11 MILP model for the initialization of ACORN

procedure `ACORNEval`(round R)
  Prepare empty MILP Model $\mathcal{M}$
  $\mathcal{M}.var \leftarrow K_i$ for $i \in \{1, 2, \ldots, 128\}$ as binary ▷ Declare the secret key bits
  $\mathcal{M}.var \leftarrow IV_i$ for $i \in \{1, 2, \ldots, 128\}$ as binary ▷ Declare the public IV bits
  $\mathcal{M}.var \leftarrow S_i^0$ for $i \in \{0, 1, \ldots, 292\}$ as binary and assign their flags as $S_i^0.F = 0_c$. ▷ The register bits are initialized as constant 0's
  for $r = 1$ to R **do**
    $(\mathcal{M}, \boldsymbol{T}) = \texttt{xorFB}(\mathcal{M}, \boldsymbol{S}^{r-1}, 289, 235, 230)$
    $(\mathcal{M}, \boldsymbol{U}) = \texttt{xorFB}(\mathcal{M}, \boldsymbol{T}, 230, 196, 193)$
    $(\mathcal{M}, \boldsymbol{V}) = \texttt{xorFB}(\mathcal{M}, \boldsymbol{U}, 193, 160, 154)$
    $(\mathcal{M}, \boldsymbol{W}) = \texttt{xorFB}(\mathcal{M}, \boldsymbol{V}, 154, 111, 107)$
    $(\mathcal{M}, \boldsymbol{X}) = \texttt{xorFB}(\mathcal{M}, \boldsymbol{W}, 107, 66, 61)$
    $(\mathcal{M}, \boldsymbol{Y}) = \texttt{xorFB}(\mathcal{M}, \boldsymbol{X}, 61, 23, 0)$
    $(\mathcal{M}, \boldsymbol{Z}, ks) = \texttt{ksg128}(\mathcal{M}, \boldsymbol{Y})$
    $(\mathcal{M}, \boldsymbol{A}, f) = \texttt{fbk128}(\mathcal{M}, \boldsymbol{Z}, ks)$
    for $i = 0$ to 291 **do**
      $S_i^r = A_{i+1}$
    end for
    $\mathcal{M}.var \leftarrow S_{292}^r$ as binary
    if $128 < r \leq 256$ **then**
      $\mathcal{M}.con \leftarrow S_{292}^r = f + IV_{r-128}$ and assign the flags $S_{292}^r.F = f.F + IV_{r-128}.F$
    else
      $\mathcal{M}.var \leftarrow TK_r$ as binary and assign its flag as $TK_r.F = K_{1+(r \mod 128)}.F$
      $\mathcal{M}.con \leftarrow S_{292}^r = f + TK_r$ and assign $S_{292}^r.F = f.F + TK^r.F$
    end if
  end for
  for $i = 0$ to 127 **do**
    $\mathcal{M}.con \leftarrow K_{i+1} = \sum_j TK_{i+128\times j}$
  end for
  $(\mathcal{M}, \boldsymbol{T}) = \texttt{xorFB}(\mathcal{M}, \boldsymbol{S}^R, 289, 235, 230)$
  $(\mathcal{M}, \boldsymbol{U}) = \texttt{xorFB}(\mathcal{M}, \boldsymbol{T}, 230, 196, 193)$
  $(\mathcal{M}, \boldsymbol{V}) = \texttt{xorFB}(\mathcal{M}, \boldsymbol{U}, 193, 160, 154)$
  $(\mathcal{M}, \boldsymbol{W}) = \texttt{xorFB}(\mathcal{M}, \boldsymbol{V}, 154, 111, 107)$
  $(\mathcal{M}, \boldsymbol{X}) = \texttt{xorFB}(\mathcal{M}, \boldsymbol{W}, 107, 66, 61)$
  $(\mathcal{M}, \boldsymbol{Y}) = \texttt{xorFB}(\mathcal{M}, \boldsymbol{X}, 61, 23, 0)$
  $(\mathcal{M}, \boldsymbol{Z}, ks) = \texttt{ksg128}(\mathcal{M}, \boldsymbol{Y})$
  for $i = 0$ to 292 **do**
    $\mathcal{M}.con \leftarrow Z_i = 0$
  end for
  $\mathcal{M}.con \leftarrow ks = 1$
  return $\mathcal{M}$
end procedure

inputs of maj are 0, the output is constant 0. This situation is properly handled with the flag technique. Furthermore, the flag technique has also handled other

situations caused by constant 0/1's, such as: if $y = z = 0/1$, $ch(x, y, z) = 0/1$. The details for handling such situations are given as Algorithm 12 and Algorithm 13. The MILP model for ACORN is given as `ACORNEval` in Algorithm 11. It can evaluate division trials for ACORN reduced to R rounds.

`xorFB` generates MILP variables and constraints for feed-back function with XOR. `ksg128` and `fbk128` generates MILP variables and constraints for $KSG128$ and $FBK128$, respectively. We detail `xorFB`, `ksg128` and `fbk128` as Algorithm 14, 15 and 16. Compared with the descriptions in [14], our algorithms have taken the effects of flags into account, so the results given by our algorithms are more precise than those of [14]. Note that Algorithm 11 is a subroutine of Algorithm 1 for generating the MILP model $\mathcal{M}$ and the constraints for the input division property of Algorithm 11 are imposed by Algorithm 1.

Algorithm 12 MILP model for maj in ACORN

procedure `maj`$(\mathcal{M}, \boldsymbol{X}, i, j, k)$
  if $X_i.F \oplus X_j.F = 0_c$ **then**
    $(\mathcal{M}, Y_i, a) \leftarrow$ `copyf`$(\mathcal{M}, X_i)$
    $(\mathcal{M}, Y_j, b) \leftarrow$ `copyf`$(\mathcal{M}, X_j)$
    $(\mathcal{M}, o) \leftarrow$ `andf`$(\mathcal{M}, a, b)$
    $Y_s = X_s$ for all $s \in \{0, \ldots, 292\} - \{i, j\}$
  else if $X_i.F \oplus X_k.F = 0_c$ **then**
    $(\mathcal{M}, Y_i, a) \leftarrow$ `copyf`$(\mathcal{M}, X_i)$
    $(\mathcal{M}, Y_k, c) \leftarrow$ `copyf`$(\mathcal{M}, X_k)$
    $(\mathcal{M}, o) \leftarrow$ `andf`$(\mathcal{M}, a, c)$
    $Y_s = X_s$ for all $s \in \{0, \ldots, 292\} - \{i, k\}$
  else if $X_j.F \oplus X_k.F = 0_c$ **then**
    $(\mathcal{M}, Y_j, b) \leftarrow$ `copyf`$(\mathcal{M}, X_j)$
    $(\mathcal{M}, Y_k, c) \leftarrow$ `copyf`$(\mathcal{M}, X_k)$
    $(\mathcal{M}, o) \leftarrow$ `andf`$(\mathcal{M}, b, c)$
    $Y_s = X_s$ for all $s \in \{0, \ldots, 292\} - \{j, k\}$
  else
    $(\mathcal{M}, Y_j, a_1, a_2) \leftarrow$ `copyf`$(\mathcal{M}, X_i)$
    $(\mathcal{M}, Y_j, b_1, b_2) \leftarrow$ `copyf`$(\mathcal{M}, X_j)$
    $(\mathcal{M}, Y_k, c_1, c_2) \leftarrow$ `copyf`$(\mathcal{M}, X_k)$
    $(\mathcal{M}, a) \leftarrow$ `andf`$(\mathcal{M}, a_1, b_1)$
    $(\mathcal{M}, b) \leftarrow$ `andf`$(\mathcal{M}, a_2, c_1)$
    $(\mathcal{M}, c) \leftarrow$ `andf`$(\mathcal{M}, b_2, c_2)$
    $(\mathcal{M}, o) \leftarrow$ `xorf`$(\mathcal{M}, a, b, c)$
    $Y_s = X_s$ for all $s \in \{0, \ldots, 292\} - \{i, j, k\}$
  end if
  return $(\mathcal{M}, \boldsymbol{Y}, o)$
end procedure

Algorithm 13 MILP model for *ch* in ACORN

procedure $\texttt{ch}(\mathcal{M}, \boldsymbol{X}, i, j, k)$
  if $X_i.F = 0_c$ or $X_j.F \oplus X_k.F = 0_c$ **then**
    $(\mathcal{M}, Y_k, o) \leftarrow \texttt{copyf}(\mathcal{M}, X_k)$
    $Y_s = X_s$ for all $s \in \{0, \ldots, 292\} - \{k\}$
  else if $X_i.F = 1_c$ **then**
    $(\mathcal{M}, Y_j, o) \leftarrow \texttt{copyf}(\mathcal{M}, X_j)$
    $Y_s = X_s$ for all $s \in \{0, \ldots, 292\} - \{j\}$
  else
    $(\mathcal{M}, Y_j, a_1, a_2) \leftarrow \texttt{copyf}(\mathcal{M}, X_i)$
    $(\mathcal{M}, Y_j, b_1) \leftarrow \texttt{copyf}(\mathcal{M}, X_j)$
    $(\mathcal{M}, Y_k, c, c_1) \leftarrow \texttt{copyf}(\mathcal{M}, X_k)$
    $(\mathcal{M}, a) \leftarrow \texttt{andf}(\mathcal{M}, a_1, b_1)$
    $(\mathcal{M}, b) \leftarrow \texttt{andf}(\mathcal{M}, a_2, c_1)$
    $(\mathcal{M}, o) \leftarrow \texttt{xorf}(\mathcal{M}, a, b, c)$
    $Y_s = X_s$ for all $s \in \{0, \ldots, 292\} - \{i, j, k\}$
  end if
  return $(\mathcal{M}, \boldsymbol{Y}, o)$
end procedure

Algorithm 14 MILP model for LFSR in ACORN

procedure $\texttt{xorFB}(\mathcal{M}, \boldsymbol{X}, i, j, k)$
  $(\mathcal{M}, Y_i, a) \leftarrow \texttt{copyf}(\mathcal{M}, X_i)$
  $(\mathcal{M}, Y_j, b) \leftarrow \texttt{copyf}(\mathcal{M}, X_j)$
  $(\mathcal{M}, Y_k) \leftarrow \texttt{xorf}(\mathcal{M}, a, b, X_k)$
  $Y_s = X_s$ for all $s \in \{1, \ldots, 293\} - \{i, j, k\}$
  return $(\mathcal{M}, \boldsymbol{Y})$
end procedure

Algorithm 15 MILP model for `ksg128` in ACORN

procedure $\texttt{ksg128}(\mathcal{M}, \boldsymbol{X})$
  $(\mathcal{M}, A_{12}, x_1) \leftarrow \texttt{copyf}(\mathcal{M}, X_{12})$
  $(\mathcal{M}, A_{154}, x_2) \leftarrow \texttt{copyf}(\mathcal{M}, X_{154})$
  $A_t = X_t$ for all $t \in \{0, \ldots, 292\} - \{12, 154\}$
  $(\mathcal{M}, \boldsymbol{B}, x_3) \leftarrow \texttt{maj}(\mathcal{M}, \boldsymbol{A}, 235, 61, 193)$
  $(\mathcal{M}, \boldsymbol{Y}, x_4) \leftarrow \texttt{ch}(\mathcal{M}, \boldsymbol{B}, 230, 111, 66)$
  $(\mathcal{M}, z) \leftarrow \texttt{xorf}(\mathcal{M}, x_1, x_2, x_3, x_4)$
  return $(\mathcal{M}, \boldsymbol{Y}, z)$
end procedure

Algorithm 16 MILP model for `fbk128` in ACORN

procedure `fbk128`$(\mathcal{M}, \boldsymbol{X}, ks)$
  $(\mathcal{M}, A_0, x_1) \leftarrow$ `copyf`$(\mathcal{M}, X_0)$
  $(\mathcal{M}, A_{107}, x_2) \leftarrow$ `copyf`$(\mathcal{M}, X_{107})$
  $(\mathcal{M}, A_{107}, x_3) \leftarrow$ `copyf`$(\mathcal{M}, X_{196})$
  $A_t = X_t$ for all $t \in \{0, \ldots, 292\} - \{0, 107, 196\}$
  $(\mathcal{M}, \boldsymbol{B}, x_4) \leftarrow$ `maj`$(\mathcal{M}, \boldsymbol{A}, 244, 23, 160)$
  $(\mathcal{M}, z) \leftarrow$ `xorf`$(\mathcal{M}, x_1, x_2, x_3, x_4, ks)$
  return $(\mathcal{M}, \boldsymbol{Y}, z)$
end procedure

C.3 Experimental Verification

We still use the cube $I = \{1, 2, 3, 4, 5, 8, 20, 125, 126, 127, 128\}$, and implement the attack on 517-round ACORN.

Example 4 (Verification of Our Attack against 517-round ACORN*).* We run Algorithm 1 with $\boldsymbol{IV} =$ `NULL` and acquire the same involved keys (K_6, K_8, K_{10}, K_{11}, K_{12}, K_{15}, K_{16}, K_{45}, and K_{49}) as [14]. By running Algorithm 2, we know the degree is 1 which is the same as the recovered superpoly

$$p_{\boldsymbol{IV}}(\boldsymbol{x}) = x_6 + x_8 + x_{10} + x_{11} + x_{12} + x_{15} + x_{16} + x_{45} + x_{49},$$

where different $\boldsymbol{IV}$ are used but the ANF remains unchanged. Both the MILP model and the practical experiments indicate that the corresponding superpoly involves the same key bits as above. This has not only proved the preciseness of the flag technique but also supported the rationality of Assumption 1 for ACORN.

On the accuracy of MILP model with flag technique. For 10000 random $\boldsymbol{IV}$, the accuracy is 100% for ACORN: when the MILP degree evaluation gives $d = 1$, the superpoly is constantly a 1-degree polynomial. There is no $\boldsymbol{IV}$ assignments making $d = 0$ and the experimentally recovered superpoly is constantly a linear expression.

C.4 Theoretical Results

We revisit the result in [14]. Using Algorithm 1, we find $|J| = 63$, 5 additional key bits are detected compared to [14] due to the flag technique. With the application of Algorithm 2, we deduce the degree of the superpoly as 7. So the complexity of this attack is $2^{93.23}$ according to Eq. (8), much lower than the previous 2^{122}. In order to attack more rounds, we construct a 96-dimensional cube that can mount to 743 rounds. The superpoly involves 72 key bits and the degree is 4. So the complexity of this attack is $2^{116.06}$. We also 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. (8). The detailed parameters of these attacks are listed in Table 10.

Table 10. Summary of theoretical cube attacks on ACORN. The time complexity in this table shows the time complexity of Phase 1 and Phase 2.

# Rounds	$\|I\|$	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 ($\|J\| = 63$)	$2^{93.23}$
743	96‡	4	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, 44, 45, 46, 49, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 69, 70, 72, 91, 93, 94, 95, 100, 103 ($\|J\| = 72$)	$2^{116.06}$
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 ($\|J\| = 81$)	$2^{125.71}$

†: I = {1, 2, ..., 32, 97,98, ..., 128}
‡: I = {1, 2, ..., 48, 81,82, ..., 128}
∗: I = {1, 2, 3, 4, 8, 10, 11, 12, 13, 14, 15, 16, 17, 18, 20, 22, 23, 24, 25, 27, 28, 29, 30, 31, 33, 34, 37, 39, 40, 42, 45, 46, 47, 48, 49, 50, 52, 53, 55, 56, 58, 59, 60, 61, 63, 64, 65, 66, 67, 68, 70, 71, 73, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 108, 109, 110, 111, 112, 113, 114, 115, 117, 118, 119, 120, 121, 122, 123, 126, 127}