IACR Transactions on Symmetric Cryptology
ISSN 2519-173X, Vol. 2021, No. 4, pp. 100–123. DOI:10.46586/tosc.v2021.i4.100-123

Automatic Search of Cubes for Attacking Stream Ciphers

Yao Sun

State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China.
School of Cyber Security, University of Chinese Academy of Sciences, Beijing, China.
sunyao@iie.ac.cn

Abstract. Cube attack was proposed by Dinur and Shamir, and it has become an important tool for analyzing stream ciphers. As the problem that how to recover the superpolys accurately was resolved by Hao et al. in EUROCRYPT 2020, another important problem is how to find "good" superpolys, which is equivalent to finding "good" cubes. However, there are two difficulties in finding "good" cubes. Firstly, the number of candidate cubes is enormous and most of the cubes are not "good". Secondly, it is costly to evaluate whether a cube is "good".

In this paper, we present a new algorithm to search for a kind of "good" cubes, called *valuable* cubes. A cube is called *valuable*, if its superpoly has (at least) a balanced secret variable. A valuable cube is "good", because its superpoly brings in 1 bit of information about the key. More importantly, the superpolys of valuable cubes could be used in both theoretical and practical analyses. To search for valuable cubes, instead of testing a set of cubes one by one, the new algorithm deals with the set of cubes together, such that the common computations can be done only once for all candidate cubes and duplicated computations are avoided. Besides, the new algorithm uses a heuristic method to reject useless cubes efficiently. This heuristic method is based on the divide-and-conquer strategy as well as an observation.

For verifications of this new algorithm, we applied it to Trivium and Kreyvium, and obtained three improvements. Firstly, we found two valuable cubes for 843-round Trivium, such that we proposed, as far as we know, the first theoretical key-recovery attack against 843-round Trivium, while the previous highest round of Trivium that can be attacked was 842, given by Hao et al. in EUROCRYPT 2020. Secondly, by finding many small valuable cubes, we presented *practical* attacks against 806- and 808-round Trivium for the first time, while the previous highest round of Trivium that can be attacked practically was 805. Thirdly, based on the cube used to attack 892-round Kreyvium in EUROCRYPT 2020, we found more valuable cubes and mounted the key-recovery attacks against Kreyvium to 893-round.

Keywords: stream cipher · cube attack · division property · monomial prediction · MILP · Trivium · Kreyvium.

1 Introduction

Cube Attack: Dinur and Shamir proposed cube attack in EUROCRYPT 2009 [DS09], and the cube attack has been successfully used to attack various stream ciphers [ADMS09, DS11, FV14, DMP+15, SBD+16]. With the help of Mix Integer Linear Programming (MILP) approach, cube attack is able to attack stream ciphers using large cubes. Todo et al. proposed the division property in [Tod15, TM16], and by combining with the cube attack, they were able to significantly improve the attacks against Trivium, Grain128a, ACORN in [TIHM17]. Cube attack utilizes a large amount of data to build special relations between

Received: 2021-06-01 Revised: 2021-09-01 Accepted: 2021-11-01 Published: 2021-12-03

secret variables. Specifically, let $\mathbf{k}$ and $\mathbf{v}$ be secret and public variables. The output bit of a stream cipher can be represented as a Boolean polynomial $f(\mathbf{k}, \mathbf{v})$. Generally, although a relation of secret variables can be obtained directly through the specialization of $\mathbf{v}$ in $f(\mathbf{k}, \mathbf{v})$, this relation has a high algebraic degree and is hard to be analyzed. Alternatively, instead of using a single value of $\mathbf{v}$, cube attack takes advantage of enormous values. A cube consists of many values of $\mathbf{v}$'s. Some bits of $\mathbf{v}$ are set as active, and this means, they take all possible combinations of values, while the other bits of $\mathbf{v}$ are set inactive and remain unchanged. By taking the values of $f(\mathbf{k}, \mathbf{v})$ over all values in the cube, the sum leads to a relation of secret variables. This relation is called the superpoly of the cube, and it is much simpler than $f(\mathbf{k}, \mathbf{v})$. By analyzing superpolys, some information about the secret variables can be achieved.

However, due to the complex structure of $f(\mathbf{k}, \mathbf{v})$, recovering the superpoly of given a cube used to be very difficult. In the original paper [DS09], a linearity test was proposed to verify linear superpolys. If the superpoly is tested to be linear, then the algebraic normal form (ANF) of this superpoly can be recovered. Later, a quadraticity test was introduced [PJ12], and was used to mount the attacks against Trivium in [FV14]. But since all these attacks were experimental cryptanalysis, there is a probability that the linearity and quadraticity tests fail.

Division Property: A breakthrough came after the division property was proposed in [Tod15, TM16]. Todo et al. revealed the relation between integral cryptanalysis [KW02] and the division property, while integral cryptanalysis was also assumed to be equivalent to square attacks [DKR97] or higher-order differential attacks [Lai94]. Division property is an effective tool to analyze the monomials in $f(\mathbf{k}, \mathbf{v})$, but the computation is very expensive. Firstly, breadth-first search algorithms were used, but these algorithms often required unacceptable time and space. Consequently, Mixed Integer Linear Programming (MILP) approach was introduced to model the propagations of the bit-based division property [XZBL16]. Thanks to the efficiency of MILP tools, the bit-based division property for six lightweight block ciphers was shown. Since then, more improvements were given in [SWW17, TIHM17, WHT$^+$18]. However, traditional division property often suffers from the accuracy problem.

Taking the method proposed in [TIHM17] for example, if there is no division trail such that the output is 1, one can confirm that some specific monomial must not appear in the superpoly with absolute confidence. But if there is such a division trail, the corresponding monomial may or may not appear. The inaccuracy of the division trail method makes many previous key-recovery attacks, e.g., [FWDM18, WHT$^+$18, YT19, WHG$^+$19, YLL19], degenerated to distinguishing attacks [HLM$^+$20]. To solve this inaccuracy problem, three-subset division property was used. The first try was a variant of the three-subset division property [HW19], and later, Wang et al. proposed a new MILP based method to model the propagations of three-subset division property [WHG$^+$19]. But their methods were still inaccurate. This inaccuracy problem was finally resolved by Hao et al. in [HLM$^+$20], where the model for three-subset division property without unknown subset was proposed.

Trivium : Trivium is a bit oriented synchronous stream cipher based on NLSFR [DCP08]. It is one of the eSTREAM hardware-oriented finalists and an International Standard under ISO/IEC 29192-3:2012. Trivium has attracted many attentions due to its simple structure and high level security. Trivium uses a 80-bit key and a 80-bit IV, so the complexity of exhaustive search for the key is 2^{80} evaluations, and any attack of Trivium must has a lower complexity than 2^{80}.

For practical attacks against Trivium, a key-recovery attack on 784-round Trivium was given in [FV14]. Recently, a practical attack against 805-round Trivium was proposed in [YT20], and many linear superpolys were found in practical time. Attacks against higher rounds of Trivium are still theoretical. An attack on 855-round Trivium was reported in [FWDM18], but some flaws were proposed later after this paper was published. The doubt was finally verified in [HLM+20], where the authors showed this attack did not work. Before this paper, the key-recovery attack against the highest round Trivium was given in [HLM+20]. By using three-subset division property without unknown subset, Hao et al. presented key-recovery attacks on 840-, 841-, and 842-round Trivium. Consequently, Hu et al. pointed out in ASIACRYPT 2020 that computing three-subset division property without unknown subset was equivalent to calculating the appearing monomials in the superpolys. They used a divide-and-conquer strategy to speed up their computations, and hence, obtained more key-recovery attacks on 840-, 841-, and 842-round Trivium with even lower complexities [HSWW20]. An early version of this paper presented, as far as we know, the first key-recovery attack against 843-round Trivium in [Sun21].

Kreyvium: Kreyvium is designed for the use of fully Homomorphic encryption [CCF+16]. It has a similar structure to Trivium but consists of more registers. The sizes of its key and IV are both 128. The previous highest round of Kreyvium that can be attacked is 892, which was proposed in [HLM+20].

Motivation: The key step of the cube attack is to recover the superpoly of a given cube. As this problem was finally settled in [HLM+20], the next crucial problem is how to find "good" superpolys. Since the superpolys are determined by the cubes, this problem becomes how to find "good" cubes. Our goal is to develop an efficient algorithm to search for "good" cubes such that we can improve current cube attacks.

A straightforward way of searching for "good" cubes is as follows. A cube is selected first, and its superpoly is then recovered. If this superpoly is "good", then it is done; otherwise, another cube is chosen and the above procedure is repeated. Clearly, this straightforward method is not efficient. Firstly, there are enormous cubes, but not all of them are "good"; secondly, it is very costly to verify whether a cube is "good", particularly when the number of rounds of a stream cipher is large.

To improve this straightforward method, on one hand, we realized that, when many cubes are to be tested, most computations for recovering their superpolys are duplicated. However, if we deal with a set of cubes together, then the common computations will be done just once, which will save much computing time. On the other hand, to verify "good" cubes efficiently, we define a special kind of "good" cubes first. We say a cube is **valuable** if the superpoly of this cube has a balanced secret variable, where we say a variable x_i is balanced in a polynomial, if x_i appears as a 1-degree monomial and any other monomials of this polynomial do not involve x_i, i.e., the superpoly p has the form $p(x_0, x_1, \ldots, x_{n-1}) = x_i + p'(x_0, \ldots, x_{i-1}, x_{i+1}, \ldots, x_{n-1})$. A valuable cube is a "good" cube, because its superpoly brings in 1 bit of information about the key. More importantly, the superpolys of valuable cubes can not only improve *theoretically* attacks, but also improve the *practical* analyses. To pick up valuable cubes from enormous candidate cubes, we observed that it may be very costly to prove a cube is valuable, which usually needs to recover most monomials of its superpoly, but there are many ways of showing that a cube is *not* valuable. For example, if we know some high degree monomial appears in the superpoly of a cube, then the secret variables involved in this monomial cannot be balanced. Moreover, the cube is not valuable if all secret variables are not balanced in the superpoly of this cube. Note that it is also very expensive to check whether a monomial appears in the superpoly. Fortunately, we have enormous cubes and we only need to find a few valuable ones. This means we can reject useless cubes in a more aggressive way, i.e., we can regard a secret variable as not balanced in the superpoly of a cube, if a high degree monomial involving this secret variable *probably* appears in the superpoly. Although this aggressive approach may reject many valuable cubes, the un-rejected cubes tend to be valuable ones with high probabilities.

Our contribution: We proposed a heuristic method of rejecting useless cubes. This method is based on the divide-and-conquer strategy and an important observation. We observed that if the whole system, which is constructed to recover a superpoly, can be split into several subsystems through the divide-and-conquer strategy, characteristics of the subsystems are likely to coincide with those of the whole system, if the number of subsystems that have solutions is not large. Thus, we can reject many useless cubes only via the information obtained from subsystems. With the method, we devised a new efficient algorithm to search for valuable cubes. Instead of testing cubes one by one, the new algorithm deals with a set of cubes together. Although many useless cubes will be rejected during the computation, the remaining ones are likely to be valuable cubes. To verify the effectiveness of the proposed algorithm, we applied it to two famous stream ciphers, Trivium and Kreyvium, and obtained three improvements.

1. We obtained the first key-recovery attack against 843-round Trivium, while the previous highest round that can be attacked was 842, given in [HLM+20].

   The new algorithm successfully found many valuable cubes for 840-, 841-, and 842-round Trivium. Particularly, we got two valuable cubes for 843-round Trivium. A summary of the results is shown in Table 1, and the details are presented in Section 4.1. In this table, the data in the first row means that, for 840-round Trivium, we found 222 valuable cubes. The dimensions of these cubes are all 78, and the secret variable k_0 is balanced in all the corresponding superpolys.

Table 1: The numbers of valuable cubes for 840-, 841-, 842-, and 843-round Trivium.

Round	Balanced secret variable	#Valuable cubes	Dim of cubes
840	k_0	222	78
	k_1	215	78
	k_2	134	78
841	k_0	2	78
	k_1	42	78
842	k_1	5	78
843	k_2	2	78

   The superpoly of a valuable cube for 843-round Trivium was recovered, and it consists of 16 561 monomials. For a key-recovery attack against 843-round Trivium, we need 2^{78} evaluations of 843-round Trivium to gain the specific value of the superpoly. Using this superpoly, we can filter out 2^{79} impossible keys, and the complexity of exhaustive search becomes 2^{79}. In this way, the overall complexity of our approach is lower than 2^{80}.

2. We found many new small valuable cubes to perform practical key-recovery attacks against 808-round Trivium, while the previous highest round of Trivium that can be attacked practically was 805, given in [YT20].

   To do practical attacks, we need many valuable cubes instead of one. To reduce the number of requests in the online phase, like done in [YT20], we preset a set of indexes of public variables, say S, and we only searched for valuables cubes that are from the subsets of S. The new algorithm successfully found many small valuable cubes for 806- and 808-round Trivium, and we also recovered all their superpolys. Although these superpolys are mostly nonlinear, we can still perform practical key-recovery attacks by delicately selecting the variables for enumeration.

   As shown in Table 2, the data in the first row means that, for 806-round Trivium, we preset a set S_a, which contains indexes of 39 public variables. We totally found 29 valuable cubes, and the dimensions of these cubes are $34 \sim 37$. Details about the cubes and $\{S_a, S_b\}$ are presented in Section 4.2.

Table 2: The numbers of small valuable cubes for 806- and 808-round Trivium.

Round	Preset Set	Size of preset set	#valuable cubes	Dim of cubes
806	S_a	39	29	$34 \sim 37$
808	S_b	44	37	$39 \sim 41$

For 806-round Trivium, the authors of [YT20] found 16 linear superpolys, and hence, they could recover 16 key bits with $2^{38.64}$ requests. They used a brute-force attack to recover the remaining 64 key bits, so the overall online complexity is $2^{64} + 2^{38.64}$, which cannot be done practically. We used the preset set S_a, whose size is 39. By the new algorithm, we found 29 new valuable cubes, which are all from the subsets of S_a. The values of the corresponding 29 superpolys can be obtained with 2^{39} requests. Thus, by using the 16 key bits from [YT20] and selecting another 35 variables for enumeration delicately, we could recover the remaining 29 key bits directly. The overall complexity reduces to $2^{39} + 2^{38.64} + 2^{35}$, which can be done practically.

For 808-round Trivium, we preset a set S_b, whose size is 44. We searched for valuable cubes from the subsets of S_b, and finally got 37 valuable ones. By delicately selecting 43 variables for enumeration, we can deduce the values of the remaining 37 secret variables within constant time. The total online complexity for attacking 808-round Trivium is $2^{44} + 2^{43}$, which can be done practically as well.

3. Based on the cube used to attack 892-round Kreyvium in [HLM+20], we found more valuable cubes and mounted the key-recovery attacks against Kreyvium to 893-round.

   For 892-round Kreyvium, Hao et al. used a cube of dimension 115 and obtained a linear superpoly. The secret k_{26} is balanced in this superpoly. We enlarged their cube to a set with 117 indexes, and set it as a preset set. We searched for valuable cubes from the subsets of this preset set, requiring k_{26} being balanced in the superpolys. We finally obtained 476 valuable cubes, including the one used by Hao et al. Besides, one superpoly of these valuable cubes can also be used to lower the complexity for attacking 892-round Kreyvium. Details come in Section 4.3.

   For 893-round Kreyvium, we used the same preset set as that for attacking 892-round Kreyvium, and searched for valuable cubes whose balanced variables include k_{26}. Finally, we obtained 1 valuable cube and recovered its superpoly. Thus, the overall complexity of the key-recovery attack is $2^{127} + 2^{115}$, which is a bit smaller than the exhaustive search complexity 2^{128}. We believe this complexity can be further lowered by searching for more valuable cubes, and we also think valuable cubes can be found in high rounds of Kreyvium as well.

The source codes of the proposed algorithm, including those searching for valuable cubes and retrieving superpolys, were released at `https://github.com/ysun0102/searchforcubes`.

This paper is organized as follows. Section 2 introduces necessary notations and some preliminaries. The new search algorithm is reported in Section 3, and its applications to Trivium and Kreyvium are in Section 4. We conclude this paper in Section 5.

2 Preliminaries

2.1 Notations

Let $\mathbb{F}_2[\mathbf{x}]$ be the polynomial ring over the field $\mathbb{F}_2 = \{0, 1\}$ in variables $\mathbf{x} = (x_0, x_1, \ldots, x_{n-1})$. Given a bit-vector $\mathbf{u} = (u_0, u_1, \ldots, u_{n-1}) \in \mathbb{F}_2^n$, $\mathbf{x}^{\mathbf{u}} = \prod_{i=0}^{n-1} x_i^{u_i}$ is called a **monomial** in

$\mathbb{F}_2[\mathbf{x}]$, and $a_\mathbf{u}\mathbf{x}^\mathbf{u}$ is called a **term** where $a_\mathbf{u} \in \mathbb{F}_2$. A **polynomial** is a sum of finite terms. To avoid confusion, we use bold letters to represent bit-vectors in this paper. Besides, we always use $\mathbf{x}$ to represent unknowns or inputs to a function, and use $\mathbf{y}$ to represent the outputs of a vectorial function. Secret and public variables are denoted by $\mathbf{k}$ and $\mathbf{v}$. And $\mathbf{u}$ and $\mathbf{w}$ are always used as the powers of monomials.

The **support** of a bit-vector $\mathbf{u} = (u_0, u_1, \ldots, u_{n-1}) \in \mathbb{F}_2^n$ is defined as the set $\mathrm{Supp}(\mathbf{u}) = \{i \mid u_i = 1\}$, and the **weight** of $\mathbf{u}$ is defined as $\mathrm{wt}(\mathbf{u}) = |\mathrm{Supp}(\mathbf{u})|$, where $|\{\cdot\}|$ is the cardinality of the set $\{\cdot\}$. Given a monomial $\mathbf{x}^\mathbf{u}$, its **degree** is defined by $\mathrm{wt}(\mathbf{u})$.

Let $f : \mathbb{F}_2^n \to \mathbb{F}_2$ be a Boolean function, then its algebraic normal form (ANF) can be represented as a polynomial in the ring $\mathbb{F}_2[\mathbf{x}]$: $\hat{f} = \sum_{\mathbf{u}\in\mathbb{F}_2^n} a_\mathbf{u}\mathbf{x}^\mathbf{u}$, where $a_\mathbf{u} \in \mathbb{F}_2$. For an ANF $\hat{f}$, we say a monomial $\mathbf{x}^\mathbf{u}$ **appears** in $\hat{f}$, if the coefficient of $\mathbf{x}^\mathbf{u}$ in $\hat{f}$ is 1, i.e., $a_\mathbf{u} = 1$, and we denote $\mathbf{x}^\mathbf{u} \to \hat{f}$; otherwise, we denote $\mathbf{x}^\mathbf{u} \not\to \hat{f}$.

In Section 3, we will use an algebraic manner to describe the search algorithm, and we will recover the superpolys by solving a system of constraints, where the constraints could be algebraic equations, conjunctive normal forms, or integer inequalities. Let $F(\mathbf{x})$ be a set of constraints, where $\mathbf{x}$ are unknowns. We use $F(\mathbf{x} \mid \textit{constraints})$ to denote the system after adding some new constraints to $F(\mathbf{x})$. The solutions to the system $F(\mathbf{x})$ and $F(\mathbf{x} \mid \textit{constraints})$ are represented by $\mathrm{Sol}(F)$ and $\mathrm{Sol}(F \mid \textit{constraints})$ respectively. For example, $\mathrm{Sol}(F \mid \mathrm{wt}(\mathbf{x}) = 2)$ is the set of $\mathbf{x}$ such that $\mathbf{x}$ is a solution to $F(\mathbf{x})$ and $\mathrm{wt}(\mathbf{x}) = 2$. Note that we always have $\mathrm{Sol}(F \mid \textit{constraints}) \subseteq \mathrm{Sol}(F)$.

2.2 Cube attack

Cube attack was proposed in EUROCRYPT 2009 [DS09]. For a cipher with n secret variables and m public variables, each output bit of this cipher can be represented as a polynomial in secret and public variables. Denote $\mathbf{k} = (k_0, k_1, \ldots, k_{n-1})$ and $\mathbf{v} = (v_0, v_1, \ldots, v_{m-1})$ as the secret and public variables respectively, where $k_i, v_j \in \mathbb{F}_2$ for $0 \le i < n$ and $0 \le j < m$. Thus, one output bit can be written as f, where f is a polynomial in the ring $\mathbb{F}_2[\mathbf{k}, \mathbf{v}]$ for $0 \le i < n$ and $0 \le j < m$.

Let $I \subseteq \{0, 1, \ldots, m-1\}$ be a set of indices of public variables. A **cube** determined by I is denoted as C_I, and contains all $2^{|I|}$ possible combinations of the values of v_j's for $j \in I$, while the value of $v_{j'}$ remains unchanged for $j' \in \{0, 1, \ldots, m-1\} \setminus I$. Then we have the following equation:

$$\sum_{C_I} f = \sum_{C_I} (t_I \cdot p + q) = \sum_{C_I} t_I \cdot p + \sum_{C_I} q = p,$$

where t_I represents the product $\prod_{i\in I} v_i$, and there is no term of q divisible by t_I. The polynomial p is called the **superpoly** of the cube C_I. By the above definitions, the superpoly of C_I only involves the inactive public variables $v_{j'}$ where $j' \in \{0, 1, \ldots, m-1\} \setminus I$, and the values of these variables are often preset to constants in cube attacks. So the superpoly p is actually a polynomial in $\mathbb{F}_2[\mathbf{k}]$.

For a polynomial $p \in \mathbb{F}_2[\mathbf{k}]$, we say p has a **balanced** variable k_i where $0 \le i < n$, if the coefficient of k_i in f is 1, and the other monomials in f do not involve k_i. If a polynomial f has a balanced variable, then f is balanced, i.e., $|\{\mathbf{k} \mid f(\mathbf{k}) = 0\}| = |\{\mathbf{k} \mid f(\mathbf{k}) = 1\}| = 2^{n-1}$. We say a cube is **valuable** if its superpoly has a balanced variable.

A valuable cube C_I may lead to a key-recovery attack against the cipher system represented by f. In the *offline* phase of cube attack, attackers recover the superpoly p of C_I. Next, in the *online* phase, attackers get the value a of p by querying the encryption oracle $2^{|I|}$ times. Since the superpoly is balanced, 2^{n-1} illegal keys will be filtered out by the equation $p = a$. To recover the whole key, it suffices to query the encryption oracle another 2^{n-1} times. And the overall complexity of this attack is $2^{|I|} + 2^{n-1}$.

Note that more balanced superpolys may filter out more illegal keys, but the complexity of obtaining the values of these superpolys increases. However, the costs of calculating these values may be lowered in a special case. That is, if there are several cubes and their indexes are all from the subsets of a set S, then it only needs $2^{|S|}$ requests to calculate all the values of the superpolys. This technique was used in [YT20] to obtain practical cube attacks against round-reduced Trivium. We also use this technique in Section 4.2.

2.3 Trivium

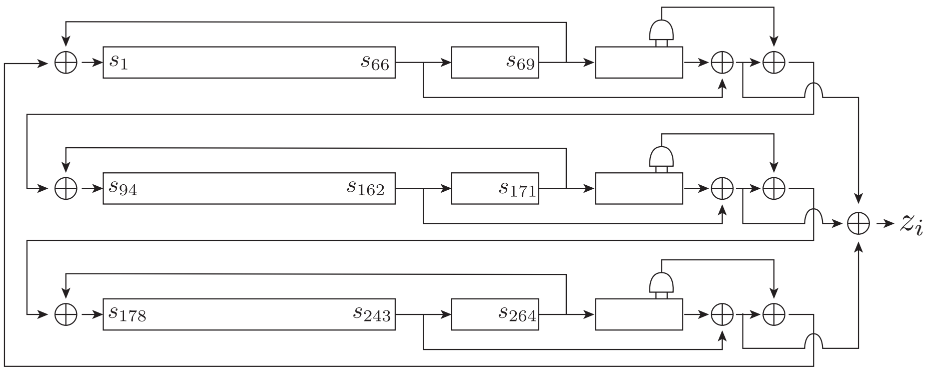

Figure 1: Structure of Trivium

Trivium is an NLFSR-based stream cipher [DCP08]. As shown in Figure 1, Trivium has a 288-bit internal state $(s_1, s_2, \ldots, s_{288})$ which is divided into three registers. The 80-bit secret key $\mathbf{k} = (k_0, k_1, \ldots, k_{79})$ is loaded into the first register, and the 80-bit initialization vector $\mathbf{v} = (v_0, v_1, \ldots, v_{79})$ is loaded into the second register. The other bits of the state are set to 0 except the last three bits in the third register. That is, we have

$$\begin{aligned}
(s_1, s_2, \ldots, s_{93}) &\leftarrow (k_0, k_1, \ldots, k_{79}, 0, \ldots, 0),\\
(s_{94}, s_{95}, \ldots, s_{177}) &\leftarrow (v_0, v_1, \ldots, v_{79}, 0, \ldots, 0),\\
(s_{178}, s_{179}, \ldots, s_{288}) &\leftarrow (0, 0, \ldots, 0, 1, 1, 1).
\end{aligned}$$

The state of Trivium is updated in the following way:

$$\begin{aligned}
t_1 &\leftarrow s_{66} + s_{93},\\
t_2 &\leftarrow s_{162} + s_{177},\\
t_3 &\leftarrow s_{243} + s_{288},\\
z &\leftarrow t_1 + t_2 + t_3,\\
t_1 &\leftarrow t_1 + s_{91} \cdot s_{92} + s_{171},\\
t_2 &\leftarrow t_2 + s_{175} \cdot s_{176} + s_{264},\\
t_3 &\leftarrow t_3 + s_{286} \cdot s_{287} + s_{69},\\
(s_1, s_2, \ldots, s_{93}) &\leftarrow (t_3, s_1, \ldots, s_{92}),\\
(s_{94}, s_{95}, \ldots, s_{177}) &\leftarrow (t_1, s_{94}, \ldots, s_{176}),\\
(s_{178}, s_{179}, \ldots, s_{288}) &\leftarrow (t_2, s_{178}, \ldots, s_{287}),
\end{aligned}$$

where z denotes the 1-bit key stream. The state is updated 1 152 times without producing an output. After the key initialization is done, one bit key stream is produced by every update function.

2.4 Monomial prediction

The concept of *monomial prediction* was proposed in [HSWW20], to resolve the problem of determining the presence or absence of a monomial in the algebraic normal form of a Boolean function. In our opinion, the monomial prediction technique is in fact an algebraic representation of the *three-subset division property without unknown subset* technique proposed in [HLM+20], because the codes provided by both authors are identical. We prefer the concept of monomial prediction, as we would like to describe our search algorithm in an algebraic manner. So we rewrite the monomial prediction technique below.

Let $\mathbf{f} : \mathbb{F}_2^n \to \mathbb{F}_2^m$ be a vectorial Boolean function, mapping $\mathbf{x} = (x_0, x_1, \ldots, x_{n-1}) \in \mathbb{F}_2^n$ to $\mathbf{y} = (y_0, y_1, \ldots, y_{m-1}) \in \mathbb{F}_2^m$. Then $\mathbf{y}^{\mathbf{w}}$ can be represented as a Boolean polynomial with respect to $\mathbf{x}$, and we are interested in whether $\mathbf{x}^{\mathbf{u}} \to \mathbf{y}^{\mathbf{w}}$, where $\mathbf{u} \in \mathbb{F}_2^n$ and $\mathbf{w} \in \mathbb{F}_2^m$. Generally, a vectorial Boolean function equals to a composition of basic vectorial Boolean functions. We present the propagation rules of the *copy*, *AND*, and *XOR* functions below.

Rule 1 (copy): Let $\mathbf{f}_{copy} : \mathbb{F}_2^n \to \mathbb{F}_2^{n+1}$ be a *copy* function, s.t.

$$(y_0, y_1, \ldots, y_n) = \mathbf{f}_{copy}(x_0, x_1, \ldots, x_{n-1}) = (x_0, x_0, x_1, \ldots, x_{n-1}).$$

For a given bit-vector $\mathbf{u} \in \mathbb{F}_2^n$, we define that $\mathbf{u}$ propagates to $\mathbf{w} \in \mathbb{F}_2^{n+1}$ under the *copy* function, or equivalently $\mathbf{u} \Rightarrow_{copy} \mathbf{w}$, by the following rule:

$$\begin{cases} (0, u_1, \ldots, u_{n-1}) \Rightarrow_{copy} (0, 0, u_1, \ldots, u_{n-1}), \\ (1, u_1, \ldots, u_{n-1}) \Rightarrow_{copy} (1, 0, u_1, \ldots, u_{n-1}) \text{ or } (0, 1, u_1, \ldots, u_{n-1}) \text{ or } (1, 1, u_1, \ldots, u_{n-1}). \end{cases}$$

It is easy to check that if $\mathbf{u} \Rightarrow_{copy} \mathbf{w}$, we have $\mathbf{x}^{\mathbf{u}} \to \mathbf{y}^{\mathbf{w}}$.

Rule 2 (AND): Let $\mathbf{f}_{AND} : \mathbb{F}_2^n \to \mathbb{F}_2^{n-1}$ be an *AND* function, s.t.

$$(y_0, y_1, \ldots, y_{n-2}) = \mathbf{f}_{AND}(x_0, x_1, \ldots, x_{n-1}) = (x_0 \cdot x_1, \ldots, x_{n-1}).$$

For a given bit-vector $\mathbf{u} \in \mathbb{F}_2^n$, we define that $\mathbf{u}$ propagates to $\mathbf{w} \in \mathbb{F}_2^{n-1}$ under the *AND* function, or equivalently $\mathbf{u} \Rightarrow_{AND} \mathbf{w}$, by the following rule:

$$\begin{cases} (0, 0, u_2, \ldots, u_{n-1}) \Rightarrow_{AND} (0, u_2, \ldots, u_{n-1}), \\ (1, 1, u_2, \ldots, u_{n-1}) \Rightarrow_{AND} (1, u_2, \ldots, u_{n-1}). \end{cases}$$

Again, if $\mathbf{u} \Rightarrow_{AND} \mathbf{w}$, we have $\mathbf{x}^{\mathbf{u}} \to \mathbf{y}^{\mathbf{w}}$.

Rule 3 (XOR): Let $\mathbf{f}_{XOR} : \mathbb{F}_2^n \to \mathbb{F}_2^{n-1}$ be an *XOR* function, s.t.

$$(y_0, y_1, \ldots, y_{n-2}) = \mathbf{f}_{XOR}(x_0, x_1, \ldots, x_{n-1}) = (x_0 + x_1, \ldots, x_{n-1}).$$

For a given bit-vector $\mathbf{u} \in \mathbb{F}_2^n$, we define that $\mathbf{u}$ propagates to $\mathbf{w} \in \mathbb{F}_2^{n-1}$ under the *XOR* function, or equivalently $\mathbf{u} \Rightarrow_{XOR} \mathbf{w}$, by the following rule:

$$\begin{cases} (0, 0, u_2, \ldots, u_{n-1}) \Rightarrow_{XOR} (0, u_2, \ldots, u_{n-1}), \\ (0, 1, u_2, \ldots, u_{n-1}) \Rightarrow_{XOR} (1, u_2, \ldots, u_{n-1}), \\ (1, 0, u_2, \ldots, u_{n-1}) \Rightarrow_{XOR} (1, u_2, \ldots, u_{n-1}). \end{cases}$$

Similarly, $\mathbf{u} \Rightarrow_{XOR} \mathbf{w}$ implies $\mathbf{x}^{\mathbf{u}} \to \mathbf{y}^{\mathbf{w}}$.

If a vectorial Boolean function, say $\mathbf{f}$, can be represented as a composition of the *copy, AND, and XOR* functions, for a given $\mathbf{x}^{\mathbf{u}}$, we can theoretically obtain the set of all possible $\mathbf{y}^{\mathbf{w}}$ such that $\mathbf{f}(\mathbf{x}) = \mathbf{y}$ and $\mathbf{u} \Rightarrow_{\mathbf{f}} \mathbf{w}$ by applying the above rules repeatedly. However, the set of all possible $\mathbf{w}$ is often too large to be computed. Fortunately, it is not necessary to compute the whole set, and instead, we are only interested in finding the bit-vectors $\mathbf{u} \in \mathbb{F}_2^n$, such that $\mathbf{u} \Rightarrow_{\mathbf{f}} \bar{\mathbf{w}}$ where $\bar{\mathbf{w}}$ is a pre-given specific bit-vector in $\mathbb{F}_2^m$. For this goal, the bit-vector $\mathbf{u}$ is set as unknowns, and a system of the (in)equations can be built with respect to $\mathbf{f}$ and $\bar{\mathbf{w}}$. Finally, we can obtain all desired $\mathbf{u}$ by solving this system.

Example 1. Let $\mathbf{x} = (x_0, x_1)$ and $\mathbf{y} = \mathbf{f}(\mathbf{x}) = (x_0x_1 + x_1, x_0)$, then $\mathbf{f}$ can be decomposed into the following procedure

$$(x_0, x_1) \Rightarrow_{copy} (x_0, x_1, x_0) \Rightarrow_{copy} (x_0, x_1, x_1, x_0) \Rightarrow_{XOR} (x_0+x_1, x_1, x_0) \Rightarrow_{AND} (x_0x_1+x_1, x_0).$$

Consider $\mathbf{u} = (0, 1)$, then we have $\mathbf{x}^{\mathbf{u}} = x_1$ and $(0, 1)$ propagates in the following way

$$\begin{aligned}(0,1) &\Rightarrow_{copy} \{(0,1,0)\} \Rightarrow_{copy} \{(0,1,0,0),(0,0,1,0),(0,1,1,0)\}\\ &\Rightarrow_{XOR} \{(1,0,0),(0,1,0),(1,1,0)\} \Rightarrow_{AND} \{(1,0)\}.\end{aligned} \tag{1}$$

Then we have $(0, 1) \Rightarrow_{\mathbf{f}} (1, 0)$. However, the propagations of $(1, 1)$ are more complicated because of the two copy rules.

Note that we have $\mathbf{y}^{\bar{\mathbf{w}}}$ in $\mathbb{F}_2$, so $\mathbf{y}^{\bar{\mathbf{w}}}$ represents a Boolean function with respect to $\mathbf{x}$. We denote this function by f which is from $\mathbb{F}_2^n$ to $\mathbb{F}_2$. We are interested in which monomial appears in the ANF of f, i.e., the set of $\mathbf{x}^{\mathbf{u}}$ such that $\mathbf{x}^{\mathbf{u}} \to \hat{f}$, where $\hat{f}$ is the ANF of f and $\hat{f} = \mathbf{y}^{\bar{\mathbf{w}}}$. But please remark that, $\mathbf{u} \Rightarrow_{\mathbf{f}} \mathbf{w}$ does not always imply $\mathbf{x}^{\mathbf{u}} \to \mathbf{y}^{\mathbf{w}}$, if the function $\mathbf{y} = \mathbf{f}(\mathbf{x})$ is not one of the basic *copy, AND, and XOR* functions. This is because there are more than one possible propagating ways in the *copy* rule, such that $\mathbf{u}$ can propagate to $\mathbf{w}$ through several different ways. To get a definite answer to whether $\mathbf{x}^{\mathbf{u}} \to \hat{f}$ holds, like division trails, Hu et al. defined the concept of monomial trails.

Definition 1. Let $\mathbf{x}^{(i+1)} = \mathbf{f}^{(i)}(\mathbf{x}^{(i)})$ for $0 \leq i < r$, and denote $\pi_{\mathbf{u}}(\mathbf{x}) = \mathbf{x}^{\mathbf{u}}$ for simplification. A sequence of monomials $(\pi_{\mathbf{u}^{(0)}}(\mathbf{x}^{(0)}), \pi_{\mathbf{u}^{(1)}}(\mathbf{x}^{(1)}), \dots, \pi_{\mathbf{u}^{(r)}}(\mathbf{x}^{(r)}))$ is called an r-round monomial trail connecting $\pi_{\mathbf{u}^{(0)}}(\mathbf{x}^{(0)})$ and $\pi_{\mathbf{u}^{(r)}}(\mathbf{x}^{(r)})$ with respect to the composite function $\mathbf{f} = \mathbf{f}^{(r-1)} \circ \mathbf{f}^{(r-2)} \circ \cdots \circ \mathbf{f}^{(0)}$, if

$$\pi_{\mathbf{u}^{(0)}}(\mathbf{x}^{(0)}) \to \pi_{\mathbf{u}^{(1)}}(\mathbf{x}^{(1)}) \to \cdots \to \pi_{\mathbf{u}^{(r)}}(\mathbf{x}^{(r)}).$$

If there is at least one monomial trail connecting $\pi_{\mathbf{u}^{(0)}}(\mathbf{x}^{(0)})$ and $\pi_{\mathbf{u}^{(r)}}(\mathbf{x}^{(r)})$, we write $\pi_{\mathbf{u}^{(0)}}(\mathbf{x}^{(0)}) \rightsquigarrow \pi_{\mathbf{u}^{(r)}}(\mathbf{x}^{(r)})$; otherwise, $\pi_{\mathbf{u}^{(0)}}(\mathbf{x}^{(0)}) \not\rightsquigarrow \pi_{\mathbf{u}^{(r)}}(\mathbf{x}^{(r)})$.

Definition 2. For $\mathbf{f}$ with a specific composition sequence, the monomial hull of $\pi_{\mathbf{u}^{(0)}}(\mathbf{x}^{(0)})$ and $\pi_{\mathbf{u}^{(r)}}(\mathbf{x}^{(r)})$, denoted by $\pi_{\mathbf{u}^{(0)}}(\mathbf{x}^{(0)}) \bowtie \pi_{\mathbf{u}^{(r)}}(\mathbf{x}^{(r)})$, is the set of all monomial trails connecting them. The number of trails in the monomial hull is called the size of the hull and is denoted by $|\pi_{\mathbf{u}^{(0)}}(\mathbf{x}^{(0)}) \bowtie \pi_{\mathbf{u}^{(r)}}(\mathbf{x}^{(r)})|$.

Using the concepts of monomial trail and monomial hull, we can determine the presence or the absence of a monomial by the following proposition.

Proposition 1 (Proposition 1 in [HSWW20])**.** $\pi_{\mathbf{u}^{(0)}}(\mathbf{x}^{(0)}) \to \pi_{\mathbf{u}^{(r)}}(\mathbf{x}^{(r)})$ *if and only if* $|\pi_{\mathbf{u}^{(0)}}(\mathbf{x}^{(0)}) \bowtie \pi_{\mathbf{u}^{(r)}}(\mathbf{x}^{(r)})|$ *is odd.*

In Example 1, Equation (1) implies a monomial trail connecting $\mathbf{x}^{(0,1)}$ and $\mathbf{y}^{(1,0)}$, and it is not difficult to see that this is the only one monomial trail in the monomial hull of $\mathbf{x}^{(0,1)}$ and $\mathbf{y}^{(1,0)}$, so by Proposition 1, we must have $x_1 = \mathbf{x}^{(0,1)} \to \mathbf{y}^{(1,0)} = x_0x_1 + x_1$. For another example, if we want to check whether $\mathbf{x}^{(1,1)} = x_0x_1$ appears in $\mathbf{y}^{(1,1)} = (x_0x_1 + x_1)x_0 = 0$, we need to compute all monomial trails connecting $\mathbf{x}^{(1,1)}$ and $\mathbf{y}^{(1,1)}$, and the monomial hull of $\mathbf{x}^{(1,1)}$ and $\mathbf{y}^{(1,1)}$ consists of the following trails

$$(x_0, x_1)^{(1,1)} \to (x_0, x_1, x_0)^{(0,1,1)} \to (x_0, x_1, x_1, x_0)^{(0,1,1,1)} \to (x_0+x_1, x_1, x_0)^{(1,1,1)} \to (x_0x_1+x_1, x_0)^{(1,1)},$$

$$(x_0, x_1)^{(1,1)} \to (x_0, x_1, x_0)^{(1,1,1)} \to (x_0, x_1, x_1, x_0)^{(1,0,1,1)} \to (x_0+x_1, x_1, x_0)^{(1,1,1)} \to (x_0x_1+x_1, x_0)^{(1,1)}.$$

Since the size of the hull is 2, by Definition 1, we have $x_0x_1 = \mathbf{x}^{(1,1)} \not\to \mathbf{y}^{(1,1)} = (x_0x_1 + x_1)x_0 = 0$.

Thus, to check where $\mathbf{x}^{\mathbf{u}}$ appears in $\hat{f}$, we only need to compute all monomial trials connecting $\mathbf{x}^{\mathbf{u}}$ and $\hat{f}$. Only if the number of these monomial trails is odd, we have $\mathbf{x}^{\mathbf{u}} \to \hat{f}$. For computing monomial trails, an MILP model can be built and solved by Gurobi [GO21]. Each feasible solution to this model corresponds exactly to one monomial trail.

3 A search algorithm for valuable cubes

The search algorithm is based on the divide-and-conquer strategy and an observation, so we introduce these two parts in Section 3.1 and 3.2, respectively. Then the search algorithm comes in Section 3.3.

3.1 A divide-and-conquer algorithm for recovering superpolys

The divide-and-conquer strategy has been used for recovering superpolys in [HLM+20, HSWW20]. Bigger systems were divided into smaller subsystems to speed up the computations. Our algorithm also needs the divide-and-conquer strategy, because we hope to obtain the characteristics from the small subsystems instead of the whole large system. We first present an algebraic description of the divide-and-conquer algorithm, and then give a specialization of this algorithm for recovering superpolys for Trivium.

A stream cipher is usually constructed in an iterative manner. Let $\mathbf{x} \in \mathbb{F}_2^n$ be the initial state and $\mathbf{y} \in \mathbb{F}_2^m$ be the final state, then we have $\mathbf{y} = \mathbf{f}^{(r-1)} \circ \mathbf{f}^{(r-2)} \circ \cdots \circ \mathbf{f}^{(0)}(\mathbf{x})$, where $\mathbf{f}^{(i)}$ are the *copy*, *AND*, or *XOR* functions for $i = 0, 1, \ldots, r-1$. Thus, denote $\mathbf{x}^{(i+1)} = \mathbf{f}^{(i)}(\mathbf{x}^{(i)})$ for $0 \le i < r$, if we have $\mathbf{u}^{(i)} \Rightarrow_{\mathbf{f}^{(i)}} \mathbf{u}^{(i+1)}$, then we have $\pi_{\mathbf{u}^{(i)}}(\mathbf{x}^{(i)}) \to \pi_{\mathbf{u}^{(i+1)}}(\mathbf{x}^{(i+1)})$, where $\pi_{\mathbf{u}}(\mathbf{x}) = \mathbf{x}^{\mathbf{u}}$. Thus, we have the following two sequences,

$$\pi_{\mathbf{u}^{(0)}}(\mathbf{x}) = \pi_{\mathbf{u}^{(0)}}(\mathbf{x}^{(0)}) \to \pi_{\mathbf{u}^{(1)}}(\mathbf{x}^{(1)}) \to \cdots \to \pi_{\mathbf{u}^{(r)}}(\mathbf{x}^{(r)}) = \pi_{\mathbf{u}^{(r)}}(\mathbf{y}), \tag{2}$$

and

$$\mathbf{u}^{(0)} \Rightarrow_{\mathbf{f}^{(0)}} \mathbf{u}^{(1)} \Rightarrow_{\mathbf{f}^{(1)}} \cdots \Rightarrow_{\mathbf{f}^{(r)}} \mathbf{u}^{(r)}.$$

For a stream cipher, the output is often one bit, and can be represented as $\mathbf{y}^{\bar{\mathbf{w}}}$ for a constant bit-vector $\bar{\mathbf{w}} \in \mathbb{F}_2^m$. The bit $\mathbf{y}^{\bar{\mathbf{w}}}$ represents a Boolean function of $\mathbf{x}$, and its algebraic normal form $\hat{f}$ can be calculated by expanding $\mathbf{y}^{\bar{\mathbf{w}}}$ with respect to $\mathbf{x}$. The complete form of $\hat{f}$ is often too complicated to obtain, but it is possible to pry into parts of $\hat{f}$. Given a cube C_I, let t_I be the product of active public variables whose indexes are in I. To recover the superpoly of C_I, it suffices to find out all monomials $\mathbf{x}^{\mathbf{u}}$ appearing in $\hat{f}$ such that t_I divides $\mathbf{x}^{\mathbf{u}}$, or equivalently $I \subseteq \mathrm{Supp}(\mathbf{u})$. So to recover the superpoly of C_I we build the following system:

$$F(\mathbf{u}) := F_{0,r}(\mathbf{u} \mid I \subseteq \mathrm{Supp}(\mathbf{u}^{(0)}), \mathbf{u}^{(r)} = \bar{\mathbf{w}}), \tag{3}$$

where

$$F_{a,b}(\mathbf{u}) = \begin{cases} \mathbf{u}^{(a)} \Rightarrow_{\mathbf{f}^{(a)}} \mathbf{u}^{(a+1)}, \\ \mathbf{u}^{(a+1)} \Rightarrow_{\mathbf{f}^{(a+1)}} \mathbf{u}^{(a+2)}, \\ \cdots \\ \cdots \\ \mathbf{u}^{(r-1)} \Rightarrow_{\mathbf{f}^{(b-1)}} \mathbf{u}^{(b)}, \end{cases}$$

is a set of constraints, and $I \subseteq \mathrm{Supp}(\mathbf{u}^{(0)})$, $\mathbf{u}^{(r)} = \bar{\mathbf{w}}$ are two new constraints added to the system $F_{a,b}(\mathbf{u})$. The set of all solutions to $F_{a,b}(\mathbf{u})$ is denoted by $\mathrm{Sol}(F_{a,b})$, and similarly, $\mathrm{Sol}(F_{a,b} \mid \mathit{constraints})$ represents the set of solutions to the system $\mathrm{Sol}(F_{a,b} \mid \mathit{constraints})$.

If a solution in $\mathrm{Sol}(F_{0,r} \mid I \subseteq \mathrm{Supp}(\mathbf{u}^{(0)}), \mathbf{u}^{(r)} = \bar{\mathbf{w}}, \mathbf{u}^{(0)} = \bar{\mathbf{u}})$ is found, we obtain a monomial trail $\pi_{\bar{\mathbf{u}}}(\mathbf{x}) \rightsquigarrow \pi_{\bar{\mathbf{w}}}(\mathbf{y})$ by Equation (2). For convenience, we say this solution is **related** to the monomial $\pi_{\bar{\mathbf{u}}}(\mathbf{x})$. Then $\mathrm{Sol}(F_{0,r} \mid I \subseteq \mathrm{Supp}(\mathbf{u}^{(0)}), \mathbf{u}^{(r)} = \bar{\mathbf{w}}, \mathbf{u}^{(0)} = \bar{\mathbf{u}})$ is the set of solutions related to the monomial $\pi_{\bar{\mathbf{u}}}(\mathbf{x})$. According to Proposition 1, we have that the monomial $\mathbf{x}^{\bar{\mathbf{u}}}$ appears in $\hat{f}$, or $\mathbf{x}^{\bar{\mathbf{u}}} \to \hat{f}$, if the cardinality of the set $\mathrm{Sol}(F_{0,r} \mid I \subseteq \mathrm{Supp}(\mathbf{u}^{(0)}), \mathbf{u}^{(r)} = \bar{\mathbf{w}}, \mathbf{u}^{(0)} = \bar{\mathbf{u}})$ is odd. Moreover, if all solutions to System (3) are found, the superpoly of C_I can be recovered by collecting all appearing monomials.

Systems are often easier to be solved if the systems have fewer unknowns and constraints. Note that System (3) can be divided into two smaller systems:

$$F_{bottom}(\mathbf{u}) := F_{0,i}(\mathbf{u} \mid I \subseteq \mathrm{Supp}(\mathbf{u}^{(0)})), \tag{4}$$

and

$$F_{top}(\mathbf{u}) := F_{i,r}(\mathbf{u} \mid \mathbf{u}^{(r)} = \bar{\mathbf{w}}). \tag{5}$$

For System (4), because there is no constraints on $\mathbf{u}^{(i)}$, the system has a huge number of solutions and is usually hard to be solved. Fortunately, System (5) can be solved with lower complexity than solving System (3). If a solution to $F_{top}(\mathbf{u})$ is found and $\mathbf{u}^{(i)} = \bar{\mathbf{v}}$, where $\bar{\mathbf{v}}$ is a bit-vector, we can add the constraint "$\mathbf{u}^{(i)} = \bar{\mathbf{v}}$" to System (4), such that the revised system can be solved easier.

Clearly, every solution to system $F_{bottom}(\mathbf{u} \mid \mathbf{u}^{(i)} = \bar{\mathbf{v}})$ can be extended to a solution to System (3) by appending a solution in $\mathrm{Sol}(F_{top} \mid \mathbf{u}^{(i)} = \bar{\mathbf{v}})$. Thus, to determine whether $\mathbf{x}^{\bar{\mathbf{u}}}$ appears in $\hat{f}$, we can deduce the following equation:

$$|\mathrm{Sol}(F \mid \mathbf{u}^{(0)} = \bar{\mathbf{u}})| = \textstyle\sum_{\bar{\mathbf{v}}} |\mathrm{Sol}(F_{bottom} \mid \mathbf{u}^{(i)} = \bar{\mathbf{v}}, \mathbf{u}^{(0)} = \bar{\mathbf{u}})| \cdot |\mathrm{Sol}(F_{top} \mid \mathbf{u}^{(i)} = \bar{\mathbf{v}})|. \tag{6}$$

Then $\mathbf{x}^{\bar{\mathbf{u}}}$ appears in $\hat{f}$ if and only if $|\mathrm{Sol}(F \mid \mathbf{u}^{(0)} = \bar{\mathbf{u}})|$ is odd. Note that $|\mathrm{Sol}(F_{top} \mid \mathbf{u}^{(i)} = \bar{\mathbf{v}})|$ contributes nothing if it is even.

By Equation (6), solving a large system $F(\mathbf{u})$ is converted to solving several small subsystems like $F_{bottom}(\mathbf{u} \mid \mathbf{u}^{(i)} = \bar{\mathbf{v}})$. The number of subsystems is number of different $\bar{\mathbf{v}}$'s that $|\mathrm{Sol}(F_{top} \mid \mathbf{u}^{(i)} = \bar{\mathbf{v}})|$ is odd.

Generally, the complexity of solving System (5) can be controlled by choosing the position of i. Particularly, if the system $F_{bottom}(\mathbf{u} \mid \mathbf{u}^{(i)} = \bar{\mathbf{v}})$ is still too complicated to be solved quickly, we can divide this system to even smaller subsystems.

In all, we have the following divide-and-conquer algorithm for recovering superpolys.

Algorithm 1: SuperPoly()

```
Input  : A system $F(\mathbf{u}) := F_{0,r}(\mathbf{u} \mid I \subseteq \mathrm{Supp}(\mathbf{u}^{(0)}), \mathbf{u}^{(r)} = \bar{\mathbf{w}})$.
Output : The set of monomials that appear in $\pi_{\bar{\mathbf{w}}}(\mathbf{x}^{(r)})$.
1 begin
2     $S \longleftarrow \emptyset$
3     Choose $i$ and solve $F_{top}(\mathbf{u}) := F_{i,r}(\mathbf{u} \mid \mathbf{u}^{(r)} = \bar{\mathbf{w}})$
4     for each $\bar{\mathbf{v}}$ such that $|\mathrm{Sol}(F_{top} \mid \mathbf{u}^{(i)} = \bar{\mathbf{v}})|$ is odd do
5         if $F_{0,i}(\mathbf{u} \mid I \subseteq \mathrm{Supp}(\mathbf{u}^{(0)}), \mathbf{u}^{(i)} = \bar{\mathbf{v}})$ can be solved quickly then
6             $S \longleftarrow S \cup \mathrm{Sol}(F_{0,i} \mid I \subseteq \mathrm{Supp}(\mathbf{u}^{(0)}), \mathbf{u}^{(i)} = \bar{\mathbf{v}})$
7         else
8             $S \longleftarrow S \cup \mathrm{SuperPoly}(F_{0,i}(\mathbf{u} \mid I \subseteq \mathrm{Supp}(\mathbf{u}^{(0)}), \mathbf{u}^{(i)} = \bar{\mathbf{v}}))$
9     return $\{\mathbf{x}^{\bar{\mathbf{u}}} \mid \mathbf{u}^{(0)} = \bar{\mathbf{u}}$ appears for odd times in $S\}$
```

Algorithm 1 presents the framework of the divide-and-conquer algorithm. To practically recover a superpoly, some details should be clarified. We take Trivium for an example, and Gurobi [GO21] is used for solving the systems.

Recovering the superpoly for round-reduced Trivium

Algorithm 2 is a specialization of Algorithm 1 for Trivium. To recover the superpoly of C_I for an r-round Trivium, it suffices to call $SuperPolyTrivium(r, C_I, \emptyset)$. Necessary procedures used in Algorithm 2 are presented in Algorithm 3, where $\mathcal{M}$ stands for a model of Gurobi, $\mathcal{M}.var$ and $\mathcal{M}.con$ refer to the variables and constraints of $\mathcal{M}$.

Algorithm 2: SuperPolyTrivium()

```
Input  : Round $R$; the set $I$ of cube $C_I$; last is empty or a state of 288 bits.
Output : A set of monomials.
1 begin
2     $S \longleftarrow \emptyset$
3     Choose $i$ and $V \longleftarrow$ Expand($R - i$, last)
4     for each $\bar{\mathbf{v}} \in V$ such that $\bar{\mathbf{v}}$ appears for odd times in $V$ do
5         if Solve($i$, $I$, $\bar{\mathbf{v}}$) terminates within some time limit then
6             $S \longleftarrow S \cup$ Solve($i$, $I$, $\bar{\mathbf{v}}$)
7         else
8             $S \longleftarrow S \cup$ SuperPolyTrivium($i$, $I$, $\bar{\mathbf{v}}$)
9     return $\{\mathbf{x}^{\bar{\mathbf{u}}} \mid \bar{\mathbf{u}}$ appears for odd times in $S\}$
```

3.2 An observation

Our goal is to search for valuable cubes efficiently. An important step of the search algorithm is to verify whether a given cube C_I is valuable. The direct way is to recover the whole superpoly, but this is very expensive if the cipher system is very complicated, e.g., 843-round Trivium. An alternative way is to only test whether a specific secret variable k_i is balanced. This method only needs to compute the monomials that involve the variable k_i, and is more efficient. However, for sake of security, the number of secret variables in stream ciphers is at least 80, and this means we need to repeat this method many times, since most secret variables are not balanced.

Instead of using a verification mode, we realized the exclusive mode would be more efficient. For example, if we know that a monomial $k_0k_2k_4$ appears in the superpoly of a cube C_I, then we are sure that the secret variables k_0, k_2, and k_4 are not balanced. But this exclusive method raises another crucial problem: *how to confirm a monomial appears in the superpoly?*

The direct and precise way of checking whether a monomial, say $\mathbf{x}^{\bar{\mathbf{u}}}$, appears in the superpoly, is to solve System (3) by adding an additional constraint "$\mathrm{Supp}(\bar{\mathbf{u}}) \cup I = \mathrm{Supp}(\mathbf{u}^{(0)})$". The complexity of this approach is much lower than that of recovering the whole or parts of the superpoly. But it is still not endurable, because there are lots of monomials to be confirmed. For sake of efficiency, we used a probabilistic method for this verification based on Observation 2, before which we introduce the following straightforward observation first.

Observation 1: *Let $F(\mathbf{u})$ be a (sub)system like System (3) or (4). Among all monomials related to the solutions of $F(\mathbf{u})$, most of them are related to EVEN numbers of solutions.*

In our experiments, this ratio (number of monomials that are related to even numbers of solutions / number of all monomials) usually went over 90%, particularly when $F(\mathbf{u})$ was complex. This means, if a monomial $x^{\bar{\mathbf{u}}}$ is related to N solutions, the probability of N being odd is low.

Observation 2: *In the last subsection, we get the following relation:*

$$|\mathrm{Sol}(F \mid \mathbf{u}^{(0)} = \bar{\mathbf{u}})| = \sum_{\bar{\mathbf{v}}} |\mathrm{Sol}(F_{bottom} \mid \mathbf{u}^{(i)} = \bar{\mathbf{v}}, \mathbf{u}^{(0)} = \bar{\mathbf{u}})| \cdot |\mathrm{Sol}(F_{top} \mid \mathbf{u}^{(i)} = \bar{\mathbf{v}})|.$$

When the number of subsystems that have solutions is not too large, we observed that, for a given monomial $\mathbf{x}^{\bar{\mathbf{u}}}$, if the cardinality of the set $\mathrm{Sol}(F_{bottom} \mid \mathbf{u}^{(i)} = \bar{\mathbf{v}}, \mathbf{u}^{(0)} = \bar{\mathbf{u}})$ is odd in a subsystem, then the number $|\mathrm{Sol}(F \mid \mathbf{u}^{(0)} = \bar{\mathbf{u}})|$ tends to be odd with a relative high probability. The probability increases when the weight of $\bar{\mathbf{u}}$ increases.

Algorithm 3: Gurobi Model for Trivium

```
procedure TriviumCore($\mathcal{M}, state, i_1, i_2, i_3, i_4, i_5$)
    $\mathcal{M}.var \longleftarrow y_1, y_2, y_3, y_4, y_5, z_1, z_2, a$ as binary variables
    $\mathcal{M}.con \longleftarrow state[i_1] = y_1 \vee z_1$
    $\mathcal{M}.con \longleftarrow state[i_2] = y_2 \vee z_2$
    $\mathcal{M}.con \longleftarrow state[i_3] = y_3 \vee a$
    $\mathcal{M}.con \longleftarrow state[i_4] = y_4 \vee a$
    $\mathcal{M}.con \longleftarrow y_5 = state[i_5] + a + z_1 + z_2$
    $state[i_j] \longleftarrow y_j$ for $j \in \{1, 2, 3, 4, 5\}$
procedure TriviumBody($R$)
    Prepare an empty MILP Model $\mathcal{M}$
    $\mathcal{M}.var \longleftarrow s_i$ as binary variable for $i \in \{1, 2, \ldots, 288\}$
    $init \longleftarrow (s_1, s_2, \ldots, s_{288})$
    $state \longleftarrow init$
    for $r = 1$ to $R$ do
        TriviumCore($\mathcal{M}, state, 66, 171, 91, 92, 93$)
        TriviumCore($\mathcal{M}, state, 162, 264, 175, 176, 177$)
        TriviumCore($\mathcal{M}, state, 243, 69, 286, 287, 288$)
        $state \longleftarrow (state[288], state[1], \ldots, state[287])$
    return $(\mathcal{M}, init, state)$
procedure Expand($R$, $last$)
    $(\mathcal{M}, init, st) \longleftarrow$ TriviumBody($R$)
    if $last$ is $\emptyset$
        $\mathcal{M}.con \longleftarrow st[i] = 0$ for $i \in \{1, 2, \ldots, 288\} \setminus \{66, 93, 162, 177, 243, 288\}$
        $\mathcal{M}.con \longleftarrow st[66] + st[93] + st[162] + st[177] + st[243] + st[288] = 1$
    else
        $\mathcal{M}.con \longleftarrow st[i] = last[i]$ for $i \in \{1, 2, \ldots, 288\}$
    $\mathcal{M}.optimize()$
    return $\{\bar{\mathbf{v}} \mid \bar{\mathbf{v}}$ is the solution to the 288 variables in $init\}$
procedure Solve($R$, $I$, $last$)
    $(\mathcal{M}, init, st) \longleftarrow$ TriviumBody($R$)
    $\mathcal{M} \longleftarrow init[i] = 0$ for $i \in \{81, 82, \ldots, 285\}$
    $\mathcal{M} \longleftarrow init[94 + i] = 1$ for $i \in I$, where $I \subseteq \{0, 1, \ldots, 79\}$
    if $last$ is $\emptyset$
        $\mathcal{M}.con \longleftarrow st[i] = 0$ for $i \in \{1, 2, \ldots, 288\} \setminus \{66, 93, 162, 177, 243, 288\}$
        $\mathcal{M}.con \longleftarrow st[66] + st[93] + st[162] + st[177] + st[243] + st[288] = 1$
    else
        $\mathcal{M}.con \longleftarrow st[i] = last[i]$ for $i \in \{1, 2, \ldots, 288\}$
    $\mathcal{M}.optimize()$
    return $\{\bar{\mathbf{u}} \mid \bar{\mathbf{u}}$ is the solution to the 80 variables $init[1], \ldots, init[80]\}$
```

The condition that "the number of subsystems that have solutions is not too large" is important to the above observation. Because in this case, the monomial $\mathbf{x}^{\bar{\mathbf{u}}}$ has a low probability of relating to solutions in other subsystems, and moreover, even if $\mathbf{x}^{\bar{\mathbf{u}}}$ relates to solutions in another subsystem, the probability of its related number being odd is also low by Observation 1.

Discussion about the success rate of Observation 2

As it is very hard to provide theoretical proofs supporting Observation 2, we only discuss about the success rate from the experimental way.

For an evidence of Observation 2, we studied the probabilities of the monomials obtained when recovering the superpolys of 841- and 842-round Trivium. For example, to recover the superpoly of the cube where the indexes of active variables are $\{0,\ldots,79\}\setminus\{18,34\}$ for 842-round Trivium, the divide-and-conquer algorithm is used, and 5 034 subsystems are constructed and solved. Among these subsystems, 48 have solutions. We finally obtained 4 917 raw monomials and 975 of them are confirmed to appear in the superpoly. For each raw monomial $\mathbf{x}^{\bar{\mathbf{u}}}$, its related solutions may be obtained from several subsystems. We are interested in whether its solution number in some subsystem is odd, and whether the overall solution number is odd. Table 3 shows the statistics according to the weights of monomials in the superpolys. The numbers "3, 458, 373, 81.44%" in the fourth line of 842-round Trivium means that, among all 3-degree monomials that were computed from subsystems, there are 458 raw monomials such that their related solution numbers are odd in some subsystems. The overall solution numbers of 373 raw monomials are still odd, i.e., these 373 monomials are confirmed to appear in the superpoly. "81.44 %" is the ratio of 373 divided by 458.

Table 3: Statistics of monomials obtained by recovering the superpolys for 841- and 842-round Trivium.

Round	Cube indexes I	Degree	Odd in subsystems	Odd in final	Probability
841	$\{0,\ldots,79\}\setminus\{8,78\}$	0	1	0	0 %
		1	18	15	83.33 %
		2	23	22	95.65 %
		3	14	14	100.0 %
		4	2	2	100.0 %
	$\{0,\ldots,79\}\setminus\{59,61\}$	0	1	0	0 %
		1	46	33	71.74 %
		2	147	136	92.52 %
		3	111	107	96.40 %
		4	57	57	100.0 %
		5	39	39	100.0 %
		6	17	17	100.0 %
		7	3	3	100.0 %
	$\{0,\ldots,79\}\setminus\{64,72\}$	0	1	0	0 %
		1	65	37	56.92 %
		2	455	393	86.37 %
		3	635	604	95.12 %
		4	492	490	99.59 %
		5	218	218	100.0 %
		6	74	74	100.0 %
		7	12	12	100.0 %
		8	1	1	100.0 %
842	$\{0,\ldots,79\}\setminus\{18,34\}$	0	1	0	0 %
		1	57	38	66.67 %
		2	375	294	78.40 %
		3	458	373	81.44 %
		4	255	209	81.96 %
		5	66	55	83.33 %
		6	6	6	100.0 %

Table 3 provides an experimental proof for Observation 2. As the degrees of monomials increase, the probabilities that Observation 2 holds increase as well. However, degree-2 monomials always have lower probabilities in Table 3, it is natural to worry about

Observation 2 will fail for these monomials. Note that monomials with degree 0 and 1 will not be used to reject useless cubes, so we do not consider them here.

To check the probabilities for degree-2 monomials, we did more experiments. Using the three cubes of 841-round Trivium in Table 3, by back-expanding (150, 200, 250, 300, 320, 340) rounds, we obtained (48, 120, 869, 4 147, 15 584, 37 371) subsystems. For the first cube, the numbers of subsystems that have solutions related to degree-2 monomials, are (2, 5, 5, 6, 8, 8). The probabilities (odd in final / odd in subsystems in Table 3) of degree-2 monomials are always 95.65%. For the other two cubes, related numbers of subsystems are (2, 4, 4, 5, 7, 7) and (2, 5, 5, 8, 10, 10). The corresponding probabilities are (98.55%, 93.79%, 93.79%, 93.15%, 92.52%, 92.52%) and (96.32%, 90.76%, 90.76%, 90.76%, 90.55%, 90.55%). We can see the numbers of subsystems that have solutions are much less than the numbers of all subsystems, and the increase of these numbers leads to the decrease of related probabilities. But in all, the probabilities are relatively high. During our practical searches of valuable cubes, we handled thousands of cubes together and generated lots of subsystems, but the monomials that related to a specific cube usually appeared in only a small number of subsystems, e.g., for 841-round Trivium, the number is $8 \sim 27$ generally. Thus, Observation 2 held with a relatively high probability in our experiments, and helped us find many valuable cubes.

Since it is hard to calculate the precise success rates of Observation 2, we have a remedy if the actual success rates of Observation 2 are really very low. We noted that, the larger the subsystems are, the higher the probabilities that Observation 2 hold will be. Particularly, if the subsystems are large enough to be the whole system, Observation 2 surely holds. So if we thought the practical success rates of Observation 2 was very low and no valuable cubes were found, we could increase the sizes of the subsystems and compute again. If there really exist some valuable cubes among the candidate ones, we eventually find some when the subsystems are large enough.

Discussion about Observation 2 and Todo et al.'s assumptions

In [TIHM17], Todo et al.'s made the strong and weak assumptions: "for a cube C_I, there are many values in the constant part of IV whose corresponding superpoly is balanced", or "is not a constant function". Todo et al.'s assumptions provided a possible way to search for lots of balanced superpolys together, i.e., considering one cube with different values in the constant part of IV. But these two assumptions were later proved wrong in [YT19].

However, Observation 2 as well as the new search algorithm use the monomial prediction technique proposed in [HSWW20], which only considers the case that the values in the constant part of IV are all 0's. So the new search algorithm handles many cubes together in a different way from that in above paragraph, i.e., it handles many different sets of active bits in IV, e.g., the set of cubes whose dimensions are 78, while the values of inactive bits are always set as 0's. So Observation 2 has no direct relations to Todo et al.'s assumptions.

3.3 A search algorithm for valuable cubes

Based on Observation 2, we devised a new algorithm to search for valuable cubes. In this algorithm, we consider a set of cubes together instead of a single one, because many duplicated computations of these cubes can be avoided when we deal with them together. Let $\mathbb{C}$ be a set of candidate cubes. For example, $\mathbb{C}$ is the set of cubes whose dimensions are 34, 40, or 78.

First of all, we need to select a secret variable to perform further searches. This selection can be done based on some heuristic computations, or just done randomly.

For a selected secret variable, say k_i, we use a table to record the status of each cube in $\mathbb{C}$. The status of every cube $C_I \in \mathbb{C}$ could be *not appear*, *linear*, and *excluded*. Initially, the status of all cubes is set as *not appear*. For the status *linear*, we also record the number of solutions that are related to the monomial $t_I k_i$ where t_I is the product of public variables with indexes in I.

Next, we use the divide-and-conquer algorithm to compute all monomials that probably appear in the superpolys of cubes in $\mathbb{C}$. As there are many cubes in $\mathbb{C}$ instead of one, we change the condition "$I \subseteq \mathrm{Supp}(\mathbf{u}^{(0)})$" in System (3) to "there exists $C_I \in \mathbb{C}$ such that $I \subseteq \mathrm{Supp}(\mathbf{u}^{(0)})$". And we also need to add the condition "k_i *divides* $\pi_{\mathbf{u}^{(0)}}(\mathbf{x})$" to the system, i.e., we only consider the monomials involving k_i.

When the computation starts, the divide-and-conquer strategy will divide the large system into several small subsystems. Once a subsystem is solved, we collect the solutions, and deal with the monomials related to these solutions. Please note that, by the first constraint added in the last paragraph, every monomial corresponds to a cube $C_I \in \mathbb{C}$. If solutions related to the monomial $t_I k_i$ are found and the status of C_I is not *excluded*, we record/add the number of solutions and mark the status of C_I as *linear*. If solutions related to some monomial with higher degree than $t_I k_i$ are found, we have three criteria to deal with this monomial. If a cube C_I is excluded by a criterion, we update the status of C_I to *excluded*, and add a constraint to the other subsystems to exclude the solutions related to the cube C_I.

Based on Observation 2, there are two criteria for dealing with the monomials with higher degrees than $t_I k_i$.

First Criterion: If *there are* solutions related to a monomial $\mathbf{x}^{\bar{\mathbf{u}}}$, such that $\mathbf{x}^{\bar{\mathbf{u}}} \neq t_I k_i$, then k_i is supposed to be unbalanced in the superpoly of C_I, and hence, we exclude the cube C_I from $\mathbb{C}$.

Second Criterion: If the number of solutions related to a monomial $\mathbf{x}^{\bar{\mathbf{u}}}$ is *odd*, where $\mathbf{x}^{\bar{\mathbf{u}}} \neq t_I k_i$, then k_i is supposed to be unbalanced in the superpoly of C_I, and hence, we exclude the cube C_I from $\mathbb{C}$.

By the above definitions, we can see that First Criterion is more aggressive but also more efficient. A brief comparison is shown in Table 4. So First Criterion can be used for simple systems, while Second Criterion is usually applied to more complicated systems.

After the search algorithm terminates, i.e., all subsystems are solved, we check the status of the cubes in $\mathbb{C}$. If a cube has the status *linear*, and the number of related solutions is odd, then we successfully find a valuable cube; otherwise, if no valuable cubes are found, we change the set $\mathbb{C}$ or change the secret variable k_i, and try again. The search algorithm is given by Algorithm 4.

In practical applications, the above search algorithm may not terminate within endurable time due to the complicated structures of the ciphers. In this case, if most of the candidate cubes have been excluded, we can check the remaining cubes one by one, via the direct way. It usually does not take much time to find a valuable cube, and we found the valuable cubes for 843-round Trivium and 893-round Kreyvium in this way.

4 Applications to Trivium and Kreyvium

We applied the search algorithm to Trivium and Kreyvium, and obtained three main improvements. Due to the page limit, superpolys used in this section can be found at https://github.com/ysun0102/searchforcubes/tree/main/results.

4.1 Theoretical cube attacks against round-reduced Trivium

To obtain theoretical cube attacks, we searched for valuable cubes whose dimensions are 78. As there are 80 secret variables in Trivium, a valuable cube with dimension 78 could lead to a theoretical attack with the complexity $2^{79} + 2^{78}$. So the candidate set $\mathbb{C}$ is set as all the cubes with 78 public variables and the size of $\mathbb{C}$ is 3 160.

Algorithm 4: SearchValuableCubes(F, $\mathbb{C}$, k_i)

```
Input  : The system $F(\mathbf{u}) := F_{0,r}(\mathbf{u} \mid \mathbf{u}^{(r)} = \bar{\mathbf{w}}$, $k_i$ divides $\pi_{\mathbf{u}^{(0)}}(\mathbf{x})$, and there
         exists $C_I \in \mathbb{C}$ s.t. $I \subseteq \mathrm{Supp}(\mathbf{u}^{(0)}))$;
         A set of candidate cubes $\mathbb{C}$;
         A secret variable $k_i$.
Output : The set of valuable cubes.
1  begin
2      for each cube $C_I \in \mathbb{C}$ do
3          $Status[C_I] \longleftarrow$ not appear
4          $Count[C_I] \longleftarrow 0$
5      for each subsystem $F'$ in SuperPoly($F(\mathbf{u})$) do
6          if $\exists\, C_I \in \mathbb{C}$ s.t. $\mathrm{Sol}(F' \mid \pi_{\mathbf{u}^{(0)}}(\mathbf{x}) = t_I k_i) \neq \emptyset$ and $Status[C_I] \neq$ excluded
            then
7              $Status[C_I] \longleftarrow$ linear
8              $Count[C_I] \longleftarrow Count[C_I] + |\mathrm{Sol}(F' \mid \pi_{\mathbf{u}^{(0)}}(\mathbf{x}) = t_I k_i)|$
9          for each monomial $\mathbf{x}^{\bar{\mathbf{u}}}$ s.t. $\mathrm{Sol}(F' \mid \mathbf{u}^{(0)} = \bar{\mathbf{u}}) \neq \emptyset$ and $\mathbf{x}^{\bar{\mathbf{u}}} \neq t_I k_i$ do
10             if the related cube $C_I$ is rejected by a criterion then
11                 $Status[C_I] \longleftarrow$ excluded
12                 $\mathbb{C} \longleftarrow \mathbb{C} \setminus \{C_I\}$
13                 Excludes solutions related to $C_I$ in other un-computed subsystems
14     return $\{C_I \mid Status[C_I] = linear$ and $Count[C_I]$ is odd$\}$
```

Cube attacks against 840-, 841-, and 842-round Trivium

For 840-round Trivium, we chose k_0, k_1, k_2 as candidate secret variables, and performed brief comparisons between these secret variables as well as the two criteria. Our platform is: AMD Threadripper 3970X with 32 cores, 256 GB memory, Ubuntu 20.04.

Table 4: Comparisons between secret variables and criteria for 840-round Trivium.

Balanced secret variable	Criterion	#rejected cubes	#valuable cubes	time (sec.)
k_0	1	2 237	98	35 137.49
	2	1607	222	60 039.48
k_1	1	1 730	76	21 985.85
	2	851	215	49 280.23
k_2	1	758	81	21 173.94
	2	331	134	40 863.25

In Table 4, we can see that Criterion 1 is more efficient, but more possible valuable cubes were rejected aggressively. We recovered 9 superpolys for the founded valuable cubes, i.e., the first 3 valuable cubes for each k_i. The computing time is shown in Table 5. Although the time differs significantly for different cubes, it is always more than 1 hour. Thus, compared with the proposed search algorithm, it takes much more time to search for valuable cubes by retrieving all superpolys for (3 160) candidate cubes.

As the algebraic structures of 841- and 842-round Trivium are not very complicated, we used First Criterion to reject useless cubes. For 841-round Trivium, we found 2 valuable cubes for k_0, and 42 valuable ones for k_1. For 842-round Trivium, we directly chose k_1 as the candidate secret variable. We found 28 cubes whose status is *linear*, and

Table 5: Time for retrieving superpolys of the founded valuable cubes. "$\{10, 26\}$" refers to the cube indexes $\{0, 1, \ldots, 79\} \setminus \{10, 26\}$, and the time is given in seconds.

k_0			k_1			k_2		
$\{10, 26\}$	$\{10, 32\}$	$\{0, 33\}$	$\{6, 9\}$	$\{7, 46\}$	$\{7, 61\}$	$\{0, 21\}$	$\{0, 22\}$	$\{0, 23\}$
6 256.86	6 445.33	5 090.17	3 858.84	5 567.27	5 418.30	11 265.39	12 098.60	15 929.83

there are 5 cubes having odd numbers of solutions.

Cube attacks against 843-round Trivium

To search for valuable cubes for 843-round Trivium, Second Criterion is used to reject useless cubes. We selected k_2 as the candidate secret variable. However, the computations were slowed down after using Second Criterion, so we cut down the program when about 3 140 cubes were excluded. For the remaining 20 cubes, we tested them directly by recovering the monomials that only involve k_2. It took about 10 000 seconds for each test. Finally, we found two valuable cubes whose inactive public variables are:

$$\{v_{30}, v_{50}\} \text{ and } \{v_{30}, v_{76}\}.$$

For recovering the superoly of the cube C_I where $I = \{0, \ldots, 79\} \setminus \{30, 76\}$, we totally obtained 1 085 554 019 solutions in more than two weeks. These solutions are related to 140 096 raw monomials, and the distribution of solutions is unbalanced. For instance, there are 396 911 938 solutions related to the monomial t_I, i.e., the constant "1" in the superpoly. By removing the raw monomials whose solution numbers are even, we obtained 16 561 monomials that really appear in the superpoly of C_I. Detailed results about this superpoly can be found at the GitHub website.

Thus, we can get the value of this superpoly by summing up all 2^{78} possible values in the cube C_I. As the value of k_2 can be deduced directly from the other secret bits by using this superpoly, we can recover the whole 80-bit key by doing 2^{79} exhaustive searches. The overall complexity is lower than 2^{80}, and it is an attack against 843-round Trivium.

4.2 Practical cube attacks against round-reduced Trivium

To perform practical cube attacks, we need many valuable cubes instead of a single one, and besides, the dimension of the cubes must be small. Using the idea in [YT20], we preset a set of indexes, and then searched for valuable cubes whose indexes are the subsets of the preset set. If the size of the preset set is l, to obtain the values of the superpolys related to the obtained valuable cubes, it suffices to query the encryption oracle 2^l times.

Please remark that, unlike the linear superpolys found in [YT20], almost all superpolys found by our technique are *nonlinear*, but these superpolys all have balanced secret variables. This characteristic enables us to deduce the values of balanced variables by linearizing the superpolys, and the linearization can be done by delicately selecting the variables that are used to enumerate values. Moreover, once the values of some balanced variables are deduced, they can be used to deduce the values of other balanced variables iteratively. We use the following toy example to illustrate this iterative deduction.

Example 2. Let $\{f_1 = x_1 + x_3 + x_2x_4x_5,\ f_2 = x_2 + x_4x_5,\ f_3 = x_2 + x_3 + x_2x_5\}$ be nonlinear polynomials with balanced variables $\{x_1, x_3\}$, $\{x_2\}$ and $\{x_3\}$ respectively. Assume we know the values of $\{f_1, f_2, f_3\}$, and our goal is to obtain the specific values of $\{x_1, x_2, \ldots, x_5\}$.

To solve this system, we can enumerate the values of x_4 and x_5 to linearize the polynomial f_2, and no matter what values they are, we will get the specific value of x_2. Then, we can get the value of x_3 after knowing the values of x_2 and x_5 in f_3. And consequently, we can deduce the value of x_1 in f_1 after knowing the specific values of all the other variables. In this way, for each enumeration of x_4 and x_5, we can solve (x_2, x_3, x_1) within constant time. So the overall complexity for solving this system is 2^2.

Please remark that, there are *three* crucial points in the above method. Firstly, the polynomials must have balanced variables. Secondly, the deducing order of variables is very important, e.g., the value of x_1 cannot be obtained from f_1 by only guessing the values of x_4 and x_5 at the beginning. Thirdly, the above method is independent with the specific values of $\{f_1, f_2, f_3\}$ and $\{x_4, x_5\}$, i.e., the values of $\{x_2, x_3, x_1\}$ can always be solved no matter what values of $\{f_1, f_2, f_3\}$ and $\{x_4, x_5\}$ are.

A practical attack against 806-round Trivium

In [YT20], the authors presented 16 cubes whose superpolys are linear, and the values of these superpolys can be obtained by $2^{38.64}$ requests. But they had to use a brute-force attack to recover the remaining 64 key bits, so their overall attack cannot be done practically.

To obtain a practical attack against 806-round Trivium, we searched for more valuable cubes. We preset a set of indexes, say S_a, which was obtained by merging some sets in [YT20]. The size of S_a is 39, so it only needs 2^{39} requests to obtain all the values of superpolys whose related cubes are from the subsets of S_a. Finally, we found 29 valuable cubes and recovered related superpolys, which can be found at the GitHub website. Details about these cubes as well as S_a are shown in Table 6.

Table 6: Valuable cubes for attacking 806-round Trivium. Sorted by the deducing order.

Indexes of cubes	balanced bits	Indexes of cubes	balanced bits
$S_a \setminus \{10, 16, 25, 38, 69\}$	$k_{25}, \mathbf{k_{67}}$	$S_a \setminus \{22, 49\}$	$\mathbf{k_{16}}$
$S_a \setminus \{16, 49, 69\}$	$\mathbf{k_{77}}$	$S_a \setminus \{22, 73\}$	$\mathbf{k_{18}}$
$S_a \setminus \{14, 16, 25, 38, 69\}$	$\mathbf{k_{2}}, k_{32}, k_{65}$	$S_a \setminus \{15, 22, 57\}$	$\mathbf{k_{13}}, k_{18}$
$S_a \setminus \{16, 25, 38, 40, 69\}$	$\mathbf{k_{7}}, k_{25}, k_{34}, k_{43}, k_{67}$	$S_a \setminus \{33, 57, 69\}$	$\mathbf{k_{35}}$
$S_a \setminus \{15, 16, 57\}$	$\mathbf{k_{29}}, k_{56}$	$S_a \setminus \{42, 69, 73\}$	$k_0, k_9, k_{18}, k_{27}, k_{30}, k_{33}, \mathbf{k_{36}}$
$S_a \setminus \{11, 15, 57\}$	$\mathbf{k_{48}}, k_{77}$	$S_a \setminus \{9, 44\}$	$\mathbf{k_{22}}, k_{49}$
$S_a \setminus \{16, 40, 49\}$	$\mathbf{k_{50}}$	$S_a \setminus \{16, 25, 38, 48, 69\}$	$k_{30}, k_{44}, \mathbf{k_{45}}$
$S_a \setminus \{0, 49\}$	$\mathbf{k_{3}}, k_{15}, k_{57}$	$S_a \setminus \{15, 33, 69\}$	$\mathbf{k_{20}}, k_{29}, k_{38}, k_{56}$
$S_a \setminus \{30, 39, 49\}$	$\mathbf{k_{26}}, k_{53}$	$S_a \setminus \{0, 35, 49\}$	$k_{16}, \mathbf{k_{31}}, k_{46}, k_{76}$
$S_a \setminus \{11, 16, 25, 38, 69\}$	$\mathbf{k_{0}}, k_{12}, k_{27}, k_{54}, k_{63}, k_{68}$	$S_a \setminus \{16, 69, 73\}$	$k_{17}, \mathbf{k_{19}}, k_{28}, k_{31}, k_{37}, k_{40}, k_{43}, k_{49}, k_{55}, k_{67}$
$S_a \setminus \{0, 26, 49\}$	$k_3, \mathbf{k_{30}}$	$S_a \setminus \{0, 7, 46\}$	$k_{18}, \mathbf{k_{21}}$
$S_a \setminus \{15, 39, 63\}$	$\mathbf{k_{60}}$	$S_a \setminus \{5, 40, 73\}$	$\mathbf{k_{6}}, k_{21}$
$S_a \setminus \{16, 25, 38, 63, 69\}$	$k_0, k_{60}, \mathbf{k_{78}}$	$S_a \setminus \{0, 49, 76\}$	$\mathbf{k_{4}}$
$S_a \setminus \{16, 63, 69\}$	$\mathbf{k_{9}}$	$S_a \setminus \{28, 57, 61\}$	$\mathbf{k_{5}}$
$S_a \setminus \{11, 15, 39\}$	$\mathbf{k_{79}}$		

$$S_a = \{0, 1, 3, 5, 7, 9, 10, 11, 12, 14, 15, 16, 18, 20, 22, 24, 25, 26, 27, 28, 30, 33, 35, 37, 38, 39, 40, 42, 44, 46, 48, 49, 57, 61, 63, 69, 73, 76, 78\}.$$

To attack 806-round Trivium practically, it takes 2^{39} requests to obtain all the values of superpolys in Table 6. By using the linear superpolys in [YT20], we can get the values of the 16 variables: $\{k_{14}, k_{15}, k_{17}, k_{28}, k_{32}, k_{33}, k_{41}, k_{42}, k_{44}, k_{46}, k_{52}, k_{55}, k_{58}, k_{59}, k_{63}, k_{65}\}$, and the complexity is $2^{38.64}$. Next, we need to enumerate the values of 35 variables: $\{k_1, k_8, k_{10}, k_{11}, k_{12}, k_{23}, k_{24}, k_{25}, k_{27}, k_{34}, k_{37}, k_{38}, k_{39}, k_{40}, k_{43}, k_{47}, k_{49}, k_{51}, k_{53}, k_{54}, k_{56}, k_{57}, k_{61}, k_{62}, k_{64}, k_{66}, k_{68}, k_{69}, k_{70}, k_{71}, k_{72}, k_{73}, k_{74}, k_{75}, k_{76}\}$, and the complexity is 2^{35}. During each enumeration, the values of the remaining 29 variables can be deduced iteratively in the following order: $(k_{67}, k_{77}, k_2, k_7, k_{29}, k_{48}, k_{50}, k_3, k_{26}, k_0, k_{30}, k_{60}, k_{78}, k_9, k_{79}, k_{16}, k_{18}, k_{13}, k_{35}, k_{36}, k_{22}, k_{45}, k_{20}, k_{31}, k_{19}, k_{21}, k_6, k_4, k_5)$, and this deduction only costs constant time. To sum up, the whole attack costs $2^{39} + 2^{38.64} + 2^{35}$ operations, which can be done practically.

A practical attack against 808-round Trivium

Similarly, to attack 808-round Trivium practically, we preset a set S_b of indexes, and found 37 valuable cubes, as shown in Table 7. The size of S_b is 44, so it takes 2^{44} requests to obtain all the values of these 37 superpolys. Next, we need to enumerate the values of 43 variables: $\{k_1, k_6, k_8, k_{10}, k_{11}, k_{19}, k_{20}, k_{22}, k_{24}, k_{26}, k_{27}, k_{28}, k_{33}, k_{34}, k_{35}, k_{37}, k_{38}, k_{40}, k_{41}, k_{43}, k_{44}, k_{45}, k_{46}, k_{47}, k_{48}, k_{49}, k_{50}, k_{51}, k_{52}, k_{54}, k_{55}, k_{59}, k_{60}, k_{61}, k_{64}, k_{65}, k_{66}, k_{69}, k_{70}, k_{72}, k_{74}, k_{75}, k_{77}\}$. and this complexity is 2^{43}. For each enumeration, the values of the remaining 37 variables can be deduced iteratively in the order: $(k_{21}, k_{56}, k_{39}, k_{76}, k_{14}, k_{67}, k_{57}, k_{62}, k_{78}, k_{12}, k_{30}, k_{36}, k_{68}, k_{42}, k_{23}, k_{53}, k_{58}, k_{25}, k_{32}, k_{31}, k_{71}, k_{13}, k_{73}, k_{63}, k_0, k_{16}, k_{18}, k_{15}, k_{29}, k_5, k_{79}, k_7, k_9, k_{17}, k_2, k_4, k_3)$. The whole attack costs $2^{44} + 2^{43}$ operations, which can be done practically.

Table 7: Valuable cubes for attacking 808-round Trivium. Sorted by the deducing order.

Indexes of cubes	balanced bits	Indexes of cubes	balanced bits
$S_b \setminus \{37, 71, 75\}$	$\mathbf{k_{21}}, k_{48}$	$S_b \setminus \{11, 50, 55\}$	$\mathbf{k_{31}}, k_{40}, k_{67}$
$S_b \setminus \{18, 37, 72\}$	$\mathbf{k_{56}}$	$S_b \setminus \{15, 20, 75\}$	$k_{44}, \mathbf{k_{71}}$
$S_b \setminus \{11, 37, 72\}$	$k_{21}, \mathbf{k_{39}}, k_{48}, k_{66}$	$S_b \setminus \{29, 50, 71\}$	$\mathbf{k_{13}}, k_{31}, k_{58}$
$S_b \setminus \{18, 37, 41\}$	$k_{49}, \mathbf{k_{76}}$	$S_b \setminus \{47, 50, 71\}$	$k_{13}, k_{31}, \mathbf{k_{73}}$
$S_b \setminus \{11, 57, 76\}$	$\mathbf{k_{14}}, k_{41}$	$S_b \setminus \{34, 71, 75\}$	$\mathbf{k_{63}}$
$S_b \setminus \{11, 50, 72\}$	$k_{40}, k_{49}, \mathbf{k_{67}}, k_{76}$	$S_b \setminus \{15, 18, 47\}$	$\mathbf{k_0}, k_{13}$
$S_b \setminus \{11, 37, 54\}$	$k_{14}, \mathbf{k_{57}}$	$S_b \setminus \{18, 29, 37\}$	$k_0, \mathbf{k_{16}}, k_{33}, k_{36}, k_{42}, k_{51}, k_{60}, k_{63}, k_{78}$
$S_b \setminus \{11, 14, 18, 57, 76\}$	$k_{40}, k_{49}, \mathbf{k_{62}}, k_{67}, k_{76}$	$S_b \setminus \{32, 54, 71\}$	$\mathbf{k_{18}}, k_{27}, k_{36}, k_{45}, k_{48}, k_{54}, k_{55}$
$S_b \setminus \{37, 57, 71\}$	$k_{21}, k_{40}, k_{48}, k_{49}, k_{51}, k_{67}, \mathbf{k_{78}}$	$S_b \setminus \{18, 39, 47\}$	$\mathbf{k_{15}}$
$S_b \setminus \{11, 50, 54\}$	$\mathbf{k_{12}}, k_{57}$	$S_b \setminus \{14, 37, 54, 57, 76\}$	$k_{27}, \mathbf{k_{29}}$
$S_b \setminus \{18, 37, 76\}$	$\mathbf{k_{30}}, k_{39}, k_{57}, k_{66}$	$S_b \setminus \{12, 18, 41\}$	$\mathbf{k_5}, k_{32}$
$S_b \setminus \{50, 53, 57, 76\}$	$\mathbf{k_{36}}$	$S_b \setminus \{12, 37, 54\}$	$k_{16}, k_{61}, \mathbf{k_{79}}$
$S_b \setminus \{37, 39, 57\}$	$k_{62}, \mathbf{k_{68}}$	$S_b \setminus \{29, 64, 71\}$	$\mathbf{k_7}, k_{29}, k_{47}$
$S_b \setminus \{15, 19, 54, 57, 76\}$	$\mathbf{k_{42}}, k_{57}, k_{69}$	$S_b \setminus \{15, 18, 37, 57, 76\}$	$\mathbf{k_9}, k_{19}$
$S_b \setminus \{37, 54, 72\}$	$\mathbf{k_{23}}, k_{50}$	$S_b \setminus \{11, 37, 64\}$	$\mathbf{k_{17}}$
$S_b \setminus \{30, 54, 57\}$	$\mathbf{k_{53}}$	$S_b \setminus \{4, 11, 37\}$	$\mathbf{k_2}, k_{17}, k_{34}$
$S_b \setminus \{50, 54, 72\}$	$k_{40}, \mathbf{k_{58}}, k_{67}$	$S_b \setminus \{16, 53, 72\}$	$k_2, \mathbf{k_4}$
$S_b \setminus \{43, 71, 75\}$	$\mathbf{k_{25}}$	$S_b \setminus \{2, 72, 75\}$	$\mathbf{k_3}$
$S_b \setminus \{18, 37, 54, 57, 76\}$	$\mathbf{k_{32}}$		

$S_b = \{0, 2, 4, 6, 8, 10, 11, 12, 14, 15, 16, 18, 19, 20, 21, 22, 23, 25, 27, 29, 30, 32, 34, 36, 37, 39, 41, 43, 45, 47, 50, 53, 54, 55, 57, 60, 62, 64, 69, 71, 72, 75, 76, 79\}$.

4.3 Cube attacks against round-reduced Kreyvium

Kreyvium is designed for the use of fully Homomorphic encryption [CCF+16]. The sizes of key and IV are both 128. Kreyvium consists of 5 registers, and 3 of them are the same as Trivium. The other two registers are used to update the keys and IVs, which makes Kreyvium more complicated than Trivium. The registers are initialized as

$$\begin{aligned}
(s_1, s_2, \ldots, s_{93}) &\leftarrow (k_0, k_1, \ldots, k_{92}),\\
(s_{94}, s_{95}, \ldots, s_{177}) &\leftarrow (v_0, v_1, \ldots, v_{83}),\\
(s_{178}, s_{179}, \ldots, s_{288}) &\leftarrow (v_{84}, v_{85}, \ldots, v_{127}, 1, \ldots, 1, 0),\\
(K_{128}, K_{127}, \ldots, K_1) &\leftarrow (k_0, k_1, \ldots, k_{127}),\\
(V_{128}, V_{127}, \ldots, V_1) &\leftarrow (v_0, v_1, \ldots, v_{127}).
\end{aligned}$$

The state of Kreyvium contains $288(s) + 128(K) + 128(V) = 544$ bites, and is updated in the following way:

$$\begin{aligned}
t_1 &\leftarrow s_{66} + s_{93}, \\
t_2 &\leftarrow s_{162} + s_{177}, \\
t_3 &\leftarrow s_{243} + s_{288} + K_1, \\
z &\leftarrow t_1 + t_2 + t_3, \\
t_1 &\leftarrow t_1 + s_{91} \cdot s_{92} + s_{171} + V_1, \\
t_2 &\leftarrow t_2 + s_{175} \cdot s_{176} + s_{264}, \\
t_3 &\leftarrow t_3 + s_{286} \cdot s_{287} + s_{69}, \\
(s_1, s_2, \ldots, s_{93}) &\leftarrow (t_3, s_1, \ldots, s_{92}), \\
(s_{94}, s_{95}, \ldots, s_{177}) &\leftarrow (t_1, s_{94}, \ldots, s_{176}), \\
(s_{178}, s_{179}, \ldots, s_{288}) &\leftarrow (t_2, s_{178}, \ldots, s_{287}), \\
(K_{128}, K_{127}, \ldots, K_1) &\leftarrow (K_1, K_{128}, \ldots, K_2), \\
(V_{128}, V_{127}, \ldots, V_1) &\leftarrow (V_1, V_{128}, \ldots, V_2),
\end{aligned}$$

where z denotes the 1-bit key stream. The state is updated 1 152 times without producing an output. After the key initialization is done, one bit key stream is produced by every update function.

Cube attack against 892-round Kreyvium

In [HLM+20], the authors used a 115-dimensional cube with $I = \{0, \ldots, 127\} \setminus \{6, 22, 38, 43, 61, 66, 72, 73, 78, 101, 106, 109, 110\}$. The superpoly of this cube is $p = k_{26} + k_{40} + k_{50} + k_{85} + k_{109} + 1$. For testing the search algorithm, we preset $S = I \cup \{38, 72\}$, and searched for valuables cubes whose balanced secret variables contain k_{26}. The set $\mathbb{C}$ is the set of cubes from the subsets of S whose sizes are 115, so there are 6 786 candidate cubes. As a result, k_{26} does not appear in the superpolys of 5 009 cubes, and 912 cubes were excluded by the search algorithm. We finally found 476 valuable cubes, including the cube used by Hao et al. For the cube $S \setminus \{0, 48\}$, its superpoly is $p = k_{26} + k_{85} + 1$. Together with Hao et al.'s superpoly, we can reduces the overall complexity from Hao et al.'s $2^{127} + 2^{115}$ to $2^{126} + 2 \times 2^{115}$. In fact, this complexity can be even lowered by using more superpolys.

Cube attack against 893-round Kreyvium

We chose the same set S as above, and searched for valuable cubes whose balanced secret variables still contain k_{26}. By excluding about 6 000 cubes, we finally obtained one valuable cube, whose indexes are $S \setminus \{54, 67\}$. The superpoly of this cube can be found at the given GitHub website. Thus, the overall complexity of cube attack against 893-round Kreyvium is $2^{127} + 2^{115}$.

5 Conclusion

In this paper, we proposed a new algorithm to search for valuable cubes for attacking stream ciphers. By dealing with candidate cubes together and using a probabilistic method of rejecting useless cubes, the search algorithm is very efficient. This technique could be used in both theoretical and practical analyses. As applications, we applied the search algorithm to Trivium and Kreyvium, and obtained three improvements. Firstly, we proposed the first theoretical key-recovery cube attack against 843-round Trivium. Secondly, we present the best practical attack against Trivium, which increases the round from 805 to 808. Lastly, we mount the key-recovery attack against Kreyvium to 893. We also believe this new search algorithm is able to find valuable cubes for higher rounds of Kreyvium as well as for other ciphers.

Acknowledgments

The author thanks the anonymous reviewers and the shepherd Patrick Derbez for many helpful comments. The work is supported by the National Natural Science Foundation of China under Grants No. 61877058 and No. 61977060.

References

[ADMS09] Jean-Philippe Aumasson, Itai Dinur, Willi Meier, and Adi Shamir. Cube testers and key recovery attacks on reduced-round md6 and trivium. In Orr Dunkelman, editor, *Fast Software Encryption*, pages 1–22, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[CCF$^+$16] Anne Canteaut, Sergiu Carpov, Caroline Fontaine, Tancrède Lepoint, María Naya-Plasencia, Pascal Paillier, and Renaud Sirdey. Stream ciphers: A practical solution for efficient homomorphic-ciphertext compression. In Thomas Peyrin, editor, *Fast Software Encryption*, pages 313–333, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.

[DCP08] Christophe De Cannière and Bart Preneel. *Trivium*, pages 244–266. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[DKR97] Joan Daemen, Lars Knudsen, and Vincent Rijmen. The block cipher square. In Eli Biham, editor, *Fast Software Encryption*, pages 149–165, Berlin, Heidelberg, 1997. Springer Berlin Heidelberg.

[DMP$^+$15] Itai Dinur, Paweł Morawiecki, Josef Pieprzyk, Marian Srebrny, and Michał Straus. Cube attacks and cube-attack-like cryptanalysis on the round-reduced keccak sponge function. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology – EUROCRYPT 2015*, pages 733–761, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.

[DS09] Itai Dinur and Adi Shamir. Cube attacks on tweakable black box polynomials. In Antoine Joux, editor, *Advances in Cryptology - EUROCRYPT 2009*, pages 278–299, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[DS11] Itai Dinur and Adi Shamir. Breaking grain-128 with dynamic cube attacks. In Antoine Joux, editor, *Fast Software Encryption*, pages 167–187, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[FV14] Pierre-Alain Fouque and Thomas Vannet. Improving key recovery to 784 and 799 rounds of trivium using optimized cube attacks. In Shiho Moriai, editor, *Fast Software Encryption*, pages 502–517, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.

[FWDM18] Ximing Fu, Xiaoyun Wang, Xiaoyang Dong, and Willi Meier. A key-recovery attack on 855-round trivium. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology – CRYPTO 2018*, pages 160–184, Cham, 2018. Springer International Publishing.

[GO21] LLC Gurobi Optimization. Gurobi optimizer reference manual, 2021.

[HLM$^+$20] Yonglin Hao, Gregor Leander, Willi Meier, Yosuke Todo, and Qingju Wang. Modeling for three-subset division property without unknown subset. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology – EUROCRYPT 2020*, pages 466–495, Cham, 2020. Springer International Publishing.

[HSWW20] Kai Hu, Siwei Sun, Meiqin Wang, and Qingju Wang. An algebraic formulation of the division property: Revisiting degree evaluations, cube attacks, and key-independent sums. In Shiho Moriai and Huaxiong Wang, editors, *Advances in Cryptology – ASIACRYPT 2020*, pages 446–476, Cham, 2020. Springer International Publishing.

[HW19] Kai Hu and Meiqin Wang. Automatic search for a variant of division property using three subsets. In Mitsuru Matsui, editor, *Topics in Cryptology – CT-RSA 2019*, pages 412–432, Cham, 2019. Springer International Publishing.

[KW02] Lars Knudsen and David Wagner. Integral cryptanalysis. In Joan Daemen and Vincent Rijmen, editors, *Fast Software Encryption*, pages 112–127, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg.

[Lai94] Xuejia Lai. *Higher Order Derivatives and Differential Cryptanalysis*, pages 227–233. Springer, Boston, MA., 1994.

[PJ12] Mroczkowski Piotr and Szmidt Janusz. The cube attack on stream cipher trivium and quadraticity tests. *Fundamenta Informaticae*, 114(3-4):309–318, 2012.

[SBD$^+$16] Md Iftekhar Salam, Harry Bartlett, Ed Dawson, Josef Pieprzyk, Leonie Simpson, and Kenneth Koon-Ho Wong. Investigating cube attacks on the authenticated encryption stream cipher acorn. In Lynn Batten and Gang Li, editors, *Applications and Techniques in Information Security*, pages 15–26, Singapore, 2016. Springer Singapore.

[Sun21] Yao Sun. Cube attack against 843-round trivium. Cryptology ePrint Archive, Report 2021/547, 2021. https://ia.cr/2021/547.

[SWW17] Ling Sun, Wei Wang, and Meiqin Wang. Automatic search of bit-based division property for arx ciphers and word-based division property. In Tsuyoshi Takagi and Thomas Peyrin, editors, *Advances in Cryptology – ASIACRYPT 2017*, pages 128–157, Cham, 2017. Springer International Publishing.

[TIHM17] Yosuke Todo, Takanori Isobe, Yonglin Hao, and Willi Meier. Cube attacks on non-blackbox polynomials based on division property. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology – CRYPTO 2017*, pages 250–279, Cham, 2017. Springer International Publishing.

[TM16] Yosuke Todo and Masakatu Morii. Bit-based division property and application to simon family. In Thomas Peyrin, editor, *Fast Software Encryption*, pages 357–377, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.

[Tod15] Yosuke Todo. Structural evaluation by generalized integral property. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology – EUROCRYPT 2015*, pages 287–314, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.

[WHG$^+$19] Senpeng Wang, Bin Hu, Jie Guan, Kai Zhang, and Tairong Shi. Milp-aided method of searching division property using three subsetsÂăand applications. In Steven D. Galbraith and Shiho Moriai, editors, *Advances in Cryptology – ASIACRYPT 2019*, pages 398–427, Cham, 2019. Springer International Publishing.

[WHT$^+$18] Qingju Wang, Yonglin Hao, Yosuke Todo, Chaoyun Li, Takanori Isobe, and Willi Meier. Improved division property based cube attacks exploiting algebraic properties of superpoly. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology – CRYPTO 2018*, pages 275–305, Cham, 2018. Springer International Publishing.

[XZBL16] Zejun Xiang, Wentao Zhang, Zhenzhen Bao, and Dongdai Lin. Applying milp method to searching integral distinguishers based on division property for 6 lightweight block ciphers. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *Advances in Cryptology – ASIACRYPT 2016*, pages 648–678, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.

[YLL19] Jingchun Yang, Meicheng Liu, and Dongdai Lin. Cube cryptanalysis of round-reduced acorn. In Zhiqiang Lin, Charalampos Papamanthou, and Michalis Polychronakis, editors, *Information Security*, pages 44–64, Cham, 2019. Springer International Publishing.

[YT19] Chen-Dong Ye and Tian Tian. Revisit division property based cube attacks: Key-recovery or distinguishing attacks? In *IACR Transactions on Symmetric Cryptology*, pages 81–102, 2019.

[YT20] Chen-Dong Ye and Tian Tian. A practical key-recovery attack on 805-round trivium. Cryptology ePrint Archive, Report 2020/1404, 2020. https://eprint.iacr.org/2020/1404.