Improved Division Property Based Cube Attacks Exploiting Low Degree Property of Superpoly

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

[1] DTU Compute, Technical University of Denmark, Lyngby, Denmark
[2] State Key Laboratory of Cryptology, P.O. Box 5159, Beijing 100878, China
[3] NTT Secure Platform Laboratories, Tokyo 180-8585, Japan
[4] imec-COSIC, Dept. Electrical Engineering (ESAT), KU Leuven, Leuven, Belgium
[5] University of Hyogo, Hyogo 650-0047, Japan
[6] FHNW, Windisch, Switzerland
quwg@dtu.dk,haoyonglin@yeah.net,todo.yosuke@lab.ntt.co.jp
chaoyun.li@esat.kuleuven.be,takanori.isobe1@gmail.com,willi.meier@fhnw.ch

Abstract. In this paper we investigate the sparse structure of the superpoly in cube attack, and further reduce the complexity of superpoly recovery.
We apply our technique to stream cipher Trivium and Kreyvium. For Trivium, benefited from our techniques, we, for the first time, can recover the superpoly of 833-rounds with cube dimension 73, and complexity $2^{76.91}$. Furthermore, for 833-rounds, we can find a new cube of dimension 74, with only one secret key bit involved, thus the complexity is 2^{75}. For 839-rounds, we find a cube of dimension 78, with only one secret key bit involved in the superpoly.
For Kreyvium, the lower complexity evaluation enables us to recover the superpoly of 849-rounds with time complexity of $2^{81.7}$. Moreover, we find a new cube of dimension 102, which can achieve 888-rounds with complexity $2^{111.38}$. So far as we know, all of our results are the best.
Key words: Cube attack, Division property, MILP, Trivium, Kreyvium

1 Introduction

Cube attack is one of the general cryptanalytic techniques against symmetric-key cryptosystems proposed by Dinur and Shamir [DS09]. Especially, cube attack has been successfully applied to various ciphers, including stream ciphers [ADMS09,DS11,FV13,SBD+16], hash functions [DMP+15,HWX+17] [LBDW17], and authenticated encryptions [LDW17,DLWQ17]. The conventional cube attack assumes that the output bit (variable) of a cipher is given as a blackbox polynomial $f(\boldsymbol{x}, \boldsymbol{v})$ of the input secret variables $\boldsymbol{x}$ and public variables $\boldsymbol{v}$. The main observation used in the attack is that when this polynomial has a (low) algebraic degree d in terms of public variables, then summing its outputs over 2^{d-1} inputs, in which a subset of public variables (*i.e.*, a *cube*) of dimension $(d-1)$ ranges over all possible values, and the other public variables are fixed to some constant, yields a linear function of secret variables (*i.e. superpoly*, denoted as $p(\boldsymbol{x}, \boldsymbol{v})$). By querying the oracle encryption, one can build a system of linear equations of the secret variables. After solving this system, the secret variables can be recovered.

In the original paper on cube attacks [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 key-recovery attack against Trivium [FV13]. Note that these are experimental cryptanalysis, and it is possible that cube attacks do not actually work. For example, if the superpoly is a highly unbalanced function for specific variables, we cannot ignore the probability that the linearity and quadraticity tests fail.

Higher-order differential has been first proposed in cryptography by Lai [Lai94] and was later applied to differential cryptanalysis of block ciphers by Knudsen [Knu94]. When a block cipher is analyzed, attackers first evaluated the algebraic degree of the reduced round (say r-round) and construct a r-round higher-order differential where the $(d+1)$st order difference is always 0 if the degree is at most d. Then, the key recovery is dependently appended after the r-round higher order differential characteristic. Namely, attackers guess round keys used in last several rounds and compute backward the $(d+1)$st order difference of the rth round. If the $(d+1)$st order difference is always 0, then the round key guessed is correct, otherwise, the guessed round keys is wrong. 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].

Cube attack can be seen as a type of higher-order differential attacks because it also evaluates the algebraic degree of the cipher. However, the major difference between cube attack and higher-order differential attack is whether or not secret variables are directly recovered from the characteristic. Note that we cannot use the strategy of key recovery attack based on higher-order differentials against many stream ciphers because for a stream cipher 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 key stream, it cannot be directly utilized for key recovery attacks by appending key recovery rounds in the key generation phase. To execute the key-recovery attack of stream ciphers, we have to recover the secret key by only using key streams that attackers can observe. Therefore, more advanced and complicated analyses are required than the simple degree estimation of 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 usually been experimentally analyzed in the cube attack.

As a generalized integral property, *division property* was proposed by Todo [Tod15b]to search more accurate integral distinguishers for symmetric-key primitives including SPNs and Feistel structures, because the algebraic degree of the S-box was ingeniously exploited in the propagation of the division property. One prominent application was the attack on MISTY1 [Tod15a], where the S-Box S_7 was shown to have an important vulnerability in terms of division property. By employing this a new 6-round integral distinguisher was constructed, and a full-round attack on MISTY1 was achieved for the first time. Motivated by narrowing the 5 rounds gap between the integral distinguishers for SIMON32 in [WLV+14] and [Tod15b], bit-based division property [TM16] was introduced at FSE 2016, where for the first time the bit-level division property was considered. As a result, the same integral distinguishers for SIMON32 in [WLV+14] were found. However, as pointed out in [TM16], for a block cipher with block size n, the time and memory complexity is lower bounded by 2^n. As most ciphers adopt block size larger than 32, this makes searching integral distinguisher by bit-based division property under this framework computationally infeasible. In order to solve this problem, Xiang *et al.* [XZBL16] built an automatic tool based on mixed integer linear programming (MILP) to study the division property of SPNs with bit-permutation linear layers (*e.g.* PRESENT). They first introduced notion *division trail* to track the division propagation and built the objective function, then represented the operations (`and`, `xor` and `copy`) of the ciphers by linear (in)equalities to constrain the objective function. After setting the required stopping rules of searching division trails, they can determine the existence of certain number of rounds integral distinguishers by optimizing the MILP. As a result, they managed to find many interesting integral distinguishers for the targeted ciphers.

For the first time, Todo *et al.* [TIHM17] applied division property to the cube attack on stream ciphers Trivium, Grain128a and authenticated encryption ACORN, and achieved the best key recovery attacks against all of them. After choose a cube C_I with the indices of the public variables as a set I, by evaluating the division property of the cipher, they manage to determine the secret variables involved in the ANF of the superpoly $p(\boldsymbol{x}, \boldsymbol{v})$, denote by J as the set of indices. 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 were evaluated by guessing $|J|$-bit secret variables, and the time complexity to recover the ANF of the superpoly is $2^{|I|+|J|}$ encryptions. Then they query the encryption oracle and get the sum over the cube. Finally, one polynomial about secret variables is deduced, from which the secret variables can be recovered. The most important step in the cube attack is the superpoly recovery and is the main focus of this paper. The time complexity for the superpoly recovery is upper bounded by $2^{|I|+|J|}$ in [TIHM17].

Our contribution. We further investigate the sparse structure of the superpoly of given cubes. By further extending the division property application in [TIHM17], we are able to find all the quadratic terms of the superpoly, and denote the indices set as J_2. More general, assume the algebraic degree of the superpoly is d, and denote J_t $(1 \leq t \leq d)$ as the indices set of the terms of degree t, then we can provide a better bound of the complexity of the superpoly recovery as $2^{|I|} \times (1 + \sum_{t=1}^{d} |J_t|)$ where $|J_1| = |J|$. Note that our new bound is much lower than the one in [TIHM17], especially when $|I|$ and d are large. We will give more details in Sect. 4.

Moreover, we represent our superpoly as a graph, where vertices correspond to the involved secret key bits in the superpoly (*i.e.* indices in J_1), and edges between any pair of the vertices are the quadratic terms involved in the superpoly (*i.e.* indices in J_2). Therefore, we deduce the computation of J_t to the problem of finding the maximal t-clique problem. And from a very simple algorithm we can easily give an upper bound of the size of J_t. By this approach, we can reduce the time complexity of the superpoly recovery in [TIHM17].

As applications, we apply our approach to stream cipher Trivium and Kreyvium. In [TIHM17], 832 initialization rounds of Trivium can be analyzed with a cube of dimension 72, and the number of secret variables involved in the superpoly is 5, so the complexity of recovering the superpoly is bounded by 2^{77}. Our new cube attack can reduce this to $2^{75.58}$, which is not great since the size of J_1 equals to 5 and the degree of the superpoly is only 3, with these small values our bound has less advantage. We find a new cube of dimension 73 for 833 rounds, where $|J_1| = 7$ and $d = 3$. If we evaluate the complexity as in [TIHM17], then it reaches $2^{73+7} = 2^{80}$ which makes the attack fail. However, when applying our new bound, the complexity can be reduced to $2^{76.91}$. In this way, we can achieve one more round than [TIHM17]. Also, we find a new cube of dimension 78 for 839 rounds, and only one secret variable is involved.

The design of Kreyvium [CCF+16] is recently motivated by the application for homomorphic friendly encryption. Kreyvium shares the same internal structure as Trivium, but allows for larger keys of 128 bits than 80 bits of Trivium. With a cube of size 61 [Liu17], we can recover 849 initialization rounds of Kreyvium with time complexity $2^{81.7}$ while the bound should be 2^{124} if we calculate it as in [TIHM17]. Furthermore, for 888 rounds with a cube of dimension 102, we can recover the superpoly with time complexity $2^{113.38}$. The summarized results of Trivium and Kreyvium are in Table 1.

The remainder of the paper is organized as: Sect. 2 introduces the preliminaries, and Sect. 3 describes the MILP models of Trivium and Kreyvium. In Sect. 4, we discusses our new bound of the time complexity, and applied it to analyze Trivium and Kreyvium in Sect. 5. Plans about the future work and conclusions are given in Sect. 6.

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, MILP has been applied to search for differential [SHW+14b,SHW+14a], linear [SHW+14a], impossible differential [CJF+16,ST17], 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.

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

Applications	# Rounds	Cube size	$\|J\|$	Complexity	Reference
Trivium	799	32 †	-	practical	[FV13]
	832	72	5	2^{77}	[TIHM17]
	832	**72**	**5**	$\mathbf{2^{75.58}}$	**Sect. 5.1**
	833	**73**	**7**	$\mathbf{2^{76.91}}$	**Sect. 5.1**
	833	**74**	**1**	$\mathbf{2^{75}}$	**Sect. 5.1**
	839	**78**	**1**	$\mathbf{2^{79}}$	**Sect. 5.1**
Kreyvium	**849**	**61**	**23**	$\mathbf{2^{81.7}}$	**Sect. 5.2**
	870	**85**	**2**	$\mathbf{2^{86.58}}$	**Sect. 5.2**
	872	**85**	**39**	$\mathbf{2^{94.61}}$	**Sect. 5.2**
	888	**102**	**36**	$\mathbf{2^{111.38}}$	**Sect. 5.2**

† 18 cubes whose size is from 32 to 37 are used, where the most efficient cube is shown to recover one bit of the secret key.

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 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[7].

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

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

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

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

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

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

where $\mathcal{D}_{\mathbb{K}_i}$ is the division property after i-round propagation. If the division property $\mathbb{K}_r$ does not have an unit vector $\boldsymbol{e}_i$ whose only ith element is 1, the ith bit of r-round ciphertexts is balanced.

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 $\{\boldsymbol{k}\} \stackrel{\text{def}}{=} \mathbb{K}_0 \to \mathbb{K}_1 \to \mathbb{K}_2 \to \cdots \to \mathbb{K}_r$. Moreover, for any vector $\boldsymbol{k}_{i+1}^* \in \mathbb{K}_{i+1}$, there must exist a vector $\boldsymbol{k}_i^* \in \mathbb{K}_i$ such that $\boldsymbol{k}_i^*$ can propagate to $\boldsymbol{k}_{i+1}^*$ by the propagation rule of the division property. Furthermore, for $(\boldsymbol{k}_0, \boldsymbol{k}_1, \ldots, \boldsymbol{k}_r) \in (\mathbb{K}_0 \times \mathbb{K}_1 \times \cdots \times \mathbb{K}_r)$ if $\boldsymbol{k}_i$ can propagate to $\boldsymbol{k}_{i+1}$ for all $i \in \{0, 1, \ldots, r-1\}$, we call $(\boldsymbol{k}_0 \to \boldsymbol{k}_1 \to \cdots \to \boldsymbol{k}_r)$ an r-round division trail.*

Let E_k be the target r-round iterated cipher. Then, if there 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.

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.

Thanks to all these efforts, the propagation of division property has now become generating a MILP model $\mathcal{M}$ whose variables and constraints are generated according to the round functions. By imposing constraints to the

2.4 The Bit-Based Division Property for Cube Attack

Recently at CRYPTO 2017, Todo *et al.* successfully apply the bit-based division property to the cube attack [TIHM17]. They prove the following Lemma 1 and Proposition 4.

Lemma 1. *Let* $f(\boldsymbol{x})$ *be a polynomial from* $\mathbb{F}_2^n$ *to* $\mathbb{F}_2$ *and* $a_{\boldsymbol{u}}^f \in \mathbb{F}_2$ $(\boldsymbol{u} \in \mathbb{F}_2^n)$ *be the ANF coefficients. 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* $\boldsymbol{u} \succeq \boldsymbol{k}$.

Proposition 4. *Let* $f(\boldsymbol{x}, \boldsymbol{v})$ *be a polynomial, where* $\boldsymbol{x}$ *and* $\boldsymbol{v}$ *denote the secret and public variables, respectively. For a set of indices* $I = \{i_1, i_2, \ldots, i_{|I|}\} \subset \{1, 2, \ldots, m\}$*, let* C_I *be a set of* $2^{|I|}$ *values where the variables in* $\{v_{i_1}, v_{i_2}, \ldots, v_{i_{|I|}}\}$ *are taking all possible combinations of values. Let* $\boldsymbol{k}_I$ *be an* m*-dimensional bit vector such that* $\boldsymbol{v}^{\boldsymbol{k}_I} = t_I = v_{i_1} v_{i_2} \cdots v_{i_{|I|}}$*,* i.e. $k_i = 1$ *if* $i \in I$ *and* $k_i = 0$ *otherwise. Assuming there is no division trail such that* $(\boldsymbol{e}_\lambda, \boldsymbol{k}_I) \xrightarrow{f} 1$*,* x_λ *is not involved in the superpoly of the cube* C_I.

Based on these theoretic findings, they can identify the key bits that might be involved in the superpoly of any cube C_I using MILP-aided bit-based division property. The method can be summarized as Algorithm 1 that outputs set of indices J satisfying: for $j \in J$, there exist assignment $\boldsymbol{v}$ s.t. the secret key bit x_j is involved in $p_{\boldsymbol{v}}(\boldsymbol{x})$; for $j \notin J$, the secret key bit x_j can never be involved in $p_{\boldsymbol{v}}(\boldsymbol{x})$ for any $\boldsymbol{v}$ assignment.

Algorithm 1 Evaluate secret variables by MILP

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

2.5 Key Recovery Attack Procedure

With J and I decided, the key recovery process can be summarized as follows:

1. **Offline Phase: Identify the superpoly.** Attackers assign the non-cube IVs a proper constant value, and prepare a cube by flipping bits in I. Then, for all possible values of the secret variables $\{x_{j_1}, x_{j_2}, \ldots, x_{j_{|J|}}\}$, they compute and store the value of the superpolys as $p_{\boldsymbol{v}}(\boldsymbol{x}) = \bigoplus_{C_I} f(\boldsymbol{x}, \boldsymbol{v})$. The $2^{|J|}$ values compose the truth table of $p_{\boldsymbol{v}}(\boldsymbol{x})$ and the ANF of the superpoly is determined accordingly. Finally, they search for the preferable superpoly (its truth table is balanced) by changing the value of the non-cube IV.
2. **Online Phase: Recover the part of secret variables.** After the truth table of the balanced superpoly and the proper assignments of non-cube IVs are given, attackers query the cube C_I to encryption oracle and get one bit $p_{\boldsymbol{v}}(\boldsymbol{x})$ through summation. 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.

Phase 1 takes $2^{|I|+|J|}$ encryptions to construct a truth table of size $2^{|J|}$. Phase 2 requires $2^{|I|}$ encryptions to sum over the cube C_I. Additional $2^{|J|}$ table lookups are also necessary in order to identify the $2^{|J|-1}$ candidate keys but such a complexity is negligible in comparison with the cube summation so the complexity of Phase 2 can be regarded as $2^{|I|}$. The complexity of Phase 3 is 2^{n-1}. The attack can be meaningful only if $|I| + |J| < n$.

If we want to attack more rounds, we have to use higher dimensional cubes which increase the size of I. On the other hand, due to the linear diffusion, more active IVs makes $|J|$ increase accordingly. Therefore, we propose several techniques that enable us to acquire the superpoly even if $|I|+|J| \geq n$.

Previous MILP model can only tell that for some assignment to the non-cube IVs, the superpoly $p_{\boldsymbol{v}}(\boldsymbol{x})$ involves the key bits J. In other words, there exist probability that for some $\boldsymbol{v}$, $p_{\boldsymbol{v}}(\boldsymbol{x})$ is constant whose value is irrelevant with any secret key bits and Phase 1 might be repeated the $2^{|I|+|J|}$ encryptions several times before identifying a preferable assignment to the non-cube IVs. Therefore, as is stated in [TIHM17], the availability of the attack is based on the following two assumptions.

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.*

We also propose a method so that different assignment to the non-cube IVs can have corresponding effect on the MILP model. In this way, the preciseness of the MILP is improved. We are able to identify the involved keys for specifically assigned non-cube IVs and improve the complexities of previous attacks as well.

3 Descriptions of Trivium and Kreyviumin MILP Model

In this section, we give a very brief description of stream cipher Trivium and Kreyvium followed by its MILP model.

3.1 Specification of Trivium

Trivium [CP06] is one of the seven stream ciphers recommended by the eSTREAM [est08] project after a 5-year international competition. Triviumis an NLFSR-based stream cipher, and the internal state is represented by 288-bit state $(s_1, s_2, \ldots, s_{288})$. The algorithm consists of two phases: *Initialization* and *key stream generation*.

In the *initialization* phase, the algorithm is initialized by loading an 80-bit key and an 80-bit IV into the 288-bit initial state, and setting all remaining bits to 0, except for s_{286}, s_{287}, and s_{288}. The *key stream generation* consists of an iterative process which extracts the values of 15 specific state bits and uses them both to update 3 bits of the state and to compute 1 bit of key stream z_i. The state bits are then rotated and the process repeats itself until the requested $N \leq 2^{64}$ bits of key stream have been generated. The update function of Trivium is depicted in Fig. 1.

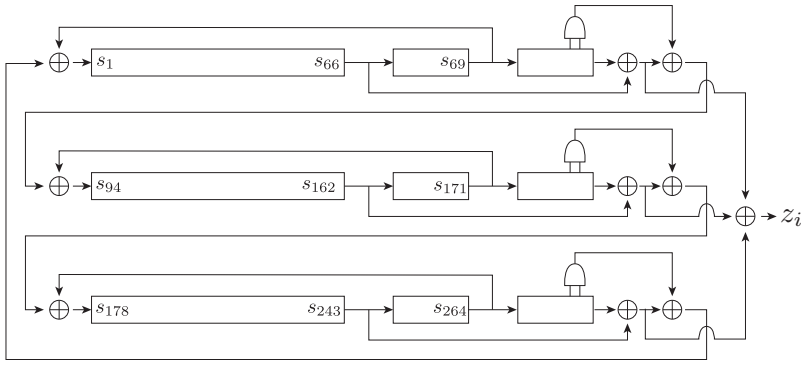

Fig. 1. Structure of Trivium

A complete description of Trivium is given by the following simple pseudo-code

$(s_1, s_2, \ldots, s_{93}) \leftarrow (K_1, \ldots, K_{80}, 0, \ldots, 0)$
$(s_{94}, s_{95}, \ldots, s_{177}) \leftarrow (IV_1, \ldots, IV_{80}, 0, \ldots, 0)$
$(s_{178}, s_{179}, \ldots, s_{288}) \leftarrow (0, \ldots, 0, 1, 1, 1)$

for $i = 1$ **to** $1152 + N$ **do**

$t_1 \leftarrow s_{66} + s_{93}$
$t_2 \leftarrow s_{162} + s_{177}$
$t_3 \leftarrow s_{243} + s_{288}$
if $i > 1152$ **then**
output $z_{i-1152} \leftarrow t_1 + t_2 + t_3$
end if
$t_1 \leftarrow t_1 + s_{91} \cdot s_{92} + s_{171}$
$t_2 \leftarrow t_2 + s_{175} \cdot s_{176} + s_{264}$
$t_3 \leftarrow t_3 + s_{286} \cdot s_{287} + s_{69}$
$(s_1, s_2, \ldots, s_{93}) \leftarrow (t_3, s_1, \ldots, s_{92})$
$(s_{94}, s_{95}, \ldots, s_{177}) \leftarrow (t_1, s_{94}, \ldots, s_{176})$
$(s_{178}, s_{179}, \ldots, s_{288}) \leftarrow (t_2, s_{178}, \ldots, s_{287})$
end for

3.2 MILP Model of Trivium Initialization

`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 TRIVIUMwhose initialization rounds are reduced to `R.TriviumCore` in Algorithm 2 generates MILP variables and constraints for each update function of register.

Algorithm 2 MILP model of division property for TRIVIUM

procedure `TriviumCore`$(\mathcal{M}, x, i_1, i_2, i_3, i_4, i_5)$
  $\mathcal{M}.var \leftarrow y_{i_1}, y_{i_2}, y_{i_3}, y_{i_4}, y_{i_5}, z_1, z_2, z_3, z_4, a$ 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 $(\mathcal{M}, y)$
end procedure
procedure `TriviumEval`(round R)
  Prepare empty MILP Model $\mathcal{M}$
  $\mathcal{M}.var \leftarrow s_i^0$ for $i \in \{1, 2, \ldots, 128\}$.
  for $r = 1$ to R **do**
    $(\mathcal{M}, \boldsymbol{x}) \leftarrow$ `TriviumCore`$(\mathcal{M}, \boldsymbol{s}^{r-1}, 66, 171, 91, 92, 93)$
    $(\mathcal{M}, \boldsymbol{y}) \leftarrow$ `TriviumCore`$(\mathcal{M}, \boldsymbol{x}, 162, 264, 175, 176, 177)$
    $(\mathcal{M}, \boldsymbol{z}) \leftarrow$ `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
  $\mathcal{M}.con \leftarrow (s_{66}^r + s_{93}^r + s_{162}^r + s_{177}^r + s_{243}^r + s_{288}^r + k_{R+1}^*) = 1$
  return $\mathcal{M}$
end procedure

We give the detailed MILP description of one of the registers update function `TriviumCore` as an example in Fig. 2. Here for simplicity we still denote s_i^r as the division property for the state bit s_i^r

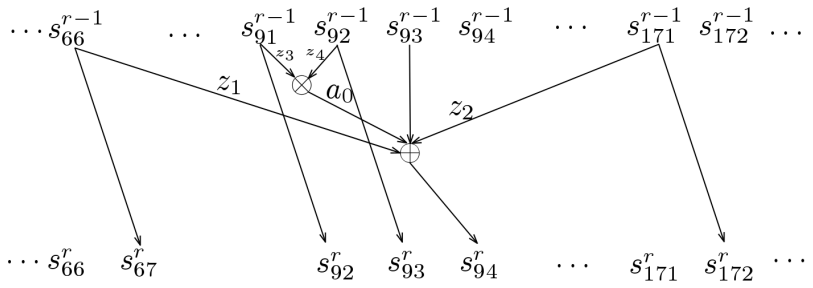

Fig. 2. An Example of `TriviumCore`

in round r, and introduce new binary variable z_i^{r-1} $(i = 1, 2, 3, 4)$ for `copy` operation in round i, and a_i $(i = 1, 2, 3)$ for the `and` operations of three registers in round r. All detailed MILP descriptions of the three registers of Trivium are listed in Table 2,

Table 2. MILP Representations of Trivium Operations

`copy`	`and`	`xor`
$s_{66}^{r-1} = s_{67}^{r} + z_1$ $s_{171}^{r-1} = s_{172}^{r} + z_2$ $s_{91}^{r-1} = s_{92}^{r} + z_3$ $s_{92}^{r-1} = s_{93}^{r} + z_4$	$a_1 \geq z_3$ $a_1 \geq z_4$	$s_{94}^{r} = s_{93}^{r-1} + a_1 + z_1 + z_2$
$s_{162}^{r-1} = s_{163}^{r} + z_5$ $s_{264}^{r-1} = s_{265}^{r} + z_6$ $s_{175}^{r-1} = s_{176}^{r} + z_7$ $s_{176}^{r-1} = s_{177}^{r} + z_8$	$a_1 \geq z_7$ $a_2 \geq z_8$	$s_{178}^{r} = s_{177}^{r-1} + a_2 + z_5 + z_6$
$s_{243}^{r-1} = s_{244}^{r} + z_9$ $s_{69}^{r-1} = s_{70}^{r} + z_{10}$ $s_{286}^{r-1} = s_{287}^{r} + z_{11}$ $s_{287}^{r-1} = s_{288}^{r} + z_{12}$	$a_3 \geq z_{11}$ $a_3 \geq z_{12}$	$s_{1}^{r} = s_{288}^{r-1} + a_3 + z_9 + z_{10}$
$s_i^r = s_{i-1}^{r-1}$ $i \neq$ 1,67,70,92,93,94,163,172,176,177,178,244,265,287,288		

3.3 Specification of Kreyvium

The design of Kreyvium is motivated by the application in homomorphic friendly encryption. The main advantage of Kreyvium is that it provides 128-bit security (instead of 80-bit) but with the same multiplicative depth, which is essential to the homomorphic encryption.

We give a very brief description of Kreyvium and take the same notations as Trivium. Kreyvium consists of 5 registers. Same with the original Trivium, three of them are NFSRs and are concatenated to make up a 288-bit state. The remaining two are LFSRs denoted as K^* and IV^* respectively. This part of the state aims at making both the filtering and transition functions key- and IV-dependent. As the same with Trivium in the key initialization, the state is updated $4*288 = 1152$ times without producing an output. After that, one bit key stream is produced by every update function.

The update function KREYVIUM is described in pseudo-code below, and depicted in Fig. 3.

$(s_1, s_2, \ldots, s_{93}) \leftarrow (K_1, K_2 \ldots, K_{93})$
$(s_{94}, s_{95}, \ldots, s_{177}) \leftarrow (IV_1, IV_2, \ldots, IV_{84})$
$(s_{178}, s_{179}, \ldots, s_{288}) \leftarrow (IV_{85}, \ldots, IV_{128}, 1, \ldots 1, 0)$
$(K^*_{128}, K^*_{127}, \ldots K^*_1) \leftarrow (K_1, K_2 \ldots, K_{128})$
$(IV^*_{128}, IV^*_{127}, \ldots IV^*_1) \leftarrow (IV_1, IV_2 \ldots, IV_{128})$
for $i = 1$ **to** $1152 + N$ **do**
  $t_1 \leftarrow s_{66} + s_{93}$
  $t_2 \leftarrow s_{162} + s_{177}$
  $t_3 \leftarrow s_{243} + s_{288} + K^*_1$
  if $i > 1152$ **then**
    output $z_{i-1152} \leftarrow t_1 + t_2 + t_3$
  end if
  $t_1 \leftarrow t_1 + s_{91} \cdot s_{92} + s_{171} + IV^*_0$
  $t_2 \leftarrow t_2 + s_{175} \cdot s_{176} + s_{264}$
  $t_3 \leftarrow t_3 + s_{286} \cdot s_{287} + s_{69}$
  $t_4 \leftarrow K^*_1$
  $t_5 \leftarrow IV^*_1$
  $(s_1, s_2, \ldots, s_{93}) \leftarrow (t_3, s_1, \ldots, s_{92})$
  $(s_{94}, s_{95}, \ldots, s_{177}) \leftarrow (t_1, s_{94}, \ldots, s_{176})$
  $(s_{178}, s_{179}, \ldots, s_{288}) \leftarrow (t_2, s_{178}, \ldots, s_{287})$
  $(K^*_{128}, K^*_{127}, \ldots K^*_0) \leftarrow (t_4, K^*_{128}, \ldots K^*_2)$
  $(IV^*_{128}, IV^*_{127}, \ldots IV^*_1) \leftarrow (t_5, \ldots, IV^*_{127}, \ldots IV^*_2)$
end for

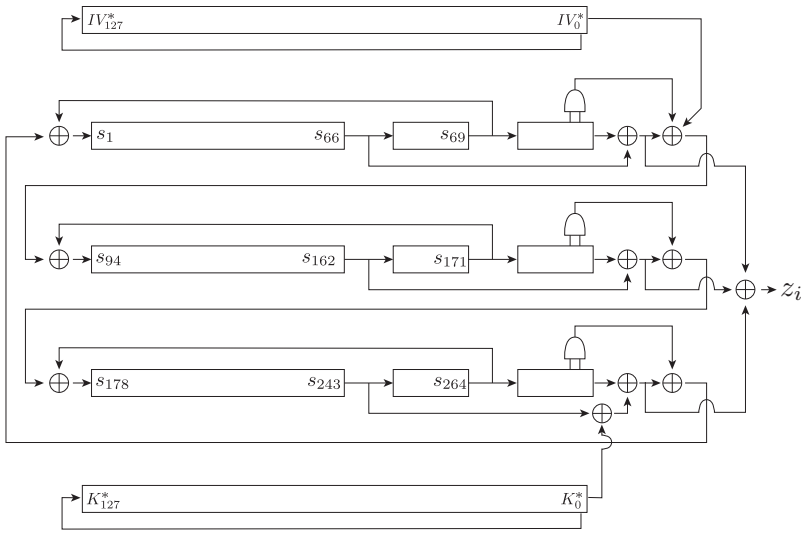

Fig. 3. Structure of KREYVIUM

3.4 MILP Description of Kreyvium Initialization

KREYVIUM shares the same core function with TRIVIUM and the division property propagation for the core function is denoted as `TriviumCore` and is defined in Algorithm 2. Besides `TriviumCore`, the LFSRs $\boldsymbol{K}^*$ and $\boldsymbol{IV}^*$ are modeled as `LFSR` of Algorithm 3. On input the current MILP model $\mathcal{M}$ and a vector $\boldsymbol{x}$ of 128 binary MILP variables, it output the updated model $\mathcal{M}$, a new 128-variable vector $\boldsymbol{y}$ describing the division property after one round of update, and an additional variable o describing the output of LFSR.

Algorithm 3 MILP model of division property for the $\boldsymbol{K}^*$ and $\boldsymbol{IV}^*$

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

With the definition of `TriviumCore` and `LFSR`, the MILP model of R-round KREYVIUM can be described as Algorithm 4.

This algorithm generates MILP model $\mathcal{M}$ as the input of Algorithm 1, and the model $\mathcal{M}$ can evaluate all division trails for KREYVIUM whose initialization rounds are reduced to R.

4 Lower the Complexity of Superpoly Recovery

In this section, we show how to further lower the complexity of recovery the superpoly in [TIHM17].

First, extensions to Propositions 4 are given, based on which we introduce a method to further reduce the complexity of recovering the superpoly. We follow the same notations with [TIHM17].

Proposition 5. *Let $f(\boldsymbol{x}, \boldsymbol{v})$ be a polynomial, where $\boldsymbol{x} \in \mathbb{F}_2^n$ and $\boldsymbol{v} \in \mathbb{F}_2^m$ denote the secret and public variables, respectively. For a subset 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 $\{v_{i_1}, v_{i_2}, \ldots, v_{i_I}\}$ are taking all possible combinations of values. Let $\boldsymbol{k}_I$ be an m-dimensional 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 $(\boldsymbol{e}_{j_1} \oplus \boldsymbol{e}_{j_2}, \boldsymbol{k}_I) \xrightarrow{f} 1$ for $1 \le j_1 < j_2 \le n$, then $x_{j_1} x_{j_2}$ is not involved in the superpoly of the cube C_I.*

Proof. According to Lemma 1, $a_{\boldsymbol{u}} = 0$ for $\boldsymbol{u} \succeq (\boldsymbol{e}_{j_1} \oplus \boldsymbol{e}_{j_2}, \boldsymbol{k}_I)$. Therefore, for any $\boldsymbol{u}_{sec}$ with the j_1- and j_2-th bit 1, *i.e.* $\boldsymbol{u}_{sec} \succeq \boldsymbol{e}_{j_1} \oplus \boldsymbol{e}_{j_2}$ for $1 \le j_1 < j_2 \le n$, the coefficient $a_{\boldsymbol{u}} = 0$, this yields

$$p(\boldsymbol{x}, \boldsymbol{v}) = \bigoplus_{\substack{\boldsymbol{u}_{sec} \not\succeq \boldsymbol{e}_{j_1} \oplus \boldsymbol{e}_{j_2}, \\ \boldsymbol{u}_{pub} \succeq \boldsymbol{k}_I}} a_{\boldsymbol{u}} (\boldsymbol{x}^{\boldsymbol{u}_{sec}} \boldsymbol{v}^{\boldsymbol{u}_{pub} \oplus \boldsymbol{k}_I}),$$

which means that $x_{j_1} x_{j_2}$ does not appear in the superpoly $p(\boldsymbol{x}, \boldsymbol{v})$. □

Similarly, we define a subset of indices $J = \{j_1, j_2, \ldots, j_{|J|}\} \subset \{1, 2, \ldots, n\}$, and $k_j = 1$ if $j \in J$ and $k_j = 0$ otherwise. $\boldsymbol{k}_J$ is an n-dimensional vector such that $\boldsymbol{x}^{\boldsymbol{k}_J} = x_{j_1} x_{j_2} \cdots x_{j_{|J|}}$.

Corollary 1. *If there is no division trail such that $(\boldsymbol{k}_J, \boldsymbol{k}_I) \xrightarrow{f} 1$ for $1 \le j \le n$, then the superpoly of the cube C_I contains no multiple of $\boldsymbol{x}^{\boldsymbol{k}_J}$.*

Algorithm 4 MILP model of division property for KREYVIUM

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

4.1 New Upper Bound of the Superpoly Recovery Complexity

We will show that Proposition 5 and Corollary 1 can be used to get more information about the superpoly. Hence, the complexity of superpoly recovery can be reduced in some cases.

Assume the superpoly $p(\boldsymbol{x}, \boldsymbol{v})$ of cube C_I contains the following secret variables

$$S_{1Y} = \{x_1, x_2, \ldots, x_{|J|}\}.$$

Then all the possible $\binom{|J|}{2}$ quadratic terms are $S_2 = \{x_1x_2, x_1x_3, \ldots, x_{|J|-1}x_{|J|}\}$. By Proposition 5, we can determine which quadratic terms are involved in superpoly $p(\boldsymbol{x}, \boldsymbol{v})$. Denote the set of all the quadratic terms might be involved in $p(\boldsymbol{x}, \boldsymbol{v})$ as S_{2Y}, and the set having all the quadratic terms which are not involved in the superpoly as S_{2N}. We represent them respectively as

$$S_{2Y} = \{x_{i_1}x_{i_2}, x_{i_3}x_{i_4}, \ldots, x_{i_{k_2-1}}x_{i_{k_2}}\} \text{ and } S_{2N} = \{x_{j_1}x_{j_2}, x_{j_3}x_{j_4}, \ldots, x_{j_{l_2-1}}x_{j_{l_2}}\},$$

where $S_{2Y} \cup S_{2N} = S_2$ and $|S_{2Y}| + |S_{2N}| \triangleq |J_2| + |F_2| = |S_2| = \binom{|J|}{2}$.

More generally, for the superpoly of cube set C_I, denote S_i as the set of all the possible $\binom{|J|}{i}$ monomials of degree $i \geq 2$ involved in superpoly $p(\boldsymbol{x}, \boldsymbol{v})$, and S_{iY} as a set having all the elements exist in the superpoly, and all the rest are included in the set S_{iN}. Denote $|S_{iY}| = |J_i|$ and $|S_{iN}| = |F_i|$, then $|J_i| + |F_i| = \binom{|J|}{i}$ where $|S_{1Y}| = |J_1| = |J|$.

For a fixed cube C_I, with the help of Proposition 4 we can calculate the exact set S_{1Y}. Moreover, based on Proposition 5 we are able to calculate the exact sets S_{2Y} by solving the corresponding MILP models. We define d as the smallest integer such that $S_{iY} = \phi$ for $i > d$. Then the time complexity of superpoly recovery can be reduced from $2^{|I|+|J|}$ to

$$2^{|I|} \times (1 + \sum_{1 \leq i \leq d} |S_{iY}|) = 2^{|I|} \times (1 + \sum_{1 \leq i \leq d} |J_i|)\,,$$

where $(1 + \sum_{1 \leq i \leq d} |S_{iY}|)$ is the number of cube sums one needs to do by using Moebius transformation to recover the superpoly.

Note that this can improve the bound $2^{|I|+|J|}$ given in [TIHM17] especially when the number of involved secret variables in the superpoly and the degree of the superpoly d are large. The new complexity exactly captures the fact that we choose cubes such that the superpoly is quite sparse. Therefore the size of S_{iY} or S_{iN} $(1 \leq i \leq d)$ eventually determines the total complexity of recovering the superpoly.

We will apply this technique to further improve the complexity to recover the superpoly of Trivium and Kreyvium in Sect. 5. But before that, we first introduce some properties of sets S_{iY} and S_{iN}, and then based on them we start to construct sets S_{iY} for $i \geq 2$ from set S_{1Y}, S_{2Y} and S_{2N}. Therefore the calculation of complexity of recovering the superpoly actually is the computation of set S_{2Y} or S_{2N}. And more important is that in this case, after we obtain S_{1Y} and S_{2Y}, we can calculate the set S_{iY} for $3 \leq i \leq d$ by a simple algorithm, and all the workload of building the MILP models and running the solver is not necessary, especially knowing that running the solver is very time and computation consuming, this will significantly accelerate the recovery of the superpoly.

4.2 Clique Problem

We first introduce the graph clique, and then explain how our construction of the set S_{iY} for $3 \leq i \leq d$ is exactly to find the maximum i-cliques of a graph representing our superpoly.

Definition 3 (Clique). *[BM76] In a graph $G = (V, E)$, where V is the set of vertices and E is the set of edges, a subset $C \subseteq V$, such that each pair of vertices in C is connected by an edge is called a clique. A i-clique is a clique of i vertices. A 2-clique is just an edge, and a 3-clique is called a triangle.*

The proof that clique is NP-complete is due to Karp [Kar72]. In a graph $G = (V, E)$ where n is the cardinal of V to find a maximal k-clique by using brute-force algorithm, one can test whether G contains a k-vertex clique, and find any such clique that it contains. This algorithm checks each subgraph with k vertices to see whether it forms a clique. The straightforward time complexity is $\mathcal{O}(n^k k^2)$. Thus, when k is a variable, the time complexity is exponential. There has also been extensive research on heuristic algorithms for solving maximum clique problems which are not our focus in this paper.

Our problem of calculating the set S_{iY} $(3 \leq i \leq d)$ is equivalent to construct a maximal i-clique in our superpoly graph $G = (S_{1Y}, S_{2Y})$, where vertices correspond to the involved secret key bits (monomials) in the superpoly (*i.e.* S_{1Y}), and edges between any pairs of the vertices are the quadratic terms involved in the superpoly (*i.e.* S_{2Y}). Now our target is to find all i-cliques for $3 \leq i \leq d$ in our superpoly graph G, and we explain how to achieve this by a very simple algorithm.

Property 1. A monomial exists in S_{iY} iff all its divisors of degree $(i-1)$ are in S_{i-1Y}.

- example: 820-round Trivium with cube of dimension 62
  we have $S_{2Y} = \{x_{40}x_{41}, x_{40}x_{42}, x_{51}x_{52}\}$, then the only possible combination of 3 elements from S_{2Y} is $x_{40}x_{41} \vee x_{40}x_{42} \vee x_{51}x_{52} \triangleq x_{40}x_{41}x_{42}x_{51}x_{52}$, the degree of which is 5, and it cannot belong to S_{3Y} where all elements there should have degree 3. Thus, $S_{3Y} = \phi$.
- example: 591-round Trivium with cube of dimension 8
  Denote set $S_1 = \{x_{23}, x_{24}, x_{25}, x_{66}, x_{67}\} \triangleq \{\bar{1}, \bar{2}, \bar{3}, \bar{4}, \bar{5}\}$. Then $S_{2Y} = \{\bar{1}\bar{2}, \bar{1}\bar{4}, \bar{2}\bar{4}, \bar{3}\bar{4}, \bar{4}\bar{5}\}$. Take the union operation among any 3 elements out of S_{2Y}, and the only element with order 3 is $\bar{1}\bar{2}\bar{4}$. Then $S_{3Y} = \{\bar{1}\bar{2}\bar{4}\}$.

Based on this property we can construct S_{iY} $(i \geq 3)$ from S_{2Y} by a very simple algorithm, *e.g.* for the construction of S_{3Y}, we take the union operation of all possible combinations of three elements from S_{2Y}, and only keep the elements of degree 3. Similarly, we construct all S_{iY} from S_{iY} for $3 < i \leq d$.

To find the simplest i-clique is to find triangles in a graph. In a graph with m edges, there are at most $\Theta(m^{3/2})$ triangles. The worst case for this formula occurs when G is itself a clique. Taking 594-round Trivium with cube of dimension 8 (shown in Table 3) as an example, $m^{3/2} \approx 3764$ while $|S_{3Y}| = 789$.

Property 2. If $|S_{i-1Y}| < i$, then $|S_{iY}| = 0$.

- example: 833-round Trivium with cube of dimension 73
  $|S_{3Y}| = 1 < 4$, there $S_{4Y} = \phi$, and of course any $S_{iY} = \phi$ with $i \geq 4$.

Property 3. Any monomial in S_{iN} can be divided by at least one monomial in S_{i-1N}.

This property provides another approach to determine the maximal i-clique, by constructing its complement S_{iN} from S_{i-1N} and S_{1Y}. Take S_{3N} for example, one can append each element in S_{1Y} to S_{2N} and remove the repeated elements, then S_{3N} is obtained as well as its complement S_{3Y}. In a similar way, S_{iN} can be derived from S_{i-1N}.

5 Applications to Trivium and Kreyvium

5.1 Application to Trivium

Now we apply our method to lower the superpoly recovery complexity in this subsection to Trivium. We follow the same rules in [TIHM17] to choose the cube, *i.e.* cube indices are chosen as the following in our experiments: the odd indices $1, 3, \ldots, 2|I|-1$ are chosen, and the even indices $2, 4, \ldots, 2(|I|-40)$ are additionally chosen. Our results are summarized in Table 3, where we list the complexities when the complexity of superpoly recovery in [TIHM17] can be significantly improved.

Table 3. Our Results of TRIVIUM. The last two columns show the time complexity to recover the superpoly by [TIHM17] and our new bound.

#Rounds	Cube Size	$\|J_1\|$	$\|J_2\|$	Degree	$\|J_i\|$ $(3 \le i \le d)$	$1+\sum_{i=1}^{d}\|J_i\|$	[TIHM17]	Ours
591	8	5	5	3	1,0	$12 \approx 2^{3.58}$	2^{13}	$2^{11.58}$
592	8	25	68	5	77,37,6	$214 \approx 2^{7.74}$	2^{33}	$2^{15.74}$
593	8	57	513	14	2359,7069,15190, 24344,29550,27264, 19040,9945,3801, 1026,185,20,	$140364 \approx 2^{17.1}$	2^{65}	$2^{25.1}$
594	8	47	242	13	789,1723,2735, 3316,3141,2315, 1293,525,146,25,2	$16367 \approx 2^{14.0}$	2^{55}	$2^{22.0}$
800	44	12	37	4	32,6	$88 \approx 2^{6.46}$	2^{56}	$2^{50.46}$
805	49	15	8	3	1	$25 \approx 2^{4.64}$	2^{64}	$2^{53.64}$
808	52	12	11	3	2	$26 \approx 2^{4.7}$	2^{64}	$2^{56.7}$
809	53	25	146	8	399,610,272, 275,69,6	$1803 \approx 2^{10.82}$	2^{78}	$2^{63.82}$
814	54	7	2	2	0	$10 \approx 2^{3.32}$	2^{61}	$2^{57.32}$
816	55	19	32	4	16,2	$70 \approx 2^{6.13}$	2^{74}	$2^{61.13}$
819	61	8	2	2	0	$11 \approx 2^{3.46}$	2^{69}	$2^{64.46}$
820	62	8	3	2	0	$12 \approx 2^{3.58}$	2^{70}	$2^{65.58}$
825	65	7	2	2	0	$10 \approx 2^{3.32}$	2	2
829	66	10	2	2	0	$13 \approx 2^{3.7}$	2^{76}	$2^{69.7}$
830	69	7	1	2	0	$9 \approx 2^{3.17}$	2	
832	72	5	5	3	1	$12 \approx 2^{3.58}$	2^{77}	$2^{75.58}$
833	73†	7	6	3	1	$15 \approx 2^{3.91}$	2^{80}	$2^{76.91}$
833	74‡	1	0	1	0	2	-	2^{75}
839	78•	1	0	1	0	2	-	2^{79}

†: $I = \{1, 2, ..., 67, 69, 71, ..., 79\}$, $J = \{49, 58, 60, 74, 75, 76\}$
‡: $I = \{1, 2, ..., 69, 71, 73, ..., 79\}$, $J = \{60\}$
•: $I = \{1, ..., 33, 35, ..., 46, 48, ..., 80\}$ and $\boldsymbol{IV}[47] = 1$, $J = \{61\}$

The best result in [TIHM17] mounts to 832-round TRIVIUM with cube dimension $|I| = 72$ and the superpoly involves $|J| = 5$ key bits. Therefore, the complexity to recover the superpoly is 2^{77} in [TIHM17]. Using our technique, we further acquire that $|J_2| = 5, |J_3| = 1$ and the degree of superpoly is 3. So the complexity for superpoly recovery is $2^{|I|} \times \sum_{1 \le i \le 3} (|J_i| + 1) = 2^{72} \times (5 + 5 + 1 + 1) = 2^{72+3.58} = 2^{75.58}$.

We for the first time find a cube of dimension 73 for 833-round TRIVIUM, and $|S_{1Y}| = 7$. By the bound given in [TIHM17], the complexity is 2^{80}, which is infeasible. Applying our new bound, the complexity can be reduced to $2^{76.91}$. In this way, we can attack one more round than [TIHM17]. For the same number of 833-rounds, we can even find a cube of dimension 74 (remember the indices are

following the same rule in [TIHM17]) with only one secret key bit involved, therefore, the complexity is $2^{74+1} = 2^{75}$.

We further construct a 78-dimensional cube ($I = \{1, \ldots, 80\} \backslash \{12, 43\}$) whose superpoly after 836-rounds of initialization only involves 1 key bit with index $J = \{61\}$. So the complexity of the attack is 2^{79}. Since there are only 2 non-cube IVs, we let $\boldsymbol{IV}$ be all 2^2 possible non-cube IV assignments and run Algorithm 1. With the $2^2 = 4$ different assignments to non-cube IVs, we know that the key bit x_{61} is involved in the superpoly for $\boldsymbol{IV} = \texttt{0x0}, \texttt{0x4000}, \texttt{0x0}$ or $\boldsymbol{IV} = \texttt{0x0}, \texttt{0x4002}, \texttt{0x0}$. In other words, the 47-th IV bit must be assigned to constant 1.

Therefore, the corresponding experimental verification shows that Assumption 1 holds for Trivium in a small example. Therefore, we can expect that theoretically recovered superpolys also fulfill Assumption 1.

5.2 Applications to Kreyvium

We show our new results of cube attack on Kreyvium in this subsection. First, we use the cube of dimension 61 in [Liu17] for 849-rounds. We can determine that there are 23 secret variables involved in the corresponding superpoly. So one can evaluate the complexity of superpoly recovery as in [TIHM17], and it is bounded by $2^{61+23} = 2^{84}$. Moreover we choose a cube of dimension 85 for 870-rounds, and can find that there are only 2 secret variables appearing in the superpoly. By sum over all possible values of the cube variables, we find there are 39 secret variables appearing in the corresponding superpoly for 872-rounds, and again as in [TIHM17], we calculate the complexity $2^{85+39} = 2^{124}$. The indices of the chosen cube and the involved secret variables are listed in Table 4. Note that for 888-rounds, with a cube of dimension 102, there are 36 secret variables involved in the superpoly, therefore if we estimate the time complexity as in [TIHM17], this will be infeasible. We will show this cube attack can be achieved with our new bound technique.

Then given the cubes for 849-round, we explore the detailed of the superpoly, and enumerate the terms in set S_{iY} of degree i ($2 \leq i \leq 9$). Based on this, we provide a better bound for the complexity of superpoly recovery, which significantly reduces the complexity from 2^{84} to $2^{73.41}$.

For 872-round, with the constructed cube of dimension 85, we know there are 39 secret variables in the corresponding superpoly. Since the degree of the superpoly is only 2, $|S_{iY}| = 0$ for $i \geq 3$. Thus, our new bound for the complexity to recover the superpoly is $1 + \sum_{i=1}^{2} |S_{iY}|$. As defined S_{2Y} is the set all quadratic terms involved in the superpoly, so $|S_{2Y}|$ is apparently smaller than $|S_2| = \binom{39}{2} = 741$. Therefore we can upper bound the time complexity by $2^{85} \times (1 + 39 + \binom{39}{2}) = 2^{85+9.61} = 2^{94.61}$. Similarly we can calculate the complexity for 888-rounds as $2^{111.38}$. The details of the calculation of the reduced complexities are given in Table 5.

6 Conclusions

We further investigate the sparse property of the superpoly of cube attacks against stream ciphers, and give a better bound of the time complexity of superpoly recovery. This new bound can reduce the complexity of superpoly recovery of Triviumespecially when the number of secret variables involved in the superpoly and the degree of superpoly are large, and even enables us to reach more rounds. We also analyze Kreyvium, and provide the best key recovery attack.

We will continue analyzing more rounds of Trivium and Kreyvium with different cube dimensions, and try applications to more ciphers like Grain128a, ACORN, etc.

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

#Rds	$\|I\|$	Degree d	Indices of involved secret variables J	Time complexity
849	61†	9	47, 49, 51, 53, 55, 64, 66, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 89, 90, 91, 92, 93 ($\|J\| = 23$)	2^{84}
870	85‡	1	39,98 ($\|J\| = 2$)	$2^{86.58}$
872	85‡	2	5, 6, 20, 21, 22, 30, 31, 37, 39, 40, 41, 49, 53, 54, 56, 57, 58, 63, 64, 65, 66, 67, 74, 75, 76, 89, 91, 92, 93, 96, 98, 99, 100, 108, 122, 123, 124, 125, 126 ($\|J\| = 39$)	2^{124}
888	102⋆	2	7, 8, 9, 17, 18, 22, 32, 33, 41, 45, 46, 47, 48, 51, 52, 53, 55, 56, 57, 58, 68, 76, 77, 78, 79, 80, 81, 83, 84, 85, 91, 100, 114, 115, 116, 117 ($\|J\| = 36$)	-

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

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

⋆: I = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 34, 36, 38, 40, 42, 44, 46, 48, 50, 52, 54, 56, 58, 60, 62, 64, 66, 68, 70, 72, 74, 76, 78, 80, 82, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128}

Table 5. Our New Results of KREYVIUM with Reduced Complexity

#Rounds	Cube Size	$\|J_1\|$	$\|J_2\|$	Degree d	$\|J_i\|$ $(3 \le i \le d)$	$1 + \sum_{i=1}^{d} \|J_i\|$	Time complexity
849	61	23	158	9	555,1162,1518,1235,618,156,26	$5452 \approx 2^{12.41}$	$2^{73.41}$
872	85	39	$\le \binom{39}{2}$	2	0	$781 \approx 2^{9.61}$	$2^{94.61}$
888	102	36	$\le \binom{36}{2}$	2	0	$667 \approx 2^{9.38}$	$2^{111.38}$

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.

BM76. John Adrian Bondy and Uppaluri Siva Ramachandra Murty. *Graph theory with applications*, volume 290. Macmillan London, 1976.

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

CCF+16. Anne Canteaut, Sergiu Carpov, Caroline Fontaine, Tancrède Lepoint, María Naya-Plasencia, Pascal Paillier, and Renaud Sirdey. Stream ciphers: A practical solution for efficient homomorphic-ciphertext compression. In Thomas Peyrin, editor, *FSE*, volume 9783 of *LNCS*, pages 313–333. Springer, 2016.

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).

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.

DLWQ17. Xiaoyang Dong, Zheng Li, Xiaoyun Wang, and Ling Qin. Cube-like attack on round-reduced initialization of ketje sr. *IACR Trans. Symmetric Cryptol.*, 2017(1):259–280, 2017.

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.

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.

HWX+17. Senyang Huang, Xiaoyun Wang, Guangwu Xu, Meiqin Wang, and Jingyuan Zhao. Conditional Cube Attack on Reduced-Round Keccak Sponge Function. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT Part II*, volume 10211 of *Lecture Notes in Computer Science*, pages 259–288. Springer, 2017.

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

Kar72. Richard M. Karp. Reducibility among combinatorial problems. In Raymond E. Miller and James W. Thatcher, editors, *Proceedings of a symposium on the Complexity of Computer Computations, held March 20-22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York.*, The IBM Research Symposia Series, pages 85–103. Plenum Press, New York, 1972.

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.

LBDW17. Zheng Li, Wenquan Bi, Xiaoyang Dong, and Xiaoyun Wang. Improved conditional cube attacks on keccak keyed modes with milp method. In *ASIACRYPT*. Springer, 2017. (to appear).

LDW17. Zheng Li, Xiaoyang Dong, and Xiaoyun Wang. Conditional cube attack on round-reduced ASCON. *IACR Trans. Symmetric Cryptol.*, 2017(1):175–202, 2017.

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.

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.

ST17. Yu Sasaki and Yosuke Todo. New impossible differential search tool from design and cryptanalysis aspects - revealing structural properties of several ciphers. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT Part III*, volume 10212 of *LNCS*, pages 185–215. Springer, 2017.

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

TIHM17. Yosuke Todo, Takanori Isobe, Yonglin Hao, and Willi Meier. Cube attacks on non-blackbox polynomials based on division property. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO Part III*, volume 10403 of *LNCS*, pages 250–279. Springer, 2017.

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.

WLV+14. Qingju Wang, Zhiqiang Liu, Kerem Varici, Yu Sasaki, Vincent Rijmen, and Yosuke Todo. Cryptanalysis of reduced-round SIMON32 and SIMON48. In *INDOCRYPT 2014*, volume 8885 of *LNCS*, pages 143–160, 2014.

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.