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

Chen-Dong Ye[1] and Tian Tian[1]

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 improvements have been proposed and lots of key-recovery attacks based on cube attacks have been established. However, among these key-recovery attacks, few attacks can recover the 80-bit full key practically. In particular, the previous best practical key-recovery attack was on 784-round Trivium proposed by Fouque and Vannet at FSE 2013 with on-line complexity about 2^{39}. To mount a practical key-recovery attack against Trivium on a PC, a sufficient number of low-degree superpolies should be recovered, which is around 40. This is a difficult task 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 at linear superpolies in cube attacks. It is shown by our experiments that the new algorithm is very effective. In our experiments, the success probability is 100% for finding linear superpolies using the constructed cubes. As a result, we mount a practical key-recovery attack on 805-round Trivium, which increases the number of attacked initialisation rounds by 21. We obtain over 1000 cubes with linear superpolies for 805-round Trivium, where 42 linearly independent ones could be selected. With these superpolies, for 805-round Trivium, the 80-bit key could be recovered within on-line complexity $2^{41.40}$, which could be carried out on a single PC equipped with a GTX-1080 GPU in several hours. Furthermore, the new algorithm is applied to 810-round Trivium, a cube of size 43 is constructed and two subcubes of size 42 with linear superpolies for 810-round Trivium are found.

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

1 Introduction

Trivium [2] is a bit oriented synchronous stream cipher designed by 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 many attention.

* Supported by organization x.

Cube attacks, proposed by Dinur and Shamir in [4], are one of the most powerful cryptanalytic techniques against Trivium. There are two main phases in cube attacks. In the first phase, called the preprocessing phase, one needs to find cubes whose superpolies are low-degree polynomials on key variables. In the second phase, called the on-line phase, by querying the encryption oracle, one could calculate the value of the superpoly under the real key for each chosen cube and so obtain an equation on key variables. Then, by solving the obtained system of equations, one could recover the values of a part of key bits or even the whole key. Finally, by exhausting all the possible values of the remaining key bits, one could recover the entire key. Since proposed, many improvements have been proposed on cube attacks such as cube testers [1], dynamic cube attacks [5, 3, 18], conditional cube attacks [10, 13], division property based cube attacks [22, 23, 25, 29, 26, 8, 9] and correlation cube attacks [16]. Due to these improvements, cube attacks have become more and more powerful.

However, note that few cube attacks against the round-reduced Trivium are practical. In particular, some attacks could only recover one or two key bits and some attacks have very marginal online complexities. For example, in [8, 7], cubes of sizes 78 were used to recovery one key bit for 840-, 841- and 842-round Trivium, respectively. In this case, it needs 2^{78} requests to recover one key bit by cube summation and 2^{79} requests to exhaustively search the remaining 79 key bits. Thus, the total complexity is $2^{78} + 2^{79}$ using only one 78-dimensional cube. It can be seen that $2^{78} + 2^{79}$ is very close to the brute-force attack complexity.

Currently, for Trivium, the number of initialisation rounds that could be reached by cube attacks with a practical complexity is 784. One main reason is that finding cubes which could be used to mount key-recovery attacks is a tough task in cube attacks.

So far, there have been few literature focusing on finding cubes which could be used to mount key-recovery attacks. 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 with linear superpolies. If it fails, then the process can be restarted with another I. With this method, for 767-round Trivium, 35 linear superpolies were found and so the key could be recovered with about 2^{45} requests. In [6], the authors try to find cube with linear superpoly from another aspect. Their main idea is constructing a candidate cube by jointing two subcubes satisfying some specific properties. Then, the Moebius transformation is used to search all the subcubes of the candidate cube to find linear superpolies. As a result, for 784-round Trivium, they found 42 linear superpolies such that the key could be recovered in less than 2^{39} requests. Furthermore, for 799-round Trivium, they found 12 linear superpolies and 6 quadratic superpolies and so the key could be recovered in about 2^{62} requests.

If we extend to the scope of finding cubes whose superpolies could be used to establish distinguishing attacks, then there are some more literature. In [20], the authors first proposed the GreedyBitSet algorithm to find cubes which could lead to distinguishers and nonrandomness detectors. As a result, they obtained good distinguishers of several ciphers including Trivium, Grain-128, and Grain-v1. Later, in [19], based on the work in [20], the authors studied the state biases as well as key-stream biases. As a result, they obtained 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 [19]. 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. Besides, in [15], the author proposed a heuristic algorithm which formed cubes by uniting small subcubes iteratively. With this method, they proposed a highly biased distinguisher on 839-round Trivium.

1.1 Our Contributions

This paper devotes to mounting practical cube attacks against Trivium variants with at least 805 initialisation rounds. To achieve this goal, it needs to find enough cubes whose superpolies could be used to set up equations on key variables in the on-line phase. This is actually a tough challenge. To complete this challenge, inspired by the GreedyBitSet algorithm and division property based cube attacks, we develop a new framework of finding linear superpolies, where a candidate cube is first constructed from a carefully selected small cube set and then a large mount of subcubes of the candidate cube is tested simultaneously with a reasonable memory complexity. This enables us to find sufficiently many cubes with linear superpolies and attack more rounds practically. In the following, we formulate our contributions into three aspects.

A Heuristic Algorithm to Construct Candidate Cubes. In previous works, the GreedyBitSet algorithm is usually applied to finding zero-sum distinguishers or non-randomness detectors. By combining the GreedyBitSet algorithm with division property, we propose a new algorithm to construct cubes which are potential to have linear superpolies. Our 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, 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 func-

tion of Trivium is the XOR of six internal state bits. It is very possible that a linear superpoly is contributed by a single internal state bit. Hence, to determine proper starting sets of the above new algorithm, we propose the concept of the preference bit together with an iterative algorithm to the preference bit. By targeting at the preference bit, it is more likely to find linear superpolies in the output of Trivium. However, the ANFs of the internal state bits become very complex as the number of initialisation rounds increases. When r is large, it is hard to determine the preference bit of r-round Trivium by calculating the ANFs of the internal state bits directly. To overcome this difficulty, based on the structure analysis of Trivium, we propose an iterative algorithm to predict the preference bit of r-round Trivium. The experimental results show that our method could predict the preference bit with a success probability 75.3%. With the knowledge of the preference bit, proper starting sets of the new algorithm could be determined easily according to the update function of Trivium.

The Improved Moebius Transformation. In cube attacks, the Moebius transformation is a powerful tool which could be used to test all the subcubes of a large cube simultaneously. However, it requires a high memory complexity. To reduce the memory complexity, we break the original Moebius transformation into a two-stage version. Let $f(x_0, x_1, \ldots, x_{n-1})$ be a Boolean function on $x_0, x_1, \ldots, x_{n-1}$. In the first stage, the Moebius transformations of $f(x_0, x_1, \ldots, x_{n-q-1}, 0, 0, \ldots, 0)$, $f(x_0, x_1, \ldots, x_{n-q-1}, 1, 0, \ldots, 0)$ and $f(x_0, x_1, \ldots, x_{n-q-1}, 1, 1, \ldots, 1)$ are calculated and only a part of each Moebius 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 Moebius transformation of ai 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 coulid 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 Moebius transformation requires 2^{43} bits (1024 GBs) memory.

As an illustration, we apply our methods to 805-round Trivium and 810-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 which are linearly independent. Besides, some cubes of 805-round Trivium could be slid to obtain 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 is practical and could be completed by a PC with a GTX-1080 GPU in several hours. Furthermore, for 810-round Trivium, by only testing one 43-dimensional cube, we find two 42-dimensional cubes with linear superpolies. It worth noting that the 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

initialisation rounds. As a comparison, we summarise the cube attacks based key-recovery attacks against the round-reduced Trivium in Table 1.

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	$2^{29.14}$	[17]
	767	28-31	35	2^{45}	$2^{45.00}$	[4]
	784	30-33	42	2^{38}	2^{39}	[6]
	805	33-38	42	2^{38}	$2^{41.40}$	Sect. 5.2
Not practical	799	32-37	18	2^{62}	$2^{62.00}$	[6]
	802	34-37	8	2^{72}	$2^{72.00}$	[28]
	805	28	7	2^{73}	$2^{73.00}$	[16]
	806	34-37	16	2^{64}	2^{64}	Sect. 5.2
	835	35	5	2^{75}	$2^{75.00}$	[16]
	832	72	1	2^{79}	$2^{79.01}$	[26, 22, 23]
	832	72	> 1	2^{79}	$< 2^{79.01}$	[30]
	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]

1.2 Organisations

The rest of this paper is organised 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 Moebius 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 practical cube attacks on 805- and 810-round Trivium. Finally, Section 6 concludes this paper.

2 Preliminaries

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

2.1 Boolean Functions and Algebraic Degree

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

Boolean function f can be represented by a polynomial on n variables over $\mathbb{F}_2$,

$$f(x_0, x_1, \ldots, x_{n-1}) = \bigoplus_{c=(c_0,c_1,\ldots,c_{n-1})\in\mathbb{F}_2^n} a_c \prod_{i=0}^{n-1} x_i^{c_i},$$

which is called the algebraic normal form (ANF) of f, where $a_c \in \mathbb{F}_2$. In this paper, $u = a_c \prod_{i=0}^{n-1} x_i^{c_i}$ $(a_c \neq 0)$ is called a term of f. The algebraic degree of a Boolean function is denoted by $\deg(f)$ and defined as

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

where $wt(c)$ is the Hamming Weight of c, i.e., $wt(c) = \sum_{i=0}^{n-1} c_i$.

2.2 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 sate 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 initialisation 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

1: $(s_1, s_2, \ldots, s_{93}) \leftarrow (x_1, x_2, \ldots, x_{80}, 0, \ldots, 0)$;
2: $(s_{94}, s_{95}, \ldots, s_{177}) \leftarrow (v_1, v_2, \ldots, v_{80}, 0, \ldots, 0)$;
3: $(s_{178}, s_{179}, \ldots, s_{288}) \leftarrow (0, \ldots, 0, 1, 1, 1)$;
4: **for** i from 1 to N **do**
5: $\quad t_1 \leftarrow s_{66} \oplus s_{93} \oplus s_{91}s_{92} \oplus s_{171}$;
6: $\quad t_2 \leftarrow s_{162} \oplus s_{177} \oplus s_{175}s_{176} \oplus s_{264}$;
7: $\quad t_3 \leftarrow s_{243} \oplus s_{288} \oplus s_{286}s_{287} \oplus s_{69}$;
8: $\quad$ **if** $i > 1152$ **then**
9: $\quad\quad z_{i-1152} \leftarrow s_{66} \oplus s_{93} \oplus s_{162} \oplus s_{177} \oplus s_{243} \oplus s_{288}$;
10: $\quad$ **end if**
11: $\quad (s_1, s_2, \ldots, s_{93}) \leftarrow (t_3, s_1, \ldots, s_{92})$;
12: $\quad (s_{94}, s_{95}, \ldots, s_{177}) \leftarrow (t_1, s_{94}, \ldots, s_{176})$;
13: $\quad (s_{178}, s_{179}, \ldots, s_{288}) \leftarrow (t_2, s_{178}, \ldots, s_{287})$;
14: **end for**

2.3 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 tweakable 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

$$f(\boldsymbol{k}, \boldsymbol{v}) = t_I \cdot p_I(\boldsymbol{k}, \boldsymbol{v}) \oplus q_I(\boldsymbol{k}, \boldsymbol{v}), \tag{1}$$

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,

$$\bigoplus_{(v_{i_1}, v_{i_2}, \ldots, v_{i_d}) \in \mathbb{F}_2^d} f(\boldsymbol{k}, \boldsymbol{v}) = p_I(\boldsymbol{k}, \boldsymbol{v}). \tag{2}$$

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. In this paper, for the sake of convenience, we also call p_I the superpoly of I in f.

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

- **Off-line Phase.** In the off-line phase, the attacker should to find cubes whose superpolies in the output bit is low-degree polynomials.
- **On-line Phase.** In the on-line phase, for each cube obtained in the off-line phase, the attacker enquires 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 on 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.

2.4 The Bit-Based Division Property

The conventional bit-based division property was introduced in [24]. The authors of [24] also introduced the bit-based division property using three subsets. In this paper, we focus on the conventional bit-based division property. The definition of the conventional bit-based division property is as follows.

Definition 1 (Bit-Based Division Property). *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 [24]. To avoid such a high memory complexity, in [27], 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 [27]). *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 must exist a vector* $\boldsymbol{\alpha}_i^* \in \mathbb{K}_i$ *such that* $\boldsymbol{\alpha}_i^*$ *can propagate to* $\boldsymbol{k}_{i+1}^*$ *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 [27], the authors described the propagation rules for AND, COPY and XOR with MILP models, see [27] for the detailed definition of AND, COPY and XOR. Therefore, they could build an MILP model to cover all the possible division trails generated during the propagation. Besides, in [22, 21], the authors made some simplifications to those MILP models in [27]. In particular, in [22], the division property based cube attacks were proposed for the first time and were applied to attacks Trivium, Grain-128 and Acorn successfully.

Later, to describe the propagation of division property more precisely, the authors of [25] proposed the flag technique. Together with some other techniques, the authors of [25] improved the division property based cube attacks. In particular, for a given set of cube variables, the method proposed in [25] could return the upper bound of the degree of the superpoly p_I, see [25] for details. In this paper, this method is used to find cubes with linear superpolies.

2.5 The Moebius Transformation

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

Let f be a polynomial in $\mathbb{F}_2[x_1, x_2, \ldots, x_n]$, whose algebraic normal form is given by

$$f(x_0, x_1, \ldots, x_{n-1}) = \bigoplus_{\boldsymbol{c}=(c_0,c_1,\ldots,c_{n-1}) \in \mathbb{F}_2^n} g(c_0, c_1, \ldots, c_{n-1}) \prod_{i=0}^{n-1} x_i^{c_i},$$

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

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

Algorithm 2 The Moebius transformation algorithm

Require: Truth Table S of f with 2^n entries

```
for i from 0 to n − 1 do
    Let Sz ← 2^i, Pos ← 0
    while Pos < 2^n do
        for j = 0 to Sz − 1 do
            S[Pos + Sz + j] ← S[Pos + j] ⊕ S[Pos + Sz + j]
        end for
        Let Pos ← Pos + 2 · Sz
    end while
end for
```

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 worth noting that Algorithm 2 be could be accelerated. For instants, 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 Moebius 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 on 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 Moebius 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 Moebius transformation, experimental test 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 Moebius 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 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 on 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 definition.

Definition 3 (Steep IV Variable). *Let $I = \{v_{i_1}, v_{i_2}, \ldots, v_{i_l}\}$ be a set containing l cube variables. Then, an IV variable $b \in B$ is called a steep IV variable of I if*

$$ds(I \cup \{b\}) = \min\{ds(I \cup \{v\}) | v \in B\},$$

where $B = \{v_0, v_1, \ldots, v_{m-1}\} \setminus I$ and $ds(I)$ is the degree of the superpoly of I.

Let I be a starting set of cube variables. It can be seen that a steep IV variable of I is exactly the one which makes the degree of the superpoly decrease as fast as possible. To construct a cube with linear superpoly from I, a natural idea is extending 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 as fast as possible. 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 variables is added in each iteration. Let v be a steep IV variable of I'. It is possible that $ds(I' \cup v) = 0$, while $ds(I') > 5$. 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 potentially 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 of 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 $I = \{v_{i_1}, v_{i_2}, \ldots, v_{i_l}\}$ be a set containing l cube variables. Then, an IV variable $b \in B$ is called a gentle IV variable of I if*

$$ds(I \cup \{b\}) = \max\{ds(I \cup \{v\}) | v \in B \text{ and } ds(I \cup \{v\}) \leq ds(I)\},$$

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, 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.

Based on the above ideas, we propose a new heuristic algorithm to construct cubes with linear superpolies, see Algorithm 3. In Algorithm 3, similar to the GreedyBitSet algorithm which is a heuristic algorithm proposed in [20], 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 [25], to calculate the upper bound of $ds(I \cup v)$ for each IV variable which are 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 cube with linear superpolies.

Remark 1. In the second stage of Algorithm 3, for I, it may encounter the case that $ds(I \cup \{v\}) > ds(I)$ or $ds(I \cup \{v\}) = 0$ 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 $ds(I \cup \{b\}) = \min\{ds(I \cup \{v\}) > ds(I) | v \in B\}$ 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

Algorithm 3 The algorithm of constructing cubes with linear superpolies

Input: a set of cube variables $I = \{v_{i_1}, v_{i_2}, \ldots, v_{i_c}\}$ of size c and the target round r

$B \leftarrow \{v_0, v_1, \ldots, v_{m-1}\} \setminus I$;
$d_s \leftarrow 100$;
/* The first stage */
while $ds > 1$ 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;
  $B \leftarrow B \setminus v$;
  $ds \leftarrow DS(I \cup \{v\})$, where $DS(I \cup \{v\})$ is the upper bound of $ds(I \cup \{v\})$
end while
if $ds(I) == 1$ **then**
  return I
end if
/* The second stage */
if $ds(I) == 0$ **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 $ds > 1$ 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
    $B \leftarrow B \setminus v$;
    $ds \leftarrow DS(I \cup \{v\})$
  end while
end if

cube, called a mother cube in this paper, and then use the Moebius transformation to test its subcubes simultaneously. Such a mother cube is constructed by jointing some cubes obtained in the last iteration. When selecting cubes, we prefer to those cubes such that the degree of the corresponding superpolies are close to 1. In another word, the mother cube is constructed as following

$$I \cup \{v \in \{v_0, v_1, \ldots, v_{m-1}\} \setminus I | \text{ the upper bound of } ds(I \cup v) \text{ is close to } 1\},$$

where I is the set of cube variables before the last iteration.

3.2 Determine Starting Cube Sets

In this subsection, to determine a starting cube set of Algorithm 3 properly, we propose the concept of the preference bit together with an algorithm to predict

it based on the structural analysis of Trivium. Finally, considering the preference bit, we show how to determine a starting cube set of Algorithm 3 properly.

The Preference Bit. Note that all the known linear superpolies of Trivium are sparse, and most of them contain only a single key variable. Recall that the output function of r-round Trivium is the linear combination of six internal state bits, i.e,

$$z_r = \bigoplus_{j=1}^{6} s_{i_j}^{(r)},$$

where $\{i_1, i_2, i_3, i_4, i_5, i_6\} = \{66, 93, 162, 177, 243, 288\}$. It is very likely that the linear superpoly is contributed by a single internal state bit of these six bits. For each bit $s_{i_j}^{(r)} (1 \leq j \leq 6)$, the probability of contributing a linear superpoly differs from each other since their ANFs are different. Considering this difference, we introduce the concept of the preference bit. Before introducing the preference bit, we shall first formally describe the concept of a superpoly is contributed by an internal state bit $s_{i_j}^{(r)}$, where $j \in \{1, 2, \ldots, 6\}$.

Let I be a set of cube variables. Assuming that the superpoly of I in z_r, denoted by p_I, is linear, where z_r is the first output bit of r-round Trivium. According to the output function of Trivium, the superpoly p_I could be decomposed into

$$p_I = p_{i_1} \oplus p_{i_2} \oplus p_{i_3} \oplus p_{i_4} \oplus p_{i_5} \oplus p_{i_6},$$

where p_{i_j} is the superpoly of I in $s_{i_j}^{(r)}$. If p_I and p_{i_j} are both linear, then, in this paper, it is called that $s_{i_j}^{(r)}$ contributes a linear superpoly. 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} of 769-round Trivium is $p_I = k_{22}$. We test the superpolies of I in $s_{66}^{(769)}, s_{93}^{(769)}, s_{162}^{(769)}, s_{177}^{(769)}, s_{243}^{(769)}, s_{288}^{(769)}$. The results show that only the superpoly $p_{66} = k_{22}$ is linear. Namely, $s_{66}^{(769)}$ contributes a linear superpoly.

Since the ANFs of these six internal bits are different, probabilities of them to contribute a linear superpoly are different. Considering this difference, we propose the concept of the preference bit, which is formally described in Definition 5.

Definition 5. *Among the six internal state bits in output function of r-round Trivium, the internal state bit which is the most likely to contribute linear superpolies is called the preference bit of r-round Trivium.*

According to Definition 5, it can be seen that when targeting at the preference bit, it is more likely to find linear superpolies than targeting at other internal state bits in the output function. However, it seems that Definition 5 is somewhat vague. In the following, we propose a lemma which could make Definition 5 more clear and offer us an approach to predict the preference bit, see Lemma 1.

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, then there is a term in the form of $\prod_{v \in I} v \cdot k_j$ in the ANF of f.*

Proof. Assume that the superpoly of I in f is $L(\boldsymbol{k}) = \sum_{i=0}^{n-1} c_i \cdot k_i \oplus c$, where $c_i, c \in \{0, 1\}$. Then, the expression $\prod_{v \in I} v \cdot L(\boldsymbol{k})$ is a part of $f(\boldsymbol{k}, \boldsymbol{v})$, namely, $f(\boldsymbol{k}, \boldsymbol{v})$ could be rewritten as

$$f(\boldsymbol{k}, \boldsymbol{v}) = \prod_{v \in I} v \cdot L(\boldsymbol{k}) \oplus g(\boldsymbol{k}, \boldsymbol{v}).$$

It can be seen that there is a term in the form of $\prod_{v \in I} v \cdot k_j$ in the ANF of f.

According to Lemma 1, the necessary condition of $s_{i_j}^{(r)}$ contributing a linear superpoly is that $s_{i_j}^{(r)}$ 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 remainder of this paper, a term in the form of $T_v \cdot k_j$ is called a VK-term for simplicity. Then, it is reasonable that the bit with more VK-terms is more likely to contribute a linear superpoly. Namely, among the six internal state bits in the output function, the bit with the most VK-terms should be regarded as the preference bit. However, as the number of initialisation rounds increases, the ANF of an internal state bit becomes very complex. In this case, it is difficult to determine the preference bit by calculating the exact number of VK-terms in the ANF of an internal state bit.

An Iterative Algorithm to Predict the Preference Bit. To overcome the above difficulty, we propose an algorithm to estimate the number of VK-terms in the ANF of an internal state bit. With this algorithm, we could predict the preference bit of the output bit after any number of initialisation rounds.

Our main idea is estimating the number of VK-terms in the ANF of an internal state bit iteratively. 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 on 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 the 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 to estimate the number of VK-terms in the ANF of an internal state bit iteratively. 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, the number of VK-terms in $s_{91}^{(t)} \cdot s_{92}^{(t)}$, denoted by $NVK(s_{91}^{(t)} \cdot s_{92}^{(t)})$, could be estimated* as

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

Consequently, $NVK_{94}^{(t+1)}$ could be estimated as

$$NVK_{94}^{(t+1)} = NV_{91}^{(t)} \cdot NVK_{92}^{(t)} + NV_{92}^{(t)} \cdot NVK_{91}^{(t)} + NVK_{93}^{(t)} + NVK_{66}^{(t)} + NVK_{171}^{(t)}.$$

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

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

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

$$NVK^{(t)} = (NVK_1^{(t)}, \ldots, NVK_{288}^{(t)}) \text{ and } NV^{(t)} = (NV_1^{(t)}, \ldots, NV_{288}^{(t)}).$$

Now, the remaining problem is how to initialise $NVK^{(0)}$ and $NV^{(0)}$. To obtain a more accurate result, we initialise $NVK^{(280)}$ 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 $j \in \{66, 93, 162, 177, 243, 288\}$ 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.

---

* Here, we do not take the terms which are eliminated by the XOR operation into consideration.

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
  $NV_{94}^{(t)} \leftarrow NV_{t_1}$;
  $NVK_{94}^{(t)} \leftarrow NVK_{t_1}$;
  $NV_{178}^{(t)} \leftarrow NV_{t_2}$;
  $NVK_{178}^{(t)} \leftarrow NVK_{t_2}$;
  $NV_1^{(t)} \leftarrow NV_{t_3}$;
  $NVK_1^{(t)} \leftarrow NVK_{t_3}$;
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 $b \in \{66, 93, 162, 171, 243, 288\}$;

Determine A Proper Starting Set. In this subsection, we shall show how to determine a starting set of Algorithm 3 with the knowledge of the preference bit. Let $s_\lambda^{(r)}$ be the preference bit of r-round Trivium. First, according to the update function of Trivium, $s_\lambda^{(r)}$ could be written as

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

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

Assume that $I' = \{v_{l_1}, v_{l_2}, \ldots, v_{l_u}\}$ is a subcube with a linear superpoly in $s_{j_1^\lambda}^{(r-\lambda)}$. According to Lemma 1, we have that there is a VK-term in the ANF of $s_{j_1^\lambda}^{(r-\lambda)}$. Hence, it is hopeful that we could extend I' to I_f whose superpoly in $s_\lambda^{(r)}$ is linear. Since $s_\lambda^{(r)}$ is the preference bit, it is hopeful that the superpoly of I_f in the output bit is linear as well. The following is an illustrative example.

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

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

is a carefully chosen starting cube set such that its superpoly in $s_{286}^{(739)}$ is k_{56}. Furthermore, we find that the superpoly of the cube set

$$I'' = \{v_1, v_2, v_4, v_6, v_8, v_{10}, v_{11}, v_{13}, v_{15}, v_{17}, v_{19}, v_{21}, v_{23}, v_{25}, v_{26}, v_{27}, v_{29}, v_{32},$$

$$v_{34}, v_{36}, v_{38}, v_{39}, v_{41}, v_{42}, v_{43}, v_{45}, v_{47}, v_{48}, v_{50}, v_{52}, v_{57}, v_{59}, v_{69}, v_{71}, v_{76}, v_{79}\},$$

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

4 An Improved Moebius Transformation

The Moebius 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 initialisation 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 the Moebius transformation, to test all the subcubes of I, the memory complexity is $O(2^{|I|})$ which expands exponentially as $|I|$ increases. In this subsection, we shall present an improved Moebius 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 Moebius 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:

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

Recall that the function g is the Moebius transformation of f. It can be seen that the Moebius transformation g is actually a Boolean function on n variables. Furthermore, the Moebius 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, see Lemma 2.

Lemma 2 ([11]). *Let $f(x_0, x_1, \ldots, x_{n-1})$ be a Boolean function on $x_0, \ldots, x_{n-1}$. Assuming $g_0(y_0, y_1, \ldots, y_{n-2})$ and $g_1(y_0, y_1, \ldots, y_{n-2})$ are the transformations of*

$$f(x_0, x_1, \ldots, x_{n-2}, 0) \text{ and } f(x_0, x_1, \ldots, x_{n-2}, 1)$$

respectively. Then, $g_0(y_0, y_1, \ldots, y_{n-2})$ *and* $g_1(y_0, y_1, \ldots, y_{n-2})$ *are related to the Moebius transformation* $g(y_0, y_1, \ldots, y_{n-1})$ *of* $f(x_0, x_1, \ldots, x_{n-1})$ *by the following equation:*

$$g(y_0, y_1, \ldots, y_{n-2}, 0) = g_0(y_0, y_1, \ldots, y_{n-2})$$
$$g(y_0, y_1, \ldots, y_{n-2}, 1) = g_0(y_0, y_1, \ldots, y_{n-2}) \oplus g_1(y_0, y_1, \ldots, y_{n-2}).$$

Lemma 2 indicates us that we could obtain the Moebius transformation of $f(x_0, x_1, \ldots, x_{n-1})$ from the Moebius transformations of $f(x_0, x_1, \ldots, x_{n-2}, 0)$ and $f(x_0, x_1, \ldots, x_{n-2}, 1)$. Furthermore, this fact 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 Moebius transformations of*

$$f(x_0, x_1, \ldots, x_{n-q-1}, 0, \ldots, 0),$$
$$f(x_0, x_1, \ldots, x_{n-q-1}, 1, \ldots, 0),$$
$$\vdots$$
$$f(x_0, x_1, \ldots, x_{n-q-1}, 1, \ldots, 1).$$

Then, the Moebius transformation g *of* f *could be determined with the knowledge of* $g_0, g_1, \ldots, g_{2^q-1}$.

Proof. According to Lemma 2, it is sufficient to calculate the Moebius transformation of f with the Moebius 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 Moebius 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 Moebius 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 Moebius transformation g of f could be determined with the Moebius transformations of

$$f(x_0, x_1, \ldots, x_{n-q-1}, 0, \ldots, 0),$$
$$f(x_0, x_1, \ldots, x_{n-q-1}, 1, \ldots, 0),$$
$$\vdots$$
$$f(x_0, x_1, \ldots, x_{n-q-1}, 1, \ldots, 1).$$

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 1.

Proposition 1. *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*

$$\begin{gathered}\prod_{i=0}^{n-q-1} x_i^{c_i},\\ x_{n-q}\cdot\prod_{i=0}^{n-q-1} x_i^{c_i},\\ \vdots\\ x_{n-q}\cdot x_{n-q+1}\cdots x_{n-1}\cdot\prod_{i=0}^{n-q-1} x_i^{c_i}.\end{gathered}$$

in the ANF of f.

Proof. Assume that $(b_{n-q}, b_{n-q+1}, \ldots, b_{n-1})$ tabkes an arbitrary value of $\mathbb{F}_2^q$. Following the prove 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 Moebius 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 $h_{0,0}(c_0, \ldots, c_{n-q-1}, b_{n-q}, \ldots, b_{n-3})$ 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 Moebius 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 $g(c_0, c_1, \ldots, c_{n-q-1}, 0, 0, \ldots, 0)$, $g(c_0, c_1, \ldots, c_{n-q-1}, 1, 0, \ldots, 0), \ldots,$ $g(c_0, c_1, \ldots, c_{n-q-1}, 1, 1, \ldots, 1)$ could be obtained. Namely, we could recover the coefficients of

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

in the ANF of f.

Based on Proposition 1, we propose an improved Moebius transformation by breaking the original Moebius 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 Moebius transformation in Algorithm 5. During the first stage of Algorithm 5, for each $0 \le j \le 2^q-1$, the Moebius 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 Moebius 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 the following 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, then 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 Moebius 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 Moebius 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**
    $tmp \leftarrow (b_0, b_1, \ldots, b_{q-1})$, where $i = \sum_{l=0}^{q-1} b_l \cdot 2^{b_l}$;
    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 subsection 3.2, we attempt to find linear superpolies for Trivium variants with at least 805 initialisation 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 experiments on the output of r-round Trivium, where $400 \leq r \leq 699$. First, for each Trivium variant with $r(400 \leq r \leq 699)$ rounds, we collect thousands of cubes with linear superpolies in its first output bit. Then, for every such cube, we test whether its superpoly in s_{i_j} is linear to determine whether s_{i_j} contributes a linear superpoly, where i_j runs over the set $\{66, 93, 162, 177, 243, 288\}$. For r-round Trivium, after testing all the found cubes, we could obtain a tuple with six elements which records the

times of contributing a linear superpoly for $s_{66}^{(r)}, s_{93}^{(r)}, s_{162}^{(r)}, s_{177}^{(r)}, s_{243}^{(r)}, s_{288}^{(r)}$. Thus, we could figure out the preference bit of r-round Trivium experimentally. For instance, we collect 3030 linear superpolies for 440-round Trivium, after testing all the six bits, we have that $s_{66}^{(440)}, s_{93}^{(440)}, s_{162}^{(440)}, s_{177}^{(440)}, s_{243}^{(440)}$, and s_{288} contributes 883, 29, 205, 59, 1551 and 300 linear superpolies respectively. Hence, $s_{243}^{(440)}$ is the preference bit of 440-round Trivium. Finally, we predict the preference bit of r-round Trivium with Algorithm 4. The result shows that we predict the preference bit correctly for 226 variants of Trivium out of the total 300 variants. It indicates that we could predict the preference bit correctly with a probability 75.3% which is significantly higher than 16.67%, i.e. the success probability of predicting the preference bit randomly.

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

In this subsection, we target at 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 Moebius transformation.

Determine Proper Starting Sets. To determine 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 Moebius 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 the similar procedure, we determine some other starting sets of Algorithm 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 2 shows the cubes and the upper bounds of the degrees of their superpolies, where

$$I_5 = \{v_4, v_6, v_{10}, v_{11}, v_{15}, v_{17}, v_{19}, v_{21}, v_{25}, v_{29}, v_{32}, v_{34}, v_{36}, v_{39}, v_{41}, v_{43},$$

$$v_{50}, v_2, v_{69}, v_{79}, v_8, v_{27}, v_0, v_1, v_{28}, v_{71}, v_{13}, v_{45}, v_{23}, v_{26}, v_{38}, v_{76}, v_{47}, v_{52}\}.$$

Table 2. The Chosen Cube Variables in the Last Iteration

chosen cube	upper bound of the degree of superpolies
$I_5 \cup \{v_{57}\}$	1
$I_5 \cup \{v_{59}\}$	1
$I_5 \cup \{v_{75}\}$	2
$I_5 \cup \{v_{53}\}$	3

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

$$I_6 = \{v_4, v_6, v_{10}, v_{11}, v_{15}, v_{17}, v_{19}, v_{21}, v_{25}, v_{29}, v_{32}, v_{34}, v_{36}, v_{39}, v_{41}, v_{43}, v_{50}, v_2, v_{69},$$

$$v_{79}, v_8, v_{27}, v_0, v_1, v_{28}, v_{71}, v_{13}, v_{45}, v_{23}, v_{26}, v_{38}, v_{76}, v_{47}, v_{52}, v_{48}, v_{42}v_{53}, v_{57}, v_{59}, v_{75}\}.$$

Linear Superpolies of 805-Round Trivium. After obtaining a potentially good cube, we use the improved Moebius 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 Moebius 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 37 cubes whose superpolies are linearly independent, see Table 3.

Table 3. Linear superpolies of 805-round Trivium

cube indies	superpolies
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 of 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 4.

Table 4. Cubes with linear superpolies in 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 Attacks 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 33 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.

Key-Recovery Attacks on 806-Round Trivium. Based on the linear superpolies of 806-round Trivium, we could recover 16 key bits with $2^{38.64}$ requests. By adding a brute-force attack, the remaining 64 key bits could be recovered in 2^{64} requests. Hence, for 806-round Trivium, the 80-bit key could be recovered with on-line complexity of $2^{64} + 2^{38.64}$.

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. Due to the limited time, we only 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 obtain a cube

$$I_8 = \{v_2, v_6, v_8, v_{10}, v_{11}, v_{15}, v_{19}, v_{21}, v_{25}, v_{29}, v_{30}, v_{32}, v_{34}, v_{36},$$

$$v_{39}, v_{41}, v_{43}, v_{45}, v_{50}, v_0, v_{75}, v_{12}, v_4, v_{14}, v_{20}, v_{22}, v_{16}, v_{27}, v_{23},$$

$$v_{72}, v_{52}, v_{55}, v_{60}, v_{37}, v_{79}, v_{62}, v_{64}, v_{47}, v_{54}, v_{69}, v_{51}, v_{71}, v_{18}, v_{53}\}.$$

The size of I_7 is 44. It is too large to perform sufficiently many times linearity tests. Hence, we try remove some cube variables from I_8 to obtain smaller cubes with low-degree superpolies. Finally, we obtain a cube I_9 of size 43, where

$$I_9 = \{v_2, v_6, v_8, v_{10}, v_{11}, v_{15}, v_{19}, v_{21}, v_{25}, v_{29}, v_{30}, v_{32}, v_{34}, v_{36},$$

$$v_{39}, v_{41}, v_{43}, v_{45}, v_{50}, v_0, v_{75}, v_{12}, v_4, v_{14}, v_{20}, v_{22}, v_{16}, v_{27}, v_{23},$$

$$v_{72}, v_{52}, v_{55}, v_{60}, v_{37}, v_{79}, v_{62}, v_{64}, v_{47}, v_{54}, v_{69}, v_{51}, v_{71}, v_{18}, v_{53}\}$$

and the degree of the superpoly of I_9 is upper bounded by 2. By using a computer with four NIVIDA V100 GPUs, we search a part of subcubes which only missing few cube variables in I_9. With the original Moebius transformation, to search subcubes of a 43-dimensional cubes, it needs 2^{43} bits memory. Benefited from the improved Moebius 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 Moebius transformation. Finally, we find 2 different cubes with linear superpolies, whichi are listed in Table 5.

Table 5. Linear Superpolies of 810-round Trivium

cube indies	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 2. We put our codes and all the found superpolies on the site 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 of 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 every cube constructed by our algorithm could lead to some linear superpolies. Hence, to practically attack 810-round Trivium, it is expected that only more time needs to be taken because of the larger cube sizes. Combing our new algorithm for selecting cubes with the bit-based division property using three subsets 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. Cryptology ePrint Archive, Report 2020/1048, 2020. https://eprint.iacr.org/2020/1048.
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, Dongdai Lin, and Wenhao Wang. Searching cubes for testing boolean functions and its application to Trivium. In *IEEE International Symposium on Information Theory, ISIT 2015, Hong Kong, China, June 14-19, 2015*, pages 496–500, 2015.
16. 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.
17. Piotr Mroczkowski and Janusz Szmidt. Corrigendum to: The cube attack on stream cipher Trivium and quadraticity tests. *IACR Cryptology ePrint Archive*, 2011:32, 2011.
18. 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.
19. 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.
20. 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.

21. 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. https://eprint.iacr.org/2016/811.
22. 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.
23. 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.
24. 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.
25. 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.
26. 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.
27. 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.
28. 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.
29. 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.
30. Chen-Dong Ye and Tian Tian. Algebraic method to recover superpolies in cube attacks. *IET Information Security*, 14(4):430–441, 2020.