A Practical Key-Recovery Attack on 805-Round Trivium*

Chen-Dong Ye and Tian Tian(✉)

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

Abstract. The cube attack is one of the most important cryptanalytic techniques against Trivium. Many key-recovery attacks based on cube attacks have been established. However, few attacks can recover the 80-bit full key information practically. In particular, the previous best practical key-recovery attack was on 784-round Trivium proposed by Fouque and Vannet at FSE 2013. To mount practical key-recovery attacks, it requires a sufficient number of low-degree superpolies. It is difficult both for experimental cube attacks and division property based cube attacks with randomly selected cubes due to lack of efficiency. In this paper, we give a new algorithm to construct candidate cubes targeting linear superpolies. Our experiments show that the success probability is 100% for finding linear superpolies using the constructed cubes. We obtain over 1000 linear superpolies for 805-round Trivium. With 42 independent linear superpolies, we mount a practical key-recovery attack on 805-round Trivium, which increases the number of attacked rounds by 21. The complexity of our attack is $2^{41.40}$, which could be carried out on a PC with a GTX-1080 GPU in several hours.

Keywords: Cube Attacks· Key-Recovery Attacks· Trivium· Heuristic Algorithm· Möbius Transformation.

1 Introduction

Trivium [2] is a bit-oriented synchronous stream cipher designed by De Cannière and Preneel, which is one of the eSTREAM hardware-oriented finalists and an International Standard under ISO/IEC 29192-3:2012. Due to the simple structure and high level security, Trivium attracts a lot of attention.

The cube attack, first proposed by Dinur and Shamir in [4], is a powerful key-recovery attack against Trivium. There are two main phases in a cube attack. In the first phase, called the preprocessing phase, one needs to find appropriate cubes and recover their superpolies which are generally low-degree polynomials in key variables. In the second phase, called the on-line phase, by querying the encryption oracle, one could evaluate the superpolies under the real key and so obtain a system of equations in key variables. Then, by solving the obtained

* Supported by the National Natural Science Foundations of China under grant nos. 61672533.

system of equations, some bits of information in key or even the whole key could be recovered. Since proposed, many improvements have been established on cube attacks such as cube testers [1], dynamic cube attacks [5, 3, 17], conditional cube attacks [10, 13], division property based cube attacks [21, 22, 24, 28, 25, 8, 9] and correlation cube attacks [15].

Most of the previous work tried to recover the ANFs of the superpolies such that the number of initialization rounds as large as possible. Some attacks could only recover one or two key bits and some attacks have very marginal online complexities. For example, in [7–9], cubes of sizes over 74 were used to recover key bits for 840-, 841- and 842-round Trivium. In these cases, one to three key bits could be recovered with the superpolies. Then, it needs at least 2^{77} requests to exhaustively search the remaining key bits. Thus, the total complexity is very close to that of the brute-force attack using these large cubes.

Those attacks targeting a large number of rounds do not immediately imply a practical attack. A practical key-recovery attack on Trivium is also an important security evaluation of Trivium and a measure of the improvement of cube attacks. Considering a practical key-recovery attack against Trivium, the difficulty lies in finding a sufficient number of useful superpolies. To randomly search cubes with linear superpolies for the round-reduced Trivium with over 800 rounds is almost impossible. Currently, for Trivium, the number of initialization rounds that could be reached by cube attacks with a practical complexity is 784.

How to construct useful cubes in cube attacks has long been a difficult problem. In [4] and [6], the authors provided some ideas for finding cubes with linear superpolies. More specifically, in [4], the authors proposed the random walk method. This method starts with a randomly chosen set I of cube variables. Then, an IV variable is removed randomly from I if the corresponding superpoly is constant and a randomly chosen IV variable is added to I if the corresponding superpoly is nonlinear. This process is repeated to find cubes which pass through a sufficient number of linearity tests. If it fails, then the process restarted with another initial I. With this method, for 767-round Trivium, 35 linear superpolies were found. In [6], the authors proposed to construct a candidate large cube by disjoint union of two subcubes yielding 12 zero polynomials on some specific internal state bits determined by the recursive relation of the six bits involved in the output function. As a result, for 784-round Trivium, they found 42 linear superpolies. Furthermore, for 799-round Trivium, the authors declared that the only way linear superpolies have been found was using this method to construct cubes.

Besides, the idea of Greedy algorithm has been found useful in constructing cube distinguishers. In [19], the authors first proposed the GreedyBitSet algorithm to construct cube distinguishers and nonrandomness detectors. Later, in [18], based on the work in [19], the authors studied the state biases as well as keystream biases. As a result, they obtained cube distinguishers for 829-round Trivium and 850-round TriviA-SC. In [12], combining the GreedyBitSet algorithm with the degree evaluation method proposed in [14], the authors improved the work in [18]. As a result, they found good distinguishers on Trivium, Kreyvium

and ACORN. In particular, they provided a zero-sum distinguisher on 842-round Trivium and a significant non-randomness up to 850-round Trivium.

1.1 Our Contributions

This paper is devoted to practical key-recovery attacks against 805-round Trivium. To achieve this goal, the key problem is to find lots of cubes with linear superpolies. As mentioned above, this is quite difficult when the number of round is over 800. Our main contribution is to propose a new method to construct cubes, which is experimentally verified to be quite effective. It consists of the following three aspects.

A Heuristic Algorithm to Construct Candidate Cubes. By combining the GreedyBitSet algorithm with the division property, we propose a new algorithm to construct cubes targeting linear superpolies. The new algorithm begins with a small set of cube variables and then extends it iteratively. More specifically, there are mainly two stages in our algorithm. During the first stage, we select an IV variable (called 'steep IV variable' in this paper) which could decrease the degrees of the superpolies as fast as possible in each iteration. If we fail in the first stage, then we step into the second stage, where we pick up IV variables (called 'gentle IV variables' in this paper) which decrease the degrees of the superpolies as slowly as possible. Benefited from this two-stage algorithm, we could successfully construct cubes such that degrees of the superpolies are close to 1. Note that, the idea of this algorithm is also applicable to other NFSR-based stream ciphers.

The Preference Bit and an Algorithm to Predict It. Note that all known linear superpolies of Trivium are very sparse, and the output bit function of Trivium is the XOR of six internal state bits. It is thought that a linear superpoly probably comes from a single internal state bit. Hence, to determine a proper starting set of the above new algorithm, we propose the concept of the preference bit. Based on the structure analysis of Trivium, an iterative algorithm is provided to roughly predict the preference bit of r-round Trivium. The experimental results show that our algorithm could predict the preference bit with a success probability 75.3%.

The Improved Möbius Transformation. In cube attacks, the Möbius transformation is a powerful tool which could be used to test all the subcubes of a large cube simultaneously. However, its memory complexity is very large. To reduce the memory complexity, we divide the original Möbius transformation into two stages. Let $f(x_0, x_1, \ldots, x_{n-1})$ be a Boolean function on $x_0, x_1, \ldots, x_{n-1}$. Let q be a positive integer less than $n-1$. In the first stage, the Möbius transformations of $f(x_0, \ldots, x_{n-q-1}, 0, 0, \ldots, 0)$, $f(x_0, \ldots, x_{n-q-1}, 1, 0, \ldots, 0), \ldots, f(x_0, \ldots, x_{n-q-1}, 1, 1, \ldots, 1)$ are calculated and only a part of each Möbius transformation is stored. In the second stage, based on these partly stored transformations, we could recover a part of the ANF of f with a method similar to the Möbius transformation of a q-variable Boolean function. With this technique, the memory complexity could be decreased from 2^n bits to about 2^{n-q} bits. When it comes to practical cube attacks, this method enables us to test a large

number of subcubes of a large cube set at once with a reasonable memory complexity. For instance, we could simultaneously test $2^{32.28}$ subcubes of a cube set of size 43 with less than 9 GBs memory, while testing such a cube with the original Möbius transformation requires 2^{43} bits (1024 GBs) memory.

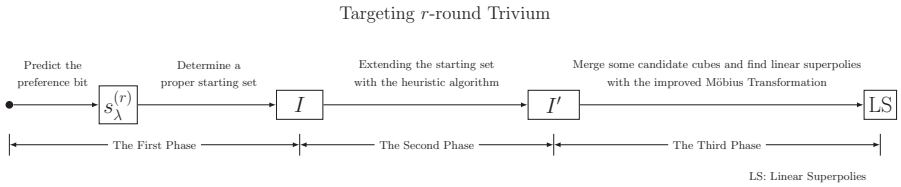

Fig. 1. The sketch of our idea

As an illustration, we apply our methods , whose sketch is shown in Fig. 1, to 805-round Trivium. As a result, we obtain more than 1000 cubes with linear superpolies for 805-round Trivium. Among these linear superpolies, there are 38 linearly independent superpolies. Besides, by sliding some cubes of 805-round Trivium to 806-round Trivium, we easily obtain several linear superpolies for 806-round Trivium. Based on the linear superpolies of 805- and 806-round Trivium, 42 key bits could be recovered for 805-round Trivium with $2^{41.25}$ requests. By adding a brute-force attack, the 80-bit key could be recovered within $2^{41.40}$ requests, which could be practically implemented by a PC with a NVIDIA GTX-1080 GPU in several hours. This attack on 805-round Trivium improves the previous best practical cube attacks by 21 more rounds, and it is the first practical attack for Trivium variants with more than 800 initialization rounds. As a comparison, we summarize the cube attacks based key-recovery attacks against the round-reduced Trivium in Table 1. Furthermore, to show the effectiveness of the heuristic algorithm to construct candidate cubes, we also applied our method to 810-round Trivium. By only testing one 43-dimensional cube, we find two 42-dimensional cubes with linear superpolies. Since it is almost impossible to find a linear superpoly for 810-round Trivium by random walk algorithm in [4] and the disjoint union method in [6], it is shown that the new heuristic algorithm to construct candidate cubes is powerful.

1.2 Organisation

The rest of this paper is organized as follows. In Section 2, we give some basic definitions and concepts. In Section 3, we show an algorithm to construct cubes which are potential to have linear superpolies. In Section 4, we propose an improved Möbius transformation which enables us to test a large mount of subcubes of a large cube simultaneously with a reasonable memory complexity. In Section 5, we apply our method to round-reduced Trivium and establish a

Table 1. A summary of key-recovery attacks on Trivium

Attack type	# of rounds	Off-line phase		On-line phase	Total time	ref.
		cube size	# of key bits			
Practical	672	12	63	2^{17}	$2^{18.56}$	[4]
	709	22-23	79	< 2	$2^{29.14}$	[16]
	767	28-31	35	2^{45}	$2^{45.00}$	[4]
	784	30-33	42	2^{38}	2^{39}	[6]
	805	32-38	42	2^{38}	$2^{41.40}$	Sect. 5
Not practical	799	32-37	18	2^{62}	$2^{62.00}$	[6]
	802	34-37	8	2^{72}	$2^{72.00}$	[27]
	805	28	7	2^{73}	$2^{73.00}$	[15]
	806	34-37	16	2^{64}	2^{64}	Sect. 5
	835	35	5	2^{75}	$2^{75.00}$	[15]
	832	72	1	2^{79}	$2^{79.01}$	[25, 21, 22]
	832	72	> 1	2^{79}	$< 2^{79.01}$	[29]
	840	78	1	2^{79}	$2^{79.58}$	[8]
	840	75	3	2^{77}	$2^{77.32}$	[9]
	841	78	1	2^{79}	$2^{79.58}$	[8]
	841	76	2	2^{78}	$2^{78.58}$	[9]
	842	78	1	2^{79}	$2^{79.58}$	[7]
	842	76	2	2^{79}	$2^{78.58}$	[9]

practical cube attack on 805-round Trivium. Finally, Section 6 concludes the paper.

2 Preliminaries

In this section, we introduce some related concepts and definitions.

2.1 Specification of Trivium

Trivium is a bit-oriented synchronous stream cipher which was one of eSTREAM hardware-oriented finalists. The main building block of Trivium is a 288-bit nonlinear feedback shift register. For every clock cycle there are three bits of the internal state updated by quadratic feedback functions and all the remaining bits of the internal state are updated by shifting. The internal state of Trivium 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, after 1152 initialization rounds, the key stream bits are generated by XORing six internal state bits. Algorithm 1 describes the pseudo-code of Trivium. For more details, please refer to [2].

Algorithm 1 Pseudo-code of Trivium

$(s_1, s_2, \ldots, s_{93}) \leftarrow (x_1, x_2, \ldots, x_{80}, 0, \ldots, 0)$;
$(s_{94}, s_{95}, \ldots, s_{177}) \leftarrow (v_1, v_2, \ldots, v_{80}, 0, \ldots, 0)$;
$(s_{178}, s_{179}, \ldots, s_{288}) \leftarrow (0, \ldots, 0, 1, 1, 1)$;
for i from 1 to N **do**
  $t_1 \leftarrow s_{66} \oplus s_{93} \oplus s_{91}s_{92} \oplus s_{171}$;
  $t_2 \leftarrow s_{162} \oplus s_{177} \oplus s_{175}s_{176} \oplus s_{264}$;
  $t_3 \leftarrow s_{243} \oplus s_{288} \oplus s_{286}s_{287} \oplus s_{69}$;
  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
  $(s_1, s_2, \ldots, s_{93}) \leftarrow (t_3, s_1, \ldots, s_{92})$;
  $(s_{94}, s_{95}, \ldots, s_{177}) \leftarrow (t_1, s_{94}, \ldots, s_{176})$;
  $(s_{178}, s_{179}, \ldots, s_{288}) \leftarrow (t_2, s_{178}, \ldots, s_{287})$;
end for

2.2 Cube Attacks

The idea of cube attacks was first proposed by Dinur and Shamir in [4]. In a cube attack against stream ciphers, an output bit z is described as a Boolean function f in key variables $\boldsymbol{k} = (k_0, k_1, \ldots, k_{n-1})$ and public IV variables $\boldsymbol{v} = (v_0, v_1 \ldots, v_{m-1})$, i.e.,$z = f(\boldsymbol{k}, \boldsymbol{v})$. Let $I = \{v_{i_1}, v_{i_2}, \ldots, v_{i_d}\}$ be a subset of IV variables. Then f can be rewritten as

$$f(\boldsymbol{k}, \boldsymbol{v}) = t_I \cdot p_I(\boldsymbol{k}, \boldsymbol{v}) \oplus q_I(\boldsymbol{k}, \boldsymbol{v}),$$

where $t_I = \prod_{v \in I} v$, p_I does not contain any variable in I, and each term in q_I is not divisible by t_I. It can be seen that the summation of the 2^d functions derived from f by assigning all the possible values to d variables in I equals to p_I, that is,

$$\bigoplus_{(v_{i_1}, v_{i_2}, \ldots, v_{i_d}) \in \mathbb{F}_2^d} f(\boldsymbol{k}, \boldsymbol{v}) = p_I(\boldsymbol{k}, \boldsymbol{v}).$$

The public variables in I are called *cube variables*, while the remaining IV variables are called non-cube variables. The set C_I of all 2^d possible assignments of the cube variables is called a *d-dimensional cube*, and the polynomial p_I is called the *superpoly* of C_I in f. For the sake of convenience, we also call p_I the superpoly of I in f. It is worth noting that the superpoly of I in f is a polynomial in key variables when all the non-cube variables are set to constant. In the following paper, we set the non-cube variables to 0's in default.

A cube attack consists of the preprocessing phase and the on-line phase.

- **Off-line Phase.** The attacker should find cubes whose superpolies in the output bit are low-degree polynomials.
- **On-line Phase.** For each cube obtained in the off-line phase, the attacker inquires the encryption oracle to get the cube summation under the real key. With the obtained cube summations corresponding to the previously found

cubes, a system of low-degree equations in key variables could be set up. Then, by solving this system of equations, some key bits could be recovered. Finally, by adding a brute-force attack (if there are some key bits remaining unknown), the whole key could be recovered.

2.3 The Bit-Based Division Property and A Degree Evaluation Algorithm Based on It

In [23], the authors proposed the conventional bit-based division property whose definition is as follows.

Definition 1 (Bit-Based Division Property [23]). *Let $\mathbb{X}$ be a multi-set whose elements take a value of $\mathbb{F}_2^n$. Let $\mathbb{K}$ be a set whose elements take an n-dimensional bit vector. When the multi-set $\mathbb{X}$ has the division property $D_{\mathbb{K}}^{1^n}$, it fulfills the following conditions:*

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

where $\boldsymbol{u} \succeq \boldsymbol{\alpha}$ if and only if $u_i \geq k_i$ for all i and $\boldsymbol{x}^{\boldsymbol{u}} = \prod_{i=0}^{n-1} x_i^{u_i}$.

Due to the high memory complexity, the bit-based division property was confined to be applied to small block ciphers such as SIMON32 and Simeck32 [23]. To avoid such a high memory complexity, in [26], the authors applied the mixed integer linear programming (MILP) methods to the bit-based division property. They first introduced the concept of division trails, which is defined as follows.

Definition 2 (Division Trail [26]). *Let us consider the propagation of the division property $\{\boldsymbol{\alpha}\} = \mathbb{K}_0 \to \mathbb{K}_1 \to \mathbb{K}_2 \cdots \to \mathbb{K}_r$. Moreover, for any vector $\boldsymbol{\alpha}_{i+1}^* \in \mathbb{K}_{i+1}$, there exist a vector $\boldsymbol{\alpha}_i^* \in \mathbb{K}_i$ such that $\boldsymbol{\alpha}_i^*$ can propagate to $\boldsymbol{\alpha}_{i+1}^*$ by the propagation rules of division property. Furthermore, for $(\boldsymbol{\alpha}_0, \boldsymbol{\alpha}_1, \ldots, \boldsymbol{\alpha}_r) \in \mathbb{K}_0 \times \mathbb{K}_1 \times \cdots \times \mathbb{K}_r$ if $\boldsymbol{\alpha}_i$ can propagate to $\boldsymbol{\alpha}_{i+1}$ for $i \in \{0, 1, \ldots, r-1\}$, we call $\boldsymbol{\alpha}_0 \to \boldsymbol{\alpha}_1 \to \cdots \to \boldsymbol{\alpha}_r$ an r-round division trail.*

In [26], the authors described the propagation rules for AND, COPY and XOR with MILP models, see [26] for the details. Therefore, they could build an MILP model to cover all the possible division trails generated during the propagation. Besides, in [21, 20], the authors simplified those MILP models in [26]. In particular, in [21], the division property based cube attacks were proposed for the first time and were applied to attacking Trivium, Grain-128 and Acorn successfully. Later, to improve the work of [21], in [24], the authors proposed a degree evaluation algorithm which was based on the following proposition.

Proposition 1 ([24]). *Let $f(\boldsymbol{x}, \boldsymbol{v})$ be a polynomial, where $\boldsymbol{x}$ and $\boldsymbol{v}$ denote the secret and public variables, respectively. For a set of indices $I = \{i_1, i_2, \ldots, i_{|I|}\} \subset \{1, 2, \ldots, m\}$, let C_I be a set of $2^{|I|}$ values where the variables in $\{v_{i_1}, v_{i_2}, \ldots, v_{i_{|I|}}\}$*

are taking all possible combinations of values. Let $\boldsymbol{k}_I$ *be an* m*-dimensional bit vector such that* $\boldsymbol{v}^{\boldsymbol{I}} = t_I = v_{i_1} v_{i_2} \cdots v_{i_{|I|}}$*. Let* $\boldsymbol{k}_\Lambda$ *be an* n*-dimensional bit vector. If there is no division trail such that* $(\boldsymbol{k}_\Lambda || \boldsymbol{k}_I) \xrightarrow{f} 1$*, then the monomial* $x^{\boldsymbol{k}_\Lambda}$ *is not involved in the superpoly of the cube* C_I*.*

If there is $d \geq 0$ such that for all $\boldsymbol{k}_\Lambda$ of Hamming Weight $hw(\boldsymbol{k}_\Lambda) > d$, the division trail $x^{\boldsymbol{k}_\Lambda}$ does not exist, then it can be seen that d is an upper bound of the algebraic degree of the superpoly. With the MILP method, this d can be naturally modeled as the maximum of the objective function $\sum_{j=1}^{n} x_j$. Therefore, for a given set of cube variables, by solving MILP models, an upper bound of the degree of the superpoly could be obtained. For more details, please refer to Section 4 of [24]. In the following paper, we shall combine this algorithm with some greedy strategies to find cubes with linear superpolies.

2.4 The Möbius Transformation

In [5], Dinur and Shamir suggested using the Möbius transformation to compute all possible subcubes of a large cube at once. Later, in [6], the author showed some ways to use the Möbius transformation in cube attacks on Trivium.

Let f be a polynomial in $\mathbb{F}_2[x_0, x_1, \ldots, x_{n-1}]$, whose algebraic normal form is given by

$$f(x_0, \ldots, x_{n-1}) = \bigoplus_{\boldsymbol{c}=(c_0,\ldots,c_{n-1}) \in \mathbb{F}_2^n} g(c_0, \ldots, c_{n-1}) \prod_{i=0}^{n-1} x_i^{c_i},$$

where the function g giving the coefficient of each term $\prod_{i=0}^{n-1} x_i^{c_i}$ is the Möbius transformation of f. With the knowledge of the truth table of f, one could calculate the ANF of f by using the Möbius transform, see Algorithm 2.

Algorithm 2 The Möbius transformation algorithm

Require: Truth Table S of f with 2^n entries
1: **for** i from 0 to $n-1$ **do**
2: Let $Sz \leftarrow 2^i$, $Pos \leftarrow 0$
3: **while** $Pos < 2^n$ **do**
4: **for** $j = 0$ to $Sz - 1$ **do**
5: $S[Pos + Sz + j] \leftarrow S[Pos + j] \oplus S[Pos + Sz + j]$
6: **end for**
7: Let $Pos \leftarrow Pos + 2 \cdot Sz$
8: **end while**
9: **end for**

For Algorithm 2, it can be found that it needs to store the whole truth table of f and so a large mount of memory is needed. Specifically, for an n-variable polynomial f, it requires 2^n bits of memory. Furthermore, the computational

complexity of Algorithm 2 is $n \cdot 2^n$ basic operations, since the innermost loop is executed $n \cdot 2^{n-1}$ times, which consists of a single assignment and a XOR operation. It is worth noting that Algorithm 2 could be accelerated. For instance, a 32-bit implementation is presented in [11] which performs roughly 32 times less operations, and so has a complexity of $n \cdot 2^{n-5}$ operations.

Now we consider the application of the Möbius transformation to cube attacks. Assume that $f(k_0, k_1, \ldots, k_{n-1}, v_0, v_1 \ldots, v_{m-1})$ is the output bit of a cipher in key variables $k_0, k_1, \ldots, k_{n-1}$ and IV variables $v_0, v_1 \ldots, v_{m-1}$. Let $I = \{v_{i_1}, v_{i_2}, \ldots, v_{i_d}\}$ be a set of cube variables. When all the other variables are set to constants, the output bit function f is reduced to a polynomial f' on cube variables in I only. Given the truth table of f', by using the Möbius transformation, the ANF of f' could be recovered. Note that, for a subset I' of I, the coefficient of the term $\prod_{v \in I'} v$ is the value of $p_{I'}$ when the variables in $I \setminus I'$ are set to 0's, where $p_{I'}$ is the superpoly of I' in f. Based on this fact, with the Möbius transformation, experimental tests such as linearity tests and quadratic tests could be done at once for all the subcubes of a large set of cube variables. It can be seen that the Möbius transformation makes finding linear/quadratic superpolies easier and so improves the efficiency of cube attacks.

3 Construct Potentially Good Cubes

Finding cubes which could be used to mount key-recovery attacks is a tough task in cube attacks. Collecting enough such cubes to establish practical attacks is even more difficult. In this section, combining the idea of GreedyBitSet algorithm with division property, we first devote to constructing cubes which are potential to have linear superpolies[1] through extending a starting cube set iteratively. Then, to obtain a proper starting cube set, we propose the concept of the preference bit and present an algorithm to predict the preference bit based on a structural analysis of Trivium. Combining these ideas, we could construct potentially good cubes successfully.

3.1 A Heuristic Algorithm of Constructing Cubes

In cube attacks, linear superpolies are of significance since linear equations in key variables could be set up based on linear superpolies. To construct cubes which potentially have linear superpolies, we combine the division property with heuristic algorithms to extend a small set of cube variables iteratively. Before illustrating our idea, we shall first give the following definitions.

Definition 3 (Steep IV Variable). *Let $I = \{v_{i_1}, v_{i_2}, \ldots, v_{i_\ell}\}$ be a set containing ℓ cube variables. Then, an IV variable $b \in B = \{v_0, v_1, \ldots, v_{m-1}\} \setminus I$ is called a steep IV variable of I if $ds(I \cup \{b\}) = \min\{ds(I \cup \{v\}) | v \in B\}$, where $ds(I)$ is the degree of the superpoly of I in key variables.*

[1] Constant polynomials are also linear. However, key bits could not be recovered from constant superpolies directly. Hence, in this paper, when talking about linear superploies, we do not take the constant linear into consideration.

Let I be a starting set of cube variables, which is a small set. It can be seen that a steep IV variable of I is exactly the one which makes the degree of the superpoly decrease most. To construct a cube with a linear superpoly from I, a natural idea is to extend I iteratively, where a steep IV variable is added to the current set I in each iteration. With this strategy, the degree of superpoly could be decreased fast. However, decreasing the degree of the superpoly too fast sometimes brings troubles to constructing cubes with linear superpolies. Assume that I' is constructed from I after several iterations, where a steep IV variable is added in each iteration. Let v be a steep IV variable of I'. It is possible that $\mathrm{ds}(I' \cup v) = 0$, while $\mathrm{ds}(I') > 5$. It indicates that adding a steep IV variable could make the degree of the superpoly decrease to 0 suddenly. Hence, it may fail to construct cubes with linear superpolies by only adding steep IV variables. We perform experiments on Trivium and the results show that this phenomenon happens frequently. We provide a concrete example happening in the case of 805-round Trivium, see Example 1.

Example 1. For 805-round Trivium, we try to construct a good cube by extending $\{v_4, v_6, v_{10}, v_{11}, v_{15}, v_{17}, v_{19}, v_{21}, v_{25}, v_{29}, v_{32}, v_{34}, v_{36}, v_{39}, v_{41}, v_{43}, v_{50}\}$. After 16 iterations, we obtain the set

$$I' = \{v_4, v_6, v_{10}, v_{11}, v_{15}, v_{17}, v_{19}, v_{21}, v_{25}, v_{29}, v_{32}, v_{34}, v_{36}, v_{39}, v_{41}, v_{43}, v_{50},$$

$$v_2, v_{69}, v_{79}, v_8, v_{27}, v_0, v_1, v_{28}, v_{71}, v_{13}, v_{45}, v_{23}, v_{26}, v_{38}, v_{76}, v_{47}\}$$

by adding a steep IV variable in each iteration. The degree of $p_{I'}$ is upper bounded by 9. For I', v_{56} is a steep IV variable. However, after adding v_{56} to I', the degree of $p_{I' \cup \{v_{56}\}}$ is 0. Namely, v_{56} decreases the degree of the superpoly from 9 to 0 suddenly. It indicates that we fail to construct a cube with a linear superpoly in the output of 805-round Trivium by only adding steep IV variables.

Recall that our aim is to construct cubes with linear superpolies rather than those with zero-constant superpolies. From Example 1, it can be seen that always adding a steep IV variable does make our aim break sometimes. To solve this problem, we propose the concept of gentle IV variables which decrease the degree of the superpoly slowly. We formally describe the definition of the gentle IV variable in Definition 4.

Definition 4 (Gentle IV Variable). *Let $I = \{v_{i_1}, v_{i_2}, \ldots, v_{i_\ell}\}$ be a set containing ℓ cube variables. Then, an IV variable $b \in B$ is called a gentle IV variable of I if $ds(I \cup \{b\}) = \max\{ds(I \cup \{v\}) | ds(I \cup \{v\}) \leq ds(I), v \in B\}$, where $B = \{v_0, v_1, \ldots, v_{m-1}\} \setminus I$ and $ds(I)$ is the degree of the superpoly of I.*

It can be seen from Definition 4 that a gentle IV variable of I is exactly the one which could decrease the degree of the superpoly as slowly as possible. With gentle IV variables, the above phenomenon could be avoided by adding gentle IV variables instead of steep IV variables to I', where I' is obtained by adding steep IV variables to I after several iterations.

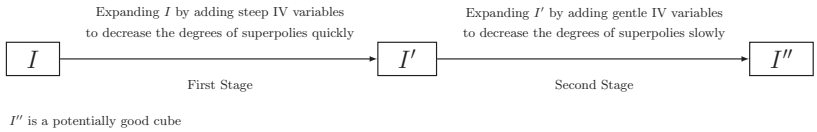

Fig. 2. The sketch of our idea

Based on the above ideas, we propose a new heuristic algorithm to construct cubes with linear superpolies. The sketch of our idea is shown in Fig. 2. Algorithm 3 describes the details of our idea. In Algorithm 3, similar to the GreedyBitSet algorithm proposed in [19], we start with a small starting set of cube variables. Then, there are two stages in Algorithm 3. During the first stage, a steep IV variable is added to the current set I of cube variables so that the degree of the superpoly could be decreased as fast as possible. To determine the steep IV variable of I, we use the degree evaluation method based on division property, which was proposed in [24], to calculate the upper bound of $ds(I \cup v)$ for each IV variable which is not in I. As illustrated above, if only steep IV variables are added, the degree of the superpoly may be decreased to 0 suddenly and so constructing cubes with linear superpolies fails. If so, Algorithm 3 would step into the second stage, where we hope to decrease the degree of the superpoly slowly. During the second stage, we add the first gentle IV variable into the current cube set in each iteration. To determine the gentle IV variables, the same method in stage one is used. By gradually adding gentle IV variables, which make the degree of the superpoly decrease slowly, it is more hopeful to construct cubes with linear superpolies.

Remark 1. In the second stage of Algorithm 3, for I, it may encounter the case that $\mathrm{ds}(I \cup \{v\}) > \mathrm{ds}(I)$ or $\mathrm{ds}(I \cup \{v\}) = 0$ holds for each $v \in B$, i.e., the gentle IV variable of I may do not exist. In this case, we select the cube variable b such that $\mathrm{ds}(I \cup \{b\}) = \min\{\mathrm{ds}(I \cup \{v\}) > \mathrm{ds}(I) | v \in B\}$ to update I.

Construct A Mother Cube. Note that the superpoly of the cube obtained with Algorithm 3 may be not linear still, since the division property based method only returns an upper bound of the degree of the superpoly. To make it more possible to find linear superpolies, we attempt to construct a large cube, called a mother cube in the following paper, and then use the Möbius transformation to test its subcubes simultaneously. Such a mother cube is constructed by jointing some cubes obtained in the last iteration.

Let I be the set of cube variables before the last iteration. When selecting cubes, we prefer to choose those cubes such that the degree of the corresponding superpolies are low. More specifically, for j starting from 1 incrementally, we gradually update the set I as follows until a mother cube with a desired size is obtained

$$I \leftarrow I \cup \{v \in B | \text{ the upper bound of } \mathrm{ds}(I \cup v) = j\},$$

where $B = \{v_0, v_1, \ldots, v_{m-1}\} \setminus I$. We offer a concrete example of constructing a mother cube in Subsection 5.2.

Algorithm 3 The algorithm of constructing cubes with linear superpolies

Require: a set of cube variables $I = \{v_{i_1}, \ldots, v_{i_c}\}$ of size c and the target round r

$B \leftarrow \{v_0, v_1, \ldots, v_{m-1}\} \setminus I$;
$d \leftarrow +\infty$;
/* The first stage */
while $d > 1$ and $|I|$ is less than a given bound **do**
  for $v \in B$ **do**
    Estimate the upper bound of ds($I \cup \{v\}$) using the division property based method;
  end for
  $I \leftarrow I \cup \{v\}$, where v is the first steep IV variable of I;
  $B \leftarrow B \setminus v$;
  $d \leftarrow DS(I \cup \{v\})$, where $DS(I \cup \{v\})$ is the upper bound of ds($I \cup \{v\}$)
end while
if $d(I) == 1$ **then**
  return I
end if
/* The second stage */
if $d(I) == 0$ **then**
  $I \leftarrow I \setminus \{v\}$, where v is the steep IV variable added in the last iteration of the first stage.
  $I \leftarrow I \cup \{v'\}$, where $DS(I \cup \{v'\})$ attains minimum except 0 in the last iteration of the first stage.
  $B \leftarrow \{v_0, v_1, \ldots, v_{m-1}\} \setminus I$;
  while $d > 1$ and $|I|$ is less than a given bound **do**
    for $v \in B$ **do**
      Estimate the upper bound of ds($I \cup \{v\}$) using the division property based method;
    end for
    $I \leftarrow I \cup \{v\}$, where v is the first gentle IV variable
    $B \leftarrow B \setminus v$;
    $d \leftarrow DS(I \cup \{v\})$
  end while
end if

3.2 Determine Starting Cube Sets

One critical point of Algorithm 3 is that it requires a small set of cube variables as its input. In this subsection, based on careful analysis of the structure of Trivium, we shall present a method to determine a proper starting set of cube variables to make Algorithm 3 work well.

Recall that the output function of r-round Trivium is the linear combination of six internal state bits, i.e, $z_r = \bigoplus_{j=1}^{6} s_{\lambda_j}^{(r)}$, where $\{\lambda_1, \lambda_2, \lambda_3, \lambda_4, \lambda_5, \lambda_6\} = \{66, 93, 162, 177, 243, 288\}$.

It is worth noting that all the known linear superpolies of Trivium are sparse, and most of them contain only a single key variable. It is very likely that there exists some $j \in \{1, 2, 3, 4, 5, 6\}$ such that $p_I = p_{\lambda_j}$ and $p_{i_\ell} = 0$ for $\ell \neq j$, where p_{λ_ℓ} is the superpoly of I in $s_{\lambda_\ell}^{(r)}$ for $\ell \in \{1, 2, 3, 4, 5, 6\}$. In this paper, for a set of cube variables I, if there exists some $j \in \{1, 2, 3, 4, 5, 6\}$ such that $p_I = p_{\lambda_j}$ and $p_{\lambda_\ell} = 0$ for $\ell \neq j$ then we say that the superpoly p_I comes from $s_{\lambda_j}^{(r)}$. The following is an illustrative example.

Example 2. For 769-round Trivium, the superpoly of

$$I = \{v_1, v_3, v_5, v_7, v_{10}, v_{12}, v_{14}, v_{16}, v_{18}, v_{20}, v_{23}, v_{26}, v_{30}, v_{39}, v_{41},$$
$$v_{42}, v_{43}, v_{47}, v_{50}, v_{52}, v_{53}, v_{55}, v_{58}, v_{60}, v_{61}, v_{64}, v_{69}, v_{71}, v_{78}\}$$

in the output bit z_{769} is $p_I = k_{22}$. We figure out the superpolies of I in $s_{66}^{(769)}$, $s_{93}^{(769)}, s_{162}^{(769)}, s_{177}^{(769)}, s_{243}^{(769)}, s_{288}^{(769)}$, respectively. The results show that only $p_{66} = k_{22}$ is linear and the rest five superpolies are 0's . Namely, the linear superpoly k_{22} comes from $s_{66}^{(769)}$.

Determine a Proper Set of Cube Variables. Inspired by the phenomenon mentioned above, when constructing cubes with linear superpolies, we could focus on only one of the six internal state bits in the output function. In the following, we shall illustrate how to determine a proper set of cube variables. Assume that $s_\lambda^{(r)}$ is the chosen target for r-round Trivium. First, according to the update function of Trivium, $s_\lambda^{(r)}$ could be written as

$$s_\lambda^{(r)} = s_{j_1^\lambda}^{(r-\lambda)} \cdot s_{j_2^\lambda}^{(r-\lambda)} \oplus s_{j_3^\lambda}^{(r-\lambda)} \oplus s_{j_4^\lambda}^{(r-\lambda)} \oplus s_{j_5^\lambda}^{(r-\lambda)}. \tag{1}$$

Then, we choose a set I of cube variables and search all its subcubes to find those cubes having linear superpolies in $s_{j_1^\lambda}^{(r-\lambda)}$ or $s_{j_2^\lambda}^{(r-\lambda)}$ with the Möbius transformation. If such subcubes are found, then we randomly choose one of them to be the starting set of Algorithm 3.

Assume that the superpoly $p_{I'}$ of $I' = \{v_{l_1}, v_{l_2}, \ldots, v_{l_u}\} \subseteq I$ in $s_{j_1^\lambda}^{(r-\lambda)}$ is linear. Then, $s_{j_1^\lambda}^{(r-\lambda)}$ could be rewritten as

$$s_{j_1^\lambda}^{(r-\lambda)}(\boldsymbol{k}, \boldsymbol{v}) = g(\boldsymbol{k}, \boldsymbol{v}) \cdot t_{I'} \cdot p_{I'}(\boldsymbol{k}) \oplus q_{I'}(\boldsymbol{k}, \boldsymbol{v}),$$

where $t_{I'} = \prod_{i=1}^{u} v_{l_i}$. Since $s_{j_1^\lambda}^{(r-\lambda)} \cdot s_{j_2^\lambda}^{(r-\lambda)}$ is the only term of degree 2 in Equ. (1), it is hopeful that we could extend I' to I whose superpoly in $s_\lambda^{(r)}$ is linear. Due to the above phenomenon, it is hopeful that the superpoly of I in the output bit is linear as well. The following is an illustrative example.

Example 3. In the case of 805-round Trivium, the superpoly of

$$I = \{v_4, v_6, v_{10}, v_{11}, v_{15}, v_{17}, v_{19}, v_{21}, v_{25}, v_{29}, v_{32}, v_{34}, v_{36}, v_{39}, v_{41}, v_{43}, v_{50}\}$$

in $s_{286}^{(739)}$ is k_{56}. Furthermore, we find that the superpoly of

$$\begin{aligned} I'' = \{&v_1, v_2, v_4, v_6, v_8, v_{10}, v_{11}, v_{13}, v_{15}, v_{17}, v_{19}, v_{21}, v_{23}, \\ &v_{25}, v_{26}, v_{27}, v_{29}, v_{32}, v_{34}, v_{36}, v_{38}, v_{39}, v_{41}, v_{42}, v_{43}, \\ &v_{45}, v_{47}, v_{48}, v_{50}, v_{52}, v_{57}, v_{59}, v_{69}, v_{71}, v_{76}, v_{79}\} \end{aligned}$$

is also k_{56} in the output of 805-round Trivium. Note that I'' contains all the cube variables in I. This indicates that it is reasonable to construct cubes with linear superpolies in the output bit by extending a starting cube selected in the way illustrated above.

The Preference Bit. Now, for r-round Trivium, the key point is which internal state bit in the output function should be chosen so that we could construct cubes with linear superpolies with a high success probability by extending a small set I of cube variables.

To study the difference of the six internal state bits in the output function with respect to constructing linear superpolies, we perform dedicated experiments on Trivium variants with from 400 to 699 initialization rounds. For each variant, we collect thousands of linear superpolies and check which internal state bit each linear superpoly comes from. The results show that there exists significant difference among six internal state bits in the output function with respect to where a linear superpoly comes from. For example, among the 2953 collected linear superpolies of 699-round Trivium, 2366 linear superpolies come from $s_{243}^{(699)}$, i.e., over 80% of the linear superpolies come from $s_{243}^{(699)}$. Table 2 shows the number of linear supeprolies comes from each internal state bit.

Table 2. The number of linear supeprolies coming from each internal state bit

Internal state bit	$s_{66}^{(699)}$	$s_{93}^{(699)}$	$s_{162}^{(699)}$	$s_{177}^{(699)}$	$s_{243}^{(699)}$	$s_{288}^{(699)}$
Number of linear superpolies	162	0	182	246	2366	0

For 699-round Trivium, linear superpolies come from $s_{243}^{(699)}$ most frequently. Let r be a positive integer. For r-round Trivium, the internal state bit $s_{\lambda_j}^{(r)}$ in the output function such that linear superpolies come from it most frequently is called the *preference bit* of r-round Trivium. For these 300 Trivium variants, there are 230 variants such that more than 40% of the collected linear superpolies come from the preference bit. It can be seen that the preference bit has a significant advantage over the other five internal state bits with respect to where a linear superpoly may come from. In other words, for r-round Trivium, it is more likely to construct cubes with linear superpolies when targeting the preference bit than the other internal state bits in the output function.

An Iterative Algorithm to Predict the Preference Bit. According to the above discussions, if we target the preference bit, then it is more likely to construct cubes with linear superpolies. In this subsection, we design an algorithm to pick up the preference bit among the six ones. Our algorithm is based on the following lemma.

Lemma 1. *Let $I = \{v_{i_1}, v_{i_2}, \ldots, v_{i_d}\}$ be a set of cube variables. If the superpoly of I in $f(\boldsymbol{k}, \boldsymbol{v})$ is linear in key variables, then there is a term in the form of $\prod_{v \in I} v \cdot t_v \cdot k_j$ in the ANF of f, where t_v is 1 or a product of some non-cube variables.*

Proof. Since the superpoly of I in f is linear in key variables, then there is a term in the form of $t_v \cdot k_j$ in the ANF of p_I, where t_v is 1 or the product of some

non-cube variables. Hence, there is a term in the form of $\prod_{v \in I} v \cdot t_v \cdot k_j$ in the ANF of f.

According to Lemma 1, a necessary condition that a linear superpoly comes from $s_{\lambda_j}^{(r)}$ is that $s_{\lambda_j}^{(r)}$ has a term in the form of $T_v \cdot k_j$ in its ANF, where T_v is a product of some IV variables. In the rest of this paper, such kind of term is called a VK-term for simplicity. Note that a VK-term does not always lead to a linear superpoly. For example, let $\prod_{i=1}^{u} v_{j_i} \cdot k_l$ be a VK-term. If $\prod_{i=1}^{u} v_{j_i} \cdot k_l \cdot k_h$ is in the ANF of z_r, then the superpoliy of $\{v_{j_1}, v_{j_2}, \ldots, v_{j_l}\}$ would be nonlinear, i.e. the VK-term $\prod_{i=1}^{u} v_{j_i} \cdot k_l$ does not lead to a linear superpoly. In other words, it is reasonable that the more VK-terms an internal state bit has, the more linear superpolies come from it. Thus, it is reasonable to assume that the preference bit contains the largest number of VK-terms.

However, it is impossible to accurately calculate the number of VK-terms by the ANF of an internal state bit when the number of initialization rounds is high. To solve this problem, we propose an iterative algorithm whose results could reflect the number of VK-terms in each internal state bit at a high level. With this algorithm, we could predict the preference bit for an arbitrary number of initialization rounds with a very low computing complexity.

Let $s^{(t)} = (s_1^{(t)}, s_2^{(t)}, \ldots, s_{288}^{(t)})$ be the internal state of Trivium after t rounds. Note that each internal state bit $s_j^{(t)} (1 \leq j \leq 288)$ is a polynomial in key variables and IV variables. Denote by $NVK_j^{(t)}$ the number of VK-terms in the ANF of $s_j^{(t)}$. Let $NV_j^{(t)}$ be the number of terms in the form of T_v, which are called V-terms for simplicity, in $s_j^{(t)}$, where T_v is a product of some IV variables. In the following, we take $s_{94}^{(t+1)}$ as an example to illustrate how our algorithm works. According to the update function of Trivium, $s_{94}^{(t+1)}$ is updated as $s_{94}^{(t+1)} = s_{91}^{(t)} \cdot s_{92}^{(t)} \oplus s_{93}^{(t)} \oplus s_{66}^{(t)} \oplus s_{171}^{(t)}$. In $s_{91}^{(t)} \cdot s_{92}^{(t)}$, there are three ways to generate a VK-term which are shown as follows.

- $s_{91}^{(t)}$ provides a V-term(or constant 1) and $s_{92}^{(t)}$ provides a VK-term;
- $s_{91}^{(t)}$ provides a VK-term and $s_{92}^{(t)}$ provides a V-term(or constant 1);
- $s_{91}^{(t)}$ and $s_{92}^{(t)}$ both provide VK-terms, where the key variable in these two VK-terms are the same.

Generally, the VK-terms formed in the third way are much fewer than those formed in the first two ways. Besides, the VK-terms obtained by multiplying constant 1 with VK-terms are also much fewer than those obtained by multiplying a V-term and a VK-term. Hence, in our algorithm, we regard $NV_{91}^{(t)} \cdot NVK_{92}^{(t)} + NV_{92}^{(t)} \cdot NVK_{91}^{(t)}$ as the number of VK-terms in $s_{91}^{(t)} \cdot s_{92}^{(t)}$ which is denoted by $NVK(s_{91}^{(t)} \cdot s_{92}^{(t)})$. Namely, $NVK(s_{91}^{(t)} \cdot s_{92}^{(t)})$ is set as

$$NVK(s_{91}^{(t)} \cdot s_{92}^{(t)}) \leftarrow NV_{91}^{(t)} \cdot NVK_{92}^{(t)} + NV_{92}^{(t)} \cdot NVK_{91}^{(t)}{}^{1}.$$

[1] Here, we only consider the VK-terms formed in the first two ways and do not take the terms which are eliminated by the XOR operation into consideration.

Consequently, $NVK_{94}^{(t+1)}$ is set as

$$NVK_{94}^{(t+1)} \leftarrow NVK(s_{91}^{(t)} \cdot s_{92}^{(t)}) + NVK_{93}^{(t)} + NVK_{66}^{(t)} + NVK_{171}^{(t)}.$$

Note that, to calculate $NVK_{94}^{(t+1)}$, it needs to know $NV_{91}^{(t)}$ and $NV_{92}^{(t)}$. Hence, it is necessary to calculate $NV_{94}^{(t+1)}$ as well. According to the update function, $NV_{94}^{(t+1)}$ could be set as

$$NV_{94}^{(t+1)} \leftarrow NV_{91}^{(t)} \cdot NV_{92}^{(t)} + NV_{93}^{(t)} + NV_{66}^{(t)} + NV_{171}^{(t)},$$

since the number of V-terms in $s_{91}^{(t)} \cdot s_{92}^{(t)}$ is dominated by those formed from multiplying two V-terms together.

Moreover, $NVK_1^{(t+1)}, NV_1^{(t+1)}, NVK_{178}^{(t+1)}, NV_{178}^{(t+1)}$ could be calculated in a similar way. Thus, we could update $NVK^{(t+1)}, NV^{(t+1)}$ from $NVK^{(t)}, NV^{(t)}$, where $NVK^{(t)} = (NVK_1^{(t)}, \ldots, NVK_{288}^{(t)})$, and $NV^{(t)} = (NV_1^{(t)}, \ldots, NV_{288}^{(t)})$.

Now, the remaining problem is how to initialize $NVK^{(0)}$ and $NV^{(0)}$. To obtain a more accurate result, we initialize $NVK^{(280)}$ and $NV^{(280)}$ by calculating the ANFs of $s_1^{(280)}, s_2^{(280)}, \ldots, s_{288}^{(280)}$. With the above method, we could figure out $NVK_j^{(r)}$ for $1 \leq j \leq 288$ gradually. Finally, the bit indexed by $j \in \{66, 93, 162, 177, 243, 288\}$ such that

$$NVK_j^{(r)} = \max\{NVK_\lambda^{(r)} | \lambda \in \{66, 93, 162, 177, 243, 288\}\}$$

is predicted as the preference bit. We formally describe our idea in Algorithm 4.

Algorithm 4 The algorithm of predicting the preference bit

Calculate the ANFs of $s_i^{(280)}$ to initialise $NVK^{(280)}$ and $NV^{(280)}$;
for $280 \leq t \leq r-1$ **do**
  $NVK_{t_1} \leftarrow NV_{91}^{(t)} \cdot NVK_{92}^{(t)} + NV_{92}^{(t)} \cdot NVK_{91}^{(t)} + NVK_{93}^{(t)} + NVK_{66}^{(t)} + NVK_{171}^{(t)}$;
  $NV_{t_1} \leftarrow NV_{91}^{(t)} \cdot NV_{92}^{(t)} + NV_{93}^{(t)} + NV_{66}^{(t)} + NV_{171}^{(t)}$;
  $NVK_{t_2} \leftarrow NV_{175}^{(t)} \cdot NVK_{176}^{(t)} + NV_{176}^{(t)} \cdot NVK_{175}^{(t)} + NVK_{177}^{(t)} + NVK_{162}^{(t)} + NVK_{264}^{(t)}$;
  $NV_{t_2} \leftarrow NV_{175}^{(t)} \cdot NV_{176}^{(t)} + NV_{177}^{(t)} + NV_{162}^{(t)} + NV_{264}^{(t)}$;
  $NVK_{t_3} \leftarrow NV_{286}^{(t)} \cdot NVK_{287}^{(t)} + NV_{287}^{(t)} \cdot NVK_{286}^{(t)} + NVK_{288}^{(t)} + NVK_{243}^{(t)} + NVK_{69}^{(t)}$;
  $NV_{t_3} \leftarrow NV_{286}^{(t)} \cdot NV_{287}^{(t)} + NV_{288}^{(t)} + NV_{243}^{(t)} + NV_{69}^{(t)}$;
  for $288 \geq j \geq 2$ **do**
    $NVK_j^{(t)} \leftarrow NVK_{j-1}^{(t)}$;
    $NV_j^{(t)} \leftarrow NV_{j-1}^{(t)}$;
  end for
  $NV_{94}^{(t)} \leftarrow NV_{t_1}$; $NV_{178}^{(t)} \leftarrow NV_{t_2}$; $NV_1^{(t)} \leftarrow NV_{t_3}$;
  $NVK_{94}^{(t)} \leftarrow NVK_{t_1}$; $NVK_{178}^{(t)} \leftarrow NVK_{t_2}$; $NVK_1^{(t)} \leftarrow NVK_{t_3}$;
end for
Choose the bit $s_b^{(t)}$ such that

$$NVK_b^{(t)} = \max\{NVK_\lambda^{(t)} | \lambda \in \{66, 93, 162, 171, 243, 288\}\}$$

as the preference bit, where $b \in \{66, 93, 162, 171, 243, 288\}$;

4 An Improved Möbius Transformation

The Möbius transformation is a powerful tool which could be used to search all the subcubes of a large cube at once. It improves the efficiency of cube attacks a lot. Note that, for Trivium variants with more than 800 initialization rounds, the sizes of all known cubes with linear superpolies are larger than 30. Hence, to find linear superpolies, for a large cube set I, it is not necessary to test its subcubes of small sizes, and only subcubes of large sizes should be taken into consideration. However, in the original Möbius transformation, to test all the subcubes of I, the memory complexity is $O(2^{|I|})$ which expands exponentially as $|I|$ increases. In this section, we shall present an improved Möbius transformation which could recover a part of ANF of $f(x_0, x_1, \ldots, x_{n-1})$ according to the truth table of $f(x_0, x_1, \ldots, x_{n-1})$. With the improved Möbius transformation, we could test a large number of subcubes of I simultaneously with a reasonable memory complexity.

Let $f(x_0, x_1, \ldots, x_{n-1})$ be a Boolean function on $x_0, x_1, \ldots, x_{n-1}$. The ANF of f is obtained by writing

$$f = \bigoplus_{(c_0,c_1,\ldots,c_{n-1})\in\mathbb{F}_2^n} g(c_0, c_1, \ldots, c_{n-1}) \prod_{i=0}^{n-1} x_i^{c_i}.$$

Recall that the function g is the Möbius transformation of f. It can be seen that the Möbius transformation g is actually a Boolean function on n variables. Furthermore, the Möbius transformations of $f(x_0, \ldots, x_{n-1})$, $f(x_0, \ldots, x_{n-2}, 0)$, and $f(x_0, \ldots, x_{n-2}, 1)$ are closely related, i.e. the Möbius transformation of $f(x_0, x_1, \ldots, x_{n-1})$ could be obtained from the Möbius transformations of $f(x_0, x_1, \ldots, x_{n-2}, 0)$ and $f(x_0, x_1, \ldots, x_{n-2}, 1)$, see Chapter 9.2 of [11] for details. Actually, it could be generalised, see Corollary 1.

Corollary 1. *Let $f(x_0, x_1, \ldots, x_{n-1})$ be a Boolean function on $x_0, x_1, \ldots, x_{n-1}$. Assume that $g_0, g_1, \ldots, g_{2^q-1}$ are the Möbius transformations of $f(x_0, \ldots, x_{n-q-1}, 0, \ldots, 0)$, $f(x_0, \ldots, x_{n-q-1}, 1, \ldots, 0)$, $\ldots$, $f(x_0, \ldots, x_{n-q-1}, 1, \ldots, 1)$. Then, the Möbius transformation g of f could be determined with the knowledge of $g_0, g_1, \ldots, g_{2^q-1}$.*

Proof. According to Chapter 9.2 of [11], it is sufficient to calculate the Möbius transformation of f with the Möbius transformations of $f(x_0, x_1, \ldots, x_{n-2}, 0)$ and $f(x_0, x_1, \ldots, x_{n-2}, 1)$. Similarly, with the knowledge of the Möbius transformations of $f(x_0, x_1, \ldots, x_{n-3}, 0, 0)$ and $f(x_0, x_1, \ldots, x_{n-3}, 1, 0)$, the Möbius transformation of $f(x_0, x_1, \ldots, x_{n-2}, 0)$ could be deduced. Recursively, for x_{n-q}, x_{n-q+1}, $\ldots$, x_{n-1}, the Möbius transformation g of f could be determined with the Möbius transformations of $f(x_0, \ldots, x_{n-q-1}, 0, \ldots, 0), f(x_0, \ldots, x_{n-q-1}, 1, \ldots, 0), \cdots, f(x_0, x_1, \ldots, x_{n-q-1}, 1, \ldots, 1)$.

Note that it requires $2^q \times 2^{n-q} = 2^n$ bits memory to store $g_0, g_1, \ldots, g_{2^q-1}$. When n is large, a huge amount of bits memory are required. To reduce the memory complexity, one natural idea is to store only a part values of $g_0, g_1, \ldots, g_{2^q-1}$.

In fact, by storing a part values of $g_0, g_1, \ldots, g_{2^q-1}$, a part of the ANF of f could still be recovered. We formally describe this fact in Proposition 2.

Proposition 2. *Let f, $g_0, g_1, \ldots, g_{2^q-1}$ be defined as Corollary 1. Assume that $\boldsymbol{c} = (c_0, c_1, \ldots, c_{n-q-1})$ is an arbitrary element in $\mathbb{F}_2^{n-q}$. With the knowledge of $g_0(\boldsymbol{c}), g_1(\boldsymbol{c}), \ldots, g_{2^q-1}(\boldsymbol{c})$, we could obtain the coefficients of $\prod_{i=0}^{n-q-1} x_i^{c_i}, x_{n-q} \cdot \prod_{i=0}^{n-q-1} x_i^{c_i}, \ldots, x_{n-q} \cdot x_{n-q+1} \cdots x_{n-1} \cdot \prod_{i=0}^{n-q-1} x_i^{c_i}$. in the ANF of f.*

Proof. Assume that $(b_{n-q}, b_{n-q+1}, \ldots, b_{n-1})$ takes an arbitrary value of $\mathbb{F}_2^q$. Following the proof of Corollary 1, $g(c_0, \ldots, c_{n-q-1}, b_{n-q}, \ldots, b_{n-1})$ could be determined by

$$h_0(c_0, \ldots, c_{n-q-1}, b_{n-q}, \ldots, b_{n-2}) \text{ and } h_1(c_0, \ldots, c_{n-q-1}, b_{n-q}, \ldots, b_{n-2}),$$

where h_0 and h_1 are the Möbius transformations of $f(x_0, x_1, \ldots, x_{n-2}, 0)$ and $f(x_0, x_1, \ldots, x_{n-2}, 1)$ respectively. Furthermore, the value of $h_0(c_0, \ldots, c_{n-q-1}, b_{n-q}, \ldots, b_{n-2})$ can be deduced from

$$h_{0,0}(c_0, \ldots, c_{n-q-1}, b_{n-q}, \ldots, b_{n-3}) \text{ and } h_{0,1}(c_0, \ldots, c_{n-q-1}, b_{n-q}, \ldots, b_{n-3}),$$

where $h_{0,0}$ and $h_{0,1}$ are the Möbius transformations of $f(x_0, \ldots, x_{n-3}, 0, 0)$ and $f(x_0, \ldots, x_{n-3}, 1, 0)$ respectively. Recursively, it is sufficient to calculate $g(c_0, \ldots, c_{n-q-1}, b_{n-q}, \ldots, b_{n-1})$ with the knowledge of $g_0(\boldsymbol{c}), g_1(\boldsymbol{c}), \ldots, g_{2^q-1}(\boldsymbol{c})$. Since $(b_{n-q}, b_{n-q+1}, \ldots, b_{n-1})$ takes an arbitrary value in $\mathbb{F}_2^q$, it indicates that $g(c_0, \ldots, c_{n-q-1}, 0, 0, \ldots, 0), g(c_0, \ldots, c_{n-q-1}, 1, 0, \ldots, 0), \ldots, g(c_0, \ldots, c_{n-q-1}, 1, 1, \ldots, 1)$ could be obtained. Namely, we could recover the coefficients of

$$\prod_{i=0}^{n-q-1} x_i^{c_i}, \quad x_{n-q} \cdot \prod_{i=0}^{n-q-1} x_i^{c_i}, \ldots, \quad x_{n-q} \cdots x_{n-1} \cdot \prod_{i=0}^{n-q-1} x_i^{c_i}$$

in the ANF of f.

Based on Proposition 2, we propose an improved Möbius transformation by breaking the original Möbius transformation into two stages and only store a part of the results during the first stage to reduce the memory complexity. We formally describe the improved Möbius transformation in Algorithm 5. During the first stage of Algorithm 5, for each $0 \leq j \leq 2^q - 1$, the Möbius transformation of g_j is calculated one by one so that the memory could be used repeatedly. Furthermore, for each g_j, only the values g_j under elements whose Hamming Weights are not smaller than ω is stored, where ω is a given bound. Then, during the second stage, by using a way similar to calculate the Möbius transformation of a q-variable polynomial, a part of the ANF of f could be recovered.

The Memory Complexity. The memory needed in Algorithm 5 consists of two parts.

- The size of S is 2^{n-q}, and so it costs 2^{n-q} bits memory.
- For each j, the size of $FS[j]$ is t, and so it requires $2^q \times t$ bits memory totally.

To sum up, it requires $2^q \times t + 2^{n-q}$ bits in Algorithm 5. If $t \lll 2^{n-q}$, then $2^q \times t + 2^{n-q} \lll 2^n$ which indicates that the memory could be decreased to about 2^{n-q} bits from 2^n bits.

Algorithm 5 An Improved Möbius Transformation

Require: A Boolean function f, the parameter q, the bound ω
/* the first stage */
for $(c_0, c_1, \ldots, c_{q-1})$ from $(0, 0, \ldots, 0)$ to $(1, 1, \ldots, 1)$ **do**
  $S \leftarrow$ the truth table of $f(x_0, x_1, \ldots, x_{n-q-1}, c_0, c_1, \ldots, c_{q-1})$;
  Call Algorithm 2 to do Möbius transformation on S;
  $t \leftarrow 0$, $j \leftarrow \sum_{l=0}^{q-1} 2^l c_l$;
  for i from 0 to $2^{n-q} - 1$ **do**
    $tmp \leftarrow (b_0, b_1, \ldots, b_{n-q-1})$, where $i = \sum_{l=0}^{n-q-1} b_l \cdot 2^l$;
    if $wt(tmp) \geq \omega$ **then**
      $FS[j][t] \leftarrow S[i]$;
      $t \leftarrow t + 1$;
    end if
  end for
end for
/* the second stage */
for i from 1 to q **do**
  $Sz \leftarrow 2^i$, $Pos \leftarrow 1$;
  while $Pos < 2^q$ **do**
    for b from 0 to $Sz - 1$ **do**
      for a from 0 to $t - 1$ **do**
        $FS[Pos + Sz + b][a] \leftarrow FS[Pos + Sz + b][a] \oplus FS[Pos + b][a]$;
      end for
    end for
    $Pos \leftarrow Pos + 2 \times Sz$;
  end while
end for

5 Experimental Results

In this section, we first perform experiments to illustrate the effect of Algorithm 4. Then, utilising the starting sets determined with the method described in Section 3.2, we attempt to find linear superpolies for Trivium variants with at least 805 initialization rounds. As a result, we find over 1000 linear superpolies for 805-round Trivium as well as several linear superpolies for 806-round Trivium and 810-round Trivium. Based on the found linear superpolies, we establish a practical attack on 805-round Trivium.

5.1 The Effect of Algorithm 4

To verify the effect of Algorithm 4, we perform extensive experiments on r-round Trivium with $400 \leq r \leq 699$. As mentioned in Section 3.2, for r-round Trivium, we collect thousands of linear superpolies and test which internal state bit each linear superpoly comes from, where r ranges from 400 to 699. Thus, we could determine the preference bit of each Trivium variant experimentally. As a comparison, we predict the preference bit of r-round Trivium by Algorithm 4. The results show that the preference bits are correctly predicted for 226 variants of Trivium out of the total 300 variants. This indicates that the preference bit could be predicted with a success probability 75.3% by Algorithm 4. Furthermore, in the experiment on 400- to 699-round Trivium, the success rate increases as the number of initialization rounds increases. More specifically, for 600- to 699-round Trivium, we could predict the preference bit with a success probability around 84% which is higher than the average value 75.3%, and for 634- to 699-round

Trivium the success probability is 100%. Hence for a higher number of rounds, say 805 or more, the probability probably will not drop. Moreover, based on the preference bit predicted by our method, we practically found a large number of cubes with linear superpolies for the 805-round Trivium. This indicates that our method could predict the preference bit with a good success probability for a higher round.

Remark 2. In the formula of computing $NVK(s_{91} \cdot s_{92})$, we dropped the terms of the form $(\prod_{v \in I} v \cdot k_j)(\prod_{v \in J} v \cdot k_j)$, that is, the key variables in the two VK-terms of s_{91} and s_{92} are the same. Because this number is very small compared with other cases. To verify this, we performed experiments which take the dropped terms into consideration in our formula. For the 300 Trivium variants from 400 to 699 initialization rounds, the result showed that only one of the 300 predicted preference bits was changed.

5.2 A Practical Key-Recovery Attack on 805-Round Trivium

In this subsection, we target 805-round Trivium. We first predict the preference bit of 805-round Trivium. Then, aiming at the preference bit, we determine some proper starting sets of Algorithm 3. For each proper starting set, we construct a potentially good cube with Algorithm 3. Finally, to find linear superpolies, we simultaneously test a large number of subcubes of the potentially good cube with the improved Möbius transformation.

Determine Proper Starting Sets. To determine a proper starting set, we first need to predict the preference bit of 805-round Trivium. With Algorithm 4, we have that the predicted preference bit is $s_{66}^{(805)}$. Since $s_{66}^{(805)} = s_{286}^{(739)} \cdot s_{287}^{(739)} \oplus s_{243}^{(739)} \oplus s_{288}^{(739)} \oplus s_{69}^{(739)}$, we choose cubes of sizes 22 and use the Möbius transformation to search all the subcubes to find proper cubes whose superpolies in $s_{286}^{(739)}$ are linear. Finally, we select some subcubes with linear superpolies to be the starting sets of Algorithm 3. In the following, we take

$$I_1 = \{v_2, v_4, v_6, v_8, v_{10}, v_{11}, v_{15}, v_{17}, v_{19}, v_{21}, v_{23}, v_{25},$$

$$v_{29}, v_{30}, v_{32}, v_{34}, v_{36}, v_{39}, v_{41}, v_{43}, v_{45}, v_{50}\}$$

as an example to illustrate how to determine a proper starting set in details. First, we search all its subcubes to find cubes with linear superpolies in $s_{286}^{(739)}$ and hundreds of such cubes are obtained. When choosing a starting set from these cubes, we prefer to choose cubes with relatively large sizes. Among these cubes, there are two cubes of size 17 and the others have smaller sizes. Among these two cubes, we randomly choose

$$I_2 = \{v_4, v_6, v_{10}, v_{11}, v_{15}, v_{17}, v_{19}, v_{21}, v_{25}, v_{29}, v_{32}, v_{34}, v_{36}, v_{39}, v_{41}, v_{43}, v_{50}\}$$

as a proper starting set. With similar procedure, we determine some other starting sets of Algorithm 3.

Table 3. The chosen cube variables in the last iteration

Chosen cube	$I_5 \cup \{v_{48}\}$	$I_5 \cup \{v_{59}\}$	$I_5 \cup \{v_{58}\}$	$I_5 \cup \{v_{63}\}$
Upper bound of the degree of superpolies	1	1	2	3

Construct Candidate Cubes. There are two main stages of constructing a potentially good cube in Algorithm 3. We take I_2 as an example to make an illustration. In the first stage, Algorithm 3 adds steep IV variables to decrease the degree of the superpoly as quickly as possible. For I_2, the first stage of Algorithm 3 terminates after 17 iterations, since the superpoly p_{I_3} is zero-constant, where

$$I_3 = \{v_4, v_6, v_{10}, v_{11}, v_{15}, v_{17}, v_{19}, v_{21}, v_{25}, v_{29}, v_{32}, v_{34}, v_{36}, v_{39}, v_{41}, v_{43}, v_{50},$$
$$v_2, v_{69}, v_{79}, v_8, v_{27}, v_0, v_1, v_{28}, v_{71}, v_{13}, v_{45}, v_{23}, v_{26}, v_{38}, v_{76}, v_{47}, v_{56}\}.$$

Then, the second phase is started with

$$I_4 = \{v_4, v_6, v_{10}, v_{11}, v_{15}, v_{17}, v_{19}, v_{21}, v_{25}, v_{29}, v_{32}, v_{34}, v_{36}, v_{39}, v_{41}, v_{43}, v_{50},$$
$$v_2, v_{69}, v_{79}, v_8, v_{27}, v_0, v_1, v_{28}, v_{71}, v_{13}, v_{45}, v_{23}, v_{26}, v_{38}, v_{76}, v_{47}, v_{52}\},$$

since the upper bound of the degree of p_{I_4} attains minimum expect 0 among all the cubes obtained after 17 iterations. In this stage, our aim is to decrease the degree of the superpoly slowly to obtain cube with linear superpolies instead of zero-sum distinguishers. After three iterations, we obtain two cubes such that the degree of their superpolies are upper bounded by 1. Besides, we also obtain several cubes such that the degree of their superpolies are not larger than 3. By jointing 4 cubes, we constructed a potentially good cube of size 40. Table 3 shows the cubes and the upper bounds of the degrees of their superpolies, where

$$I_5 = \{v_4, v_6, v_{10}, v_{11}, v_{15}, v_{17}, v_{19}, v_{21}, v_{25}, v_{29}, v_{32}, v_{34}, v_{36}, v_{39}, v_{41}, v_{43}, v_{50}, v_2,$$
$$v_{69}, v_{79}, v_8, v_{27}, v_0, v_1, v_{28}, v_{71}, v_{13}, v_{45}, v_{23}, v_{26}, v_{38}, v_{76}, v_{47}, v_{52}, v_{57}, v_{42}\}.$$

Finally, the potentially good cube I_6 constructed from I_2 is as follows,

$$I_6 = I_5 \cup \{v_{48}, v_{58}, v_{59}, v_{63}\}.$$

Linear Superpolies for 805-Round Trivium. After obtaining a potentially good cube, we use the improved Möbius transformation to search its subcubes which miss few cube variables. For instance, in the case of I_6, we set the parameter $q = 7$ and $\omega = 26$ in the improved Möbius transformation, and we find 201 subcubes with linear superpolies eventually. Among these 201 linear superpolies, there are 22 linear superpolies which are linearly independent. Together with some other candidate cubes, we find more than 1000 cubes with linear superpolies in the output of 805-round Trivium. Among these cubes, we could pick up 38 cubes whose superpolies are linearly independent, see Table 4.

Table 4. Linear superpolies for 805-round Trivium

Cube indices	Superpoly
0,1,2,4,6,8,11,13,15,17,19,21,23,26,27,28,29,32,34, 36,38,39,41,42,45,47,48,50,52,53,57,69,71,75,76,79	$1 \oplus k_2 \oplus k_{65}$
0,1,2,4,6,8,10,11,12,13,15,16,19,21,23,25,26,27,29, 31,34,36,38,39,40,43,45,47,49,62,64,70,74,77,79	$1 \oplus k_3$
0,1,2,4,6,8,10,11,13,15,17,19,21,23,26,27,29,31, 34,36,38,39,40,41,43,45,47,49,58,62,64,77,79	$k_4 \oplus k_{19} \oplus k_{34}$
0,1,2,4,6,8,10,13,15,17,19,21,23,25,26,27,28,29,32,34, 36,38,39,41,42,43,47,48,50,52,57,59,69,71,75,76,79	k_{14}
0,1,2,4,6,8,10,11,13,15,17,19,21,23,25,26,27,29,32,34, 36,38,39,41,42,43,47,48,50,52,53,57,59,69,71,76,79	k_{15}
0,1,2,4,6,8,10,13,15,17,19,21,23,25,26,27,28,29,32, 34,36,38,39,41,42,43,47,48,50,52,59,69,71,75,76,79	$1 \oplus k_{16}$
0,1,2,4,6,8,10,11,13,15,17,19,21,23,25,26,27,28,29,32, 34,36,38,39,41,42,43,45,47,48,50,53,57,69,71,75,76,79	$1 \oplus k_{17}$
0,1,2,4,6,8,10,11,12,13,15,16,19,21,23,25,27,28, 29,34,36,38,40,41,43,45,47,49,50,64,70,74,77,79	k_{18}
0,1,2,4,6,8,10,11,12,13,15,16,19,23,25,27,28,31,34, 36,38,39,40,41,43,45,47,49,50,58,62,64,74,77,79	$1 \oplus k_{19} \oplus k_{34} \oplus k_{51}$
0,2,4,6,8,10,12,13,15,17,19,21,23,25,26,27,28,29,31, 34,38,39,40,41,43,45,47,49,50,58,62,64,70,74,77,79	k_{21}
1,2,4,6,8,10,11,12,13,15,17,19,21,23,25,26,27,28,29, 31,34,36,38,39,40,41,43,47,49,50,58,62,70,74,77,79	$1 \oplus k_{29}$
0,1,2,4,6,8,10,11,13,15,17,19,21,23,25,26,27,29,32,34, 36,38,39,42,43,45,47,48,50,52,53,57,59,69,71,75,76,79	$k_{31} \oplus k_{46} \oplus k_{56}$
0,1,2,4,6,8,10,13,15,17,19,21,23,25,26,28,29,32,34, 36,38,39,41,42,45,47,48,50,52,57,59,69,71,75,76,79	$k_{17} \oplus k_{32}$
0,1,2,4,6,8,10,11,13,15,17,19,21,23,25,26,27,29,32,34, 36,38,39,42,43,45,47,48,50,52,53,57,59,69,71,76,79	$1 \oplus k_{33}$
0,1,2,4,6,8,10,11,13,15,17,19,21,23,25,26,27,29,32, 34,36,39,41,42,43,45,47,48,50,52,57,59,69,71,76,79	k_{34}
0,1,2,4,6,8,10,11,13,15,17,19,21,23,25,26,27,29,32, 34,36,38,39,41,42,43,45,47,50,52,53,57,69,71,75,79	k_{36}
0,1,2,4,6,8,10,12,13,15,17,19,21,23,25,26,27,28, 29,31,34,36,39,40,41,43,47,49,50,62,64,70,77,79	k_{40}
0,1,2,4,6,8,10,11,13,15,17,19,21,23,26,27,28,31, 34,36,38,40,41,43,45,47,49,50,58,62,64,70,77,79	k_{42}
0,1,2,4,6,8,10,11,13,15,16,19,21,23,26,27,28,29,31, 34,36,38,39,41,43,45,47,49,50,58,62,64,74,77,79	k_{43}
0,1,2,4,6,8,10,11,13,15,17,19,21,23,25,27,29,32, 34,36,38,42,45,47,48,50,53,57,59,69,71,75,76,79	k_{44}
0,1,2,4,6,8,10,11,13,15,17,19,21,23,25,27,28,29, 32,34,36,38,41,42,43,45,47,50,53,59,69,71,76,79	$1 \oplus k_{45}$
0,1,2,4,6,8,10,11,13,15,17,19,21,23,25,27,28,29,32, 34,36,38,39,42,43,45,48,50,52,57,59,69,71,75,76,79	$k_{46} \oplus k_{56}$
0,1,2,4,6,8,10,11,13,15,17,19,21,23,25,27,29,32,34, 36,38,39,41,42,43,45,47,48,50,57,59,69,71,76,79	$1 \oplus k_{47}$
0,1,2,4,6,8,11,13,15,17,19,21,23,26,27,28,29, 34,36,38,41,43,45,47,49,50,62,64,70,74,77,79	k_{49}
0,1,2,4,6,8,11,13,15,17,19,21,23,25,27,28,29,32,34, 36,38,39,41,42,43,45,47,52,53,57,69,71,75,76,79	k_{51}
0,2,4,6,8,10,11,12,13,15,16,19,21,23,25,26,27,28,29, 31,34,36,38,39,41,43,47,49,58,62,64,70,74,77,79	k_{53}
0,1,4,6,8,10,11,13,15,17,19,21,23,25,26,28,29,32,34,36, 38,39,41,42,43,45,47,48,50,52,53,57,59,69,71,75,76,79	k_{54}
0,1,2,4,6,8,10,11,13,15,17,19,21,23,25,27,29,32,34, 36,38,39,42,43,45,47,48,50,53,57,59,69,71,75,79	k_{56}
0,1,2,4,6,8,10,11,12,13,15,17,19,21,23,25,26,27,28, 29,31,34,36,38,39,40,41,45,47,49,58,62,64,70,79	$k_{57} \oplus k_{59}$
0,1,2,4,6,8,10,13,15,17,19,21,23,25,27,28,29,32, 34,36,38,39,42,43,45,47,48,53,57,59,69,71,75,79	k_{58}
0,1,2,4,6,8,10,11,13,15,17,19,21,23,25,26,27,28,29, 32,34,36,38,39,41,42,43,45,47,50,53,57,59,69,76,79	$1 \oplus k_{47} \oplus k_{59}$
0,1,2,4,6,8,10,11,13,15,17,19,21,23,25,27,28,29,32, 34,36,38,39,41,42,43,45,47,48,50,59,69,71,75,76,79	k_{60}
0,1,2,4,6,8,10,11,13,15,17,19,21,23,25,27,28,29,32, 34,36,38,39,41,42,43,45,47,48,50,52,59,71,76,79	k_{61}
0,2,4,6,8,10,11,12,13,15,16,19,21,23,25,26,27,28,31, 34,36,38,39,40,41,43,45,47,49,50,58,62,64,77,79	k_{62}
0,1,2,4,6,8,10,11,13,15,16,19,21,23,25,27,28,29,31, 34,36,39,41,43,45,47,49,62,64,70,74,77,79	k_{63}
0,1,4,6,8,10,11,12,13,15,17,19,21,23,25,26,27,28,29, 34,36,38,39,41,43,45,47,49,58,62,64,70,74,77,79	k_{64}
0,2,4,6,8,10,11,13,15,17,19,21,23,25,26,27,28,29,32, 34,36,39,41,43,45,47,48,50,52,57,59,69,71,76,79	k_{65}
0,1,2,4,6,8,11,12,13,15,17,19,21,23,25,27,28,29,31, 34,36,39,40,41,43,45,47,49,50,62,64,70,74,77,79	k_{68}

Linear Superpolies for 806-Round Trivium. For the cubes found for 805-round Trivium, we slide some of them, i.e. decrease the index of each cube

variables by 1, to find cubes with linear superpolies for 806-round Trivium. Finally, we find several cubes whose superpolies in the output bit of 806- round Trivium, see Table 5.

Table 5. Linear superpolies for 806-round Trivium

Cube indices	Superpoly
0,1,3,5,7,9,10,11,12,14,15,18,20,22,24,27,28,30,33,35,37,39,40,42,44,46,48,49,57,61,63,73,76,78	$k_{14} \oplus k_{44}$
0,1,3,5,7,9,10,11,12,14,15,18,20,22,24,26,28,30,33,35,37,39,40,42,44,46,48,49,57,61,63,73,76,78	k_{15}
0,1,3,5,7,9,10,11,12,14,15,18,20,22,24,26,28,30,33,35,37,39,40,42,44,46,48,49,57,61,63,76,78	$1 \oplus k_{17}$
0,1,3,5,7,9,10,11,12,14,16,18,20,22,24,25,26,27,28,30,33,35,37,38,39,40,42,46,48,49,57,61,69,73,76,78	$1 \oplus k_{28}$
0,1,3,5,7,9,10,11,12,14,15,16,18,20,22,24,25,26,27,28,30,33,35,37,38,39,40,42,46,48,49,57,61,63,76,78	k_{32}
0,1,3,5,7,9,10,11,12,14,15,18,20,22,24,26,27,28,33,35,37,39,40,42,44,46,48,49,57,61,63,73,76,78	k_{33}
0,3,5,7,9,11,14,15,18,20,22,24,25,26,27,30,33,35,37,39,40,42,44,46,48,49,57,61,63,69,73,76,78	k_{41}
0,1,3,5,7,9,10,11,12,14,15,18,20,22,24,26,27,28,30,33,35,37,40,42,44,46,48,49,57,61,63,73,76,78	k_{42}
1,3,5,7,9,10,11,12,14,16,18,20,22,24,25,26,27,28,30,33,35,37,38,39,40,42,46,48,49,57,61,63,73,76,78	k_{44}
0,1,3,5,7,9,10,12,14,16,18,20,22,24,26,27,28,30,33,35,37,38,40,42,44,46,48,57,61,63,73,76,78	k_{46}
0,1,3,5,7,9,10,11,12,14,16,18,20,22,24,26,27,28,33,35,37,39,40,42,44,46,48,49,57,61,63,76,78	k_{52}
0,1,3,5,9,10,11,12,14,16,18,20,22,24,26,27,28,30,33,35,37,38,40,42,44,46,48,49,57,61,63,69,73,76,78	k_{55}
0,1,3,5,7,9,10,11,12,14,16,18,20,22,24,26,27,28,30,33,35,37,38,42,44,46,48,49,57,61,63,69,76,78	k_{58}
0,1,3,5,7,9,10,11,12,14,15,18,20,22,24,25,26,27,28,30,33,35,37,38,39,40,42,46,48,57,61,63,69,76,78	k_{59}
0,1,3,5,7,9,11,12,14,15,16,18,20,22,24,25,27,28,30,33,35,37,38,40,42,44,46,48,49,57,61,63,69,73,76,78	k_{63}
0,3,5,7,9,10,11,12,14,15,18,20,22,24,25,26,27,28,33,35,37,38,39,40,42,44,46,48,57,61,63,69,73,76,78	k_{65}

A Practical Key-Recovery Attack on 805-Round Trivium. Based on the linear superpolies of 805- and 806-round Trivium, we could recover 42 key bits for 805-round Trivium. The sizes of the chosen cubes are from 32 to 38, and 42 key bits could be recovered with $2^{41.25}$ requests. By adding a brute-force attack, the remaining 38 key bits could be recovered within 2^{38} requests. Consequently, to recover the whole key for 805-round Trivium, the on-line complexity is not larger than $2^{41.40}$ requests. Under a PC with a GTX-1080 GPU, we could recover 42 key bits in several hours. For remaining key bits, they could be recovered in less than 2^{38} requests which is much easier. Consequently, our attack on 805-round Trivium is practical.

5.3 Experimental Results on 810-Round Trivium

We do the similar experiments on 810-round Trivium. In this case, the preference bit is $s_{66}^{(810)}$ as well. We perform experiments on the starting cube set

$$I_7 = \{v_2, v_6, v_8, v_{10}, v_{11}, v_{15}, v_{19}, v_{21}, v_{25}, v_{29}, v_{30}, v_{32}, v_{34}, v_{36}, v_{39}, v_{41}, v_{43}, v_{45}, v_{50}\}.$$

With Algorithm 3, we finally get a cube I_8 given by

$$\begin{aligned} I_8 = \{&v_2, v_6, v_8, v_{10}, v_{11}, v_{15}, v_{19}, v_{21}, v_{25}, v_{29}, v_{30}, v_{32},\\ &v_{34}, v_{36}, v_{39}, v_{41}, v_{43}, v_{45}, v_{50}, v_0, v_{75}, v_{12}, v_{22},\\ &v_{16}, v_{27}, v_{23}, v_{72}, v_4, v_{14}, v_{20}, v_{52}, v_{55}, v_{60}, v_{37},\\ &v_{79}, v_{62}, v_{64}, v_{47}, v_{54}, v_{69}, v_{51}, v_{71}, v_{18}, v_{53}\}. \end{aligned}$$

The size of I_8 is 44. Because it is too time consuming to perform linearity tests, we try to remove some cube variables from I_8 to obtain a smaller cube with low-degree superpolies. Finally, we obtain the cube I_9 of size 43, where

$$\begin{aligned} I_9 = \{&v_2, v_6, v_8, v_{10}, v_{11}, v_{15}, v_{19}, v_{21}, v_{25}, v_{29}, v_{30}, v_{32},\\ &v_{34}, v_{36}, v_{39}, v_{41}, v_{43}, v_{45}, v_{50}, v_0, v_{75}, v_{12}, v_{22},\\ &v_{16}, v_{27}, v_{23}, v_{72}, v_4, v_{14}, v_{20}, v_{52}, v_{55}, v_{60},\\ &v_{37}, v_{79}, v_{62}, v_{64}, v_{47}, v_{54}, v_{69}, v_{71}, v_{18}, v_{53}\}. \end{aligned}$$

and the degree of the superpoly of I_9 is upper bounded by 2. By using a computer with four NVIDIA V100 GPUs, we search a part of subcubes which only misses few cube variables in I_9. With the original Möbius transformation, to search subcubes of a 43-dimensional cubes, it needs 2^{43} bits memory. Benefited from the improved Möbius transformation, we could perform linearity tests on $2^{32.28}$ subcubes of I_9 with several GBs memory which is much less than the memory (1024 GB) required by the original Möbius transformation. Finally, we find 2 different cubes with linear superpolies, which are listed in Table 6.

Table 6. Linear superpolies for 810-round Trivium

Cube indices	Superpoly
0,2,4,6,8,10,11,12,14,15,16,18,19,20,21,22,23,25,27,29,30,32, 34, 36,37,39,41,43,45,47,50,53,54,55,60,62,64,69,71,72,75,79	k_{62}
0,2,4,6,8,10,11,12,14,15,16,18,19,20,21,22,23,25,27,29,30,32, 34, 36,37,39,41,43,45,47,50,51,53,54,60,62,64,69,71,72,75,79	k_{62}

Remark 3. We put our codes and all the found superpolies on https://github.com/YT92/Practical-Cube-Attacks.

6 Conclusion

In this paper, we focus on practical full key-recovery attacks on Trivium. We design a new framework for finding linear superpolies in cube attacks by presenting a new algorithm to construct cubes which potentially yield linear superpolies. With this new framework, we find sufficiently many linear superpolies and establish a practical full key-recovery attack on 805-round Trivium. To show the effectiveness of our algorithm for constructing cubes, we also tried 810-round Trivium. As a result, by constructing one 43-dimensional cube, we find two subcubes of size 42 with linear superpolies for 810-round Trivium. So far the success rate of our algorithm for finding linear superpolies is 100%. The 805-round Trivium is just chosen for an example. We believe that the new algorithm could

also be applicable to Trivium up to 810 rounds with a bit more time since cube sizes increases a little. Since we use linearity test and Moebius transformation to recover superpolies, large cube sizes could not be explored. Recently, Hao et al. at EUROCRYPT 2020 proposed a new MILP modeling method for the three-subset division property which could be used to recover the exact superpoly for a given cube. Combing our new algorithm for selecting cubes with the three-subset division property to recover low-degree superpolies for large cubes will be one subject of our future work.

References

1. 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.
2. Christophe De Cannière and Bart Preneel. Trivium. In *New Stream Cipher Designs - The eSTREAM Finalists*, pages 244–266. 2008.
3. Itai Dinur, Tim Güneysu, Christof Paar, Adi Shamir, and Ralf Zimmermann. An experimentally verified attack on full grain-128 using dedicated reconfigurable hardware. In Dong Hoon Lee and Xiaoyun Wang, editors, *Advances in Cryptology - ASIACRYPT 2011 - 17th International Conference on the Theory and Application of Cryptology and Information Security, Seoul, South Korea, December 4-8, 2011. Proceedings*, volume 7073 of *Lecture Notes in Computer Science*, pages 327–343. Springer, 2011.
4. 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.
5. 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.
6. 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.
7. Yonglin Hao, Gregor Leander, Willi Meier, Yosuke Todo, and Qingju Wang. Modeling for three-subset division property without unknown subset. *IACR Cryptol. ePrint Arch.*, 2020:441, 2020.
8. Yonglin Hao, Gregor Leander, Willi Meier, Yosuke Todo, and Qingju Wang. Modeling for three-subset division property without unknown subset - improved cube attacks against trivium and grain-128aead. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology - EUROCRYPT 2020 - 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10-14, 2020, Proceedings, Part I*, volume 12105 of *Lecture Notes in Computer Science*, pages 466–495. Springer, 2020.
9. 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 - 26th International Conference on the Theory and Application of Cryptology and Information Security, Daejeon, South Korea, December 7-11, 2020, Proceedings, Part I, volume 12491 of *Lecture Notes in Computer Science*, pages 446–476. Springer, 2020.

10. 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.
11. Antoine Joux. *Algorithmic Cryptanalysis (Chapman & Hall/CRC Cryptography and Network Security Series) 1st Edition.* Chapman and Hall/CRC, 2009.
12. Abhishek Kesarwani, Dibyendu Roy, Santanu Sarkar, and Willi Meier. New cube distinguishers on nfsr-based stream ciphers. *Des. Codes Cryptogr.*, 88(1):173–199, 2020.
13. Zheng Li, Wenquan Bi, Xiaoyang Dong, and Xiaoyun Wang. Improved conditional cube attacks on keccak keyed modes with MILP method. In *Advances in Cryptology - ASIACRYPT 2017 - 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part I*, pages 99–127, 2017.
14. 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.
15. 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.
16. Piotr Mroczkowski and Janusz Szmidt. Corrigendum to: The cube attack on stream cipher Trivium and quadraticity tests. *IACR Cryptology ePrint Archive*, 2011:32, 2011.
17. Majid Rahimi, Mostafa Barmshory, Mohammad Hadi Mansouri, and Mohammad Reza Aref. Dynamic cube attack on grain-v1. *IET Inf. Secur.*, 10(4):165–172, 2016.
18. Santanu Sarkar, Subhamoy Maitra, and Anubhab Baksi. Observing biases in the state: case studies with Trivium and Trivia-SC. *Des. Codes Cryptography*, 82(1-2):351–375, 2017.
19. Paul Stankovski. Greedy distinguishers and nonrandomness detectors. In Guang Gong and Kishan Chand Gupta, editors, *Progress in Cryptology - INDOCRYPT 2010 - 11th International Conference on Cryptology in India, Hyderabad, India, December 12-15, 2010. Proceedings*, volume 6498 of *Lecture Notes in Computer Science*, pages 210–226. Springer, 2010.
20. Ling Sun, Wei Wang, and Meiqin Wang. Milp-aided bit-based division property for primitives with non-bit-permutation linear layers. Cryptology ePrint Archive, Report 2016/811, 2016.
21. 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.

22. Yosuke Todo, Takanori Isobe, Yonglin Hao, and Willi Meier. Cube attacks on non-blackbox polynomials based on division property. *IEEE Trans. Computers*, 67(12):1720–1736, 2018.
23. Yosuke Todo and Masakatu Morii. Bit-based division property and application to simon family. In *Fast Software Encryption - 23rd International Conference, FSE 2016, Bochum, Germany, March 20-23, 2016, Revised Selected Papers*, pages 357–377, 2016.
24. 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 *Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part I*, pages 275–305, 2018.
25. SenPeng Wang, Bin Hu, Jie Guan, Kai Zhang, and Tairong Shi. Milp-aided method of searching division property using three subsets and applications. In *Advances in Cryptology - ASIACRYPT 2019 - 25th International Conference on the Theory and Application of Cryptology and Information Security, Kobe, Japan, December 8-12, 2019, Proceedings, Part III*, pages 398–427, 2019.
26. 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 *Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part I*, pages 648–678, 2016.
27. 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.
28. Chen-Dong Ye and Tian Tian. Revisit division property based cube attacks: Key-recovery or distinguishing attacks? *IACR Trans. Symmetric Cryptol.*, 2019(3):81–102, 2019.
29. Chen-Dong Ye and Tian Tian. Algebraic method to recover superpolies in cube attacks. *IET Information Security*, 14(4):430–441, 2020.