Deterministic Cube Attacks:

A New Method to Recover Superpolies in Practice

Chen-Dong Ye and Tian Tian

National Digital Switching System Engineering & Technological Research Center, P.O. Box 407, 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 well-known cube attacks including original, division property based, and correlation cube attacks, the algebraic normal forms of superpolies could hardly be proved to be exact, which involves a small failure probability or unpractical computations. In this paper, we propose a new variant of cube attacks called deterministic cube attacks, which aims at recovering the exact algebraic normal forms of superpolies efficiently and practically. These new attacks are developed based on degree evaluation method proposed by Liu in CRYPTO2017. We apply our new cube attacks to the round-reduced Trivium. As a result, we recover the exact algebraic normal forms of some superpolies for the 818-, 819-, 837-, and 838-round Trivium. By the way, it is proved that the best cube of size 37 given by Liu in CRYPTO2017 is not a zero-sum distinguisher but a zero-biased distinguisher by recovering its exact superpoly for the first time. To the best of our knowledge, it is the first time that superpolies of cubes with the sizes less than 40 could be practically recovered for Trivium up to 838 rounds. Hopefully, our new attacks would provide some new insights on cube attacks against NFSR-based ciphers.

Keywords: Trivium · cube attacks · key recovery attack · deterministic algorithms

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 cubes with low-degree superpolies in key variables. 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 C_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 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 [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 secret key bits of the 835-round Trivium could be recovered with 2^{44} time complexity, 2^{45} keystream bits, and 2^{51} preprocessing time.

Recently, in [11], 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 to z. If so $(P_1 \oplus 1)z = (P_1 \oplus 1)P_3$ 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 [11] is that no estimation was given on the successful probability for wrong guesses.

1.1 Our Contributions.

In this paper, we propose a new variant of cube attacks, named deterministic cube attacks, which improves the experimental 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 a r-round cipher into a r_1-round polynomial representation $\boldsymbol{s}^{(r_1)}(\boldsymbol{key}, \boldsymbol{iv})$ and a 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 superpolis algebraically for a class of cubes which are called *useful cubes* in the following 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 equals 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 the attacks to some variants of round-reduced Trivium, and we obtain the following results.

1. 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. It indicates that useful cubes exist widely and can be found easily. In particular, by exhausting cubes of size 35 (with conditions), we find more than 20000 useful cubes for the 819-round Trivium within several hours on a PC.

2. We recover the superpolies of some useful cubes for the 818-, 819-, 837-, and 838-round Trivium. To the best of our concern, for the Trivium variants with more than 810 initialization rounds, it is the first time that the superpolies could be recovered exactly and practically.

3. In [12], Liu proposed a cube of 37 which was proved to be a zero-sum distinguisher for the 837-round Trivium by the degree evaluation and was tested to be zero-constant under 100 random secret keys for the 838-round Trivium. For this cube, we prove that its superpoly is not zero-constant for the 838-round Trivium by recovering its exact superpoly with our method. Consequently, based on the newly recovered superpoly, we easily give several keys under which the values of the superpoly are 1's.

We further compare our attacks with the original cube attacks, the division property based cube attacks, and correlation cube attacks. 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. Furthermore, in division property based cube attacks, they need cubes of sizes over 70 to attack Trivium variants with more than 830 initialization rounds. Although, in correlation cube attacks, they can attack the 835-round Trivium with cubes of sizes 36-37, they can not recover the exact superpoly of a cube. We summarise these comparisons in Table 1.

Table 1: Comparison with previous cube attacks

cube attacks	exact superpolies	attack rounds/cube size	ref.
Original cube attacks	no	767/28-31 709/19-23 799/32-37 802/34-36	[1] [8] [9] [10]
DP[1] based cube attacks	yes	832/72 839/78	[5, 13] [6]
Correlation cube attacks	no	835/36-37	[7]
Deterministic cube attacks	yes	838/36-37	Sect.4

1.2 Organization

The rest of this paper is organized as follows. In Section 2, we provide necessary preliminaries. 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 $wt_I(c) = |\{i|c_i \neq 0 \text{ and } x_i \in I\}|$.

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 [14].

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.

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

2.4 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 are assigned to 0 (or fixed constants). The set C_I is called a d-dimensional cube defined by I. 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 [12], which was the core technique of the degree evaluation method for NFSR-based cryptosystems in [12]. 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

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

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 [12], 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 [15], 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 Deterministic Cube Attacks

Recall that in an original cube attack, a desirable superpoly is not algebraically calculated 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 a deterministic and algebraical algorithm 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 Attacks

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

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

It follows from (5) that

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

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 calculate 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 in our attacks will be correct with probability 1. In specific, for a given set of cube variables I 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 this attack framework. First, all IV variables are set to zero except the d cube variables in I. 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 algebraic normal form 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 algebraic normal form of $g_t(s_1^{(t)}, s_2^{(t)}, \ldots, s_N^{(t)})$ will become more and more complex.

Rule 2 For i form 1 to N, one can compute the algebraic normal form of $s_i^{(t)}(\boldsymbol{key}, \boldsymbol{iv})$. As t *increases*, the algebraic normal form 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 the product $s_{i_1}^{(t)}s_{i_2}^{(t)}\cdots s_{i_l}^{(t)}$ is difficult to expand in ANF form when treating $s_{i_j}^{(t)}$ as a polynomial in key and IV variables. To solve this problem, in Subsection 3.2, we give a criterion to choose useful cubes for which we could calculate $Q_{\{s_{i_1}^{(t)},s_{i_2}^{(t)},\ldots,s_{i_l}^{(t)}\}}$ in practice without completely expanding the product $s_{i_1}^{(t)}s_{i_2}^{(t)}\cdots s_{i_l}^{(t)}$ in its ANF form.

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

$$Q_{\{s_{i_1}^{(t)},s_{i_2}^{(t)},\ldots,s_{i_l}^{(t)}\}} = \bigoplus \frac{t_1t_2\cdots t_l}{t_I}, \tag{9}$$

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 $t_1t_2\cdots t_l$, need to compute. If $t_1t_2\cdots t_l$ is not divisible by t_I, then we have

$$\frac{t_1t_2\cdots t_l}{t_I} = 0,$$

which implies that $t_1t_2\cdots t_l$ has no contribution to $Q_{\{s_{i_1}^{(t)},s_{i_2}^{(t)},\ldots,s_{i_l}^{(t)}\}}$. Hence an effective $t_1t_2\cdots t_l$ should satisfy that $t_1t_2\cdots t_l$ is divisible by t_I. To make this point clear we rewrite (9) as

$$Q_{\{s_{i_1}^{(t)},s_{i_2}^{(t)},\ldots,s_{i_l}^{(t)}\}} = \bigoplus_{t_I|t_1t_2\cdots t_l} \frac{t_1t_2\cdots t_l}{t_I}, \tag{10}$$

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 of $s_i^{(t)}$ 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 $s_i^{(t)} = v_1v_2k_1 \oplus v_2v_3k_1k_2 \oplus v_4k_3$. Then $v_1v_2k_1$ and $v_2v_3k_1k_2$ 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 $t_1t_2\cdots t_l$ is divisible by t_I where t_j is a term of $s_{i_j}^{(t)}$ 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 $I = \{v_0, v_1, v_2, v_3\}$, $s_1 = v_0v_1k_0 \oplus v_2 \oplus v_3$, $s_2 = v_2v_3k_5 \oplus v_0 \oplus v_2$, $s_3 = v_2 \oplus v_3$, $u_1 = s_1s_2$, and $u_2 = s_2s_3$. Then it can be seen that u_1 is a tight term for I, while u_2 is not a tight term for I. The sets of maximum degree terms of s_1 and s_2 are $\{v_0v_1k_0\}$ and $\{v_2v_3k_5\}$, respectively. According to Property 1, we have

$$\frac{u_1}{t_I} = k_0k_5,$$

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

Based on 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 $s_{i_1}^{(t)} s_{i_2}^{(t)} \cdots s_{i_l}^{(t)}$ of g_t satisfies*

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

then C_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 C_I, we simply have

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

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

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 C_I is useful we need to solve

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

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 results of a degree evaluation algorithm are far from real degrees. Considering this issues, we choose to use the idea of numeric mapping in [12]. Details are given in Algorithm 2. As for the definition and methodology of numeric mapping and numeric degree please refer to [12] 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 every term $s_{i_1}^{(t)} s_{i_2}^{(t)} \cdots s_{i_l}^{(t)}$ of 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 $\mathrm{DEG}_I(s_{i_1}^{(t)} s_{i_2}^{(t)} \cdots s_{i_l}^{(t)}) \leq |I|$ for every term $s_{i_1}^{(t)} s_{i_2}^{(t)} \cdots s_{i_l}^{(t)}$ of g_t, then we keep C_I in Algorithm 2 which is a useful cube. 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.

Algorithm 2 Finding Useful Cubes with Degree Evaluation

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

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

3.4 Recover the Exact Superpoly of a Useful Cube

After finding a useful cube C_I, we will recover the 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 **RecoverCoefficient** is called to calculate $Q_{\{s_{i_1}^{(t)}, s_{i_2}^{(t)}, \ldots, s_{i_l}^{(t)}\}}$. 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 $MaxDegTerms[j]$, where $MaxDegTerms$ is a list of sets and $MaxDegTerms[j]$ represents the j-th set in $MaxDegTerms$.

Our goal is to find all the combinations $(t_1, t_2, \ldots, t_l)$ such that $\prod_{j=1}^{l} t_j$ is divisible by t_I, where $t_j \in MaxDegTerms[j]$ for $1 \le j \le l$. 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 $MaxDegTerms[j]$ for $1 \le j \le l$. In the *Reduce* operation, we first do IV representation for each term in $MaxDegTerms[j]$. Then, for the repeated terms which are generated by the terms containing the same cube variables, we only keep one of them. Finally, the reduced terms of $MaxDegTerms[j]$ are stored in $RedMaxDegTerms[j]$, which is called a set of reduced maximum degree terms.

In this paper, a combination $(t_{j_1}^1, t_{j_2}^2, \ldots, t_{j_l}^l)$ satisfying $\prod_{i=1}^{l} t_{j_i}^i = t_I$, where $t_{j_i}^i \in RedMaxDegTerms[i]$ for $1 \le i \le l$, is called a valid combination. It can be seen that, by finding all the valid combinations, we could find all the combinations $(t_1, t_2, \ldots, t_l)$ such that $\prod_{i=1}^{l} t_i$ is divisible by t_I, where $t_i \in MaxDegTerms[i]$ for $1 \le i \le l$. Note that $\prod_{i=1}^{l} |RedMaxDegTerms[i]|$ would be much smaller than $\prod_{i=1}^{l} |MaxDegTerms[i]|$, since $MaxDegTerms[i]$ $(1 \le i \le l)$ 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 $\prod_{i=1}^{l} |RedMaxDegTerms[i]|$ would be much smaller than $\prod_{i=1}^{l} |MaxDegTerms[i]|$, 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 $t_{j_i}^i \in RedMaxDegTerms[i]$ for $1 \le i \le l$. Instead, we pick up elements from $RedMaxDegTerms$ 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., $t^1_{j_1}, t^2_{j_2}, \ldots, t^d_{j_d}$.

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

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

$$\begin{aligned}\deg_I(t^1_{j_1} t^2_{j_2} \cdots t^l_{j_l}) &\leq \deg_I(t^1_{j_1} t^2_{j_2} \cdots t^d_{j_d}) + \deg_I(t^{d+1}_{j_{d+1}} t^{d+2}_{j_{d+2}} \cdots t^l_{j_l}) \\ &< \sum_{i=1}^{d} \deg_I(t^i_{j_i}) + \deg_I(t^{d+1}_{j_{d+1}} t^{d+2}_{j_{d+2}} \cdots t^l_{j_l}) \\ &\leq \sum_{i=1}^{l} \deg_I(t^i_{j_i}) = |I|.\end{aligned}$$

Namely, $\deg_I(t^1_{j_1} t^2_{j_2} \cdots t^l_{j_l}) < |I|$. 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 $t_w \in RedMaxDegTerms[w]$ satisfies that $\deg_I(t_w \cdot t^1_{j_1} t^2_{j_2} \cdots t^d_{j_d}) < \deg_I(t_w) + \deg_I(t^1_{j_1} t^2_{j_2} \cdots t^d_{j_d})$, then we would throw away all the combinations whose first d components are $t^1_{j_1}, t^2_{j_2}, \ldots, t^d_{j_d}$.

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

$$\begin{aligned}\deg_I(t^1_{j_1} t^2_{j_2} \cdots t^l_{j_l}) &\leq \deg_I(t^1_{j_1} t^2_{j_2} \cdots t^{d+1}_{j_{d+1}}) + \deg_I(t^{d+2}_{j_{d+2}} t^{d+3}_{j_{d+3}} \cdots t^l_{j_l}) \\ &< \deg_I(t^1_{j_1} t^2_{j_2} \cdots t^d_{j_d}) + \deg_I(t^{d+1}_{j_{d+1}}) + \deg_I(t^{d+2}_{j_{d+2}} t^{d+3}_{j_{d+3}} \cdots t^l_{j_l}) \\ &\leq \sum_{i=1}^{l} \deg_I(t^i_{j_i}) = |I|.\end{aligned}$$

Namely, $\deg_I(t^1_{j_1} t^2_{j_2} \cdots t^l_{j_l}) < |I|$. 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 $RedMaxDegTerms[d+1]$. 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 t_I in u. The final step is to recover the superpoly of t_I in u according to all the valid combinations. Let $(t^1_{j_1}, t^2_{j_2}, \ldots, t^l_{j_l})$ be a valid combination. Since the terms in $RedMaxDegTerms[i]$ are reduced from $MaxDegTerms[i]$, the combination $(t^1_{j_1}, t^2_{j_2}, \ldots, t^l_{j_l})$ 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 $t_j \in MaxDegTerms[j]$ for $1 \leq j \leq l$. All the combinations which $(t^1_{j_1}, t^2_{j_2}, \ldots, t^l_{j_l})$ corresponds to can be covered by a vector $(\lambda^1_{j_1}, \lambda^2_{j_2}, \ldots, \lambda^l_{j_l})$, where $\lambda^w_{j_w} = \frac{s^{(t)}_{i_w}}{t^w_{j_w}}$ for $w \in \{1, 2, \ldots, l\}$. In this paper, $(\lambda^1_{j_1}, \lambda^2_{j_2}, \ldots, \lambda^l_{j_l})$ is called the superpoly vector of $(t^1_{j_1}, t^2_{j_2}, \ldots, t^l_{j_l})$. Then, the contribution of the valid combination

Algorithm 3 Recover the Exact ANF of a Useful Cube

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

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

$(t_{j_1}^1, t_{j_2}^2, \ldots, t_{j_l}^l)$ to the superpoly of t_I in u is $\prod_{i=1}^{l} \lambda_{j_w}^w$. Thus,

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

Algorithm 4 Recover the Superpoly of t_I in a Tight Term

procedure RecoverCoefficient(u,I)
  Assume that $u = s_{i_1}^{(t)} s_{i_2}^{(t)} \cdots s_{i_l}^{(t)}$;
  for $1 \le j \le l$ **do**
    Collect maximum degree terms of $s_{i_j}^{(t)}$ and store them in $MaxDegTerms[j]$;
    Apply the *Reduce* operation to $MaxDegTerms[j]$ and store the reduced terms in $RedMaxDegTerms[j]$;
  end for
  Figure out all valid combinations;
  Set $Q_{\{s_{i_1}^{(t)}, s_{i_2}^{(t)}, \ldots, s_{i_l}^{(t)}\}} = 0$;
  for each valid combination $(t_{j_1}^1, t_{j_2}^2, \ldots, t_{j_l}^l)$ **do**
    Recover the corresponding superpoly vector $(\lambda_{j_1}^1, \lambda_{j_2}^2, \ldots, \lambda_{j_l}^l)$;
    Set $Q_{\{s_{i_1}^{(t)}, s_{i_2}^{(t)}, \ldots, s_{i_l}^{(t)}\}} = Q_{\{s_{i_1}^{(t)}, s_{i_2}^{(t)}, \ldots, s_{i_l}^{(t)}\}} \oplus \prod_{w=1}^{l} \lambda_{j_w}^w$;
  end for
  return $Q_{\{s_{i_1}^{(t)}, s_{i_2}^{(t)}, \ldots, s_{i_l}^{(t)}\}}$;
end procedure

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

$$s_1 = v_4v_5k_2k_3 \oplus v_4v_5k_4 \oplus v_4v_5k_5 \oplus v_2v_3 \oplus v_5v_6 \oplus k_3,$$

$$\begin{aligned} s_4 =& v_0v_1v_2v_3k_0 \oplus v_0v_1v_2v_3k_1 \oplus v_0v_1v_3v_4k_0 \oplus v_0v_1v_4v_6k_2 \\ &\oplus v_1v_2v_3k_2 \oplus v_1v_2v_3k_{44} \oplus v_2v_3 \oplus v_4k_0 \oplus v_5k_1, \end{aligned}$$

$$s_6 = v_3v_6 \oplus v_4v_6 \oplus v_6v_7, \text{ and } s_8 = v_6v_9 \oplus v_7v_9 \oplus v_8v_9.$$

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

$$MaxDegTerms[1] = \{v_4v_5k_2k_3, v_4v_5k_4, v_4v_5k_5, v_2v_3, v_5v_6\}.$$

Then, the *Reduce* operation is applied and the set of reduced maximum degree terms of s_1 is

$$RedMaxDegTerms[1] = \{v_2v_3, v_4v_5, v_5v_6\}.$$

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

$$RedMaxDegTerms[2] = \{v_0v_1v_2v_3, v_0v_1v_3v_4, v_0v_1v_4v_6\},$$

$$RedMaxDegTerms[3] = \{v_3v_6, v_4v_6, v_6v_7\},$$

and

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

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 $\{(v_2v_3, v_0v_1v_2v_3), (v_2v_3, v_0v_1v_3v_4), (v_4v_5, v_0v_1v_3v_4), (v_4v_5, v_0v_1v_4v_6), (v_5v_6, v_0v_1v_4v_6)\}$. 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 again and again to form a full combination. As a result, we can obtain the only valid combination $(v_0v_1v_2v_3, v_4v_5, v_6v_7, v_8v_9)$ without checking every combination. The superpoly vector of $(v_0v_1v_2v_3, v_4v_5, v_6v_7, v_8v_9)$ is $(k_0 \oplus k_1, k_2k_3 \oplus k_4 \oplus k_5, 1, 1)$. Immediately, we have that

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

4 Applications to Trivium

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

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. Moreover, we introduce some heuristic rules of choosing cubes. 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.

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 based on 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 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 [12]. 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. We make a small improvement of the degree evaluation method proposed by Liu in [12]. 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. It worth noting that $s_{287}^{t-92} = s_{288}^{t-91}$. Hence, 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 [12]. 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. For some cubes, this situation would happen in early rounds. Moreover, 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, two adjacent internal state bits are treated as a whole. Namely, we would collect the reduced maximum degree terms of the product of two adjacent internal state bits instead of collecting the reduced maximum degree terms of these two bits 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 such that Rules 1 and 2 mentioned in Subsection 3.1 are satisfied at the same time. Fortunately, this dilemma can be solved by taking an extra step when calculating the superpoly p_I. Following from (12), we have

$$\begin{aligned} p_I &= \bigoplus_{\substack{s_{i_1}^{(t)} s_{i_2}^{(t)} \cdots s_{i_l}^{(t)} \\ \text{is a tight term in } T(g_t)}} Q_{\{s_{i_1}^{(t)}, s_{i_2}^{(t)}, \ldots, s_{i_l}^{(t)}\}} \\ &= \bigoplus_{\substack{s_{i_1}^{(t)} s_{i_2}^{(t)} \cdots s_{i_l}^{(t)} \text{is a} \\ \text{tight term in } T(g_t)}} \; \bigoplus_{\substack{s_{j_1}^{(t_0)} s_{j_2}^{(t_0)} \cdots s_{j_d}^{(t_0)} \text{ is a} \\ \text{term in } T(f_{\{s_{i_1}^{(t)}, s_{i_2}^{(t)}, \ldots, s_{i_l}^{(t)}\}})}} Q_{\{s_{j_1}^{(t_0)}, s_{j_2}^{(t_0)} \ldots s_{j_d}^{(t_0)}\}}, \end{aligned} \tag{13}$$

where $t_0 < t$ and $f_{\{s_{i_1}^{(t)}, s_{i_2}^{(t)}, \ldots, s_{i_l}^{(t)}\}}$ is the expressed polynomial of $s_{i_1}^{(t)} s_{i_2}^{(t)} \cdots s_{i_l}^{(t)}$ on the internal state $s^{(t_0)}$. According to (13), when calculating the superpoly of t_I in the tight term $s_{i_1}^{(t)} s_{i_2}^{(t)} \cdots s_{i_l}^{(t)}$ of g_t, we first express it as a polynomial on the internal state $s^{(t_0)}$ and then calculate the superpoly of t_I in each term of $f_{(s_{i_1}^{(t)}, s_{i_2}^{(t)}, \ldots, s_{i_l}^{(t)})}$. 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 $1 \leq i \leq 288$.

4.1.3 Rules of Choosing Cubes

Recall that, for a term u, $\mathrm{DEG}_I(u)$ equals to the summation of some $\mathrm{DEG}_I(s_i^{(t)})$'s and some $\mathrm{DEG}_I(s_i^{(t)} s_{i+1}^{(t)})$'s in Algorithm 5. In order to make $\mathrm{DEG}_I(u) \leq |I|$ hold for each

term u of g_t, $\mathrm{DEG}_I(s_i^{(t)})$'s and $\mathrm{DEG}_I(s_i^{(t)}s_{i+1}^{(t)})$'s should be as small as possible. Note that $\mathrm{DEG}_I(s_i^{(t)})$'s and $\mathrm{DEG}_I(s_i^{(t)}s_{i+1}^{(t)})$'s are evaluated iteratively. Accordingly, we should choose cube variables such that $\mathrm{DEG}_I(s_i)$'s and $\mathrm{DEG}_I(s_is_{i+1})$'s are kept small for relatively early time instances. As a result, we set the following heuristic rule of choosing cubes variables.

Rule 3. The chosen set of cube variables I does not contain v_i and v_j satisfying $|i-j|=1$, $|i-j|=14$ or $|i-j|=16$.

For a relatively early t, we have $s_{286}^{(t)} = v_i \oplus v_{i+13}v_{i+14} \oplus v_{i+15}$ and $s_{287}^{(t)} = v_{i+1} \oplus v_{i+14}v_{i+15} \oplus v_{i+16}$. When the first condition is satisfied (the non-cube variables are set to 0), the nonlinear terms of $s_{286}^{(t)}$ and $s_{287}^{(t)}$ would disappear, since we would not choose cube variables with adjacent indices. Moreover, if the second and third conditions are satisfied as well, then

$$s_{286}^{(t)}s_{287}^{(r)} = (v_i \oplus v_{i+15})(v_{i+1} \oplus v_{i+16}) = v_iv_{i+1} \oplus v_{i+15}v_{i+16} \oplus v_iv_{i+16} \oplus v_{i+1}v_{i+15}$$

would equal to 0, since each quadratic term at least one variable which is not in I and set to 0. Namely, the nonlinear term of the feedback bit

$$s_1^{t+1} = s_{243}^{(t)} \oplus s_{286}^{(t)}s_{287}^{(t)} \oplus s_{288}^{(t)} \oplus s_{69}^{(t)}$$

would disappear. Thus, $\mathrm{DEG}_I(s_1^{t+1}) = \mathrm{DEG}_I(s_{243}^{(t)} \oplus s_{288}^{(t)} \oplus s_{69}^{(t)})$, which is usually smaller than $\mathrm{DEG}_I(s_{243}^{(t)} \oplus s_{286}^{(t)}s_{287}^{(t)} \oplus s_{288}^{(t)} \oplus s_{69}^{(t)})$. As a result, for relatively early time instances, $\mathrm{DEG}_I(s_i)$'s and $\mathrm{DEG}_I(s_is_{i+1})$'s would be relatively small. It is reasonable that cubes chosen according to the above rule would have a larger chance to be a useful cube.

Another key point is determining the sizes of cubes. Note that, for a cube $C_{I'}$ yielding a zero-sum distinguisher, the degree on I' of each term of g_t is less than $|I'|$. Meanwhile, for a useful cube C_I, g_t only has tight terms for I and terms whose degrees on I is less than I. In this respect, a useful cube is close to one yielding a zero-sum distinguisher. Hence, when searching for useful cubes, it is reasonable to set the sizes of cubes close to those of cubes yielding zero-sum distinguishers.

4.2 Experimental Results

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

4.2.1 Results for Trivium Variants with 800-832 Rounds.

First, we try to find useful cubes for Trivium variants with 800-832 initialization rounds. For these variants, in order to get enough cube variables, we choose cubes which do not contain variables v_i, v_j satisfying $|i-j|=1$ or $|i-j|=16$. Second, we use cubes of sizes 33-36, since experiments show that the sizes of distinguishers found for these variants are around 33-36. Finally, for each variant with r initialization rounds ($800 \le r \le 832$), we randomly test 10000 cubes of size 33-36. 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- and 819-round Trivium.

We take the 818- and 819-round Trivium as an example to show more details of our attacks. To find useful cubes, we exhaust all cubes of size 35 which do not contain variables v_i, v_j satisfying $|i-j|=1$ or $|i-j|=16$ with Algorithm 5. In this phase, we express the output

Table 2: The number of tight terms of useful cubes for the 819-round Trivium

#Tight terms	$[1,10]$	$[11,30]$	$[31,60]$	$[61,100]$	$[100,+\infty]$
#Useful cubes	1352	7175	11800	876	94

bit z_{819} as a polynomial on the internal state $s^{(418)}$, i.e., $z_{819} = g_{418}(s^{(418)})$. We complete this phase in several hours and find 21297 useful cubes, see Table 2.

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_{418}. Then, we call Algorithm 6 to calculate the superpoly p_I. In Algorithm 6, to recover the superpoly of t_I in each tight term u of g_{418}, we first express it as a polynomial on the internal state $s^{(363)}$, denoted by $f_u(s^{(363)})$, and then calculate the superpoly of t_I in each term of $f_u(s^{(363)})$. Hence, we only need to calculate the exact ANFs of $s_1^{(363)}, s_2^{(363)}, \ldots, s_{288}^{(363)}$. The key point in this phase is finding all the valid combinations in each tight term u' of f_u. This could be done efficiently with the help of the two strategies proposed in Subsection 3.4. For instance,

$$u' = s_{64}^{(363)} s_{65}^{(363)} s_{102}^{(363)} s_{103}^{(363)} s_{124}^{(363)} s_{133}^{(363)} s_{136}^{(363)} s_{137}^{(363)} s_{145}^{(363)} s_{147}^{(363)} s_{148}^{(363)} s_{154}^{(363)} s_{155}^{(363)}$$

is a tight term of f_u, where $u = s_{121}^{(418)} s_{122}^{(418)} s_{157}^{(418)} s_{158}^{(418)} s_{193}^{(418)} s_{194}^{(418)} s_{203}^{(418)} s_{204}^{(418)} s_{211}^{(418)}$. There are totally

$$94 \times 99 \times 52 \times 42 \times 755 \times 34 \times 676 \times 542 = 191155823581282560 \approx 2^{57}$$

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 vectors for each valid combination. Finally, we obtain the superpoly p_I within about ten minutes, see Table 3 for details.

Among the found useful cubes, we choose some cubes with relatively few tight terms to recover their exact superpolies for the 819-round Trivium. Furthermore, by sliding some useful cubes of the 819-round Trivium, we get useful cubes for the 818-round Trivium, and then recover the corresponding superpolies. We list a part of our results in Table 3.

Table 3: Useful cubes and corresponding superpolies of the 818- and 819-round Trivium

#Rounds	Indices of cube variables	Superpoly
818	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	$h_{65}g_{95}g_{914}g_{918}g_{924}g_{929}g_{931}g_{938}g_{940}g_{944}g_{948}g_{951}$
818	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	$h_{65}g_{95}g_{918}g_{927}g_{929}g_{933}g_{938}g_{942}(g_{50}\oplus 1)(f_{22}\oplus 1)$
818	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	$h_{57}h_{65}g_{95}g_{916}g_{923}g_{927}g_{929}g_{931}g_{938}g_{940}g_{942}g_{946}g_{948}(f_{22}\oplus 1)$
818	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	$h_{57}h_{64}g_{95}g_{916}g_{927}g_{929}g_{931}g_{938}g_{940}g_{942}g_{946}g_{948}(f_{22}\oplus 1)$
819	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	$h_{57}h_{65}h_{68}g_{94}g_{921}g_{925}g_{927}g_{938}g_{940}g_{946}g_{948}f_{20}$ $\oplus h_{57}h_{65}g_{96}g_{921}g_{925}g_{927}g_{938}g_{940}g_{946}g_{948}f_{20}$ $\oplus h_{38}h_{57}h_{65}g_{96}g_{925}g_{927}g_{938}g_{940}g_{948}(g_{929}\oplus g_{938})f_{20}$

$f_i = k_i k_{i+1} \oplus k_{i+2} \oplus k_{i+44} \oplus k_{i+53}$ 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 $0 \le i \le 52$
$g_{53} = k_{53} \oplus k_{78} k_{79}$
$h_i = k_i$ for $0 \le i \le 79$

Table 4: Cubes of the 837- and 838-round Trivium

#Rounds	Indices of cube variables	#Superpoly vectors
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	3395
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,	1596

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

For the 837- and 838-round Trivium, we do the similar experiments. As a result, we find several useful cubes, we list a part of them in Table 7 in the Appendix 3. In this case, we choose the cubes with fewest tight terms to recover their superpolies and list the number of superpoly vectors for these cubes in Table 4. For the detailed ANF, please refer to our account on GitHub[1].

For the sake of convenience, we denote the cubes of the 837- and 838-round Trivium by C_{I_1} and C_{I_2} respectively. Interestingly, we find that these two superpolies are heavily zero-biased. However, based on the recovered ANF, we find some keys under which the values of these two superpolies are 1's. We list them in Table 5, where $\boldsymbol{key}[i] = 0\text{x}k_{8i+7}k_{8i+6}\ldots k_{8i}$ $(0 \leq i \leq 9)$.

Table 5: The found secret keys

Cubes	$\boldsymbol{key}[0]$	$\boldsymbol{key}[1]$	$\boldsymbol{key}[2]$	$\boldsymbol{key}[3]$	$\boldsymbol{key}[4]$	$\boldsymbol{key}[5]$	$\boldsymbol{key}[6]$	$\boldsymbol{key}[7]$	$\boldsymbol{key}[8]$	$\boldsymbol{key}[9]$
C_{I_1}	0x4f	0xe7	0xaf	0x8e	0x2e	0x5e	0x72	0x7b	0x31	0xf9
	0x4f	0xe7	0xaf	0x8e	0x2e	0x5e	0x72	0x7b	0x31	0xf9
	0x4f	0xe7	0xaf	0x8e	0x2e	0x5e	0x73	0x7b	0x31	0xf1
	0x4e	0xe7	0xaf	0x86	0x2e	0x5e	0x72	0x7b	0x31	0xf9
	0x4e	0xe7	0xaf	0x86	0x2e	0x5e	0x73	0x7b	0x31	0xf1
C_{I_2}	0xff	0xff	0xdf	0xff	0x0f	0x7f	0xf5	0x3f	0xf8	0xff
	0xff	0xff	0xdf	0xff	0x0f	0x7f	0xc5	0x3f	0xf8	0xff
	0xfb	0xcb	0xd7	0xdf	0xd4	0xbf	0xbd	0xbd	0x5c	0xfc
	0xfb	0xcb	0xd7	0xdf	0xd4	0xbf	0xbd	0xbd	0x3c	0xfc
	0xfb	0xcb	0xd7	0x5f	0xd5	0xaf	0xbd	0xbd	0x7c	0xfc

It worth noting that C_{I_2} is the same as the cube proposed by Liu in [12]. Recall that Liu tested 100 random keys for the superpoly of this cube, the values are always 0. Our experiments show that its superpoly is not zero-constant, which means that the output of the 838-round Trivium achieves the maximum degree 37 over this subset of IV bits, and so the degree given by Liu for this cube is tight. This also implies that 100 random keys are not enough for zero-constant function testing.

4.3 Discussions

4.3.1 How to Recover the Information of Secret Key.

Since f_i's and g_i's in Table 3 are very simple expressions on key variables, we can recover the information of key variables easily from them. Hence, when recovering the information of key variables with the superpolies found by us, we would treat f_i's, g_i's as a whole. For the sake of convenience, we call $u = \prod_{a\in A} f_a \prod_{b\in B} g_b \prod_{c\in C} h_c$ a big term. According to the number of big terms, we can apply different strategies to use our superpolies.

If a superpoly has few big terms, then we can generate a nonlinear equation on f_i's,

[1] https://github.com/yechendong/Deterministic-Cube-Attacks

Table 6: Some bases of the superpoly of C_{I_2}

Group	Basis
G_1	$\{k_{55}\}$
G_2	$\{k_6 \oplus k_{31}k_{32} \oplus k_{33}\}$
G_3	$\{k_{21} \oplus k_{46}k_{47} \oplus k_{48}\}$
G_4	$\{k_{18}k_{19} \oplus k_{20} \oplus k_{62}, k_{20}k_{21} \oplus k_{22} \oplus k_{64}, k_{24}k_{25} \oplus k_{26} \oplus k_{68}\}$
G_5	$\{k_4 \oplus k_{29}k_{30} \oplus k_{31}, k_{27} \oplus k_{52}k_{53} \oplus k_{54}, k_{29} \oplus k_{54}k_{55} \oplus k_{56}, k_{24}k_{25} \oplus k_{26} \oplus k_{68}\}$

g_i's and h_i's. For example, with the first cube in Table 3, we have the following equation

$$h_{65}g_5g_{14}g_{18}g_{24}g_{29}g_{31}g_{38}g_{40}g_{44}g_{48}g_{51} = q,$$

where q is value of the this superpoly under the real key.

If a superpoly has many big terms, then we can use the correlation cube attacks proposed by Liu in [7] to recover some information of key variables. In this case, there would be some benefits brought by knowing the exact ANF of the superpoly. First, we can find several different groups of basis for the same superpoly. For instance, we list some evident groups of basis the superpoly of C_{I_2} in the following table. Second, we can calculate the correlation relationship between the basis and the superpoly according to the exact ANF of the superpoly. Note that, in correlation cube attacks, the correlation relationship between the basis and the superpoly is estimated by summing all the output bits over the chosen cube under some randomly chosen keys. Consequently, we can reduce the complexity in the preprocessing phase of correlation cube attacks dramatically.

4.3.2 Extra Benefits of Our New Attacks.

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 distinguishers could not be detected by Liu's degree evaluation method and Todo's division property method since their methods 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.

4.3.3 A More Generalized Criterion

In the above, we introduce a new criterion of useful cubes under which the superplolies 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 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 and apply the *Reduce* operation to them. Store the reduced terms in $RedMaxDegTerms$.

Step 2. Collect the set of reduced sub-maximum degree terms for each bit in u and apply the *Reduce* operation to them. Store the reduced terms in $RedSubMaxDegTerms$.

Step 3. Find all the valid 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 $t_{i_j}^j \in RedMaxDegTerms[j]$ $(1 \le j \le l)$. 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 $1 \le j \le l$, 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 $t_{i_w}^w \in RedMaxDegTerms[w]$ for $w \ne j$ and $t_{i_w}^w \in$

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

5 Conclusion

In this paper, we propose deterministic 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 illustrations, we apply our attacks to Trivium. In applications to Trivium, we choose cubes which do not contain variables v_i, v_j satisfying $|i - j| = 1$ or $|i - j| = 16$. It makes the sizes of chosen cubes are always less than 40. With such cubes, we find some useful cubes and recover the corresponding superpolies for up to the 838-round Trivium. However, it seems hard to increase the number of attacking rounds with such 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.

References

[1] Itai Dinur and Adi Shamir. Cube attacks on tweakable black box polynomials. In *Advances in Cryptology - EUROCRYPT 2009, 28th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cologne, Germany, April 26-30, 2009. Proceedings*, pages 278–299, 2009.

[2] Jean-Philippe Aumasson, Itai Dinur, Willi Meier, and Adi Shamir. Cube testers and key recovery attacks on reduced-round MD6 and trivium. In *Fast Software Encryption, 16th International Workshop, FSE 2009, Leuven, Belgium, February 22-25, 2009, Revised Selected Papers*, pages 1–22, 2009.

[3] Itai Dinur and Adi Shamir. Breaking grain-128 with dynamic cube attacks. In *Fast Software Encryption - 18th International Workshop, FSE 2011, Lyngby, Denmark, February 13-16, 2011, Revised Selected Papers*, pages 167–187, 2011.

[4] Senyang Huang, Xiaoyun Wang, Guangwu Xu, Meiqin Wang, and Jingyuan Zhao. Conditional cube attack on reduced-round keccak sponge function. In *Advances in Cryptology - EUROCRYPT 2017 - 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Paris, France, April 30 - May 4, 2017, Proceedings, Part II*, pages 259–288, 2017.

[5] Yosuke Todo, Takanori Isobe, Yonglin Hao, and Willi Meier. Cube attacks on non-blackbox polynomials based on division property. In *Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part III*, pages 250–279, 2017.

[6] Qingju Wang, Yonglin Hao, Yosuke Todo, Chaoyun Li, Takanori Isobe, and Willi Meier. Improved division property based cube attacks exploiting low degree property of superpoly. Cryptology ePrint Archive, Report 2017/1063, 2017. https://eprint.iacr.org/2017/1063.

[7] Meicheng Liu, Jingchun Yang, Wenhao Wang, and Dongdai Lin. Correlation cube attacks: From weak-key distinguisher to key recovery. In *Advances in Cryptology - EUROCRYPT 2018 - 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29 - May 3, 2018 Proceedings, Part II*, pages 715–744, 2018.

[8] Piotr Mroczkowski and Janusz Szmidt. Corrigendum to: The cube attack on stream cipher Trivium and quadraticity tests. *IACR Cryptology ePrint Archive*, 2011:32, 2011.

[9] Pierre-Alain Fouque and Thomas Vannet. Improving key recovery to 784 and 799 rounds of Trivium using optimized cube attacks. In *Fast Software Encryption - 20th International Workshop, FSE 2013, Singapore, March 11-13, 2013. Revised Selected Papers*, pages 502–517, 2013.

[10] Chen-Dong Ye and Tian Tian. A new framework for finding nonlinear superpolies in cube attacks against trivium-like ciphers. In Willy Susilo and Guomin Yang, editors, *Information Security and Privacy - 23rd Australasian Conference, ACISP 2018, Wollongong, NSW, Australia, July 11-13, 2018, Proceedings*, volume 10946 of *Lecture Notes in Computer Science*, pages 172–187. Springer, 2018.

[11] Ximing Fu, Xiaoyun Wang, Xiaoyang Dong, and Willi Meier. A key-recovery attack on 855-round trivium. *IACR Cryptology ePrint Archive*, 2018:198, 2018.

[12] Meicheng Liu. Degree evaluation of NFSR-Based cryptosystems. In *Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part III*, pages 227–249, 2017.

[13] Yosuke Todo, Takanori Isobe, Yonglin Hao, and Willi Meier. Cube attacks on non-blackbox polynomials based on division property (full version). *IACR Cryptology ePrint Archive*, 2017:306, 2017.

[14] Christophe De Cannière and Bart Preneel. Trivium. In *New Stream Cipher Designs - The eSTREAM Finalists*, pages 244–266. 2008.

[15] Ximing Fu, Xiaoyun Wang, and Jiazhe Chen. Determining the nonexistent terms of non-linear multivariate polynomials: How to break grain-128 more efficiently. *IACR Cryptology ePrint Archive*, 2017:412, 2017.

Appendix

1. The Optimization For Trivium

2. A Part of Experimental Results for the 837- and 838-round 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., $u = f_u(s^{(t_0)})$;
  for each term $u' = s_{j_1}^{(t_0)} s_{j_2}^{(t_0)} \cdots s_{j_w}^{(t_0)}$ of f_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 $MaxDegTerms[i]$;
    Apply the *Reduce* operation to $MaxDegTerms[i]$, and store the reduced terms in $RedMaxDegTerms[i]$;
  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

Table 7: 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