Searching Cubes in Division Property Based Cube Attack: Applications to Round-Reduced ACORN

Jingchun Yang[1,2] and Dongdai Lin[1,2]

[1] State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China
[2] School of Cyber Security, University of Chinese Academy of Sciences, Beijing, China
{yangjingchun,ddlin}@iie.ac.cn

Abstract. Recently, division property based cube attack has acheived new progress and some cryptanalytic results against well-known stream ciphers. At EUROCRYPT 2020, Hao *et al.* proposed a new modeling method for three-subset division property without unknown subset. With this method, the exact expression of the superpoly in cube attack can be recovered.
In this paper, we propose a method to search good cubes for both distinguishing attacks and key recovery attacks in the division property based cube attack scenario. Our cube searching procedure is based on the algorithm of degree evaluation of the superpoly and the algorithm of superpoly recovery. In the process of cube searching, we mainly use the embedded property to narrow down the searching space. As a result, we find some new cube testers of dimension 126 on 775-round ACORN. We also find a new key recovery attack on 775-round ACORN with a 126-dimensional cube, whose corresponding superpoly is a 2-degree polynomial with respect to key bits.

Keywords: division property based cube attack · cube searching · degree evaluation · superpoly recovery · embedded property · ACORN.

1 Introduction

In symmetric-key cryptography, integral cryptanalysis [11] is one of general cryptanalytic techniques. The basic idea of integral attack is to traverse some active plaintext bits and check whether the summation of the corresponding ciphertext bits have zero-sum property.

Division property [15], proposed at EUROCRYPT 2015, is a generalization of the integral property. With division property, attackers can evaluate more accurate integral characteristics. The division property for a multiset of texts $\mathbb{X} \subseteq \mathbb{F}_2^n$ is defined by dividing a set of $\boldsymbol{u}$'s into two subsets: the *0-subset* of vectors $\boldsymbol{u} \in \mathbb{F}_2^n$ satisfy $\bigoplus_{\boldsymbol{x}\in\mathbb{X}} \boldsymbol{x}^{\boldsymbol{u}} = 0$, and the *unknown subset* satisfy the value of $\bigoplus_{\boldsymbol{x}\in\mathbb{X}} \boldsymbol{x}^{\boldsymbol{u}}$ is unknown. Moreover, the bit-based division property was introduced in [18], and three propagation rules for basic operations, `and`, `xor`, and `copy` are shown. While arbitrary block ciphers can be evaluated by using the bit-based division property, it requires much time and memory complexity [18]. In [24], Xiang *et al.* used the mixed integer linear programming (MILP) to model the propagation of the bit-based division property, and the propagation can be evaluated efficiently with the MILP solvers (e.g., Gurobi [5]).

Although (conventional) bit-based division property can find more accurate integral distinguishers, some practically verified integral distinguishers (e.g., [21]) cannot be proved with it. To find the exact integral characteristics, Todo and Morii [18] proposed the three-subset division property, and the set of vector $\boldsymbol{u}$ is divided into three subsets rather than two ones: the *0-subset*, the *unknown subset*, and the *1-subset* satisfying $\bigoplus_{\boldsymbol{x}\in\mathbb{X}} \boldsymbol{x}^{\boldsymbol{u}} = 1$. With the three-subset division property, the 15-round integral distinguisher of Simon32 was proved [18].

Cube attack, proposed by Dinur and Shamir [2] at EUROCRYPT 2009, can be seen as a type of higher-order differential (integral) attacks [12]. Given a cipher f with public variables $\boldsymbol{v} \in \mathbb{F}_2^m$ and secret variables $\boldsymbol{x} \in \mathbb{F}_2^n$, the cipher can be viewed as a polynomial $f(\boldsymbol{x},\boldsymbol{v})$. Denote a set of indices $I = \{i_1, i_2, \ldots, i_{|I|}\} \subset \{0, 1, \ldots, m-1\}$ by *cube indices.* Such an I determines a specific structure called *cube* C_I, which contains $2^{|I|}$ values where the variables in $\{v_{i_1}, v_{i_2}, \ldots, v_{i_{|I|}}\}$ take all possible combinations of values and the remaining variables (key and non-cube IV variables) are fixed to static values. Then the sum of f over all values of the cube C_I is

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

where $t_I = v_{i_1} \cdots v_{i_{|I|}}$, $p(\boldsymbol{x},\boldsymbol{v})$ is independent of $\{v_{i_1}, v_{i_2}, \ldots, v_{i_{|I|}}\}$, and $q(\boldsymbol{x},\boldsymbol{v})$ misses at least one variable from $\{v_{i_1}, v_{i_2}, \ldots, v_{i_{|I|}}\}$. Then, $p(\boldsymbol{x},\boldsymbol{v})$ is called the *superpoly* of the cube C_I. The cube attack consists of two phases. In offline phase, attackers select some cubes, and recover their corresponding superpolys. In online phase, attackers compute the value of the superpoly by summing the output bit $f(\boldsymbol{x},\boldsymbol{v})$ over the cube. If the expression of superpoly $p(\boldsymbol{x},\boldsymbol{v})$ is simple, then we get some simple equations from these superpolys, and we can immediately recover partial key information by solving the system of equations.

In [1] the notion of cube tester was proposed. The main idea of cube tester is similar to cube attack, however, it aims to find some superpolys which have the distinguishable properties (*e.g.*, the superpoly is equal to a constant, which is most commonly used).

For cube attack, recovering the superpoly plays an important role. Traditional approaches [2, 3] treat the cipher as a black-box polynomial, and attackers can test the low-degree property of the superpoly by executing the linearity or quadraticity tests. Then the ANF of the superpoly can be experimentally recovered by interpolation. However, the most time-consuming part is to calculate the cubesums, if the size of the cube is large (e.g., exceeds 40), the computation is practically infeasible.

Breakthroughs have been made by Todo *et al.* at CRYPTO 2017 [16] where they proposed the *division property based cube attack* to analyze the ANF of the superpoly. Based on the propagation of the (conventional) bit-based division property of stream ciphers, they gave a proposition to decide which key bits do not exist in the superpoly. By using high-dimensional cubes, they proposed the theoretical cube attacks on 832-round Trivium/183-round Grain128a/704-round ACORN/872-round Kreyvium [17]. Later at CRYPTO 2018, Wang *et al.* [20, 19] proposed some new techniques (flag technique, degree evaluation, and term enumeration) to improve the division property based cube attack, and new theoretical key recovery attacks were proposed.

Since the theory of conventional division property can not guarantee that the superpoly is non-constant, the key recovery attack may be degenerated to a distinguisher. As reported in [26, 22], the key recovery attack on 839-round Trivium [20] is in fact a zero-sum distinguisher. Recently, several methods have been proposed to model for the three-subset version. In [9], a variant of the three-subset division property was proposed. Using MILP to model propagation rules, it improves some integral distinguishers at the cost of some accuracy loss. In [22], Wang *et al.* proposed a method to accurately model the propagation for the three-subset division property. By combining the MILP with the original breadth-first search algorithm [18], they could recover some superpolies for 832- and 839-round Trivium. However, as pointed in [8], this method is not always efficient, it requires an assumption that almost all elements in 1-subset can be pruned.

Very recently, at EUROCRYPT 2020, Hao *et al.* [8] proposed a new modeling method for the three-subset division property without unknown subset. They introduced a modified three-subset division property that is equivalent with the three-subset division property without unknown subset. This modified version is defined by using the multiset of the 1-subset, and it is suited to modeling with MILP or SAT/SMT solvers. By counting all feasible solutions that are enumerated, this new method can recover the superpoly efficiently. As a result, the exact expression of superpolies on 842-round Trivium/190-round Grain-128AEAD/774-round ACORN/892-round Kreyvium can be determined [7].

Our Contributions. In this paper, we propose a method to search good cubes for distinguishing attacks and key recovery attacks in the division property based cube attack. Our cube searching procedure is based on the algorithm of degree evaluation of the superpoly [20] and the algorithm of superpoly recovery [8]. In the process of cube searching, we mainly use the embedded property to narrow down the searching space. Using this method, we give some new cryptanalytic results on ACORN cipher [23].

Our cube searching procedure consists of two phases. In the first phase, we search a set of good cube testers in the *conventional division property based cube attack*. We first demonstrate that the embedded property [14] proposed at ASIACRYPT 2017 can also be applied to the search of cube testers. Based on the embedded property for cube tester, we propose several algorithms to search good cube testers efficiently.

Since the conventional division property based cube attack can not guarantee that the superpoly is non-constant, the key recovery attack may be degenerated to a distinguisher. In the second phase, we use the *modified three-subset division property based cube attack* to improve the cube testers, and to search cubes for key recovery attack. In the process of cube searching, we try to maximize the number of attacked arounds under the condition that the estimated degree of the superpoly is relatively low. Meanwhile, we keep the cube dimension as small as possible.

We apply our cube searching method to ACORN. As a result, we find some new cube testers for 775-round ACORN with a complexity of 2^{126}. We also find a key recovery attack on 775-round ACORN with a 126-dimensional cube, whose corresponding superpoly is a 2-degree polynomial with respect to key bits. Our results for ACORN are summarized in Table 1, and the comparisions with previous attacks are also included.

Table 1. Summary of the distinguishing attacks and key recovery attacks on ACORN.

Attack types	#Rounds	Cube size	Complexity	Ref.
distinguishing attacks	676	29	$\approx 2^{40.6}$	[4]
	690	38	2^{38}	[25]
	700	40	2^{40}	[10]
	706	46	2^{46}	[25]
	775	127	2^{127}	[25]
	775	126	2^{126}	Section 4.3
key recovery attacks	503	5	practical	[13]
	704	64	$2^{64} + 2^{127}$	[17]
	750	101	$2^{101} + 2^{127}$	[19]
	763	116	$2^{116} + 2^{127}$	[6]
	772	123	$2^{123} + 2^{127}$ †	[25]
	773	125	$2^{125} + 2^{127}$	[7]
	774	126	$2^{126} + 2^{127}$	[7]
	775	126	$2^{126} + 2^{127}$	Section 4.4

† This attack has been proved to be a constant-sum distinguisher by Hao *et al.* [7] using the modified three-subset division property based cube attack.

Organization. The rest of the paper is organized as follows. In Section 2 we introduce some basic definitions and theories. In Section 3, we propose several algorithms to search good cube testers and suitable cubes for key recovery attacks in the division property based cube attack. Then we use these algorithms to analyze the security of ACORN in Section 4. Section 5 concludes the paper.

2 Preliminaries

2.1 Boolean Function and Algebraic Degree

Let $\mathbb{F}_2$ denote the binary field and $\mathbb{F}_2^n$ denote the n-dimensional vector space over $\mathbb{F}_2$. A Boolean function f is a mapping from $\mathbb{F}_2^n$ to $\mathbb{F}_2$. In general, Algebraic Normal Form (ANF) is used to represent a Boolean function. An n-variable Boolean function f can be uniquely represented as

$$f(x_1, x_2, \cdots, x_n) = \bigoplus_{u=(u_1,\cdots,u_n)\in\mathbb{F}_2^n} a_u^f \prod_{i=1}^{n} x_i^{u_i},\ a_u^f \in \mathbb{F}_2,$$

where a_u^f is a constant depending on f and u. The algebraic degree of f is defined as $\deg(f) = \max\{wt(u) \mid a_u^f \neq 0\}$, where $wt(u)$ is the Hamming weight of u.

2.2 Cube Attack and Cube Tester

Cube attack, proposed by Dinur and Shamir [2] at EUROCRYPT 2009, can be seen as a type of higher-order differential (integral) attacks [12]. Given a cipher f with public variables $\boldsymbol{v} \in \mathbb{F}_2^m$ and secret variables $\boldsymbol{x} \in \mathbb{F}_2^n$, the cipher can be viewed as a polynomial $f(\boldsymbol{x}, \boldsymbol{v})$. Denote a set of indices $I = \{i_1, i_2, \ldots, i_{|I|}\} \subset \{0, 1, \ldots, m-1\}$ by *cube indices*. Such an I determines a specific structure called *cube* C_I, which contains $2^{|I|}$ values where the variables in $\{v_{i_1}, v_{i_2}, \ldots, v_{i_{|I|}}\}$ take all possible combinations of values and the remaining variables (key and non-cube IV variables) are fixed to static values. Then the sum of f over all values of the cube C_I is

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

where $t_I = v_{i_1} \cdots v_{i_{|I|}}$, $p(\boldsymbol{x}, \boldsymbol{v})$ is independent of $\{v_{i_1}, v_{i_2}, \ldots, v_{i_{|I|}}\}$, and $q(\boldsymbol{x}, \boldsymbol{v})$ misses at least one variable from $\{v_{i_1}, v_{i_2}, \ldots, v_{i_{|I|}}\}$. Then, $p(\boldsymbol{x}, \boldsymbol{v})$ is called the *superpoly* of the cube C_I. The cube attack consists of two phases. In offline phase, attackers select some cubes, and recover their corresponding superpolys. In online phase, attackers compute the value of the superpoly by summing the output bit $f(\boldsymbol{x}, \boldsymbol{v})$ over the cube. If the expression of superpoly $p(\boldsymbol{x}, \boldsymbol{v})$ is simple, then we get some simple equations from these superpolys, and we can immediately recover partial key information by solving the system of equations.

In [1] the notion of cube tester was proposed. The main idea of cube tester is similar to cube attack, however, it aims to find some superpolys which have the distinguishable properties (*e.g.*, the superpoly is equal to a constant, which is most commonly used).

2.3 Conventional Division Property and its MILP Representation

The (conventional) division property, proposed at EUROCRYPT 2015 [15], is a generalization of the integral property for the detection of better integral characteristics for word-oriented cryptographic primitives. Moreover, the bit-based version was proposed in [18] to describe the propagation of integral characteristics for bit-oriented ciphers. The bit-based division property is defined as follows.

Definition 1 ((Bit-Based) Division Property [18]). *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 fulfills the following conditions:*

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

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

In [15, 18], the propagation rules are provided when the bitwise operations COPY, XOR, AND are applied to the elements in $\mathbb{X}$. To evaluate the propagation of bit-based division property, attackers determine indices $I = \{i_1, i_2, \ldots, i_{|I|}\}$ and prepare $2^{|I|}$ chosen plaintexts (IVs for stream ciphers) where variables indexed by I are taking all possible combinations of values. If the division property of the corresponding ciphertexts (keystream for stream ciphers) does not contain a unit vector $\boldsymbol{e}_i$ whose only i-th element is 1, the i-th bit of the r-round ciphertexts is balanced.

Represent the Propagation of Division Property using MILP. At ASIACRYPT 2016, Xiang *et al.* [24] first introduced a new concept *division trail* to describe the propagation of the division property, and showed that the basic propagation rules of the division property can be translated as some variables and constraints of an MILP model. With this method, all possible division trails can be covered with an MILP model $\mathcal{M}$ and the division property of some output bit can be known according to the solutions of $\mathcal{M}$.

Definition 2 (Division Trail [24]). *Let us consider the propagation of the division property $\{\boldsymbol{k}\} \overset{def}{=} \mathbb{K}_0 \to \mathbb{K}_1 \to \mathbb{K}_2 \to \cdots \to \mathbb{K}_r$. 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 division property propagation rules. Furthermore, for $(\boldsymbol{k}_0, \boldsymbol{k}_1, \cdots, \boldsymbol{k}_r) \in (\mathbb{K}_0 \times \mathbb{K}_1 \times \cdots \times \mathbb{K}_r)$, if $\boldsymbol{k}_i$ can propagate to $\boldsymbol{k}_{i+1}$ for all $i \in \{0, 1, \cdots, r-1\}$, 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 r-round iterated cipher. If there is a division trail $\boldsymbol{k}_0 \overset{E_k}{\to} \boldsymbol{k}_r = \boldsymbol{e}_j (j = 1, \ldots, n)$, the summation of j-th bit is unknown; otherwise, if there is no division trail *s.t.* $\boldsymbol{k}_0 \overset{E_k}{\to} \boldsymbol{k}_r = \boldsymbol{e}_j$, we know the j-th bit of the ciphertext is balanced.

2.4 Division Property and Cube Attack

In cube attack, we want to recover the superpoly $p(\boldsymbol{x}, \boldsymbol{v})$. Let $x_0, x_1, \ldots, x_{n-1}$ be all key bits. If the initialization is not enough for thorough diffusion, the superpoly may only be related to a part of key bits $J \subsetneq \{0, 1, \cdots, n-1\}$. At CRYPTO 2017, Todo *et al.* [16] proposed an algorithm for determining such a set J by using the bit-based division property. This algorithm is based on the following proposition.

Proposition 1 (Key bits that are not involved in the superpoly [16]). *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 $I = \{i_1, i_2, \ldots, i_{|I|}\} \subset \{0, 1, \ldots, m-1\}$, 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 $\boldsymbol{v}^{\boldsymbol{k}_I} = t_I = v_{i_1} v_{i_2} \ldots v_{i_{|I|}}$, 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.*

According to this proposition, one can check that whether there is a division trail $(\boldsymbol{e}_\lambda, \boldsymbol{k}_I) \xrightarrow{f} 1$ for $\lambda = 0, 1, \ldots, n-1$ via the MILP modeling method. If the division trail $(\boldsymbol{e}_\lambda, \boldsymbol{k}_I) \xrightarrow{f} 1$ exists, then $\lambda \in J$; otherwise, $\lambda \notin J$. With the knowledge of J, one can recover the superpoly by computing its truth table in offline phase.

Later at CRYPTO 2018, Wang *et al.* [20, 19] proposed some techniques to improve the division property based cube attack. The main contribution of [19] can be summarized as follows.

Flag Technique. In previous MILP modeling of the bitwise operations COPY, XOR, AND, each intermediate state bit b is assigned a binary value $b.val$ to represent its bit-based division property value. In [19], Wang *et al.* added a 'flag' value for each state bit. The flag value $b.F$ can be $0_c, 1_c$ or δ to indicate whether the state bit is constant 0, constant 1 or variable. This change mainly affects the MILP model for AND. If the flag value $b.F$ of state bit b is 0_c, then we add a constraint $b.val = 0$, thus may improve the accuracy of MILP model description of the division property propagation. With flag technique, the new MILP model for COPY, XOR, AND are called `copyf`, `xorf`, `andf`. We refer to [19] for more details.

Degree Evaluation and Term Enumeration. To recover the superpoly more efficiently, Wang *et al.* [19] proposed another two algorithms to compute the algebraic degree and enumerate all possible terms of the superpoly, respectively. The two algorithms are based on the following proposition, which is actually a generalization of Proposition 1.

Proposition 2 (Degree evaluation and term enumeration of the superpoly [19]). *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 $I = \{i_1, i_2, \ldots, i_{|I|}\} \subset$*

$\{0, 1, \ldots, m-1\}$, *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* $\boldsymbol{v}^{\boldsymbol{k}_I} = t_I = v_{i_1} v_{i_2} \ldots v_{i_{|I|}}$. *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 term* $x^{\boldsymbol{k}_\Lambda}$ *is not involved in the superpoly of the cube* C_I.

The complexity of the superpoly recovery can be reduced from $2^{|I|+|J|}$ to $2^{|I|} \times (1 + \sum_{t=1}^{d} |J_t|)$, where I is the cube indices, J is the involved key bits in the superpoly, d is the algebraic degree of the superpoly, and J_t is all possible terms of degree t. In the rest of this paper, we denote the algorithm of the degree evaluation of the superpoly [19] by Algorithm $\mathtt{A}$.

2.5 Three-Subset Division Property and its Propagation Rules

Although (conventional) bit-based division property can find more accurate integral distinguishers, some practically verified integral distinguishers (e.g., [21]) cannot be proved with it. To find the exact integral characteristics, Todo and Morii [18] proposed the three-subset division property.

Definition 3 (Three-Subset Division Property [18]). *Let* $\mathbb{X}$ *be a multiset whose elements take a value of* $\mathbb{F}_2^n$. *Let* $\mathbb{K}, \mathbb{L}$ *be the set whose elements take an* n*-dimensional bit vector. When the multiset* $\mathbb{X}$ *has the three-subset division property* $\mathcal{D}^{1^n}_{\mathbb{K},\mathbb{L}}$, *it fulfills the following conditions:*

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

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

Propagation Rules for Three-Subset Division Property. In [18], the propagation rules of the three-subset division property for the bitwise operations COPY, AND, XOR are given.

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

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

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

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

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

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

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

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

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

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

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

$$\mathbb{L}' \overset{x}{\leftarrow} (l[1]+l[2], l[3], l[4], \dots, l[n]).$$

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

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

2.6 Three-Subset Division Property and Cube Attack

In [16], the authors assume $a^f_{\boldsymbol{u}} = 1$ when the division property $\mathcal{D}^{1^n}_{\boldsymbol{u}}$ can propagate to $\mathcal{D}^1_1$. When these assumptions do not hold, the superpoly can be much simpler than estimated, and in the extreme case, the superpoly becomes a constant function. Then, the key recovery attack degenerates into the distinguishing attack. In [22], the authors proposed proposition 4 to remove these assumptions by using three-subset division property. A simplified version is shown below.

Lemma 1 (Simple case of [22]). *Let $f(x)$ be a polynomial from $\mathbb{F}_2^n$ to $\mathbb{F}_2$ and $a^f_{\boldsymbol{u}} \in \mathbb{F}_2$ ($\boldsymbol{u} \in \mathbb{F}_2^n$) be the ANF coefficients. Let $\boldsymbol{l}$ be an n-dimensional bitvector. Then, assuming that the initial division property $\mathcal{D}^{1^n}_{\phi,\{\boldsymbol{l}\}}$ propagates to $\mathcal{D}^1_{\phi,1}$ after evaluating the function f, $a^f_{\boldsymbol{l}} = 1$.*

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

2.7 Three-Subset Division Property w/o Unknown Subset

In [22], Wang *et al.* proposed a method to accurately model the propagation for the three-subset division property. However, as pointed in [8], this method is not always efficient, it requires an assumption that almost all elements in 1-subset can be pruned. To address this problem, Hao *et al.* [8] proposed a new modeling method for the three-subset division property without unknown subset. They introduced a modified three-subset division property that is equivalent with the three-subset division property without unknown subset.

Definition 4 (Modified Three-Subset Division Property [8]). *Let $\mathbb{X}$ be a multiset whose elements take a value of $\mathbb{F}_2^n$. Let $\tilde{\mathbb{L}}$ be also a* ***multiset*** *whose elements take a value of $\mathbb{F}_2^n$. When the multiset $\mathbb{X}$ has the modified three-subset division property $\mathcal{T}_{\tilde{\mathbb{L}}}^{1^n}$, it fulfills the following conditions:*

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

where $\boldsymbol{x}^{\boldsymbol{u}} = \prod_{i=1}^{n} {x_i}^{u_i}$.

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

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

$$\tilde{\mathbb{L}}' \leftarrow (l[1] + l[2], l[3], l[4], \dots, l[n]) .$$

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

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

Based on this modified three-subset division property and the new propagation rules, Hao *et al.* [8] proposed its MILP models and Algorithm 2 to recover the superpoly. We denote the new MILP models for COPY, XOR, AND by `copyt`, `xort`, `andt`. We refer to [8] for more details. In the rest of this paper, we denote the algorithm of superpoly recovery [8] by Algorithm `B`.

3 Searching Cubes in Division Property Based Cube Attack

In this section, we propose a method to search good cubes for an iterated cipher in the division property based cube attack. Our cube searching procedure consists of two phases. In the first phase, we use conventional division property based cube attack to search a set of good cube testers. In the second phase, we use the modified three-subset division property based cube attack to improve the cube testers, and to search cubes for key recovery attack.

3.1 Phase 1: Searching Cube Testers using Conventional Division Property

For conventional bit-based division property, there is an embedded property [14] reflecting the features of its propagation. The embedded property says that, for different initial division properties $\boldsymbol{k}_0$ and $\boldsymbol{k}_1$ *s.t.* $\boldsymbol{k}_0 \succeq \boldsymbol{k}_1$, there is no need to test $\boldsymbol{k}_1$, if the output multi-set under $\boldsymbol{k}_0$ does not have integral property, likewise, it is not necessary to test $\boldsymbol{k}_0$, if the output multi-set under $\boldsymbol{k}_1$ has integral property.

In the following, we will show that, the embedded property can apply to the search of cube testers as well. We first introduce the following lemma which was proposed in [19].

Lemma 2 ([19]). *If $\boldsymbol{k} \succeq \boldsymbol{k}'$ and there is division trail $\boldsymbol{k} \xrightarrow{f} \boldsymbol{l}$, then there is also division trail $\boldsymbol{k}' \xrightarrow{f} \boldsymbol{l}'$ s.t. $\boldsymbol{l} \succeq \boldsymbol{l}'$.*

Based on this lemma, we propose the following proposition.

Proposition 3. *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 $I = \{i_1, i_2, \ldots, i_{|I|}\} \subset \{0, 1, \ldots, m-1\}$, let $\boldsymbol{k}_I$ be an m-dimensional bit vector such that $\boldsymbol{v}^{\boldsymbol{k}_I} = t_I = v_{i_1} v_{i_2} \ldots v_{i_{|I|}}$. Let $\boldsymbol{k}_\Lambda$ be an n-dimensional bit vector. For a given set of indices $I_S \subsetneq I$, if there is no division trail such that $(\boldsymbol{k}_\Lambda || \boldsymbol{k}_{I_S}) \xrightarrow{f} 1$ for any $\boldsymbol{k}_\Lambda \in \mathbb{F}_2^n$, then there is also no division trail such that $(\boldsymbol{k}_\Lambda || \boldsymbol{k}_I) \xrightarrow{f} 1$ for any $\boldsymbol{k}_\Lambda \in \mathbb{F}_2^n$.*

Proof. From Lemma 2, if $\boldsymbol{k} \succeq \boldsymbol{k}'$ and there is division trail $\boldsymbol{k} \xrightarrow{f} 1$, then there is also division trail $\boldsymbol{k}' \xrightarrow{f} 1$. Suppose there is a division trail such that $(\boldsymbol{k}_\Lambda^* || \boldsymbol{k}_I) \xrightarrow{f} 1$ for a fixed $\boldsymbol{k}_\Lambda^* \in \mathbb{F}_2^n$, then there is also a division trail such that $(\boldsymbol{k}_\Lambda^* || \boldsymbol{k}_{I_S}) \xrightarrow{f} 1$ (since $(\boldsymbol{k}_\Lambda^* || \boldsymbol{k}_I) \succeq (\boldsymbol{k}_\Lambda^* || \boldsymbol{k}_{I_S})$), which leads to a contradiction. □

Denote the superpoly of cube I by $p_S(I)$. From the above Proposition and Proposition 2 in Section 2, we know that,

Proposition 4 (Embedded Property for Cube Tester). *For an r-round iterated cipher, if I_S is a subset of cube I, and there is no monomials in $p_S(I_S)$ (*i.e.*, $\deg(p_S(I_S)) = 0$), then there is also no monomials in $p_S(I)$ (*i.e.*, $\deg(p_S(I)) = 0$). Likewise, if $\deg(p_S(I)) \neq 0$, then $\deg(p_S(I_S)) \neq 0$.*

Proposition 4 is useful to search cube testers efficiently in the conventional division property based cube attack.

In the following, we propose Algorithm 1, 2, and 3 to efficiently reduce the complexity of searching. Algorithm 1 is used to determine the maximum number of distinguishable rounds r_m for a given cube indices I. Algorithm 1 can be seen as a subfunction of Algorithm 2. In Algorithm 2, we determine the maximum number of distinguishable rounds for a specific cipher, and restrict the search scope. In Algorithm 3, we use the output of Algorithm 2 as input, and returns a set of constant-sum cube testers.

Algorithm 1 Determining the Maximum Number of Distinguishable Rounds for a Given Cube

```
1: procedure DetermineMaximumRoundsForCube(Given iterated cipher $f$ with
   $R$ initialzation rounds, cube indices $I$, return the maximum number of distinguish-
   able rounds $r_m$ for cube $I$.)
2:     $r_h = R,\ r_l = 0,\ r_m = 0,\ r = 0,\ flag = 0$;
3:     while $r_h - r_l > 1$ do
4:         $r = \lfloor (r_h + r_l)/2 \rfloor$
5:         use Algorithm A to evaluate the degree $d$ of the superpoly of cube $I$ for $f$
   reduced to $r$ rounds;
6:         if $d = 0$ then
7:             $r_l = r$, $flag = 0$;
8:         else
9:             $r_h = r$, $flag = 1$;
10:        end if
11:    end while
12:    if $flag = 0$ then
13:        $r_m = r$;
14:    else
15:        $r_m = r - 1$;
16:    end if
17:    return $r_m$;
18: end procedure
```

The basic idea of Algorithm 1 is binary search, which can reduce the complexity of searching. In Algorithm 1, we set two variables r_h and r_l to indicate the upper bound and lower bound of the maximum number of distinguishable rounds r_m for a specific cipher. For a cipher f with R initialzation rounds, we first use Algorithm A to evaluate the degree d of the superpoly of cube I for f reduced to $\lfloor (r_h + r_l)/2 \rfloor = \lfloor R/2 \rfloor$ rounds. If $d = 0$, then r_m is at least $\lfloor R/2 \rfloor$, so we set $r_l = r$. Otherwise, we set $r_h = r$. We iteratively repeat this process, so the distance between r_h and r_l can be reduced quickly. In the end, we can determine the value of r_m with at most $\lceil \log_2 R \rceil$ iterations.

In Algorithm 2, we first check all cubes of dimension $m - 1$, where m is the number of public variables. For each $(m - 1)$-dimensional cube I, we use

Algorithm 2 Determining the Maximum Number of Distinguishable Rounds & Restricting the Search Scope

```
1: procedure DetermineMaxRoundsAndRestrictScope(Given iterated cipher $f$ with $m$ public variables $(v_0, \dots, v_{m-1})$, return the maximum number of distinguishable rounds $r_{max}$ of cube testers, and the index set $\mathbb{S}$.)
2:     $r_{max} = 0,\ \mathbb{S} = \emptyset$;
3:     for $i = 0$; $i < m$ do
4:         let cube indices $I_i = \{0, 1, \dots, m-1\} \setminus \{i\}$;
5:         use Algorithm 1 to compute the maximum number of distinguishable rounds $r_i$ for cube $I_i$, and store $(I_i, r_i)$;
6:         if $r_{max} < r_i$ then
7:             $r_{max} = r_i$;
8:         end if
9:     end for
10:    for $i = 0$; $i < m$ do
11:        if $r_i = r_{max}$ then
12:            $\mathbb{S} = \mathbb{S} \cup \{i\}$;
13:        end if
14:    end for
15:    return $r_{max}, \mathbb{S}$;
16: end procedure
```

Algorithm 1 to compute its maximum number of distinguishable rounds as cube testers. Among all m cubes, we select those cubes which can lead to the longest (r_{max}-round) cube tester, and store their missing index i of public variables in $\mathbb{S}$. We claim that the elements in the complementary set $\bar{\mathbb{S}} = \{0, 1, \dots, m-1\} \setminus \mathbb{S}$ of $\mathbb{S}$ are 'necessary' bit indices to obtain an r_{max}-round cube tester. By Proposition 4, if any index which belongs to $\bar{\mathbb{S}}$ is not in cube indices I, then this cube will not lead to an r_{max}-round cube tester. In the following, we call $\bar{\mathbb{S}}$ the *necessary set*, whose elements must be in the cube indices, while $\mathbb{S}$ is called the *sufficient set*, and the elements in $\mathbb{S}$ are called *sufficient indices*.

In Algorithm 3, we first test whether the cube $I = \{0, 1, \dots, m-1\} \setminus \mathbb{S}$ will lead to the r_{max}-round cube tester. If not, we gradually increase the dimension of cubes by reducing the value of t where we pick t indices from $\mathbb{S}$, and check whether the cube tester exists or not. After t is fixed (Line 15 in Algorithm 3), there exists at least one cube which will lead to the r_{max}-round cube tester, so Algorithm 3 returns a set of constant-sum cube testers.

3.2 Phase 2: Searching Cubes using Modified Three-Subset Division Property

In phase 2, we propose Algorithm 4 to search more cubes for improved distinguishers and key recovery attack. Given r-round cipher f with public variables $(v_0, \dots, v_{m-1})$ and secret variables $(x_0, \dots, x_{n-1})$, Algorithm 4 returns a set $\mathbb{D}$ containing the cubes for improved distinguishers, and a set $\mathbb{K}$ containing the cubes for key recovery attack. Moreover, the maximum number of

Algorithm 3 Searching Constant-Sum Cube Testers

```
1: procedure SearchConstantSumCubeTesters(Given iterated cipher $f$ with $m$
   public variables $(v_0, \ldots, v_{m-1})$, the maximum number of distinguishable rounds
   $r_{max}$ of cube testers, and the sufficient set $\mathbb{S}$, return a set $Res$ containing the
   constant-sum cube testers.)
2:     $Res = \emptyset$,  $flag = 0$;
3:     $t = |\mathbb{S}|$;
4:     while $flag = 0$ do
5:         for every $t$-tuple $(i_0, i_1, \ldots, i_{t-1})$ of $\mathbb{S}$ do
6:             let cube indices $I = \{0, 1, \ldots, m-1\} \setminus \{i_0, i_1, \ldots, i_{t-1}\}$;
7:             use Algorithm A to evaluate the degree $d$ of the superpoly of cube $I$ for
   $f$ reduced to $r_{max}$ rounds;
8:             if $d = 0$ then
9:                 $flag = 1$;
10:                break;
11:            end if
12:        end for
13:        $t = t - 1$;
14:    end while
15:    $t = t + 1$;
16:    for every $t$-tuple $(i_0, i_1, \ldots, i_{t-1})$ of $\mathbb{S}$ do
17:        let cube indices $I = \{0, 1, \ldots, m-1\} \setminus \{i_0, i_1, \ldots, i_{t-1}\}$;
18:        use Algorithm A to evaluate the degree $d$ of the superpoly of cube $I$ for $f$
   reduced to $r_{max}$ rounds;
19:        if $d = 0$ then
20:            $Res = Res \cup \{I\}$;
21:        end if
22:    end for
23:    return $Res$;
24: end procedure
```

distinguishable rounds r_{max} and the sufficient set $\mathbb{S}$ returned by Algorithm 2, and the set Res containing the constant-sum cube testers returned by Algorithm 3 are also taken as the input of Algorithm 4, since we can use them to restrict our search space.

Since the theory of conventional division property can not guarantee that the superpoly is non-constant, the return value of Algorithm `A` is actually an upper bound of the real degree of the superpoly. On the other hand, if the return value of Algorithm `A` is zero, then a constant superpoly is guaranteed. As the modified three-subset division property based cube attack can recover the exact expression of superpolys, we use Algorithm `B` to verify more cubes whose superpoly has a low estimated degree returned by Algorithm `A`.

For a cube I, we first use Algorithm `A` to estimate the degree d of the superpoly of cube I. In general, the running time for Algorithm `B` is much longer than Algorithm `A`. Therefore, we introduce a parameter δ, and we only run Algorithm `B` when the estimated degree d returned by Algorithm `A` satisfies

$0 < d < \delta$, so the running time for Algorithm $\mathtt{B}$ will not be too long. If Algorithm $\mathtt{B}$ returns a constant superpoly, then we obtain a new distinguisher, and we add I to set $\mathbb{D}$; otherwise, we obtain a new key recovery attack if $|I| \leq m-2$, and we add I to set $\mathbb{K}$.

At the beginning of Algorithm 4, we make use of the set Res containing the constant-sum cube testers, which is returned by Algorithm 3. For cube $c \in Res$, $|c|$ represents the minimal dimension of cube where the cube will lead to r_{max}-round cube tester. Therefore, for r-round cipher f, if $r \leq r_{max}$, we first test cubes of dimension $|c|$. As we are going to find cubes for improved distinguishers and key recovery, we will test cubes of smaller dimension after that. However, if $r > r_{max}$, we do not know which dimension would be enough for r-round cube tester, therefore we start with dimension $m-1$.

In Algorithm 4, we still use the embedded property to restrict our search scope. The sufficient set $\mathbb{S}$ returned by Algorithm 2 is a set of good missing indices of cube indices. Let $\mathbb{S}' \subseteq \mathbb{S}$, for different cubes with the same dimension, the cube indices $I = \{0,1,\ldots,m-1\} \setminus \mathbb{S}'$ is more likely to reach more rounds for distinguishing attacks. Therefore testing these cubes would be our priority.

For cubes with dimension $m-t$, we first test cube indices $I = \{0,1,\ldots,m-1\} \setminus \{i_0,i_1,\ldots,i_{t-1}\}$ which satisfy $\{i_0,i_1,\ldots,i_{t-1}\} \subseteq \mathbb{S}$. For a cube I, if the estimated degree d of superpoly is less than δ, then by running Algorithm $\mathtt{B}$ we obtain a cube for new distinguisher or key recovery attack. However, if all cubes are turned to be distinguishers, then we need to test cubes randomly, since other cubes might still have non-constant superpolys. We introduce parameter $trial$ and we will test cubes randomly $trial$ times for cubes with dimension $m-t$. After that, we will test cubes of dimension $m-t-1$. On the other hand, for cube indices $I = \{0,1,\ldots,m-1\} \setminus \{i_0,i_1,\ldots,i_{t-1}\}$ which satisfy $\{i_0,i_1,\ldots,i_{t-1}\} \subseteq \mathbb{S}$, if the estimated degree d of superpolies of all these cubes satisfy that $d \geq \delta$, then we will not test cubes of dimension less than $m-t$, since superpolies of small cubes are more likely to have higher degrees, and we can not run Algorithm $\mathtt{B}$ efficiently.

It should be pointed out that, Algorithm 4 is only a general solution for searching good cubes. Sometimes, we will not run Algorithm 4 strictly. In the application to ACORN, we only test a part of all tested cubes to save time, and we will not test more cubes after we find a key recovery attack.

Algorithm 4 Searching Cubes for Improved Distinguishers and Key Recovery Attack

```
1: procedure SearchCubesForDistinguisherAndKeyRecovery(Given $r$-round cipher $f$ with public variables $(v_0,\ldots,v_{m-1})$ and secret variables $(x_0,\ldots,x_{n-1})$, the number of distinguishable rounds $r_{max}$, the sufficient set $\mathbb{S}$, and the set $Res$ containing the constant-sum cube testers, return a set $\mathbb{D}$ containing the cubes for improved distinguishers, and a set $\mathbb{K}$ containing the cubes for key recovery attack.)
2:     $\mathbb{D} = \mathbb{K} = \emptyset, flag = 1, \delta = 4, trial = 10;$    ▷ the parameter $\delta, trial$ can be set to other suitable values.
3:     let $|c|$ be the dimension of cube $c \in Res$;
4:     let $t = m - |c|$ if $r \le r_{max}$, and $t = 1$ otherwise;
5:     while $flag = 1$ and $t \le |\mathbb{S}|$ do
6:         $flag = 0$;
7:         for every $t$-tuple $(i_0, i_1, \ldots, i_{t-1})$ of $\mathbb{S}$ do
8:             let cube indices $I = \{0, 1, \ldots, m-1\} \setminus \{i_0, i_1, \ldots, i_{t-1}\}$;
9:             use Algorithm A to evaluate the degree $d$ of the superpoly of cube $I$;
10:            if $d < \delta$ then
11:                $flag = 1$;
12:                use Algorithm B to recover the superpoly $p(\boldsymbol{x})$ of cube $I$ if $d > 0$;
13:                if $d = 0$ or $p(\boldsymbol{x}) = 0$ or $p(\boldsymbol{x}) = 1$ then
14:                    $\mathbb{D} = \mathbb{D} \cup \{I\}$;
15:                else if $t \ge 2$ then
16:                    $\mathbb{K} = \mathbb{K} \cup \{I\}$;
17:                end if
18:            end if
19:        end for
20:        if $flag = 1$ and $\mathbb{K} = \emptyset$ and $t \ge 2$ then
21:            for $i = 0$; $i < trial$ do
22:                randomly pick $t$-tuple $(i_0, i_1, \ldots, i_{t-1})$ of $\{0, 1, \ldots, m-1\}$;
23:                let cube indices $I = \{0, 1, \ldots, m-1\} \setminus \{i_0, i_1, \ldots, i_{t-1}\}$;
24:                use Algorithm A to evaluate the degree $d$ of the superpoly of cube $I$;
25:                if $d < \delta$ then
26:                    use Algorithm B to recover the superpoly $p(\boldsymbol{x})$ of cube $I$ if $d > 0$;
27:                    if $d = 0$ or $p(\boldsymbol{x}) = 0$ or $p(\boldsymbol{x}) = 1$ then
28:                        $\mathbb{D} = \mathbb{D} \cup \{I\}$;
29:                    else
30:                        $\mathbb{K} = \mathbb{K} \cup \{I\}$;
31:                        break;
32:                    end if
33:                end if
34:            end for
35:        end if
36:        $t = t + 1$;
37:    end while
38:    return $\mathbb{D}, \mathbb{K}$;
39: end procedure
```

4 Applications to ACORN

In this section, we apply our cube searching methods to ACORN. As a result, we find some new cube testers of dimension 126 on 775-round ACORN. We also find a new key recovery attack on 775-round ACORN with a 126-dimensional cube, whose corresponding superpoly is a 2-degree polynomial with respect to key bits.

4.1 A Brief Description of ACORN

ACORN [23] is an authenticated encryption stream cipher, and it has been selected as one of the 6 algorithms in the final portfolio of the CAESAR competition. ACORN has a 128-bit key and a 128-bit initialization vector. As an authenticated encryption scheme, ACORN has 4 procedures: initialization, processing the associated data, encryption, and finalization. In this paper, we only focus on the process of initialization, because the number of rounds we can attack is smaller than the 1792 initialization rounds. For more details about ACORN, we refer to [23].

Denote the internal state (at step t) of ACORN by $S_t = (s_t, s_{t+1}, \dots, s_{t+292})$, where $t \in \{0, \dots, 1791\}$. The initial state $S_0 = (s_0, s_1, \dots, s_{292})$ is set to $(0, \dots, 0)$. Denote the key and initialization vector by K and IV respectively. Let

$$m_t = \begin{cases} K_t & \text{for } t = 0 \text{ to } 127, \\ IV_{t-128} & \text{for } t = 128 \text{ to } 255, \\ K_0 \oplus 1 & \text{for } t = 256, \\ K_{t \bmod 128} & \text{for } t = 257 \text{ to } 1791. \end{cases}$$

At each step t, where $t \in \{0, \dots, 1791\}$, the state is updated as follows.

1. update using six LFSRs.
   $s_{t+289} = s_{t+289} \oplus s_{t+235} \oplus s_{t+230}$;
   $s_{t+230} = s_{t+230} \oplus s_{t+196} \oplus s_{t+193}$;
   $s_{t+193} = s_{t+193} \oplus s_{t+160} \oplus s_{t+154}$;
   $s_{t+154} = s_{t+154} \oplus s_{t+111} \oplus s_{t+107}$;
   $s_{t+107} = s_{t+107} \oplus s_{t+66} \oplus s_{t+61}$;
   $s_{t+61} = s_{t+61} \oplus s_{t+23} \oplus s_t$;
2. generate the keystream bit.
   $ks_t = s_{t+12} \oplus s_{t+154} \oplus s_{t+235}s_{t+61} \oplus s_{t+235}s_{t+193} \oplus s_{t+61}s_{t+193} \oplus s_{t+230}s_{t+111} \oplus s_{t+230}s_{t+66} \oplus s_{t+66}$;
3. generate the nonlinear feedback bit.
   $f_t = s_t \oplus s_{t+107} \oplus 1 \oplus s_{t+244}s_{t+23} \oplus s_{t+244}s_{t+160} \oplus s_{t+23}s_{t+160} \oplus s_{t+196} \oplus ks_t$;
4. update with the feedback bit f_t.
   $s_{t+293} = f_t \oplus m_t$;

4.2 ACORN's MILP Models for Conventional Division Property and Modified Three-Subset Division Property

It should be noticed that, there are some typing errors and inaccurate descriptions of ACORN cipher in the two MILP models given in [19] and [7]. To remove the ambiguity, we give the refined version of the two MILP models in Appendix B and Appendix C.

Our experiments are based on the two MILP models of ACORN, Algorithm `A` [19], and Algorithm `B` [7]. We use Gurobi Optimizer [5] with Python/C++ interface to solve the MILP problems.

4.3 Cube Testers of 775-round ACORN

In phase 1, we search cubes using conventional division property based cube attack. We first use Algorithm 2 to find the maximum distinguishable rounds r_{max} and the sufficient set $\mathbb{S}$. Our experiments show that $r_{max} = 775$ and $\mathbb{S} = \{1, 2, 11, 18, 26, 27\}$. Then we use Algorithm 3 to find constant-sum cube testers with the smallest dimension (in the conventional division property based cube attack scenario). As a result, only 6 cubes of dimension 127 can lead to the constant-sum cube tester for 775-round ACORN. The cube indices are as follows.

$$I = \{0, 1, \ldots, 127\} \setminus \{i\}, \quad i \in \{1, 2, 11, 18, 26, 27\}$$

In phase 2, we use Algorithm 4 to search more cubes for improved distinguishers. We set all the non-cube IV bits to constant 0 when we use Algorithm `B` to recover the superpoly. As a result, we find 5 cubes with dimension 126, all of which can lead to the constant-sum cube tester for 775-round ACORN. The cube indices are as follows.

$$I = \{0, 1, \ldots, 127\} \setminus \{i, j\}, \quad \{i, j\} \in \{\{1, 2\}, \{1, 11\}, \{1, 18\}, \{2, 18\}, \{11, 18\}\}$$

Table 2. 5 constant superpolies against 775-round ACORN.

missing indices $\{i, j\}$ for cube $I = \{0, 1, \ldots, 127\} \setminus \{i, j\}$	superpoly $p(\boldsymbol{x})$
$\{1, 2\}$	0
$\{1, 11\}$	0
$\{1, 18\}$	0
$\{2, 18\}$	1
$\{11, 18\}$	1

4.4 A Key Recovery Attack of 775-round ACORN

Using Algorithm 4, we find a cube of dimension 126 which can lead to a key recovery attack for 775-round ACORN. The cube indices is as follows,

$$I = \{0, 1, \ldots, 127\} \setminus \{1, 26\}.$$

We set all the non-cube IV bits to 0. By running Algorithm `A`, we know the estimated degree of the superpoly of this cube is 3. Using Algorithm `B`, we know the superpoly is a summation of 40 monomials as follows.

$$\begin{aligned} p(\boldsymbol{x}) = {} & x_{96} + x_{95} + x_{92} + x_{90} + x_{75} + x_{71} + x_{70} + x_{63} + x_{62} + x_{61} + x_{59} + x_{56} + \\ & x_{55} + x_{53} + x_{51} + x_{42} + x_{41} + x_{38} + x_{37} + x_{33} + x_{32} + x_{31} + x_{29}x_{71} + \\ & x_{28} + x_{27} + x_{26} + x_{25} + x_{17}x_{29} + x_{16} + x_{14} + x_{12}x_{29} + x_{11} + x_9 + x_8 + \\ & x_7 + x_5 + x_4 + x_3 + x_1 + x_0. \end{aligned}$$

Since there is a monomial x_0 that is independent of other monomials, this superpoly is a balanced Boolean function. Therefore, we can recover 1 bit secret information by summing over the cube with a complexity of 2^{126}. After that, the remaining bits can be recovered by exhaustive search with 2^{127} time complexity.

5 Conclusions

In this paper, we propose a method to search good cubes for both distinguishing attacks and key recovery attacks in the division property based cube attack scenario. Our cube searching procedure combines the conventional division property and the modified three-subset division property. In the process of cube searching, we mainly use the embedded property to narrow down the searching space. As a result, we find some new cube testers of dimension 126 on 775-round ACORN. We also find a new key recovery attack on 775-round ACORN.

Acknowledgements. We would like to thank Yonglin Hao for his suggestions and helping us to run our codes on Linux servers with high performance.

A Detailed Result for Cube Attacks against ACORN

Table 3. Detailed result for superpoly for 775-round ACORN.

parity	#trails	monomial
0	475072	
0	16	x_{127}
0	162	x_{126}
0	56	x_{125}
0	1490	x_{123}
0	248	x_{121}
0	300	x_{120}
0	248	x_{117}
0	176	x_{116}
0	248	x_{115}
0	88	x_{114}
0	176	x_{112}
0	88	x_{111}
0	88	x_{110}
0	88	x_{108}
0	252	x_{105}
0	8	x_{102}
0	248	x_{100}
1	3189	x_{96}
1	1949	x_{95}
0	16	x_{94}
0	178	x_{93}
1	1919	x_{92}
0	56	x_{91}
1	3207	x_{90}
0	1652	x_{89}
0	320	x_{88}
0	462	x_{87}
0	1862	x_{86}
0	1746	x_{84}
0	724	x_{83}
0	688	x_{82}
0	396	x_{81}
0	344	x_{80}
0	352	x_{79}
0	544	x_{78}
0	492	x_{77}
0	336	x_{76}
1	2193	x_{75}
0	196	x_{74}
0	508	x_{73}
0	254	x_{72}
1	2025	x_{71}
1	1965	x_{70}
0	1954	x_{69}
0	278	x_{68}
0	642	x_{67}
0	360	x_{66}
0	1850	x_{65}
0	1578	x_{64}
1	2293	x_{63}
1	2429	x_{62}
1	2353	x_{61}
0	270	x_{60}
1	2191	x_{59}
0	4224	x_{58}
0	5732	x_{57}
1	3809	x_{56}
1	3979	x_{55}
0	1760	x_{54}
0	248	$x_{54}x_{96}$
1	4061	x_{53}
0	3690	x_{52}
1	5349	x_{51}
0	2504	x_{50}
0	3450	x_{49}
0	1058	x_{48}
0	2474	x_{47}
0	1744	x_{46}
0	2666	x_{45}
0	1440	x_{44}
0	1620	x_{43}
1	4589	x_{42}
0	248	$x_{42}x_{54}$
1	3269	x_{41}
0	1300	x_{40}
0	1250	x_{39}
1	3003	x_{38}
1	4237	x_{37}
0	248	$x_{37}x_{54}$
0	6676	x_{36}
0	2380	x_{35}
0	1646	x_{34}
1	2949	x_{33}
1	4827	x_{32}
1	4071	x_{31}
0	6308	x_{30}
0	8626	x_{29}
1	1701	$x_{29}x_{71}$
1	3979	x_{28}
1	4741	x_{27}
0	16	$x_{27}x_{69}$
1	5319	x_{26}
0	162	$x_{26}x_{68}$
1	6287	x_{25}
0	56	$x_{25}x_{67}$
0	14318	x_{24}
0	11820	x_{23}
0	1490	$x_{23}x_{65}$
0	7188	x_{22}
0	13522	x_{21}
0	248	$x_{21}x_{105}$
0	248	$x_{21}x_{96}$
0	248	$x_{21}x_{63}$
0	248	$x_{21}x_{51}$
0	248	$x_{21}x_{47}$
0	248	$x_{21}x_{46}$
0	248	$x_{21}x_{42}$
0	248	$x_{21}x_{37}$
0	8868	x_{20}
0	248	$x_{20}x_{96}$
0	300	$x_{20}x_{62}$
0	248	$x_{20}x_{42}$
0	248	$x_{20}x_{37}$
0	8570	x_{19}
0	18516	x_{18}
0	10278	x_{17}
0	248	$x_{17}x_{96}$
0	248	$x_{17}x_{59}$
0	248	$x_{17}x_{42}$
0	248	$x_{17}x_{37}$
1	1701	$x_{17}x_{29}$
1	11923	x_{16}
0	176	$x_{16}x_{58}$
0	11436	x_{15}
0	248	$x_{15}x_{96}$
0	248	$x_{15}x_{57}$
0	248	$x_{15}x_{42}$
0	248	$x_{15}x_{37}$
0	16	$x_{15}x_{27}$
1	7449	x_{14}
0	88	$x_{14}x_{56}$
0	162	$x_{14}x_{26}$
0	5802	x_{13}
0	56	$x_{13}x_{25}$
0	13738	x_{12}
0	176	$x_{12}x_{54}$
1	1701	$x_{12}x_{29}$
1	8419	x_{11}
0	88	$x_{11}x_{53}$
0	1490	$x_{11}x_{23}$
0	8834	x_{10}
0	88	$x_{10}x_{52}$
0	16	$x_{10}x_{27}$
1	6405	x_9
0	162	x_9x_{26}
0	248	x_9x_{21}
1	7385	x_8
0	88	x_8x_{50}
0	56	x_8x_{25}
0	248	x_8x_{21}
0	300	x_8x_{20}
1	6021	x_7
0	6886	x_6
0	1490	x_6x_{23}
1	7245	x_5
0	252	x_5x_{47}
0	248	x_5x_{21}
0	248	$x_5x_{21}x_{47}$
0	248	x_5x_{17}
1	10017	x_4
0	248	x_4x_{21}
0	176	x_4x_{16}
1	11945	x_3
0	300	x_3x_{20}
0	248	x_3x_{15}
0	8978	x_2
0	8	x_2x_{44}
0	88	x_2x_{14}
1	7421	x_1
1	6727	x_0
0	248	x_0x_{96}
0	496	x_0x_{42}
0	248	x_0x_{37}
0	248	x_0x_{17}
0	176	x_0x_{12}

B ACORN's MILP Model for Conventional Division Property

Algorithm 5 MILP model for maj in ACORN [19]

```
1: procedure MAJ($\mathcal{M}, \boldsymbol{X}, i, j, k$)
2:     if $X_i.F \oplus X_j.F = 0_c$ then
3:         $(\mathcal{M}, Y_i, a) \leftarrow$ copyf $(\mathcal{M}, X_i)$
4:         $(\mathcal{M}, Y_j, b) \leftarrow$ copyf $(\mathcal{M}, X_j)$
5:         $(\mathcal{M}, o) \leftarrow$ andf $(\mathcal{M}, a, b)$
6:         $Y_s = X_s$ for all $s \in \{0, \ldots, 292\} - \{i, j\}$
7:     else if $X_i.F \oplus X_k.F = 0_c$ then
8:         $(\mathcal{M}, Y_i, a) \leftarrow$ copyf $(\mathcal{M}, X_i)$
9:         $(\mathcal{M}, Y_k, c) \leftarrow$ copyf $(\mathcal{M}, X_k)$
10:        $(\mathcal{M}, o) \leftarrow$ andf $(\mathcal{M}, a, c)$
11:        $Y_s = X_s$ for all $s \in \{0, \ldots, 292\} - \{i, k\}$
12:    else if $X_j.F \oplus X_k.F = 0_c$ then
13:        $(\mathcal{M}, Y_j, b) \leftarrow$ copyf $(\mathcal{M}, X_j)$
14:        $(\mathcal{M}, Y_k, c) \leftarrow$ copyf $(\mathcal{M}, X_k)$
15:        $(\mathcal{M}, o) \leftarrow$ andf $(\mathcal{M}, b, c)$
16:        $Y_s = X_s$ for all $s \in \{0, \ldots, 292\} - \{j, k\}$
17:    else
18:        $(\mathcal{M}, Y_i, a_1, a_2) \leftarrow$ copyf $(\mathcal{M}, X_i)$
19:        $(\mathcal{M}, Y_j, b_1, b_2) \leftarrow$ copyf $(\mathcal{M}, X_j)$
20:        $(\mathcal{M}, Y_k, c_1, c_2) \leftarrow$ copyf $(\mathcal{M}, X_k)$
21:        $(\mathcal{M}, a) \leftarrow$ andf $(\mathcal{M}, a_1, b_1)$
22:        $(\mathcal{M}, b) \leftarrow$ andf $(\mathcal{M}, a_2, c_1)$
23:        $(\mathcal{M}, c) \leftarrow$ andf $(\mathcal{M}, b_2, c_2)$
24:        $(\mathcal{M}, o) \leftarrow$ xorf $(\mathcal{M}, a, b, c)$
25:        $Y_s = X_s$ for all $s \in \{0, \ldots, 292\} - \{i, j, k\}$
26:    end if
27:    return $(\mathcal{M}, \boldsymbol{Y}, o)$
28: end procedure
```

Algorithm 6 MILP model for `ch` in ACORN [19]

```
1: procedure CH($\mathcal{M}, \boldsymbol{X}, i, j, k$)
2:     if $X_i.F = 0_c$ or $X_j.F \oplus X_k.F = 0_c$ then
3:         $(\mathcal{M}, Y_k, o) \leftarrow$ copyf $(\mathcal{M}, X_k)$
4:         $Y_s = X_s$ for all $s \in \{0, \dots, 292\} - \{k\}$
5:     else if $X_i.F = 1_c$ then
6:         $(\mathcal{M}, Y_j, o) \leftarrow$ copyf $(\mathcal{M}, X_j)$
7:         $Y_s = X_s$ for all $s \in \{0, \dots, 292\} - \{j\}$
8:     else
9:         $(\mathcal{M}, Y_i, a_1, a_2) \leftarrow$ copyf $(\mathcal{M}, X_i)$
10:        $(\mathcal{M}, Y_j, b_1) \leftarrow$ copyf $(\mathcal{M}, X_j)$
11:        $(\mathcal{M}, Y_k, c, c_1) \leftarrow$ copyf $(\mathcal{M}, X_k)$
12:        $(\mathcal{M}, a) \leftarrow$ andf $(\mathcal{M}, a_1, b_1)$
13:        $(\mathcal{M}, b) \leftarrow$ andf $(\mathcal{M}, a_2, c_1)$
14:        $(\mathcal{M}, o) \leftarrow$ xorf $(\mathcal{M}, a, b, c)$
15:        $Y_s = X_s$ for all $s \in \{0, \dots, 292\} - \{i, j, k\}$
16:    end if
17:    return $(\mathcal{M}, \boldsymbol{Y}, o)$
18: end procedure
```

Algorithm 7 MILP model for LFSR in ACORN [19]

```
1: procedure XORFB($\mathcal{M}, \boldsymbol{X}, k, i, j$)
2:     $(\mathcal{M}, Y_i, a) \leftarrow$ copyf $(\mathcal{M}, X_i)$
3:     $(\mathcal{M}, Y_j, b) \leftarrow$ copyf $(\mathcal{M}, X_j)$
4:     $(\mathcal{M}, Y_k) \leftarrow$ xorf $(\mathcal{M}, a, b, X_k)$
5:     $Y_s = X_s$ for all $s \in \{0, \dots, 292\} - \{i, j, k\}$
6:     return $(\mathcal{M}, \boldsymbol{Y})$
7: end procedure
```

Algorithm 8 MILP model for `ksg128` in ACORN [19]

```
1: procedure KSG128($\mathcal{M}, \boldsymbol{X}$)
2:     $(\mathcal{M}, A_{12}, x_1) \leftarrow$ copyf $(\mathcal{M}, X_{12})$
3:     $(\mathcal{M}, A_{154}, x_2) \leftarrow$ copyf $(\mathcal{M}, X_{154})$
4:     $A_t = X_t$ for all $t \in \{0, \dots, 292\} - \{12, 154\}$
5:     $(\mathcal{M}, \boldsymbol{B}, x_3) \leftarrow$ maj $(\mathcal{M}, \boldsymbol{A}, 235, 61, 193)$
6:     $(\mathcal{M}, \boldsymbol{Y}, x_4) \leftarrow$ ch $(\mathcal{M}, \boldsymbol{B}, 230, 111, 66)$
7:     $(\mathcal{M}, z) \leftarrow$ xorf $(\mathcal{M}, x_1, x_2, x_3, x_4)$
8:     return $(\mathcal{M}, \boldsymbol{Y}, z)$
9: end procedure
```

Algorithm 9 MILP model for `fbk128` in ACORN [19]

```
1: procedure FBK128($\mathcal{M}, \boldsymbol{X}, ks$)
2:     $(\mathcal{M}, A_0, x_1) \leftarrow$ copyf $(\mathcal{M}, X_0)$
3:     $(\mathcal{M}, A_{107}, x_2) \leftarrow$ copyf $(\mathcal{M}, X_{107})$
4:     $(\mathcal{M}, A_{196}, x_3) \leftarrow$ copyf $(\mathcal{M}, X_{196})$
5:     $A_t = X_t$ for all $t \in \{0, \ldots, 292\} - \{0, 107, 196\}$
6:     $(\mathcal{M}, \boldsymbol{B}, x_4) \leftarrow$ maj $(\mathcal{M}, \boldsymbol{A}, 244, 23, 160)$
7:     $(\mathcal{M}, z) \leftarrow$ xorf $(\mathcal{M}, x_1, x_2, x_3, x_4, ks)$
8:     $z.F = z.F \oplus 1_c$
9:     return $(\mathcal{M}, \boldsymbol{B}, z)$
10: end procedure
```

Algorithm 10 MILP model for the initialization of ACORN [19]

```
1: procedure ACORNEVAL(round $R$)
2:     Prepare empty MILP model $\mathcal{M}$
3:     $\mathcal{M}.var \leftarrow x_i$ for $i \in \{0, 1, \ldots, 127\}$ as binary        ▷ Declare the secret key bits
4:     $\mathcal{M}.var \leftarrow v_i$ for $i \in \{0, 1, \ldots, 127\}$ as binary        ▷ Declare the public IV bits
5:     $\mathcal{M}.var \leftarrow S_i^0$ 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
6:     for $r = 1$ to $R$ do
7:         $(\mathcal{M}, \boldsymbol{T}) = \texttt{xorFB}\ (\mathcal{M}, \boldsymbol{S}^{r-1}, 289, 235, 230)$
8:         $(\mathcal{M}, \boldsymbol{U}) = \texttt{xorFB}\ (\mathcal{M}, \boldsymbol{T}, 230, 196, 193)$
9:         $(\mathcal{M}, \boldsymbol{V}) = \texttt{xorFB}\ (\mathcal{M}, \boldsymbol{U}, 193, 160, 154)$
10:        $(\mathcal{M}, \boldsymbol{W}) = \texttt{xorFB}\ (\mathcal{M}, \boldsymbol{V}, 154, 111, 107)$
11:        $(\mathcal{M}, \boldsymbol{X}) = \texttt{xorFB}\ (\mathcal{M}, \boldsymbol{W}, 107, 66, 61)$
12:        $(\mathcal{M}, \boldsymbol{Y}) = \texttt{xorFB}\ (\mathcal{M}, \boldsymbol{X}, 61, 23, 0)$
13:        $(\mathcal{M}, \boldsymbol{Z}, ks) = \texttt{ksg128}\ (\mathcal{M}, \boldsymbol{Y})$
14:        $(\mathcal{M}, \boldsymbol{A}, f) = \texttt{fbk128}\ (\mathcal{M}, \boldsymbol{Z}, ks)$
15:        $\mathcal{M}.con \leftarrow A_0 = 0$
16:        for $i = 0$ to 291 do
17:            $S_i^r = A_{i+1}$
18:        end for
19:        $\mathcal{M}.var \leftarrow S_{292}^r$ as binary
20:        if $128 < r \leq 256$ then
21:            $\mathcal{M}.con \leftarrow S_{292}^r = f + v_{r-129}$ and assign the flags $S_{292}^r.F = f.F \oplus v_{r-129}.F$
22:        else
23:            $\mathcal{M}.var \leftarrow TK_{r-1}$ as binary and assign its flag as
               $$TK_{r-1}.F = \begin{cases} x_{(r-1) \bmod 128}.F \oplus 1_c & \text{if } r = 257, \\ x_{(r-1) \bmod 128}.F & \text{otherwise.} \end{cases}$$
24:            $\mathcal{M}.con \leftarrow S_{292}^r = f + TK_{r-1}$ and assign $S_{292}^r.F = f.F \oplus TK_{r-1}.F$
25:        end if
26:    end for
27:    for $i = 0$ to 127 do
28:        $\mathcal{M}.con \leftarrow x_i = \sum_j TK_{i+128 \times j}$
29:    end for
30:    $(\mathcal{M}, \boldsymbol{T}) = \texttt{xorFB}\ (\mathcal{M}, \boldsymbol{S}^R, 289, 235, 230)$
31:    $(\mathcal{M}, \boldsymbol{U}) = \texttt{xorFB}\ (\mathcal{M}, \boldsymbol{T}, 230, 196, 193)$
32:    $(\mathcal{M}, \boldsymbol{V}) = \texttt{xorFB}\ (\mathcal{M}, \boldsymbol{U}, 193, 160, 154)$
33:    $(\mathcal{M}, \boldsymbol{W}) = \texttt{xorFB}\ (\mathcal{M}, \boldsymbol{V}, 154, 111, 107)$
34:    $(\mathcal{M}, \boldsymbol{X}) = \texttt{xorFB}\ (\mathcal{M}, \boldsymbol{W}, 107, 66, 61)$
35:    $(\mathcal{M}, \boldsymbol{Y}) = \texttt{xorFB}\ (\mathcal{M}, \boldsymbol{X}, 61, 23, 0)$
36:    $(\mathcal{M}, \boldsymbol{Z}, ks) = \texttt{ksg128}\ (\mathcal{M}, \boldsymbol{Y})$
37:    for $i = 0$ to 292 do
38:        $\mathcal{M}.con \leftarrow Z_i = 0$
39:    end for
40:    $\mathcal{M}.con \leftarrow ks = 1$
41:    return $\mathcal{M}$
42: end procedure
```

C ACORN's MILP Model for Modified Three-Subset Division Property

Algorithm 11 MILP model for `majt` in ACORN

```
1: procedure MAJT($\mathcal{M}, \boldsymbol{X}, i, j, k$)
2:     $(\mathcal{M}, Y_i, a_1, a_2) \leftarrow$ copyt $(\mathcal{M}, X_i)$
3:     $(\mathcal{M}, Y_j, b_1, b_2) \leftarrow$ copyt $(\mathcal{M}, X_j)$
4:     $(\mathcal{M}, Y_k, c_1, c_2) \leftarrow$ copyt $(\mathcal{M}, X_k)$
5:     $(\mathcal{M}, a) \leftarrow$ andt $(\mathcal{M}, a_1, b_1)$
6:     $(\mathcal{M}, b) \leftarrow$ andt $(\mathcal{M}, a_2, c_1)$
7:     $(\mathcal{M}, c) \leftarrow$ andt $(\mathcal{M}, b_2, c_2)$
8:     $(\mathcal{M}, o) \leftarrow$ xort $(\mathcal{M}, a, b, c)$
9:     $Y_s = X_s$ for all $s \in \{0, \dots, 292\} - \{i, j, k\}$
10:    return $(\mathcal{M}, \boldsymbol{Y}, o)$
11: end procedure
```

Algorithm 12 MILP model for `cht` in ACORN

```
1: procedure CHT($\mathcal{M}, \boldsymbol{X}, i, j, k$)
2:     $(\mathcal{M}, Y_i, a_1, a_2) \leftarrow$ copyt $(\mathcal{M}, X_i)$
3:     $(\mathcal{M}, Y_j, b_1) \leftarrow$ copyt $(\mathcal{M}, X_j)$
4:     $(\mathcal{M}, Y_k, c, c_1) \leftarrow$ copyt $(\mathcal{M}, X_k)$
5:     $(\mathcal{M}, a) \leftarrow$ andt $(\mathcal{M}, a_1, b_1)$
6:     $(\mathcal{M}, b) \leftarrow$ andt $(\mathcal{M}, a_2, c_1)$
7:     $(\mathcal{M}, o) \leftarrow$ xort $(\mathcal{M}, a, b, c)$
8:     $Y_s = X_s$ for all $s \in \{0, \dots, 292\} - \{i, j, k\}$
9:     return $(\mathcal{M}, \boldsymbol{Y}, o)$
10: end procedure
```

Algorithm 13 MILP model for `xorFBt` in ACORN

```
1: procedure XORFBT($\mathcal{M}, \boldsymbol{X}, k, i, j$)
2:     $(\mathcal{M}, Y_i, a) \leftarrow$ copyt $(\mathcal{M}, X_i)$
3:     $(\mathcal{M}, Y_j, b) \leftarrow$ copyt $(\mathcal{M}, X_j)$
4:     $(\mathcal{M}, Y_k) \leftarrow$ xort $(\mathcal{M}, a, b, X_k)$
5:     $Y_s = X_s$ for all $s \in \{0, \dots, 292\} - \{i, j, k\}$
6:     return $(\mathcal{M}, \boldsymbol{Y})$
7: end procedure
```

Algorithm 14 MILP model for `ksg128t` in ACORN

```
1: procedure KSG128T($(\mathcal{M}, \boldsymbol{X})$)
2:     $(\mathcal{M}, A_{12}, x_1) \leftarrow$ copyt $(\mathcal{M}, X_{12})$
3:     $(\mathcal{M}, A_{154}, x_2) \leftarrow$ copyt $(\mathcal{M}, X_{154})$
4:     $A_t = X_t$ for all $t \in \{0, \ldots, 292\} - \{12, 154\}$
5:     $(\mathcal{M}, \boldsymbol{B}, x_3) \leftarrow$ majt $(\mathcal{M}, \boldsymbol{A}, 235, 61, 193)$
6:     $(\mathcal{M}, \boldsymbol{Y}, x_4) \leftarrow$ cht $(\mathcal{M}, \boldsymbol{B}, 230, 111, 66)$
7:     $(\mathcal{M}, z) \leftarrow$ xort $(\mathcal{M}, x_1, x_2, x_3, x_4)$
8:     return $(\mathcal{M}, \boldsymbol{Y}, z)$
9: end procedure
```

Algorithm 15 MILP model for `fbk128t` in ACORN

```
1: procedure FBK128T($(\mathcal{M}, \boldsymbol{X}, ks, r)$)
2:     $(\mathcal{M}, A_0, x_1) \leftarrow$ copyt $(\mathcal{M}, X_0)$
3:     $(\mathcal{M}, A_{107}, x_2) \leftarrow$ copyt $(\mathcal{M}, X_{107})$
4:     $(\mathcal{M}, A_{196}, x_3) \leftarrow$ copyt $(\mathcal{M}, X_{196})$
5:     $A_t = X_t$ for all $t \in \{0, \ldots, 292\} - \{0, 107, 196\}$
6:     $(\mathcal{M}, \boldsymbol{B}, x_4) \leftarrow$ majt $(\mathcal{M}, \boldsymbol{A}, 244, 23, 160)$
7:     if $r = 257$ then
8:         $(\mathcal{M}, z) \leftarrow$ xort $(\mathcal{M}, x_1, x_2, x_3, x_4, ks)$
9:     else
10:        $\mathcal{M}.var \leftarrow o$ as binary
11:        $(\mathcal{M}, z) \leftarrow$ xort $(\mathcal{M}, x_1, x_2, x_3, x_4, ks, o)$          ▷ o corresponds to constant 1
12:    end if
13:    return $(\mathcal{M}, \boldsymbol{B}, z)$
14: end procedure
```

Algorithm 16 ACORN's MILP model for modified three-subset division property

```
1: procedure ACORNEvalThree(round $R$)
2:     Prepare empty MILP model $\mathcal{M}$
3:     $\mathcal{M}.var \leftarrow x_i$ for $i \in \{0, 1, \dots, 127\}$ as binary
4:     $\mathcal{M}.var \leftarrow v_i$ for $i \in \{0, 1, \dots, 127\}$ as binary
5:     $\mathcal{M}.var \leftarrow S_i^0$ for $i \in \{0, 1, \dots, 292\}$ as binary
6:     $\mathcal{M}.con \leftarrow S_i^0 = 0$ for $i \in \{0, 1, \dots, 292\}$
7:     for $r = 1$ to $R$ do
8:         $(\mathcal{M}, \boldsymbol{T})$ = xorFBt $(\mathcal{M}, \boldsymbol{S}^{r-1}, 289, 235, 230)$
9:         $(\mathcal{M}, \boldsymbol{U})$ = xorFBt $(\mathcal{M}, \boldsymbol{T}, 230, 196, 193)$
10:        $(\mathcal{M}, \boldsymbol{V})$ = xorFBt $(\mathcal{M}, \boldsymbol{U}, 193, 160, 154)$
11:        $(\mathcal{M}, \boldsymbol{W})$ = xorFBt $(\mathcal{M}, \boldsymbol{V}, 154, 111, 107)$
12:        $(\mathcal{M}, \boldsymbol{X})$ = xorFBt $(\mathcal{M}, \boldsymbol{W}, 107, 66, 61)$
13:        $(\mathcal{M}, \boldsymbol{Y})$ = xorFBt $(\mathcal{M}, \boldsymbol{X}, 61, 23, 0)$
14:        $(\mathcal{M}, \boldsymbol{Z}, ks)$ = ksg128t $(\mathcal{M}, \boldsymbol{Y})$
15:        $(\mathcal{M}, \boldsymbol{A}, f)$ = fbk128t $(\mathcal{M}, \boldsymbol{Z}, ks, r)$
16:        $\mathcal{M}.con \leftarrow A_0 = 0$
17:        for $i = 0$ to 291 do
18:            $S_i^r = A_{i+1}$
19:        end for
20:        $\mathcal{M}.var \leftarrow S_{292}^r$ as binary
21:        if $128 < r \leq 256$ then
22:            $\mathcal{M}.con \leftarrow S_{292}^r = f + v_{r-129}$
23:        else
24:            $\mathcal{M}.var \leftarrow TK_{r-1}$
25:            $\mathcal{M}.con \leftarrow S_{292}^r = f + TK_{r-1}$
26:        end if
27:    end for
28:    for $i = 0$ to 127 do
29:        for all possible $j$ do
30:            $\mathcal{M}.con \leftarrow TK_{i+128\times j} \leq x_i$
31:        end for
32:        $\mathcal{M}.con \leftarrow \sum_j TK_{i+128\times j} \geq x_i$
33:    end for
34:    $(\mathcal{M}, \boldsymbol{T})$ = xorFBt $(\mathcal{M}, \boldsymbol{S}^R, 289, 235, 230)$
35:    $(\mathcal{M}, \boldsymbol{U})$ = xorFBt $(\mathcal{M}, \boldsymbol{T}, 230, 196, 193)$
36:    $(\mathcal{M}, \boldsymbol{V})$ = xorFBt $(\mathcal{M}, \boldsymbol{U}, 193, 160, 154)$
37:    $(\mathcal{M}, \boldsymbol{W})$ = xorFBt $(\mathcal{M}, \boldsymbol{V}, 154, 111, 107)$
38:    $(\mathcal{M}, \boldsymbol{X})$ = xorFBt $(\mathcal{M}, \boldsymbol{W}, 107, 66, 61)$
39:    $(\mathcal{M}, \boldsymbol{Y})$ = xorFBt $(\mathcal{M}, \boldsymbol{X}, 61, 23, 0)$
40:    $(\mathcal{M}, \boldsymbol{Z}, ks)$ = ksg128t $(\mathcal{M}, \boldsymbol{Y})$
41:    for $i = 0$ to 292 do
42:        $\mathcal{M}.con \leftarrow Z_i = 0$
43:    end for
44:    $\mathcal{M}.con \leftarrow ks = 1$
45:    return $\mathcal{M}$
46: end procedure
```

References

1. Aumasson, J., Dinur, I., Meier, W., Shamir, A.: Cube testers and key recovery attacks on reduced-round MD6 and trivium. In: Dunkelman, O. (ed.) Fast Software Encryption, 16th International Workshop, FSE 2009, Leuven, Belgium, February 22-25, 2009, Revised Selected Papers. Lecture Notes in Computer Science, vol. 5665, pp. 1–22. Springer (2009)
2. Dinur, I., Shamir, A.: Cube attacks on tweakable black box polynomials. In: Joux, A. (ed.) Advances in Cryptology - EUROCRYPT 2009, 28th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cologne, Germany, April 26-30, 2009. Proceedings. Lecture Notes in Computer Science, vol. 5479, pp. 278–299. Springer (2009)
3. Fouque, P., Vannet, T.: Improving key recovery to 784 and 799 rounds of trivium using optimized cube attacks. In: Moriai, S. (ed.) Fast Software Encryption - 20th International Workshop, FSE 2013, Singapore, March 11-13, 2013. Revised Selected Papers. Lecture Notes in Computer Science, vol. 8424, pp. 502–517. Springer (2013)
4. Ghafari, V.A., Hu, H.: A new chosen IV statistical distinguishing framework to attack symmetric ciphers, and its application to acorn-v3 and grain-128a. J. Ambient Intell. Humaniz. Comput. **10**(6), 2393–2400 (2019)
5. Gurobi: Gurobi Optimizer. `http://www.gurobi.com/`
6. Hao, Y., Jiao, L., Li, C., Meier, W., Todo, Y., Wang, Q.: Links between division property and other cube attack variants. IACR Trans. Symmetric Cryptol. **2020**(1), 363–395 (2020)
7. Hao, Y., Leander, G., Meier, W., Todo, Y., Wang, Q.: Modeling for three-subset division property without unknown subset. IACR Cryptol. ePrint Arch. **2020**, 441 (2020), `https://eprint.iacr.org/2020/441`
8. Hao, Y., Leander, G., Meier, W., Todo, Y., Wang, Q.: Modeling for three-subset division property without unknown subset - improved cube attacks against trivium and grain-128aead. In: Canteaut, A., Ishai, Y. (eds.) Advances in Cryptology - EUROCRYPT 2020 - 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10-14, 2020, Proceedings, Part I. Lecture Notes in Computer Science, vol. 12105, pp. 466–495. Springer (2020)
9. Hu, K., Wang, M.: Automatic search for a variant of division property using three subsets. In: Matsui, M. (ed.) Topics in Cryptology - CT-RSA 2019 - The Cryptographers' Track at the RSA Conference 2019, San Francisco, CA, USA, March 4-8, 2019, Proceedings. Lecture Notes in Computer Science, vol. 11405, pp. 412–432. Springer (2019)
10. Kesarwani, A., Roy, D., Sarkar, S., Meier, W.: New cube distinguishers on nfsr-based stream ciphers. Des. Codes Cryptogr. **88**(1), 173–199 (2020)
11. Knudsen, L.R., Wagner, D.A.: Integral cryptanalysis. In: Daemen, J., Rijmen, V. (eds.) Fast Software Encryption, 9th International Workshop, FSE 2002, Leuven, Belgium, February 4-6, 2002, Revised Papers. Lecture Notes in Computer Science, vol. 2365, pp. 112–127. Springer (2002)
12. Lai, X.: Higher order derivatives and differential cryptanalysis. In: Communications and Cryptography, pp. 227–233. Springer (1994)
13. 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: Applications and Techniques in Information Security - 6th International Conference, ATIS 2016, Cairns, QLD, Australia, October 26-28, 2016, Proceedings. pp. 15–26 (2016)

14. 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.) Advances in Cryptology - ASIACRYPT 2017 - 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part I. Lecture Notes in Computer Science, vol. 10624, pp. 128–157. Springer (2017)
15. Todo, Y.: Structural evaluation by generalized integral property. In: Oswald, E., Fischlin, M. (eds.) Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I. Lecture Notes in Computer Science, vol. 9056, pp. 287–314. Springer (2015)
16. Todo, Y., Isobe, T., Hao, Y., Meier, W.: Cube attacks on non-blackbox polynomials based on division property. In: Katz, J., Shacham, H. (eds.) Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part III. Lecture Notes in Computer Science, vol. 10403, pp. 250–279. Springer (2017)
17. Todo, Y., Isobe, T., Hao, Y., Meier, W.: Cube attacks on non-blackbox polynomials based on division property. IACR Cryptol. ePrint Arch. **2017**, 306 (2017), `http://eprint.iacr.org/2017/306`
18. Todo, Y., Morii, M.: Bit-based division property and application to simon family. In: Peyrin, T. (ed.) Fast Software Encryption - 23rd International Conference, FSE 2016, Bochum, Germany, March 20-23, 2016, Revised Selected Papers. Lecture Notes in Computer Science, vol. 9783, pp. 357–377. Springer (2016)
19. Wang, Q., Hao, Y., Todo, Y., Li, C., Isobe, T., Meier, W.: Improved division property based cube attacks exploiting algebraic properties of superpoly. IACR Cryptol. ePrint Arch. **2017**, 1063 (2017), `http://eprint.iacr.org/2017/1063`
20. Wang, Q., Hao, Y., Todo, Y., Li, C., Isobe, T., Meier, W.: Improved division property based cube attacks exploiting algebraic properties of superpoly. In: Shacham, H., Boldyreva, A. (eds.) Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part I. Lecture Notes in Computer Science, vol. 10991, pp. 275–305. Springer (2018)
21. Wang, Q., Liu, Z., Varici, K., Sasaki, Y., Rijmen, V., Todo, Y.: Cryptanalysis of reduced-round SIMON32 and SIMON48. In: Meier, W., Mukhopadhyay, D. (eds.) Progress in Cryptology - INDOCRYPT 2014 - 15th International Conference on Cryptology in India, New Delhi, India, December 14-17, 2014, Proceedings. Lecture Notes in Computer Science, vol. 8885, pp. 143–160. Springer (2014)
22. Wang, S., Hu, B., Guan, J., Zhang, K., Shi, T.: Milp-aided method of searching division property using three subsets and applications. In: Galbraith, S.D., Moriai, S. (eds.) Advances in Cryptology - ASIACRYPT 2019 - 25th International Conference on the Theory and Application of Cryptology and Information Security, Kobe, Japan, December 8-12, 2019, Proceedings, Part III. Lecture Notes in Computer Science, vol. 11923, pp. 398–427. Springer (2019)
23. Wu, H.: ACORN: a lightweight authenticated cipher (v3). Candidate for the CAESAR Competition (2016), `https://competitions.cr.yp.to/round3/acornv3.pdf`
24. 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.) Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and

Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part I. Lecture Notes in Computer Science, vol. 10031, pp. 648–678 (2016)

25. Yang, J., Liu, M., Lin, D.: Cube cryptanalysis of round-reduced ACORN. In: Lin, Z., Papamanthou, C., Polychronakis, M. (eds.) Information Security - 22nd International Conference, ISC 2019, New York City, NY, USA, September 16-18, 2019, Proceedings. Lecture Notes in Computer Science, vol. 11723, pp. 44–64. Springer (2019)
26. Ye, C., Tian, T.: Revisit division property based cube attacks: Key-recovery or distinguishing attacks? IACR Trans. Symmetric Cryptol. **2019**(3), 81–102 (2019)