An Algebraic Method to Recover Superpolies in Cube Attacks

Chen-Dong Ye and Tian Tian

PLA Strategic Support Force Information Engineering University, 62 Kexue Road, Zhengzhou, 450001, China. ye_chendong@126.com,tiantian_d@126.com

Abstract. Cube attacks are an important type of key recovery attacks against NFSR-based cryptosystems. The key step in cube attacks closely related to key recovery is recovering superpolies. However, in the previous cube attacks including original, division property based, and correlation cube attacks, the algebraic normal form of superpolies could hardly be shown to be exact due to an unavoidable failure probability or a requirement of large time complexity. In this paper, we propose an algebraic method aiming at recovering the exact algebraic normal forms of superpolies practically. Our method is developed based on degree evaluation method proposed by Liu in Crypto-2017. As an illustration, we apply our method to Trivium. As a result, we recover the algebraic normal forms of some superpolies for the 818-, 835-, 837-, and 838-round Trivium. Based on these superpolies, on a large set of weak keys, we can recover at least five key bits equivalently for up to the 838-round Trivium with a complexity of about 2^{37}. Besides, for the cube proposed by Liu in Crypto-2017 as a zero-sum distinguisher for the 838-round Trivium, it is proved that its superpoly is not zero-constant. Hopefully, our method would provide some new insights on cube attacks against NFSR-based ciphers.

Key Words: Trivium, cube attacks, key recovery attacks, algebraic normal form

Keywords: No keywords given.

1 Introduction

The cube attack was first proposed by Dinur and Shamir at Eurocrypt-2009 in [1]. Later, there were many improvements on it such as cube testers [2], dynamic cube attacks [3], conditional cube attacks [4], division property based cube attacks [5, 6] and correlation cube attacks [7]. Due to these improvements, cube attacks have become more and more powerful. In particular, it is one of the most important cryptanalytic tools against Trivium.

In the original cube attacks [1, 8, 9, 10], the main aim is to find low-degree superpolies on key variables by performing experimental tests. In [1], the authors recovered 35 linear superpolies of the 767-round Trivium. In [8], quadraticity tests were first applied to the cube attacks against Trivium. As a result, the authors found 41 linear and 38 quadratic superpolies for the 709-round Trivium. In [9], the authors proposed two new ideas concerning cube attacks against Trivium. One was a recursive method to construct useful cubes. The other was simultaneously testing a lot of subcubes of a large cube using the Meobius transformation. They found 12 linear and 6 quadratic superpolies for the 799-round Trivium. In [10], by exploiting a kind of linearization technique, the authors proposed a new framework to find nonlinear superpolies with low complexities. As a result, they found 6 linear and 2 nonlinear superpolies for the 802-round Trivium. In the above experimental cube attacks, it needs to test a large number of cubes to find desirable

ones, while testing cubes of size greater than 35 is time consuming. Hence, the sizes of cubes are typically confined to 40.

In [5], Todo et al. introduced division property to cube attacks. For a cube I, by solving the corresponding mixed integer linear programming (MILP) models built according to the propagation rules of division property, they could identify a set of key variables which includes the key variables appearing in the superpoly p_I. Then, by constructing the truth tables of p_I corresponding to randomly chosen assignments of non-cube variables, they attempted to find a proper one ensuring that p_I was non-constant. Finally they recovered p_I by its truth table. Due to division property and the power of MILP solvers, large cubes could be explored. For example, in [5], it was shown that the superpoly of a given 72-dimensional cube was dependent on at most five key variables for the 832-round Trivium. In [6], the authors improved the work in [5] in finding a proper non-cube variables assignment and reducing the complexity of recovering the superpoly. It was shown in [6] that the superpoly of a given 78-dimensional cube was dependent on at most one key variable for the 839-round Trivium. For division property based cube attacks, the advantage is that large cubes could be explored and the complexity of recovering superpolies could be estimated theoretically. The implicit disadvantage is that the theory of division property could not ascertain that a superpoly for a cube is non-constant. Hence the key recovery attacks on the 832-round Trivium in [5] and on the 839-round Trivium in [6] are only possible which may be only a distinguisher.

In [11], the authors proposed a method to recover the superpoly of a cube with the help of bit-based division property. According to the results of [11], the superpoly of the cube proposed in [6] to attack the 839-round Trivium is zero-constant. Besides, for the cube given in [5] to attack the 832-round Trivium, they found an assignment to noncube IV variables which ensures the corresponding superpoly is non-constant. However, the complexity of the method proposed in [11] largely depends on the number of key variables appearing in the superpoly, and so it is not suitable to recover superpolies which include many key variables. Furthermore, in [11], the authors did not establish new attacks on Trivium variants with more than 832 initialization rounds.

In [7], the authors proposed the correlation cube attacks. For a cube C_I, the authors first tried to find a set of low-degree polynomials G, called a basis, such that the superpoly p_I could be factored into $p_I = \bigoplus_{g \in G} g \cdot f_g$ formally. Then, by exploiting the correlation relations between the low-degree basis G and the superpoly p_I, they could obtain a set of probabilistic equations on the secret key variables since f_g is unknown. It was reported in [7] that five key variables of the 835-round Trivium could be recovered with 2^{44} time complexity, 2^{45} keystream bits, and 2^{51} preprocessing time.

In [12], Fu et al. gave a dedicated attack on the 855-round Trivium which somewhat resembled dynamic cube attacks. Their main idea is finding a simple polynomial P_1 such that the output bit polynomial z could be formally represented as $z = P_1P_2 \oplus P_3$ where P_2 is complex while P_3 is a low-degree polynomial on IV variables compared with z. Consequently, $(P_1 \oplus 1)z = (P_1 \oplus 1)P_3$ will be a low-degree polynomial on IV variables. Then they guessed some secret key expressions involved in P_1. For a group of right guesses, $(P_1 \oplus 1)z$ will be a low-degree polynomial on IV variables. Otherwise, $(P_1 \oplus 1)z$ is expected to be a high-degree polynomial. They declared that three secret key bits could be recovered for the 855-round Trivium with the online complexity of 2^{74}. A shortage of the attack described in [12] is that no estimation was given on the successful probability for wrong guesses.

However, in [13], the authors pointed out that the attacks in [12] is questionable. More specifically, the attack against 721-Trivium was experimentally verified to fail and some complexity analysis also indicated that the 855-round attack was questionable.

1.1 Our Contributions.

In this paper, we propose an algebraic method to recover superpolies in cube attacks, which improves the original cube attacks in recovering the superpolies with proved correctness and overcoming the low-degree restriction.

The basic idea of our attacks is, by making use of internal state bit variables $\boldsymbol{s}^{(r_1)} = (s_1^{(r_1)}, s_2^{(r_1)}, \ldots, s_N^{(r_1)})$, dividing the polynomial representation $f_r(\boldsymbol{key}, \boldsymbol{iv})$ of an r-round cipher into an r_1-round polynomial representation $\boldsymbol{s}^{(r_1)}(\boldsymbol{key}, \boldsymbol{iv})$ and an r_2-round polynomial representation $g_{r_2}(\boldsymbol{s}^{(r_1)})$ such that $f_r = g_{r_2}(\boldsymbol{s}^{(r_1)}(\boldsymbol{key}, \boldsymbol{iv}))$. Then it is possible for us to compute superpolies algebraically for a class of cubes which are called *useful cubes* in the rest of this paper. The criterion of a useful cube plays a key role in calculating superpolies in practice. In particular, for a useful cube I, to compute the superpoly $Q_{\{s_{i_1}^{(r_1)}, s_{i_2}^{(r_1)}, \ldots, s_{i_l}^{(r_1)}\}}$ of I in the polynomial $s_{i_1}^{(r_1)}(\boldsymbol{key}, \boldsymbol{iv}) s_{i_2}^{(r_1)}(\boldsymbol{key}, \boldsymbol{iv}) \cdots s_{i_l}^{(r_1)}(\boldsymbol{key}, \boldsymbol{iv})$ for a term $s_{i_1}^{(r_1)} s_{i_2}^{(r_1)} \cdots s_{i_l}^{(r_1)}$ appearing in the algebraic normal form of $g_{r_2}(\boldsymbol{s}^{(r_1)})$, it is only necessary to know the *maximum degree terms* in each $s_{i_j}^{(r_1)}(\boldsymbol{key}, \boldsymbol{iv})$ for $1 \leq j \leq l$. Hence a superpoly $Q_{\{s_{i_1}^{(r_1)}, s_{i_2}^{(r_1)}, \ldots, s_{i_l}^{(r_1)}\}}$ could be computed in practice and so is the targeted superpoly p_I in $f_r(\boldsymbol{key}, \boldsymbol{iv})$ which is equal to the summation of all possible $Q_{\{s_{i_1}^{(r_1)}, s_{i_2}^{(r_1)}, \ldots, s_{i_l}^{(r_1)}\}}$.

As an illustration, we apply our method to the round-reduced Trivium, and we obtain the following results.

- For Trivium variants with 800-832 initialization rounds, we randomly test 10000 cubes of size 33-36, and obtain about 175 useful cubes for each variant in average. This indicates that useful cubes exist widely.
- We recover the superpolies of some useful cubes for the 818-, 835-, 837-, and 838-round Trivium. With these recovered superpolies, we could establish the following attacks.
  - For the 818-round Trivium, we could recover at least 10 key bits equivalently on about 2^{70} weak keys;
  - For the 835-round Trivium, we could recover at least 12 key bits equivalently on about 2^{67} weak keys;
  - For the 837-round Trivium, we could obtain 5 deterministic and 2 probabilistic equations on about $2^{72.20}$ weak keys;
  - For the 838-round Trivium, we could obtain 5 deterministic and 2 probabilistic equations on about $2^{71.75}$ weak keys.

We further compare our attacks with the original cube attacks (OCA), the division property based cube attacks (DPCA), and correlation cube attacks (CCA). First, our attacks can recover the superpoly exactly. Second, we can attack Trivium variants with high rounds using cubes of relative small sizes, i.e., we can reach the 838-round Trivium with cubes of sizes 36-37. Compared with original cube attacks, we can improve more than 30 rounds at the cost of increasing the sizes of cubes slightly. In division property based cube attacks, they need cubes of sizes over 70 to attack Trivium variants with more than 830 initialization rounds. Besides, since it can not ensure that the superpoly of a cube is non-constant in division property based cube attacks, the attacks in [5, 14, 6] may be only distinguishing attacks. In correlation cube attacks, they can attack the 835-round Trivium with cubes of sizes 36-37, but they can not recover the exact superpoly of a cube and the correlation probability is computed experimentally. We summarise these comparisons in Table 1, where the "key size" column lists the size of keys which are vulnerable to each attack.

[3]The attacks proposed in [6] against the 833-, 835- and 836-round Trivium may be only distinguishing attacks.

Table 1: Comparison with previous cube attacks

attacks	rounds/cube size	# of recovered bits	key size	ref.
OCA	767/28-31	35	2^{80}	[1]
	709/19-23	79	2^{80}	[8]
	799/32-37	12	2^{80}	[9]
	802/34-36	8	2^{80}	[10]
DPCA	832/72	1	2^{80}	[5, 14]
	833/73-74	2	2^{80}	[6][3]
	835/77	1	2^{80}	
	836/78	1	2^{80}	
	839/78	distinguisher	- -	
CCA	805/28	7	2^{80}	[7]
	835/36-37	5	2^{80}	
Our Method	818/33	10	2^{72}	Sect.4
	835/35	12	2^{67}	
	837/36-37	5	$2^{72.20}$	
	838/36-37	5	$2^{71.75}$	

1.2 Organization

The rest of this paper is organized as follows. In Section 2, we give some necessary introductions on cube attacks, the Numeric Mapping method, and the IV Representation techniques. In Section 3, we show the general idea of our attacks. In Section 4, we apply our attacks to Trivium. Finally, Section 5 concludes this paper.

2 Preliminaries

2.1 Boolean Functions and Algebraic Degree.

A Boolean function on n variables is a mapping from $\mathbb{F}_2^n$ to $\mathbb{F}_2$, where $\mathbb{F}_2$ is the finite field of two elements and $\mathbb{F}_2^n$ is an n-dimensional vector space over $\mathbb{F}_2$. A Boolean function f can be represented by a polynomial on n variables over $\mathbb{F}_2$,

$$f(x_1, x_2, \ldots, x_n) = \bigoplus_{c=(c_1,c_2,\ldots,c_n)\in\mathbb{F}_2^n} a_c \prod_{i=1}^{n} x_i^{c_i},$$

which is called the algebraic normal form (ANF) of f. In this paper, $u = a_c \prod_{i=1}^{n} x_i^{c_i}$ ($a_c \neq 0$) is called a term of f. The algebraic degree of a Boolean function is denoted by $\deg(f)$ and defined as

$$\deg(f) = \max\{wt(c) | a_c \neq 0\},$$

where $wt(c)$ is the Hamming Weight of c. In this paper, we also care about the algebraic degree of f on a subset I of $\{x_1, x_2, \ldots, x_n\}$, which is denoted by $\deg_I(f)$ and defined as

$$\deg_I(f) = \max\{wt_I(c) | a_c \neq 0\},$$

where $wt_I(c) = |\{i | c_i \neq 0 \text{ and } x_i \in I\}|$.

Algorithm 1 Pseudo-code of Trivium

$(s_1, s_2, \ldots, s_{93}) \leftarrow (k_0, k_1, \ldots, k_{79}, 0, \ldots, 0)$;
$(s_{94}, s_{95}, \ldots, s_{177}) \leftarrow (v_0, v_1, \ldots, v_{79}, 0, \ldots, 0)$;
$(s_{178}, s_{179}, \ldots, s_{288}) \leftarrow (0, \ldots, 0, 1, 1, 1)$;
for i from 1 to N **do**
  if $i > 1152$ **then**
    $z_{i-1152} \leftarrow s_{66} \oplus s_{93} \oplus s_{162} \oplus s_{177} \oplus s_{243} \oplus s_{288}$;
  end if
  $t_1 \leftarrow s_{66} \oplus s_{91} \cdot s_{92} \oplus s_{93} \oplus s_{171}$;
  $t_2 \leftarrow s_{162} \oplus s_{175} \cdot s_{176} \oplus s_{177} \oplus s_{264}$;
  $t_3 \leftarrow s_{243} \oplus s_{286} \cdot s_{287} \oplus s_{288} \oplus s_{69}$;
  $(s_1, s_2, \ldots, s_{93}) \leftarrow (t_3, s_1, \ldots, s_{92})$;
  $(s_{94}, s_{95}, \ldots, s_{177}) \leftarrow (t_1, s_{94}, \ldots, s_{176})$;
  $(s_{178}, s_{179}, \ldots, s_{288}) \leftarrow (t_2, s_{178}, \ldots, s_{287})$;
end for

2.2 Description of Trivium

Trivium is a bit oriented synchronous stream cipher designed by Cannière and Preneel, which is one of eSTREAM hardware-oriented portfolio ciphers. It accepts an 80-bit key and an 80-bit initialization vector. For a more detailed and formal description, we refer the reader to [15].

The main building block of Trivium is a 288-bit Galois nonlinear feedback shift register with three registers. In every clock cycle, there are three bits of the internal state updated by quadratic feedback functions and all the other bits of the internal state are updated by shifting. The internal state of Trivium, denoted by $(s_1, s_2, \ldots, s_{288})$, is initialized by loading an 80-bit secret key and an 80-bit IV into the registers, and setting all the remaining bits to 0 except for the last three bits of the third register. Then, the algorithm would not output any keystream bit until the internal state is updated 1152 rounds, see Algorithm 1 for details.

2.3 Superpoly.

The concept of superpoly was first proposed in [1]. Let $f(x_1, x_2, \ldots, x_m)$ be an m-variable polynomial and $I = \{x_{i_1}, x_{i_2}, \ldots, x_{i_d}\}$ be a subset of $\{x_1, x_2, \ldots, x_m\}$. Denote $t_I = \prod_{j=1}^{d} x_{i_j}$, the product of variables in I. Then it is clear that the following representation

$$f = f_1 \cdot t_I + f_2$$

for f is unique, where f_1 does not contain any common variable with t_I and every term in f_2 is not divisible by t_I. The polynomial f_1 is called the *superpoly* of t_I in f. For the sake of convenience, we denote f_1 by $\frac{f}{t_I}$ in the following paper.

2.4 Cube Attacks.

The idea of cube attack was first proposed by Dinur and Shamir in [1]. In the cube attack against stream ciphers, an output bit z is described as a tweakable polynomial f on key variables $\boldsymbol{key} = (k_0, k_1, \ldots, k_{n-1})$ and public IV variables $\boldsymbol{iv} = (v_0, v_1, \ldots, v_{m-1})$, where n and m are positive integers, i.e.,

$$z = f(\boldsymbol{key}, \boldsymbol{iv}).$$

Let I be a subset containing d public variables called cube variables, where $1 \leq d \leq m$. Without loss of generality, we assume that $I = \{v_0, v_1, \ldots, v_{d-1}\}$. Let us denote

$$p_I(\boldsymbol{key}) = \frac{f}{t_I}, \tag{1}$$

the superpoly of t_I in f under the condition that all m IV variables are set to 0 except the d variables in I. Let C_I be a set of assignments for IV variables containing 2^d m-tuples in which the variables in I are assigned to all the possible combinations of 0/1 while all the other IV variables, called non-cube IV variables, are assigned to constants. The set C_I is called a d-dimensional cube defined by I. In this paper, we set all the non-cube IV variables to 0's and call I a cube for simplicity. A key observation in cube attacks is that the summation of f over all the 2^d possible vectors in C_I leads to p_I, i.e.,

$$p_I(\boldsymbol{key}) = \bigoplus_{\boldsymbol{v} \in C_I} f(\boldsymbol{key}, \boldsymbol{v}). \tag{2}$$

If $p_I(\boldsymbol{key})$ is not a constant polynomial, then this means that by choosing IVs, one can obtain an equation in key variables. Otherwise, (2) provides a distinguisher on the cipher. Hence an attacker in cube attacks focuses on recovering $p_I(\boldsymbol{key})$. Because f is treated as a black-box polynomial, in practice p_I is not algebraically calculated from (1). Hence, original cube attacks resort to low-degree polynomial tests with a certain failure probability.

2.5 The Numeric Mapping.

The numeric mapping was firstly introduced by Liu in [16], which was the core technique of the degree evaluation method for NFSR-based cryptosystems in [16]. Let

$$f(x_1, x_2, \ldots, x_m) = \bigoplus_{c=(c_1,c_2,\ldots,c_m) \in \mathbb{F}_2^m} a_c \prod_{i=1}^{m} x_i^{c_i}$$

be an m-variable Boolean function. The numeric mapping, denoted by DEG, is defined as follows

$$\begin{aligned} \mathrm{DEG} : \mathbb{B}_m \times \mathbb{Z}^m &\to \mathbb{Z} \\ (f, D) &\mapsto \max_{a_c \neq 0} \sum_{i=1}^{m} c_i d_i, \end{aligned}$$

where $D = (d_1, d_2, \ldots, d_m)$, $\mathbb{B}_m$ is the set of all m-variable Boolean functions.

With the numeric mapping, the numeric degree of a composite function can be defined. Assume that $g_1, g_2, \ldots, g_m$ are n-variable Boolean functions and $h = f(g_1, g_2, \ldots, g_m)$ is a composite function. The numeric degree of h is defined as $\mathrm{DEG}(f, \deg(G))$, where $G = (g_1, g_2, \ldots, g_m)$ and $\deg(G) = (\deg(g_1), \deg(g_2), \ldots, \deg(g_m))$. Furthermore, if we have $\deg(g_i) \leq d_i$ for $1 \leq i \leq m$, then it can be seen that

$$\deg(h) \leq \mathrm{DEG}(f, \deg(G)) \leq \mathrm{DEG}(f, D)$$

where $D = (d_1, d_2, \ldots, d_m)$.

Based on the numeric mapping, in [16], Liu proposed an iterative algorithm for giving an upper bound on the algebraic degree of the output bit after r rounds for a Trivium-like cipher. In this algorithm, they first initialized the degrees of the initial internal state bits and then iteratively estimated the algebraic degree of the internal state bits at time instance t for $1 \leq t \leq r$. Thus, the estimated degree of the output bit could be calculated according to the output function. Moreover, when estimating the algebraic degree of the update bits, the author treated the product of two adjacent internal state bits as a whole and recursively expressed these two bits to obtain a more accurate estimation.

2.6 The IV Representation.

The IV representation was first proposed by Fu et al. in [17], which was used to determine the nonexistence of some IV terms in the output bit of Grain-128. For a stream cipher with m IV variables, i.e., $v_0, v_1, \ldots, v_{m-1}$, and n key variables, i.e., $k_0, k_1, \ldots, k_{n-1}$, an internal state bit (or the output bit) s can be seen as a polynomial on key and IV variables, i.e.,

$$s = f(\boldsymbol{key}, \boldsymbol{iv}) = \bigoplus_{I,J} \prod_{v_i \in I} v_i \prod_{k_j \in J} k_j.$$

The IV representation of a term $u = \prod_{v_i \in I} v_i \prod_{k_j \in J} k_j$ is defined as $u_{IV} = \prod_{v_i \in I} v_i$. Based on the definition of IV representation of a term, the IV representation of s is defined as follows,

$$s_{IV} = \sum_{I} \prod_{v_i \in I} v_i.$$

3 An Algebraic Method to Recover Superpolies

Recall that in an original cube attack, a desirable superpoly is not algebraically computed from (1), since the output bit polynomial is treated as a black-box polynomial. Hence original cube attacks resort to low-degree polynomial tests such as BLR linearity tests with a certain failure probability. Consequently, previously recovered superpolies in original cube attacks are convincing but without proved correctness. In this section, we shall give an algebraic method to recover superpolies in cube attacks against an NFSR-based stream cipher.

In Subsection 3.1, we describe the rationality of our idea and the general framework for realizing the idea. Then to make our idea practical, we introduce a new criterion of useful cubes in Subsection 3.2. Consequently, an algorithm is proposed to find useful cubes efficiently in Subsection 3.3. Finally, in Subsection 3.4, for a useful cube, we show how to recover its superpoly as well as some auxiliary techniques.

3.1 An Overview of Our Method

We represent the superpoly of a cube by internal state bits for the target cipher. Let us fix a time instance $t \geq 0$ which is less than the number of initialization rounds and denote the internal state bits of the target cipher at the time instance t by $\boldsymbol{s} = (s_1^{(t)}, s_2^{(t)}, \ldots, s_N^{(t)})$, where N is the internal state size of the target cipher. Then an output bit z of the target cipher also can be described by a polynomial on $\boldsymbol{s} = (s_1^{(t)}, s_2^{(t)}, \ldots, s_N^{(t)})$, i.e.,

$$z = g_t(s_1^{(t)}, s_2^{(t)}, \ldots, s_N^{(t)}) = \bigoplus_{c=(c_1,c_2,\ldots,c_N) \in \mathbb{F}_2^N} a_c \prod_{i=1}^{N} (s_i^{(t)})^{c_i}, \tag{3}$$

where $a_c \in \{0, 1\}$. Furthermore, note that each internal state bit $s_i^{(t)}$ $(1 \leq i \leq N)$ could be represented by a polynomial on key and IV variables, i.e.,

$$s_i^{(t)} = s_i^{(t)}(\boldsymbol{key}, \boldsymbol{iv}). \tag{4}$$

Taking (4) into (3) yields

$$z = g_t(s_1^{(t)}(\boldsymbol{key}, \boldsymbol{iv}), s_2^{(t)}(\boldsymbol{key}, \boldsymbol{iv}), \ldots, s_N^{(t)}(\boldsymbol{key}, \boldsymbol{iv})). \tag{5}$$

Following from (5), the superpoly of I in z can be computed as

$$\begin{aligned} p_I &= \frac{g_t(s_1^{(t)}(\boldsymbol{key},\boldsymbol{iv}), s_2^{(t)}(\boldsymbol{key},\boldsymbol{iv}), \ldots, s_N^{(t)}(\boldsymbol{key},\boldsymbol{iv}))}{t_I} \\ &= \frac{\bigoplus\limits_{c=(c_1,c_2,\ldots,c_N)\in\mathbb{F}_2^N} a_c \prod_{i=1}^N (s_i^{(t)}(\boldsymbol{key},\boldsymbol{iv}))^{c_i}}{t_I} \\ &= \bigoplus_{\substack{c=(c_1,c_2,\ldots,c_N)\in\mathbb{F}_2^N \\ a_c=1}} \frac{a_c \prod_{i=1}^N (s_i^{(t)}(\boldsymbol{key},\boldsymbol{iv}))^{c_i}}{t_I}. \end{aligned} \tag{6}$$

For the sake of convenience, we simply denote $s_i^{(t)}(\boldsymbol{key},\boldsymbol{iv})$ by $s_i^{(t)}$ in the rest of the paper. Then (6) implies that

$$p_I = \bigoplus_{\substack{c=(c_1,c_2,\ldots,c_N)\in\mathbb{F}_2^N \\ a_c=1}} \frac{a_c \prod_{i=1}^N (s_i^{(t)})^{c_i}}{t_I}. \tag{7}$$

Note that $a_c \prod_{i=1}^N (s_i^{(t)})^{c_i}$ with $a_c = 1$ is a term of g_t. If we denote all terms of g_t by $T(g_t)$, i.e.,

$$T(g_t) = \{a_c \prod_{i=1}^N (s_i^{(t)})^{c_i} | a_c = 1, c = (c_1, c_2, \ldots, c_N) \in \mathbb{F}_2^N\},$$

then it follows from (7) that

$$p_I = \bigoplus_{s_{i_1}^{(t)} s_{i_2}^{(t)} \cdots s_{i_l}^{(t)} \in T(g_t)} \frac{s_{i_1}^{(t)} s_{i_2}^{(t)} \cdots s_{i_l}^{(t)}}{t_I}. \tag{8}$$

This indicates that if we could compute the superpoly of t_I in $s_{i_1}^{(t)} s_{i_2}^{(t)} \cdots s_{i_l}^{(t)}$ for every term $s_{i_1}^{(t)} s_{i_2}^{(t)} \cdots s_{i_l}^{(t)}$ of g_t, then their summation is the desirable superpoly p_I in cube attacks.

In the following paper, we shall recover p_I based on the equality (8). Hence it is clear that the superpolies recovered with our method will be correct with probability 1. In specific, for a given set I of cube variables and a time instance t, there are three main steps:

- Step 1. Compute the ANF of g_t.
- Step 2. For each term $s_{i_1}^{(t)} s_{i_2}^{(t)} \cdots s_{i_l}^{(t)} \in T(g_t)$, compute the superpoly

$$Q_{\{s_{i_1}^{(t)}, s_{i_2}^{(t)}, \ldots, s_{i_l}^{(t)}\}} = \frac{s_{i_1}^{(t)} s_{i_2}^{(t)} \cdots s_{i_l}^{(t)}}{t_I}.$$

- Step 3. Compute

$$p_I = \bigoplus_{s_{i_1}^{(t)} s_{i_2}^{(t)} \cdots s_{i_l}^{(t)} \in T(g_t)} Q_{\{s_{i_1}^{(t)}, s_{i_2}^{(t)}, \ldots, s_{i_l}^{(t)}\}}.$$

We give some explanations for our method. First, all the non-cube IV variables are set to 0. That is to say, $s_i^{(t)}(\boldsymbol{key},\boldsymbol{iv})$ is a polynomial on $n+d$ variables consisting of n key variables and d cube variables, $1 \le i \le N$. Second, the choice of the time instance t obeys the following two rules:

Rule 1 One can compute the ANF of $g_t(s_1^{(t)}, s_2^{(t)}, \ldots, s_N^{(t)})$, where $s_i^{(t)}$ is treated as a bit variable. As t *decreases*, the ANF of $g_t(s_1^{(t)}, s_2^{(t)}, \ldots, s_N^{(t)})$ will become more and more complex.

Rule 2 For i from 1 to N, one can compute the ANF of $s_i^{(t)}(\boldsymbol{key}, \boldsymbol{iv})$. As t *increases*, the ANF of $s_i^{(t)}(\boldsymbol{key}, \boldsymbol{iv})$ will become more and more complex.

Third, we point out that to compute $Q_{\{s_{i_1}^{(t)}, s_{i_2}^{(t)}, \ldots, s_{i_l}^{(t)}\}}$ is a difficult problem in this framework even when the above two rules are satisfied, since it is difficult to compute the complete ANF of the product $s_{i_1}^{(t)} s_{i_2}^{(t)} \cdots s_{i_l}^{(t)}$ when treating $s_{i_j}^{(t)}$ as a polynomial on key and IV variables. To solve this problem, in Subsection 3.2, we give a criterion to choose useful cubes for which we could compute $Q_{\{s_{i_1}^{(t)}, s_{i_2}^{(t)}, \ldots, s_{i_l}^{(t)}\}}$ in practice without completely expanding the product $s_{i_1}^{(t)} s_{i_2}^{(t)} \cdots s_{i_l}^{(t)}$ in its ANF.

3.2 A New Criterion of Useful Cubes

Let $Q_{\{s_{i_1}^{(t)}, s_{i_2}^{(t)}, \ldots, s_{i_l}^{(t)}\}}$ be as in the previous subsection. To compute $Q_{\{s_{i_1}^{(t)}, s_{i_2}^{(t)}, \ldots, s_{i_l}^{(t)}\}}$, we use the following expression

$$Q_{\{s_{i_1}^{(t)}, s_{i_2}^{(t)}, \ldots, s_{i_l}^{(t)}\}} = \bigoplus \frac{t_1 t_2 \cdots t_l}{t_I}, \tag{9}$$

where t_j runs through $T(s_{i_j}^{(t)})$ independently for $1 \le j \le l$. The difficulty of computing (9) lies in that there are too many products, say $t_1 t_2 \cdots t_l$, need to compute. If $t_1 t_2 \cdots t_l$ is not divisible by t_I, then we have

$$\frac{t_1 t_2 \cdots t_l}{t_I} = 0,$$

which implies that $t_1 t_2 \cdots t_l$ has no contribution to $Q_{\{s_{i_1}^{(t)}, s_{i_2}^{(t)}, \ldots, s_{i_l}^{(t)}\}}$. Hence an effective $t_1 t_2 \cdots t_l$ should satisfy that $t_1 t_2 \cdots t_l$ is divisible by t_I. To make this point clear we rewrite (9) as

$$Q_{\{s_{i_1}^{(t)}, s_{i_2}^{(t)}, \ldots, s_{i_l}^{(t)}\}} = \bigoplus_{t_I | t_1 t_2 \cdots t_l} \frac{t_1 t_2 \cdots t_l}{t_I}, \tag{10}$$

where t_j runs through $T(s_{i_j}^{(t)})$ independently for $1 \le j \le l$. To reduce the number of effective terms or summation in (10) we propose a criterion for useful cubes in this subsection.

To characterize a useful cube, we shall give some definitions first.

Definition 1. Let I be a set of cube variables, $t \ge 0$, and $1 \le i \le N$. If a term $u \in T(s_i^{(t)})$ satisfies $\deg_I(u) = \deg_I(s_i^{(t)})$, then u is called a maximum degree term of $s_i^{(t)}$ on I.

A maximum degree term of $s_i^{(t)}$ on I is a term whose degree on I attains the maximum. It is obvious that a maximum degree term of $s_i^{(t)}$ is not unique. For example, $I = \{v_1, v_2, v_3, v_4\}$ and $s_i^{(t)} = v_1 v_2 k_1 \oplus v_2 v_3 k_1 k_2 \oplus v_4 k_3$. Then $v_1 v_2 k_1$ and $v_2 v_3 k_1 k_2$ are maximum degree terms of $s_i^{(t)}$ whose degrees on I are 2.

Definition 2. Let I be a set of cube variables and $t \ge 0$. For a term $u = \prod_{j=1}^{l} s_{i_j}^{(t)}$, if

$$\sum_{j=1}^{l} \deg_I(s_{i_j}^{(t)}) = |I|,$$

then u is called a tight term for I.

The following property gives a relationship between tight terms and maximum degree terms concerning the right hand side of (10).

Property 1. Let I be a set of cube variables and $u = s_{i_1}^{(t)} s_{i_2}^{(t)} \cdots s_{i_l}^{(t)}$ be a tight term for I. If $t_1 t_2 \cdots t_l$ is divisible by t_I where $t_j \in T(s_{i_j}^{(t)})$ for $1 \le j \le l$, then t_j is a maximum degree term of $s_{i_j}^{(t)}$ on I.

Let $s_{i_1}^{(t)} s_{i_2}^{(t)} \cdots s_{i_l}^{(t)}$ be a tight term for I. Due to Property 1, when computing $\frac{s_{i_1}^{(t)} s_{i_2}^{(t)} \cdots s_{i_l}^{(t)}}{t_I}$, we only need to consider maximum degree terms of $s_{i_j}^{(t)}$ on I. Moreover, maximum degree terms are usually a very small part of $s_{i_j}^{(t)}$. Thus, in this case the computation of $\frac{s_{i_1}^{(t)} s_{i_2}^{(t)} \cdots s_{i_l}^{(t)}}{t_I}$ could be simplified greatly.

Example 1. Let $I = \{v_0, v_1, v_2, v_3\}$, $s_1 = v_0 v_1 k_0 \oplus v_2 \oplus v_3$, $s_2 = v_2 v_3 k_5 \oplus v_0 \oplus v_2$, $s_3 = v_2 \oplus v_3$, $u_1 = s_1 s_2$, and $u_2 = s_2 s_3$. Then it can be seen that u_1 is a tight term for I. However, u_2 is not a tight term for I. The sets of maximum degree terms of s_1 and s_2 are $\{v_0 v_1 k_0\}$ and $\{v_2 v_3 k_5\}$, respectively. According to Property 1, we have

$$\frac{u_1}{t_I} = k_0 k_5,$$

which could be computed only using the maximum degree terms of s_1 and s_2.

Based on the concept of tight terms, we propose a new criterion of useful cubes.

Criterion 1. *Let I be a set of cube variables and $z = g_t(s_1^{(t)}, s_2^{(t)}, \ldots, s_N^{(t)})$. If every term $u = s_{i_1}^{(t)} s_{i_2}^{(t)} \cdots s_{i_l}^{(t)} \in T(g_t)$ satisfies*

$$\sum_{j=1}^{l} \deg_I(s_{i_j}^{(t)}) \le |I|, \tag{11}$$

then I is called a useful cube.

In Criterion 1, if $\sum_{j=1}^{l} \deg_I(s_{i_j}^{(t)}) = |I|$, then $s_{i_1}^{(t)} s_{i_2}^{(t)} \cdots s_{i_l}^{(t)}$ is a tight term of g_t for I; otherwise we have

$$\sum_{j=1}^{l} \deg_I(s_{i_j}^{(t)}) < |I|$$

which implies that $s_{i_1}^{(t)} s_{i_2}^{(t)} \cdots s_{i_l}^{(t)}$ is not divisible by t_I, and so

$$\frac{s_{i_1}^{(t)} s_{i_2}^{(t)} \cdots s_{i_l}^{(t)}}{t_I} = 0.$$

Therefore, this criterion implies that every term u of g_t is either a tight term or $\frac{u}{t_I} = 0$. Accordingly for a useful cube I, we simply have

$$\begin{aligned} p_I &= \bigoplus_{\substack{s_{i_1}^{(t)} s_{i_2}^{(t)} \cdots s_{i_l}^{(t)} \\ \text{is a tight term of } g_t}} Q_{\{s_{i_1}^{(t)}, s_{i_2}^{(t)}, \ldots, s_{i_l}^{(t)}\}} \\ &= \bigoplus_{\substack{s_{i_1}^{(t)} s_{i_2}^{(t)} \cdots s_{i_l}^{(t)} \\ \text{is a tight term of } g_t}} \left(\bigoplus_{\substack{t_j \text{ is a maximum} \\ \text{degree term of } s_{i_j}^{(t)}}} \frac{t_1 t_2 \cdots t_l}{t_I} \right). \end{aligned} \tag{12}$$

It can be seen that the computation of p_I is relatively easier for a useful cube I. In the next subsection, we will show how to pick up useful cubes.

Algorithm 2 Finding Useful Cubes with Degree Evaluation

Require: the chosen cube variables I, the chosen time instance t

1: Express the output bit z as $z = g_t(s^{(t)})$;
2: Iteratively calculate $\mathrm{DEG}_I(s_i^{(t)})$ for $i \in \{1, 2, \ldots, N\}$;
3: **for** each term $u = s_{i_1}^{(t)} s_{i_2}^{(t)} \cdots s_{i_l}^{(t)}$ of g_t **do**
4: Set $\mathrm{DEG}_I(u) = \sum_{j=1}^{l} \mathrm{DEG}_I(s_{i_j}^{(t)})$;
5: **if** $\mathrm{DEG}_I(u) > |I|$ **then**
6: **return** useless;
7: **end if**
8: **end for**
9: **return** useful;

3.3 An Algorithm to Find Useful Cubes

In this subsection, we discuss how to find useful cubes efficiently. According to Rule 1, we assume that g_t is known, and so $T(g_t)$ is known. It can be seen from Criterion 1 that to judge whether I is useful we need to calculate

$$\sum_{j=1}^{l} \deg_I(s_{i_j}^{(t)})$$

for every term $s_{i_1}^{(t)} s_{i_2}^{(t)} \cdots s_{i_l}^{(t)}$ in $T(g_t)$. This is in essence a degree evaluation problem. On one hand, to quickly judge whether a cube is useful, we need an efficient degree evaluation algorithm. On the other hand, to accurately identify a useful cube, we need an accurate degree evaluation algorithm since a useful cube may be missed if the estimated degrees are far from real degrees. Considering these issues, we choose to use the idea of numeric mapping proposed in [16]. Details are given in Algorithm 2. As for the definition and methodology of numeric mapping and numeric degree please refer to [16] and Section 2.

The general idea of Algorithm 2 is first computing the numeric degree of $s_i^{(t)}$ on I for $1 \leq i \leq N$ denoted by $\mathrm{DEG}_I(s_i^{(t)})$ and then computing the numeric degree for each term $s_{i_1}^{(t)} s_{i_2}^{(t)} \cdots s_{i_l}^{(t)} \in T(g_t)$ by

$$\mathrm{DEG}_I(s_{i_1}^{(t)} s_{i_2}^{(t)} \cdots s_{i_l}^{(t)}) = \sum_{j=1}^{l} \mathrm{DEG}_I(s_{i_j}^{(t)}).$$

If

$$\mathrm{DEG}_I(s_{i_1}^{(t)} s_{i_2}^{(t)} \cdots s_{i_l}^{(t)}) \leq |I|$$

holds for every term $s_{i_1}^{(t)} s_{i_2}^{(t)} \cdots s_{i_l}^{(t)} \in T(g_t)$, then we regard I as a useful cube in Algorithm 2. Since the algebraic degree of $s_i^{(t)}$ is always less than or equal to the numeric degree of $s_i^{(t)}$, i.e., $\deg_I(s_i^{(t)}) \leq \mathrm{DEG}_I(s_i^{(t)})$, it follows that a cube outputted by Algorithm 2 satisfies Criterion 1.

3.4 Recover the Exact Superpoly of a Useful Cube

After finding a useful cube I, we can recover the corresponding superpoly p_I by (12). The critical part of this phase is calculating the superpoly of t_I in each tight term for I. We present the details in Algorithms 3 and 4.

Let $u = s_{i_1}^{(t)} s_{i_2}^{(t)} \cdots s_{i_l}^{(t)}$ be a tight term for I. In Algorithm 3, the procedure **Recover-Coefficient** is called to compute $Q_{\{s_{i_1}^{(t)}, s_{i_2}^{(t)}, \ldots, s_{i_l}^{(t)}\}}$. Following from (12), $Q_{\{s_{i_1}^{(t)}, s_{i_2}^{(t)}, \ldots, s_{i_l}^{(t)}\}}$

is the summation of

$$\frac{t_1 t_2 \cdots t_l}{t_I},$$

where $t_1 t_2 \cdots t_l$ is divisible by t_I and t_j is a maximum degree term of $s_{i_j}^{(t)}$ for $j \in \{1, 2, \ldots, l\}$. Hence, we need to find all such products of maximum degree terms of $s_{i_1}^{(t)}, s_{i_2}^{(t)}, \ldots, s_{i_l}^{(t)}$ to obtain $Q_{\{s_{i_1}^{(t)}, s_{i_2}^{(t)}, \ldots, s_{i_l}^{(t)}\}}$. In **RecoverCoefficient**, it can be done in the following three steps.

Collect and Preprocess the Maximum Degree Terms. The first step is to collect and preprocess the maximum degree terms of $s_{i_j}^{(t)}$ for $j \in \{1, 2, \ldots, l\}$. Assume that the maximum degree terms of $s_{i_j}^{(t)}$ are stored in $MDT[j]$, where MDT is a list of sets and $MDT[j]$ represents the j-th set in MDT.

Our goal is to find all the combinations

$$(t_1, t_2, \ldots, t_l) \in MDT[1] \times MDT[2] \times \cdots \times MDT[l]$$

such that $\prod_{j=1}^{l} t_j$ is divisible by t_I. Let t'_j be the IV representation of t_j for $1 \le j \le l$. Then, if $\prod_{j=1}^{l} t_j$ is divisible by t_I, then $\prod_{j=1}^{l} t'_j = t_I$ (the IV variables except cube variables are set to 0). Therefore, we apply the *Reduce* operation to $MDT[j]$ for $1 \le j \le l$. In the *Reduce* operation, we first do IV representation for each term in $MDT[j]$, and so we could obtain a multi-set $VMDT[j] = \{u_{IV} | u \in MDT[j]\}$, where u_{IV} is the IV representation of u. Then, we could obtain a set $RMDT[j]$ from $VMDT[j]$, where only one of the repeated terms in $VMDT[j]$ are kept. For simplicity, $RMDT[j]$ is called a set of reduced maximum degree terms in the rest of this paper.

In this paper, a combination

$$(t_{j_1}^1, t_{j_2}^2, \ldots, t_{j_l}^l) \in RMDT[1] \times RMDT[2] \times \cdots \times RMDT[l]$$

satisfying $\prod_{i=1}^{l} t_{j_i}^i = t_I$ is called a valid combination. It can be seen that, by finding all the valid combinations, we could deduce all the combinations

$$(t_1, t_2, \ldots, t_l) \in MDT[1] \times MDT[2] \times \cdots \times MDT[l]$$

such that $\prod_{i=1}^{l} t_i$ is divisible by t_I. Note that $\prod_{i=1}^{l} |RMDT[i]|$ would be much smaller than $\prod_{i=1}^{l} |MDT[i]|$, since $MDT[i]$ $(1 \le i \le l)$ may have many terms whose results of IV representation are the same. Thus, the complexity could be reduced dramatically.

Find All the Valid Combinations. Accordingly, the second step is to find all the valid combinations. Although $\prod_{i=1}^{l} |RMDT[i]|$ would be much smaller than $\prod_{i=1}^{l} |MDT[i]|$, it may still be very large. Hence, we would not check each combination $(t_{j_1}^1, t_{j_2}^2, \ldots, t_{j_l}^l)$ directly, where $t_{j_i}^i \in RMDT[i]$ for $1 \le i \le l$. Instead, we pick up elements from $RMDT$ gradually to form a full combination. Moreover, we propose the following two strategies to throw away some invalid combinations in advance. To illustrate these two strategies, assume that we have picked up the first d elements of a combination, i.e., $t_{j_1}^1, t_{j_2}^2, \ldots, t_{j_d}^d$.

Strategy 1. If $\deg_I(t_{j_1}^1 \cdots t_{j_d}^d) < \deg_I(t_{j_1}^1) + \cdots + \deg_I(t_{j_d}^d)$, then we would throw away all the combinations whose first d components are $t_{j_1}^1, t_{j_2}^2, \ldots, t_{j_d}^d$.

Let $(t_{j_1}^1, t_{j_2}^2, \ldots, t_{j_d}^d, t_{j_{d+1}}^{d+1}, \ldots, t_{j_l}^l)$ be a combination such that the condition in Strategy 1 is satisfied. Then,

$$\begin{aligned}
\deg_I(t_{j_1}^1 \cdots t_{j_l}^l) &\le \deg_I(t_{j_1}^1 \cdots t_{j_d}^d) + \deg_I(t_{j_{d+1}}^{d+1} \cdots t_{j_l}^l) \\
&< \sum_{i=1}^{d} \deg_I(t_{j_i}^i) + \deg_I(t_{j_{d+1}}^{d+1} \cdots t_{j_l}^l) \\
&\le \sum_{i=1}^{l} \deg_I(t_{j_i}^i) = |I|.
\end{aligned}$$

Namely, $\deg_I(t_{j_1}^1 t_{j_2}^2 \cdots t_{j_l}^l) < |I|$. Hence, combinations satisfying the condition in Strategy 1 are not valid ones and should be thrown away.

Strategy 2. For some $d+1 \leq w \leq l$, if each term $t_w \in RMDT[w]$ satisfies that $\deg_I(t_w \cdot t_{j_1}^1 t_{j_2}^2 \cdots t_{j_d}^d) < \deg_I(t_w) + \deg_I(t_{j_1}^1 t_{j_2}^2 \cdots t_{j_d}^d)$, then we would throw away all the combinations whose first d components are $t_{j_1}^1, t_{j_2}^2, \ldots, t_{j_d}^d$.

Let $(t_{j_1}^1, t_{j_2}^2, \ldots, t_{j_d}^d, t_{j_{d+1}}^{d+1}, \ldots, t_{j_l}^l)$ be a combination such that the condition in Strategy 2 is satisfied. Without loss of generality, we assume that $w = d+1$. Then,

$$\begin{aligned}
\deg_I(t_{j_1}^1 \cdots t_{j_l}^l) &\leq \deg_I(t_{j_1}^1 \cdots t_{j_{d+1}}^{d+1}) + \deg_I(t_{j_{d+2}}^{d+2} \cdots t_{j_l}^l) \\
&< \deg_I(t_{j_1}^1 \cdots t_{j_d}^d) + \deg_I(t_{j_{d+1}}^{d+1}) \\
&\quad + \deg_I(t_{j_{d+2}}^{d+2} \cdots t_{j_l}^l) \\
&\leq \sum_{i=1}^{l} \deg_I(t_{j_i}^i) = |I|.
\end{aligned}$$

Namely, $\deg_I(t_{j_1}^1 t_{j_2}^2 \cdots t_{j_l}^l) < |I|$. Hence, combinations satisfying the condition in Strategy 2 are not valid ones and should be thrown away.

If the chosen first d components do not satisfy the condition in Strategy 1 nor the condition in Strategy 2, then we would pick up the $(d+1)$-th component from $RMDT[d+1]$. Note that, Strategies 1 and 2 can be applied again to judge whether the combinations which contain the chosen first $d+1$ components should be thrown away. Namely, Strategies 1 and 2 can be used again and again until a full combination is formed. Benefited from these two strategies, we can throw away many combinations in advance and the phase of finding all the valid combinations can be accelerated dramatically.

Recover the Superpoly of I in a Tight Term u. The final step is to recover the superpoly of I in u according to all the valid combinations. Let $(t_{j_1}^1, t_{j_2}^2, \ldots, t_{j_l}^l)$ be a valid combination. Since the terms in $RMDT[i]$ are reduced from $MDT[i]$, the combination $(t_{j_1}^1, t_{j_2}^2, \ldots, t_{j_l}^l)$ may correspond to several combinations $(t_1, t_2, \ldots, t_l)$ such that $\prod_{j=1}^{l} t_j$ is divisible by t_I, where $t_j \in MDT[j]$ for $1 \leq j \leq l$. All the combinations which $(t_{j_1}^1, t_{j_2}^2, \ldots, t_{j_l}^l)$ corresponds to can be covered by a vector $(\lambda_{j_1}^1, \lambda_{j_2}^2, \ldots, \lambda_{j_l}^l)$, where

$$\lambda_{j_w}^w = \frac{s_{i_w}^{(t)}}{t_{j_w}^w}$$

for $w \in \{1, 2, \ldots, l\}$. In this paper, $(\lambda_{j_1}^1, \lambda_{j_2}^2, \ldots, \lambda_{j_l}^l)$ is called the superpoly vector of $(t_{j_1}^1, t_{j_2}^2, \ldots, t_{j_l}^l)$. Then, the contribution of the valid combination $(t_{j_1}^1, t_{j_2}^2, \ldots, t_{j_l}^l)$ to the superpoly of I in u is $\prod_{w=1}^{l} \lambda_{j_w}^w$. Thus,

$$Q_{\{s_{i_1}^{(t)}, s_{i_2}^{(t)}, \ldots, s_{i_l}^{(t)}\}} = \bigoplus_{(t_{j_1}^1, t_{j_2}^2, \ldots, t_{j_l}^l) \text{is valid}} \prod_{w=1}^{l} \lambda_{j_w}^w.$$

As an illustration of Algorithms 3 and 4, we provide the following example.

Example 2. Let $I = \{v_0, v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8, v_9\}$ be a set of cube variables. Assume that $u = s_1 s_4 s_6 s_8$, where

$$s_1 = v_4 v_5 k_2 k_3 \oplus v_4 v_5 k_4 \oplus v_4 v_5 k_5 \oplus v_2 v_3 \oplus v_5 v_6 \oplus k_3,$$

$$\begin{aligned}
s_4 =& v_0 v_1 v_2 v_3 k_0 \oplus v_0 v_1 v_2 v_3 k_1 \oplus v_0 v_1 v_3 v_4 k_0 \oplus v_0 v_1 v_4 v_6 k_2 \\
&\oplus v_1 v_2 v_3 k_2 \oplus v_1 v_2 v_3 k_{44} \oplus v_2 v_3 \oplus v_4 k_0 \oplus v_5 k_1,
\end{aligned}$$

Algorithm 3 Recover the Exact Superpoly of a Useful Cube

Require: The set of cube variables I, the chosen time instance t to compute g_t

1: Call Algorithm 2, and store the tight terms in the set $\mathbb{T}(I)$;
2: Calculate the ANF of $s_i^{(t)}$ on cube and key variables for $i \in \{1, 2, \ldots, N\}$;
3: Set $p_I = 0$;
4: **for** each $u = s_{i_1}^{(t)} s_{i_2}^{(t)} \cdots s_{i_l}^{(t)} \in \mathbb{T}(I)$ **do**
5: Set $Q_{\{s_{i_1}^{(t)}, s_{i_2}^{(t)}, \ldots, s_{i_l}^{(t)}\}}$ =**RecoverCoefficient**(u,I);
6: Set $p_I = p_I \oplus Q_{\{s_{i_1}^{(t)}, s_{i_2}^{(t)}, \ldots, s_{i_l}^{(t)}\}}$;
7: **end for**
8: **return** p_I;

Algorithm 4 Recover the Superpoly of I in a Tight Term

1: **procedure RecoverCoefficient**(u,I)
2: Assume that $u = s_{i_1}^{(t)} s_{i_2}^{(t)} \cdots s_{i_l}^{(t)}$;
3: **for** $1 \le j \le l$ **do**
4: Collect maximum degree terms of $s_{i_j}^{(t)}$ and store them in $MDT[j]$;
5: Apply the *Reduce* operation to $MDT[j]$ and store the reduced terms in $RMDT[j]$;
6: **end for**
7: Figure out all valid combinations;
8: Set $Q_{\{s_{i_1}^{(t)}, s_{i_2}^{(t)}, \ldots, s_{i_l}^{(t)}\}} = 0$;
9: **for** each valid combination $(t_{j_1}^1, t_{j_2}^2, \ldots, t_{j_l}^l)$ **do**
10: Recover the corresponding superpoly vector $(\lambda_{j_1}^1, \lambda_{j_2}^2, \ldots, \lambda_{j_l}^l)$ according to the ANFs of $s_{i_1}^{(t)}, s_{i_2}^{(t)}, \ldots, s_{i_l}^{(t)}$;
11: Set

$$Q_{\{s_{i_1}^{(t)}, s_{i_2}^{(t)}, \ldots, s_{i_l}^{(t)}\}} = Q_{\{s_{i_1}^{(t)}, s_{i_2}^{(t)}, \ldots, s_{i_l}^{(t)}\}} \oplus \prod_{w=1}^{l} \lambda_{j_w}^w;$$

12: **end for**
13: **return** $Q_{\{s_{i_1}^{(t)}, s_{i_2}^{(t)}, \ldots, s_{i_l}^{(t)}\}}$;
14: **end procedure**

$$s_6 = v_3 v_6 \oplus v_4 v_6 \oplus v_6 v_7, \text{ and } s_8 = v_6 v_9 \oplus v_7 v_9 \oplus v_8 v_9.$$

The first step is to collect the reduced maximum degree terms. It can be obtained that the set of maximum degree terms of s_1 is

$$MDT[1] = \{v_4 v_5 k_2 k_3, v_4 v_5 k_4, v_4 v_5 k_5, v_2 v_3, v_5 v_6\}.$$

Then, the Reduce operation is applied. After applying the IV representation, we could obtain a multi-set

$$VMDT = \{v_4 v_5, v_4 v_5, v_4 v_5, v_2 v_3, v_5 v_6\}.$$

By removing the repeated terms in $VMDT$, we derive the set of reduced maximum degree terms of s_1 is

$$RMDT[1] = \{v_2 v_3, v_4 v_5, v_5 v_6\}.$$

Similarly, the sets of reduced maximum degree terms of s_4,s_6 and s_8 are

$$RMDT[2] = \{v_0 v_1 v_2 v_3, v_0 v_1 v_3 v_4, v_0 v_1 v_4 v_6\},$$

$$RMDT[3] = \{v_3 v_6, v_4 v_6, v_6 v_7\},$$

and

$$RMDT[4] = \{v_6v_9, v_7v_9, v_8v_9\}$$

respectively.

After obtaining the sets of reduced maximum degree terms, we need to find all the valid combinations. Due to the first strategy, we can throw away the combinations whose first two components are in the set $\{(v_2v_3, v_0v_1v_2v_3), (v_2v_3, v_0v_1v_3v_4), (v_4v_5, v_0v_1v_3v_4), (v_4v_5, v_0v_1v_4v_6), (v_5v_6, v_0v_1v_4v_6)\}$. Furthermore, according to the second strategy, we can throw away the combinations whose first two components belong to $\{(v_5v_6, v_0v_1v_2v_3), (v_5v_6, v_0v_1v_3v_4), (v_2v_3, v_0v_1v_4v_6)\}$. Totally, we throw away 8 out of all the 9 combinations for the first two components. We use these two strategies iteratively to form a full combination. As a result, we can obtain the only valid combination $(v_4v_5, v_0v_1v_2v_3, v_6v_7, v_8v_9)$ without checking every combination. The superpoly vector of $(v_4v_5, v_0v_1v_2v_3, v_6v_7, v_8v_9)$ is $(k_2k_3 \oplus k_4 \oplus k_5, k_0 \oplus k_1, 1, 1)$. Immediately, we have that

$$Q_{(s_1,s_4,s_6,s_8)} = (k_2k_3 \oplus k_4 \oplus k_5)(k_0 \oplus k_1).$$

4 Applications to Trivium

In this section, we apply our method to Trivium. First, we introduce some details of applications to Trivium. Then, we perform experiments on several variants of round-reduced Trivium. Finally, we have some discussion on our method.

4.1 The Optimization for Applications to Trivium

In this subsection, according to the structure of Trivium, we do some optimization for Algorithms 2, 3 and 4. For the sake of convenience, we assume that the r-round Trivium is our target and the output bit z_r is presented by $g_t(s^{(t)})$ for some properly chosen t, i.e., $z_r(\boldsymbol{key}, \boldsymbol{iv}) = g_t(s_1^{(t)}(\boldsymbol{key}, \boldsymbol{iv}), \cdots, s_{288}^{(t)}(\boldsymbol{key}, \boldsymbol{iv}))$.

4.1.1 The Algorithm of Finding Useful Cubes for Trivium

In order to identify useful cubes more accurately, we make some optimization and improvements to Algorithm 2 according to the structure of Trivium.

Treating Two Adjacent Internal State Bits as a Whole. Let $u = s_{i_1}^{(t)}s_{i_2}^{(t)}\cdots s_{i_l}^{(t)}$ be a term of g_t. When judging whether u is a tight term, if u has two adjacent internal state bits in the same register, i.e., $s_j^{(t)}$ and $s_{j+1}^{(t)}$ for some j, then we would treat the product of these two bits as a whole. Namely, we evaluate the degree of the product of two adjacent bits instead of estimating the degrees of these two bits separately. Liu also did so in [16]. In the rest of this paper, we refer to two adjacent internal state bits in the same register as two adjacent internal state bits for short. The following is an illustrative example.

Example 3. Let $u = s_1^{(t)}s_2^{(t)}s_3^{(t)}s_4^{(t)}s_5^{(t)}$ be a term of g_t. The degree of u is evaluated as $\mathrm{DEG}_I(u) = \mathrm{DEG}_I(s_1^{(t)}s_2^{(t)}) + \mathrm{DEG}_I(s_3^{(t)}s_4^{(t)}) + \mathrm{DEG}_I(s_5^{(t)})$.

A Small Improvement. When evaluating the degree of $s_j^{(t)}s_{j+1}^{(t)}$, we make a small improvement of the degree evaluation method proposed by Liu in [16]. We take $s_{91}^{(t)}s_{92}^{(t)}$ as an example to illustrate this improvement. For any $t \geq 92$, $s_{91}^{(t)}$ and $s_{92}^{(t)}$ can be recursively represented by

$$s_{91}^{(t)} = s_{286}^{t-91}s_{287}^{t-91} \oplus s_{288}^{t-91} \oplus s_{243}^{t-91} \oplus s_{69}^{t-91}$$

and

$$s_{92}^{(t)} = s_{286}^{t-92}s_{287}^{t-92} \oplus s_{288}^{t-92} \oplus s_{243}^{t-92} \oplus s_{69}^{t-92},$$

respectively. Since $s_{287}^{t-92} = s_{288}^{t-91}$, we evaluate the degree of $s_{286}^{t-92}s_{287}^{t-92}(s_{288}^{t-91} \oplus s_{243}^{t-91} \oplus s_{69}^{t-91})$ as

$$\mathrm{DEG}_I(s_{286}^{t-92}s_{287}^{t-92}) + \mathrm{DEG}_I(s_{243}^{t-91} \oplus s_{69}^{t-91})$$

instead of

$$\mathrm{DEG}_I(s_{286}^{t-92}s_{287}^{t-92}) + \mathrm{DEG}_I(s_{288}^{t-91} \oplus s_{243}^{t-91} \oplus s_{69}^{t-91})$$

as Liu did in [16]. When the degree of $s_{288}^{t-91} \oplus s_{243}^{t-91} \oplus s_{69}^{t-91}$ is determined by s_{288}^{t-91}, our improvement would work. Similarly, this improvement could also be made in the cases of $s_{175}^{(t)}s_{176}^{(t)}$ and $s_{286}^{(t)}s_{287}^{(t)}$, since the update functions of three registers are similar. Based on the above optimization and improvements, we propose a more accurate algorithm of finding useful cubes for Trivium, see Algorithm 5 in Appendix 1.

Coincident with Algorithm 5, when recovering the superpoly of t_I in a tight term u, for two adjacent bits $s_j^{(t)}$ and $s_{j+1}^{(t)}$ in u, we collect the reduced maximum degree terms of $s_j^{(t)}s_{j+1}^{(t)}$ instead of collecting the reduced maximum degree terms of $s_j^{(t)}$ and $s_{j+1}^{(t)}$ separately. The detailed procedure is described in Algorithm 7 in Appendix 1.

4.1.2 Recovering Superpolies for Trivium Variants with High Initialization Rounds.

As r (the target round) increases, it is hard to choose t satisfying Rules 1 and 2 mentioned in Subsection 3.1 at the same time. Fortunately, this dilemma can be solved by taking an extra step when calculating the superpoly p_I of the chosen cube I. Following from (12), we have that

$$\begin{aligned} p_I &= \bigoplus_{\substack{s_{i_1}^{(t)}s_{i_2}^{(t)}\cdots s_{i_l}^{(t)} \\ \text{is a tight term of } g_t}} Q_{\{s_{i_1}^{(t)},s_{i_2}^{(t)},\ldots,s_{i_l}^{(t)}\}} \\ &= \bigoplus_{\substack{u=s_{i_1}^{(t)}\cdots s_{i_l}^{(t)}\text{is} \\ \text{a tight term of } g_t}} \bigoplus_{\substack{s_{j_1}^{(t_0)}\cdots s_{j_d}^{(t_0)} \text{ is a} \\ \text{term in } T(f_{t_0}^u)}} Q_{\{s_{j_1}^{(t_0)},\ldots,s_{j_d}^{(t_0)}\}}, \end{aligned} \tag{13}$$

where

$$u = \prod_{j=1}^{l} s_{i_j}^{(t)}(\boldsymbol{key}, \boldsymbol{iv}) = f_{t_0}^u(s_{j_1}^{(t_0)}(\boldsymbol{key}, \boldsymbol{iv}), \ldots, s_{j_d}^{(t_0)}(\boldsymbol{key}, \boldsymbol{iv})).$$

According to (13), when calculating the superpoly of I in the tight term $u = s_{i_1}^{(t)}s_{i_2}^{(t)}\cdots s_{i_l}^{(t)}$ of g_t, we first express it as a polynomial on the internal state $s^{(t_0)}$, which is denoted by $f_{t_0}^u$, and then calculate the superpoly of I in each term of $f_{t_0}^u$. The detailed procedure is presented in Algorithm 6 in Appendix 1. Note that, in Algorithm 6, we choose the smallest t of those satisfying **Rule** 1 and the largest t_0 such that we can compute the ANFs of $s_i^{(t_0)}$ for $1 \le i \le 288$.

4.2 Experimental Results

In this subsection, to illustrate the efficiency and effectiveness of our method, we perform various experiments on several round-reduced variants of Trivium. All of our experiments are completed on a PC with an i7-7700k CPU inside.

4.2.1 Towards Finding Useful Cubes.

We try to find useful cubes for Trivium variants with 800-832 initialization rounds. For these variants, similar to [16], we choose cubes which do not contain variables v_i, v_j satisfying $|i-j| = 1$ of sizes 33-36. For each variant with r initialization rounds ($800 \le r \le 832$),

we randomly test 10000 such cubes. As a result, we find useful cubes for each variant, and the average number is about 175. This indicates that useful cubes exist widely and can be found easily.

4.2.2 Results for the 818-round Trivium.

In this subsection, we apply our method to attack the 818-round Trivium. In the following, we would show the details of recovering the superpoly of useful cube by taking

$$\begin{aligned} I \quad = \quad & \{v_1, v_3, v_6, v_8, v_{10}, v_{12}, v_{14}, v_{16}, v_{21}, v_{23}, v_{25}, v_{27}, v_{29}, \\ & v_{31}, v_{34}, v_{36}, v_{38}, v_{40}, v_{42}, v_{44}, v_{49}, v_{51}, v_{53}, v_{55}, v_{57}, \\ & v_{59}, v_{62}, v_{64}, v_{66}, v_{68}, v_{70}, v_{72}, v_{74}, v_{77}, v_{79}\} \end{aligned}$$

as an example. First, we filter out all the 24 tight terms for I of g_{417}. Then, we call Algorithm 6 to calculate the superpoly p_I. In Algorithm 6, to recover the superpoly of I in each tight term u of g_{417}, we first express it as a polynomial on the internal state $s^{(363)}$, denoted by $f^u_{363}(s^{(363)})$, and then calculate the superpoly of I in each term of $f^u_{363}(s^{(363)})$. Hence, we only need to calculate the exact ANFs of $s^{(363)}_1, s^{(363)}_2, \ldots, s^{(363)}_{288}$. After expressing u as f^u_{363}, the key point is to find all the valid combinations in each tight term u' of f^u_{363}. This could be done efficiently with the help of the two strategies proposed in Subsection 3.4. For instance, let u' be a tight term of f^u_{363} given by

$$u' = ds^{(363)}_{64} ds^{(363)}_{102} s^{(363)}_{124} s^{(363)}_{133} ds^{(363)}_{136} s^{(363)}_{145} ds^{(363)}_{147} ds^{(363)}_{154},$$

where $ds^{(363)}_i = s^{(363)}_i s^{(363)}_{i+1}$ and

$$u = s^{(417)}_{121} s^{(417)}_{122} s^{(417)}_{157} s^{(417)}_{158} s^{(417)}_{193} s^{(417)}_{194} s^{(417)}_{203} s^{(417)}_{204} s^{(417)}_{211}.$$

Note that there are totally

$$94 \times 99 \times 52 \times 42 \times 755 \times 34 \times 676 \times 542 \approx 2^{57}$$

combinations in u' (two adjacent bits are treated as a whole). It is not easy for a PC to run over all the 2^{57} combinations. Fortunately, benefited from the two strategies introduced in Subsection 3.4, we can figure out all the valid combinations in u' in seconds with our PC. Then, we recover the superpoly vector for each valid combination. Finally, we obtain the superpoly p_I within about ten minutes, see Table 2 for details.

For the 818-round Trivium, among the found useful cubes, we recover the exact superpolies of those with relatively few tight terms. For each recovered superpoly p_I, it could be rewritten as a product of some simple polynomials on key variables, i.e., $p_I = \prod_{g \in \Gamma_I} g(\boldsymbol{key})$. When $p_I = 1$, we have that $g(\boldsymbol{key}) = 1$ for each $g \in \Gamma_I$. On the other hand, when $p_I = 0$, by checking whether the value of p_I under a specific key is equal to 0, we could still discard a large amount of wrong keys. Let us take I_1 as an example. If the superpoly $p_{I_1} = 1$, then we have that

$$k_{65}, g_5, g_{14}, g_{18}, g_{24}, g_{29}, g_{31}, g_{38}, g_{40}, g_{44}, g_{48}, \text{ and } g_{51}$$

are equal to 1, i.e., we could recover 12 key bits equivalently. If $p_{I_1} = 0$, then we could discard 2^{68} wrong keys.

Let $\mathrm{WK}_I = \{\boldsymbol{k} | \boldsymbol{k} \in \mathbb{F}_2^{80}, p_I(\boldsymbol{k}) = 1\}$. It can be seen that WK_I is exactly the set of keys under which we could recover several key variables with p_I. Namely, with respect to p_I, these keys could be recovered more easily, which are called weak keys in this paper. For each superpoly p_I, by calculating the Grobner basis of the ideal generated by the set Γ_I, where $p_I = \prod_{g \in \Gamma_I} g$, we figure out the corresponding set WK_I exactly. We summarize our

results in Table 2[1], where DE_I is the set of equations derived from the superpoly under weak keys. Since the sizes of I_1, I_2, I_3, and I_4 are 35, for the 818-round Trivium, we could recover at least 10 key bits equivalently under about 2^{70} weak keys with a complexity of 2^{37}.

Table 2: Some superpolies for the 818-round Trivium

cube	superpoly	$\lvert\mathrm{WK}_I\rvert$	$\lvert\mathrm{DE}_I\rvert$
I_1	$p_{I_1} = k_{65} \cdot g_5 \cdot g_{14} \cdot g_{18} \cdot g_{24} \cdot g_{29} \cdot g_{31} \cdot g_{38} \cdot g_{40} \cdot g_{44} \cdot g_{48} \cdot g_{51}$	2^{68}	12
I_2	$p_{I_2} = k_{65} \cdot g_5 \cdot g_{18} \cdot g_{27} \cdot g_{29} \cdot g_{33} \cdot g_{38} \cdot g_{42} \cdot (g_{50} \oplus 1) \cdot (f_{22} \oplus 1)$	2^{70}	10
I_3	$p_{I_3} = k_{57} \cdot k_{65} \cdot g_5 \cdot g_{16} \cdot g_{23} \cdot g_{27} \cdot g_{29} \cdot g_{31} \cdot g_{38} \cdot g_{40} \cdot g_{42} \cdot g_{46} \cdot g_{48} \cdot (f_{22} \oplus 1)$	2^{66}	14
I_4	$p_{I_4} = k_{57} \cdot k_{64} \cdot g_5 \cdot g_{16} \cdot g_{27} \cdot g_{29} \cdot g_{31} \cdot g_{38} \cdot g_{40} \cdot g_{42} \cdot g_{46} \cdot g_{48} \cdot (f_{22} \oplus 1)$	2^{67}	13
$f_i = k_i k_{i+1} \oplus k_{i+2} \oplus k_{i+44} \oplus k_{i+53}$ for $1 \le i \le 12$			
$f_i = k_i k_{i+1} \oplus k_{i+2} \oplus k_{i+44}$ for $13 \le i \le 24$			
$g_i = k_i \oplus k_{i+25} k_{i+26} \oplus k_{i+27}$ for $0 \le i \le 52$			
$g_{53} = k_{53} \oplus k_{78} k_{79}$			

4.2.3 Results for the 835-round Trivium.

For the 835-round Trivium, we find several useful cubes and recover their superpolies[2]. Since the recovered superpolies are complex, we could not rewrite them as products of simple polynomials. Hence, we attempt to utilize the relationship between the superpolies and some simple polynomials to perform attacks.

Let $\Omega = \{f_i | 1 \le i \le 24\} \cup \{g_i | 0 \le i \le 53\} \cup \{k_i | 0 \le i \le 79\}$, where f_i's and g_i's are defined as in Table 2. For each superpoly p_I, we check whether $(g \oplus 1) \cdot p_I = 0$ or $g \cdot p_I = 0$ holds, and so we could obtain a corresponding set

$$G_I = \{g | (g \oplus 1) \cdot p_I = 0\}.$$

Note that, when $p_I = 1$, we have that $g = 1$ for each $g \in G_I$. Namely, with the set G_I, we could derive simple equations on key variables under weak keys. Then, for each superpoly p_I, we estimate the size of WK_I by evaluating the values of each superpoly under 2^{20} random keys. For example, we have that $G_{I_5} = \{g_8, g_{21}, g_{32}, g_{34}, g_{40}, g_{49}, f_{22} \oplus 1\}$. Hence, we could obtain eight equations on key variables when $p_{I_5} = 1$. We summarize our results in Table 3[1], where DE_I is the set of equations derived from the superpoly p_I under weak keys.

Table 3: superpolies of 835-round Trivium

cube	G_I	$\lvert\mathrm{WK}_I\rvert$	$\lvert\mathrm{DE}_I\rvert$
I_5	$g_8, g_{21}, g_{32}, g_{34}, g_{40}, g_{49}, f_{22} \oplus 1$	$2^{69.6}$	8
I_6	g_8, g_{32}, g_{40}	$2^{71.3}$	3
I_7	$g_{17}, g_{23}, g_{26}, g_{32}, g_{34}, g_{49}, f_5 \oplus 1, f_{14}, f_{23} \oplus 1, k_{58}, k_{65}, k_{66} \oplus 1$	2^{67}	12
I_8	$g_9, g_{24}, g_{32}, f_{23} \oplus 1, k_{58}$	$2^{70.32}$	5

Since the sizes of I_5, I_6, I_7, and I_8 are 37, for the 835-round Trivium, we could recover at least 12 key bits equivalently on more than 2^{67} weak keys with a complexity of 2^{39}.

4.2.4 Results for the 837-round and 838-round Trivium

We do similar experiments on the 837- and 838-round Trivium. We find several useful cubes, and we list a part of them in Table 6 in Appendix 2. Among these useful cubes, we recover the exact superpolies of those with fewest tight terms[2]. For each recovered superpoly p_I, we figure out the corresponding set G_I and estimate the size of WK_I with 2^{20} random keys. Furthermore, to recover more key variables, we calculate the conditional probability $\boldsymbol{Pr}(g = 1|p_I = 1)$ for each $g \in \Omega$, and so we could obtain a set

$$PG_I = \{g|g \in \Omega, \boldsymbol{Pr}(g = 1|p_I = 1) \in (0, 0.25] \cup [0.75, 1)\}.$$

When $p_I = 1$, we have that $g = 1$ holds with a probability of $\boldsymbol{Pr}(g = 1|p_I = 1)$ for $g \in PG_I$. Namely, with the set PG_I, we could obtain several probabilistic equations on key variables under weak keys. We summarise our results in Table 4[1].

Table 4: Superpolies of 837- and 838- round Trivium

cube	G_I	PG_I	Pr	$\lvert\mathrm{WK}_I\rvert$
I_9	$g_7, g_{22}, g_{30}, f_{21}, k_{56}$	g_0 g_{13}	0.8788 0.9394	$2^{72.20}$
I_{10}	$f_{20}, g_6, g_{21}, g_{29}, k_{55}$	g_{12} k_{64}	0.8846 0.78	$2^{71.75}$

As a result, for the 837-round Trivium, we could obtain 5 deterministic and 2 probabilistic equations on about $2^{72.2}$ weak keys with a complexity of $2^{37}(|I_9| = 37)$. Similarly, in the case of the 838-round Trivium, we could obtain 5 deterministic and 2 probabilistic equations on about $2^{71.75}$ weak keys with a complexity of 2^{37} $(|I_{10}| = 37)$.

Interestingly, I_{10} is the same as the cube proposed by Liu in [16]. Recall that Liu tested 100 random keys for the superpoly of this cube, the values are always 0. However, after obtaining the ANF of $p_{I_{10}}$ with our method, we know that $p_{I_{10}}$ is not 0-constant. It indicates that the output of the 838-round Trivium achieves the maximum degree 37 over this subset of IV variables, and so the degree given by Liu for this cube is tight. In Table 5, we list several keys under which the values of these two superpolies are 1's, where $\boldsymbol{key} = k_7||k_6 \cdots ||k_0||k_{15}||k_{14} \cdots ||k_8|| \cdots ||k_{79}||k_{78}|| \cdots ||k_{72}$.

Table 5: The found secret keys

cube	***key***	cube	***key***
I_9	0x4fe7af8e2e5e727b31f9	I_{10}	0xffffdfff0f7ff53ff8ff
	0x4fe7af8e2e5e737b31f1		0xffffdfff0f7fc53ff8ff
	0x4ee7af862e5e727b31f9		0xfbcbd7dfd4bfbdbd5cfc
	0x4ee7af8c2e5e727b31f9		0xfbcbd7dfd4bfbdbd3cfc
	0x4ee7af862e5e737b31f1		0xfbcbd75fd5afbdbd7cfc

4.3 Discussions

4.3.1 Extra Benefits of Our Method

In our experiments, we find several cubes whose superpolies become 0, i.e., zero-sum distinguishers, because all the terms are vanished by the xor operations. Such zero-sum

[1]The details of $I_1, I_2, \cdots, I_{10}$ could be found in Table 7 in Appendix 1.

[2]For the detailed ANFs of these found superpolies, please refer to https://github.com/yechendong/Deterministic-Cube-Attacks.

distinguishers could not be detected by Liu's degree evaluation method since this method did not take xor-vanished terms into consideration at all. Hence, this shows that xor-vanished terms could be dealt with in our method to some degree, and so our method could potentially improve degree evaluation method. This will be a future subject of our work.

4.3.2 A More Generalized Criterion

In the above, we introduce a new criterion of useful cubes under which the superpolies can be calculated with a low complexity. Actually, this criterion could be loosen. Assume that $u = s_{i_1}^{(t)} s_{i_2}^{(t)} \cdots s_{i_l}^{(t)}$ satisfies $\sum_{j=1}^{l} deg_I(s_{i_j}^{(t)}) = |I| + 1$ for a set of cube variables I. Then, the superpoly of t_I in u can be calculated by using the maximum degree terms and the sub-maximum degree terms whose degrees are equal to $deg_I(s_{i_j}^{(t)}) - 1$. The detailed procedure is described in the following five steps.

Step 1. Collect the set of reduced maximum degree terms for each bit in $u = \prod_{j=1}^{l} s_{i_j}^{(t)}$ and apply the *Reduce* operation to them. Store the reduced maximum degree terms of $s_{i_j}^{(t)}$ in $RMDT[j]$.

Step 2. Collect the set of reduced sub-maximum degree terms for each bit in $u = \prod_{j=1}^{l} s_{i_j}^{(t)}$ and apply the *Reduce* operation to them. Store the reduced sub-maximum degree terms of $s_{i_j}^{(t)}$ in $SRMDT[j]$.

Step 3. Find all the valid combinations

$$(t_{i_1}^1, t_{i_2}^2, \ldots, t_{i_l}^l) \in RMDT[1] \times RMDT[2] \times \cdots \times RMDT[l]$$

such that $t_{i_1}^1 t_{i_2}^2 \cdots t_{i_l}^l = t_I$. Recover the contribution of all these valid combinations to the superpoly of t_I in u and denote it by λ_1.

Step 4. For $1 \leq j \leq l$, figure out all the possible combinations $(t_{i_1}^1, t_{i_2}^2, \ldots, t_{i_l}^l)$ such that $t_{i_1}^1 t_{i_2}^2 \cdots t_{i_l}^l = t_I$, where $t_{i_w}^w \in RMDT[w]$ for $w \neq j$ and $t_{i_w}^w \in RSMDT[w]$ if $w = j$. Recover the contribution of all these valid combinations to the superpoly of t_I in u and denote it by λ_2.

Step 5. The final superpoly of t_I in u is $\lambda_1 \oplus \lambda_2$.

Therefore, for a cube I, if each term $u = s_{i_1}^{(t)} s_{i_2}^{(t)} \cdots s_{i_l}^{(t)}$ of the g_t satisfies $\sum_{j=1}^{l} deg_I(s_{i_j}^{(t)}) \leq |I| + 1$, then it should be regarded as a useful cube.

5 Conclusion

In this paper, we propose an algebraic method to recover superpolies in cube attacks. First, we introduce a new criterion of useful cubes whose superpolies can be calculated with a low complexity. Then, we design an algorithm which could find useful cubes efficiently. Finally, a new algorithm together with some techniques are proposed to recover the exact superpolies of useful cubes. As an illustration, we apply our attacks to Trivium. In applications to Trivium, we choose cubes which do not contain variables v_i, v_j satisfying $|i - j| = 1$. Then, the sizes of chosen cubes are always not more than 40. With such cubes, we find useful cubes for the 818-,835-,837- and 838-round Trivium. By recovering the superpolies of useful cubes, we establish attacks for up to the 838-round Trivium under a large set of weak keys. However, it seems hard to increase the number of attacking rounds with such kind of cubes. On the other hand, we tested some larger cubes but we did not find useful cubes under our criterion for higher rounds. How to increases the number of attacking rounds still needs further research.

6 Acknowledgments

This work was supported by the National Natural Science Foundations of China under grant nos. 61672533 and 61521003.

References

[1] Dinur, I., Shamir, A.: 'Cube attacks on tweakable black box polynomials'. Proc. Advances in Cryptology - EUROCRYPT 2009, Germany, April 2009, pp. 278–299

[2] Aumasson, J., Dinur, I., Meier, W., Shamir, A.: 'Cube testers and key recovery attacks on reduced-round MD6 and Trivium'. Proc. Fast Software Encryption, 16th Int. Workshop, FSE 2009, Leuven, Belgium, February 2009, pp. 1–22

[3] Dinur, I., Shamir, A.: 'Breaking grain-128 with dynamic cube attacks'. Proc. Fast Software Encryption - 18th Int. Workshop, FSE 2011, Lyngby, Denmark, February 2011, pp. 167–187

[4] Huang, S., Wang, X., Xu, G., Wang, M., Zhao, J.: 'Conditional cube attack on reduced-round keccak sponge function'. Proc. Advances in Cryptology - EUROCRYPT 2017, Paris, France, April 2017, pp. 259–288

[5] Todo, Y., Isobe, T., Hao, Y., Meier, W.: 'Cube attacks on non-blackbox polynomials based on division property'. Proc. Advances in Cryptology - CRYPTO 2017, Santa Barbara, CA, USA, August 2017, pp. 250–279

[6] Wang, Q., Hao, Y., Todo, Y., Li, C., Isobe, T., Meier, W.: 'Improved division property based cube attacks exploiting algebraic properties of superpoly'. Proc. Advances in Cryptology - CRYPTO 2018, Santa Barbara, CA, USA, August 2018, pp. 275–305

[7] Liu, M., Yang, J., Wang, W., Lin, D.: 'Correlation cube attacks: From weak-key distinguisher to key recovery'. Proc. Advances in Cryptology - EUROCRYPT 2018, Tel Aviv, Israel, April 2018, pp. 715–744

[8] Mroczkowski, P., Szmidt, J.: 'Corrigendum to: The cube attack on stream cipher Trivium and quadraticity tests', Cryptology ePrint Archive, Report 2011/032, 2011. Available from: http://eprint.iacr.org/2011/032

[9] Fouque, P., Vannet, T.: 'Improving key recovery to 784 and 799 rounds of Trivium using optimized cube attacks'. Proc. Fast Software Encryption - 20th Int. Workshop, FSE 2013, Singapore, March 2013, pp. 502–517

[10] Ye, C., Tian, T.: 'A new framework for finding nonlinear superpolies in cube attacks against Trivium-like ciphers'. Proc. Information Security and Privacy - 23rd Australasian Conf., ACISP 2018, Wollongong, NSW, Australia, July 2018 (LNCS 10946), pp. 172–187

[11] Wang, S., Hu, B., Guan, J., Zhang, K., Shi, T.: 'A practical method to recover exact superpoly in cube attack'. Cryptology ePrint Archive, Report 2019/259, 2019. Available from: https://eprint.iacr.org/2019/259

[12] Fu, X., Wang, X., Dong, X., Meier, W. 'A key-recovery attack on 855-round Trivium'. Proc. Advances in Cryptology - CRYPTO 2018, Santa Barbara, CA, USA, August, 2018, pp. 160–184

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

[14] Todo, Y., Isobe, T., Hao, Y., Meier, W.: 'Cube attacks on non-blackbox polynomials based on division property', *IEEE Trans Computers*, 2018, **67**, (12), pp. 1720–1736.

[15] Cannière, C.D., Preneel, B.: 'Trivium'. In: New Stream Cipher Designs - The eSTREAM Finalists, 2008. pp. 244–266

[16] Liu, M.: 'Degree evaluation of NFSR-Based cryptosystems'. Proc. Advances in Cryptology - CRYPTO 2017, Santa Barbara, CA, USA, August 2017, pp. 227–249

[17] Fu, X., Wang, X., Chen, J.: 'Determining the nonexistent terms of non-linear multivariate polynomials: How to break Grain-128 more efficiently', Cryptology ePrint Archive, Report 2017/412, 2017. Available from: http://eprint.iacr.org/2017/412

Appendix

1. The Optimization For Trivium

Algorithm 5 Finding Useful Cubes for Trivium

Require: the set of cube variables I, the number of initialization rounds r, the chosen time instance t

Express z_r as $z_r = g_t(s^{(t)})$;
Calculate $\mathrm{DEG}_I(s_i^{(t)})$ for $i \in \{1, 2 \ldots, 288\}$;
Calculate $\mathrm{DEG}_I(s_i^{(t)} s_{i+1}^{(t)})$ for $i \in \{1, 2 \ldots, 288\} \setminus \{93, 177, 288\}$;
for each term $u = s_{i_1}^{(t)} s_{i_2}^{(t)} \cdots s_{i_l}^{(t)}$ of g_t **do**
  Set $\mathrm{DEG}_I(u) = 0$ and $j = 1$;
  while $j \leq l$ **do**
    if $i_j + 1 = i_{j+1}$ and $j < l$ and $i_j \notin \{93, 177, 288\}$ **then**
      Set $\mathrm{DEG}_I(u)$=$\mathrm{DEG}_I(u)$+$\mathrm{DEG}_I(s_{i_j}^{(t)} s_{i_{j+1}}^{(t)})$;
      Set j=j+2;
    else
      Set $\mathrm{DEG}_I(u)$ =$\mathrm{DEG}_I(u)$+$\mathrm{DEG}_I(s_{i_j}^{(t)})$;
      Set j=j+1;
    end if
  end while
  if $\mathrm{DEG}_I(u) > |I|$ **then**
    return useless;
  end if
end for
return useful;

Algorithm 6 Recover the Exact ANF of a Useful Cube for Trivium

Require: the set of cube variables I, the number of initialization rounds r, the chosen time t to compute g_t, the time instant t_0 to compute the ANFs

Call Algorithm 2, and store the tight terms for I of g_t in the set $T(I)$;
Compute the ANF of $s_i^{(t_0)}$ on cube and key variables for $i \in \{1, 2, \ldots, 288\}$;
Set $p_I = 0$;
for each $u = s_{i_1}^{(t)} s_{i_2}^{(t)} \cdots s_{i_l}^{(t)} \in T(I)$ **do**
  Set $p_I^u = 0$;
  Represent u as a polynomial on the internal state bits of $s^{(t_0)}$, i.e., $u = f_{t_0}^u(s^{(t_0)})$;
  for each term $u' = s_{j_1}^{(t_0)} s_{j_2}^{(t_0)} \cdots s_{j_w}^{(t_0)}$ of $f_{t_0}^u$ **do**
    Evaluate the degree of u' using the similar method used in Algorithm 5;
    if $\mathrm{DEG}_I(u') = |I|$ **then**
      Set $Q_{\{s_{j_1}^{(t_0)} s_{j_2}^{(t_0)} \cdots s_{j_w}^{(t_0)}\}}$ =**RecoverCoefficient**(u', I);
      Set $p_I^u = p_I^u \oplus Q_{\{s_{j_1}^{(t_0)} s_{j_2}^{(t_0)} \cdots s_{j_w}^{(t_0)}\}}$;
    end if
  end for
  Set $p_I = p_I \oplus p_I^u$;
end for
return p_I;

Algorithm 7 Recover the Superpoly of t_I in a Tight Term for Trivium

procedure RecoverCoefficient(u,I)
  Assume that $u = s_{j_1}^{(t_0)} s_{j_2}^{(t_0)} \cdots s_{j_w}^{(t_0)}$;
  Rewrite u as $u = h_1 h_2 \cdots h_L$, where h_i is a single internal state bit or the product of two adjacent internal state bits;
  for $1 \le i \le L$ **do**
    Collect and store the maximum degree terms of h_i in $MDT[i]$;
    Apply the *Reduce* operation to $MDT[i]$, and store the reduced terms in $RMDT[i]$;
  end for
  Figure out all valid combinations;
  Set $Q_{\{s_{j_1}^{(t_0)} s_{j_2}^{(t_0)} \cdots s_{j_w}^{(t_0)}\}} = 0$;
  for each valid combination $(t_{i_1}^1, t_{i_2}^2, \ldots, t_{i_L}^L)$ **do**
    Recover the corresponding superpoly vector $(\lambda_{i_1}^1, \lambda_{i_2}^2, \ldots, \lambda_{i_L}^L)$;
    Set $Q_{\{s_{j_1}^{(t_0)} s_{j_2}^{(t_0)} \cdots s_{j_w}^{(t_0)}\}} = Q_{\{s_{j_1}^{(t_0)} s_{j_2}^{(t_0)} \cdots s_{j_w}^{(t_0)}\}} \oplus \prod_{j=1}^{L} \lambda_{i_j}^j$;
  end for
  return $Q_{\{s_{j_1}^{(t_0)} s_{j_2}^{(t_0)} \cdots s_{j_w}^{(t_0)}\}}$;
end procedure

2. A Part of Experimental Results for the 837- and 838-round Trivium

Table 6: A part of useful cubes of the 837- and 838-round Trivium

rounds	indies of cube variables	tight terms
837	1,3,5,7,9,11,13,16,18,20,22,24,26,28,31,33,35,37,39,41, 43,46,48,50,52,54,56,58,61,63,65,67,69,71,73,76,78	6
	2,4,6,8,10,12,14,17,19,21,23,25,27,29,32,34,36,38,40,42, 44,47,49,51,53,55,57,59,62,64,66,68,70,72,74,77,79	56
	0,2,4,6,8,10,12,15,17,19,21,23,25,27,30,32,34,36,38,40, 42,45,47,49,51,53,55,57,60,62,64,66,68,70,75,79	91
	0,2,4,6,8,10,13,15,17,19,21,23,25,28,30,32,34,36,38,40, 43,45,47,49,51,53,55,58,60,62,64,66,68,70,73,79	83
	0,2,4,6,8,10,15,17,19,21,23,25,28,30,32,34,36,38,40, 43,45,47,49,51,53,55,58,60,62,64,66,68,70,73,75,79	95
	1,3,5,7,9,11,13,16,18,20,22,24,26,28,31,33,35,37,39, 41,43,46,48,50,52,54,56,58,61,63,65,67,69,71,76,78	88
	2,4,6,8,10,12,14,17,19,21,23,25,27,29,32,34,36,38,40, 42,44,47,49,51,53,55,57,59,62,64,66,68,70,72,77,79	96
838	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	6
	1,3,5,7,9,11,13,16,18,20,22,24,26,28,31,33,35,37,39,41, 43,46,48,50,52,54,56,58,61,63,65,67,69,71,73,76,78	56
	0,2,4,6,8,10,12,15,17,19,21,23,25,27,30,32,34,36,38,40, 42,45,47,49,51,53,55,57,60,62,64,66,68,70,75,79,	56
	1,3,5,7,9,11,13,16,18,20,22,24,26,28,31,33,35,37,39,41, 43,46,48,50,52,54,56,58,61,63,65,67,69,71,76,78	96
	0,2,4,6,8,10,13,15,17,19,21,23,25,28,30,32,34,36,38,40, 43,45,47,49,51,53,55,58,60,62,64,66,68,70,73,79	137
	0,2,4,6,8,12,15,17,19,21,23,25,27,30,32,34,36,38,40,42, 45,47,49,51,53,55,57,60,62,64,66,68,70,72,75,79	148
	0,2,4,6,8,10,15,17,19,21,23,25,28,30,32,34,36,38,40,43, 45,47,49,51,53,55,58,60,62,64,66,68,70,73,75,79	157
	1,3,5,7,9,11,14,16,18,20,22,24,26,29,31,33,35,37,39,41, 44,46,48,50,52,54,56,59,61,63,65,67,69,71,74,78	158
	0,2,4,6,8,10,12,15,17,21,23,25,27,30,32,34,36,38,40,42, 45,47,49,51,53,55,57,60,62,64,66,68,70,72,75,79	160
	1,3,5,7,9,11,16,18,20,22,24,26,29,31,33,35,37,39,41,44, 46,48,50,52,54,56,59,61,63,65,67,69,71,74,76,78	178

Table 7: The chosen useful cubes

round	cube	indies of cube variables
818	I_1	1,3,6,8,10,12,14,16,21,23,25,27,29,31,34,36, 38,40,42,44,46,49,53,55,57,59,61,64,66,68,70,72,74,76,79
	I_2	1,3,6,8,10,12,14,16,18,21,25,27,29,31,33,36, 38,40,42,46,48,51,53,55,57,59,61,63,66,68,70,72,74,76,78
	I_3	1,3,6,8,10,12,14,16,21,23,25,27,29,31,34,36,38,40, 42,44,49,51,53,55,57,59,61,64,66,68,70,72,74,76,79
	I_4	1,3,6,8,10,12,14,16,21,23,25,27,29,31,34,36,38,40, 42,44,49,51,53,55,57,59,62,64,66,68,70,72,74,77,79
835	I_5	0,2,4,6,8,10,13,15,17,19,21,23,25,28,30,32, 34,36,38,40,43,45,47,49,51,53,55,58,60,62,64,66,68,70,73,75,79
	I_6	0,2,4,6,8,11,13,15,17,19,21,23,26,28,30,32, 34,36,38,41,43,45,47,49,51,53,56,58,60,62,64,66,68,71,73,75,79
	I_7	0,2,4,7,9,11,13,15,17,19,22,24,26,28,30,32, 34,37,39,41,43,45,47,49,52,54,56,58,60,62,64,67,69,71,73,75,79
	I_8	0,3,5,7,9,11,13,15,18,20,22,24,26,28,30,33,35,37,39,41,43,45, 48,50,52,54,56,58,60,63,65,67,69,71,73,75,78
837	I_9	1,3,5,7,9,11,13,16,18,20,22,24,26,28,31,33,35,37,39,41, 43,46,48,50,52,54,56,58,61,63,65,67,69,71,73,76,78
838	I_{10}	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