Cube Attacks on Non-Blackbox Polynomials Based on Division Property

Yosuke Todo[1], Takanori Isobe[2], Yonglin Hao[3], and Willi Meier[4]

[1] NTT Secure Platform Laboratories, Tokyo 180-8585, Japan
todo.yosuke@lab.ntt.co.jp
[2] University of Hyogo, Hyogo 650-0047, Japan
[3] Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China
[4] FHNW, Windisch, Switzerland

Abstract. The cube attack is a powerful cryptanalytic technique and is especially powerful against stream ciphers. Since we need to analyze the complicated structure of a stream cipher in the cube attack, the cube attack basically analyzes it by regarding it as a blackbox. Therefore, the cube attack is an experimental attack, and we cannot evaluate the security when the size of cube exceeds an experimental range, e.g., 40. In this paper, we propose cube attacks on non-blackbox polynomials. Our attacks are developed by using the division property, which is recently applied to various block ciphers. The clear advantage is that we can exploit large cube sizes because it never regards the cipher as a blackbox. We apply the new cube attack to Trivium, Grain128a, and ACORN. As a result, the secret keys of 832-round Trivium, 183-round Grain128a, and 704-round ACORN are recovered. These attacks are the current best key-recovery attack against these ciphers.

Keywords: Cube attack, Stream cipher, Division property, Higher-order differential cryptanalysis, MILP, Trivium, Grain128a, ACORN

1 Introduction

Cube attack is one of general cryptanalytic techniques against symmetric-key cryptosystems proposed by Dinur and Shamir [DS09]. Especially, the cube attack has been successfully applied to various stream ciphers [ADMS09,DS11,FV13,DMP$^+$15,SBD$^+$16]. Let $\boldsymbol{x}$ and $\boldsymbol{v}$ be secret and public variables of stream ciphers, respectively, and let $f(\boldsymbol{x},\boldsymbol{v})$ be the first bit of key stream. Some bits in $\boldsymbol{v}$ are active, where they take all possible combinations of values. The set of these values is denoted as a *cube*, and the sum of $f(\boldsymbol{x},\boldsymbol{v})$ over all values of the cube is evaluated. Then, this sum is also represented as a polynomial whose inputs are $\boldsymbol{x}$ and $\boldsymbol{v}$, and the polynomial is denoted as a *superpoly* of the cube. The superpoly is more simplified than the original $f(\boldsymbol{x},\boldsymbol{v})$, and secret variables $\boldsymbol{x}$ are recovered by analyzing this simplified polynomial. Unfortunately, it is really difficult to analyze the structure of the superpoly. Therefore, the target stream cipher $f(\boldsymbol{x},\boldsymbol{v})$ is normally regarded as a blackbox polynomial in the cube attack, and this blackbox polynomial is experimentally evaluated. In the original paper of the cube attack [DS09], the authors introduced a linearity test to reveal the structure of the superpoly. If the linearity test always passes, the Algebraic Normal Form (ANF) of the superpoly is recovered by assuming that the superpoly is linear. Moreover, a quadraticity test was introduced in [MS12], and the ANF of the superpoly is similarly recovered. The quadraticity test was also used in the current best key-recovery attack against Trivium [FV13]. Note that they are experimental cryptanalysis, and it is possible that cube attacks do not actually work. For example, if the superpoly is highly unbalanced function for specific variables, we cannot ignore the probability that the linearity and quadraticity tests fail.

The difference between the cube attack and higher-order differential attack has been often discussed. The higher-order differential attack was proposed by Lai [Lai94]. Assuming the algebraic degree of f is at most d, Lai showed that the algebraic degree of the ith order difference is at most $d - i$. Then, Knudsen showed the effectiveness of the higher-order differential attack on toy block ciphers [Knu94]. Nowadays, many advanced techniques similar to the higher-order differential attack have been developed to analyze block ciphers, e.g., integral attack [DKR97,Luc01,KW02].

The cube attack can in some way be seen as a type of higher-order differential attacks because it also evaluates the behavior of higher-order difference. However, the most major difference between the cube attack and common higher-order differential attack is whether or not secret variables are directly recovered from the characteristic, and understanding this difference is very important to consider key-recovery attacks against stream ciphers. When a block cipher is analyzed, attackers first evaluate the algebraic degree of the reduced-round block cipher and construct a higher-order differential characteristic, where the $(d + 1)$th order difference is always 0 if the degree is at most d. Then, the key recovery is independently appended after the higher-order differential characteristic. Namely, attackers guess round keys used in last several rounds and compute the $(d + 1)$th order difference of ciphertexts of the reduced-round block cipher. If the correct round key is guessed, the $(d+1)$th order difference is always 0. In other words, if the $(d+1)$th order difference is not 0, guessed round keys are incorrect.

Note that we cannot use this strategy for the key-recovery attack against many stream ciphers because the secret key is generally used during the initialization phase and is not involved when generating a keystream, i.e. even if there is a distinguisher in the keystream, it cannot be directly utilized for key recovery attacks by appending key recovery rounds in the key generation phase, unlike key recovery attacks of block ciphers. To execute the key-recovery attack of stream ciphers, we have to recover the secret key by using only key streams that attackers can observe. Therefore, more advanced and complicated analyses are required than the simple degree estimation of the common higher-order differential attack or square, saturation, and integral characteristics. In the context of the cube attack, we have to analyze the ANF of the superpoly. It is unlikely to well analyze because symmetric-key cryptosystems are complicated. Therefore, stream ciphers have been experimentally analyzed in the cube attack.

Another important related work to understand this paper is the division property, which is a new method to construct higher-order differential (integral) characteristics [Tod15b]. The division property is the generalization of the integral property [KW02] that can also exploit the algebraic degree at the same time, and it allows us to evaluate more accurate higher-order differential characteristics. Moreover, the bit-based division property was introduced in [TM16], and three propagation rules for basic operations, `and`, `xor`, and `copy` are shown. While arbitrary block ciphers are evaluated by using the bit-based division property, it requires much time and memory complexity [TM16]. Therefore, the application is first limited to block ciphers with small block length, like Simon32 or Simeck32. In [XZBL16], Xiang et al. showed how to model the propagation of the bit-based division property by using the mixed integer linear programming (MILP). Moreover, they showed that MILP solvers can efficiently evaluate the propagation. To demonstrate the effectiveness, accurate propagations of the bit-based division property for six lightweight block ciphers including Simon128 were shown.

Our Contribution. The most important step in a cube attack is the *superpoly recovery*. If the superpoly is more efficiently recovered than the brute-force search, it brings some vulnerability of symmetric-key ciphers. Superpolys are experimentally recovered in the conventional cube attack. The advantage of such approach is that we do not need to analyze the structure of f in detail. On the other hand, there are significant drawbacks in the experimental analysis.

- The size of a cube is limited to the experimental range because we have to compute the sum of f over a cube. It may be possible that we try a cube whose size is at most 40 in current computers, but it requires incredible effort in the aspect to both money and time. Therefore, it is practically infeasible to execute the cube attack when the cube size exceeds 40.

Table 1. Summary of results. The time complexity in this table shows the time complexity to recover the superpoly of a cube.

Applications	# rounds	cube size	complexity	key recovery	reference
Trivium	799	32 †	practical	✓	[FV13]
	832	72	2^{77}	✓	Sect. 5.1
Grain128a	177	33	practical		[LM12]
	183	92	2^{108}	speculative	Sect. 5.2
ACORN	503	5 ‡	practical ‡	✓	[SBD$^+$16]
	704	64	2^{122}	✓	Sect. 5.3

† 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.
‡ The attack against 477 rounds is mainly described for the practical attack in [SBD$^+$16]. However, when the goal is the superpoly recovery and to recover one bit of the secret key, 503 rounds are attacked.

- The prediction of the true security of target stream ciphers is an important motivation of cryptanalyses. Since the evaluation is limited to the experimental range, it is difficult to predict the impact of the cube attack under future high-performance computers.
- Since the stream cipher is regarded as a blackbox, the feedback to designers is limited.

To overcome these drawbacks, we propose the cube attack on non-blackbox polynomials.

Our analysis is based on the propagation of the (bit-based) division property, and as far as we know, it is the first application of the division property to stream ciphers. Since the division property is a tool to find higher-order differential characteristics, the trivial application is only useful to find zero-sum integral distinguishers, where the sum of the first bit of the key stream over the cube is always 0 for any secret key. As mentioned earlier, it is nontrivial to recover the secret key of stream ciphers by using zero-sum integral distinguisher. Therefore, we propose a novel application of the division property to recover the secret key. Our technique uses the division property to analyze the ANF of $f(\boldsymbol{x}, \boldsymbol{v})$ by evaluating propagations from multiple input division property according to a cube. Finally, we can evaluate secret variables that are not involved to the superpoly of the cube. This allows us to compute the upper bound of the time complexity for the superpoly recovery. Note that the superpoly recovery directly brings some vulnerability of symmetric-key ciphers, and we discuss this issue in Sect. 4.

Let I be a set of cube indices. After the evaluation of the division property, we get a set of indices J, where x_j $(j \in J)$ is involved to the superpoly. Then, the variation of the sum over the cube is at most $2^{|J|}$ for each constant part of public variables, where $|J|$ denotes the size of J. All sums are evaluated by guessing $|J|$-bit secret variables, and the time complexity to recover the ANF of the superpoly is $2^{|I|+|J|}$ encryptions. Finally, we query the encryption oracle and get the sum over the cube. Then, we can get one polynomial about secret variables, and the secret variable is recovered from the polynomial.

Table 1 shows the summary of applications. We applied our new cube attack to Trivium [CP06], Grain128a [ÅHJM11], and ACORN [Wu16]. Trivium is part of the eSTREAM portfolio [est08], and it is one of the most analyzed stream ciphers. The initialization is 1152 rounds. The secret key of Trivium with 767 initialization rounds was recovered in the proposal paper of the cube attack [DS09]. Then, an improved cube attack was proposed in [FV13], and the secret key of Trivium with 799 initialization rounds is recovered. This is the current best key-recovery attack against Trivium. Our new cube attack recovers the secret key of Trivium with 832 initialization rounds. Grain128a is a member of Grain family of stream ciphers and is standardized by ISO/IEC 29167-13 [ISO15].

The initialization is 256 rounds. The conditional differential cryptanalysis was applied to Grain128a, and a distinguishing attack against Grain128a with 177 initialization rounds was shown under the single-key setting [LM12]. On the other hand, the key-recovery attack is not known. Our new cube attack recovers the secret key of Grain128a with 183 initialization rounds. Unfortunately, when we applied our technique to practical cube attack, i.e., the cube size is small, we could not find balanced superpoly. In such case, the size of recovered bit of information is smaller than 1 bit. Since we cannot say that balanced superpoly is efficiently found in the large cube size, the feasibility of the key recovery is speculative. However, 183 rounds are at least vulnerable because the superpoly recovery is more efficient than the brute-force search. ACORN is an authenticated encryption and one of the 3rd round candidates in CAESAR competition [cae14]. The structure is based on non-linear feedback shift register (NLFSR) like Trivium and Grain. Before the output of key streams, the secret key and initialization vector (iv) are sequentially XORed with the NLFSR, and then associated data is sequentially XORed. In the nonce-respecting setting, we cannot select cube bits from the associated data. Therefore, the initialization is regarded as 2048 rounds when there is no associated data. The cube attack was applied in [SBD+16], and the secret key of ACORN with 503 initialization is recovered. Our new cube attack recovers the secret key of ACORN with 704 initialization rounds.

2 Preliminaries

2.1 Mixed Integer Linear Programming

The deployment of the mixed integer linear programming (MILP) to cryptanalysis was shown by Mouha et al. in [MWGP11]. Then, the MILP has been applied to search for differential [SHW+14b,SHW+14a], linear [SHW+14a], impossible differential [CJF+16,ST16], zero-correlation linear [CJF+16], and integral characteristics with division property [XZBL16]. The use of MILP for the integral characteristic with division property is expanded in this paper.

The MILP problem is an optimization or feasibility program where variables are restricted to integers. We create an MILP model $\mathcal{M}$, which consists of variables $\mathcal{M}.var$, constraints $\mathcal{M}.con$, and an objective function $\mathcal{M}.obj$. As an example, let us consider the following optimization program.

Example 1.

$$\begin{aligned}
\mathcal{M}.var &\leftarrow x, y, z \text{ as binary.}\\
\mathcal{M}.con &\leftarrow x + 2y + 3z \leq 4\\
\mathcal{M}.con &\leftarrow x + y \geq 1\\
\mathcal{M}.obj &\leftarrow \text{maximize } x + y + 2z
\end{aligned}$$

The answer of the model $\mathcal{M}$ is 3, where $(x, y, z) = (1, 0, 1)$.

MILP solver can solve such optimization problem, and it returns *infeasible* if there is no feasible solution. Moreover, if there is no objective function, the MILP solver only evaluates whether this model is feasible or not.

We used Gurobi optimization as the solver in our experiments [Inc15].

2.2 Cube Attack

The cube attack is a key-recovery attack proposed by Dinur and Shamir in 2009 [DS09] and is the extension of the higher-order differential cryptanalysis [Lai94].

Let $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)$ and $\boldsymbol{v} = (v_1, v_2, \ldots, v_m)$ be n secret variables and m public variables, respectively. Then, the symmetric-key cryptosystem is represented as $f(\boldsymbol{x}, \boldsymbol{v})$, where f denotes a polynomial and the size of input and output is $n + m$ bits and 1 bit, respectively. In the case of stream ciphers, $\boldsymbol{x}$ is the secret key, $\boldsymbol{v}$ is the initialization vector (iv), and $f(\boldsymbol{x}, \boldsymbol{v})$ is the first bit of the key stream. The core idea of the cube attack is to simplify the polynomial by computing the higher-order differential of $f(\boldsymbol{x}, \boldsymbol{v})$ and to recover secret variables from the simplified polynomial.

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 $t_I = v_{i_1} \cdots v_{i_{|I|}}$. Then, we can decompose $f(\boldsymbol{x}, \boldsymbol{v})$ as

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

where $p(\boldsymbol{x}, \boldsymbol{v})$ is independent of $\{v_{i_1}, v_{i_2}, \ldots, v_{i_{|I|}}\}$ and the effective number of input variables of p is $n + m - |I|$ bits. Moreover, $q(\boldsymbol{x}, \boldsymbol{v})$ misses at least one variable from $\{v_{i_1}, v_{i_2}, \ldots, v_{i_{|I|}}\}$.

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

$$\begin{aligned}\bigoplus_{C_I} f(\boldsymbol{x}, \boldsymbol{v}) &= \bigoplus_{C_I} t_I \cdot p(\boldsymbol{x}, \boldsymbol{v}) + \bigoplus_{C_I} q(\boldsymbol{x}, \boldsymbol{v}) \\ &= p(\boldsymbol{x}, \boldsymbol{v}).\end{aligned}$$

The first term is reduced to $p(\boldsymbol{x}, \boldsymbol{v})$ because t_I becomes 1 for only one case in C_I. The second term is always canceled out because $q(\boldsymbol{x}, \boldsymbol{v})$ 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.

Blackbox Analysis. If the cube is appropriately chosen such that the superpoly is enough simplified to recover secret variables, the cube attack succeeds. However, $f(\boldsymbol{x}, \boldsymbol{v})$ in real symmetric-key cryptosystems is too complicated. Therefore, the cube attack regards f as a blackbox polynomial.

In the preprocessing phase, attackers first try out various cubes, change values of public and secret variables, and analyze the feature of the superpoly. The goal of this phase is to reveal the structure of $p(\boldsymbol{x}, \boldsymbol{v})$. Especially, the original cube attack searches for linear superpoly $p(\boldsymbol{x}, \boldsymbol{0})$ by the summation over the chosen cube. If the superpoly is linear,

$$p(\boldsymbol{x} \oplus \boldsymbol{x}', \boldsymbol{0}) = p(\boldsymbol{x}, \boldsymbol{0}) \oplus p(\boldsymbol{x}', \boldsymbol{0}) \oplus p(\boldsymbol{0}, \boldsymbol{0})$$

always holds for arbitrary $\boldsymbol{x}$ and $\boldsymbol{x}'$. By repeating this linearity test enough, attackers can know that the superpoly is linear with high probability, and the Algebraic Normal Form (ANF) of the superpoly is recovered by assuming its linearity.

In the online phase, attackers query to an encryption oracle by controlling only public variables and recover secret variables. Attackers evaluate the sum of $f(\boldsymbol{x}, \boldsymbol{v})$ over all values of the cube C_I. Since the sum is right hand side of the superpoly, the part of secret variables is recovered. Please refer to [DS09] and [ADMS09] to well understand the principle of the cube attack.

2.3 Higher-Order Differential Cryptanalysis and Division Property

Underlying mathematical background of the cube attack is the same as that of the higher-order differential attack. Unlike the cube attack, the common higher-order differential attack never regards the block cipher as a blackbox polynomial. Attackers analyze the structure of a block cipher and construct higher-order differential characteristics, where attackers prepare the set of chosen plaintexts such that the sum of corresponding ciphertexts of reduced-round block cipher is 0. After the proposal of the higher-order differential attack, many advanced techniques similar to the higher-order differential attack have been developed to analyze block ciphers, e.g., square attack [DKR97], saturation attack [Luc01], multi-set attack [BS01], and integral attack [KW02].

Division Property. At 2015, the division property, which is an improved technique to find higher-order differential (integral) characteristics for iterated ciphers, was proposed in [Tod15b]. Then, the bit-based variant was introduced in [TM16], and it is defined as follows[5].

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:*

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

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

We first evaluate the division property of the set of chosen plaintexts and then evaluate the division property of the set of corresponding ciphertexts by evaluating the propagation for every round function.

Some propagation rules for the division property are proven in [Tod15b,TM16]. Attackers determine indices $I = \{i_1, i_2, \ldots, i_{|I|}\} \subset \{1, 2, \ldots, n\}$ and prepare $2^{|I|}$ chosen plaintexts where variables indexed by I are taking all possible combinations of 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

$$\{\boldsymbol{k}\} \overset{\text{def}}{=} \mathbb{K}_0 \rightarrow \mathbb{K}_1 \rightarrow \mathbb{K}_2 \rightarrow \cdots \rightarrow \mathbb{K}_r,$$

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.

Propagation of Division Property with MILP. Evaluating the propagation of the division property is not easy because the size of $\mathbb{K}_i$ extremely increases. At ASIACRYPT 2016, Xiang et al. showed that the propagation is efficiently evaluated by using MILP [XZBL16]. First, they introduced the *division trail* as follows.

Definition 2 (Division Trail). *Let us consider the propagation of the division property $\{\boldsymbol{k}\} \overset{\text{def}}{=} \mathbb{K}_0 \rightarrow \mathbb{K}_1 \rightarrow \mathbb{K}_2 \rightarrow \cdots \rightarrow \mathbb{K}_r$. Moreover, for any 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 $(\boldsymbol{k}_0 \rightarrow \boldsymbol{k}_1 \rightarrow \cdots \rightarrow \boldsymbol{k}_r)$ an r-round division trail.*

Let E_k be the target r-round iterated cipher. Then, if there are division trails $\boldsymbol{k}_0 \xrightarrow{E_k} \boldsymbol{k}_r = \boldsymbol{e}_i$, attackers cannot know whether the ith bit of r-round ciphertexts is balanced or not. On the other hand, if we can prove that there is no division trail $\boldsymbol{k}_0 \xrightarrow{E_k} \boldsymbol{e}_i$, the ith bit of r-round ciphertexts is always balanced. Therefore, we have to evaluate all possible division trails to verify whether each bit of ciphertexts is balanced or not. In [Tod15b], [Tod15a], and [TM16], all possible division trails are evaluated by using a breadth-first search. Unfortunately, such a search requires enormous memory and time complexity. Therefore, it is practically infeasible to apply this method to iterated ciphers whose block length is not small.

MILP can efficiently solve this problem. We generate an MILP model that covers all division trails, and the solver evaluates the feasibility whether there are division trails from the input division property to the output one or not. If the solver guarantees that there is no division trail, higher-order differential (integral) characteristics are found.

[5] Two kinds of bit-based division property are proposed in [TM16]. In this paper, we only focus on the conventional bit-based division property.

Let copy, xor, and and be three fundamental operations, where 1 bit is copied into m bits in copy, the xor of m bits is computed in xor, and the and of m bits is computed in and. Note that MILP models for copy, xor, and and are sufficient to represent any circuit.

Proposition 1 (MILP Model for COPY). *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). *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). *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*.*

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

To accept multiple inputs and outputs, three propositions are generalized from the original ones shown in [XZBL16]. Moreover, Propositions 1 and 2 are also introduced in [SWW16]. Note that Proposition 3 includes redundant propagations of the division property, but they do not affect obtained characteristics.

3 How to Analyze Non-Blackbox Polynomials

The cube attack basically regards $f(\boldsymbol{x}, \boldsymbol{v})$ as a blackbox polynomial and analyzes it experimentally because real $f(\boldsymbol{x}, \boldsymbol{v})$ are too complicated to analyze the structure in detail. Such experimental analysis is often advantageous but has significant drawbacks, e.g., the size of cube is limited to the experimental range.

In this section, we propose a new technique to analyze the polynomial, where our technique never regards the polynomial as a blackbox and can analyze the structure in detail. Accurately, we propose a new application of the division property that enables us to analyze the Algebraic Normal Form (ANF) coefficients of f. Secret variables that are not involved in the superpoly of a cube C_I are efficiently identified by using our new method. As a result, we can estimate the time complexity that the ANF of the superpoly of a cube C_I is recovered.

3.1 What is Guaranteed by Division Property

We first revisit the definition of the division property and consider what the division property can do for stream ciphers.

Zero-Sum Integral Distinguisher. The trivial application is to find zero-sum integral distinguishers. Let us consider $f(\boldsymbol{x}, \boldsymbol{v})$ as a stream cipher, where $\boldsymbol{x}$ and $\boldsymbol{v}$ denote the secret and public variables, respectively, and f is designed by using iterative structure. For a cube C_I where the variables in $\{v_{i_1}, v_{i_2}, \ldots, v_{i_{|I|}}\}$ are taking all possible combinations of values, the propagation of the division property enables us to evaluate whether or not the sum of $f(\boldsymbol{x}, \boldsymbol{v})$ over all values of the cube C_I is balanced. Therefore, if the goal of attackers is to find zero-sum integral distinguishers, we can trivially use the division property.

Analysis of ANF Coefficients. Even if we can find a zero-sum integral distinguisher on stream ciphers, it is nontrivial to recover secret variables unlike block ciphers. Therefore, new techniques are required for the extension to the key-recovery attack.

We propose a novel application of the division property, where the division property is not used to find zero-sum integral distinguishers but used to analyze the ANF coefficients of f. Since our goal is to analyze the ANF coefficients, we do not need to distinguish public variables from secret ones. For the simplicity of notation, we consider $f(\boldsymbol{x})$ instead of $f(\boldsymbol{x}, \boldsymbol{v})$, and the ANF of $f(\boldsymbol{x})$ is represented as follows.

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

where $a_{\boldsymbol{u}}^f \in \mathbb{F}_2$ denotes the ANF coefficients. Then, the following Lemma is derived.

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. Let $\boldsymbol{k}$ be an n-dimensional bit vector. Then, assuming there is no division trail such that $\boldsymbol{k} \xrightarrow{f} 1$, $a_{\boldsymbol{u}}^f$ is always 0 for $\boldsymbol{u} \succeq \boldsymbol{k}$.*

Proof. According to $\boldsymbol{k}$, we first decompose $f(\boldsymbol{x})$ into

$$\begin{aligned} f(\boldsymbol{x}) &= \bigoplus_{\boldsymbol{u} \in \mathbb{F}_2^n | \boldsymbol{u} \succeq \boldsymbol{k}} a_{\boldsymbol{u}}^f \cdot \boldsymbol{x}^{\boldsymbol{u}} \oplus \bigoplus_{\boldsymbol{u} \in \mathbb{F}_2^n | \boldsymbol{u} \not\succeq \boldsymbol{k}} a_{\boldsymbol{u}}^f \cdot \boldsymbol{x}^{\boldsymbol{u}}, \\ &= \boldsymbol{x}^{\boldsymbol{k}} \cdot \bigoplus_{\boldsymbol{u} \in \mathbb{F}_2^n | \boldsymbol{u} \succeq \boldsymbol{k}} a_{\boldsymbol{u}}^f \cdot \boldsymbol{x}^{\boldsymbol{u} \oplus \boldsymbol{k}} \oplus \bigoplus_{\boldsymbol{u} \in \mathbb{F}_2^n | \boldsymbol{u} \not\succeq \boldsymbol{k}} a_{\boldsymbol{u}}^f \cdot \boldsymbol{x}^{\boldsymbol{u}}. \end{aligned}$$

Assume that there is no division trail such that $\boldsymbol{k} \xrightarrow{f} 1$. Then, no division trail guarantees that the sum of $f(\boldsymbol{x})$ over all values of the cube C_I is always balanced independent of x_i $(i \in \{1, 2, \dots, n\} - I)$. Namely,

$$\begin{aligned} \bigoplus_{C_I} f(\boldsymbol{x}) &= \bigoplus_{C_I} \left(\boldsymbol{x}^{\boldsymbol{k}} \cdot \bigoplus_{\boldsymbol{u} \in \mathbb{F}_2^n | \boldsymbol{u} \succeq \boldsymbol{k}} a_{\boldsymbol{u}}^f \cdot \boldsymbol{x}^{\boldsymbol{u} \oplus \boldsymbol{k}} \right) \\ &= \bigoplus_{\boldsymbol{u} \in \mathbb{F}_2^n | \boldsymbol{u} \succeq \boldsymbol{k}} a_{\boldsymbol{u}}^f \cdot \boldsymbol{x}^{\boldsymbol{u} \oplus \boldsymbol{k}} = 0 \end{aligned}$$

holds independent of x_i $(i \in \{1, 2, \dots, n\} - I)$. It holds only if $a_{\boldsymbol{u}}^f$ is always 0 for all $\boldsymbol{u}$ such that $\boldsymbol{u} \succeq \boldsymbol{k}$. □

Lemma 1 is very important observation for our attack.

3.2 Superpoly Recovery

The most important part of a cube attack is to recover the superpoly, and we simply call it the *superpoly recovery* in this paper. Since public variables $\boldsymbol{v}$ are known and chosen for attackers, the ANF of $p_{\boldsymbol{v}}(\boldsymbol{x}) = p(\boldsymbol{v}, \boldsymbol{x})$ is evaluated, and the goal is to recover $p_{\boldsymbol{v}}(\boldsymbol{x})$ whose $\boldsymbol{v}$ is fixed. Once the superpoly $p_{\boldsymbol{v}}(\boldsymbol{x})$ is recovered, attackers query the cube to an encryption oracle and compute the sum of $f(\boldsymbol{x}, \boldsymbol{v})$ over the cube. Then, attackers can get one polynomial about secret variables, and the secret variables are recovered from the polynomial.

The size of secret variables recovered from one superpoly depends on the structure of the superpoly $p_{\boldsymbol{v}}(\boldsymbol{x})$. If a balanced superpoly is used, one bit of information in involved secret variables is always recovered. If an unbalanced superpoly is used, the size of recovered secret variables is less than 1 bit but some information of secret variables is leaked to attackers. Moreover, it is possible to recover more bits of information in secret variables by exploiting multiple cubes. As an extreme case, if the superpoly is constant function, no secret variable is recovered, but it trivially implies constant-sum integral distinguishers. Therefore, the superpoly recovery directly brings vulnerability of symmetric-key cryptosystems, and some information of secret variables is always recovered unless the superpoly is constant function.

Previous Method to Recover Superpoly. The previous cube attack experimentally recovered the superpoly of a cube whose size is feasible for current computer. Therefore, not every superpoly can be evaluated. Linearity and quadraticity tests are repeated, and the superpoly is regarded as the linear or quadratic polynomial if these tests are sufficiently passes. Then, assuming the superpoly is linear or quadratic, the superpoly is recovered.

Analyze ANF Coefficients of Superpoly by Division Property. Lemma 1 implies that the division property can be used as a tool to analyze ANF coefficients of the superpoly. The following proposition is shown from Lemma 1 and is useful to evaluate the upper bound of the complexity to recover the ANF of the superpoly.

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 $(\boldsymbol{e}_j, \boldsymbol{k}_I) \xrightarrow{f} 1$, x_j 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

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

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

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

This superpoly is independent of x_j because u_j is always 0 and $(x_j)^0 = 1$. □

Algorithm 1 Evaluate secret variables by MILP

```
procedure attackFramework(MILP model $\mathcal{M}$, cube indices $I$)
    Let $\boldsymbol{x}$ be $n$ MILP variables of $\mathcal{M}$ corresponding to secret variables.
    Let $\boldsymbol{v}$ be $m$ MILP variables of $\mathcal{M}$ corresponding to public variables.
    $\mathcal{M}.con \leftarrow v_i = 1$ for all $i \in I$
    $\mathcal{M}.con \leftarrow v_i = 0$ for all $i \in (\{1, 2, \ldots, n\} - I)$
    $\mathcal{M}.con \leftarrow \sum_{i=1}^{m} x_i = 1$
    do
        solve MILP model $\mathcal{M}$
        if $\mathcal{M}$ is feasible then
            pick index $j \in \{1, 2, \ldots, n\}$ s.t. $x_j = 1$
            $J = J \cup \{j\}$
            $\mathcal{M}.con \leftarrow x_j = 0$
        end if
    while $\mathcal{M}$ is feasible
    return $J$
end procedure
```

We can evaluate which secret variables are involved to the superpoly of a given cube, and Algorithm 1 shows the algorithm supported by MILP. The input $\mathcal{M}$ is an MILP model, where the target stream cipher is represented by the context of the division property. How to construct $\mathcal{M}$ for each specific stream cipher is shown in each application in Sect. 5. First, we pick MILP variables $\boldsymbol{x}$ and $\boldsymbol{v}$ from $\mathcal{M}$, where $\boldsymbol{x}$ and $\boldsymbol{v}$ correspond to MILP variables for secret and public variables, respectively. As an example, in Algorithm 2 for Trivium, let $\boldsymbol{x} = (s_1^0, s_2^0, \ldots, s_{80}^0)$ and $\boldsymbol{v} = (s_{93}^0, s_{94}^0, \ldots, s_{172}^0)$. Then, to represent the input division property, elements of $\boldsymbol{v}$ indexed by I are constrained by 1, and the others are constrained by 0. Since at least one element in secret variables is additionally constrained to 1 in our cube attack, the sum of $\boldsymbol{x}$ is constrained to 1. Next, we solve this MILP model by using the solver. If $\mathcal{M}$ is infeasible, there is no involved secret variables in superpoly and $\bigoplus_{C_I} f(\boldsymbol{x}, \boldsymbol{v}) = p(\boldsymbol{x}, \boldsymbol{v})$ is always constant. If $\mathcal{M}$ is feasible, we can get a satisfying division trail and pick an index $j \in \{1, 2, \ldots, n\}$ such that $x_j = 1$ in the division trail. Then, x_j is involved to the superpoly and the index j is stored to a set J. Once we detect that x_j is involved, we additionally constrain $x_j = 0$. By repeating this procedure, we can get the set J whose elements are an index of secret variables involved to the superpoly.

After the analysis of the superpoly by using Algorithm 1, we know that only x_j $(j \in J)$ are involved to the superpoly of the cube C_I. Attackers choose a value in constant part of iv and prepare the cube C_I by flipping bits in I. They then recover the superpoly by trying out all possible combinations of secret variables $\{x_{j_1}, x_{j_2}, \ldots, x_{j_{|J|}}\}$. The time complexity to recover the superpoly is $2^{|I|+|J|}$. Therefore, if $|I| + |J|$ is smaller than the security bit level, we can efficiently recover the superpoly.

4 Toward Key Recovery

The time complexity to recover the superpoly is estimated in Sect. 3. As described in Sect. 3, the superpoly recovery directly brings vulnerability of stream ciphers. On the other hand, if our goal is to recover secret variables, we have to find a preferable superpoly that is close to balancedness for secret variables. Under the condition that we already get the cube index I and index of involved secret variables J by using Algorithm 1, our attack strategy to recover secret variables consists of three phases: *offline phase*, *online phase*, and *brute-force search phase*.

1. **Offline phase.** The goal of this phase is to find a preferable superpoly. Attackers choose a value in the constant part of iv, and prepare a cube by flipping bits in I. They then compute

$\bigoplus_{C_I} f(\boldsymbol{x}, \boldsymbol{v}) = p_{\boldsymbol{v}}(\boldsymbol{x})$ in local, where all possible combinations of secret variables $\{x_{j_1}, x_{j_2}, \ldots, x_{j_{|J|}}\}$ are tried out, and the superpoly is recovered. Finally, we search for the preferable superpoly by changing the constant part of iv.

2. **Online phase.** The goal of this phase is to recover the part of secret variables by using the preferable superpoly. After the balanced superpoly is given, attackers query the cube C_I to encryption oracle and get one bit $p_{\boldsymbol{v}}(\boldsymbol{x})$. Then, we get one polynomial about involved secret variables, and the half of values in involved secret variables is discarded because the superpoly is balanced.
3. **Brute-force search phase.** Attackers guess the remaining secret variables to recover the entire value in secret variables.

We cannot know whether the superpoly is balanced or not unless it is actually recovered, and the actual superpoly recovery requires $2^{|I|+|J|}$ time complexity. Therefore, if $|I| + |J|$ exceeds the experimental range, it is practically infeasible to search for preferable superpolys. As a consequence, we introduce the following two assumptions about collecting preferable superpolys.

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.*

Assumption 2 is weaker than Assumption 1 because the superpoly satisfying Assumption 1 always holds Assumption 2. As long as Assumption 2 holds, the size of recovered secret variables is less than 1 bit but some secret information is at least leaked to attackers. If Assumption 1 holds and such superpoly is used in the online phase, values in involved secret variables are divided in exactly half, i.e., $p_{\boldsymbol{v}}(\boldsymbol{x})$ is 0 for $2^{|J|-1}$ values and is 1 for the others. Therefore, we can recover one bit of information in secret variables.

4.1 Evaluating Time Complexity.

Assuming that Assumption 1 holds, we show the time complexity to recover the entire secret key. Then, the time complexity of the offline phase is estimated as $k \times 2^{|I|+|J|}$, where k denotes the required number of trials for finding a preferable superpoly. Note that we can expect that such superpoly can be reasonably found with high probability without trying out all possible values in involved secret variables. We evaluate a part of values in involved secret variables at random and check whether $p_{\boldsymbol{v}}(\boldsymbol{x})$ is almost balanced or not. If the output is highly biased for $\boldsymbol{x}$, the superpoly $p_{\boldsymbol{v}}$ is not preferable and changes to other values in the constant part of iv. The complexity of this method is $O(2^{|I|})$. Once we find an almost preferable superpoly, we entirely try out $2^{|J|}$ values in secret variables.

Even if the preferable superpoly is used, the size of recovered secret information is at most 1 bit. Therefore, when only one cube is used, the time complexity of the brute-force search phase is $2^{\kappa-1}$, where κ denotes the security bit level. Therefore, the total time complexity is

$$k \times 2^{|I|+|J|} + 2^{|I|} + 2^{\kappa-1}, \tag{1}$$

From Eq. (1), when $|I|+|J| = \kappa-1$, the total time complexity is greater than 2^{κ} because k is at least 1. Therefore, such cube is not applied to the key-recovery attack. Moreover, when $|I| + |J| = \kappa - 2$, this attack is valid only if the best case ($k = 1$), where a preferable superpoly is found in the first trial.

If only one cube is exploited, the dominant time complexity is always that for the brute-force search phase. When ℓ cubes are found in the evaluation phase and all found cubes are exploited, the total time complexity is reduced to

$$\ell \times \left(k \times 2^{|I|+|J|} + 2^{|I|}\right) + 2^{\kappa-\ell}.$$

However, this paper only focuses on the case that only one cube is exploited for the simplicity. Note that the detection of one cube brings at least cryptographic vulnerability.

5 Applications

We apply our general attack method to three NLFSE-based ciphers. The first target is Trivium [CP06], which is one of eSTREAM portfolio [est08] and one of the most analyzed stream ciphers. Another target is Grain128a [ÅHJM11], which is standardized by ISO/IEC 29167-13 [ISO15]. The final application is ACORN [Wu16], which is one of the 3rd round CAESAR candidates [cae14], and its design is based on stream ciphers.

5.1 Application to Trivium

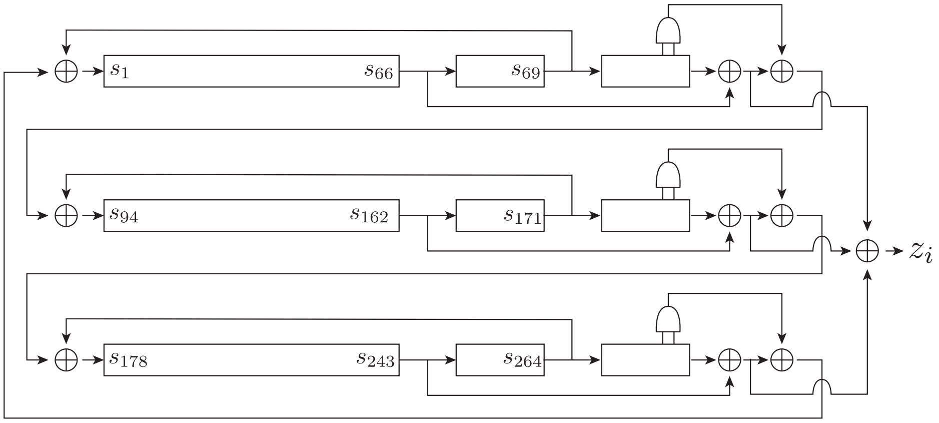

Fig. 1. Structure of Trivium

Specification. Trivium is an NLFSR-based stream cipher, and the internal state is represented by 288-bit state $(s_1, s_2, \ldots, s_{288})$. Figure 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.

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

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.

Algorithm 2 MILP model of division property for Trivium

```
procedure TriviumCore($\mathcal{M}, \boldsymbol{x}, i_1, i_2, i_3, i_4, i_5$)
    $\mathcal{M}.var \leftarrow y_{i_1}, y_{i_2}, y_{i_3}, y_{i_4}, y_{i_5}, z_1, z_2, z_3, z_4, a$ as binary
    $\mathcal{M}.con \leftarrow y_{i_j} = x_{i_j} - z_j$ for all $j \in \{1, 2, 3, 4\}$
    $\mathcal{M}.con \leftarrow a \geq z_3$
    $\mathcal{M}.con \leftarrow a \geq z_4$
    $\mathcal{M}.con \leftarrow y_{i_5} = x_{i_5} + a + z_1 + z_2$
    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
procedure TriviumEval(round $R$)
    Prepare empty MILP Model $\mathcal{M}$
    $\mathcal{M}.var \leftarrow s_i^0$ for $i \in \{1, 2, \ldots, 288\}$
    for $r = 1$ to $R$ do
        $(\mathcal{M}, \boldsymbol{x})$ = TriviumCore($\mathcal{M}, \boldsymbol{s}^{r-1}, 66, 171, 91, 92, 93$)
        $(\mathcal{M}, \boldsymbol{y})$ = TriviumCore($\mathcal{M}, \boldsymbol{x}, 162, 264, 175, 176, 177$)
        $(\mathcal{M}, \boldsymbol{z})$ = TriviumCore($\mathcal{M}, \boldsymbol{y}, 243, 69, 286, 287, 288$)
        $\boldsymbol{s}^r = \boldsymbol{z} \ggg 1$
    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
```

MILP Model. `TriviumEval` in Algorithm 2 generates MILP model $\mathcal{M}$ as the input of Algorithm 1, and the model $\mathcal{M}$ can evaluate all division trails for Trivium whose initialization rounds are reduced to R. `TriviumCore` in Algorithm 2 generates MILP variables and constraints for each update function of register. Since one `TriviumCore` creates 10 MILP variables and 7 constraints, one update function creates 30 MILP variables and 21 constraints. Therefore, generated MILP model $\mathcal{M}$ consists of $288 + 30R$ MILP variables and $21R + 282 + 1$ MILP constraints. Note that constraints by the input division property are operated by Algorithm 1.

Table 2. Involved secret variables in the superpoly of the cube $C_{\{1,11,21,31,41,51,61,71\}}$.

# rounds	involved secret variables J	size of J
576	$48, 73, 74, 75$	4
577	$40, 65, 66, 67$	4
583	$48, 50, 62, 63, 66, 73, 74, 75, 76, 77$	10
584	$48, 50, 60, 61, 66, 67, 73, 74, 75, 76, 77$	11
586	$20, 30, 40, 45, 46, 47, 55, 56, 57, 58, 61, 65, 66, 67$	14
587	$30, 55, 56, 57, 58$	5
590	60	1
591	$23, 24, 25, 66, 67$	5
592	$\cdots$	25
593	$\cdots$	57
594	$\cdots$	47

Experimental Verification. We implemented the MILP model $\mathcal{M}$ for the propagation of the division property on TRIVIUM and evaluated involved secret variables by using Algorithm 1, where Gurobi optimizer [Inc15] was used as the solver of MILP. Before the theoretical evaluation, we verify our attack and implementation by using small cube as $I = \{1, 11, 21, 31, 41, 51, 61, 71\}$. Table 2 summarizes involved secret variables from 576 to 594 rounds.

Example 2 (Verification of Our Attack against 590-round TRIVIUM*).* We actually execute the offline phase against 590-round TRIVIUM, and only K_{60} is involved to the superpoly. We randomly chose 100 superpolys by changing the constant part of iv and evaluated the sum of the cube. As a result, the sum is always 0 independent of K_{60} in 42 superpolys, where `0x00CA6124DE5F12043D62` is its example of the constant part of iv. Moreover, the sum corresponds to the value of K_{60} in 22 superpolys, where `0x2F0881B93B251C7079F2` is its example. Then, the ANF of the superpoly is represented as

$$p_{\boldsymbol{v}}(\boldsymbol{x}) = x_{60}.$$

Finally, the sum corresponds to the value of $K_{60} \oplus 1$ in 36 superpolys, where `0x5745A1944411D1374828` is its example. Then, the ANF of the superpoly is represented as

$$p_{\boldsymbol{v}}(\boldsymbol{x}) = x_{60} \oplus 1.$$

Balanced superpolys are preferable, and we found $22 + 36 = 58$ such superpolys. Therefore, the required number of trials for finding preferable superpolys is about $k = 2$.

Example 3 (Verification of Our Attack against 591-round TRIVIUM*).* We execute the offline phase against 591-round TRIVIUM, and $K_{23}, K_{24}, K_{25}, K_{66}, K_{67}$ are involved to the superpoly. Similarly to the attack against 590 rounds, we randomly chose 100 superpolys by changing the constant part of iv and evaluated the sum of the given cube. As a result, the sum is always 0 independent of 5 involved secret variables in 64 superpolys, where `0x39305FDD295BDACD2FBE` is its example of the constant part of iv. There are 11 superpolys such that the sum is 1 only when

$$K_{23} \| K_{24} \| K_{25} \| K_{66} \| K_{67} \in \{\texttt{00}, \texttt{05}, \texttt{08}, \texttt{0D}, \texttt{10}, \texttt{15}, \texttt{19}, \texttt{1C}\}$$

as the hexadecimal notation, where 0x03CC37748E34C601ADF5 is its example of the constant part of iv. Then, the ANF of the superpoly is represented as

$$p_{\boldsymbol{v}}(\boldsymbol{x}) = (x_{66} \oplus 1)(x_{23}x_{24} \oplus x_{25} \oplus x_{67} \oplus 1).$$

There are 9 superpolys such that the sum is 1 when

$$K_{23} \| K_{24} \| K_{25} \| K_{66} \| K_{67} \in \{\texttt{02}, \texttt{07}, \texttt{0A}, \texttt{0F}, \texttt{12}, \texttt{17}, \texttt{1B}, \texttt{1E}\}$$

as the hexadecimal notation, where 0x78126459CB2384E6CCCE is its example of the constant part of iv. Then, the ANF of the superpoly is represented as

$$p_{\boldsymbol{v}}(\boldsymbol{x}) = x_{66}(x_{23}x_{24} \oplus x_{25} \oplus x_{67} \oplus 1).$$

Moreover, there are 16 superpolys such that the sum is 1 when the value of $K_{23} \| K_{24} \| K_{25} \| K_{66} \| K_{67}$ belongs to

$$\{\texttt{00}, \texttt{02}, \texttt{05}, \texttt{07}, \texttt{08}, \texttt{0A}, \texttt{0D}, \texttt{0F}, \texttt{10}, \texttt{12}, \texttt{15}, \texttt{17}, \texttt{19}, \texttt{1B}, \texttt{1C}, \texttt{1E}\}$$

as the hexadecimal notation, where 0x644BD671BE0C9241481A is its example of the constant part of iv. Then, the ANF of the superpoly is represented as

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

and this superpoly is balanced. Note that x_{66} is not involve to this superpoly. Balanced superpolys are preferable, and we found 16 such superpolys. Therefore, the required number of trials for finding preferable superpolys is about $k = 6$.

Table 3. Summary of theoretical cube attacks on Trivium. The time complexity in this table shows the time complexity to recover the superpoly.

#rounds	$\lvert I \rvert$	involved secret variables J	time complexity
800	44	8, 33, 34, 35, 48, 59, 60, 61, 64, 73, 74, 75	$2^{44+12} = 2^{56}$
802	46	32, 34, 57, 58, 59, 60, 61, 62	$2^{46+8} = 2^{54}$
805	49	14, 39, 40, 41, 42, 44, 46, 58, 67,...,73	$2^{49+15} = 2^{64}$
806	51	42, 67, 68, 69	$2^{51+4} = 2^{55}$
808	52	26, 28, 40, 51, 52, 53, 54, 55, 58, 65, 66, 67	$2^{52+12} = 2^{64}$
809	53	24, 26, 36, 38, 40, 49,...,56, 58, 61,...,67, 77,...,80	$2^{53+25} = 2^{78}$
814	54	32, 34, 57, 58, 59, 60, 61	$2^{54+7} = 2^{61}$
816	55	6, 31, 32, 33, 48, 50, 52, 57,...,60, 62, 73,...,79	$2^{55+19} = 2^{74}$
818	58	34, 59, 60, 61	$2^{58+4} = 2^{62}$
819	61	15, 17, 40, 41, 42, 43, 44, 58	$2^{61+8} = 2^{69}$
820	62	15, 26, 40, 41, 42, 51, 52, 53	$2^{62+8} = 2^{70}$
822	64	42, 67, 68, 69	$2^{64+4} = 2^{68}$
825	65	52, 54, 66, 77, 78, 79, 80	$2^{65+7} = 2^{72}$
829	66	23, 25, 26, 27, 36, 42, 56, 67, 68, 69	$2^{66+10} = 2^{76}$
830	69	1, 37, 42, 56, 67, 68, 69	$2^{69+7} = 2^{76}$
831	71	49, 74, 75, 76	$2^{71+4} = 2^{75}$
832	72	34, 58, 59, 60, 61	$2^{72+5} = 2^{77}$

For any size of cube $|I|$, the odd index $1, 3, \ldots, 79$ and even index $2, 4, \ldots, 2(|I| - 40)$ is chosen as cube indices.

Theoretical Results. As experimental verification shows, Assumption 1 holds for Trivium in small example. Therefore, we can expect that theoretically recovered superpolys also fulfill Assumption 1.

Cube indices are chosen as the following in our experiments: the odd index $1, 3, \ldots, 2|I| - 1$ is chosen, and the even index $2, 4, \ldots, 2(|I| - 40)$ is additionally chosen. Then, we exhaustively evaluated involved secret variables, and Table 3 summarizes the result in our theoretical cube attack. Table 3 shows indices of involved secret variables and the time complexity for the superpoly recovery against Trivium with at least 800 initialization rounds. Since the previous best key-recovery attack is 799 rounds, all results at least improve the current best key-recovery attack. Under the condition that the time complexity for the superpoly recovery is less than 2^{79}, the largest number of initialization rounds that we can attack is 832 rounds. Compared with previous best key-recovery attack, it updates $832 - 799 = 33$ rounds.

We do not have plausible evidence that our choice of cube indices is appropriate, and the choice is still difficult because we need to try out $\binom{80}{|I|}$ cubes when we want to evaluate all cubes whose size is $|I|$. How to choose appropriate cubes is left as an open question.

5.2 Application to Grain128a

Specification. Grain128a 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 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

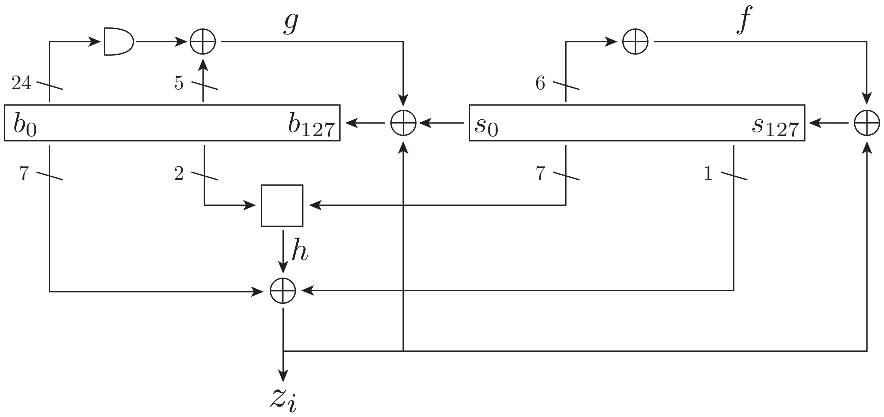

Fig. 2. Structure of Grain128a

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

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

$$\begin{aligned}
g \leftarrow\ & b_0 + b_{26} + b_{56} + b_{91} + b_{96} \\
& + b_3 b_{67} + b_{11} b_{13} + b_{17} b_{18} + b_{27} b_{59} + b_{40} b_{48} + b_{61} b_{65} + b_{68} b_{84} \qquad (2) \\
& + b_{88} b_{92} b_{93} b_{95} + b_{22} b_{24} b_{25} + b_{70} b_{78} b_{82}.
\end{aligned}$$

$$f \leftarrow s_0 + s_7 + s_{38} + s_{70} + s_{81} + s_{96} \tag{3}$$

$$h \leftarrow b_{12} s_8 + s_{13} s_{20} + b_{95} s_{42} + s_{60} s_{79} + b_{12} b_{95} s_{94} \tag{4}$$

$$z \leftarrow h + s_{93} + \sum_{j \in A} b_j \tag{5}$$

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

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

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. Figure 2 shows the state update function of Grain128a.

Algorithm 3 MILP model for the initialization of Grain128a

```
procedure Grain128aEval(round $R$)
    Prepare empty MILP Model $\mathcal{M}$
    $\mathcal{M}.var \leftarrow b_i^0$ for $i \in \{0, 1, \ldots, 127\}$ as binary
    $\mathcal{M}.var \leftarrow s_i^0$ for $i \in \{0, 1, \ldots, 127\}$ as binary
    for $r = 1$ to $R$ do
        $(\mathcal{M}, \boldsymbol{b}', \boldsymbol{s}', z) =$ funcZ$(\mathcal{M}, \boldsymbol{b}^{r-1}, \boldsymbol{s}^{r-1})$
        $\mathcal{M}.var \leftarrow z_g, z_f$ as binary
        $\mathcal{M}.con \leftarrow z = z_g + z_f$
        $(\mathcal{M}, \boldsymbol{b}'', g) =$ funcG$(\mathcal{M}, \boldsymbol{b}')$
        $(\mathcal{M}, \boldsymbol{s}'', f) =$ 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
        $\mathcal{M}.var \leftarrow b_{127}^r, s_{127}^r$ as binary
        $\mathcal{M}.con \leftarrow b''_0 = 0$
        $\mathcal{M}.con \leftarrow b_{127}^r = g + s''_0 + z_g$
        $\mathcal{M}.con \leftarrow s_{127}^r = f + z_f$
    end for
    $(\mathcal{M}, \boldsymbol{b}', \boldsymbol{s}', z) =$ 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
```

MILP Model. `Grain128aEval` in Algorithm 3 generates MILP model $\mathcal{M}$ as the input of Algorithm 1, and the model $\mathcal{M}$ can evaluate all division trails for Grain128a whose initialization rounds are reduced to R. `funcZ` generates MILP variables and constraints for Eq. (4) and Eq. (5), and it consists of 45 MILP variables and 32 MILP constraints. `funcG` generates MILP variables and constraints for Eq. (2), and it consists of 70 MILP variables and 55 MILP constraints. `funcF` generates MILP variables and constraints for Eq. (3), and it consists of 13 MILP variables and 7 MILP constraints. As a result, the MILP model for every round consists of $45 + 70 + 13 + 4 = 132$ MILP variables and $32 + 55 + 7 + 4 = 98$ MILP constraints. Therefore, generated MILP model $\mathcal{M}$ consists of $256 + 45 + 132R$ MILP variables and $98R + 32 + 256 + 1$ MILP constraints. Note that constraints by the input division property are operated by Algorithm 1.

Experimental Verification. We implemented the MILP model $\mathcal{M}$ for the propagation of the division property on Grain128a and evaluated involved secret variables by using Algorithm 1. To verify our attack and implementation, the offline phase is executed by using small cube as $I = \{1, 2, \ldots, 9\}$.

Example 4 (Verification of Our Attack against 106-round Grain128a). The cube $C_{\{1,2,3,\ldots,9\}}$ brings the superpoly that involves only seven secret variables, (K_{46}, K_{53}, K_{85}, K_{119}, K_{122}, K_{126}, and K_{127}), and this result comes out of Algorithm 1. In our experiments, the Hamming weight of all superpolys $p_{\boldsymbol{v}}(\boldsymbol{x})$ is only 4. Specifically, in arbitrary iv satisfying $IV_{76} = 0$, $p_{\boldsymbol{v}}(\boldsymbol{x})$ is 1 only when the involved secret variables are represented as

$$\begin{aligned}(K_{46}, K_{53}, K_{85}, K_{119}, K_{122}, K_{126}, K_{127}) = &(*, 1, 0, 1, 1, 1, 1) \text{ or}\\ &(*, 0, 1, 1, 1, 1, 1),\end{aligned}$$

where $*$ is any bit. Moreover, in arbitrary iv satisfying $IV_{76} = 1$, $p_{\boldsymbol{v}}(\boldsymbol{x})$ is 1 only when the involved secret variables are represented as

$$\begin{aligned}(K_{46}, K_{53}, K_{85}, K_{119}, K_{122}, K_{126}, K_{127}) = &(*, 1, 0, 1, 0, 1, 1) \text{ or}\\ &(*, 0, 1, 1, 0, 1, 1).\end{aligned}$$

Namely, the superpoly is represented as

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

This superpoly is independent of x_{46}. Moreover, it is not balanced, and the Hamming weight of $p_{\boldsymbol{v}}(\boldsymbol{x})$ is 2 for six involved input bits. Therefore, the recovered bit of information in secret variables is represented as

$$\left|\log_2\left(\frac{2 \times \frac{2}{2^6} + (62 \times \frac{62}{2^6})}{2^6}\right)\right| \approx 0.09.$$

Double bit of information can be recovered by flipping the bit IV_{76}, but the recovered information is still smaller than 1.

Theoretical Results. We cannot find superpolys satisfying Assumption 1 in our experiments using small cube. On the other hand, Assumption 2 holds. Therefore, we can expect that theoretically recovered superpolys also fulfill Assumption 2, and it leaks at least some information in secret variables which is smaller than 1 bit. Moreover, by collecting these superpolys, we can expect that multiple bits of information in secret variables are recovered.

Table 4 shows indices of involved secret variables and the time complexity for the superpoly recovery against Grain128a. Since the previous best attack is 177 rounds in the single-key setting, all results at least improve the current best key-recovery attack. Under the condition that the time complexity for the superpoly recovery is less than 2^{127}, the largest number of initialization rounds that we can attack is 183 rounds. Compared with previous best distinguishing attack, it updates $183 - 177 = 6$ rounds. Moreover it allows for some key recovery.

Table 4. Summary of theoretical cube attacks on Grain128a. The time complexity in this table shows the time complexity to recover the superpoly.

#rounds	$\lvert I \rvert$	involved secret variables J	time complexity
182	88 †	36, 40, 51, 52, 53, 54, 55, 56, 61, 62, 69, 79, 81, 82, 121, 122, 126, 127	$2^{88+18} = 2^{106}$
183	92 ‡	48, 49, 50, 51, 52, 54, 55, 61, 63, 83, 84, 90, 93, 95, 120, 128	$2^{92+16} = 2^{108}$

† Following set of indices $I = \{1, ..., 40, 42, 44, \ldots, 51, 53, ..., 87, 89, 91, 93, 95\}$ is used as the cube.
‡ Following set of indices $I = \{1, ..., 51, 53, ..., 91, 93, 95\}$ is used as the cube.

5.3 Application to ACORN

Specification. 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

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

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

$$\begin{aligned} maj(x, y, z) &= (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z), \\ ch(x, y, z) &= (x \wedge y) \oplus ((x \oplus 1) \wedge z). \end{aligned}$$

Then, the update function is given as follows.

$$\begin{aligned} &S_{289} \leftarrow S_{289} \oplus S_{235} \oplus S_{230} \\ &S_{230} \leftarrow S_{230} \oplus S_{196} \oplus S_{193} \\ &S_{193} \leftarrow S_{193} \oplus S_{160} \oplus S_{154} \\ &S_{154} \leftarrow S_{154} \oplus S_{111} \oplus S_{107} \\ &S_{107} \leftarrow S_{107} \oplus S_{66} \oplus S_{61} \\ &S_{61} \leftarrow S_{61} \oplus S_{23} \oplus S_0 \\ &ks = KSG128(S) \\ &f = FBK128(S, ca, cb) \\ &(S_0, S_1, \ldots, S_{291}, S_{292}) \leftarrow (S_1, s_2, \ldots, S_{292}, f \oplus m) \end{aligned}$$

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. Figure 3 shows the structure of ACORN. Please refer to [Wu16] in detail.

MILP Model. `ACORNEval` in Algorithm 4 generates MILP model $\mathcal{M}$ as the input of Algorithm 1, and the model $\mathcal{M}$ can evaluate all division trails for ACORN whose initialization rounds are reduced

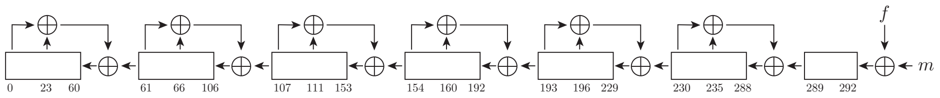

Fig. 3. Structure of ACORN

Algorithm 4 MILP model for the initialization of ACORN

```
procedure ACORNEval(round $R$)
    Prepare empty MILP Model $\mathcal{M}$
    $\mathcal{M}.var \leftarrow K_i$ for $i \in \{1, 2, \dots, 128\}$ as binary
    $\mathcal{M}.var \leftarrow IV_i$ for $i \in \{1, 2, \dots, 128\}$ as binary
    $\mathcal{M}.var \leftarrow S_i^0$ for $i \in \{0, 1, \dots, 292\}$ as binary
    for $r = 1$ to $R$ do
        $(\mathcal{M}, \boldsymbol{T})$ = xorFB$(\mathcal{M}, \boldsymbol{S}^{r-1}, 289, 235, 230)$
        $(\mathcal{M}, \boldsymbol{U})$ = xorFB$(\mathcal{M}, \boldsymbol{T}, 230, 196, 193)$
        $(\mathcal{M}, \boldsymbol{V})$ = xorFB$(\mathcal{M}, \boldsymbol{U}, 193, 160, 154)$
        $(\mathcal{M}, \boldsymbol{W})$ = xorFB$(\mathcal{M}, \boldsymbol{V}, 154, 111, 107)$
        $(\mathcal{M}, \boldsymbol{X})$ = xorFB$(\mathcal{M}, \boldsymbol{W}, 107, 66, 61)$
        $(\mathcal{M}, \boldsymbol{Y})$ = xorFB$(\mathcal{M}, \boldsymbol{X}, 61, 23, 0)$
        $(\mathcal{M}, \boldsymbol{Z}, ks)$ = ksg128$(\mathcal{M}, \boldsymbol{Y}, r)$
        $(\mathcal{M}, \boldsymbol{A}, f)$ = fbk128$(\mathcal{M}, \boldsymbol{Z}, r)$
        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
            $\mathcal{M}.con \leftarrow S_{292}^r = f \oplus IV_{r-128}$
        else
            $\mathcal{M}.var \leftarrow TK_r$ as binary
            $\mathcal{M}.con \leftarrow S_{292}^r = f \oplus TK_r$
        end if
    end for
    for $i = 0$ to 127 do
        $\mathcal{M}.con \leftarrow K_i = \sum_j TK_{i+128\times j}$
    end for
    $(\mathcal{M}, \boldsymbol{T})$ = xorFB$(\mathcal{M}, \boldsymbol{S}^R, 289, 235, 230)$
    $(\mathcal{M}, \boldsymbol{U})$ = xorFB$(\mathcal{M}, \boldsymbol{T}, 230, 196, 193)$
    $(\mathcal{M}, \boldsymbol{V})$ = xorFB$(\mathcal{M}, \boldsymbol{U}, 193, 160, 154)$
    $(\mathcal{M}, \boldsymbol{W})$ = xorFB$(\mathcal{M}, \boldsymbol{V}, 154, 111, 107)$
    $(\mathcal{M}, \boldsymbol{X})$ = xorFB$(\mathcal{M}, \boldsymbol{W}, 107, 66, 61)$
    $(\mathcal{M}, \boldsymbol{Y})$ = xorFB$(\mathcal{M}, \boldsymbol{X}, 61, 23, 0)$
    $(\mathcal{M}, \boldsymbol{Z}, ks)$ = 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
```

to R. `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.

Table 5. Summary of theoretical cube attacks on ACORN. The time complexity in this table shows the time complexity to recover the superpoly.

#rounds	$\|I\|$	involved secret variables J	time complexity
647	35 †	1, 2, 3, 5, 6, 7, 8, 9, 10, 11, 12, 13, 15, 16, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 31, 32, 33, 35, 40, 45, 49, 52, 55, 57, 60, 61, 62, 65, 66, 94, 99	$2^{35+43} = 2^{78}$
649	35 †	1, 2,...,39, 41,...,49, 52,...,69, 78, 86, 96, 97, 98, 100, 101, 102	$2^{35+74} = 2^{109}$
704	64 ‡	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	$2^{64+58} = 2^{122}$

† Following set of indices $I = \{1, 2, \ldots, 16, 22, 29, 31, 113, 114, \ldots, 128\}$ is used as the cube.
‡ Following set of indices $I = \{1, 2, \ldots, 32, 97, 98, \ldots, 128\}$ is used as the cube.

If there are zero constant bit in input of $KSG128$ and $FBK128$, the propagation of the division property for two functions `ksg128` and `fbk128` is limited. For example, when $maj(x, y, z)$ is computed under the condition $y = z = 0$, this function is represented as

$$maj(x, 0, 0) = 0,$$

and the division property of x never propagates to the output of maj. Such limitations of the propagation only happens in the first several rounds because the state $\boldsymbol{S}$ is initialized to $\mathbf{0}$. To control this behavior, there is the current number of rounds as the input of `ksg128` and `fbk128`. Note that constraints by the input division property are operated by Algorithm 1.

Experimental Verification. We implemented the MILP model $\mathcal{M}$ for the propagation of the division property on ACORN and evaluated involved secret variables by using Algorithm 1. We searched the small cube such that $|I| + |J|$ is practically feasible, and the following small cube

$$C_{\{1,2,3,4,5,8,20,125,126,127,128\}}$$

is used to verify our attack and implementation.

Example 5 (Verification of Our Attack against 517-round ACORN). The cube $C_{\{1,2,3,4,5,8,20,125,126,127,128\}}$ brings the superpoly that involves only nine secret variables, (K_6, K_8, K_{10}, K_{11}, K_{12}, K_{15}, K_{16}, K_{45}, and K_{49}), and this result comes out of Algorithm 1. We try out 100 randomly chosen constant part of iv. As a result, all superpolys $p_{\boldsymbol{v}}(\boldsymbol{x})$ are balanced independent of the value of the constant part of iv. Specifically, $p_{\boldsymbol{v}}(\boldsymbol{x})$ corresponds to the sum of involved secret variables. Namely, the superpoly is represented as

$$p_{\boldsymbol{v}}(\boldsymbol{x}) = x_6 \oplus x_8 \oplus x_{10} \oplus x_{11} \oplus x_{12} \oplus x_{15} \oplus x_{16} \oplus x_{45} \oplus x_{49}.$$

Theoretical Results. As experimental verification shows, Assumption 1 holds for ACORN in small example. Therefore, we can expect that theoretically recovered superpolys also fulfill Assumption 1.

Table 5 shows indices of involved secret variables and the time complexity for the superpoly recovery against ACORN. Since the previous best attack is 503 rounds, all results at least improve the current best key-recovery attack. As far as we searched various cubes, the largest number of initialization rounds that we can attack is 704 rounds, where the cube size is 64 and the number of involved secret variables is 58. Compared with previous best key-recovery attack, it updates $704 - 503 = 201$ rounds.

6 Discussions

6.1 Validity of Assumption 1 and 2

Whether the two assumptions hold depends on the structure of analyzed ciphers. In the three applications shown in this paper, we could easily find balanced superpoly for Trivium and ACORN by actually evaluating the offline phase using small cube. Therefore, we can expect that Assumption 1 holds in theoretical recovered superpolys for these two ciphers. On the other hand, we could not find balanced superpolys for Grain128a. This implies that Assumption 1 does not hold in theoretical recovered superpolys for Grain128a. However, since we could easily find non-constant superpolys, we can expect that Assumption 2 holds.

Note that Assumption 1 is introduced to estimate the time complexity to recover the entire secret key, and some information of secret variables is leaked to attackers even if only Assumption 2 holds. Moreover, even if both assumptions do not hold, the recovered superpoly is useful for distinguishing attacks. Therefore, if the superpoly recovery is more efficient than the brute-force attack, it immediately brings some vulnerability of symmetric-key cryptosystems. Therefore, the time complexity for the superpoly recovery discussed in this paper is very important.

Conventional cube attacks also have similar assumption because they experimentally verify whether the superpoly is linear, quadratic, or not. For example, in [DS09], the authors judged that the superpoly is linear if the superpoly passes at least 100 linearity tests. Moreover, Fouque and Vannet also introduced heuristic linearity and quadraticity tests in [FV13], where the superpoly is judged as linear and quadratic if it passes constant-order linearity and quadraticity tests, respectively. These constant-order tests may fail if there are terms of the superpoly that are highly biased. For example, assuming that the superpoly is represented as $K_1 + f(K_2, K_3, K_4, ..., K_{32})$ where f is unbalanced, the test used in previous cube attacks may judge the superpoly as K_1 in error. Namely, the conventional cube attack also assumes that the superpoly is balanced for each involved secret variables, and it fails to recover secret variables if this assumption is incorrect.

6.2 Multiple-Bits Recovery only from One Cube

There is a possibility that we can recover multiple bits from given cube by changing a value in constant part of iv. Indeed, Example 3 recovers more than one bit of information in secret variables by using an $\boldsymbol{v}$ = `0x03CC37748E34C601ADF5` or $\boldsymbol{v}$ = `0x78126459CB2384E6CCCE` together with $\boldsymbol{v}$ = `0x644BD671BE0C9241481A`. Moreover, two bits of information in secret variables are recovered if we find two independent balanced superpolys. On the other hand, the superpoly must be enough simplified for the key recovery. While we may be able to recover multiple bits only from one cube by changing values of the constant part of iv when the number of involved secret variables is high, we cannot claim that there are many independent balanced superpolys when the number of involved secret variables is small. Therefore, we do not claim that multiple bits are recovered from one cube by changing values of the constant part of iv.

6.3 Comparison with Previous Techniques

There is previous work that exploits non-randomness in high degree monomial structure in the ANF for the key recovery of stream ciphers: In [FKM08], it is examined if every key bit in the parametrized expression of a coefficient of some high degree monomial in iv bits does occur, or more generally, how much influence each key bit does have on the value of the coefficient. If a coefficient depends on less than all key bits, this fact is exploited to filter those keys which do not satisfy the imposed value for the coefficient. As opposed to the present work, this method is mostly statistical in nature, whereas division property is fully algebraic.

Secondly, in [KMN10], conditions are identified on the internal state to obtain a deterministic differential characteristic for some large number of rounds. Depending on whether these conditions involve public variables only, or also key variables, distinguishing and partial key-recovery attacks are derived. The technique is extended to (conditional) higher order differentials and enables to distinguish reduced round versions of some stream ciphers, and to recover parts of the key. Again, this method is quite different from the methods of this paper, and is not purely algebraic.

A third more recent approach is dynamic cube attack [DS11]. In contrast to standard cube attack that finds the key by solving a system of (linear) equations in the key bits, dynamic cube attack recovers the secret key by exploiting distinguishers obtained from cube testers. Dynamic cube attacks aim at creating lower degree representations of the given cipher. This method has been successfully applied to break the stream cipher Grain-128 [DGP+11]. All the previous methods share the restriction that they are experimental rather than theoretical, i.e., they are dependent on computing with cubes as large as practically feasible.

7 Exploiting Cube whose Non-Cube Bits Fill Up With 0

The non-cube bits in iv are set to 0 in the previous cube attack [DS09,FV13,Liu17]. Especially, when Trivium is analyzed, such cube is more useful than the cube whose non-cube bits are random. We can also exploit such cube, and it brings more powerful cube attacks. On the other hand, since we cannot collect many superpolys by changing the non-cube bits, two assumptions introduced in this paper become speculative. However, as we said in previous section, the superpoly recovery immediately brings vulnerability like the distinguisher or key recovery.

Algorithm 5 MILP model that exploits zero constant bit

```
procedure TriviumCore($\mathcal{M}, \boldsymbol{x}, i_1, i_2, i_3, i_4, i_5$)
    $\mathcal{M}.var \leftarrow y_{i_1}, y_{i_2}, y_{i_3}, y_{i_4}, y_{i_5}, z_1, z_2, z_3, z_4, a$ as binary
    $\mathcal{M}.con \leftarrow y_{i_j} = x_{i_j} - z_j$ for all $j \in \{1,2,3,4\}$
    $\mathcal{M}.con \leftarrow a \geq z_3$
    $\mathcal{M}.con \leftarrow a \geq z_4$
    if we know either $s_{i_3} = 0$ or $s_{i_4} = 0$ then
        $\mathcal{M}.con \leftarrow a = 0$
    end if
    $\mathcal{M}.con \leftarrow y_{i_5} = x_{i_5} + a + z_1 + z_2$
    for all $i \in \{1, 2, \dots, 288\}$ 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
```

The main idea is to control the propagation of the division property, and the same technique was applied to ACORN already. In the case of Trivium, we first evaluate whether s_i^r is fixed to 0 in initial several hundred rounds. Then, TriviumCore is modified to Algorithm 5, where the division property of s_{i_3} and s_{i_4} never propagates to y_{i_5} if either s_{i_3} and s_{i_4} is 0.

Application and Comparison with Liu's Method. We applied this technique to Trivium. Very recently, Liu showed a new algorithm to guarantee the upper bound of the algebraic degree on NFSR-based ciphers. He evaluated a cube shown as

$$\begin{aligned}I = \{&0,2,4,6,8,10,12,15,17,19,21,23,25,27,30,32,34,36,38,40,42,\\&45,47,49,51,53,55,57,60,62,64,66,68,70,72,75,79\},\end{aligned}$$

where non-active cube bits are filled with 0. Then Liu showed that 837-round Trivium has zero-sum cube distinguisher. We also evaluated the same cube by using our method, and it guaranteed that the first key-stream bit is balanced up to the same number of rounds.

We next evaluated cube C_I, where 80 iv bits are active. From Table 3 in [Liu17], Liu's method guarantees that the first key-stream bit of 793-round Trivium is balanced. On the other hand, our method can guarantee that the first key-stream bit of 839-round Trivium is balanced.

From their comparisons, we can conclude that our method is more accurate than the Liu's algorithm. On the other hand, since we need to ask for the help from solvers, Liu's method is more efficient than our method.

8 Conclusion

This paper revisited the cube attack proposed by Dinur and Shamir at Eurocrypt 2009. The conventional cube attack regards a target symmetric-key cryptosystem as a blackbox polynomial and analyzes the polynomial experimentally. Therefore, it is practically infeasible to evaluate the security when the size of cube exceeds the experimental size. In this paper, we proposed the cube attack on non-blackbox polynomials, and it leads the cube attack exploiting large number of cube size. Our method was developed by the division property, and as far as we know, this is the first application of the division property to stream ciphers. The trivial application brings only zero-sum integral distinguishers, and it is non-trivial to recover the secret key of stream ciphers by using the distinguisher. The novel application of the division property was proposed, where it is used to analyze the Algebraic Normal Form coefficients of polynomials. As a result, we can estimate the time complexity for the superpoly recovery. Then, the superpoly recovery immediately brings the vulnerability. We applied the new technique to Trivium, Grain128a, and ACORN, and the superpoly of 832-round Trivium, 183-round Grain128a, and 704-round ACORN are more efficiently recovered than the brute-force search. For Trivium and ACORN, we can expect that the recovered superpoly is useful for the key recovery attack, and they bring the current best key-recovery attacks. On the other hand, for Grain128a, we cannot expect that the recovered superpoly is balanced, and then the recovered bit of information may be significantly small. Therefore, the feasibility of the key recovery is speculative, but 183 rounds are at least vulnerable. We expect that our new tool becomes a new generic tool to measure the security of stream ciphers.

References

ADMS09. Jean-Philippe Aumasson, Itai Dinur, Willi Meier, and Adi Shamir. Cube testers and key recovery attacks on reduced-round MD6 and Trivium. In Orr Dunkelman, editor, *FSE*, volume 5665 of *LNCS*, pages 1–22. Springer, 2009.

ÅHJM11. Martin Ågren, Martin Hell, Thomas Johansson, and Willi Meier. Grain-128a: a new version of grain-128 with optional authentication. *IJWMC*, 5(1):48–59, 2011.

BS01. Alex Biryukov and Adi Shamir. Structural cryptanalysis of SASAS. In Birgit Pfitzmann, editor, *EUROCRYPT*, volume 2045 of *LNCS*, pages 394–405. Springer, 2001.

cae14. CAESAR: Competition for authenticated encryption: Security, applicability, and robustness, 2014.

CJF$^+$16. Tingting Cui, Keting Jia, Kai Fu, Shiyao Chen, and Meiqin Wang. New automatic search tool for impossible differentials and zero-correlation linear approximations, 2016.

CP06. Christophe De Cannière and Bart Preneel. Trivium specifications, 2006. eSTREAM portfolio, Profile 2 (HW).

DGP$^+$11. Itai Dinur, Tim Güneysu, Christof Paar, Adi Shamir, and Ralf Zimmermann. An experimentally verified attack on full Grain-128 using dedicated reconfigurable hardware. In Dong Hoon Lee and Xiaoyun Wang, editors, *ASIACRYPT*, volume 7073 of *LNCS*, pages 327–343. Springer, 2011.

DKR97. Joan Daemen, Lars R. Knudsen, and Vincent Rijmen. The block cipher Square. In Eli Biham, editor, *FSE*, volume 1267 of *LNCS*, pages 149–165. Springer, 1997.

DMP$^+$15. Itai Dinur, Pawel Morawiecki, Josef Pieprzyk, Marian Srebrny, and Michal Straus. Cube attacks and cube-attack-like cryptanalysis on the round-reduced Keccak sponge function. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT Part I*, volume 9056 of *LNCS*, pages 733–761. Springer, 2015.

DS09. Itai Dinur and Adi Shamir. Cube attacks on tweakable black box polynomials. In Antoine Joux, editor, *EUROCRYPT*, volume 5479 of *LNCS*, pages 278–299. Springer, 2009.

DS11. Itai Dinur and Adi Shamir. Breaking Grain-128 with dynamic cube attacks. In Antoine Joux, editor, *FSE*, volume 6733 of *LNCS*, pages 167–187. Springer, 2011.

est08. eSTREAM: the ECRYPT stream cipher project, 2008.

FKM08. Simon Fischer, Shahram Khazaei, and Willi Meier. Chosen IV statistical analysis for key recovery attacks on stream ciphers. In Serge Vaudenay, editor, *AFRICACRYPT*, volume 5023 of *LNCS*, pages 236–245. Springer, 2008.

FV13. Pierre-Alain Fouque and Thomas Vannet. Improving key recovery to 784 and 799 rounds of trivium using optimized cube attacks. In Shiho Moriai, editor, *FSE*, volume 8424 of *LNCS*, pages 502–517. Springer, 2013.

Inc15. Gurobi Optimization Inc. Gurobi optimizer 6.5. Official webpage, `http://www.gurobi.com/`, 2015.

ISO15. ISO/IEC. JTC1: ISO/IEC 29167-13: Information technology – automatic identification and data capture techniques – part 13: Crypto suite grain-128a security services for air interface communications, 2015.

KMN10. Simon Knellwolf, Willi Meier, and María Naya-Plasencia. Conditional differential cryptanalysis of NLFSR-based cryptosystems. In Masayuki Abe, editor, *ASIACRYPT*, volume 6477 of *LNCS*, pages 130–145. Springer, 2010.

Knu94. Lars R. Knudsen. Truncated and higher order differentials. In Bart Preneel, editor, *FSE*, volume 1008 of *LNCS*, pages 196–211. Springer, 1994.

KW02. Lars R. Knudsen and David Wagner. Integral cryptanalysis. In Joan Daemen and Vincent Rijmen, editors, *FSE*, volume 2365 of *LNCS*, pages 112–127. Springer, 2002.

Lai94. Xuejia Lai. Higher order derivatives and differential cryptanalysis. In *Communications and Cryptography*, volume 276 of *The Springer International Series in Engineering and Computer Science*, pages 227–233, 1994.

Liu17. Meicheng Liu. Degree evaluation of NFSR-based cryptosystems. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO Part III*, volume 10403 of *LNCS*, pages 227–249. Springer, 2017.

LM12. Michael Lehmann and Willi Meier. Conditional differential cryptanalysis of Grain-128a. In Josef Pieprzyk, Ahmad-Reza Sadeghi, and Mark Manulis, editors, *CANS*, volume 7712 of *LNCS*, pages 1–11. Springer, 2012.

Luc01. Stefan Lucks. The saturation attack - A bait for Twofish. In Mitsuru Matsui, editor, *FSE*, volume 2355 of *LNCS*, pages 1–15. Springer, 2001.

MS12. Piotr Mroczkowski and Janusz Szmidt. The cube attack on stream cipher Trivium and quadraticity tests. *Fundam. Inform.*, 114(3-4):309–318, 2012.

MWGP11. Nicky Mouha, Qingju Wang, Dawu Gu, and Bart Preneel. Differential and linear cryptanalysis using mixed-integer linear programming. In Chuankun Wu, Moti Yung, and Dongdai Lin, editors, *Inscrypt*, volume 7537 of *LNCS*, pages 57–76. Springer, 2011.

SBD$^+$16. Md. Iftekhar Salam, Harry Bartlett, Ed Dawson, Josef Pieprzyk, Leonie Simpson, and Kenneth Koon-Ho Wong. Investigating cube attacks on the authenticated encryption stream cipher ACORN. In Lynn Batten and Gang Li, editors, *ATIS*, volume 651 of *CCIS*, pages 15–26. Springer, 2016.

SHW$^+$14a. Siwei Sun, Lei Hu, Meiqin Wang, Peng Wang, Kexin Qiao, Xiaoshuang Ma, Danping Shi, and Ling Song. Towards finding the best characteristics of some bit-oriented block ciphers and automatic enumeration of (related-key) differential and linear characteristics with predefined properties, 2014.

SHW$^+$14b. Siwei Sun, Lei Hu, Peng Wang, Kexin Qiao, Xiaoshuang Ma, and Ling Song. Automatic security evaluation and (related-key) differential characteristic search: Application to SIMON, PRESENT, LBlock, DES(L) and other bit-oriented block ciphers. In Palash Sarkar and Tetsu Iwata, editors, *ASIACRYPT Part I*, volume 8873 of *LNCS*, pages 158–178. Springer, 2014.

ST16. Yu Sasaki and Yosuke Todo. New impossible differential search tool from design and cryptanalysis aspects, 2016. this paper is accepted in Eurocrypt 2017.

SWW16. Ling Sun, Wei Wang, and Meiqin Wang. MILP-aided bit-based division property for primitives with non-bit-permutation linear layers, 2016.

TM16. Yosuke Todo and Masakatu Morii. Bit-based division property and application to Simon family. In Thomas Peyrin, editor, *FSE*, volume 9783 of *LNCS*, pages 357–377. Springer, 2016.

Tod15a. Yosuke Todo. Integral cryptanalysis on full MISTY1. In Rosario Gennaro and Matthew Robshaw, editors, *CRYPTO Part I*, volume 9215 of *LNCS*, pages 413–432. Springer, 2015.

Tod15b. Yosuke Todo. Structural evaluation by generalized integral property. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT Part I*, volume 9056 of *LNCS*, pages 287–314. Springer, 2015.

Wu16. Hongjun Wu. Acorn v3, 2016. Submission to CAESAR competition.

XZBL16. Zejun Xiang, Wentao Zhang, Zhenzhen Bao, and Dongdai Lin. Applying MILP method to searching integral distinguishers based on division property for 6 lightweight block ciphers. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *ASIACRYPT Part I*, volume 10031 of *LNCS*, pages 648–678. Springer, 2016.

A Components of MILP model for Division Property in Grain128a

This appendix shows components of MILP model for division property in Grain128a.

Algorithm 6 MILP model for XOR and AND in Grain128a

```
procedure AND($\mathcal{M}, \boldsymbol{b}, \boldsymbol{s}, I, J$)
    $\mathcal{M}.var \leftarrow b'_i, x_i$ for all $i \in I$ as binary
    $\mathcal{M}.var \leftarrow s'_j, y_j$ for all $j \in J$ as binary
    $\mathcal{M}.var \leftarrow z$ as binary
    $\mathcal{M}.con \leftarrow b'_i = b_i - x_i$ for all $i \in I$
    $\mathcal{M}.con \leftarrow s'_j = s_j - y_j$ for all $j \in J$
    $\mathcal{M}.con \leftarrow z \geq x_i$ for all $i \in I$
    $\mathcal{M}.con \leftarrow z \geq y_j$ 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
    return $(\mathcal{M}, \boldsymbol{b}', \boldsymbol{s}', z)$
end procedure
procedure XOR($\mathcal{M}, \boldsymbol{b}, \boldsymbol{s}, I, J$)
    $\mathcal{M}.var \leftarrow b'_i, x_i$ for all $i \in I$ as binary
    $\mathcal{M}.var \leftarrow s'_j, y_j$ for all $j \in J$ as binary
    $\mathcal{M}.var \leftarrow z$ as binary
    $\mathcal{M}.con \leftarrow b'_i = b_i - x_i$ for all $i \in I$
    $\mathcal{M}.con \leftarrow s'_j = s_j - y_j$ for all $j \in J$
    $\mathcal{M}.con \leftarrow z = \sum_{i \in I} x_i + \sum_{j \in J} y_j$
    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
    return $(\mathcal{M}, \boldsymbol{b}', \boldsymbol{s}', z)$
end procedure
```

Algorithm 7 MILP model for NLFSR and LFSR in Grain128a

```
procedure funcZ($\mathcal{M}, \boldsymbol{b}, \boldsymbol{s}$)
    $(\mathcal{M}, \boldsymbol{b}_1, \boldsymbol{s}_1, a_1) = \texttt{AND}(\mathcal{M}, \boldsymbol{b}, \boldsymbol{s}, \{12\}, \{8\})$
    $(\mathcal{M}, \boldsymbol{b}_2, \boldsymbol{s}_2, a_2) = \texttt{AND}(\mathcal{M}, \boldsymbol{b}_1, \boldsymbol{s}_1, \phi, \{13, 20\})$
    $(\mathcal{M}, \boldsymbol{b}_3, \boldsymbol{s}_3, a_3) = \texttt{AND}(\mathcal{M}, \boldsymbol{b}_2, \boldsymbol{s}_2, \{95\}, \{42\})$
    $(\mathcal{M}, \boldsymbol{b}_4, \boldsymbol{s}_4, a_4) = \texttt{AND}(\mathcal{M}, \boldsymbol{b}_3, \boldsymbol{s}_3, \phi, \{60, 79\})$
    $(\mathcal{M}, \boldsymbol{b}_5, \boldsymbol{s}_5, a_5) = \texttt{AND}(\mathcal{M}, \boldsymbol{b}_4, \boldsymbol{s}_4, \{12, 95\}, \{94\})$
    $(\mathcal{M}, \boldsymbol{b}_6, \boldsymbol{s}_6, x) = \texttt{XOR}(\mathcal{M}, \boldsymbol{b}_5, \boldsymbol{s}_5, \{2, 15, 36, 45, 64, 73, 89\}, \{93\})$
    $\mathcal{M}.var \leftarrow z$ as binary
    $\mathcal{M}.con \leftarrow z = x + \sum_{i=1}^{5} a_i$
    return $(\mathcal{M}, \boldsymbol{b}_6, \boldsymbol{s}_6, z)$
end procedure
procedure funcF($\mathcal{M}, \boldsymbol{s}$)
    $(\mathcal{M}, \phi, \boldsymbol{s}_1, f) = \texttt{XOR}(\mathcal{M}, \phi, \boldsymbol{s}, \phi, \{0, 7, 38, 70, 81, 96\})$
    return $(\mathcal{M}, \boldsymbol{s}_1, f)$
end procedure
procedure funcG($\mathcal{M}, \boldsymbol{b}$)
    $(\mathcal{M}, \boldsymbol{b}_1, \phi, a_1) = \texttt{AND}(\mathcal{M}, \boldsymbol{b}, \phi, \{3, 67\}, \phi)$
    $(\mathcal{M}, \boldsymbol{b}_2, \phi, a_2) = \texttt{AND}(\mathcal{M}, \boldsymbol{b}_1, \phi, \{11, 13\}, \phi)$
    $(\mathcal{M}, \boldsymbol{b}_3, \phi, a_3) = \texttt{AND}(\mathcal{M}, \boldsymbol{b}_2, \phi, \{17, 18\}, \phi)$
    $(\mathcal{M}, \boldsymbol{b}_4, \phi, a_4) = \texttt{AND}(\mathcal{M}, \boldsymbol{b}_3, \phi, \{27, 59\}, \phi)$
    $(\mathcal{M}, \boldsymbol{b}_5, \phi, a_5) = \texttt{AND}(\mathcal{M}, \boldsymbol{b}_4, \phi, \{40, 48\}, \phi)$
    $(\mathcal{M}, \boldsymbol{b}_6, \phi, a_6) = \texttt{AND}(\mathcal{M}, \boldsymbol{b}_5, \phi, \{61, 65\}, \phi)$
    $(\mathcal{M}, \boldsymbol{b}_7, \phi, a_7) = \texttt{AND}(\mathcal{M}, \boldsymbol{b}_6, \phi, \{68, 84\}, \phi)$
    $(\mathcal{M}, \boldsymbol{b}_8, \phi, a_8) = \texttt{AND}(\mathcal{M}, \boldsymbol{b}_7, \phi, \{88, 92, 93, 95\}, \phi)$
    $(\mathcal{M}, \boldsymbol{b}_9, \phi, a_9) = \texttt{AND}(\mathcal{M}, \boldsymbol{b}_8, \phi, \{22, 24, 25\}, \phi)$
    $(\mathcal{M}, \boldsymbol{b}_{10}, \phi, a_{10}) = \texttt{AND}(\mathcal{M}, \boldsymbol{b}_9, \phi, \{70, 78, 82\}, \phi)$
    $(\mathcal{M}, \boldsymbol{b}_{11}, \phi, x) = \texttt{XOR}(\mathcal{M}, \boldsymbol{b}_{10}, \phi, \{0, 26, 56, 91, 96\}, \phi)$
    $\mathcal{M}.var \leftarrow g$ as binary
    $\mathcal{M}.con \leftarrow g = x + \sum_{i=1}^{10} a_i$
    return $(\mathcal{M}, \boldsymbol{b}_{11}, g)$
end procedure
```